Factory Product Lines: Tackling the Compatibility Problem

Andreas Bayha
fortiss GmbH, Guerickestr. 25, 80805 Munich
bayha@fortiss.org

Levi Lúcio
fortiss GmbH, Guerickestr. 25, 80805 Munich
lucio@fortiss.org

Vincent Aravantinos
fortiss GmbH, Guerickestr. 25, 80805 Munich
aravantinos@fortiss.org

Kenji Miyamoto
fortiss GmbH, Guerickestr. 25, 80805 Munich
miyamoto@fortiss.org

Georgeta Igna
fortiss GmbH, Guerickestr. 25, 80805 Munich
igna@fortiss.org

ABSTRACT

Variability poses enormous challenges to the manufacturing domain: the demand for highly customized and personalized goods requires bringing a large amount of flexibility to traditional mass production techniques. In particular, when a factory is asked to produce of a new variant of a good, production planning engineers need to manually examine that factory in question, in order to understand whether that new variant can indeed be produced. We call this the "compatibility between the variant and the factory" problem. In this paper, we present a product line-based modelling technique and a tool for assisting production planning engineers in the resolution of this problem.

CCS Concepts

• **Software and its engineering** → System modeling languages; • **Computing methodologies** → *Modeling methodologies;* • **Applied computing** → *Computer-aided design;*

Keywords

Manufacturing Planning; Variability; Modelling; Variability Management

1. INTRODUCTION

Personalized goods[1] are becoming an increasingly important marketing argument. Thus, the ability to provide personalized goods at a low cost is receiving a great deal of attention [13]. This naturally brings new challenges to goods manufacturing: production lines should be able to produce a whole set of good *variants* while requiring minimal changes. Furthermore, such production lines should quickly adapt to evolution in demand. This means that producing a new variant of a good should not, as is the case now, require completely stopping the manufacturing process in order to: a) check if a factory can produce that new variant of the good; b) perform the necessary changes to the factory in case of need.

Tackling variability problems requires bringing more "intelligence" into the manufacturing process. It is thus natural to attempt making use of software techniques to tackle these issues. These ideas yielded the *Industry 4.0* [9] or the *Industrial Internet*[2].

In this paper we focus on how to assist production planning engineers when deciding on whether it is possible to produce a new variant of a good in an existing factory. Throughout this paper we will call this the *compatibility problem*. For this purpose, we introduce in this paper the SFIT tool (for **S**mart **F**actory **IT**). An initial version of our tool was presented in [12], but included very rudimentary support for variability. As such, and although the tool provided a good base for experimenting with modelling concepts in the factory domain, it was of very limited usefulness to our industrial partners at BMW for whom variability is of prime importance. Our partner's feedback led us to rebuild our SFIT tool in order to integrate full-fledged variability in the classical product-line sense. The main contributions of the work we present here, is the tool integration of: a feature model, a variability-aware good model, a variability-aware manufacturing process model, as well as many automated checks to support the user in handling the inherent complexity of variability management when addressing the compatibility problem. The SFIT tool aims to be, to the best of our knowledge, the first software that models in an integrated fashion all the variability information to automatically solve the compatibility problem between the variant and the factory (compatibility problem).

2. BACKGROUND AND MOTIVATION

The existence of numerous variants of a given good raises some very concrete production problems: consider a factory that manufactures a given set of variants of one good (e.g., only Diesel variants of a given car for the US market and

[1]We will use the term "good" instead of "product" to denote the products which are the outcomes of a manufacturing process. This is to avoid the collision with the term "product" as understood in "product line".

[2]http://www.iiconsortium.org/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

VaMoS '16, January 27 - 29, 2016, Salvador, Brazil

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4019-9/16/01…$15.00

DOI: http://dx.doi.org/10.1145/2866614.2866623

all sorts of cars for the European market). Suppose now a request to manufacture a new variant of the good (e.g., a car for the Japanese market) arrives to that factory. The immediate questions are: can this variant be manufactured using the existing factory setup? Is adapting the factory necessary? Are new supplies required for the manufacturing chain? Nowadays these questions are answered through manual, time-consuming and error-prone analysis of the factory. This analysis is carried out by experts (the production planning engineer, possibly in collaboration with the designer of the good) making use of their knowledge of the factory's physical structure, the various machines involved in the manufacturing process, as well as the various requirements for the new product.

Generally, the result of this analysis then implies stopping the production in order to perform changes and iteratively test the new setup. This situation is less and less sustainable: on the one hand, the number of features, and therefore of variants, is growing drastically; on the other hand, the market itself drives the evolution of the goods at a faster pace than ever before. It is thus crucial to provide tools that can help production planning engineers in understanding as quickly as possible to which level a new variant of a product is compatible with an existing factory. The compatibility problem is nowadays a bottleneck in production and in particular for our industrial partners: as in software, the cost of finding incompatibilities rises exponentially as the stages of production advance.

