



Systems and software product line engineering with SysML, UML and the IBM Rational Rhapsody BigLever Gears Bridge.

Integrating MDD and SPL to effectively manage product line diversity

Charles W. Krueger, Ph.D, BigLever Software

Martin Bakal, IBM Rational Software

Contents

2 Introduction

3 MDD and SPL

5 Background on SPLs and BigLever Software Gears

5 Product-centric thinking impedes portfolio production

7 Shift in perspective to an efficient means of production

9 Background on MDD and Rational Rhapsody

11 Rhapsody/Gears Bridge—integrating MDD and SPL

12 A handset product line example

19 Conclusions

Introduction

The key to business success depends on the infusion of new ideas for how products are brought to market. To achieve this goal, today's software-based product development organizations must deliver a product line—a portfolio of similar products with variations in features and functions—rather than just an individual product. A new approach referred to as software product lines (SPL), or more precisely systems and software product line engineering and delivery, has emerged to enable organizations to develop, deliver and evolve an entire product line portfolio, through each stage of the development lifecycle, with much higher degrees of efficiency than has been possible before.

IBM® Rational® and BigLever Software have joined forces to help provide an innovative and pragmatic new SPL solution, offering organizations the infrastructure, tools, best practices and methods needed to create the industry's most advanced and efficient means of production for their software-based product lines. Similar to what is seen in manufacturing, companies that invest in a more efficient means of production for SPL engineering and delivery can get the help they need to make a discontinuous jump in productivity, quality, time-to-market and product line scalability.

The IBM Rational and BigLever SPL Solution is comprised of the following key elements:

- *BigLever's Gears SPL Lifecycle Framework, for the SPL integration of existing or new tools, assets and processes across the full system and software development lifecycle.*
- *IBM Rational Toolset integrations into the Gears SPL lifecycle framework, extend the full systems and software development lifecycle capabilities for the delivery of software-based product lines. The integrations include the IBM Rational DOORS®, IBM Rational Rhapsody®, IBM Rational ClearCase, IBM Rational Synergy, Team Center and Quality Manager solutions.*

Highlights

SPL engineering helps provide the ability to efficiently and effectively create, maintain and evolve a portfolio of similar products with the simplicity of a single system rather than the complexity of deploying a multitude of products.

This report focuses on one of these integrations, the Rational Rhapsody/BigLever Software Gears Bridge. The Rational Rhapsody/Gears Bridge extends the abstraction, comprehension and communication benefits provided by model-driven development (MDD) with SPL capabilities to capitalize on product line commonality and variation. The Rhapsody/Gears Bridge provides SPL extensions for both SysML systems modeling and UML software modeling.

MDD and SPL

The motivation for integrating Rational Rhapsody and BigLever Gears via the Rational Rhapsody/Gears Bridge is to capitalize on a strong synergy that results by combining MDD and Software Product Line engineering. MDD is known for its ability to accelerate system and software development by leveraging the higher level of abstraction provided by the Unified Modeling Language (UML) and the Systems Modeling Language (SysML). These two modeling languages, UML and SysML, are the Object Management Group's (OMG) domain-specific languages for software development and embedded systems engineering. SPL engineering helps provide the ability to efficiently and effectively create, maintain and evolve a portfolio of similar products with the simplicity of a single system rather than the complexity of deploying a multitude of products. Combined, an entire product line can be expressed and engineered from a single, configurable SPL/MDD model.

With MDD, creating software for a portfolio of similar products has traditionally relied on one of two different approaches, clone-and-own or one-size-fits-all. In the clone-and-own approach, the model for every new product is created by making a copy—or clone—of the model for a similar existing product and then modify that model so that it implements the unique features and characteristics of the new product. There is 100% reuse at the time of the cloning, but similar to clone-and-own of conventional source code, the duplication can lead to divergence over time and require merging or replicated development among the different models, adding to the time and cost of maintenance and evolution.

Highlights

Some organizations adopt the one-size-fits-all approach to avoid the overhead of clone-and-own. With the one-size-fits-all approach, the product features and product diversity for an entire product line portfolio are implemented in a single model, using meta-logic and configuration files to allow decisions about which product feature alternatives to include in any particular product to be made at runtime. This eliminates the need for cloned models. Similar to one-size-fits-all in conventional source code, this approach can lead to models that continue to grow larger and more complex over time as more and more products and features are added to the portfolio.

With SPL engineering, specialized tools and methods are provided for efficiently creating, maintaining and evolving software assets for a product line portfolio of similar products. Early generation SPL tools and methods primarily focused on conventional source code, requirements and test cases. SPL support was not commercially available for variation points that captured product diversity in UML and SysML model elements in MDD. SPL support was not commercially available to allow MDD models to serve as reusable SPL core assets that could be automatically configured by an SPL product configurator.

