Enterprise Feature Ontology for Feature-based Product Line Engineering and Operations

Charles Krueger, Paul Clements
BigLever Software
{ckrueger,pclements}@biglever.com

ABSTRACT
Feature trees have been the standard data structure for representing product diversity in feature-based systems and software product line engineering (PLE). For basic product lines of modest size or complexity, one or several modular feature trees can be sufficient for managing the and resolving the variation present across the engineering assets in the systems engineering 'V' — from requirements, to design, through implementation, verification, validation, documentation, and more — in the software, mechanical, and electrical disciplines. However, enterprises seeking to adopt PLE at all levels of their organization, including areas such as product marketing, portfolio planning, manufacturing, supply chain, product sales, product service and maintenance, Internet-of-Things, resource planning, and much more are finding that thousands of non-engineering users need different views and interaction scenarios with a feature diversity representation. This paper describes a feature ontology (a specification of the meaning of terms in the feature modeling realm) that is suitable for managing the feature-based product line engineering and operations in the largest and most complex product line organizations. This ontology is based on layers of abstraction that each incrementally constrain the complexity and combinatorics and targets specific roles in the organization for greater degrees of efficiency, precision, and automation across an entire business enterprise.

CCS CONCEPTS
• Software and its engineering Software product lines;

KEYWORDS
Product line engineering, software product lines, feature modeling, feature profiles, bill-of-features, variation points, product portfolio, product configurator, feature-based product line engineering, PLE factory, enterprise feature ontology

ACM Reference Format: Charles Krueger, Paul Clements, Enterprise Feature Ontology for Feature-Based Product Line Engineering and Operations. *In Proceedings of SPLC '17*, Sevilla, Spain, September 25-29, 2017, 10 pages.

DOI: 10.1145/3106195.3106218

ACM ISBN 978-1-4503-5221-5/17/09…$15.00

http://dx.doi.org/10.1145/3106195.3106218

1. INTRODUCTION
Product line engineering (PLE) is an approach for engineering a portfolio of related products in an efficient manner, taking full advantage of the products' similarities while respecting and managing their differences. By "engineer," we mean all of the activities involved in planning, producing, delivering, deploying, sustaining, and retiring products. However, modern PLE is expanding beyond just engineering and into other enterprise-critical areas such as manufacturing, operations, marketing, supply chain management, portfolio planning, and more. As this occurs, the concept of "feature" that has long held sway in the engineering realm may no longer be appropriate to these new and non-engineering-oriented stakeholders.

Recently, a form of PLE known as *Feature-Based Software and Systems Product Line Engineering* ("Feature-Based PLE") has emerged as a modern, repeatable, codified, and proven-in-practice specialization of generic PLE practice. Supported by industrial-strength automation and methodology, Feature-Based PLE is the subject of an upcoming ISO standard that is in progress with involvement and support from INCOSE through its Product Line Engineering International Working Group [14]. Feature-Based PLE involves automation-supported configuration of engineering artifacts to reflect the feature choices embodied by a product. Configuration of artifacts based on feature choices is a powerful paradigm, and there are many successful applications of Feature-Based PLE in the literature [1][3][5][6][7][13].

As implied by the name, the notion of *feature* lies at the conceptual heart of Feature-Based PLE. We, along with many others, adopt the definition that a feature is a distinguishing characteristic that sets products in a product line apart from each other [11]. We find this definition to be intuitive, easy to teach, and helpfully exclusionary: In Feature-Based PLE, a capability common to all products in a product line is *not* considered a feature.

And, for small product lines built by small organizations, it has (in our experience) been quite sufficient. However, we find that for extremely large product lines (with thousands to millions of members) produced by extremely large enterprises (with many thousands of engineers and large numbers of non-engineering stakeholders), a more textured definition is needed.

And such organizations are in existence today. The automotive product line at General Motors, for instances, comprises 9-10 million vehicle instances in some 30,000 unique electrical/electronic configurations, all put together by some 5,000 product line engineers [20]. As we tour the ontology, this is the kind of organization to bear in mind.

What constitutes a distinguishing characteristic depends on organizational roles. To a test engineer working to fully define a test procedure that can be used for one or more products in a product line, a distinguishing characteristic might be as fine-grained as the color of the start button for a piece of equipment: Is it red or black? Such a distinction could not be further from the mind of, say, a vice president of product strategy in the same company, for whom a distinguishing characteristic might be which of her company's automotive vehicles are going to offer autonomous driving three years from now.

The PLE literature is rife with examples of feature models, typically shown in a notional box-and-line representation such as provided by FODA [11]. A car might have bucket seats or bench seats; a user interface might present in English or German; a smartphone might have a five-inch screen or a seven-inch screen; and so forth. Interestingly, *those* features correspond to *neither* the test engineer's notion of a distinguishing characteristic nor the vice president's. Rather, it would seem that they are somewhere "in between," and of (legitimate) interest to stakeholders somewhere "in between" the test engineer and the vice president.

Indeed, others have recognized that "feature" is a concept in need of elaboration for application to large multi-role organizations; a simple one-size-fits-all notion simply won't do [10][16][18].

This paper takes these observations as a launch point and explores how different organizational roles in a large PLE organization view "distinguishing characteristics" – that is, features. The result of this exploration is a *feature ontology* that defines a partitioned space into which the kinds of features we described above, and more, fit, as well as to describe the kind of enterprise stakeholder interested in each partition. We do this for three reasons:

- We need to recognize that, for very large organizations, a one-size-fits-all definition of "feature" is not helpful for people to understand or to use. Having a fuller and role-specific picture of what constitutes a feature will help such an organization adopt PLE more quickly and effectively.
- Product line enterprises are now recognizing that feature-based variant management also has value in organizational operations "on the wings" of the engineering lifecycle, in areas such as product marketing, portfolio planning, manufacturing, supply chain, product sales, product service and maintenance, resource planning, financial projections, and much more. Thousands of non-engineering users within an enterprise need different views and scenarios to interact with an all-embracing feature representation.
- PLE tools that capture and model features will need to embrace all of the meanings and distinguish among them in order to speak to the many different kinds of users that will be viewing, creating, and putting to use all kinds of features throughout the product line.