Solving the compatibility problem requires a large amount of information, which usually only exists in the mind of domain experts such as the production planner and the designer of the good. Modelling the physical organization of the factory, which machines exist there, how they are connected and which supplies are available where is naturally of prime importance.

Then, the information on how each variant is produced in terms of concrete logical operations also needs to be provided in an artifact called the *recipe*. The recipe mainly defines the logical dependencies between assembly line operations. Finally, the composition of the good is described in the *bill of materials*, thus making the connection to the required supply of parts to the factory.

More precisely, the following views over the manufacturing process need to be modelled:

- *Bill of materials*, describing the parts that compose goods.
- *Recipe*, describing the logical manufacturing process for assembling those parts into goods.
- *Assembly*, modeling the assembly line that physically assembles goods.

In order to solve the compatibility problem, it is also necessary to describe how the artifacts described above are connected. As mentioned before, modelling the recipe is necessary to understand which logical operations are needed to manufacture the good. On the other hand, modelling the assembly allows knowing its physical capabilities. These two views need to be connected in order to describe which operation is achieved in which part of the factory. To do that, the physical capabilities required by the operations in the recipe and the physical capabilities provided by the machines in the factory need to be exposed. These connections thus yield the two following additional "integrating" views:

- *Capabilities*, for describing the technical abilities which can be offered by a factory and required by each operation of the recipe (e.g., "drilling", "painting", "welding").
- *Process assembly deployment* view, which shows how the recipe is implemented by the assembly. Concretely, this view describes the mapping between the operations in the recipe and elements of the assembly.

As mentioned before, we have previously introduced in [12] a simplified version of an integrated metamodel to describe the views enumerated above. However, this metamodel had very limited support for variability as only parts for goods were allowed to vary and no reference feature model existed. In practice this meant that the compatibility problem could not be addressed on real use cases.

In the remainder of this paper we will introduce the new SFIT integrated modelling tool. Section 3 presents a running case study we have devised together with our project partners at ZeMA GmbH[3]. In Section 4, the feature model view of the SFIT tool is introduced, providing the basis to describe variability in all other views of the tool. In Section 5, we give a detailed description of the metamodels used to describe the remaining views in our tool. Sections 6 provides a discussion on our research and Section 7 presents a summary of the related work on the topic. Section 8 concludes the paper.

3. RUNNING CASE STUDY

To illustrate a concrete usage of the SFIT tool, we will use throughout this document a case study modeling the production of *hardware controllers* intended to be used in factories: such controllers are physically located at stations in a factory and control the operations which are achieved by this station, e.g., "the driller should go down", "the LED should be switched on". Note that our case study describes the manufacturing process of goods which, as final products, will themselves be involved in the manufacturing process of other goods. The reason for such a self-referring case study is essentially historical: since the involved partners work in the industrial automation domain, a controller is simply an example of a product which is well-known to them.

In this paper, we model the industrial assembly of three variants of controllers for stations: two produced by ZeMA, and one produced by another company, which we will call "ACompany" for confidentiality reasons.

Each controller is made of an application board and a main board. Every variant contains a different application board. The main board differs only from one company to the other, e.g., for ZeMA controllers, it is an Arduino. The ZeMA controllers are equipped with a connection board, whereas for ACompany, communication is realized through an embedded RFID system. Finally, the ZeMA controllers have distinct displays: binary coded decimal (BCD) vs. bar graph.

We show in Figure 1 a (partly assembled) ZeMA version of a hardware controller for factories.

[3]http://www.zema.de

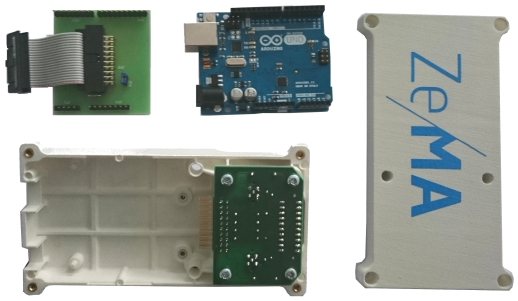


Figure 1: A Partly Assembled Controller.

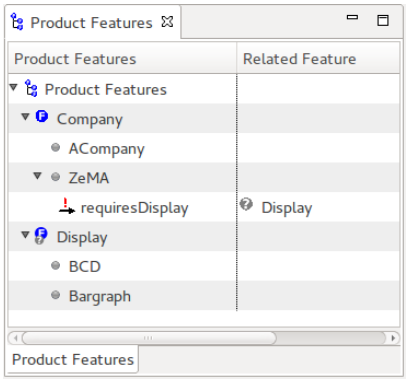


Figure 2: Feature Model for the ZeMA Case Study.