In response to customer demand to combine the latest generation of SPL engineering technology with modern MDD tool support, BigLever Software and IBM Rational teamed up to create a bidirectional integration between the Rational Rhapsody MDD tool and the BigLever Gears SPL framework.

In response to customer demand to combine the latest generation of SPL engineering technology with modern MDD tool support, BigLever Software and IBM Rational teamed up to create a bidirectional integration between the Rational Rhapsody MDD tool and the BigLever Gears SPL framework. The result of this collaborative effort is the Rational Rhapsody/Gears Bridge, which helps to combine the strengths of the MDD and SPL approaches in one work area.

With the Rational Rhapsody/Gears Bridge, Rational Rhapsody models can now be first-class SPL core assets in a Gears software production line. These model core assets can be integrated with other types of SPL core assets across the full development lifecycle, including requirements, conventional source code, documentation, test cases and so forth.

Highlights

The characteristic that distinguishes the SPL approach from previous efforts is when an organization invests in a means of production that enables them to efficiently create a product line of similar software systems from a shared set of software assets.

With the Rhapsody/Gears Bridge solution, UML and SysML models explicitly show both the common and varying parts of the product line design. The design of each SPL variation point—including the options, alternatives and the logical specification that differentiates them—is directly visible in the model.

The Rhapsody/Gears Bridge solution provides the combined simplicity and benefits offered by MDD and SPL approaches—including significant gains in productivity, reduction in defect density and faster time-to-market with new products—as well as synergistic benefits that dramatically increase the scalability of a product line portfolio.

Background on SPLs and BigLever Software Gears

The characteristic that distinguishes the SPL approach from previous efforts is when an organization invests in a means of production that enables them to efficiently create a product line of similar software systems from a shared set of software assets. Manufacturers have long employed analogous engineering methods to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the product line.

Product-centric thinking impedes portfolio production

Throughout the first five decades of the software engineering field, the methods and tools of the trade have predominately promoted a product-centric perspective. The state of the industry today is a bevy of sophisticated product-centric development tools and processes that can be effectively applied to the software development lifecycle of an individual product, from early inception through design, implementation, testing, deployment and maintenance.

However, these product-centric tools do not independently or collectively offer an effective means to engineer and deliver a software-based product line. With product-centric tools, it is left as an exercise for the tool user to craft the homegrown techniques for managing the “commonalities and variabilities” among products during the development of their product line portfolios.

The repercussions of taking a product-centric perspective in a product line setting are shown in Figure 1. The vertical blue bars highlight the product-centric focus on the development lifecycle of the individual products (A, B, through N) in a product line. The red lines illustrate the complex, tangled and labor-intensive interactions, dependencies and coordination activities required to take advantage of what is common and manage all the variations among the similar products in the product line portfolio.

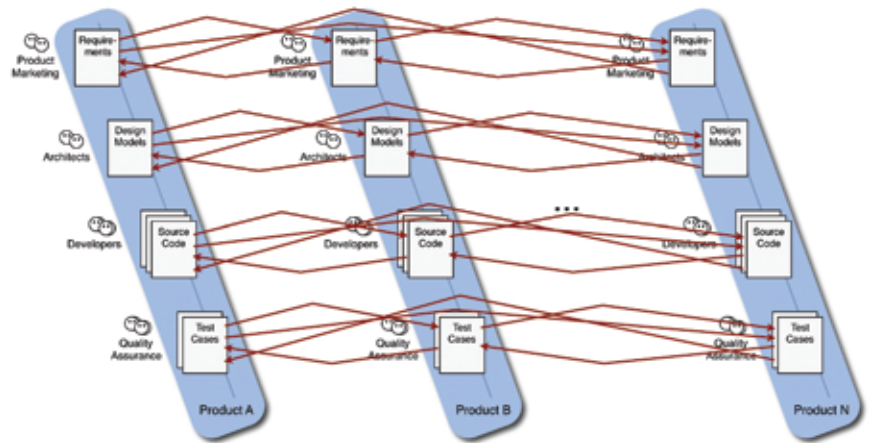


Figure 1: Complex interdependencies from the product-centric perspective

The crux of the problem lies in the fact that the number of red interdependency lines grows by the square of the number of products in the product line, explaining why complexity and effort increase exponentially faster than the growth of the product line. Making matters even worse, the conventional product-centric traceability relationships between the different stages of the lifecycle for an individual product interact with the red product interdependency relationships, multiplying the complexity and introducing dissonance across the stages of the lifecycle.