2. ONTOLOGIES

This paper presents an enterprise *ontology* for features that is suitable for managing feature-based product line engineering and operations in the largest and most complex product line organizations. An ontology is an "explicit formal specification of the terms in a domain and relations among them" [9], in order to "share common understanding of the structure of information among people or software agents" [15]. The people are those who will use the Feature-Based PLE paradigm. The "software agents" to inform include the PLE tools that are the technological engines of the Feature-Based PLE paradigm.

Our ontology will introduce terminology for features that recognize the kinds of levels to which the Introduction alluded. It will have the following characteristics:

- The levels fully partition the space of features. No feature can be in two levels, which we will call *layers*.
- Features in each layer can be described in terms of the choices they make available, and *instances* that reflect choices actually made.
- Features in each layer can be described to typical roles in a large organization that will find features at that level useful in doing their jobs.
- The layers are connected to each other as follows: The choices made at a particular layer constitute the choices that are available at the next higher level.

Before we present our ontology, it is necessary to explore the form of PLE to which it applies, to see how features and feature choices lead to the production of products.

3. FEATURE-BASED PRODUCT LINE ENGINEERING: AN OVERVIEW

Feature-based PLE uses a reference model that is based on a factory metaphor. The PLE factory produces digital assets across the entire lifecycle for products in a product family, analogous to a conventional factory producing physical assets and products. All development happens inside the factory; the output of the factory is for validation and delivery.

The components of the PLE Factory (Figure 1) are as follows:

1. A product line's **Feature Catalog** is a model of the collection of all of the feature options and variants that are available across the entire product line. Feature Owners, Feature Architects create their respective sections in the feature catalog.
2. A **Bill-of-Features** is a specification for a product in the product line portfolio, rendered in terms of the specific features from the Feature Catalog that are included in the product. In other words, it is a feature-based product specification, defined as an instantiation of the available feature choices in the Feature Catalog. The Bill-of-Features Portfolio is the collection of Bills-of-Features that, together, define the entire product line. Portfolio teams, typically working with product marketing teams, create Bill-of-Features for product families and "flavors", based on the features available in the Feature Catalog.