4. FEATURE MODEL

The feature model is the first and foremost building block of variability in an SFIT integrated model. In our context, a *feature* is a characteristic of a good being produced in a factory. A feature model allows describing which such characteristics exist and can co-exist. The feature model language employed in the SFIT tool is a FODA-like tree notation [14], including the following constructs:

- A *primitive feature* is the primary notion of a feature. In a feature model tree, a primitive feature is always a leaf. In the feature model example given in Figure 2, "ZeMA" is a primitive feature.
- An *alternative feature* is composed of a set of sub-features (at least two) and allows only one of those to exist in any variant of a good. In Figure 2, *Display* is an alternative feature, the *BCD* and the *Bargraph* features are mutually exclusive. This means a controller either has the binary coded decimal display feature, or the bar graph display feature.
- A *container feature* requires all of its sub-features to be present in any given variant of the good. In Figure 2, "Product Features" is an example of a container feature. Note that the sub-features inside a container feature can be made optional, in which case they may be present or not in a variant regardless of the fact that they are declared inside a container feature. Optional features are annotated with a question mark in the feature model editor.
- A *constraint* is a reference from a feature to another feature in the feature tree. A constraint can be of two different types: *requires* or *excludes*. *Requires* and *excludes* enforce that referenced features are present or excluded from a configuration whenever the feature containing the constraint is present in the configuration. In the feature model in Figure 2, *ZeMA* controllers necessarily include a display.

For clarity of the subsequent text, we now formally introduce some of the vocabulary we have used in this section.

DEFINITION 1 (SFIT FEATURE CONFIGURATION). *An SFIT feature configuration for an SFIT feature model f is a subset of all primitive features declared in f that respects the semantics of all constructs present in f's tree. When a primitive feature is present in an SFIT feature configuration we say that the feature is* selected.

According to Definition 1, examples of SFIT feature configurations for the feature model in Figure 2 would be: $\{ACompany, BCD\}$ or $\{ZeMA, BCD\}$.

DEFINITION 2 (PRESENCE CONDITION). *A* presence condition *is a propositional logic formula whose propositional variables are the primitive features of an SFIT feature configuration.*

An example of a presence condition according to Definition 2 is the formula $\neg BCD$. This presence condition enforces that the only allowed SFIT feature configurations are the configurations in which the BCD feature is not selected.

Static Checks: Feature models may contain inconsistencies. As a concrete example, consider the set of configurations corresponding to the feature model in Figure 2. If we add a constraint that "Display" excludes "ZeMA", this is not the case anymore. This happens because with such a feature model, selecting the "ZeMA" feature implies that "ZeMA" itself should be excluded. *Static checks* allows to ensure that such situations do not exist.

As explained in Definition 1, we can associate to every feature model f a set of configurations. If a feature model contains inconsistencies such as in the example above, it could be that: a) some primitive features are never represented in the set of configurations, meaning no good with that feature can be manufactured; or b) the set of configurations is empty, meaning no good at all can be manufactured. As such cases are undesired, we automatically check for these issues.

These checks are implemented by extracting a propositional logic formula from the feature model and by checking if that formula is satisfiable. Note also that presence conditions are always checked for satisfaction together with the feature model propositional logic formula they refer to.

5. INTEGRATED MANUFACTURING MODEL

Figure 3 depicts the modeling concepts of the SFIT tool. In order to manufacture a family of goods, we model a *family of bills of materials* and a *family of recipes*. The notion of *family* is central to product line engineering. In our tool such families (i.e., family of goods, family of bills of materials and family of recipes) are modelled as single generic items, describing in an abstract manner all the possible variants of the artifacts they represent. Concretely, the family of bills of materials – that in fact is a whole set of bills of materials

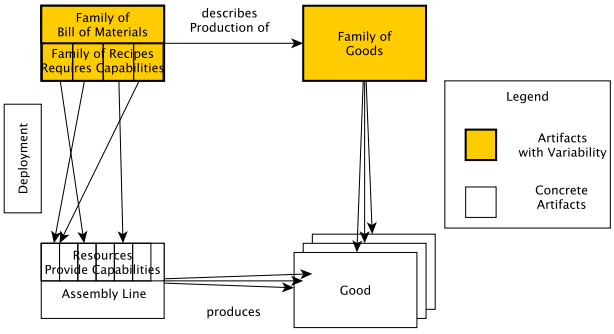


Figure 3: Concept: Producing Multiple Goods in the Same Factory.

– is represented by a single generic *bill of material* and the family of recipes is represented by a single generic *recipe*. For simplicity, in the SFIT tool and in the rest of the paper, we refer to these representations in the singular (e.g. the bill of materials instead of family of bills of materials). Note that the family of goods is not represented explicitly in the SFIT model – it is captured by the bill of materials in conjunction with the feature model.

The assembly line itself is fixed in the sense that it does not need to be changed for every variant. Hence, the corresponding *assembly line* model does not need to include variability.

The semantic connection between the variable, abstract recipe and the non-variable, concrete *resources* in the assembly line is done by an explicitly modeled, manual mapping we call *deployment*. The compatibility check can then be done by comparing the required *capabilities* of the recipe to the provided capabilities of the resources of the assembly line.