These tactical development challenges are so large that they impede a company's ability to achieve strategic business objectives, such as hitting market windows, offering competitive pricing while maximizing profitability,

Highlights

meeting product quality demands, and expanding the scale and scope of their portfolio. Comparing the ad hoc, complex and labor-intensive nature of the product-centric perspective to the sophisticated means of production found in semiconductor fabrication or in automotive manufacturing makes clear that there is an extraordinary need and opportunity for dramatic improvements in software-based product line engineering and delivery.

Shift in perspective to an efficient means of production

“We can’t solve problems by using the same thinking we used when we created them.” –Albert Einstein

Organizations mired in the complexity, inefficiency and pain of product line engineering from a product-centric perspective, experience an SPL epiphany when a shift in perspective reveals a simpler solution to the problem. Analogous to engineering a product line of hard goods, it is much more effective to take the perspective that views product line engineering as creating a means of production—a single system capable of automatically producing all of the products in a product line—rather than viewing product line engineering as creating a multitude of interrelated products. The powerful, though subtle, essence of the SPL epiphany is the focus on that singular means of production rather than a focus on the multitude of products.

The powerful, though subtle, essence of the SPL epiphany is the focus on that singular means of production rather than a focus on the multitude of products.

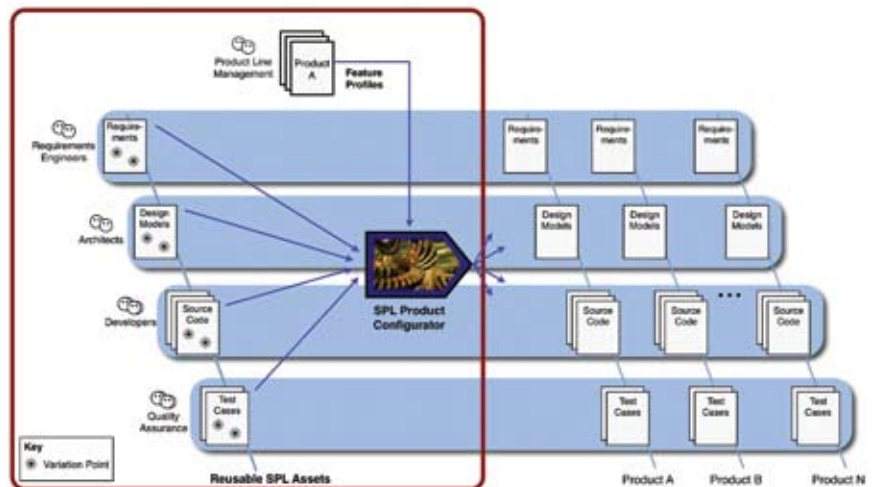


Figure 2: Simple and efficient means of production from the single-system perspective

Highlights

Figure 2 shows the single-system perspective for producing the same product line as in Figure 1, where now the focus is on the means of production inside of the red box. The same products, A through N (on the right side of the diagram), are automatically produced by a singular means of production consisting of:

- *Reusable SPL Assets (left) such as requirements, architectures, models and designs, source code components, test cases, documentation, and so forth, that can be configured and composed in different ways to create all instances of assets and products in a product line.*
- *Feature Profiles (top) that describe optional and variable features for the products in the product line, where each product in the product line is uniquely defined by its own feature profile—choices for each of the optional and variable features.*
- *SPL Product Configurator (center) that automatically composes and configures products from the reusable SPL assets, using the feature profiles to determine which reusable software assets to use and how to configure variation points within the assets.*

As highlighted by the blue bars in Figures 1 and 2, tilting your head 90-degrees provides the critical shift in perspective, from the vertical product-centric focus of Figure 1 to the horizontal single-system focus in Figure 2.

By shifting perspective to focus on the singular means of production rather than the multitude of products, products are relegated from the primary focus to a consequential corollary of the automated means of production. The complexity of managing product interdependencies is eliminated and replaced by automated production, resulting in dramatic increases in the number of products that can be effectively created, deployed and maintained.

With the single-system perspective, focused on the automated means of production, the scale of a product line and the scope of diversity within the product line can be based on business opportunities and profitability rather than being constrained by the complexity limitations imposed by the product-centric perspective.

With the single-system perspective, focused on the automated means of production, the scale of a product line and the scope of diversity within the product line can be based on business opportunities and profitability rather than being constrained by the complexity limitations imposed by the product-centric perspective.

Highlights

By focusing both your business and engineering teams on the operation of your software production line, your organization can plan, develop, deploy and evolve your product line portfolio, seamlessly and efficiently, across the full development and delivery lifecycle.

