The Promise and Challenge of Runtime Variability

Rafael Capilla, Rey Juan Carlos University, Spain
Jan Bosch, Chalmers University of Technology, Sweden

Runtime variability offers a good choice for many systems that experience dynamic changes in their quality and context.

Since the advent of software product lines (SPLs) in the 1990s as a successful approach for building multiple, related products, feature models (FMs)—compact representations of all the features of the products in the SPL—have increased in popularity. Large- and medium-size software companies now rely on numerous second-generation SPL tools such as Gears SPL Lifecycle Framework (www.biglever.com) and pure::variants (www.pure-systems.com) that use FMs to describe the variability of their products.

Today, systems with adaptive and context-aware architectures—including autonomic and ubiquitous computing systems and software ecosystems—require more dynamic capabilities to address runtime needs (S. Hallsteinsen et al., “Dynamic Software Product Lines,” Computer, Apr. 2008, pp. 93-95). In service-oriented computing, for example, the selection of services motivated by changing conditions or service-level agreements implies an automated binding of the current service to a different one. Likewise, autonomous software designed to load a new system configuration, based on a set of allowed choices, must cope with different situations.

Such systems must be able to adapt to runtime conditions and manage them with minimal human intervention.

FEATURE MODELING

When companies like Nokia, Philips, or Volvo plan to introduce a software product, they often think more in terms of user and system features rather than traditional requirements. Feature modeling is a common approach to designing products that share a set of common characteristics or are in the same market segment.

Representing variability in software architecture is still an open challenge, as capturing design choices is not part of the design itself. Standard UML notation still lacks precise mechanisms to model all variability-related issues in architecture views—for example, a runtime binding time can only be represented using stereotypes. Thus, developers use FMs to capture the design's structural variability.

In feature modeling, a system's primary functional elements are grouped into related variants to more efficiently manage the units as a whole and to identify the variants with the same binding time. In an SPL context, mass customization demands the use of concrete and valid values to establish this variability. Automated analysis of FMs can help software engineers identify incompatible configurations. However, they use these mechanisms during product derivation, not during runtime.

BINDING DESIGN-TIME AND RUNTIME VARIABILITY

Runtime variability defines those product design choices visible to customers and system users, who can select among available configurable options. Design-time variability is hidden to the user and managed by...
the product’s developers, who may decide, for cost or other reasons, to activate certain design options in a specific variant.

Software engineers use design-time variability to develop different products from a set of shared software assets. These products can exist at the same time, or developers can create them sequentially. Consequently, design-time variability, in part, “future proofs” the SPL. Ideally, business and customer needs drive the strategic decisions to introduce specific variation points, either at design time or at runtime.

Developers bind activation of a feature to variation points, and different binding times require different implementation mechanisms. When developers manage the binding time, they can delay design decisions to a later stage; sometimes the decision to activate a feature is up to the customer.

THE CHALLENGES OF RUNTIME VARIABILITY

Most existing variability models deal with the structural changes that SPL managers introduce manually. They capture the relationships defined in the variation points as well as the constraint rules and dependencies between variations.


From the perspective of runtime variation support, contemporary variability models face three main challenges:

- representing runtime variability, modifying variation points in existing and new software units during system execution, and automating system reconfiguration;
- automating runtime validation and checking reconfigured feature models to maintain system consistency and stability; and
- automating deployment and rebinding of reconfigured products at runtime with minimal interruption.

Figure 1 shows runtime variability mechanisms that are needed to transition traditional SPLs to dynamic SPLs (DSPLs).

OPEN AND CLOSED VARIABILITY MODELS

As many software systems need to reconfigure themselves dynamically, runtime variability facilitates this process, in accordance with different strategies or reconfiguration plans.

Static or closed variability FM s can’t change system features at runtime, as all variants are created and deployed a priori; runtime changes are limited to built-in variants such as feature activation. Several classes of systems thus require extensible or open variability models to add, remove, and modify variants and variation points in a controlled manner.

Adding a variant to an FM model requires determining whether it will be placed at an existing variation point or in a new location. The software must be modified accordingly. For example, it might be necessary to add require and exclude rules for the new variant to avoid incompatible product configurations. Type compatibility checks might also be needed, especially if the new variant is placed at an existing variation point. Finally, the software implementing the new variant must be added to the system.

Removing a variant means dropping it from the software that connects it to the FM. Future products won’t include this feature.

In its simplest form, modifying a variant implies changing its values; in a more complex situation, it means moving the variant to a different location in the FM and reorganizing the variability model. Moving variants from one place to another can be seen as a removal operation, followed by adding the variant to a new location.

Changing the variation points could affect the structural variability and dependency rules between features, leading to partial or complete restructuring of the variability model. As such changes could involve human intervention, complete automation would be difficult to apply in purely runtime variability models.
RUNTIME CHECKING AND REBINDING

To ensure that adding features or making changes to existing features in a system doesn’t result in incompatible configurations or, worse, system inconsistency, developers apply runtime checking techniques to a new variability model.

Tools like FaMa (www.isa.us.es/fama) consider FMs as a constraint satisfaction problem that can be solved automatically with algorithms or satisfiability solvers. These tools check variant configuration correctness during system execution, warn users about potential conflicts, and, when necessary, suggest a valid alternative. FaMa usually operates in offline mode but could be integrated with variability mechanisms to check FM changes at runtime.

Reconfiguring or simply activating new features in real-time or autonomous systems requires automatic redeployment facilities—for example, remote software that activates a sensor according to new context conditions.

As developers can reconfigure and redeploy products several times, on-demand monitoring and analysis capabilities are needed to address configuration problems during system execution. Runtime rebind mechanisms exploit variability, and configurable dynamic libraries reconnect the system or part of it to a different target or configuration.

Runtime variability offers a good choice for many systems that experience dynamic changes in their properties and context. However, selecting and activating features at runtime is far more complicated than reconfiguration. Automating decision making in FMs would go a long way toward achieving DSPLs.

Rafael Capilla is a tenured assistant professor in the Department of Computer Science and heads the Software Architectures and Internet Technologies (SAIT) research group at Rey Juan Carlos University, Madrid, Spain. Contact him at rafael.capilla@urjc.es.

Jan Bosch is a professor of software engineering and codirector of the Software Center at Chalmers University of Technology, Gothenburg, Sweden. Contact him at jan@janbosch.com.

Editor: Mike Hinchey, Lero—The Irish Software Engineering Research Centre; mike.hinchey@lero.ie

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.