Adding Variants on-the-fly: Modeling Meta-Variability in Dynamic Software Product Lines

Alexander Helleboogh[1], Danny Weyns[1], Klaus Schmid[2], Tom Holvoet[1], Kurt Schelfthout[3], Wim Van Betsbrugge[3]
[1]*DistriNet Labs, Dept. of Computer Science, Katholieke Universiteit Leuven, Belgium*
[2]*Institute of Computer Science, University of Hildesheim, Germany*
[3]*Egemin International N.V., Zwijndrecht, Belgium*
Contact: alexander.helleboogh@cs.kuleuven.be

Abstract

Dynamic software product lines (DSPL) are software product lines (SPL) that support runtime variability. Runtime variability is typically interpreted as binding variation points at runtime. We emphasize meta-variability *as an important dimension of runtime variability in DSPL. Whereas dynamic binding considers the runtime (de)activation of variants within the scope of a given variability model, meta-variability considers runtime changes to the variability model itself. Meta-variability is essential to support long-lived software products that are subject to evolution.*

In this paper, we consider meta-variability in an industrial DSPL that is developed in a joint project with Egemin N.V., a leading company that provides full life cycle support for automated transportation systems (ATS). The contribution of this paper is threefold. First, we introduce a way to model meta-variability in DSPL in an explicit manner. Second, we put forward a meta-variability meta model that extends the variability meta model with concepts that explicitly support meta-variability. Third, we capture and apply meta-variability in an industrial DSPL for automated transportation systems.

1. Introduction

Egemin N.V. is a leading company that provides full life cycle support for automated transportation systems (ATS). An ATS consists of a number of automated guided vehicles (AGVs) to transport goods in a warehouse. ATS products are tailored to the specific requirements of a particular customer such as the kind of materials to be handled (e.g. bulk or packaged materials), the specific layout of the warehouse, the desired characteristics in terms of throughput and availability, the required interaction with humans and other machinery, etc.

In a joint project DistriNet Labs and Egemin N.V. are deploying a software product line (SPL) for ATS [1]. The ATS SPL provides a structured approach for applying customer-specific customizations, improves the quality of the ATS software and increases the productivity for building ATS. However, as the customer requirements typically evolve during the lifespan of a typical ATS, there is growing demand for changing an ATS after installation. In the project, this has lead to extending the ATS SPL with support for dynamic variability, towards a dynamic software product line (DSPL) for ATS. The research presented in this paper is underpinned by our experiences with modeling variability for an ATS DSPL.

A distinctive characteristic of DSPL compared to SPL is the binding time of the variation points. Whereas the binding time of variation points in an SPL is static, a DSPL is characterized by the fact that variation points can be bound dynamically, i.e. when the product is running [2].

However, our experience in the project reveals that variability in a DSPL entails more than *dynamic binding* of variation points specified in the variability model. We experienced that for long-lived software systems that are designed to cope with evolution, the variability model itself can be subject to changes at runtime. An example is a system that supports dynamically adding new variants for particular variation points. We use the term *meta-variability* to refer to anticipated changes that affect the variability model itself.

The contribution of this paper is threefold. First, we introduce a way to model meta-variability in DSPL in an explicit manner. Second, we put forward a *meta-variability meta model* that extends the variability meta model with concepts that explicitly support meta-variability. Third, we capture and apply meta-variability in an industrial ATS DSPL.

This paper is structured as follows. We discuss related work in Section 2. In Section 3, we pinpoint the importance of meta-variability using a typical example scenario in the context of the industrial ATS DSPL and we use an orthogonal variability modeling technique [3] to capture the scenario and to illustrate the problem. In Section 4, we put forward a meta-variability meta model that extends the variability meta model introduced in [3] with explicit concepts to capture meta-variability. In Section 5, we illustrate how the extended model can be used to model the meta-variability scenario of Section 3. We discuss the software architecture of the ATS DSPL in Section 6 and illustrate how meta-variability

is supported. Finally, we reflect on our work and draw conclusions in Section 7

2. Related Work

Variability in an SPL is typically modeled in a specific model, called variability model. A variability model supports the application engineer in deriving products from an SPL. Over the past few years, several variability modeling techniques have been proposed. Currently, most approaches rely on feature-based modelling of variability (e.g., FODA [4], featureRSEB [5], Riebisch et al. [6], Forfamel [7], RequiLine [8], cardinality-based feature modelling [9], and COVAMOF [10]), though the interpretation of a feature in these approaches varies. Another set of approaches represents variability in terms of decisions that must be made by an application engineer in order to arrive at a specific product (so-called decision modelling). Examples of this category include the RSP [11], the approach by Schmid and John [12], Decision King [13]). Other approaches include VSL [14] or the OVM [3]. A survey of variability modelling techniques can be found in [15].

While mostly the focus of software product line engineering has implicitly been on binding at design or compile time [3], [16], more recently dynamic binding of variability has become a topic, mostly under the heading of DSPL [17]. Thus, DSPL refers to systems that can be interpreted as a product line, where the change among product line variants is performed during runtime. While none of the aforementioned variability modelling techniques excludes dynamic binding of variability, most of the work does not explicitly support it. In addition there is some work, which aims at bridging the gap between design-time and runtime binding. This type of work focuses on explicitly integrating runtime binding and development time binding in a unified framework. Examples of this type of work are timeline variability [18], [19] and anytime variability [20]. One should note, however, that these approaches assume very specific forms of realizing the variability within the final products.

