Modeling and Managing Context-Aware Systems' Variability

Kim Mens, Université catholique de Louvain

Rafael Capilla, Rey Juan Carlos University

Herman Hartmann, NXP Semiconductors

Thomas Kropf, Robert Bosch GmbH

0740-7459/17/$33.00 © 2017 IEEE

MANY MODERN-DAY SOFTWARE systems exploit knowledge about their user's preferences and the environment, to trigger runtime adaptations so that they exhibit smart behavior adapted to the current situation. Such variability must happen dynamically at postdeployment time, and the variety of runtime scenarios is huge. Techniques for modeling and managing dynamic variability on the basis of context knowledge provide a powerful solution for many runtime reconfiguration challenges. This special issue provides an updated perspective on such techniques to manage variability[1] at runtime, as a way to make software systems smarter and less dependent on human intervention.

Voices from the Industry

Today's devices are ever-more connected. The number of such devices is expected to grow from 12.5 billion in 2010 to over 50 billion in 2020.[2] These devices will increasingly use information from their environment and the Internet to enhance the user experience. One example is wearable devices that collect medical data and compare it with centrally stored information to provide instant drug delivery and give feedback to the user and a physician.

Another example is the smart connected car (see Figure 1). Already today, many of its functions are heavily software-driven.[3] In the future, cars' behavior will be determined increasingly by contextual information provided by external systems.

In a smart connected car, information from the Internet, radar, and car-to-car and car-to-infrastructure systems is used to find the optimal route, gather diagnostics to monitor the vehicle's condition, deliver software updates, provide infotainment, and so forth.[4] In addition, the car's functioning is based on information coming from sensors and other car components. For instance, the optimal functioning of the engine, either combustion or electrical, depends on temperature, air pressure, and the available power resources. Optimal energy recovery needs close information exchange between the electrical powertrain and braking system.[5]

A connected car is basically a system of systems in which each subsystem operates independently to a certain extent. Both the external and internal systems contain variation points, some of which are bound at design or compile time, whereas others operate dynamically. The external systems have a different and changing feature set, depending on the location, country, weather conditions, driver preferences, or time of day. Each internal system contains its own variability and uses information from sensors and other components, again containing variation points, which all together constitute the system's context. This means that the context differs from car to car and changes dynamically as the car is driving. The car's internal software must be able to handle this plethora of information and adapt to the changing context.

However, not all contextual information can be trusted. For example, someone might hack a car to gather information about its owner or even to put the driver in unsafe situations. A connected car can become successful only when its security and safety are guaranteed. So, the contextual information must be analyzed, classified, and sometimes ignored.

A further complication is that each internal system might come from a different specialist supplier.[6] These components support a range of customers and application areas and will therefore contain variation points that might not be relevant for each receiving party. The receiving parties must bind these variation points on the basis of the context in which the systems are used—for example, the type of car or the region in which it's used.[7]

Challenges in Context-Aware Systems

Context-aware systems offer a range of contextual variability:

- simple activation or deactivation of certain system options relevant to a certain context,
- mobile applications that adapt their mode of operation to data coming from a mobile device's sensors,[8]
- web applications that activate or deactivate certain features on demand,[9]
- sophisticated adaptive software (such as in robots and unmanned vehicles), or
- critical infrastructure with stringent monitoring and reconfiguration requirements.

This diversity of runtime scenarios leads to the following key questions.

What Is Context?

Different context-aware application domains and types of systems use different kinds of physical sensors. They also have different needs regarding what context properties to sense and which ones to react to.

How Should a System Adapt?

Different systems have different needs regarding how to exhibit smart adaptation or reconfiguration. Not all systems require the same degree of adaptation. Modern context-aware systems should at least be able to activate or deactivate certain system

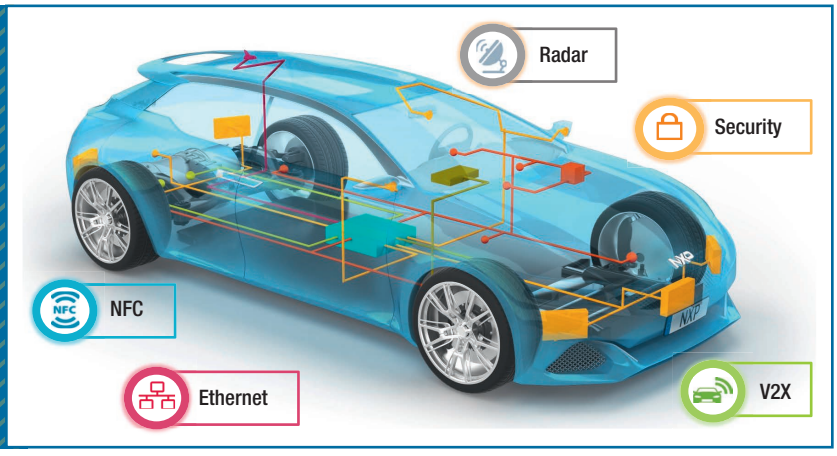

FIGURE 1. The smart connected car uses a plethora of sensors and variability to react to its environment and driver. V2X = vehicle-to-infrastructure; NFC = near-field communication. (© NXP Semiconductors; used with permission.)

features in well-defined predicted scenarios and maybe even adapt to unforeseen scenarios.