5.1 The Bill of Materials

The *bill of materials (BOM)* captures information about all individual parts required to build the goods in the family of goods. It contains atomic parts, sub-assemblies and alternative parts. Presence conditions can limit any of these parts to be required only for subsets of the family of goods.

The BOM is structured hierarchically: it contains atomic parts (i.e. material or parts which do not need to be assembled anymore), as well as sub-assemblies which are themselves made up of several atomic parts or further sub-assemblies. In the tool, there is no explicit distinction between atomic parts and sub-assemblies. In the following, the term *part* refers to atomic parts as well as sub-assemblies. Figure 4 shows an excerpt of the BOM from our case study. Herein "Electronics" is a sub-assembly, that consists of three alternative parts – "Application_Board", "Mainboard" and "Display". The "Connecting_Board" would be an example of an atomic part.

Since the various goods to be assembled vary, also the set of parts required to assemble these goods will naturally vary. The SFIT tool hence provides the capability to specify variation points in the BOM. As a result, the user is able to attach presence conditions (see Definition 2) to any part to specify for which goods they each are required. With this, a part can be restricted to be assembled into only a subset of goods in the family of goods.

Apart from declaring individual parts to be only required

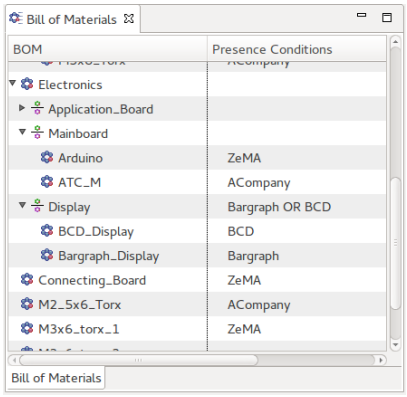


Figure 4: An Excerpt of the *bill of materials* of the Case Study.

for some goods in this way, one usually will have the need to express that there are several variants of a part to be used for different goods of the family. This is possible by imposing mutually exclusive presence conditions on those parts. To help with building such patterns the SFIT tool provides a dedicated model construct in the BOM, called the *alternative part*. Informally, an alternative part, can be considered as an abstract part representing a set of concrete part variants, from which exactly one needs to be chosen for manufacturing a single good. Presence conditions need to be specified for the part variants to describe under which circumstances the respective variant shall be chosen for the respective alternative part.

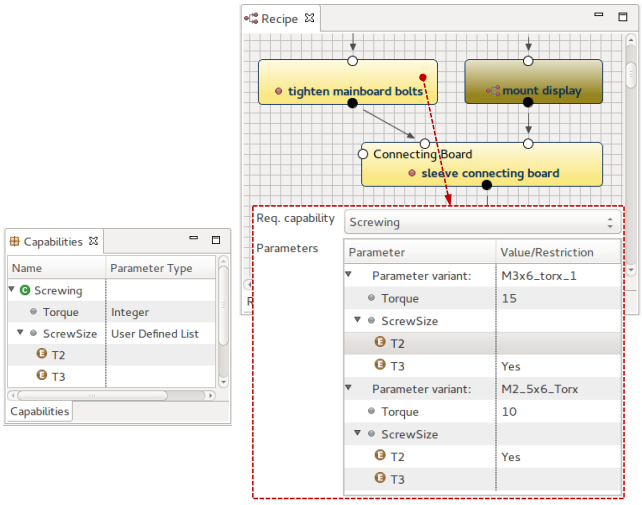
Figure 4 shows a subset of the parts present in our case study. The part "Connecting_Board" is only required for the "ZeMA" variants. Variants for the alternative part "Display" are mutually exclusive, as the presence conditions for them ("BCD" and "Bargraph") are also mutually exclusive.

Static Checks: Variability allows a great gain in abstraction, however it also potentially introduces discrepancies between the abstraction and the concrete family of objects that this abstraction is meant to represent. For instance, given a BOM and an SFIT configuration, one should be able to build the concrete BOM (i.e. without variation points) corresponding to the configuration. Concretely, for every alternative part, given a feature configuration c, it should be possible to select the part variant whose presence condition holds for c. For example in Figure 4 given a configuration, one should be able to say if *BCD_Display* or *Bargraph_Display* are selected for *Display*. This entails that, for an alternative part, for every configuration:

- *at least one* of the presence conditions (e.g., of *BCD_Display* or *Bargraph_Display*) is satisfied.
- *at most one* of the presence conditions is satisfied.

Both these conditions can be automatically checked by ensuring that the presence conditions are not "overlapping" and that these conditions cover the complete set of possible configurations. Take for example the BOM in Figure 4: if an empty presence condition (i.e., true) was assigned to "ATC_M", then the second rule above would not be satisfied since two *part variants* of "Mainboard" would be selected with the feature "ZeMA".

This static check is implemented by extracting a number of propositional formulas from the BOM, and checking if



(a) Definition of the Capability "Screwing". (b) The Step "tighten mainboard bolts" Requires the Capability "Screwing".