Staged configuration [21], [22] describes the process of transitioning between different variability models. In staged configuration, at each stage some variability is resolved, leading to a reduced variability diagram for the next stage until none is left. The configuration choices at each stage are specified in a variability model. In this paper, we focus on meta-variability to specify the way the configuration choices defined in the variability model of a particular stage can evolve, and in particular be extended, as a result from software evolutions such as dynamic software updates.

In [23], meta-variability is defined as "variability with respect to basic variability attributes." Examples of meta-variability mentioned by the authors are the binding time of variation points and changes in constraints. The case study in [23] only addresses the modification of binding times and is thus mostly comparable to timeline variability [18] and anytime variability [20]. In this paper, we use meta-variability to refer to changes that affect the variability model itself. Rather than switching the binding time of variation points between static and dynamic, we support meta-variability in terms of dynamically adding and removing variants in a DSPL in order to meet the evolution requirements of long-lived software systems.

3. Example Case

We start this section with a brief introduction of AGV transportation systems in Section 3.1. Subsequently, in Section 3.2 we describe the part of the variability model that captures dynamic variability of the ATS DSPL. Finally, in Section 3.3 we introduce a scenario in which this variability model falls short.

3.1. Automated Transportation Systems

Figure 1. AGV at work in DaimlerChrysler

An automated transportation system (ATS) consists of a number of AGVs (see Figure 1). AGVs are fully automated, custom made vehicles that are are instructed by control software to perform transportation tasks. Figure 2 shows the typical setup of an ATS.

- The right hand side depicts a number of AGVs. AGVs are provided with control software connected to sensors and actuators to move safely through the warehouse. While moving, the vehicles follow specific paths in the warehouse by means of a navigation system which uses stationary beacons in the work area, typically laser reflectors on walls or magnet strips in the floor. AGVs are equipped with infrastructure for wireless communication. Important functionalities of the AGV control software include transport assignment to negotiate with

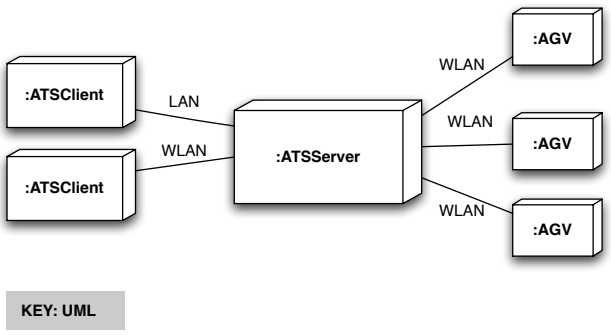

Figure 2. Setup of an Automated Transportation System.

other AGVs which transports to execute, and routing to navigate over the available paths to a particular destination.

- In the center, the ATS server is depicted. The ATS server uses wireless communication to poll the status of AGVs and send tasks to AGVs. The ATS server maintains a global view on the system as a whole, and is responsible for ensuring that all transports are handled by the AGVs. Functionalities of the ATS server include location management of all loads in the warehouse, announcing new transport tasks to AGVs and tracking transports.
- On the left hand side, a number of ATS clients is depicted. An ATS client is responsible for providing an interface to human operators by means of modules for visualization, diagnosis, statistics, and manual assignment of tasks to AGVs. ATS clients can be desktop computers or handheld devices that are used to interact with the ATS.

For each customer, an ATS is composed that matches the customer's specific requirements. ATS can differ in terms of number and type of AGVs, characteristics of materials to be handled, warehouse layout, interfacing with legacy ERP (Enterprise Resource Planning) systems, required level of human control, etc. To support deploying ATS that are tailored to each customer while improving quality and reducing development time and costs, Egemin has recently adopted an SPL approach. Reoccurring functionalities are extracted and reified as core assets and a structured approach is used to use these assets to derive an ATS product tailored to each customer.

However, in the market for ATS there is an increasing demand for dynamic variability. Customers require more features to be dynamically adjustable. The demand for dynamic variability is fueled by (1) the long lifespan of an ATS, during which user requirements are likely to change, (2) the high degree of availability that is required of an ATS makes taking an ATS offline undesirable, and (3) the dynamic and changing operating conditions that occur in the warehouse of the ATS itself. This growing market for dynamic variability in ATS has triggered the evolution of Egemin's SPL towards a dynamic SPL. An important capability that can be dynamically adjusted in the ATS DSPL is the operation mode of AGVs. Typically, the AGVs of an ATS are used in *standard operation mode*. In this mode, AGVs transport goods within a warehouse, and employ algorithms for transport assignment and routing that are tailored to this kind of work.

However, customers asked to extend the use of the ATS beyond their standard operation mode. For example:

- Some customers have sites where periodically goods arrive by truck, and would like the AGVs to assist in unloading the trucks. This requires the AGVs to switch to an *unload operation mode* that uses different mechanisms for transport assignment and routing due to the specific nature of the transport tasks (in terms of frequency, urgency and spatial distribution of these tasks) to unload a truck.
- Other customers want to use AGVs to carry out maintenance tasks during off-peak periods. Maintenance tasks include repositioning goods across the warehouse to optimize accessibility based on demand statistics from the past. This requires the AGVs to switch to a *maintenance operation mode* in which tasks are treated less urgently, such that a proportion of AGV can recharge its battery or park and wait on stand-by.

Depending on the range of different operating conditions that occur at the customer's site, the AGVs of a particular ATS are loaded with a custom set of operation modes that can be switched at runtime.

