Applying Product Line Engineering (PLE) to achieve architectural singularity and HW/SW platform independence of automotive IVI systems

Eran Gery
Lead Architect, Rational Systems Solutions
eran.gery@il.ibm.com

Kinichi Mitsui
Senior Manager, Rational Systems Development, Tokyo Software Development Laboratory
mitsui@jp.ibm.com
Agenda

- Why PLE matters
  - GM Case Study
- Overall Rational PLE Solution Architecture
- Three PLE Patterns
  - Multi-Branch Variant Management
  - Parametric PLE
  - Feature-based Variant Management
- Summary
Product Differentiation

*Vital to consumer buying decision*

- Differentiation helps create compelling offerings at different price-points
- A differentiated product range helps capture the targeted market spectrum
- Infotainment helps create much needed product differentiation
- Infotainment has biggest Influence on consumer buying decision
- Five of the top 10 features consumers look for when shopping for their next vehicle are related to infotainment

Differentiation is difficult to implement
Reuse complexities multiply over time

- Resources also have variants reflecting changes in other properties such as the market or geography for a car, or the intended platform for a piece of software
- Resources have revisions reflecting changes in time
- A configuration selects the right version and variant of the correct resource for a given offering
- There is no fundamental difference between these two types of versions – In fact it is an N-Dimensional space
Product Line Engineering

• PLE is a solution for engineering organizations that need to develop and deliver multiple co-existing product “variants”.

• The key to effectiveness here is facilitating strategic reuse as a key factor of the engineering process.

• We plan to address PLE by adding capabilities to our tools and articulating best practices – rather than creating a new “PLE tool”

“Product lines are a development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers. Software product line engineering can also enable rapid market entry and flexible response, and provide a capability for mass customization”

- SPL handbook by SEI
The Rational Systems & Software Engineering Platform

SSE Foundation
A core set of integrated capabilities primarily provided by
• DOORS & DOORS NG
• Rhapsody
• RQM
• RTC
• RELM

Industry and domain solutions
• Practices
• Templates
• RMC content
• Demonstrations
• Proofs of Technology
• and much more
SSE is based on an open standards integration platform

- “Linked Data” approach
- Integration at scale
- Avoid data replication and synchronization
- Enable collaboration of disparate teams over international networks

**OSLC is an open and scalable approach to lifecycle integration.**
*It simplifies key integration scenarios across heterogeneous tools*
Key PLE challenges we address…

- Maximizing reuse of engineering assets across product variants
  - Reuse of assets across variants
  - Avoid cloning and owning of assets
- Specifying a multi-domain product structure
  - Create multi-domain reusable components
- Consistently manage asset configurations across all lifecycle domains
  - Create cross domain baselines
  - Support end to end consistent traceability
- Effectively handling change propagation to multitude of variants
  - “where is this asset used”
  - “where does this change need to go”
- Effectively creating new product variants using generic architectures
- Transitioning from “clone and own” to reuse architecture
Reuse in the context of the SSE

- **The reuse is realized on how we can effectively manage the SSE work products**
  - Requirements, Specifications & Architectures (models), Test and V&V data, Source code
- **Artifact level reuse**
  - Artifacts (“resources”) can be requirements, test-cases, model fragments, source files etc
  - Assume system with 1000 artifacts and multiple variants
  - For example variants share about 80% of the artifacts, and 20% are different
  - A configuration management system would manage the sharing of the 800 artifacts, and the different versions of the other 200
    - Much more efficient than “clone and own”
  - The CM system can also tell us what is being shared and what is already “branched”
- **Parameterized artifacts**
  - We can use the same artifact for two variants, where the final content is determined by providing parameters
- **Component reuse: partitioning the system artifacts to components (collections)**
  - A modular reuse
  - More abstract than atomic artifact reuse
  - Components can have different configurations
  - Our approach strongly promotes component reuse
## Progressive delivery of PLE Patterns