Figure 5: A Capability Definition and an Excerpt of a Recipe.

these formulas are satisfiable.

5.2 Capabilities

In order to assemble and process the parts in the BOM, certain technical *capabilities* are required. Examples could be operations like "screwing" or "drilling". In SFIT these capabilities are defined in a capability list. Capabilities can be parametrized to specify detailed technical information, e.g., for the "screwing" capability, it is important to specify the torque to be used. In our tool, such *parameters* can be of a numerical type or user-defined enumerations. Figure 5a shows the definition of "Screwing" with a numeric parameter "Torque" and an enumeration parameter "ScrewSize".

The capability list in SFIT has the purpose to specify the capabilities required to assemble the parts defined in the bill of materials (see Section 5.1) in order to manufacture actual goods (this aspect will be introduced later in 5.3) as well as to specify which capabilities a certain production resource can provide (see Section 5.4).

5.3 Recipes

The BOM describes the parts availabe to form the goods, while the capability list contains the capabilities that are required for assembling the goods from these parts. However neither the BOM nor the capability list capture the order in which these capabilities are required to assemble the parts in the BOM. This is described in the assembly *recipe*. A recipe defines a list of steps that need to be followed in order to assemble a good or a family of goods. It also specifies the precedence order between these steps, the capabilities and materials needed at each step.

Each step of a recipe has an interface with the parts that should be available before the step is executed (intuitively the *input* parts of a step) and the parts are available after the step is completed (the steps' *output* parts). By making connections between these inputs and outputs, the material flow is modeled as well as the precedence order.

As an example, the excerpt of the case study in Figure 5b depicts a part of the recipe for assembling the controllers (as outlined in Section 3). The black and white connectors attached to the steps contain the specification of the "input" (represented as a white circle) and "output" (black circle) of the steps. The directed edges between these connectors model material flow and precedence order between the corresponding steps.

Recipes may become large and difficult to understand for complex goods. In order to address this side effect, recipes can be hierarchically structured. Accordingly, in the SFIT tool, we have defined the following categories of steps:

- *Atomic steps*, that are not further decomposed and specify which capability they require to be executed.
- *Sub-recipes*, that only define an interface in terms of the "input" and "output" parts and hierarchically contain further sub steps.

Since a single recipe could embed the production steps encountered for an entire family of goods, variability also needs to be taken into account. For instance, consider the atomic step "tighten mainboard bolts" in Figure 5b. It requires the capability "Screwing" with parameters "Torque" and "ScrewSize". However, there are different "Mainboard" alternatives this step needs to deal with (recall that "Mainboard" is an alternative part of the BOM of Figure 4). According to the concrete "Mainboard" alternative, different sizes of screws need to be used. As a result, the parameter "ScrewSize" needs to be instantiated differently, depending on which screw is actually used. In this particular case, one can see that for the screw "M3x6_torx_1" the capability to screw "T3" screws is required, where as for a "M2" screw, "T2" is needed.

As showed in the example above, the parameters of a required capability can have multiple values – each annotated with a presence condition. As indicated in Figure 5b, these presence conditions may also contain *parts* as propositions (instead of features only), e.g., "M3x6_torx_1". Such parts still represent a set of features, but only implicitly: for instance, in the example above, the condition actually means "in all the feature configurations such that M3x6_torx_1 is present". The parts are then used as "aliases" for features. The benefit is that the user who designs the recipe might have a more technical perspective than the good designer: the choice of a capability or another might depend on the presence of a material, rather than on a feature.

In order to model variability in the steps to be performed themselves, we introduce a third kind of step:

- *Variable sub-recipes*, which are hierarchical steps that specify an interface (in terms of input and output parts) as sub-recipes do, but can also contain multiple *sub-recipe variants* that specify multiple sub-recipes that can be chosen alternatively. Here again presence conditions are used to select exactly one sub-recipe variant for every SFIT feature configuration.

In Figure 6, the variable sub-recipe "mount display" contains three sub-recipe variants – one for "Bargraph", one for "BCD" and one for manufacturing controllers without a display. The figure also shows the implementations of "Mount-Bargraph" and "DoNothing".

Static Checks: Just like for bills of materials, the gain of abstraction introduced by variability, also potentially introduces discrepancies between the abstraction and the concrete objects that this abstraction represents. Therefore, it