3.2. Example Variability Model

Figure 3 depicts the part of the variability model that captures transport assignment and routing in the ATS DSPL. We use the notation for orthogonal variability used in [3].

To support different operation modes for AGVs, two variation points that can be bound at runtime:

- *TA mechanism.* The TA mechanism variation point allows selecting a transport assignment (TA) mechanism for AGVs in the ATS. For an extensive comparison of the trade-offs between different TA mechanisms for AGVs, see [24]. In Figure 3 there are two variants for the TA mechanism variation point:
  - *RuBaTA* is Rule-Based transport assignment mechanism that relies on a set of layout-specific rules to assign transports to AGVs. An example of such a rule is "if an AGV arrives at node 18 not carrying a load, and there is a transport requested for a load next to node 18, then the AGV should always pick up the load next to node 18." RuBaTa is typically used in *standard operation mode*.

  - *CNET* uses a Contract Net protocol to assign transports among AGVs. In CNET, an initiator that offers a task calls for proposals and participants offer proposals to perform the task. When the initiator has received the proposals from all participants, it evaluates the proposals and assigns the task to the participant with the best offer. CNET is typically used in *unload operation mode*.

- *Routing Mechanism.* The Routing Mechanism variation point enables selecting the routing mechanism used by the AGVs in an ATS. In Figure 3 there are two variants for the Routing Mechamism variation point:
  - *A* Routing* [25] is a shortest path routing mechanism. A* routing is a static routing mechanism: once the destination has been chosen, the route is fixed and cannot be reconsidered.
  - *Dynamic A* Routing* is an approach that takes into account a dynamic cost based on the traffic caused by other AGVs on the road network. Based on the traffic, the route to reach a particular destination is continuously being reconsidered. Dynamic A* Routing is typically used in *unload operation mode*, where there is a high probability that AGVs hinder each other near the truck.

There is a constraint dependency between the variants of the TA mechanism and variants the Routing mechanism. The RuBaTA variant of task assignment requires the A* Routing variant. The fixed set of rules in RuBaTA cannot be combined with a dynamic routing of the Dynamic A* Routing variant. The CNET variant of task assignment can be combined with either A* Routing or Dynamic A* Routing.

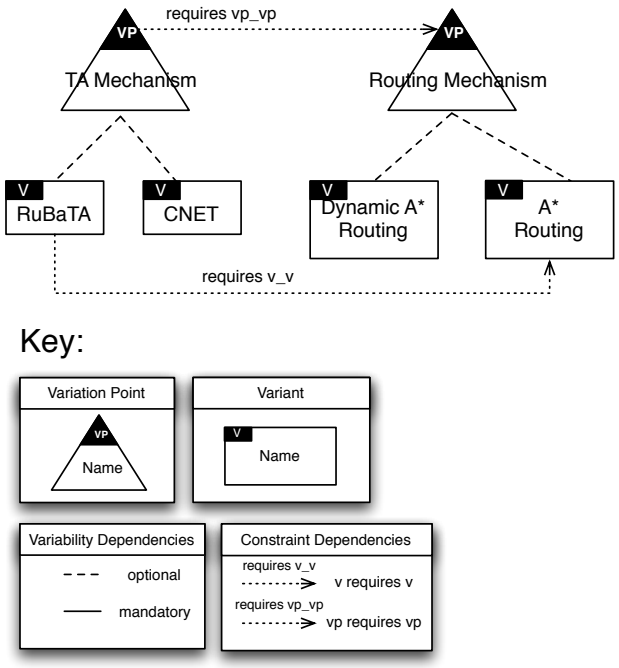

Figure 3. Orthogonal variability model for task assignment and routing in an ATS.

3.3. Evolution Scenario

A variability model precisely defines all variability supported by a DSPL. We now illustrate the problem we experienced when applying variability modeling techniques to capture dynamic variability in an ATS DSPL. We describe an example scenario of runtime variability that is supported in the ATS DSPL but remains implicit in the variability model.

3.3.1. Scenario: Adding TA Mechanisms. A particular kind of runtime variability supported by the ATS DSPL is adding TA mechanisms on the fly. For example, after deployment the existing AGVs of the ATS of a particular customer can be updated with a new Transport Assignment mechanism called Dynamic Contract Net (DynCNET) [24]. The default DynCNET protocol consists of four steps: (1) the initiator sends a call for proposals; (2) the participants respond with proposals; (3) the initiator notifies the provisional winner; and finally, (4) the selected participant informs the initiator that the task is started. These four steps are basically the same as in the standard CNET protocol. The flexibility of DynCNET is based on the possible revision of the provisional task assignment between the third and fourth step of the protocol. DynCNET enables AGVs to continuously reconsider the situation in the environment and the commitment to a transport task is delayed until the load is actually picked, which improves the flexibility of the system. For ATS that have to deal with unpredictable transport streams, this improves performance a lot [24], and these ATS are updated with DynCNET. However, due to frequent transport renegotiations, DynCNET requires a much higher communication bandwidth, preventing DynCNET being applied in some ATS.

As TA mechanisms of an ATS DSPL are subject to frequent evolution, dynamically updating TA mechanisms is identified as an important form of dynamic variability that the ATS DSPL must be designed to cope with. Without explicit support, adding a TA mechanism to an ATS DSPL is typically a very costly process: the ATS DSPL has to be shut down, the new TA mechanism must be added to the ATS DSPL and the updated ATS DSPL must be redeployed on all machines.