BigLever's Gears SPL Lifecycle Framework provides the automated means of production for your software-based product line. By focusing both your business and engineering teams on the operation of your *software production line*, your organization can help plan, develop, deploy and evolve your product line portfolio, seamlessly and efficiently, across the full development and delivery lifecycle—from business case and analysis, to requirements, design, implementation, testing, delivery, maintenance and evolution.

The Gears SPL framework enables the SPL integration of existing or new tools, assets and processes across the full system and software development lifecycle. With the award-winning Gears SPL lifecycle framework, you have a common set of industry standard SPL concepts and constructs for all tools and assets, including:

- *A **feature** model that can uniformly express the full product line feature diversity for all assets in all stages of the system and software development lifecycle.*
- *A **single variation point** mechanism that can be uniformly applied to all tools and their associated assets in all stages of the system and software development lifecycle.*
- *An automated product **configurator** that can automatically assemble and configure assets from each stage of the lifecycle to produce all products in a product line with the push of a single button.*

Background on MDD and IBM Rational Rhapsody

MDD technology assists professionals to achieve unparalleled productivity gains over traditional document driven approaches by enabling users to specify the system design and architecture graphically, simulate and automatically validate the system as it is being built. This helps engineers and developers to ultimately produce a quality systems specification that is correct, non-ambiguous and satisfies original requirements.

Highlights

Having your engineers work in on unified environment helps provide another advantage, in that your team can work in a single platform—the Rational Rhapsody environment aids your workflow by helping enable a UML and SysML design space contained under a single tool family.

UML and SysML are the same language, just aimed at different disciplines. This allows systems and software engineers to collaborate better than ever before. In addition, SPL is made more effective because of the collaboration amongst all the team members. These two languages are founded on the same metamodel, yet having them in different areas of the design enables specific teams to design with tools that help them do their work yet also clearly communicate their design to a larger audience. Having your engineers work in on unified environment helps provide another advantage, in that your team can work in a single platform—the Rational Rhapsody environment aids your workflow by helping enable a UML and SysML design space contained under a single tool family.

As illustrated in Figure 3, the Rational Rhapsody solution helps provide systems engineers and software developers with UML/SysML compliant products that can be extended for domain-specific modeling, aiding in creating a collaborative development environment that assists both large and small teams to communicate effectively and productively. Integrated requirements management and traceability features help ensure that the design always meets the requirements. Design for Testability (DFT) capabilities help reduce defects early in the process and always validate against the requirements. The Rational Rhapsody solution helps accelerate development by generating full applications, rather than just code frames. These technologies, packaged in an easy-to-use format, help make the Rational Rhapsody environment a powerful solution for software and systems engineers.

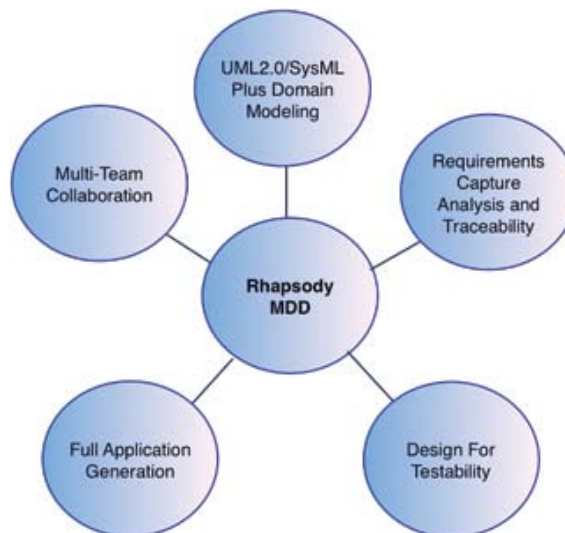


Figure 3: Model-driven development with Rational Rhapsody

Highlights

Through the Rational Rhapsody tool's Model Driven Architecture (MDA) support, development teams can target the Platform Independent Model (PIM) to a realtime embedded operating system in seconds.

The Rational Rhapsody/Gears Bridge is a dual plugin between Rational Rhapsody and Gears, shown in Figure 4. The bridge provides SPL capabilities in MDD and extends the set of supported SPL core assets to include MDD models.

Through the Rational Rhapsody tool's Model Driven Architecture (MDA) support, development teams can target the Platform Independent Model (PIM) to a realtime embedded operating system in seconds. The Rational Rhapsody solution helps provide a design approach where the software can be constantly executed and validated on the host environment, then brought down to the embedded target for target based testing. By fully integrating the specific demands of the systems engineer and the software developer, the Rational Rhapsody tool places a powerful, feature-loaded tool in the user's hands assisting them in creating high quality systems and software in a shorter timeframe.