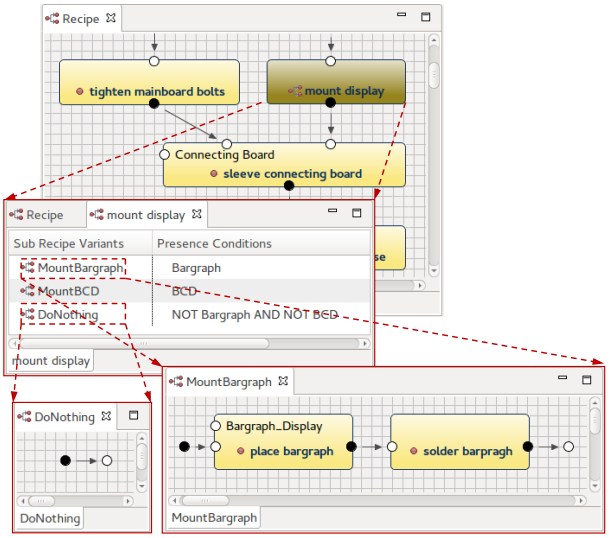


Figure 6: The *Variable Sub-Recipe* "mount display" Contains Three *Sub-Recipe Variants*.

should also be checked whether, for a given recipe and an SFIT configuration, one is to retrieve the concrete recipe corresponding to that configuration. Therefore, for every variable sub-recipe and for all parameters of atomic steps in the recipe we have that:

- *at least one* of the presence conditions (e.g., "M3x6_torx_1" or "M2_5x6_torx" in Figure 5b) is satisfied;
- *at most one* of the presence conditions is satisfied.

5.4 Resources

The recipe describes which steps on which parts have to be performed, in order to manufacture a good. While this is an abstract, logical description, it does not yet describe the physical assembly line.

For planning a physical facility, one first needs to reason about the means to provide the capabilities that the steps of the recipe require. The SFIT tool represents these means by *resources*. Every resource provides one or multiple capabilities. For each parameter of these capabilities a resource specifies which values or values ranges they can provide. A resource could model a certain machine, tool or robot.

5.5 Assembly Line Model

The *assembly line* model represents the topology of the actual factory. It clusters resources into *stations*. Hereby a station can contain one or multiple resources. A station can be connected to other stations by directed connections that (abstractly) express the possibility to transport parts from one station to another – i.e. models the possibility for physical material flow. Our model does not further specify which kind of transportation means these are or which technical capabilities they have. Figure 7 contains a simple example for an assembly line in the lower sub diagram. It contains four stations, that are sequentially connected.

As can be seen in Figure 3, the assembly line model does not contain any variability. As it is a model for the actual physical factory that is supposed to manufacture the whole family of goods (or at least a subset of it) **without** changing its topology, there is no need for variability in this artifact.

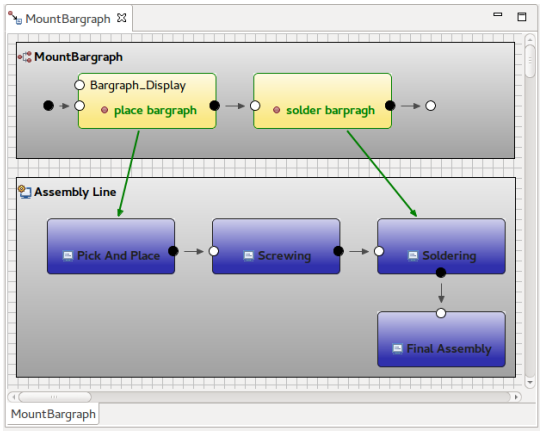


Figure 7: A *Recipe Assembly Deployment* Assigning the *Sub Recipe Variant* "MountBargraph" to a Assembly Line.

5.6 Recipe Assembly Deployment

In order to explicitly map the logical steps of a recipe we use the *recipe assembly deployment* view, or *deployment* in short. A deployment consists of pairwise mappings from steps in the recipe to stations of the assembly line. This kind of mapping is possible for atomic steps as well as for any kind of hierarchical step (sub-recipes and variable sub-recipes). The semantics for a mapping of a hierarchical step to a station is that all of the steps inside of it are mapped to that particular station. Conversely, once a hierarchical step is deployed like that, then none of its children are allowed to be mapped to any different station.

Static Checks: Deployments are also subject to static checks. First of all, a deployment should be *complete* in the sense that every step of a recipe shall be deployed on a station.

Furthermore, some negative cases of the compatibility problem can already be detected: if a step requiring a given capability – e.g., in Figure 5b, "Screwing" – is deployed on a station containing no resource providing this capability – e.g., no screwdriver – then we can already report to the user a compatibility issue exists here.

Finally, step precedence is also checked: the stations on which steps are deployed are checked for preservation of the ordering of the steps in the recipe.

5.7 Solving the Compatibility Problem

The *static checks* mentioned in the previous sections are constantly performed in the background by the SFIT tool. As soon as any of the checked constraints is violated, the user is warned by an error marker at the respective model element and by a textual description of the problem. Figure 8 displays an error marker for a coherence violation: a red cross is attached to the "Mainboard" feature indicating that its sub-features are not mutually exclusive.