In essence, adding DynCNET results in evolving the variability model from Figure 3 into Figure 4. However, the point is that the variability model of Figure 3 does not explicitly describe that the ATS DSPL supports adding new

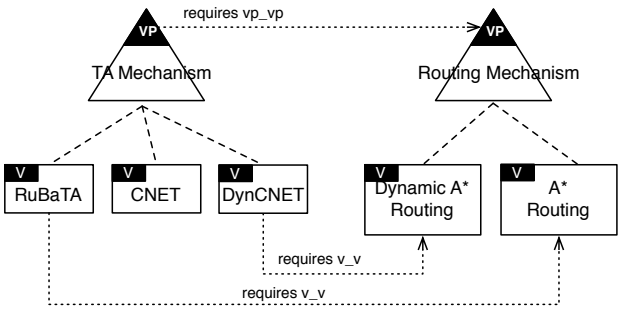

Figure 4. Orthogonal variability model of Figure 3 extended with the DynCNET variant. The key is the same as in Figure 3.

variants or variant constraint dependencies for the TA Mechanism variation point (and, vice versa, that such updates are not supported for the Routing Mechanism variation point).

3.3.2. Framing the Problem. A DSPL can be endowed with explicit support for dynamic updates. Such updates are a form of runtime variability that is anticipated by the designers of a DSPL, and hence should be documented explicitly in the variability model of a DSPL. However, to our knowledge existing approaches to variability modeling lack support for documenting how a DSPL supports anticipated changes to the variability model itself, allowing particular parts (e.g. variants and constraint dependencies) of the variability model to alter in a well-defined way.

We employ the term *meta-variability* to denote runtime changes that are supported by the DSPL and have an impact on the variability model itself. Currently, there is no explicit way to document meta-variability in variability models.

4. A Meta Model for Meta Variability

In this section, we explore meta-variability in DSPL to a further extent. We describe basic concepts to capture meta-variability. We start from the variability meta model defined in [3] using UML 2 notation. Afterwards, we put forward a meta-variability meta model that extends the variability meta model with explicit modeling constructs to capture meta-variability.

Figure 5 shows the variability meta model together with the meta-variability meta model. We elaborate on both meta models.

4.1. Variability Meta Model

The lower part of Figure 5 is the variability meta model described in [3].

The two central elements are *Variation Point* and *Variant*. A Variation Point is a representation of a variable item within domain artefacts, whereas a Variant is a representation of a particular instance of a variable item within domain artefacts. The *Binding Time* associated with a Variation Point represents the time that Variants of that Variation Point are to be bound.

A *Variability Dependency* is an association class that states that a Variation Point offers a certain Variant. A distinction is made between *Optional* and *Mandatory* variability dependencies, expressing that a Variant is optional or mandatory if the corresponding variation point is selected. The *Alternative Choice* groups a set of optional variants of the same variation point and defines a range for the amount of optional variants to be selected in the group.

Different kinds of variability constraints can be used to express restrictions in a variability model.

A *Variant Constraint Dependency* describes a relation between two variants: *Requires_V_V* expresses that the selection of a variant V_1 requires the selection of another variant V_2. *Excludes_V_V* expresses that the selection of a variant V_1 excludes the selection of another variant V_2.

A *Variant to Variation Point Constraint Dependency* describes a relation between a variant and a variation point: *Requires_V_VP* expresses that the selection of a variant V requires the consideration a variation point VP. *Excludes_V_VP* expresses that the selection of a variant V excludes the consideration of a variation point VP.

A *Variation Point to Variation Point Constraint Dependency* describes a relation between two variation points: *Requires_VP_VP* expresses that a variation point VP_1 requires the consideration of another variation point VP_2. *Excludes_VP_VP* expresses that a variation point VP_1 excludes the consideration of another variation point VP_2.

Finally, the meta model supports traceability between a variability model and development artefacts. The *Development Artefact* class represents a software development artefact, such as requirements artefacts, design artefacts, realization artefacts and test artefacts. An *Artefact Dependency* captures that a variant is realized by one or several development artefacts. A *VP Artefact Dependency* captures that a variation point is represented by one or several development artefacts.

4.2. Meta Variability Meta Model

The upper part of Figure 5 is the *meta*-variability meta model. In fact, the meta-variability meta model is a specialization of the variability meta model: the meta-variability meta model defines constructs for describing meta-variability by refining the concepts in the variability meta model.

A meta-variability model describes the variability of *Variability Model Artefacts*. A Variability Model Artefact represents an entity in the variability model. Any class in

