Model-Driven Engineering for Mission-Critical IoT Systems

Federico Ciccozzi, Mälardalen University

Ivica Crnkovic, Chalmers University of Technology and University of Gothenburg

Davide Di Ruscio, University of L'Aquila

Ivano Malavolta, Vrije Universiteit Amsterdam

Patrizio Pelliccione, Chalmers University of Technology and University of Gothenburg

Romina Spalazzese, Malmö University

// In mission-critical Internet of Things systems, applications require not only high availability, reliability, safety, and security but also regulatory compliance, scalability, and serviceability. In addition, they're exposed to uncertainty and variability. Model-driven engineering is a candidate for meeting these challenges. //

THE INTERNET OF Things (IoT) is characterized by an unprecedented set of heterogeneous, distributed, and intelligent things such as simple actuators, sensors, and RFID tags, as well as more complex devices such as computers, self-driving vehicles, and autonomous robots.[1] In IoT systems, heterogeneity embraces both software and hardware. Such diversification isn't trivial to handle and is exacerbated by an elemental peculiarity of the IoT: the very same software functionalities are expected to be deployable on different devices, each of them having only a limited set of core common characteristics. Moreover, things can be small or have limited resources; that is, they can have limited battery capacity, storage resources, or computational capabilities. This adds a level of complexity to deployment and redeployment of software functionalities on differently capable devices.

On one hand, the availability of many diverse heterogeneous devices collaborating in the IoT represents an unprecedented opportunity to improve quality of life, in addition to quality of service, through collaboration among industrial and consumer devices. On the other hand, to benefit from the great advantages the IoT will unleash, we must deal with a whole set of new challenges at all levels. Heterogeneity, runtime adaptability, reusability, interoperability, data mining, security, abstraction, automation, privacy, middleware, and architectures are just some of the aspects we need to consider at both design time and runtime and for which new software engineering approaches will be envisioned.[2]

Among IoT systems, an important category is mission-critical IoT (MC-IoT) systems, which run applications whose failure might have severe consequences. Mission-critical computing is becoming increasingly important, as a Hewlett Packard Enterprise survey of 200 information