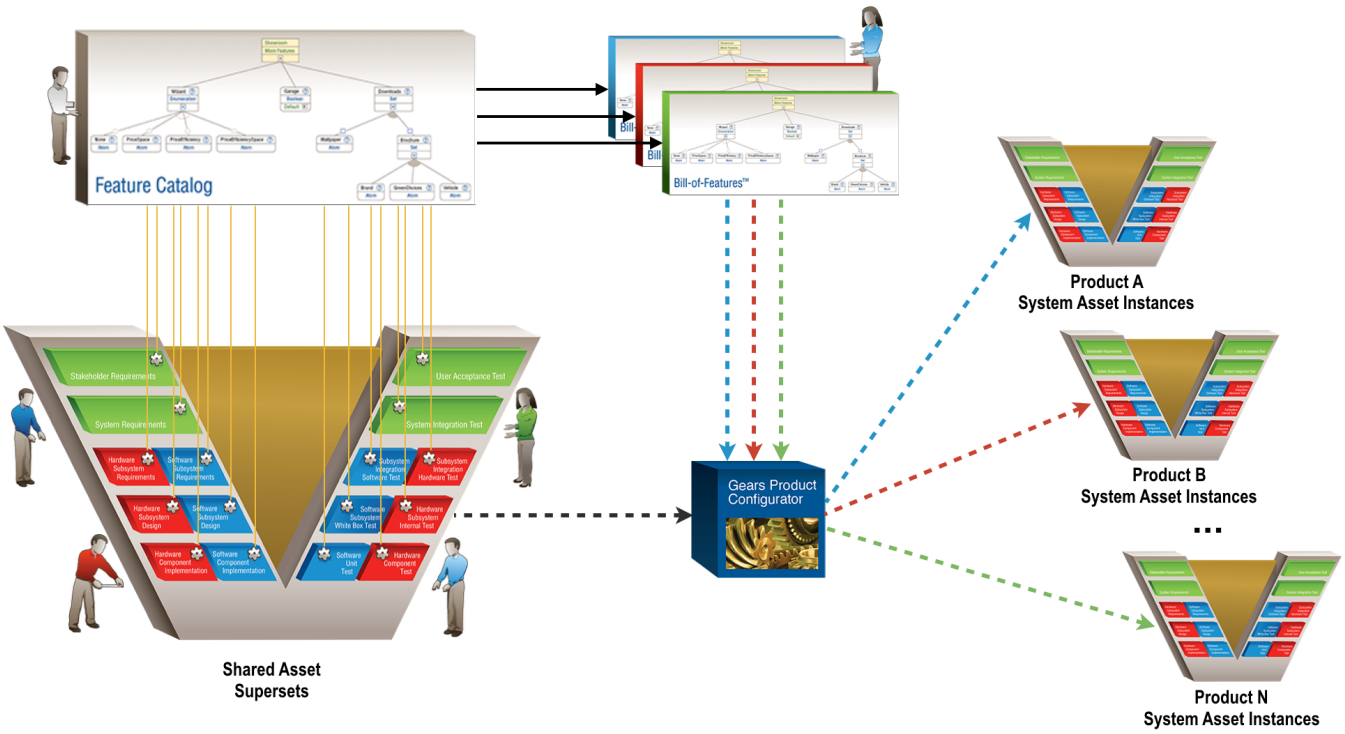

Figure 1 Reference model for the PLE Factory in Feature-Based PLE

3. **Shared assets** are the digital artifacts associated with the systems and software engineering lifecycle of the product line. Shared assets can be whatever digital artifacts compose a part of a delivered product or support the engineering process to create and maintain a product. Shared assets can include, but are not limited to, requirements, design specifications, design models, source code, build files, test plans and test cases, user documentation, repair manuals and installation guides, project budgets, schedules, and work plans, product calibration and configuration files, data models, parts lists, and more. Assets in PLE are engineered to be shared across the product line. In Feature-Based PLE, shared assets are maintained as *supersets*; that is, any content needed by any of the products can be found in the superset. The supersets include *variation points*, which are declarations that specify under what feature choice combinations a specific piece of content is needed and in what form it is needed. Variation point content will be included in a product's digital assets if that feature or feature combination has been chosen, and omitted otherwise. Asset engineers create variation points in their subsystem assets, based on the Features available in the Feature Catalog. Asset engineers typically specialize into their own specific disciplines: Requirements Engineers, System Modeling Engineers, Test Engineers, BoM Engineers, Document Engineers, and so forth.
4. The **PLE Factory Configurator** is the mechanism that automatically produces application assets for the digital twin of a specific product. In Feature-Based PLE, the configurator is an automated tool, as opposed to a manual process. It performs its task by processing the Bill-of-Features for that product, and exercising the shared assets' variation points in light of the feature choices in that Bill-of-Features. The configurator provides the abstraction-driven automation the eliminates the labor-intensive and error-prone activity of manually assembling and modifying engineering assets for the digital twin for each product in the product line. An example is Gears [1], although there are others as well.
5. **System Asset Instances** are product-specific instances of the Shared Asset Supersets, automatically produced by the PLE Factory Configurator. System asset instances are validated and delivered to the next lifecycle phase (e.g., manufacturing), to the customer, or to the market. Suppliers, testers, manufacturing, sales and more use the generated assets, configured by the Bill-of-Features for a system or product family

The factory operates as follows: Shared Asset Supersets are configured by the PLE Factory Configurator based on a Bill-of-Features in the Bill-of-Features Portfolio, derived from the Feature Catalog, to produce a System Asset Instance. The complete collection of digital engineering assets across the systems engineering and software engineering and operations lifecycle for a single physical product is sometimes referred to as the "digital twin" of that product. Thus the Feature-Based PLE Factory in this reference model gives birth to the digital twin for each product in a product line portfolio.

4. VARIATION POINTS: THE ULTIMATE "DISTINGUISHING CHARACTERISTICS"

We begin our exploration of the ontology by returning to our test engineer from the Introduction, intent on correctly specifying for each product whether a start button was red or black (and hundreds of other similar details), so that he or she can produce a correct test spec for each product.

Extrapolating that scenario across the whole panoply of engineering assets that support the set of products in a product line, we can construct a helpful thought experiment. Imagine performing a "diff" operation across all of the engineering artifacts that represent all of the product instances in a product line. It would sort through, for example, requirements specifications, design models, test cases, software, files mechanical parts listed in a Bill of Materials, sections or paragraphs or even individual words in a user's manual, slides in training courseware, and much more. What would the diff operation return? No doubt:

- Words, phrases, or lines of requirements
- Terms or lines of source code
- Words or lines or paragraphs of test procedures
- Model elements or attributes in design specs

And so on – in other words, all of the places where the digital engineering representations of any two products differ from each other.

In a product line comprising thousands to tens of thousands of instances such as in the automotive domain, for example, there may easily be *millions* of these differences, meaning that the variation space is roughly $2^{1,000,000}$. While they are undeniably "distinguishing characteristics," albeit tiny ones, and undeniably important to manage correctly, it becomes immediately clear that they cannot be what we wish to consider as features: No organization could create and manage a bank of millions of features. Nor do these tiny differences lend any insight into meaningful ways in which products differ from each other.

Thus, there cannot be a one-to-one correspondence between a feature and a variation in an engineering artifact. We need a more abstract concept to power the Feature-Based PLE approach in a practical and powerful way.

5. FEATURES ARE ABSTRACTIONS

Let us accept that features cannot correspond one-for-one with variations in product artifacts (which we can now call System Asset Instances based on our picture of Feature-based PLE). In order for a Feature Catalog to be tractable to represent and manage, a feature needs to be more powerful than a switch that turns tiny bits of content on and off in each Shared Asset Superset. Therefore, in order to realize the Feature-based PLE narrative (feature choices are used to configure engineering artifacts), there must be a one-to-*many* correspondence between features and engineering artifact variation[1].

Happily, another word for a one-to-many mapping is *abstraction* [19], which is exactly the concept we need: Features are abstractions of variations; one feature can "drive" multiple (many) artifact-level variations. This one-to-many mapping matches our engineering intuition about features: If a capability is included in a system, then requirements, design elements, source code, test cases, user documentation and more, *all* corresponding to that capability, should be included in the product's digital twin. If the capability is omitted, then all of that material should be omitted.

It also matches our practical intuition about features. To elicit the distinguishing characteristics among products, we would not expect our product portfolio experts to tell us about differences in lines of code or test cases. Rather, we would expect to hear about more abstract differences expressed in terms of capability, function, usage environment, etc.

It turns out that each layer of our feature ontology provides an abstraction – a many-to-one mapping – of the layer beneath it.

6. ABSTRACTIONS HAVE INSTANCES

We only need one more concept before we can describe our ontology, and our test engineer will once again help us to verbalize it.

The test engineer is worried about the button color because on some products the button is red whereas on other products the button is black. That is, there is a *choice available* between red and black, but each individual product will reflect a *choice made* of either red or black.

Likewise, every level in our feature ontology will define a space of *choices available*, against which individual products will be assigned a *choice made*. Bench seats or bucket: Those are the choices available. The sports car will have bucket chosen for it, whereas the sedan will get bench.

With these concepts – different levels of features appropriate for different organizational concerns, features as abstractions, and choices available and choices made – we are ready to introduce our feature ontology.

7. ENTERPRISE FEATURE ONTOLOGY

7.1 ONTOLOGY LAYER 1: PRIMITIVE STANDALONE FEATURES

A primitive standalone feature is a single distinguishing characteristic that can "drive" the setting of one or more variation points in one or more Shared Asset Supersets. It is characterized not in terms of the specific variation point content but rather as a more abstract characterization of a product.

Each primitive standalone feature, by its very existence, constitutes and establishes a choice to be made on behalf of each product. The choice can be whether the feature is present or not, and/or which flavor or flavors of the feature can be chosen. In the latter case, there are also rules about whether the flavors are mutually exclusive or not; that is, whether or not you are allowed to choose more than one flavor for a product. A "flavor" may itself be a primitive standalone feature, leading to the notion of a tree of features.

- **Roles of concern**: Engineers concerned with specific product capabilities work with primitive standalone features.
- **Data structure**: A primitive standalone feature is a name (the name of the feature) and a set of choices available (such as in, out, and/or a list of flavors).
- **Representation**: A primitive standalone feature is conveniently (but not necessarily) represented graphically, using a tree-shaped box-and-line structure. A type may be introduced to constrain the choices; for example:
    - A Boolean type limits the choice to whether the feature is selected or not

[1] This means that each feature serves a role in potentially many variation points. In practice, a variation point's resolution may depend on more than one feature, combined using normal Boolean operators: (Feature1 OR Feature2) AND NOT Feature 3, and so forth.