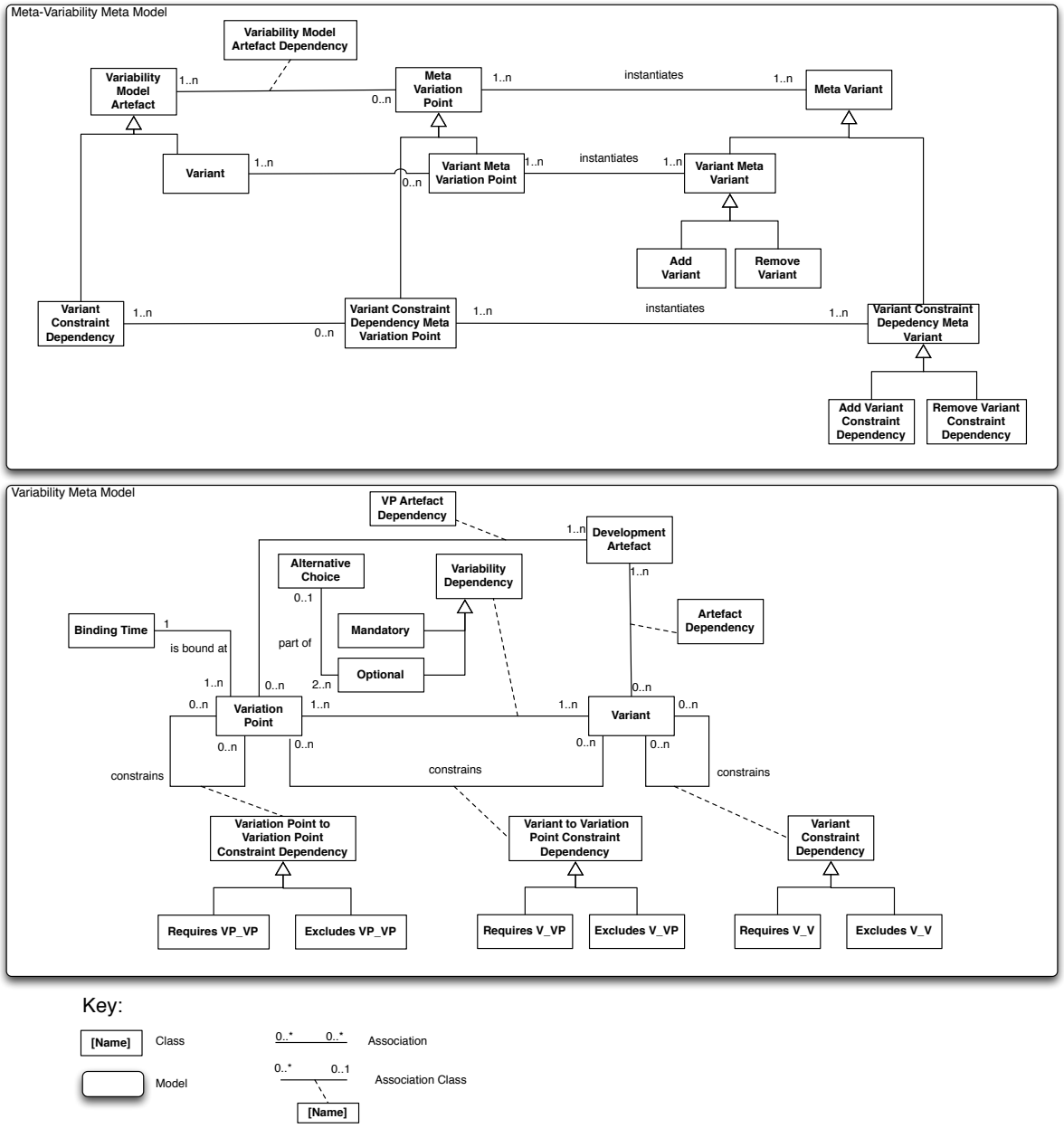

Figure 5. The lower part of Figure 5 is the variability meta model described in [3]. The upper part of Figure 5 is the meta-variability meta model.

the Variability Meta Model described in Section 4.1 is a specialization of the Variability Model Artefact class. However, we only include Variant and Variation Point Dependency as specialization of Variability Model Artefact, to match the focus of this paper.

The two central elements are *Meta Variation Point* and *Meta Variant*.

A *Meta Variation Point (MVP)* is a representation of a variable item with respect to a Variability Model Artefact. A meta variation point is further specialized in a *Variant MVP* and a *Variant Constraint Dependency MVP*. A Variant MVP is a representation of a variable item with respect to the variants in a variability model. A Variant Constraint Dependency MVP is a representation of a variable item with respect to the variant constraint dependencies in a variability model. A *Variability Model Artefact Dependency* captures the relation between a meta variation point a the variability model artefacts it describes the variability of.

A *Meta Variant* is a representation of a particular instance of a variable item with respect to the variability model. A meta variant is further specialized in a *Variant Meta Variant* and a *Variant Constraint Dependency Meta Variant*.

A Variant Meta Variant is associated with a Variant MVP and represents a particular instance of a variable item with respect to variants in the variability model. It is specialized in two classes: *Add Variant* and *Remove Variant*. Add Variant represents the creation of a new variant in a variability model. Remove Variant represents the removal of variants from the variability model.

A Variant Constraint Dependency Meta Variant is associated with a Variant Constraint Dependency MVP and represents a particular instance of a variable item with respect to variant constraint dependencies in the variability model. It is specialized in two classes: *Add Variant Constraint Dependency* and *Remove Variant Constraint Dependency*. Add Variant Constraint Dependency represents the creation of a new variant constraint dependency in a variability model. Remove Variant Constraint Dependency represents the removal of variant constraint dependencies from the variability model.

5. Modeling Meta-Variability

We use the concepts of the meta-variability meta model described in the previous section to extend the variability model of Figure 3. Afterwards, we revisit the example scenario and discuss how it is supported in the extended model.

5.1. Example Variability Model augmented with Meta Variability

Figure 6 shows the example variability model of Figure 3 extended with a meta-variability model.

The lower part of Figure 6 is the variability model depicted in Figure 3. The upper part of Figure 6 is a meta-variability model that explicitly documents meta-variability that is supported with respect to the variability model.