Rhapsody/Gears Bridge—integrating MDD and SPL

The Rational Rhapsody/Gears Bridge is a dual plugin between Rational Rhapsody and Gears, shown in Figure 4. The bridge provides SPL capabilities in MDD and extends the set of supported SPL core assets to include MDD models.

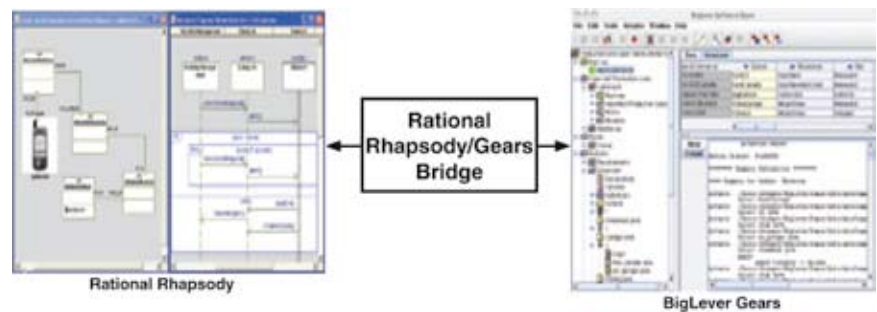


Figure 4: Rational Rhapsody/Gears Bridge

On the Gears side of the plug-in bridge, Rational Rhapsody MDD models can be included as Gears SPL modules, or core assets. The Gears product configurator can automatically instantiate Rational Rhapsody models based on the feature selections in Gears feature profiles, along with the consistent configuration of other Gears core assets such as requirements, documentation, conventional source code and test cases.

On the Rational Rhapsody side of the plugin bridge, Rhapsody model elements can be converted into first-class Gears variation points. This helps allow optional and variant elements and alternative behaviors to be specified for model elements, where the options and alternatives reflect the feature diversity that needs to be supported at the model level. Gears SPL operations, editors and power tools are available directly from the Rational Rhapsody menus, providing seamless interoperation.

A handset product line example

The simple handset product line shown in Figure 5 illustrates by example the integration of MDD and SPL via the Rational Rhapsody/Gears Bridge. This product line comprises three products and four features. Each feature column has an enumerated set of choices:

- *Call Connectivity models the types of phone calls that are support on the handset, **Voice** only or both **Video & Voice**.*
- *Memory models three levels of flash RAM that can be configured on the handset, **Low**, **Medium** or **High**.*
- *Call Recording models the which types of calls can be recorded for replay, **None**, **Voice** or **Voice & Video**.*
- *LED Flashlight models a boolean choice of whether or not the handset contains a built-in LED flashlight.*

The three different product “flavors” have feature profiles that can be seen across the rows:

- *Low end handsets are oriented towards developing regions around the world, where low cost is important and intermittent power grids make the flashlight feature valuable.*
- *Mid range handsets provide full call connectivity functionality, but the moderate amount of flash RAM makes video call recording impractical.*
- *High end handsets with the highest configuration of flash RAM can support voice and video call recording.*

Highlights




Phone	Call Connectivity	Memory	Call Recording	LED Flashlight
 Low End	Voice	Low	None	Yes
 Mid Range	Video & Voice	Medium	Voice	No
 High End	Video & Voice	High	Voice & Video	No

Figure 5: Example handset product line portfolio.

Rather than using conventional hand-coded source level development to create the software for the handsets, Rational Rhapsody allows the system structures and behaviors to be expressed using the higher level and visually rich abstractions of SysML and UML.

Rather than using conventional hand-coded source level development to create the software for the handsets, Rational Rhapsody allows the system structures and behaviors to be expressed using the higher level and visually rich abstractions of SysML and UML. The firmware source code can then be automatically generated and compiled directly from the Rational Rhapsody models.

As described earlier, the traditional approaches for creating MDD models for the three handsets in this product line are:

- ***Clone-and-own.*** *The MDD model for one device would be initially created. A Cloned copy of this model would then be made and modified for the next device. Similarly for the third clone. While there is 100% reuse at the time a cloned copy is made, there is 0% subsequent reuse since enhancements and bug fixes must be made repeatedly to each of the three copies.*

- ***One-size-fits-all.*** *One MDD model would be created for all three handsets. Feature and behavioral variations among the three devices would be managed through runtime variations in the model and controlled by different static configuration settings installed on each device. While this eliminates the need for duplicate model copies, the resulting model can become large and complex to maintain. Furthermore, the large footprint executable needed for the high end device must also fit on the low end device, driving up hardware requirements and cost for the low end device.*

The integrated MDD/SPL approach offered by the Rational Rhapsody/Gears Bridge provides the benefits of consolidation offered by the one-size-fits-all approach and the benefits of precise customization offered by the clone-and-own approach, without the associated drawbacks. Referring to Figure 2, the reusable Design Model for the handset product line contains all the feature variations required for the three handsets encapsulated in Gears variation points. These variation points are automatically configured by the Gears product configurator, based on a Gears feature profile, to produce the precisely customized SysML and UML models needed for each of the three handsets.