- An enumeration type requires exactly one flavor to be chosen
- A set type allows any number of flavors to be chosen.
- Flavors themselves may have types such as "integer" or "string," which limit the values they may take on.

- **Instance**: We call a choice made against a primitive standalone feature a *standalone primitive feature instance*. A valid instance obeys the rules imposed by the types.
- **Effect on complexity**: The point of a single primitive standalone feature is that it may configure multiple variation points in multiple shared asset supersets. If it configures on the order of 10 variation points, then our variation space of millions in the shared asset superset realm reduced to a variation space measured in the hundreds of thousands overall. An individual engineer is likely to be concerned with primitive standalone features numbering in the dozens.

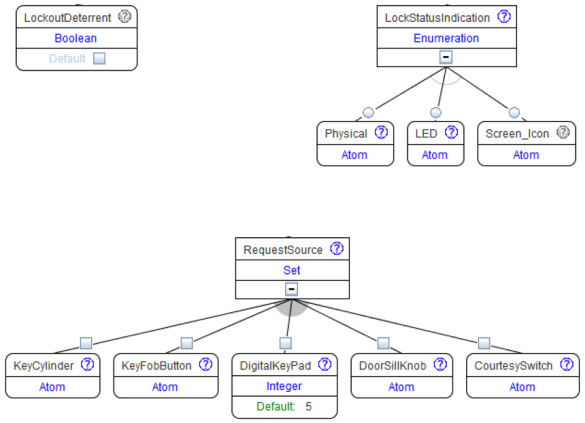

Figure 2 Three examples of primitive standalone features

Making features abstract with respect to specific variation points in specific engineering artifacts reduces the number possibilities by orders of magnitude. A product line as large as the portfolio of a company like General Motors may comprise on the order of ten thousand features like the ones we are describing here [6]. For a company as large as General Motors, with thousands of engineers working in the product line context, ten thousand is a manageable amount.

7.2 ONTOLOGY LAYER 2: BUNDLING PRIMITIVE STANDALONE FEATURES INTO FEATURE MODELS

We find that primitive standalone features are conceptually very helpful but in practice too fine-grained to be an ongoing focus of attention. In product lines of complex systems, product line engineers tend to be assigned to specific areas of knowledge and expertise, and the variation in these areas transcends individual features. A bundled construct is more helpful to let them capture variation in those broader areas. A bundle of individual features also is a place where constraints on choices can be conveniently expressed and captured – for example, that two features are mutually exclusive (so at most one may be chosen) or mutually required (so that zero or both must be chosen).

We call this bundle a *feature model.*

- **Roles of concern**: Product line engineers who are concerned with variation in a specific area of a system that transcends individual features work with feature models. These are typically subject matter experts for a specific area of capability within a product. They may collaborate with the portfolio planning organization to determine which differentiating characteristics should be offered as primitive feature choices. The list of feature profiles that should be offered of also determined and defined by the subject matter experts collaborating with the business to determine which offerings are technologically feasible and provide the best economic return for the enterprise.
- **Data structure**: A feature model is essentially a concatenation of a set of primitive standalone features.
- **Representation**: A feature model is conveniently (but not necessarily) represented as a tree structure, with the roots of the individual primitive features becoming children of a new parent root node.
- **Instance**: An instance of a feature model is called a *feature profile*. Each feature profile is assigned a name, which describes (and stands in for) the set of choices made against the choices made available by the feature model.
- **Effect on complexity**: We find that a typical feature model combines on the order of ten primitive standalone features. Smaller ones are certainly possible. Larger ones are certain possible as well, but we find that feature models with many more than ten features become difficult to grasp. We also find that a typical feature model offers on the order of ten or so feature profiles. Thus, overall, it is not unusual to see feature models with up to 100 or so features offered with a dozen or less feature profiles, thus reducing the available choices by more orders of magnitude. Our complexity space is now around 2^{1000}.

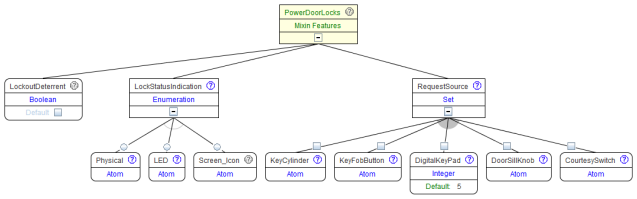

Figure 3 Feature model that bundles the three primitive standalone features of Figure 2

7.2.1 Feature Models as Modularity Constructs

Just as large software systems are composed into coherent modules, to facilitate ease of change and productive work by separate members of a large team [16], so it is with feature models. The bundling manifested by a feature model provides a cleanly packaged and coherent set of features, to achieve ease of change and to enable productive work by members of a team.

Feature models have names and, in practice, generally correspond to specific capabilities, or subsystems, or some other generally acknowledged unit of system decomposition. Feature models may also be used to distinguish between customer-facing features (for example, would you like a cruise control on your car that detects the car ahead of you and keeps you from running into it?) and inward-facing or implementation-oriented features (to detect a car in front of us, shall we use a camera on the front bumper, a LIDAR sensor, or a short-range radar?). Both decisions represent important and legitimate feature choices, but are meaningful to different audiences, and so it is convenient to put them in different feature models. Assuming any of the inward-facing choices would do the job, the actual decision there is based not on

functionality, but on the achievement of quality attributes: *this* sensor costs less, but *that* sensor weighs less and produces less heat; however we happen to have a warehouse full of this *other* sensor, all ready to use.

Just as dividing software into parts requires keen architectural insight and ability, defining the scope of a feature model requires the same kind of architectural thinking. Just as a software module should have high cohesion, features in a feature model should be similarly cohesive, and a feature model should be assigned to an appropriate subject matter expert (or team) to own and manage its content and evolution.

Typically, a particular shared asset, such as a requirements module, source code component, or test suite, won't need all of the features to configure its variation points. Instead, one or a few feature models will often suffice. Later in the ontology, we'll see how to make use of this.

7.2.2 Feature Profiles as Supported Bundles of Primitive Feature Instances

A feature model such as one in Figure 3 only lays out choices that are available. Someone has to actually choose, and we need a way to record the choices.