The meta-variability model contains two meta variation points that describe what changes to the underlying variability model are supported in the ATS DSPL. We describe each of the meta variation points and their associated meta variants in detail.

- *Manage TA Mechanism Variants* is a Variant Meta Variation Point that explicitly documents how the variants at the TA Mechanism variation point in the variability model can evolve at runtime. The two meta variants document that TA Mechanism variants may be added or removed.
- *Manage TA Requires_v_v Constraints* is a Variant Constraint Dependency Meta Variation Point that explicitly documents how variant constraint dependencies of TA Mechanism variants in the variability model can evolve at runtime. The two meta variants document that variant constraint dependencies of TA Mechanism variants may be added or removed.

5.2. Scenario Revisited

We describe how Figure 6 offers support for the scenario described in Section 3.3 in which a DynCNET mechanism for transport assignment is loaded on the existing AGVs of the ATS of a particular customer.

In this scenario, the DSPL supports dynamically updating the TA Mechanism variation point with new variants, e.g. for DynCNET, and new variant constraint dependencies, e.g. between the DynCNET variant and the Dynamic A* Routing variant. Recall that this kind of dynamic variability is not explicitly supported in the variability model depicted in Figure 3.

However, in Figure 6, this scenario is explicitly supported in the meta-variability model: the *Add TA Variant* Meta Variant of the *Manage TA Mechanism Variants* Meta Variation Point describes that a DynCNET variant can be dynamically added to the TA mechanism variation point of the variability model. Moreover, the *Manage TA Requires_v_v Constraints* Meta Variation Point documents that a new Variant Constraint Dependency can be created between the newly created DynCNET variant and the Dynamic A* Routing variant.

We conclude that the meta-variability model of Figure 6 explicitly captures that specific parts of the variability model depicted in Figure 3 must have support for particular dynamic updates.

6. Architectural Support for Meta-Variability

In this section, we zoom in on the software architecture of the ATS DSPL and illustrate how dynamic variability and meta-variability are supported in the concrete case of the ATS DSPL.

6.1. Overview of the Software Architecture

Figure 7 zooms in on Figure 2 and shows the main components and connectors of the software architecture of an ATS DSPL.

At an architectural level, the ATS DSPL uses an *ATS Client Component Manager*, an *ATS Server Component Manager* and an *ATS AGV Component Manager* to support variability for ATS Clients, ATS Servers and AGVs respectively. The component managers allow an ATS to be derived from the ATS DSPL by instantiating a custom set of client components, server components and AGV components that suit the requirements of a a particular customer.

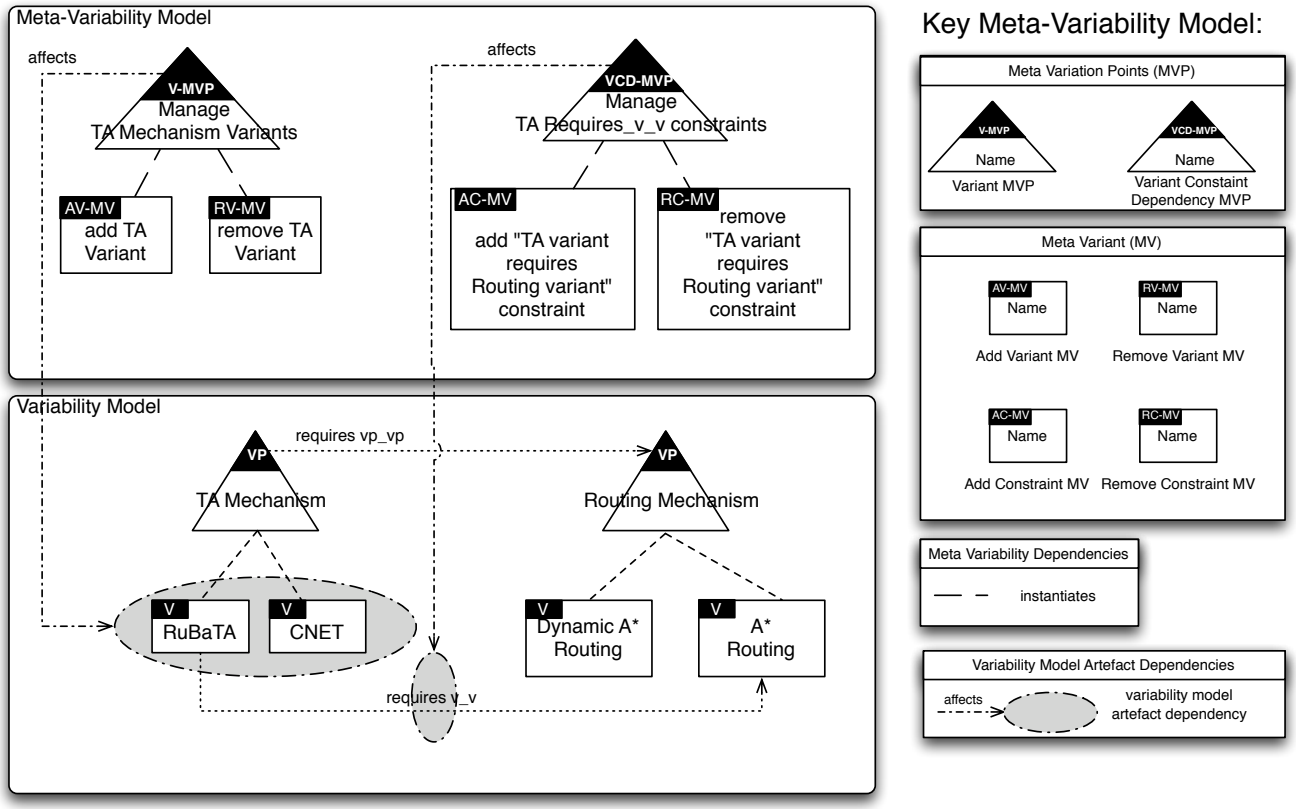

Figure 6. Orthogonal variability model for task assignment and routing in an ATS extended with meta-variability. The key for the variability model is the same as in Figure 3.