A Gears variation point in a Rational Rhapsody UML model is illustrated in Figure 6. The two model elements in the diagram—*In Call* and *ToggleCallRecording*—are part of a UML activity diagram for managing the handset call recording feature. *In Call* is a common model element that is the same in each of the three handset devices. *Toggle-CallRecording* is a variation point model element, as indicated by the gear annotation, that implements the alternate behaviors needed for the Call Recording feature on the different handsets (see the fourth column in Figure 5).

Comparing Figure 6 and Figure 7 shows the result of running the Gears product configurator for the high end versus the low end handset configuration. For the low end handset in Figure 7, where no Call Recording capability is needed, the *Toggle-CallRecording* activity and the *ToggleRecording* transition are grayed out in the model, indicating that they are not part of the model for the low end handset.

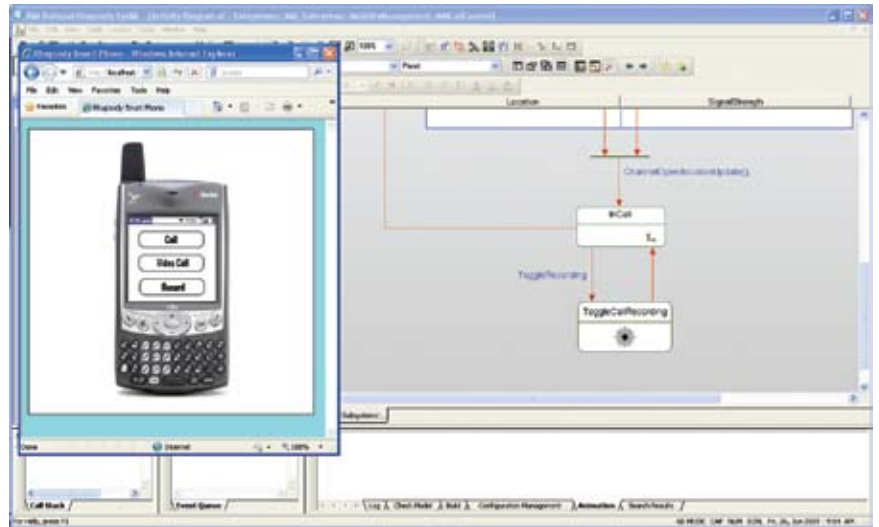


Figure 6: Gears variation point in Rational Rhapsody UML—high end handset with ToggleCallRecording enabled

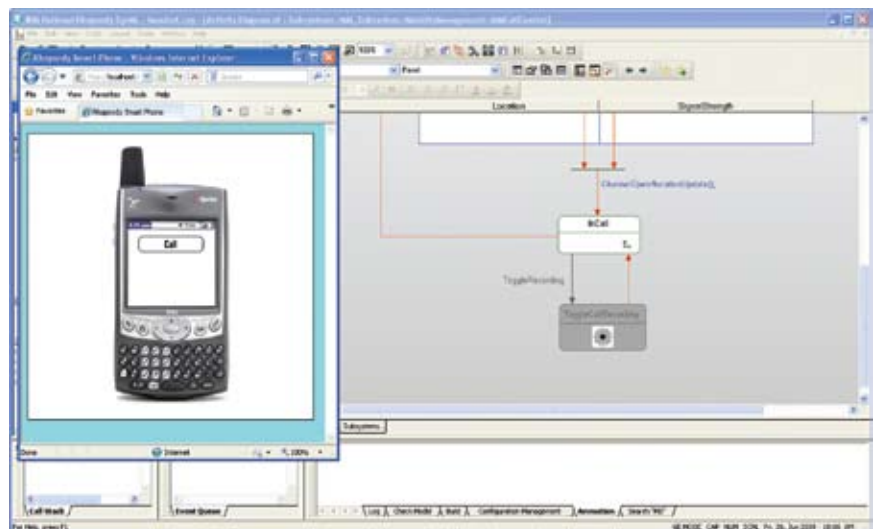


Figure 7: Gears variation point in Rational Rhapsody UML—low end handset with ToggleCallRecording disabled