<table>
<thead>
<tr>
<th>Scope</th>
<th>A. Multi-Stream, Multi-branch</th>
<th>B. Parametric</th>
<th>C. Feature-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product-focused</td>
<td>Platform-focused</td>
<td>Platform-focused</td>
<td></td>
</tr>
<tr>
<td>with relatively few product</td>
<td>with moderate number of</td>
<td>with very large number of</td>
<td></td>
</tr>
<tr>
<td>variants</td>
<td>product variants</td>
<td>product variants</td>
<td></td>
</tr>
<tr>
<td>Theme</td>
<td>Variants of products are</td>
<td>Product variants are</td>
<td>A product feature model</td>
</tr>
<tr>
<td></td>
<td>created by using traditional</td>
<td>derived automatically from the</td>
<td>provides the skeleton for</td>
</tr>
<tr>
<td></td>
<td>versioning techniques (e.g.</td>
<td>platform based on parameters.</td>
<td>variant management. Products</td>
</tr>
<tr>
<td></td>
<td>streams) or cloning artifacts</td>
<td></td>
<td>are assembled from features</td>
</tr>
<tr>
<td></td>
<td>for opportunistic reuse.</td>
<td></td>
<td>as required.</td>
</tr>
<tr>
<td>Key user capabilities needed</td>
<td>• Versioning sets of</td>
<td>• Variant artifacts</td>
<td>• Feature-driven management of</td>
</tr>
<tr>
<td></td>
<td>engineering artifacts</td>
<td>(parametric and manually-created)</td>
<td>variability</td>
</tr>
<tr>
<td></td>
<td>• Defining product / component</td>
<td>• Conditional inclusion of</td>
<td>• Employing business</td>
</tr>
<tr>
<td></td>
<td>variants</td>
<td>components in product</td>
<td>partners’ feature modeling</td>
</tr>
<tr>
<td></td>
<td>• Hierarchical baselining</td>
<td>definitions</td>
<td>tools</td>
</tr>
<tr>
<td></td>
<td>• Work in variant’s cross-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lifecycle context including</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traceability</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multi-branch Variant Management
Multi-branch variant organization

- Each product variant is essentially a *cross-domain* branch of evolving artifacts.
- A branch evolution is a sequence of baselines which record a particular state of the entire artifact set.
- New variants can be branched from existing variants at certain baselines and evolve in parallel.
- Branches can be isolated or can continue to be impacted by changes in parent branch(es).

![Diagram of multi-branch variant organization]

- **Initial product** → **Variant 1** → **Variant 2** → **Variant 3** → **Variant 4** → **Variant 5**
- **Function**
- **Artifact propagation**
- **Baseline**
- **Requirements**
- **Architecture**
- **Test**
Example: Requirement configurations DOORS NG

- Diff contents of two configurations
- Diff contents of module in two configurations
RELM PD – Product composition from components

- RELM/PD specifies the top level structure of product component variants
  - Represents **global configurations** of the system

- The PD structure represents the key functional blocks of a product and the associated domain components (e.g. requirements, design, code, tests)

- PD references configurations of domain components
  - Every variant utilizes specific configurations of the domain components

- PD also serves as a coarse grain configurator to domains lacking version management (e.g. DOORS 9).
Example: Creating the product structure in RELM-PD

```
Car-US
  Engine
  Gear
```

Diagram:
- RELM-PD
  - Car-PD
    - Car-US
      - Engine
      - Gear

- New/Modified Links
- Unmodified links
- version
- Shared version
Baselining the requirements in product context

RELM-PD

Car-PD

CarUS

Engine 1.0

Gear 1.0

Requirements (DNG)

GC: CarUS

R-Engine 1.0

R-Gear 1.0

New/Modified Links

Unmodified links

version

Shared version

GC – Global Configuration

Code (RTC)

Architecture (Rhp/DM)
Adding system architecture and traceability to requirements

RELM-PD

Car-PD
  CarUS
    Engine 1.0
    Gear 1.0

Requirements (DNG)

GC: CarUS
  R-Engine 1.0
  R-Gear 1.0

Architecture (Rhp/DM)

GC: CarUS
  Eng
  M-Engine 1.0
  M-Gear 1.0

Unmodified links
New/Modified Links

Code (RTC)

Version
Shared version
 GC – Global Configuration
Adding implementation

RELM-PD

Car-PD

CarUS

Engine 1.0

Gear 1.0

Requirements (DNG)

GC: CarUS

R-Engine

R-Gear

version

Shared version

GC – Global Configuration

Architecture (Rhp/DM)

GC: CarUS

Eng

Gear

M-Engine 1.0

M-Gear 1.0

Code (RTC)

Stream C-CarUS

C-Engine

C-Gear

New/Modified Links

Unmodified links
Creating a new variant – New requirement configurations

**RELM-PD**

- Car-PD
  - CarUS
    - Engine 1.0
    - Gear 1.0
  - CarEU
    - Engine 2.0
    - Gear 1.0
    - ABS 1.0