How Should We Model and Manage Dynamic Variability?

We need dedicated mechanisms to model and manage the variability of context-aware systems at runtime. These mechanisms also should adapt the structural variability on the basis of context changes[10] when functionality is dynamically added to or removed from the system. Researchers have provided dedicated modeling notations[11] or programming languages that include the notion of context as a key abstraction.[12,13]

Context Analysis and Modeling Strategies

According to Eli Rohn, a system is context-aware "if it can extract, interpret and use context information and adapt its functionality to the current context of use."[14] The growing need for context-aware software systems requires appropriate techniques for modeling, representing, and handling context-aware software variability. Understanding the key properties of and differences between different context-aware approaches is essential to understanding what approach best suits a particular class of systems.[15]

Some attempts to model the variability of context properties[7,16] rely on traditional feature models. In such approaches, context features are handled by traditional software variability techniques in which changes to the runtime context trigger changes to the software's functionality. Table 1 summarizes strategies for modeling context features using traditional software variability techniques.

The transition from closed to open variability models means that variability models can be extended with new features after design time for extensibility and evolvability. Runtime variability approaches can be used for adding or removing features, unlike with static variability models.

Facing Runtime Variability Today

Ideally, context-aware, self-adaptive systems perform some kind of reconfiguration at runtime, but facing the complexity and diversity of runtime scenarios might be challenging for critical systems. Today, many context-aware systems can activate and deactivate system options or perform updates and complex reconfigurations to adapt the system's behavior to varying scenarios. Managing context variability dynamically after postdeployment requires dedicated runtime managers that can analyze and change the state of context properties and provide a smart reaction of the system to those properties. Table 2 outlines different runtime needs that require different solutions to manage the context features.

Orthogonal to approaches inspired by feature modeling and software product lines or by self-adaptive software systems, *context-oriented programming* (COP)[12] emerged as a way to achieve dynamic context-aware software variability at the programming-language level. Most COP languages add a notion of contexts or layers as first-class entities to describe behavioral properties associated with particular contexts. As soon as a context becomes active (or inactive), the corresponding features are applied (or removed), thus making the associated behavior available (or unavailable).

In parallel with COP, the feature-oriented community has started looking into language-engineering solutions, as exemplified by *feature-oriented programming languages*.

In This Issue

This special issue presents a collection of high-quality articles that address different aspects of contextual-variability modeling, implementation, and management:

In "Learning Contextual-Variability Models," Paul Temple and his colleagues state that a system's

Table 1. Context-feature-modeling strategies using software variability.

Strategy	No. of variability models	Description	Pros	Cons
Two separate feature models	2	Context and noncontext features are modeled in two separate feature models.	Easy to model	• The need to maintain two separate models • Many dependencies between both models
One feature model with subbranches	1	Context features are modeled as subbranches of a main variability model.	• Easy to model • High reusability and maintainability of context features	Many dependencies between context and noncontext features
One feature model entangling all features	1	Context and noncontext features are modeled in one combined model.	Fewer dependencies between context and noncontext features	• Harder to model • Less reusability and maintainability of context features

Table 2. Runtime needs using contextual-variability approaches.

Runtime needs using context information	Effect on context features	Effect on the variability model	Typical application domains
Activate and deactivate sensors	Activate or deactivate features, depending on contexts triggered by the sensors.	No visible effect.	• Smart homes • Wireless sensor networks
Activate and deactivate features	Activate or deactivate a context feature.	No visible effect.	• Mobile or web applications • Software systems providing services on demand
Software update	Replace one context feature with another.	A feature is replaced by another, perhaps with a different functionality, but the variability model's structure remains the same. In more complex cases, an entire subbranch of the variability model might be replaced.	• Robots • Smart TV sets • Smart cars • Critical systems that demand postdeployment updating and reconfiguration • OSs
Adding or removing a functionality	Add or remove a context feature.	The variability model's structure is modified when features are added or removed.	• Critical systems that require operations in unattended mode • Smart cities • Cyber-foraging systems • Systems of systems

configuration space depends highly on expert knowledge and could be error-prone. They argue the potential of machine-learning techniques to learn which context factors will likely activate or deactivate system features.

In "Dynamically Adaptable Software Is All about Modeling Contextual Variability and Avoiding Failures," Ismayle de Souza Santos and his colleagues highlight the limitations of context feature modeling to describe real-world constraints. They evaluate an extended context-aware feature-modeling technique offering higher expressiveness and comprehensibility.

In "Group-Based Behavior Adaptation Mechanisms in Object-Oriented

ABOUT THE AUTHORS

KIM MENS is a professor of computer science at Université catholique de Louvain, where he leads a research laboratory on software evolution and software development technology. His research interests include software development, software maintenance, software evolution, and context-oriented programming languages. Mens received a PhD in computer science from Vrije Universiteit Brussel. Contact him at kim.mens@uclouvain.be.