In addition to variations in the model diagrams, the Rational Rhapsody/Gears Bridge helps support variations in the source code that is generated from a Rhapsody UML model. The variability in the executable implementation of the *ToggleCallRecording* variation point is illustrated by the two white boxes in Figure 8, the Gears *Variants* and the Gears *Logic*. There are three Gears variants, indicated with “V” icons in the Variants section. These are the three alternate code generation fragments needed by The Rational Rhapsody tool to help generate the different call recording variants in the handsets:

- **ActionNoRecording.** *Empty code variant for when no call recording is supported in handset. This will also result in the disabled state of the activity, as in Figure 7.*
- **ActionRecordVoice.** *Model element code fragment for voice-only call Recording in handset. This will also result in the enabled state of the activity, as in Figure 6.*
- **ActionRecordVoiceVideo.** *Model element code fragment for voice and video call recording in handset. This will also result in the enabled state of the activity, as in Figure 6.*

The Gears Logic is executed by the Gears SPL Product Configurator (see Figure 2) to instantiate the *ToggleCallRecording* variation point, differently for each handset. The *When* clauses refer to the values in the feature profile for a handset, in order to *Select* the appropriate Variant based on the feature settings.

When Gears configures all of the variation points in a Rational Rhapsody model to instantiate the firmware for a particular handset, the result is the precise firmware footprint needed for that device—nothing more, nothing less.

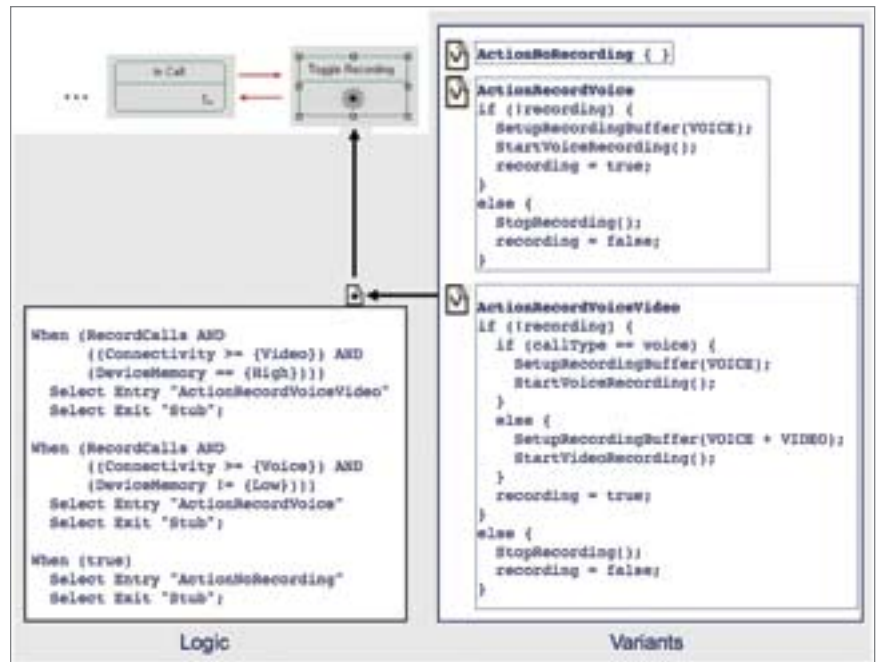


Figure 8: Code generation variants for a variation point in Rational Rhapsody

The Rational Rhapsody/Gears Bridge also supports SysML variation points. A SysML variation point for the handset call recording example is illustrated in Figure 9. The *Recording* variation point in the diagram, as indicated by the gear annotation, is for a subsystem Block that is part of a SysML Block Definition Diagram.

Comparing Figure 9 and Figure 10 shows the result of running the Gears product configurator for the high end versus the low end handset configuration. For the low end handset in Figure 10, where there is no call recording capability is needed, the *Recording* block is grayed out in the model, indicating that it is not part of the model for the low end handset.

