2012 Software Product Line Conference
Best Paper Award in Industrial Track

Mega-Scale Product Line Engineering at General Motors

Rick Flores
General Motors
30003 Van Dyke Ave
Warren MI 48093
+1 248 207-3494
rick.r.flores@gm.com

Charles Krueger
BigLever Software
10500 Laurel Hill Cove
Austin, Texas 78730 USA
+1 512 426 2227
ckrueger@biglever.com

Paul Clements
BigLever Software
10500 Laurel Hill Cove
Austin, Texas 78730 USA
+1 512 994 9433
pclements@biglever.com

ABSTRACT
General Motors faces probably the most complex Systems and Software Product Line Engineering (PLE) challenges ever, in terms of product complexity, richness of variation, size of organization, and an unforgiving requirement to support over a dozen simultaneous development streams all geared towards each new model year. To meet this challenge, GM turned to an advanced set of explicitly defined product line engineering solutions, which have been referred to as *Second Generation PLE (2GPLE)*. This includes reliance on features as the *lingua franca* to express product differences in *all phases* of the lifecycle, deeply nested hierarchical product lines, industrial strength automation to provide modeling consistency throughout, and more. This paper explains how 2GPLE is being applied at General Motors, and the technical and organizational lessons learned so far.

Categories and Subject Descriptors
D.2.2 [**Design tools and techniques**]: *product line engineering, software product lines, feature modeling, hierarchical product lines*

General Terms
Management, Design, Economics.

Keywords
Product line engineering, software product lines, feature modeling, feature profiles, bill-of-features, hierarchical product lines, variation points, product baselines, product portfolio, product configurator

1. INTRODUCTION
This is the story of a product line engineering effort under way at General Motors. The product line involves the electronic control systems placed aboard vehicles during manufacturing. These control systems includes electrical components (sensors and actuators), electronic control units laid out in a given topology around the car, wires and data networks to connect the components appropriately, and the software that runs it – all loaded correctly onto each vehicle. Like all product line stories, this one focuses on a particular set of aspects that set this one apart from others. While the effort is very much a work in progress, the piloting and roll-out effort is far enough along to allow us to confidently describe these aspects of the solution:

1. How solving this product line engineering problem requires every dimension of what has come to be called the *second generation approach* to product line engineering. The dimensions that play the largest role in the GM story include (1) consistent and traceable treatment variation points, chosen from a very small set of variation mechanisms, in artifacts from every phase of the engineering lifecycle, from requirements through design and implementation, to deployment on hardware, to calibrations; (2) the role of features as the way in which variation is expressed throughout the product line; and (3) the introduction of deeply nested hierarchical product lines and the ways in which the product lines that populate the hierarchy "interface" with each other and respect the information boundaries that correspond to long-established organizational structures.
2. How a very small but consistent set of product line constructs are proving to be adequate to provide the necessary expressive power for this product line.
3. How the automation that is required to power the product line solution depends not only the its own technical capabilities but also on vendor business partnerships that allow it to work seamlessly with a variety of life cycle engineering tools that store artifacts in proprietary formats – artifacts that need to have variation points injected into them.

These aspects are made compelling because of the unprecedented complexity involved in this product line. If these solutions work here, it is unlikely they will be found inadequate anywhere else.

2. A MEGA-SCALE PRODUCT LINE
General Motors is the largest automotive manufacturer in the world [1]. In 2011 it sold over 9 million vehicles, produced (with its partners) in 31 countries around the world. That works out to over 1,000 vehicles rolling off assembly lines *every hour*.

The product line we describe is built under the Next Generation Tools (NGT) initiative at General Motors. GM introduced NGT to tackle the complexity brought on by (among other things) the introduction of hybrid and alternative-fuel vehicles and new "active safety" features that require intricate and unprecedented orchestration among vehicle subsystems. Product line engineering is a key ingredient of NGT.

SPLC '12, September 02 - 07 2012, Salvador, Brazil

General Motors may well represent the most challenging domain in all of product line engineering. We characterize it as *mega-scale PLE* due to the fact that engineers must deal with multiple product line characteristics that measure in the millions although, as we will see, even this term's implied order of magnitude fails by a wide margin to do justice to the problem space:

1. **The vehicles are complex**. As a group, GM vehicles comprise some 300 engineered subsystems such as brakes, exterior lighting, interior lighting, entry controls, and many more. The Chevrolet Volt runs approximately 10 million lines of code, which is several million more than either the Boeing 787 or the F-35 Joint Strike Fighter [12].

Figure 1 Chevy Volt: Ten million lines of code, ready to roll (© GM Company)

2. **The variation among vehicles is enormous.** GM builds over 60 models under seven brands and divisions. The vehicles may be internal combustion, electric, or both. Customer-visible options include everything from power windows to "lane keep assist" (a system to help the car stay in the correct highway lane). These options, and many dozens more, fundamentally affect the electronics and software aboard the vehicle.

   Legislation, not to mention cultural preferences, in the 150+ countries where GM does business also imposes feature constraints. To choose one of many dozens of examples, there are complex interactions between the vehicle's exterior lights (low beam headlights, high beam headlights, tail lamps, brake lights, parking lights, daytime running lights, front fog lamps, rear fog lamps, cornering lamps, reversing lamps, and hazard flashers) in terms of which lights are allowed, disallowed, or required to come on with which others. The "lead me to my car" feature makes lights come on or flash when the driver presses a button on the key fob. Which lights come on, whether they flash or not, and how long they stay on all are specific to the region and (of course) what exterior lights are actually on the vehicle. The electronics aboard *every* car has to get that behavior right for *that* car.

   A simple thought experiment helps to grasp the astronomical magnitude of the variation involved. We can think of vehicle rolling off an assembly line as the result of making a very large set of yes-or-no decisions. The set of all possible vehicles results from all possible combinations of those yes-or-no choices. The size of that product space is 2^x, where x is the number of decisions. If x>260, then the product space comprises more combinations than the number of atoms in the observable universe [19]. For GM, x is in the low thousands. (The number of variants that GM actually produces is much less than that, obviously – a number in the low tens of thousands.)