**Doors9**

- GC CarUS
  - M-Engine 1.0
  - R-Gear 1.0
- GC CarEU
  - R-Engine 2.0
  - R-ABS 1.0
  - R-Gear 1.0

**Architecture (Rhp/DM)**

- GC: CarUS
  - Eng
  - Gear
    - M-Engine 1.0
    - M-Gear 1.0

**Code: RTC**

- Stream C-CarUS
  - C-Engine
  - C-Gear
Creating a new variant – adding architecture and code impact
The platform architecture enables high reuse through variant management of configurable lifecycle domain components.
Parametric PLE
Parametric PLE

- Parametric PLE enhances multi-branch PLE by further automation
  - Addresses the complexity of handling the functional variant dimension
- Defining a “platform” with a set of reusable parametric artifacts
  - A parametric artifact includes elements that depend on certain parameters
  - Increases the reuse of individual resources for multiple products
- A derived artifact is created from a parametric artifact by applying concrete values to parameters
Parametric components and their reuse

- Two primary parameterization patterns:
  - Optional elements in a structure
  - Parametric properties
- Some common examples for parametric resources:
  - Configuration files driving pre-processor (#ifdefs) on source code
  - “Hiding” properties of requirements for a requirement variant
  - Modifying requirement property values for a specific product
Example: deriving a structural model with optional elements

A generic configuration (“120%”)
[Parameters: Trim= L, XL]

A derived configuration (100%)
[Trim= L]
Platform Based Parametric PLE

- Product line (platform)
- Variability parameters
- Parameter configuration 1
- Parameter configuration 2
- Parameter configuration 3
- Parameter configuration 4
- Variant 1
- Variant 2
- Variant 3
- Variant 4

- Baseline
- Parameterized baseline
- Artifact derivation

- Platform Artifacts
- Test
- Code

- Variant Artifacts
- Design

- Time

- = Baseline
- = Parameterized baseline
- = Artifact derivation
Feature-based Variant Management
Feature Driven PLE – Configuring products using feature abstraction

- Feature driven PLE introduces a “problem domain” abstraction of the product line as a collection of variable “features”, and a product as realizing a concrete set of features.
- The feature based pattern augments the parametric pattern by driving configurations based on features.
- Features selections are mapped to parameter selections.

<table>
<thead>
<tr>
<th>Platform</th>
<th>User/Problem Domain</th>
<th>Solution Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Models</td>
<td></td>
<td>Shared PLE Assets with feature-based</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variants</td>
</tr>
<tr>
<td>Product Feature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td></td>
<td>Configured assets for a Product</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instance</td>
</tr>
</tbody>
</table>
Variability (Feature) Modeling

- Feature models represent the “problem domain” abstraction of the product line
- Capture a functional view of the system components and their variabilities from a product line management standpoint
- Feature models have configurations called *feature profiles*, which drive the variability parameters of the solution
  - In our case they can be mapped to dimension and dimension values
- We plan to enable 3rd party feature modeling tools to integrate with the platform PLE services
Summary

- Rational PLE solution considers OSLC-based connected tools as a whole
- The layering of PLE solution is based on 3 patterns:
  - Multi branch
  - Parametric
  - Feature based
- Multi-branch pattern is a basis for both basic and advanced PLE
  - Handles cross-domain branch of evolving artifacts
  - RELM/Product Definition specifies the top level structure of product component variants
- Parametric enhances multi-branch PLE by further automation
- Feature based pattern allows problem-domain thinking rather solution domain thinking
  - Augments parametric pattern by driving configurations based on features
Questions
Acknowledgements and disclaimers

**Availability**: References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

The workshops, sessions and materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. While efforts were made to verify the completeness and accuracy of the information contained in this presentation, it is provided AS-IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. Nothing contained in this presentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

© Copyright IBM Corporation 2014. All rights reserved.

- U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

IBM, the IBM logo, ibm.com, Rational, the Rational logo, Telelogic, the Telelogic logo, Green Hat, the Green Hat logo, and other IBM products and services are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

If you have mentioned trademarks that are not from IBM, please update and add the following lines:

[Insert any special third-party trademark names/attributions here]

Other company, product, or service names may be trademarks or service marks of others.
Please note the following

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.
Thank You