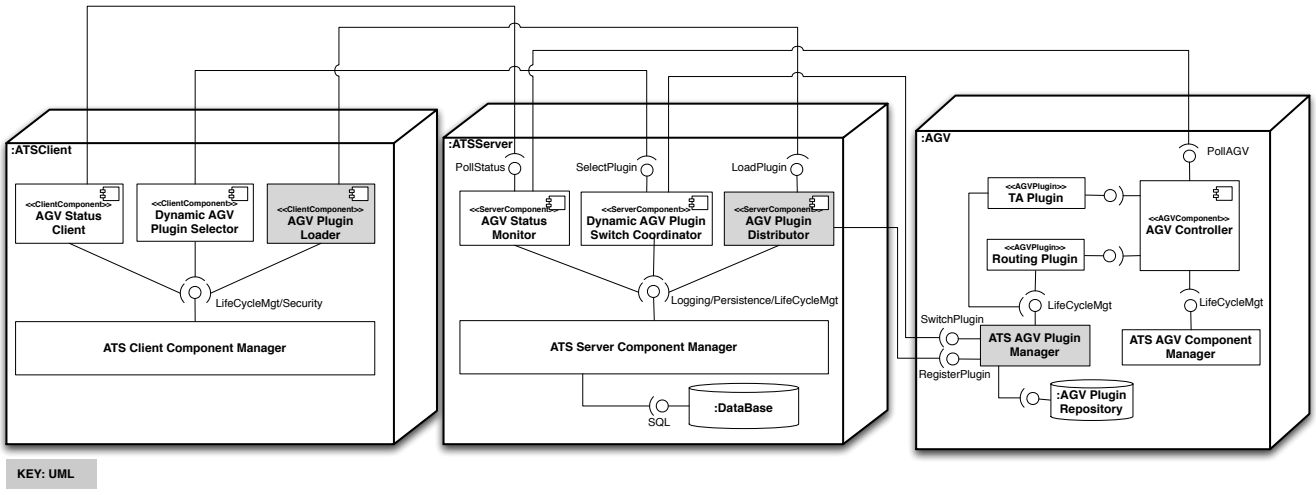

Figure 7. Architectural support for meta-variability in a dynamic software product line for ATS. The shaded components offer support for meta-variability with respect to the transport assignment mechanisms of an AGV.

Client components provide a graphical interface to interact with the ATS. The *ATS Client Component Manager* offers services such as life cycle management and security to support client components. An example of a client component is the *AGV Status Client* that enables a human operator to inspect the status of AGVs.

Server Components provide the application logic of an ATS. Typical services the *ATS Server Component Manager* offers to support server components are life cycle management, logging and persistence. An example of a server component is *AGV Status Monitor*, responsible for polling the status of AGVs and publishing it to AGV Status Clients. Status information that can be retrieved from an AGV includes the position of the AGV, the log of transports, whether there is currently a load on the lift, diagnostic information about the battery level, etc.

AGV Components provide the functionality to control an AGV. An example is an *AGV Controller* that contains all logic to interact with the sensors and actuators on a physical AGV. Specific AGV Controllers exist for each type of physical AGV.

6.2. Supporting Dynamic Variability

We illustrate how the runtime binding of variation points defined in Figure 3 is supported in the software architecture of the ATS DSPL. Dynamic variability of transport assignment mechanisms and routing mechanisms is supported by representing them as *AGV Plugins* that can be dynamically bound and unbound. The *ATS AGV Plugin Manager* deployed on an AGV is responsible for dynamically binding and unbinding AGV plugins. The *AGV Plugin Repository* stores the set of AGV Plugins that can be bound/unbound at runtime.

Currently, the selection of the AGV plugins for transport assignment and routing is done using the *Dynamic AGV Plugin Selector* client. Concrete examples of such clients are (1) a switch on the wall that can be pushed by a human operator to (de)activate *unload operation mode* on all AGVs upon arrival or departure of a truck and (2) an automated timer that activates the *maintenance operation mode* on all AGVs during nights and weekends. The AGV Plugin Selector client notifies the *Dynamic AGV Plugin Switch Coordinator* that is responsible for coordinating the plugin switch on all AGVs. The Dynamic AGV Plugin Switch Coordinator contacts the *ATS AGV Plugin Manager* on each AGV to activate the selected plugins.

6.3. Supporting Meta Variability

We illustrate that meta-variability defined in Figure 6 has resulted in additional infrastructure in the software architecture of the ATS DSPL. Via an additional *Register-Plugin* interface, the *ATS AGV Plugin Manager* supports manipulations to the set of AGV plugins that are available in the *AGV Plugin Repository*.

The *AGV Plugin Loader* client was created to support an administrator in manipulating the set of available AGV plugins. The *AGV Plugin Loader* client is connected to the *AGV Plugin Distributor* on the ATS server. The *AGV Plugin Distributor* is responsible for distributing the changes to the set of available AGV Plugins made by the administrator to all AGVs in the ATS by contacting the *ATS AGV Plugin Manager* on each AGV to store or remove AGV Plugins in the *AGV Plugin Repository*.