3. **Feature interaction abounds.** The lighting example above illustrates interactions within a subsystem (exterior lighting) but other features require complex interactions among completely different subsystems. For example, the presence of "park assist" (a feature to help park the car) requires the presence of a sensor to gauge the car's position relative to the parking space. On some cars this will be a sonar detector, while on others it will be a camera. Park assist also requires brakes that accept software control, and some versions of park assist require particular versions of steering controls. Thus, the presence of a customer-visible feature can affect multiple subsystems, requiring communication and coordination among the subsystems on the car, and among the groups that are responsible for the subsystems involved.

4. **The product line must be in lockstep with current and future model years.** GM has to plan their production years in advance. Features that won't be in the showroom for 3-5 years are already part of today's engineering. And the entire product line marches in unwavering lockstep with the calendar, fixed and unforgiving, which defines each new model year. This means that the product line infrastructure must support concurrent engineering streams for each of the fixed yearly cadences, as well as concurrent development cadences for release cycles scheduled every 6 weeks throughout the year. There may be as many as 15 active, concurrent engineering baselines that engineers must contribute to and coordinate among.

   The temporal dimension of the problem exhibits astronomical complexity as well. Each of the 300 or so GM subsystems will typically undergo enhancements or fixes within 10 or more cadences within a 2 year period, resulting in 10^{300} possible subsystem version combinations. As with the number of feature combinations, this also vastly exceeds the 10^{80} atoms in the observable universe [19].

5. **Consistency and traceability across the life cycle are required.** Each vehicle is the result of an engineering process that spans requirements, design, implementation, calibration, layout and interconnection of electronic control units (ECUs), allocation of software to the ECU network, production of a manufacturing bill-of-materials, and testing. Each of the artifacts must be consistent with each other, in that they must all be accurate with respect to the vehicle to which they apply. Further, that consistency must be demonstrable through feature interdependency constraints as well as traceability among lifecycle phases..

6. **The organization is very large.** Ultimately up to 5,000 engineers will be directly working on artifacts that are part of the product line, some in roles newly defined expressly to support the PLE effort.

The emergence of hybrid and alternative fuel vehicles and new active safety features, which dramatically increase the amount of product line diversity, plus the new economic reality in the automotive industry that leaves little margin for technical error, drove GM to plan to overhaul its engineering tools and processes. The result is the Next Generation Tools (NGT) initiative.

3. NGT AND GM'S MARCH TOWARDS "CONVERGENCE"

Product line engineering at its heart is about sharing, and about eliminating duplication. At GM this is called "convergence," and has been an ongoing work in progress for decades. NGT is the latest manifestation of a long-standing strategy. It began with the merging of branded divisions in 1991, continued with the adoption of a common electrical and electronic architecture and management of requirements to give features a common look and feel in the late 1990s, and grew with the commitment to adopt the AUTOSAR (AUTomotive Open System ARchitecture). open standards for automotive E/E (Electrics/Electronics) architectures [2] in the early 2000's.

To create a roadmap for this new convergence, GM created the Next Generation Tools (NGT) initiative in 2008. NGT was originally intended to answer the question "What common tools and processes shall we all adopt to power this convergence?" GM wanted an open tooling solution, with the ability to obtain the best-of-class life cycle tooling solutions from different vendors. After an extensive search, GM settled on a number of different IBM Rational tools, including

- DOORS for requirements management (precipitating a migration from Microsoft Word, GM's previous choice for requirements)
- Rhapsody for system design and models management
- RPE (Rational Publishing Engine) for documentation production
- Team Concert for change management and Synergy for configuration management

As GM investigated these and other tools, they recognized that AUTOSAR was only one step on the road to a more advanced product line engineering capability. As this realization dawned, GM recognized that none of the tools they had selected were "product-line-aware," and that they needed a tool to manage variation points in their engineering artifacts and help configure vehicle-specific engineering products. For this, they chose Gears from BigLever Software, currently in use to power other large-scale industrial product line engineering efforts [8][10][4] and three SPLC Hall of Fame members [17].

4. MEANWHILE, PLE EVOLVES

While the stage was being set at GM for the unfolding of a massive product line story, the field of product line engineering was not standing still. Indeed, it was evolving a new set of concepts and technology that has been referred to as *second generation product line engineering (2GPLE)* [9]. This characterization represents seen-in-practice extensions to an early paradigm centered mainly on core asset production and product derivation.

Although generational definitions based on industry trends are imprecise, 2GPLE can be said to comprise five aspects. None of these facets of 2GPLE are incompatible with or contradict earlier approaches to software product line engineering [13][20][5] – indeed, all five are mentioned as possible. The difference is that in 2GPLE they have emerged in a central role, essential to support large-scale practice. The five facets of 2GPLE will be discussed in turn.

4.1 Features as the *lingua franca* to express product differences across the lifecycle

The concept of "feature" allows a consistent abstraction to be employed when making choices from vehicle configuration all the way down to the deployment of software components onto an electronics architecture. GM is elevating what they call a *bill-of-features* to the role of communication vehicle between business, product marketing, and engineering units. The goal is to use this to express and automatically derive content for vehicles in terms of desired features and capabilities, rather than describing vehicles in terms of its bill-of-materials – that is, its listing of parts and pieces. Although a bill-of-materials will still be needed for manufacturing, the vision of GM's PLE effort is that the bill-of-materials for a vehicle's electronics is generated from its bill-of-features.

To capture features, here is the set of feature modeling constructs (provided by Gears) that GM is using for its product line work. They are:

- **Feature declarations** are parameters that express the diversity in the product line for a system or subsystem. Feature declarations typically express the customer-visible diversity among the products in a product line.

  Feature declarations have types. When a feature is chosen for inclusion in a product, it must be given a value consistent with its type. Table 1 shows the feature types supported by Gears.

Table 1 Gears feature types

Boolean	true, false	Enum-eration	**Select exactly one** value from sub-ordinate features.
Integer, Float	Signed or unsigned numeric value	Set	**Select zero or more** values from subordinate features.
String	Character string delimited by double quotes	Record	**Select all** values from subordinate features.
Character	Single character delimited by single quotes	Atom	Named member/ value of a set or enumeration.