We could let people browse through the feature model and choose any combination of features present. However, this quickly reveals itself to be an undesirable choice. Combinatorics of feature models are such that even quite modest feature models may lead to many thousands of possible choice combinations – too many to implement and too many to test.

Thus, we don't want to let people pick and choose among individual features. Rather, we want to offer only those sets of pre-packaged choices that (a) contribute to products that are technically and economically feasible to engineer and build; and (b) contribute to products that a customer or the market would actually want to buy. Feature profiles serve that role. A feature profile represents a unique and *supported* configuration of a feature model. These, then, are the combinations we are willing to build and test and offer to the product line at large – not just any combination at random, but a set that has been identified with analysis and forethought.

Figure 4 illustrates a feature profile for the feature model shown in Figure 3. Graphically, it replicates Figure 3 except that now checkboxes are filled in, indicating the choices made.

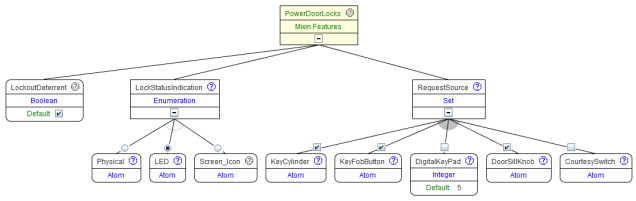

Figure 4 A feature profile for the feature model shown in Figure 3

7.2.3 Feature Model as a Higher-Order Feature Abstraction

The feature model and its list of named feature profiles together represent a higher order feature abstraction in our feature ontology. A named feature model (Power Door Locks in our example) becomes a single higher-order feature that encapsulates and hides the details of its primitive feature model tree. Each named feature profile becomes a single higher-order choice and the list of named feature profiles become a higher-order enumeration of mutually exclusive choices for the higher-order feature model.

7.3 ONTOLOGY LAYER 3: BUNDLING FEATURE MODELS INTO SUBSYSTEM PRODUCTION LINES

After primitive standalone features are assembled into feature models and their individual feature instances bundles into feature profiles, the result is a collection of feature models and feature profiles that can still be quite large – possibly thousands of feature models (and tens of thousands of feature profiles) in a large product line.

Also, our heuristic of not wanting feature models to be too large (and therefore difficult to comprehend, work with, and change) has led us to have multiple feature models that may belong to the same subject matter area. A construct to bundle those would be helpful.

Furthermore, remembering that the ultimate objective of features is to configure shared assets into product-specific System Asset Instances, there needs to be an ontological element that combines one or more feature models and the Shared Asset Supersets that are configured, so that the PLE Factory Configurator can apply the former to the latter.

(It would have been possible to "attach" assets to either of the previously described layers. In fact, for consistency's sake, purists can consider assets to be attachable at any layer. To be attached, the scope of the asset's subject matter should match the scope of the features captured at that layer. In practice, however, we find that the number of assets matching that description for standalone primitive features and individual feature models tends towards zero, so we delayed introducing shared assets until this point in the ontology.)

These combined needs – bundling related and not-too-large feature models while introducing shared assets to be configured by feature choices in a coherent subject matter area – lead us to the next layer in our ontology. This encapsulation for one or more feature models and assets is a *subsystem production line*.

- **Roles of concern**: The roles responsible for engineering the subsystem production line are typically systems engineering experts for a subsystem within a product. They may collaborate with the portfolio planning organization to determine which differentiating characteristics should be offered as higher-order feature choices. The list of production line profiles that should be offered of also determined and defined by the systems engineering experts collaborating with the business to determine which offerings provide the best economic return for the enterprise.
- **Data structure**: The choices available in a subsystem production line are the choices made available by the constituent feature models – that is, their feature profiles. If a production line includes four feature models, then there are four choices available at the production line level – the feature profile for each feature model.
- **Representation**: This can be represented graphically using a tree structure similar to that for feature models. Each feature model would be assigned one node; its feature profiles would be its children; its type would effectively be Enumeration (one choice only). Since the depth of a tree will not exceed one level below the root, another convenient form of representation is a matrix. Each row specifies a set of

choices; there is one column for each feature model. A cell holds the choice of profile for that column's feature model, for that row's choice bundle. (We will assume the matrix form of representation from here on.)

- **Instance**: The instances at this level are named, bundled combinations of instances at the next lower level – that is, feature profiles. For example, if our production line contains feature models PowerDoorLocks (with profiles Passive, Physical, and Auto) and FuelFillerDoorUnlock (with feature profiles Manual, InCabin, and Remote) then instances might be
    - HighEnd: {Auto, Remote}
    - LowEnd: {Physical, Manual}.

  HighEnd and LowEnd would then (using the matrix representation) become the names of matrix rows, and constitute bundled offerings to the next higher layer in the ontology. We call HighEnd and LowEnd *production line profiles.*
- **Effect on complexity**: Because subsystem production lines typically combine on the order of ten or so feature models, our variation space is now roughly 2^{100}.

Figure 5 shows a subsystem production line called Entry Controls that (using the matrix representation) combines the offerings (feature profiles) of three feature models (PowerDoorLocks, RearEnclosureUnlock, and FuelFillerDoorUnlock into seven offerings (production line profiles) of its own. This matrix shows that, for the first three offerings, the choice for FuelFillerDoorUnlock is to omit it entirely – always a possibility.

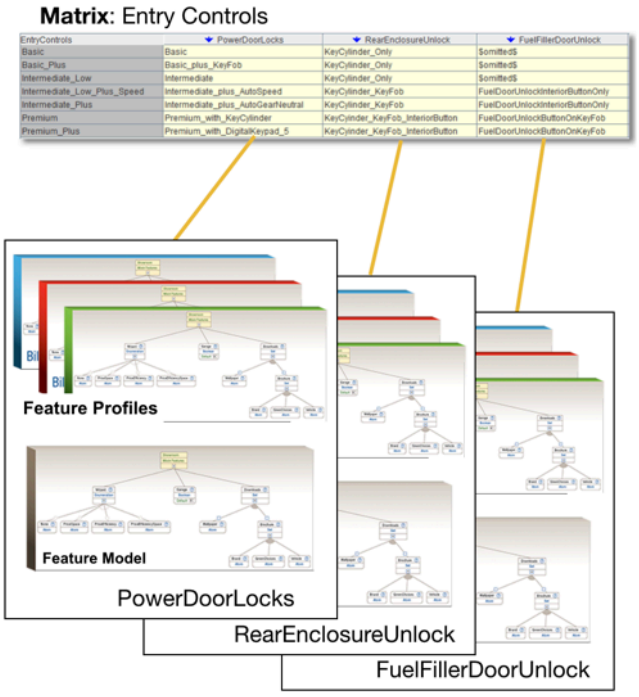

Figure 5 A matrix that combines choices from three feature models.