The most important of these checks is however the check of whether the *compatibility problem*, that we have initially presented in Section 1, has a satisfying answer or not. In short, can the modelled factory assemble the modelled family of goods? This is achieved by the static checks described in Section 5.6 that verify that each step in the recipe us assigned a station with suitable capabilities. Also, variability in the recipe is taken into consideration by making sure

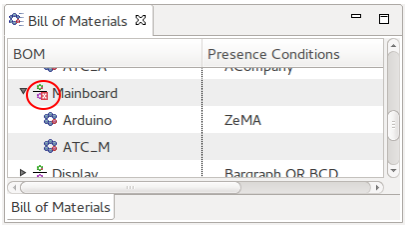


Figure 8: Error Marker: The *Presence Conditions* for the *Part Variants* "Arduino" and "BCD" are not *Coherent*.

that all variants of an atomic step (e.g. as in Figure 5b) or of variable sub-recipes (e.g. as in Figure 6) are deployed to stations with sufficient capabilities.

Note that the checks per se are simple, the bulk of the problem lying essentially on the gathering of all the necessary information about the manufacturing process.

6. DISCUSSION

In this section we present what we consider to be the most important threats to validity of the SFIT tool.

Experimentation: As mentioned, SFIT has been developed in tight collaboration with members of the planning engine assembly department of BMW AG. Early versions have been successfully experimented by planning engineers at BMW AG and at Platos GmbH (A German consulting firm specialized in planning, http://www.platos.de/). The feedback has been extremely positive, with engineers reporting favouring our tool over industrial solutions. This feedback should nevertheless be taken with a grain of salt: the selected planning engineers were "friendly" users, i.e., open to change, and were fully aware that the tool was a research prototype. Experiments and evaluations in a real environment are necessary to assess the real benefits of SFIT.

Abstraction Level of the SFIT Models: The metamodel we have implemented in the SFIT tool is tailored to the compatibility problem between a factory and a variant. Naturally, many details of reality are left out. We have recently received feedback from our industrial partners at BMW AG regarding the potential need for modelling the 3D shape of an object going trough the product line, as that information can add additional precision to decisions on compatibility problems. While maintaining complete 3D models the shape of physical parts and assembly lines might prove to be computationally heavy, we envisage we can include in our tool abbreviated models of reality that can be used to implement such requirements coming from our industrial partners. In general we believe the level of abstraction we have used in our tool to describe the domain knowledge is adequate for the purpose of our work. Nonetheless, we envisage that extensions such as the one above may need to be added to the metamodel we have currently implemented in the SFIT tool.

Usability of the SFIT Tool: Solving the compatibility problem per se is of great help for production planners, but it naturally raises additional questions. Once it is discovered the factory is not compatible with the requested variant, the production planner must then still analyze the factory in order to investigate possible options to allow the production of that variant. Currently, if the factory is incompatible with the variant, an error message explaining *why* the variant is not producible is displayed. An example of such a message would be: "at step s, the variant v requires the material m, which is not available at the station st on which s is deployed". Despite the length of this message, our interaction with BMW AG proved that this message was greatly appreciated because it precisely indicated the source of the incompatibility. It is however possible to go further by suggesting possible tips to solving the incompatibility identified in an error message. An example of such tip could be using a different, more powerful, resource from the available resources if the required capability is not available. Coming up with such suggestions is however a significantly more difficult problem as just answering the compatibility problem, and would most probably require a transformation of the problem into an SMT problem [1].

7. RELATED WORK

The SFIT approach relates to different aspects of manufacturing modeling that have been explored in the literature:

Bill of Materials: The *bill of material* we introduce here, is partly similar to the proposal of *generic-bill-of-materials* (GBOM) in [6]. A GBOM is a BOM, that lists the parts for a whole set of goods - even though some of them might not be required for all goods of that set. It may further contain generic parts, for which different part variants are available - similar to the *alternative parts* in our approach. Based on the GBOM, [7] defines a *generic bill-of-materials-and-operations* (BOMO). This BOMO extends the GBOM with the operations required to assemble the parts it describes. The BOMO further specifies the order in which those operations have to be performed. We also model such operations, however using a dedicated modeling view – the *recipe*.

Liaison Graph: the *liaison graph* captures information similarly to the *BOM* and has been used in [2] for an assembly-oriented product family representation. However, liaison graphs model the actual physical contact points between parts of the product, which exceeds the level of detail we are interested in for our work.

Routing: Our representation of *recipes* is close to what is denoted as *routing* in literature - i.e. a (usually graph-based) description of the precedence order of assembly operations that additionally incorporates information about which parts are required for which operations. In the literature there is also work dedicated to routing in the presence of variability, such as the *process platform* presented in [15] and [8]. This approach models the generic product structure and a generic assembly process structure. The authors then integrate these two views into one *generic product and process structure* from which product specific routings can be derived. In contrast to our work, the authors do not model variability explicitly – variability is achieved by removing parts of the model. In contrast to the SFIT approach, this approach and its implementation does not consider actual assembly lines and does not maintain a dedicated variability model as we do. Another concept for describing "workflows" for mass customization – similar to our recipes – is presented in [11]. In comparison to our recipe language, their concept allows more complex constructs like forking, however does not go into detail on the deployment to the assembly line.