- **Feature assertions** describe constraints and dependencies among the feature declarations. Feature assertions in Gears express REQUIRES or EXCLUDES relations. They express the constraint that a feature (or combination of features), if present, either requires or excludes the presence of another feature (or combination of features). For example, an assertion could express the need for software-actuated brakes to be present whenever the park assist option is on the vehicle, or the need for certain switches to be present if certain lights are installed.
- **Feature profiles** are used to select and assign values to the feature declaration parameters for the purpose of instantiating a product. A feature profile is associated with a product, and reflects the actual choices you make: Two door with sport package but no moon roof; or four door with luxury package and moon roof. The values assigned in feature profiles must satisfy the constraints and dependencies expressed by the assertions in the feature declarations.
- **Assets** are the abstraction for systems and software artifacts in a production line. They are the building blocks of the products in the product line. Assets may

be requirements, architecture and design documents, source code files, calibration sets, test cases, and so forth – artifacts from any phase of the development life cycle.

- **Variation points** encapsulate the variations in the assets used to build products. Feature declarations are mapped to these variation points, and a feature profile is mapped to the choices made at each variation point when building a product. In Gears, a variation point is instantiated by one or more variants, one of which will "stand in" for the variation point when a feature profile is used to build a product. A variant can "stand in" as is (in which case, the variation is accomplished by choosing which variant to use), or it can "stand in" after being transformed by applying a match-substitution pattern expressed in the regular-expression language of Perl.

Figure 2 illustrates how this small set of constructs – a feature model composed of feature declarations, feature assertions, and feature profiles, plus assets and variation points – give us the concept of a *production line* (the part of the figure inside the red box). Assets are built and maintained on the left; each is endowed with one or more variation points (indicated by the gear symbol). Feature profiles determine how the assets are instantiated (by exercising their variation points) to produce product-ready artifacts. Under this paradigm, organizations become production-centric rather than product-centric.

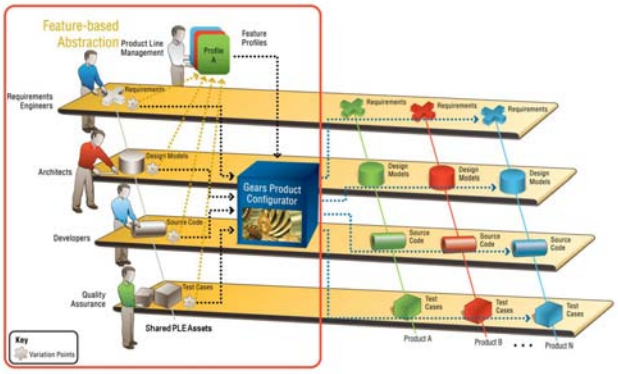

Figure 2 A production line. Feature profiles drive instantiation of assets' variation points, which are exercised by the configurator to produce product-ready instances. (© BigLever Software, Inc.)

4.2 Consistent variation management in artifacts across the full engineering lifecycle

It has long been a stated tenet of product line practice that core assets include more than software. For example, the Software Engineering Institute's Framework for Product Line Practice [14] states that "architecture, requirements specifications, testing-related artifacts, budgets, schedules, plans, and production infrastructure can all constitute core assets." However, a complete systems and software PLE lifecycle solution requires more than just a statement of eligibility. It requires *consistent treatment of the artifacts' variation points* under the production infrastructure, so that a full set of demonstrably consistent supporting artifacts can be systematically generated for each product. The alternative, trying to translate between the different representations and characterizations of features and variations across the boundaries between stages in the lifecycle, is untenable in large-scale practice.

The artifacts at GM to support this process include requirements, system architectures and designs, source code implementation, calibration parameters, test cases, and documentation. Some of the documentation is intended for suppliers, who will provide some of the necessary software and hardware components. GM's long-term goal is that all of these are endowed with variation points, which can be exercised to correspond to feature choices.

Common representation of variation points is key to achieving traceability from requirements to deployment. Traceability is of great concern for GM. Every requirement needs to be traceable to one or more design elements that satisfy that requirements, and each design element should be traceable back to one or more requirements that it satisfies. Each design element needs to be traceable forward to its implementation and vice versa. Each requirement needs to be traceable to one or more test cases that validate whether or not the requirement is satisfied in the final product. Managing all of these artifacts consistently, by tying their variations to features, is the key to achieving this.

4.3 CM that maintains assets, not products or asset instantiations

The most important aspect of CM in 2GPLE is that the full superset of available PLE assets (and not the individual products or systems) are managed under CM. A new version of a product is not derived from a previous version of the same product, but from the shared superset of PLE assets themselves.

Contrast this to product-centric CM, illustrated in Figure 3. Suppose a defect is discovered in Product B after it's been deployed, and the defect is traced back to product B's requirements. The Product B team fixes the defect and re-deploys. But Product B's requirements might have been borrowed from Product A's requirements, and Product N's code might have been borrowed from (defective) product B's. By the time all of the potential dependencies have been run to ground to make sure the defect is eliminated from every place it might occur in n products, $n(n\text{-}1)$ interactions have occurred, for an $O(n^2)$ complexity.

By contrast, using the scheme shown in Figure 2, the requirements defect will be fixed in the asset, not the products. The affected products will be re-generated. This is an $O(n)$ proposition.

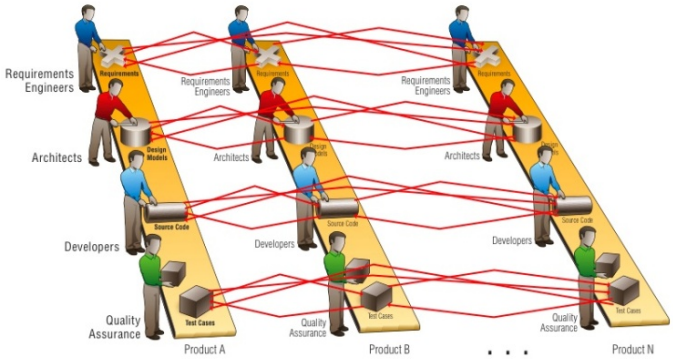

Figure 3 A product-centric perspective with $O(n^2)$ complexity (© BigLever Software, Inc.)

The configurator in the heart of the production line enables this simplified CM scheme and complexity reduction, because it makes it practical to re-generate any number of end products affected by a change in a shared asset.

4.4 Product lines across organizational boundaries

For PLE to work at large organizations, it may be impractical to have a single organizational unit tasked with the care and feeding of the shared PLE assets [18]. Certainly having one global collection of feature declarations for an entire production line is impractical. Large feature sets, as we have seen, engender intractable and incomprehensible combinatorics. Subsystem engineers have no interest or need to see all of the feature diversity in other subsystems. For example, engineers for an automotive transmission system do not need to see feature abstractions that capture the diversity in the entertainment or GPS navigation system. It makes no sense to comingle them.