7. Conclusion

Long-lived systems are subject to evolution. Features of such systems are subject to dynamic updates, due to changes in customer requirements, deployment context or technology. A DSPL can be designed to explicitly support particular dynamic update scenarios. As a consequence, the variability and the associated variability model of such systems are not static, but subject to anticipated changes.

In this paper, we emphasize meta-variability as an essential concept to support evolution in DSPL. We have illustrated that concrete examples of meta-variability such as adding or removing variants and constraint dependencies are deeply grounded in industrial practice. So far, support was lacking to express meta-variability in DSPL in an explicit manner. We introduced a *meta-variability model* as an explicit artefact to document the way the variability itself of a DSPL can evolve at runtime. From our experience, describing a meta-variability model enables DSPL engineers to anticipate future updates and explicitly capture the way such updates are supported.

Finally, we introduced a *meta-variability meta model* that clarifies and relates the main concepts to express meta-variability and that shows how variability and meta-variability are related. Besides consolidating our current knowledge, the meta-variability meta model is an important stimulus for future work on meta-variability. Currently, we only considered meta-variability with respect to the variants and variant constraint dependencies in a variability model. An important track for future research is to investigate meta-variation points and meta-variants related to other artefacts of a variability model, e.g. to describe the evolution of variation points, variability dependencies and artefact dependencies. Other challenges are to investigate in depth the relation between meta-variability and MDA, to develop a formal underpinning for meta-variability, to extend other variability modeling techniques with support for meta-variability, to extend the meta-variability meta model with dependencies to development artefacts, and to investigate in depth the impact of meta-variability on requirements specifications, software architectures, frameworks and testsuites for DSPL.

Acknowledgment

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven and the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). Danny is supported by the Foundation for Scientific Research in Flanders (FWO-Vlaanderen).

References

[1] D. Weyns, A. Helleboogh, T. Holvoet, K. Schelfthout, and W. V. Betsbrugge, "Towards of software product line for automated transportation systems," in *Proceedings of the Second International Workshop on Dynamic Software Product Lines pages*, 2008, pp. 105–118.

[2] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, 2008.

[3] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[4] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," CMU/SEI-90-TR-21, SEI, CMU, Tech. Rep., 1990.

[5] M. L. Griss, J. Favaro, and M. d. Alessandro, "Integrating feature modeling with the rseb," in *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*. Washington, DC, USA: IEEE Computer Society, 1998, p. 76.

[6] M. Riebisch, D. Streitferdt, and I. Pashov, "Modeling variability for object-oriented product lines." in *ECOOP Workshops*, ser. Lecture Notes in Computer Science, vol. 3013. Springer, 2003, pp. 165–178.

[7] T. Asikainen, "Modelling methods for managing variability of configurable software product families," Master's thesis, Helsinki University of Technology, 2004.

[8] T. von der Massen and H. Lichter, "Requiline: A requirements engineering tool for software product lines," in *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, ser. Lecture Notes in Computer Science, vol. 3014. Siena, Italy: Springer Verlag, 2003, pp. 168–180.

[9] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," in *Software Process: Improvement and Practice*, vol. 10, no. 1, 2005, pp. 7–29.

[10] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A Framework for Modeling Variability in Software Product Families," in *Proceedings of the Third Software Product Line Conference (SPLC04)*, San Diego, CA, 2004.

[11] *Reuse-Driven Software Processes Guidebook, Version 02.00.03*, Software Productivity Consortium Services Corporation, Technical Report SPC-92019-CMC, 1993.

[12] K. Schmid and I. John, "A Customizable Approach To Full-Life Cycle Variability Management," *Science of Computer Programming*, vol. 53, no. 3, pp. 259–284, 2004.

[13] D. Dhungana, P. Grnbacher, and R. Rabiser, "Decisionking: A flexible and extensible tool for integrated variability modeling," in *Proceedings of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS07)*, 2007.

[14] M. Becker, "Towards a general model of variability in product families," in *Proceedings of the First Workshop on Software Variability Management*, February 2003.

[15] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Inf. Softw. Technol.*, vol. 49, no. 7, pp. 717–739, 2007.

[16] F. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[17] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, 2008.

[18] E. Dolstra, G. Florijn, and E. Visser, "Timeline variability: The variability of binding time of variation," in *Workshop on Software Variability Management*, 2003.

[19] E. Dolstra, G. Florijn, M. de Jonge, and E. Visser, "Capturing timeline variability with transparent configuration environments," in *International Workshop on Software Variability Management*. ICSE Workshop, 2003.

[20] A. van der Hoek, "Design-time product line architectures for any-time variability," *Sci. Comput. Program.*, vol. 53, no. 3, pp. 285–304, 2004.

[21] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multi-level configuration of feature models," *Software Process Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[22] A. Classen, A. Hubaux, and P. Heymans, "A formal semantics for multi-level staged configuration," in *Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'09)*, 2009.

[23] K. Schmid and H. Eichelberger, "Model-based implementation of meta-variability constructs: A case study using aspects." in *Proceedings of the Second International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'08)*, 2008.

[24] D. Weyns, N. Boucké, and T. Holvoet, "A field-based versus a protocol-based approach for adaptive task assignment," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 288–319, 2008.

[25] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.