Unlike the approaches previously mentioned in this section, the methodology proposed in [3] takes into account the assembly line layout, This methodology includes modeling of an assembly plan (similar to a liaison graph) with generic parts, assembly operations plus a precedence order

between them, as well as the mentioned assembly line layout including conveyor structures. The methodology proposed by the authors structures the family of goods to be manufactured using so called functional entities – parts and sub-assemblies are clustered according to their functionality. This is achieved differently in our feature modelling approach: given our features are more abstract, they allow to also express variability in non-functional issues, such as marketing, legal or quality constraints. As far as we know, there has not yet been a tool implementation for this methodology.

Integrated Modeling and Tooling: Existing software for manufacturing planning such as Siemens Teamcenter© or similar tools mostly concentrate on issues other than the compatibility problem, in particular assembly line balancing or geometry [12]. The authors of [5], propose an extensible tooling approach based on a language workbench and DSLs. It supports different views on manufacturing while at the same time maintaining an integrated model. The proposed views are similar to the ones we provide. The tool also incorporates a feature diagram, a bill of material and assembly sequence plus additional views we do not consider, such as tolerance models, assembly line balancing, throughput or work instructions. However most of these views are not yet implemented. The same authors also propose a technique for the automated analysis of the effects of variability on the assembly operations in [4]. Their work is more advanced than ours regarding reasoning about the correct precedence order of assembly operations, despite the fact that it only concentrates on this aspect of manufacturing.

The tool presented in [10] models the capabilities and the topology of the factory similarly to us. They also consider the modeled recipe and the compatibility with the assembly line and go even further by e.g. also reasoning about production scheduling. However, they do not consider variability as explicitly as we do.

8. CONCLUSION

In this paper we presented the SFIT tool for assisting production planning engineers in bringing more flexibility into manufacturing. This is achieved by proposing an integrated metamodel of the manufacturing process, including a whole range of artifacts from the description of the good to a description of the machines involved in the production. Variability is taken into account in our modelling framework by using techniques coming from the software product lines domain. Furthermore, SFIT implements a large amount of automated static checks to help the user while handling the complexity inherent to variability in the design of their manufacturing process. Based on these checks, the SFIT tool is capable of alerting the production planning engineer for incompatibilities between the variants of the good that should be manufactured and the physical factory.

Acknowledgements

SFIT is developed by fortiss in collaboration with the BMW Group and Platos GmbH, in the context of the research project “SmartF- IT”, funded by the German Federal Ministry of Education and Research and supervised by the German Aerospace Center under the funding code 01S13015.

9. REFERENCES

[1] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. satisfiability modulo theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, Feb. 2009.

[2] P. De Lit, J. Danloy, A. Delchambre, and J.-M. Henrioud. An assembly-oriented product family representation for integrated design. *Robotics and Automation, IEEE Transactions on*, 19(1), 2003.

[3] P. De Lit, A. Delchambre, and J.-M. Henrioud. An integrated approach for product family and assembly system design. *Robotics and Automation, IEEE Transactions on*, 19(2):324–334, 2003.

[4] A. H. Ebrahimi, K. Akesson, P. E. Johansson, and T. Lezama. Formal analysis of product variability and the effects on assembly operations. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–4. IEEE, 2015.

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

[6] H. Hegge and J. Wortmann. Generic bill-of-material: a new product model. *Intl. Journal of Production Economics*, 23(1):117–128, 1991.

[7] J. Jiao, M. M. Tseng, Q. Ma, and Y. Zou. Generic bill-of-materials-and-operations for high-variety production management. *Concurrent Engineering*, 8(4):297–321, 2000.

[8] J. Jiao, L. Zhang, and S. Pokharel. Process platform planning for variety coordination from design to production in mass customization manufacturing. *Eng. Man., IEEE Trans. on*, 54(1):112–129, 2007.

[9] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group.* Forschungsunion, 2013.

[10] N. Keddis, G. Kainz, and A. Zoitl. Capability-based planning and scheduling for adaptable manufacturing systems. In *Emerging Technology and Factory Automation, 2014 IEEE*, pages 1–8. IEEE, 2014.

[11] N. Keddis, G. Kainz, A. Zoitl, and A. Knoll. Modeling production workflows in a mass customization era. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*. IEEE, 2015.

[12] A. Kondeva, V. Aravantinos, L. Hermanns, and L. Hörauf. The sfit tool: Supporting assembly planners to deal with new product variants. In *Emerging Technology and Factory Automation, 2015 IEEE*, 2015.

[13] B. J. Pine. *Mass customization: the new frontier in business competition*. Harvard Business Press, 1999.

[14] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE ’06, pages 136–145. IEEE Computer Society, 2006.

[15] L. Zhang, J. R. Jiao, and S. Pokharel. Process platform and production configuration for integrated manufacturing and service. In *Industrial Informatics, 2006 IEEE Intl. Conf. on*, pages 498–503. IEEE, 2006.