7.3.1 Subsystem production lines as modularity constructs

Defining the scope of a subsystem production line is, like scoping a feature model, also an architectural activity. We find that, at this level, a production line will often represent a subsystem that is part of the products of the product line, with the shared assets associated with that subsystem.

Each production line has in it the set of feature models that contain the features needed to configure the variation points in those assets. If the feature model is defined to be part of the production line, it can be said to be *owned* by that production line. But sometimes a production line needs to refer to other feature models in order to configure an asset. In that case, it can *import* a feature model (by reference) from the owning production line. A feature model can be owned by one production line, but imported into multiple production lines if its features are crosscutting – a common occurrence. Imported feature models are used to manage feature dependencies and constraints on feature selections across subsystems.

7.3.2 Subsystem Production Lines as Higher-Order Feature Abstractions

The subsystem production line and the list of named production line profiles represent, once again, a higher order feature abstraction in our feature ontology. A named production line (Entry Controls in our example) now becomes a single higher-order feature that encapsulates and hides the low level details in its collection of feature models and feature profile choices. Each named subsystem production line becomes a single higher-order feature offering the choices enumerated in (for example) its matrix. To the outside, our example looks exactly like an Enumeration feature named Entry Controls with seven possible and mutually exclusive flavors.

It is not unusual to see a production line with millions of possible combinations of profile combinations having ten or less production line profiles, thus once again reducing the available choices by orders of magnitude.

7.4 ONTOLOGY LAYER 4: BUNDLING SUBSYSTEM PRODUCTION LINES INTO SYSTEM PRODUCTION LINES

Subsystem production lines tend, in practice, to capture the variation and shared assets associated with a subsystem. A subsystem may be clearly identified in a common architecture for the product line, or it may be informally identified by the presence of a dedicated group of people who work on it, and a collection of shared assets devoted to engineering it.

Clearly another level of scope beckons. We need to bundle subsystem production lines into product offerings at the system or whole-product level.

After all of the feature models are partitioned into subsystem production lines, the result is a collection of production lines that can still be quite large – possibly hundreds or even thousands of production lines in a large product line, as may be found in the automotive industry [20]. There subsystem production lines may exist for brakes, transmissions, lighting, infotainment, entry controls, climate control, and a whole host more.

Since production lines act as higher-order features in the ontology, we can extend the definition of a production line to also include other production lines, resulting in a hierarchy, or tree (see Figure 6). In this way, we can mirror a system-of-systems architecture with a *production-line-of-production-lines hierarchy*. The constituent production lines provide places to make feature decisions corresponding to that part of the system-of-systems structure.

- Roles of concern: Anyone concerned with the variation of systems at the highest level is concerned with the system production lines. This can include portfolio planners and managers, marketers and business development specialists, and the engineering and business leadership in a business area or the enterprise at large.

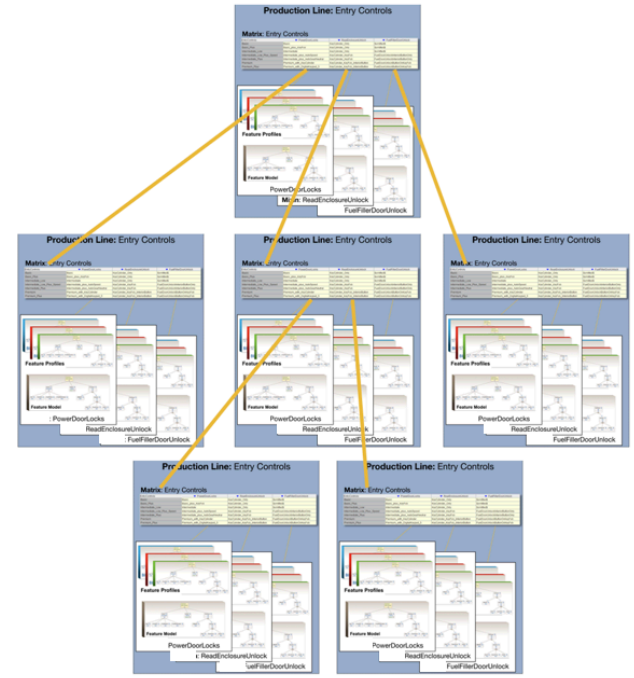

Figure 6 Subsystem production lines bundled into a system production line. An imported production line appears as a column in the matrix of its parent, the importing production line. Rows in the imported production line become available as cell choices in that column.

- **Data structure**: The same as for subsystem production lines.
- **Instance**: Each instance of a system production line is a Bill-of-Features, as illustrated in Figure 1. A Bill-of-Features is the ultimate feature-based description for a whole product.
- **Effect on complexity**: Each level in the production line hierarchy often provides a one- to two-orders of magnitude reduction in the combinatoric complexity by limiting the number of specified profiles relative to the number of possible profiles. A two- or three-level hierarchy (typical, in our experience) yields a complexity space of about 2^{50}.

Once again, defining the production line hierarchy is much like scoping an individual production line or scoping a feature model – it is an architectural activity. In this case the architectural objective is aligning it to a system-of-systems structure. The production line hierarchy can go to any level of depth. Each level in the hierarchy represents a new level of abstraction, where the production line becomes a new higher-order feature and its production line profiles (rows in the production line matrix) become the available choices for that higher order feature.

Finally, although the concept could be applied theoretically at lower levels in the ontology, we find that this is a particularly helpful place to introduce staged configuration [4][12]. This is essentially a mechanism that allows different stakeholders, each with specialized knowledge about the choices appropriate for a product, to make their choices in a way to produce a coherent, consistent whole product.

8. OVERLAY: PROLIFERATION BUNDLES

At this point in our ontology, we are down to defining a product by making choices numbering in the dozens, say 50 or so. For very large production lines, the number might be in the hundreds, but 50 is a nominal number to consider going forward.