It makes much more sense to modularize the feature model in a way that corresponds to the organizational structure of the enterprise. Although these structures can change over time [6], they make an excellent starting point and let the organization begin to adopt PLE using familiar units.

At GM, a vehicle is composed from a set of integration areas (such as *safety* or *human-vehicle integration*), which assemble combinations of subsystems, which are in turn composed of functional elements, which are implemented by compositions of software components and calibrations that are loaded onto hardware components arranged in one or more physical architecture topologies. At each level in this decomposition – which is not necessarily hierarchical – engineers are assigned responsibility for managing the artifacts and configurations at that level, all of which are imbued with rich and numerous kinds of variation. Assembling a vehicle from the most primitive elements would simply be intractable. By contrast, a vehicle is more like a system of systems [11], which is managed as a product-line-of-product-lines. At GM the nesting is at least four levels deep.

Each of these units represents a domain, by which we mean a body of knowledge [7]. Integration areas and subsystems are part of the fabric of the company. Building a subsystem for a vehicle, or combining subsystems in an integration area, or implementing a functional element requires specialized knowledge. In a PLE context, that specialized knowledge becomes knowledge about the variations that are possible, and the result is a number of product lines that each contribute instances to the overall vehicle product line.

In addition to the constructs outlined in Section 4.1, there are three more constructs to facilitate the interfacing and coordination between levels in the hierarchical product line: mixins, matrices, and imported production lines.

1. **Mixins**. Although feature declarations may fall cleanly into the realm of one asset or another, there are many cases where a feature declaration applies to two or more assets. For example, the automotive platform (Buick Regal? Chevy Cruze? Cadillac CTS?) and the region for which the vehicle is being marketed (North America? Brazil? China?) constitute features that determine how an asset should be configured at many levels: Integration area, subsystem, functional element, component. Rather than duplicating the same feature declaration in multiple assets, a mixin allows creation of a feature declaration in one place to "mix it into" the feature declarations of multiple assets, by reference.

   Mixins are more than a convenience to avoid duplicative feature declarations. They also encapsulate, in a single location, the feature profiles built from the feature declarations. Having a single location for the feature profiles prevents inconsistencies when composing assets to create a complete system.

2. **Matrices.** A production line is the "virtual factory" that knows how to build products by configuring assets in accordance with selected feature profiles. To build a product, you need to tell the configurator what feature profile to use for each asset and each mixin in the production line. A matrix is a table showing the choices to build a complete and consistent product. Each row specifies one product. Each column specifies a choice of feature profile for a mixin or an asset.

   A complete product instance is "actuated" by actuating every asset and nested production line column that has an entry for that product. Each asset and nested production line is actuated according to its cell value in the row. If an asset imports a mixin, the mixin feature profile to be used is determined by its cell value in that row.

   Some products may not need all of the assets. For example, low-end products in a production line may not include "luxury" assets that are aimed at high-end products. Each matrix allows you to include or exclude individual mixins and assets to accommodate such product diversity.

AutoCommerce	Global	Showroom	Site	Showroom	Requirements
ChevroletOfUS	Minimalist_ChevUS	BasicNew	Simple	$defined$	$defined$
ChevroletOfCanada	Minimalist_ChevCan	BasicNewAndUsed	Desktop	$defined$	$defined$
CadillacOfFrance	Extreme_CadillacFr	FullService	Mobile	$defined$	$defined$

Figure 4 A Gears matrix, with three rows for three products. The yellow columns show feature profile choices for mixins; the blue columns show feature profile choices for assets.

3. **Imported production lines.** Gears allows you to create a hierarchy of production lines by nesting one production line into another production line. In order to use a production line as a nested production line, it must first be imported. An imported production line will be added as a column in the matrices for the importing production line, just like an asset or mixin. For example, engineers at GM have defined a production line for the Safety integration area. In order to provide a Safety package to a vehicle, the Safety production line must include specifically configured subsystems from a number of subsystems (such as Body and Active Safety), which are their own production lines. A subsystem production line, in turn, can import production lines corresponding to functional elements, and so forth.

4.5 Industrial-strength automation

The last ingredient in 2GPLE is a configurator employed to maintain configurations, and translate feature profiles into assets with their variation points exercised in prescribed ways. The tooling needs to be able to support the construction and management of feature models (including feature declarations, assertions, and profiles), assets and their variation points, support hierarchical production lines, and map from feature choices to asset instances (this is the job of the matrices). Further, it needs to either provide version control for the models and artifacts or (even better) work seamlessly on top of the user's own choice of change management system.

A major requirement for the tooling is that it supports the specification and selection of variation in assets and artifacts from across the entire spectrum of the product lifecycle. This means that the tool will have to support variation in, for example, DOORS requirements modules, Microsoft Word documents and

Excel spreadsheets, build files for Make or Ant, Rhapsody UML models, and many more.

There are fundamentally three ways to achieve variation in an asset, depending on what you know about the digital representation of the associated artifact:

- The representation of the artifact is proprietary and closed, or editors for it are not available or are impractical. For example, if our products include a picture that is different from product to product, some of our artifacts may be GIF or JPG files. To achieve variation, the variation point can simply choose from a selection of variants, using each one as is, as opposed to trying to change the picture by editing the image stored in a one-size-fits-all picture file.
- The representation of the artifact is "open," so that you can change it using an available open-market tool. For example, artifacts stored as simple text files may be transformed by simple word or line substitution. Artifacts that are Microsoft Word documents stored in Office Open XML format can be transformed by third-party tools. In this case, the variation point operates by transforming a single variant by transforming it appropriately for each product being built.
- The representation of the artifact is proprietary, but the owning organization offers a business relationship to allow your tool to edit their artifacts. Suppose your requirements are stored in DOORS, using hundreds and hundreds of DOORS requirements objects. The representation of those objects is proprietary, but using the strategy in the first bullet is out of the question: Swapping in and out whole requirements documents or databases that each differ by just a little bit is untenable. It would be much better to write a piece of software that can insert variation points throughout the DOORS representation of a body of requirements. That requires an arrangement with the vendor (IBM Rational in this case) to open up their representation.