RAFAEL CAPILLA is an associate professor of computer science at Rey Juan Carlos University. His research interests are software architecture, software-product-line engineering, software variability management, software sustainability, and technical debt. Capilla received a PhD in computer science from Rey Juan Carlos University. Contact him at rafael.capilla@urjc.es.

HERMAN HARTMANN is a senior enterprise architect at NXP Semiconductors. His work includes the improvement of architectural practices in IT and the introduction of applications to support hardware and software development and integrated-circuit manufacturing. As part of this work, he introduced variability management to NXP. Hartmann received a PhD in computer science from the University of Groningen. Contact him at herman.hartmann@nxp.com.

THOMAS KROPF is the senior vice president of engineering at Robert Bosch GmbH and an adjunct professor of computer science at the University of Tübingen. His research interests include software product lines with robust reuse concepts; open source software in high-quality automotive applications; validation of software with high variability; and adaptive software for different users, car lines, countries, or languages. Kropf received a habilitation in computer science from the Karlsruhe Institute of Technology. Contact him at thomas.kropf@uni-tuebingen.de.

Systems," Patrick Rein and his colleagues discuss seven mechanisms that allow object-oriented software systems to define scopes for behavior adaptation that are more detailed than mere class definitions. Behavior adaptation can be defined on groups of individual objects matching certain conditions.

In "Context-Aware Software Variability through Adaptable Interpreters," Walter Cazzola and Albert Shaqiri explore a variation of COP. Rather than offering context-oriented variability through dedicated programming-language abstractions, they move context-aware software variability support to the level of programming-language interpreters.

Creating software systems that can reconfigure at runtime on the basis of context knowledge remains a big challenge. However, this special issue presents current approaches that exemplify the combination of software variability techniques with context properties to modify software system behavior dynamically.

References

1. R. Capilla, J. Bosch, and K.C. Kang, *Systems and Software Variability Management: Concepts, Tools and Experiences*, Springer, 2013.
2. D. Evans, *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*, white paper, Cisco Internet Business Solutions Group, Apr. 2011; www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.
3. G. Spreitz, A. Zahir, and T. Kropf, "Software for the Connected Car—a Secure Open Source Platform for an App-Centric SW Architecture," *Proc. 17th Int'l Congress Electronics in Vehicles* (ELIV 15), 2015, pp. 573–587.
4. T. Kropf, "Connected Infotainment Systems—the Internet of Things in a Car," invited talk, 9th European Conf. Software Architecture (ECSA 15), 2015; s3.amazonaws.com/ieeecs.cdn.csdl.public/mags/so/.../mso2016030029s1.pdf.

5. M. Duval-Destin et al., "Impacts of an Electric Powertrain on the Braking System," *ATZ Worldwide*, vol. 113, no. 9, 2011; www.springerprofessional.de/en/impacts-of-an-electric-powertrain-on-the-braking-system/6428400.
6. H. Hartmann, T. Trew, and J. Bosch, "The Changing Industry Structure of Software Development for Consumer Electronics and Its Consequences for Software Architectures," *J. Systems and Software*, vol. 85, no. 1, 2012, pp. 178–192.
7. H. Hartmann and T. Trew, "Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains," *Proc. 12th Int'l Software Product Lines Conf.* (SPLC 08), 2008; ieeexplore.ieee.org/document/4626836.
8. S. González et al., "Subjective-C: Bringing Context to Mobile Platform Programming," *Proc. 2010 Int'l Conf. Software Language Eng.* (SLE 10), 2010, pp. 246–265.
9. N. Cardozo et al., "Features on Demand," *Proc. 8th Int'l Workshop Variability Modeling of Software-Intensive Systems*, 2014, article 18.
10. J. Bosch and R. Capilla, "Dynamic Variability in Software-Intensive Embedded System Families," *Computer*, vol. 45, no. 10, 2012, pp. 28–35.
11. N. Cardozo et al., "Semantics for Consistent Activation in Context-Oriented Systems," *Information and Software Technology*, vol. 58, 2015, pp. 71–94.
12. R. Hirschfeld, P. Costanza, and O. Nierstrasz, "Context-Oriented Programming," *J. Object Technology*, vol. 7, no. 3, 2008, pp. 125–151; www.jot.fm/issues/issue_2008_03/article4.
13. S. González, K. Mens, and A. Cádiz, "Context-Oriented Programming with the Ambient Object System," *J. Universal Computer Science*, vol. 14, no. 20, 2008, pp. 3307–3332.
14. E. Rohn, "Predicting Context-Aware Computing Performance," *Ubiquity*, Feb. 2003; ubiquity.acm.org/article.cfm?id=764011.
15. K. Mens et al., "A Taxonomy of Context-Aware Software Variability Approaches," *Modularity Companion 2016*, 2016, pp. 119–124.
16. R. Capilla, O. Ortiz, and M. Hinchey, "Context Variability for Context-Aware Systems," *Computer*, vol. 47, no. 2, 2014, pp. 85–87.