Organizations that specify thousands or tens of thousands of products don't want to painstakingly fill out a matrix row for every one of them. Modern PLE Factory Configurators have a "proliferate" feature that can automatically produce rows in a matrix at the leaf of the hierarchy for every combination of profile choices that are still unbound. A proliferated matrix lays out all the choices for all the products that we can now, for example, take to manufacturing.

However, 50 or so choices still leaves us with 2^{50} possible

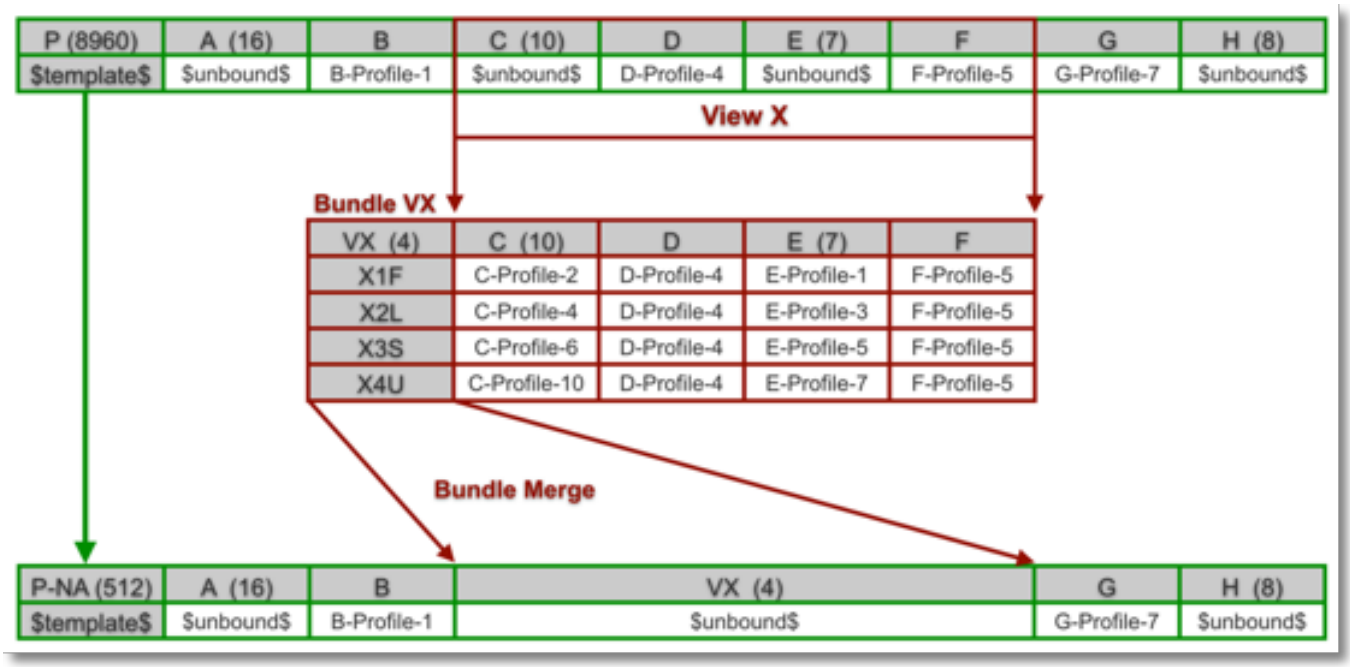

Figure 7: Proliferation bundles

combinations of choices, which means we could define 2^{17} (about 131,000) absolutely unique products for every one of the 2^{33} people on Earth. Before we push the "proliferate" button, we need a way to further pare down the choices.

In the same way that feature profiles reduced the potential complexity inherent in primitive feature models, we will use *proliferation bundles* to whittle down the potential choices. The idea is to create meaningful bundles of the unbound selections to limit proliferation.

Figure 7 illustrates the concept. At the top is a template matrix with eight columns A-H. We have annotated the number of choices that remain available in each column: There are 16 choices available for A, 10 for C, 7 for E, and 8 for H. All other columns have fully bound selections. At this point we have the possibility of 16 x 10 x 7 x 8 = 8960 unique products. This is too many; this wide variety would overwhelm our manufacturing capability.

We realize that we don't need or want the 70 combinations possible from columns C through F. Instead, we decide that we create a bundle called VX that offers 4 combinations instead. By defining that bundle and letting subsequent (descendant) matrices choose from its choices, we have slashed our possible number of products to 512.

These manufacturing bundles, in which combinations of different features are limited and offered as packages, are common in the automotive industry. General Motors calls them "Regular Production Options;" other auto-makers have their own terminology. These so-called "RPO codes" comprise feature bundles that are available to a customer, such as a Sports package that combines a high-performance engine, a particular transmission, a tight suspension system, a prescribed steering wheel cover, external paint trim, and more.

Proliferation bundles have, for very large product lines, finally brought us down into a decision space that is manageable, understandable, and manufacture-able.

We call this construct an overlay to our feature ontology (as opposed to its own layer) because we could have applied it to any layer. Like shared assets, which could have been introduced into the ontology from the beginning but which practicality led us to introduce only when we got to production lines, we find proliferation bundles to be useful only at the system production line level.

9. SUMMARY AND CONCLUSIONS

A feature is an abstraction that can describe variations among products in a way that applies across an enterprise, and be used to configure the associated artifacts.

The concept of feature allows a consistent abstraction to be employed when defining and making choices, from a whole product configuration all the way down to the deployment of components within a low-level subsystem. Features provide the common communication vehicle - a *lingua franca* - among all stakeholders in the product line, from requirements engineers to testers, from marketers to executives, from designers to customers. Our ontology, summarized, is:

- Variation points in shared assets are configured by defined sets of choices (feature profiles) from primitive features.
- Features and profiles are packaged into feature models to achieve modularity.
- Production lines are mini-factories that contain related feature models and assets that they configure.
- Production lines can be structured into a hierarchy, enabling a system of systems to be represented as a production line of production lines.
- Choices available to products can be packaged into proliferation bundles to further reduce combinatoric complexity.