Figure 5 shows how Gears supports various lifecycle artifacts maintained under the proprietary auspices of various tools. In that figure, a *bridge* is a piece of software that "knows" the other-tool representation, and presents a "product-line-aware" user interface for that tool that allows product line engineers to insert variation points in the artifacts maintained by that tool.

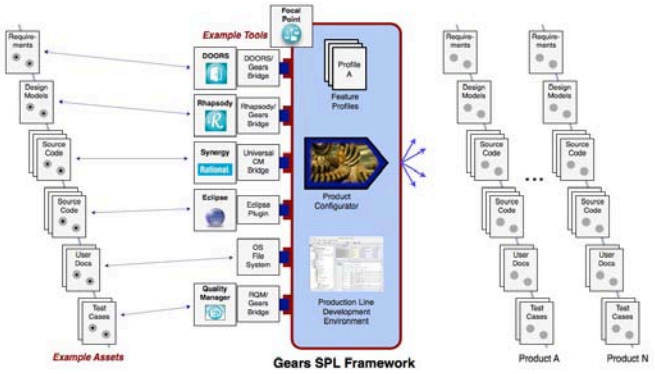

Figure 5 Gears and its bridges to other lifecycle tools (© BigLever Software, Inc.)

First-generation approaches always discussed the need for automation; second-generation approaches require it. Further, they don't just require technical proficiency from the tool but interface relationships to lifecycle tools and their providers.

5. GM'S APPROACH FOR MEGA-SCALE PLE

This section describes in greater detail how GM has adopted 2GPLE as their technical roadmap for the future.

5.1 GM's architectural decomposition

GM's architectural strategy plays a key role in how it is rolling out PLE. The strategy is one of logical decomposition as a way to gain control over the complexity of building a vehicle's electronics, and a way to allot the thousands of engineers into organizational units with clearly scoped roles and responsibilities.

- **Functional architecture**: First, a vehicle consists of a number of *domains*. These are "containers" for capturing the requirements necessary to describe the electronics terms applicable to an entire vehicle. Domains define areas of related functionality. For example, Powertrain is a domain, as is HVAC (heating, ventilation, and air conditioning).

  Orthogonal to domains are *integration areas*. Integration areas can be thought of knowledge areas for satisfying high-level stakeholder requirements for vehicles. Requirements here span domains. For example, Noise and Vibration is an integration area; it "touches" any domain that can contribute noise or vibration to the occupants' driving experience: Powertrain, Body, Chassis, HVAC, and more.

  GM refers to integration areas and domains together as its *functional architecture*. The functional architecture provides the over-arching structure to host the hierarchical PLE models. Each domain or integration area team will build the PLE models for their area of concern in corresponding part of the functional architecture hierarchy. Figure 6 illustrates.

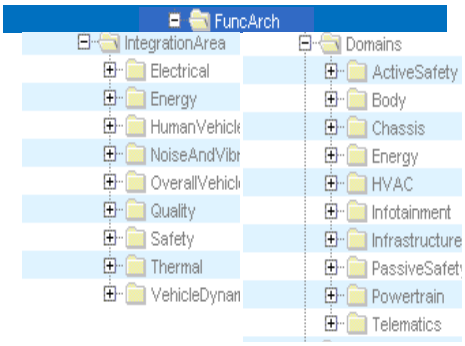

Figure 6 Tool view of GM's Functional Architecture, showing some of the integration areas and domains

- **Implementation architecture**. Domains comprise *subsystems*. Subsystems represent physical systems on vehicles. There are subsystems for brakes, external lighting, internal lighting, entry and egress, and many more. Subsystems have their own requirements, which must permit the subsystems to play their proper role in the domains and (in turn) integration areas that need them. Subsystem designers in turn decompose their subsystems into *functions*, and functions into *functional elements*, and write requirements for each. *Components* are units of implementation that satisfy the requirements for functions and functional elements. Components are arranged in a decomposition hierarchy; leaf nodes are components; higher nodes (which are just aggregations of their descendants in the tree) are called *compositions*. Components may be software components or hardware components, depending on how the functional elements are satisfied. GM calls this component

structure (with components mapped to the functional elements they satisfy) its *implementation architecture*.

- **Deployment architecture**. Next, the components have to be assigned a place in the onboard electronic architecture topology. Software components need to be assigned to an electronic control unit (ECU), and hardware components have to be assigned a spot in the topology. The selection of a topology from a small number available, the assignment of ECUs to spots in the topology, and the assignment of software to ECUs all constitute what GM calls its *deployment architecture.*
- **Vehicle application architecture.** Finally, the components need to be laid out on a vehicle. This architecture determines where the ECUs are stationed, and the type, position, and routing of the wire harnesses to connect the sensors, actuators, and ECUs.

These architectures – functional, implementation, deployment, and vehicle application – institutionalize and add structure to concepts that are deeply ingrained in the organizational and technical fabric at GM. For instance, there are centers of deep expertise in brakes and lighting and keyed/keyless entry systems and dozens of other domains. As part of PLE adoption, these centers are not going to be discarded in favor of a massive re-organization involving the re-orientation, re-assignment, and re-training of some 4000-5000 engineers.

5.2 Roles

GM's embrace of 2GPLE has led to the creation of a few new and refined engineering roles that have come about as a direct result of piloting their hierarchical product line. Figure 7 sketches the major PLE roles and their broad responsibilities vis-a-vis maintaining the PLE models and artifacts. This section will highlight a few of those.

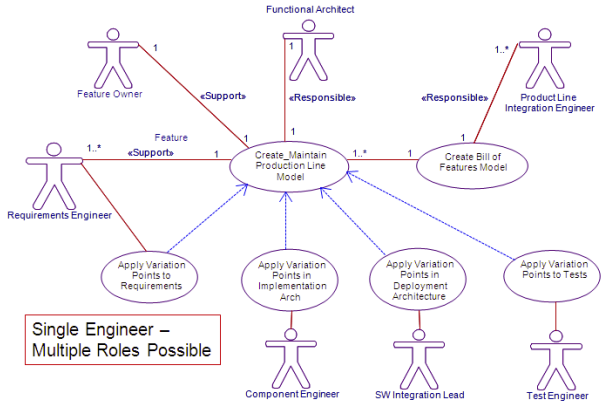

Figure 7 Product line engineering roles at GM

5.2.1 FEATURE OWNER

Feature owners take ownership of GM features (customer-visible features such as cruise control or lane keep assist or hundreds of others that are visible and bring value to car owners).