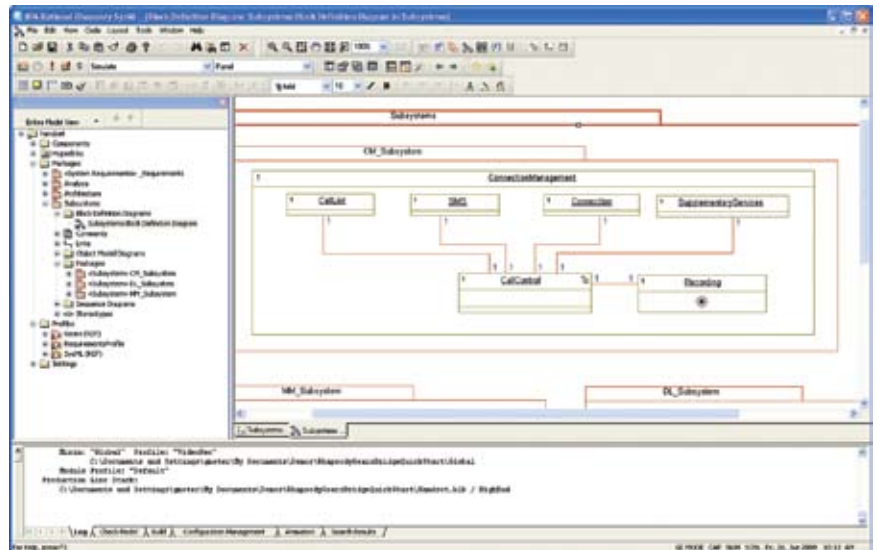


Figure 9: Gears Variation Point in Rational Rhapsody SysML—high end handset with call Recording enabled

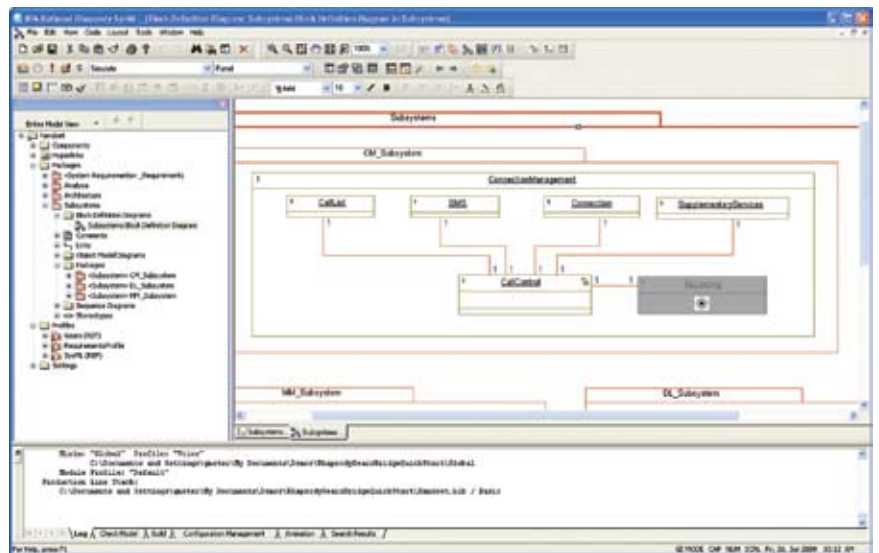


Figure 10: Gears Variation Point in Rhapsody SysML—low end handset with call Recording disabled

Highlights

Compared to conventional hand coding approaches, SysML and UML provides a powerful enabler for the rapid development of individual products within a product line, as well as better abstraction and visualization for the maintenance and evolution of those products over time.

Conclusions

Companies face complex challenges in creating and maintaining the embedded software needed to support a rapidly expanding product line portfolio. To better address this challenge, the Rational Rhapsody/BigLever Gears Bridge solution is a convergence of the synergistic MDD and SPL technologies. Compared to conventional hand coding approaches, SysML and UML provides a powerful enabler for the rapid development of individual products within a product line, as well as better abstraction and visualization for the maintenance and evolution of those products over time. SPL engineering methods and tools are specifically designed to provide the essential capabilities of expressing, encapsulating and managing the feature diversity within a product line portfolio.

The integration of MDD and SPL technologies helps provide a simple, elegant approach that enables companies to effectively incorporate the management of product diversity into their MDD processes. The integration of these highly complementary technologies allows development organizations to more effectively deal with software product line diversity across the entire portfolio development lifecycle.

With this integration, development organizations can get the help they need to achieve productivity gains and improved efficiency by:

- *Developing with Rational Rhapsody MDD models rather than conventional source code.*
- *Using Gears to create consolidated MDD core assets and automated production capabilities rather than creating cloned or one-size-fits-all MDD models.*

This reduced level of complexity enables companies to quickly and efficiently deliver more new products and features, while reducing the development effort and optimizing product quality.



What if the innovation, economy of scale, competitive advantage and profitability of your product line was limited only by your imagination rather than limited by the capacity of your engineering team?

For more information

To learn more about IBM Rational Rhapsody software from IBM, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rational

© Copyright IBM Corporation 2009

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
June 2009
All Rights Reserved

IBM, the IBM logo, ibm.com, DOORS, Rational and Rhapsody 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: ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

The information contained in this document is provided for informational purposes only and provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. Without limiting the foregoing, all statements regarding IBM future direction or intent are subject to change or withdrawal without notice and represent goals and objectives only. Nothing contained in this documentation 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.