Error! Reference source not found. summarizes our enterprise feature ontology, and shows the kinds of roles involved in decision-making and selection at each level. At each step along the way, the number of choices available to engineers and other enterprise stakeholders working at that level shrinks by many orders of magnitude. A decision space on the order of $2^{1,000,000}$ (at the shared asset variation point level) becomes a decision space on the order of 2^{10}-2^{20}.

Table 1. Enterprise Feature Ontology and Complexity Management

Element	Purpose	Roles that Utilize	Potential Complexity
Overlay: Production Line Proliferation Bundles	Sales Feature Options. Constrain proliferation.	Sales Configuration Engineers	2^{10} - 2^{20}
Level 4: System Production Line	Partially bind and down-select offered Matrix Profile options for a Product Family. Proliferation	Product Portfolio Design Engineers	2^{50}
Level 3: Subsystem Production Line with Product Profiles	Compose and scope feature models and configure shared assets. Provide desired offerings of a system or subsystem product line.	Systems Engineering, Product Line Architects, Product Marketing, Product Portfolio, System and Subsystem Domain Engineers	2^{100} - 2^{1000}
Level 2: Feature Model with Feature Profiles	Modularize and scope related features	Feature Architects	2^{1000} - $2^{10,000}$
Level 1: Primitive Feature	Root-cause Feature Abstraction. functional, nonfunctional, and deployment variation.	System/ Subsystem Feature Designers	$2^{10,000}$
Shared Asset Variation Point	Feature-Based asset variation management	Asset Engineers	$2^{1,000,000}$

For smaller production lines with, say, dozens of product instances, the ontology will produce tractable decision spaces much earlier. In that case, a few of the ontology's lower layers (e.g., features, profiles, and feature models, packaged into a single production line) can suffice; however, in practice, we observe that

most product lines in this size still avail themselves of the separation of concerns brought about by the production-line-of-production-lines structure.

We began with a question: How do you bridge the gap between potentially 1,000,000 variation points in a product line's shared assets down to a few dozen customer-facing decisions? Our feature ontology, which is in industrial use today, is the answer. This ontology came about through practice and experience, not speculation or imagination. As Feature-Based PLE made larger and larger strides into larger and larger product lines (e.g., [6][8]), each layer in the model was added on top of previous layers as a result of need. At each level, we combined large numbers of available choices into smaller numbers of pre-packaged selections. There is no reason the ontology could not be extensible in this way by adding still more layers to the top, should the need arise to work with product lines orders of magnitude larger than the largest ones today. We look forward to seeing Feature-Based PLE applied in those settings.

10. REFERENCES

[1] Acher, M., Collet, P., Lahire, P., France, R. "FAMILIAR: A domain-specific language for large scale management of feature models," Sci. Comput. Program. 78(6): 657-681 2013.

[2] BigLever Software, "BigLever Software Gears," http://www.biglever.com/solution/product.html

[3] Clements, P., Gregg, S., Krueger, C., Lanman, J., Rivera, J., Scharadin, R., Shepherd, J., and Winkler, A., "Second Generation Product Line Engineering Takes Hold in the DoD," Crosstalk, The Journal of Defense Software Engineering, USAF Software Technology Support Center, 2013, in publication.

[4] Czarnecki, K., Helsen, S., Eisenecker, U. "Staged Configuration Using Feature Models," International Conference on Software Product Lines, 2004. dx.doi.org/10.1007/978-3-540-28630-1_17

[5] Dillon, M., Rivera, J., Darbin, R., Clinger, B., "Maximizing U.S. Army Return on Investment Utilizing Software Product-Line Approach," Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), 2012.

[6] Flores, R., Krueger, C., Clements, P. "Mega-Scale Product Line Engineering at General Motors," Proceedings of the 2012 Software Product Line Conference (SPLC), Salvador Brazil, August 2012.

[7] Gregg, S, Scharadin, R., Clements, P. "The More You Do, the More You Save: The Superlinear Cost Avoidance Effect of Systems and Software Product Line Engineering, Proceedings Software Product Line Conference 2015, Nashville, 2015.

[8] Gregg, S., Scharadin, R., LeGore, E., Clements, P. "Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment," Proceedings, Software Product Line Conference 2014, Florence, Italy, 2014.

[9] Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition* **5**: 199-220.

[10] Hubaux, A., Heymans, P., Schobbens, P., Deridder, D. "Towards Multi-view Feature-Based Configuration," International Working Conference on Requirements Engineering: Foundation for Software Quality, 2010. http://dx.doi.org/10.1007/978-3-642-14192-8_12

[11] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; & Peterson, A. "Feature-Oriented Domain Analysis (FODA) Feasibility Study" (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[12] Krueger, C., "Multistage Configuration Trees for Managing Product Family Trees," *Proceedings SPLC2013,* Tokyo, August 2013.

[13] Krueger, C. and Clements, P. "Systems and Software Product Line Engineering," *Encyclopedia of Software Engineering*, Philip A. LaPlante ed., Taylor and Francis, 2013, in publication.

[14] INCOSE Product Line Engineering International Working Group, http://www.incose.org/ChaptersGroups/WorkingGroups/analytic/product-lines, downloaded 09 November 2016.

[15] Noy, Natalya F., and McGuinness, Deborah L., "Ontology Development 101: A Guide to Creating Your First Ontology," Stanford University, Stanford University, Stanford, CA, 94305, http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html, downloaded 09 November 2016.

[16] Parnas, D.L., "On the criteria to be used in decomposing systems into modules," Communications of the ACM, Volume 15 Issue 12, pp. 1053-1058, Dec. 1972.

[17] Rabiser, R., Wolfinger, R., Grunbacher, P. "Three-Level Customization of Software Products Using a Product Line Approach," 42nd Hawaii International Conference on System Sciences, Big Island HI, 2009. https://doi.org/10.1109/HICSS.2009.460

[18] Reiser, M., and Weber, M. "Managing Highly Complex Product Families with Multi-Level Feature Trees," 14th IEEE International Conference on Requirements Engineering, Minneapolis / St. Paul, Sept. 2006. https://doi.org/10.1109/RE.2006.39

[19] Saitta, Lorenza, and Zucker, Jean-Daniel, *Abstraction in Artificial Intelligence and Complex Systems,* Springer Science & Business Media, 2013.

[20] Wozniak, L., Clements, P. "How Automotive Engineering Is Taking Product Line Engineering to the Extreme," *Proc. SPLC 2015*, Nashville, 2015.