These features are, in a sense, abstractions. They only become tangible when realized by concrete artifacts: requirements, functions, software components, electronic control units, and wiring. In GM's PLE environment, each of those artifacts will also embody variation. It is the feature owner's job to make sure that all of those artifacts in "supporting roles" are adequate and correctly provide the feature to GM's customers.

A feature owner is the main technical contact to external teams who need to know about the feature from the point of view of assembling a vehicle from this and other features. This engineer is the recognized expert for the functional area regarding the feature's required variants, the system constraints it imposes, and how it integrates onto a vehicle platform.

A feature owner is also responsible for modeling the feature and its variations in Gears.

5.2.2 FUNCTIONAL ARCHITECT

This engineer owns a specific area of the functional architecture and as such establishes ownership and boundaries between system level assets. Together, the functional architects maintain the functional architecture taxonomy introduced above.

With the advent of the NGT PLE effort, functional architects have taken on a new and critical role: Together, they are the keepers and centralized owners of all of the PLE models. Their job is to ensure that the PLE models produced inside their assigned area by feature owners, asset owners, and others are consistent, fit together, and represent best PLE practice.

Functional architects are each assigned a domain, which will involve several subsystems. As PLE practices are introduced into each domain, functional architects might actually build the PLE models, working with feature owners. Under this scheme, the feature owners remain the subject matter experts about their features; the functional architects translate (or help the feature owners translate) that knowledge into well-structured and consistent PLE models.

5.2.3 PRODUCT LINE INTEGRATION ENGINEER

This is another new role at GM, brought about by PLE. This engineer collaborates with Vehicle Product Teams in the selection of a 'bill-of-features' for a vehicle being planned. The product line integration engineer also collaborates with the feature owners in the identification of the top-level subsystem production line 'products' that will be offered up to vehicles. The vehicle team for a vehicle will need the services and advice of a team of product line integration engineers, who together will put together the bill-of-features for that vehicle's electronics system. When the bill-of-features for a vehicle is created, the product line integration engineers will be at the table.

For example, the vehicle team for the Buick Verano wants to understand what kind of climate control options they can offer with the car (or, to put it another way, what climate control features are eligible for the Verano's bill-of-features, and what the downstream implications are of each). The product line integration engineer responsible for heating, ventilation, and air conditioning (HVAC) systems will offer up various automatic and manual climate control systems. If a vehicle might one day be powered by hybrid or next-generation energy and propulsion systems, this might mandate another kind of HVAC system.

The vehicle teams aren't interested in the details of the features' implementations, but only in how the features will appear to the customer and how they interact with each other. The product line integration engineers, then, manage subsystem "products" that are exposed at the vehicle and bill-of-features level.

5.2.4 ASSET OWNER

The remaining roles in Figure 7 are asset owners. These engineers manage various kinds of assets across their life cycle, and establish variation points in the assets.

A requirements engineer is one kind of asset owner. Their responsibilities include migrating requirements from legacy requirements assets (mostly Word documents) into DOORS and,

along the way, imbuing those requirements with variation points that support features.

Asset owners, including requirements engineers, are responsible for modeling the features that their assets make available to the consumers of those assets, and the variations in those features. These features are often strongly suggested, if not identified outright, in existing technical specifications. Thus, feature creation is more of an identification and extraction process, as opposed to an invention process. This helps make things go more smoothly and predictably.

5.3 Organizational adoption

Launching and institutionalizing [16] this approach at GM has required significant investment over the last two years or more, and that investment is ongoing. There has been a group of champions and advocates of the PLE approach throughout the effort. Early on, they sponsored a two-week workshop to show how the approach (using Gears in concert with DOORS) could tame the requirements for a major subsystem, with variations clearly identified and managed. This pilot effort produced more strong advocates, and steered GM towards their current tooling approach.

After that followed a steady series of workshops and technical meetings with senior engineers to work out how to apply the concepts at GM; the eventual results of these meetings include the architectures and roles described above, plus a vision of how features could be used across all of the architectures to describe variation. All the while, the champions practiced internal evangelizing, advocating the approach to management and engineers alike. One-day requirements workshops were held with subsystem owners to duplicate the results of the first two-week workshop.

The latter part of 2011 saw the launch of a series of some two dozen *Bill-of-Features Workshops*. These workshops bring together a small group of feature owners and subsystem experts in a particular area – for example, interior lighting. They spend a day learning the PLE approach and then actually using Gears to model the features in their domain. An important goal is to have participants experience the "PLE epiphany," when they see how 2GPLE and the NGT tool suite will help them do their jobs better.

At the start of 2012, after two years of establishing buy-in, GM launched a series of training courses. The course series kicks off with a short introduction to PLE at GM, and continues with one-day classroom courses in each of the tools and how they will be used. In concert with the training are the establishment of resources to help engineers once they go back to their desks: Discussion boards, FAQ lists, help desks, and the like.

In a PLE undertaking of this magnitude and complexity, it is unreasonable to expect that the engineering will be formulaic and without incident. Questions and issues are being captured and their answers stored in a "GM PLE Cookbook," which will include a set of patterns and anti-patterns for good practice, a list of FAQs (including those above), and a set of naming conventions for product line objects shared across organizations. This will represents a trove of practical knowledge not usually divulged in the product line literature, as well as another aid to institutionalization at GM.

5.4 What is the end game?

One of GM's senior electronics engineers characterizes the electronics division's job this way: "We build silver boxes," he said, "load software in them, and wire them together." If they can do that correctly for every vehicle they build, their job is done.

Whatever PLE and NGT can do to help achieve that purpose is a win for GM. The long-term vision is to create a bill-of-features for a vehicle (which manifests as a vehicle-level feature profile in Gears) and automatically derive as much as feasible of the bill-of-materials for that vehicle, including requirements, designs, models, software, calibration data, tests, documentation, allocation of software to hardware, wiring diagrams, and so forth. That vision is years away from being achieved.

However, short of that, there are some intermediate steps that GM is working towards. Examples include

- migrating all requirements to incorporate Gears variation points that formalize feature-based variations in the system, subsystem, and component requirements,
- generating calibration files and values that will need to be loaded onto the electronics modules, or
- given a set of features on a car and the components that need to be onboard to support those features, generating a list of all of the digital signals that the serial networks will have to accommodate.

Longer-term goals include calculating certain additive non-functional properties of the electronics, such as weight or generated heat or cost.

Even short of this capability, GM is already getting value out of their PLE efforts even before they have started producing instantiated engineering artifacts. Just defining an internally consistent vehicle with consistent versions of subsystems, functional elements, components, and hardware allocations represents a very big step in managing the complexity at hand. To be able to do this in an end-to-end fashion under the auspices of fully interoperating tool suite is a capability not available at GM before now. The automation – in this case, Gears – can do a semantic check on the feature model and report anomalies, such as the fact that this vehicle is supposed to support the lane-keep-assist feature but the instrument cluster chosen for it doesn't have the correct display for that feature, or the chosen physical architecture topology cannot support the serial data communication required.

6. EXAMPLE: DAYTIME RUNNING LIGHTS

We conclude by illustrating some of the points in this paper through an example, which necessarily must be a small one. A *daytime running light* (DRL) is a "lighting device on the front of a roadgoing motor vehicle, installed in pairs, automatically switched on when the vehicle is moving forward, emitting white, yellow, or amber light to increase the conspicuity of the vehicle during daylight conditions" [21].

6.1 DRL Requirements

DRLs are considered a feature at GM; they're certainly visible to the user. But not all cars have them. DRLs are required equipment in Canada, Norway, and Sweden, prohibited in Japan and China and optional in the United States, Europe, Australia, and the rest of the world. (Region of sale turns out to be a major discriminator among features, permitting or precluding a plethora of other features.)

On vehicles that have them, DRLs can be "implemented" by lamps dedicated to that purpose, or by front turn signal lamps, reduced intensity low beam headlamps, full intensity low beam

headlamps, parking lamps, or a combination of parking lamps and dedicated lamps.

Just as there are many ways to effect DRLs, there are many choices for how to turn them on and off (including none at all, leaving it up to the car to do so automatically). There is a thicket of requirements concerning when DRLs may, must, and may not be on. For example, in Europe DRLs must switch off automatically when the front fog lamps (if the car has front fog lamps) or headlamps are switched on, except when the headlamps are "used to give intermittent luminous warnings at short intervals" – that is, flashed.

These and other impinging factors consume page after page in the requirements document for the exterior lighting subsystem, of which DRLs are a member. These requirements are rife with information about what requirements apply under what conditions, and be used to identify variations in the DRL feature.

Besides being a feature by themselves, DRLs play a part in other features as well. Some realizations of the "Lead me to my car" feature flash the DRLs when the key fob is pressed. Police vehicles have turn-everything-on features, which include dedicated DRLs if the car is so equipped. Cornering lamps, another feature, can only come on under certain conditions, and affect DRLs if they share output devices.

6.2 Modeling DRLs

The feature owner for DRLs is responsible for understanding how the DRL feature is realized, the variations it includes, and any variant capabilities required because of its appearance in other features.

Figure 8 shows a preliminary feature model for DRLs. The feature model captures the output type (what lamps on the vehicle can be used), if and how DRLs can be disabled, and how DRLs are integrated with the car's headlamp controls turn signals, respectively.

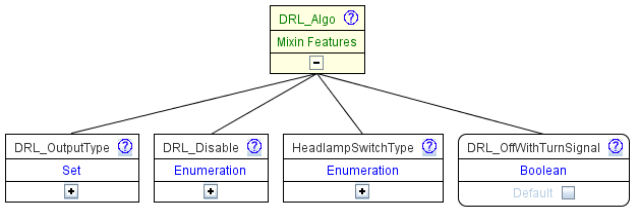

Figure 8 Feature model for daytime running lights

Figure 9 shows how the output type sub-feature is expanded to take into account all the possibilities. The output type is modeled as a set; a vehicle can have any number of ways of realizing the DRL feature, or none at all.

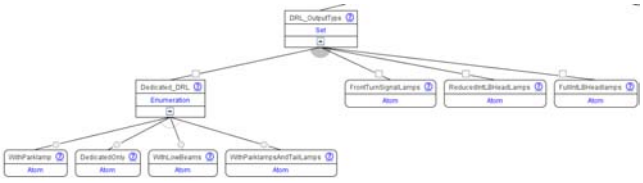

Figure 9 Expansion of the OutputType feature of DRLs

However, if a vehicle realizes the DRL feature with low-intensity headlamps, then it cannot realize it with high-intensity headlamps as well, and vice versa. An assertion captures this:

NOT DRL_Algo.DRL_OutputType >= {ReducedIntLBHeadLamps, FullIntLBHeadLamps}

This says that the OutputType set cannot include both of the indicated values; in Gears, the symbol ">=" when used in an expression involving sets means "is a superset of."

The DRL feature owner builds these feature models in Gears, under the conventions and standards developed by the functional architects, and in particular the functional architect for the exterior lighting domain. He or she will also build a matrix for the DRL feature model that specifies a small number of flavors of the DRL feature that can be made available to PLIE's working to assemble vehicles from features.

Simultaneously, requirements engineers who own the requirements for this feature work to turn the DRL requirements into a Gears asset, with variation points that (when exercised) produce requirements that correspond to the requirements needed for each case.

6.3 DRLs and deployment

Those responsible for choosing a deployment architecture are another kind of asset owner; their asset is the set of ECUs needed for the features on board, and the variation points they can provide based on features and feature combinations chosen. These variation points include basic network topology (currently two are available; more may be added), how many ECUs will populate a topology, what the choice of ECU hardware will be, and the allocation of software components to each ECU.

6.4 DRLs and other features

Feature owners for other features that interact with DRLs (such as the lead-me-to-my-car feature owner) will need to reference DRLs in their feature models and profiles. They will coordinate with the DRL feature owner, under the auspices of the functional architects for the including domain or domains, to make sure that the DRL feature can be referenced, by importing the DRL domain-level mixin into their domain production line.

Other domains than exterior lighting will need the same ways to refer to DRL in their feature models. For instance, the switches that turn DRLs on and off are part of the Body domain, whereas any indication that DRLs are on are part of the Displays domain.

6.5 DRLs and the vehicle

Finally, those defining a vehicle type and the myriad of features combinations that GM wishes to offer with it, can do so by importing all of the domains' and integration areas' production lines, adding the highest layer to the product line hierarchy. They will also define a matrix of "products" for each vehicle, defining combinations of features in concert with each other.

7. CONCLUSIONS

The guiding PLE vision at GM is the ability to engineer vehicles – across the full lifecycle – according to a "bill-of-features" rather than the traditional "bill-of-materials." Although still very much a work in progress, the GM experience has already revealed a number of lessons about mega-scale product line engineering.

First, the product line experience at General Motors can be seen as intensively applying aspects of what has been called Second Generation Product Line Engineering. This new perspective brings the following ideas, which previous approaches always allowed but never stressed, to the forefront:

- A focus on features as the "lingua franca" of variation and product selection; the "bill-of-features" replaces the "bill-of-materials" as the key engineering artifact for product derivation.

- Treatment of artifacts across the entire lifecycle completely consistently with each other, and consistently with the software, as first-class components of the product line and the derived products
- An emphasis on high-quality automation at the center of a production line, to quickly turn a bill-of-features into a set of instantiated lifecycle assets
- A CM and PLE approach geared to multi-baseline multi-product management in a way to reduce the order of complexity from $O(n^2)$ to $O(n)$.
- Taking multi-organizational management in stride, by providing feature model concepts such as mixins and imported (hierarchical) production lines, to reflect the structure of engineering activities and domain knowledge present in an ultra-large organization.

Second, the GM experience also validates a small set of consistent concepts is sufficient to model product lines of inordinate complexity. Features (declarations, types, assertions, and profiles), assets (that embody features, as well as variation points), mixins, and matrices constitute a production line, the "factory" that turns feature choices into asset instances. Allowing production lines to import other production lines gives us unlimited hierarchy, which can map to any organizational structure in which specialized bodies of knowledge are encapsulated throughout, no matter how many levels deep.

Third, one of the most important aspects of PLE and GM is the application of consistent variation management in artifacts from all across the lifecycle (the second bullet above), artifacts that are currently maintained using other lifecycle tools. In order to accomplish this, the automation engine has to embody business partnerships with important tool vendors.

Future work involves continuing the march towards the ultimate "end game" described in Section 5.4: Generating a complete bill-of-materials for a vehicle by starting with its bill-of-features. Ultimately, GM may investigate merging PLE with PLM (product lifecycle management), which is the technology used in vehicle manufacturing. That would represent a convergence with repercussions across the entire manufacturing industry.

8. ACKNOWLEDGMENTS

Our thanks to everyone at General Motors who is involved in making 2GPLE a success, and especially those who have been involved through the roll-out effort. Their contributions have made this case study, and the story it tells, possible.

9. REFERENCES

[1] "G.M. Regains the Top Spot in Global Automaking," Business Day, *New York Times*, January 19, 2012.

[2] Automotive Open System Architecture (AUTOSAR), "Background," www.autosar.org.

[3] Bosch, J. "Organizing for Software Product Lines," 117-134. Proceedings of the 3rd International Workshop on Software Architectures for Product Families (IWSAPF-3). Las Palmas de Gran Canaria, Spain, March 15-17, 2000. Berlin, Germany: Springer, 2000.

[4] Cezo, J., Krueer, C., "Use product line engineering to reduce the total costs required to create, deploy & maintain systems & software," *EE Times*, December 10, 2008.

[5] Clements, P. & Northrop, L. Software Product Lines: Practices and Patterns. Boston, MA: Addison-Wesley, 2002.

[6] Clements, P., Brownsword, L, "A Case Study in Successful Product Line Development," Software Engineering Institute CMU/SEI-96-TR-016, September 1996.

[7] 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.

[8] Krueger, C., Churchett, D., Buhrdorf, R., "HomeAway's Transition to Software Product Line Practice: Engineering and Business Results in 60 Days," Proceedings, 12th International Software Product Line Conference, p 297-306.

[9] Lanman, J., Kemper, B., Rivera, J., Krueger, C., "Employing the Second Generation Software Product-line for Live Training Transformation," *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2011.*

[10] Lombardi, C., "IBM partners on 'smart' wind system," CNET, October 6, 2010, http://news.cnet.com/8301-11128_3-20018733-54.html

[11] Office of the Deputy Under Secretary of Defense for Acquisition and Technology. Systems and Software Engineering. *Systems Engineering Guide for Systems of Systems, Version 1.0*. Washing-ton, DC: ODUSD(A&T)SSE, 2008. http://www.acq.osd.mil/sse/docs/SE-Guide-for-SoS.pdf

[12] Paur, J. "Chevy Volt: King of (Software Cars)," *Wired*, November 5, 2010, http://www.wired.com/autopia/2010/11/chevy-volt-king-of-software-cars/

[13] Pohl, K., Böckle, G., van der Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 1998.

[14] Software Engineering Institute, "A Framework for Software Product Line Practice, version 5.0," http://www.sei.cmu.edu/productlines/frame_report/index.html

[15] Software Engineering Institute, "A Framework for Software Product Line Practice, version 5.0: Configuration Management," http://www.sei.cmu.edu/productlines/frame_report/config.man.htm

[16] Software Engineering Institute, "A Framework for Software Product Line Practice, version 5.0: Launching and Institutionalizing," http://www.sei.cmu.edu/productlines/frame_report/launch.inst.PL.htm

[17] SPLC Software Product Line Hall of Fame, http://splc.net/fame/gm.html

[18] Van der Linden, F., Schmid, K., Rommes, E. *Software Product Line in Action*, Chapter 5, Springer, 2007.

[19] Villanueva, J. C., "Atoms in the Universe," http://www.universetoday.com/36302/atoms-in-the-universe 2009. The number of atoms in the observable universe is approximately 10^{80} or 2^{260}.

[20] Weiss, D. M. & Lai, C. T. R. *Software Product-Line Engineering: A Family-Based Software Development Process*. Reading, MA: Addison-Wesley, 1999.

[21] Wikipedia, "Daytime running lamp," http://en.wikipedia.org/wiki/Daytime_running_lamp

[22] Williams, Cheryl. "Algorithms, Algorithm Modeling, Software and Software Architecture," slide presentation, available at http://www.eecs.umich.edu/courses/eecs486/win03/notes/GMVisit.pdf