 0740-7459/17/$33.00 © 2017 IEEE

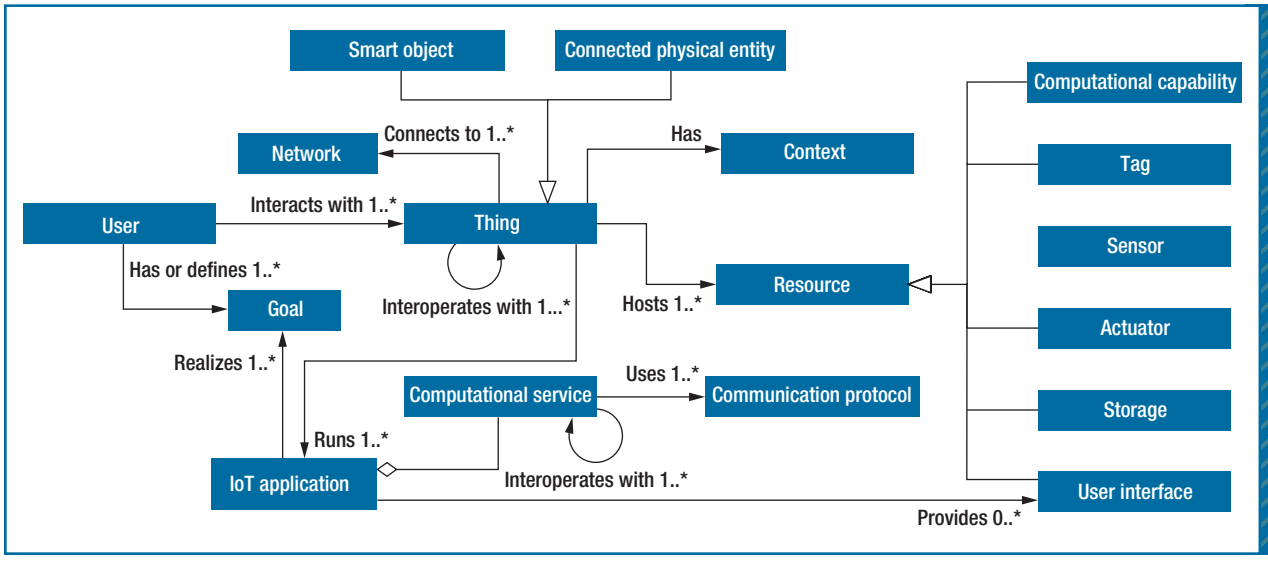

FIGURE 1. A conceptual model of mission-critical Internet of Things (MC-IoT) systems.[2,4] Model-driven engineering can help meet the technical challenges of MC-IoT system development and runtime management.

technology and business decision makers made evident.[3] In mission-critical applications, every element can be critical, from the simple RFID tag to a database or even a robot. Regarding the user experience, damage to any element in a mission-critical system would produce the same effect.

Model-driven engineering (MDE) can help meet the technical challenges of MC-IoT system development and runtime management. Toward that end, we focus here on

- high-level abstraction to address heterogeneity and system complexity;
- separation of concerns for collaborative development and to tame complexity and maintainability;
- automated mechanisms to enable runtime self-adaptation of distributed MC-IoT systems; and
- reusability to enable sustainable development in terms of time, costs, and effort.

Our research is based on an extensive study of the MC-IoT literature and on our collaboration with IoT companies at the IoTaP (Internet of Things and People; iotap.mah.se) research center and in the ECOS (Emergent Configurations of Connected Systems; iotap.mah.se/ecos) project, where the work is in coproduction. IoTaP project application areas include smart energy, smart living, smart transportation, smart cities, smart health, and smart learning.

MC-IoT Systems in a Nutshell

MC-IoT systems are characterized by the unique concepts in Figure 1. These concepts are a refinement of the most relevant concepts in "Enabling High-Level Application Development for the Internet of Things"[2] and "IoT Reference Model,"[4] in addition to the concepts specific to mission-critical systems. (For a different, general look at MC-IoT system characteristics, see the sidebar.)

Context

In Figure 1, a thing is a physical device hosting a number of resources and running different IoT applications. A thing can be specialized according to one of two concepts:

- A *connected physical entity* is a simple generic object enhanced with connectivity—for example, a sensor or actuator.
- A *smart object* is a more complex object that performs some kind of computation and exposes some intelligence.

IoT applications consist of computational services, which can communicate among themselves by exploiting specific communication protocols. Every IoT application aims to realize user-defined goals.

A physical device can host the following resources. Tags encode information related to the given physical entities (for example, RFID tags). Sensors measure speed, temperature, water level, and so on. Actuators include

CHARACTERISTICS OF MISSION-CRITICAL IOT SYSTEMS

On one side is the Internet of Things (IoT), a worldwide network of interconnected objects uniquely addressable and based on standard communication protocols (for example, HTTP).[1] On the other side are mission-critical systems, whose failure can cause significant economic, human, or environmental losses.[2] Critical systems have several basic characteristics, each involving different computer science research areas: dependability, safety, security, and real-time systems.[3] Here we look at those characteristics from the viewpoint of mission-critical IoT (MC-IoT) systems.

DEPENDABILITY

Here, the focus is on ultrareliable, fault-tolerant IoT systems, in which dependability means the extent to which users can trust a system, taking into account reliability and availability, among other things.[2] For example, Ivanovich Silva and his colleagues proposed fault tree analysis for reasoning on the probability of permanent faults (of both hardware and links) occurring in a device or group of devices.[4] More recently, researchers have proposed IoT device virtualization to support dependability patterns and implement them according to a target application's requirements.[5] This approach resulted in 98.57 percent maximum availability and 0.45 percent probability of failure.

SAFETY

Researchers have applied hazard analysis and system-safety-engineering principles to design and analyze IoT systems with safety as a first-class system property. In this context, safety implies the ability to detect and prevent unintended behavior in IoT devices (especially actuators). Researchers and practitioners are attacking such challenges as the devices' extreme heterogeneity, the lack of standardization (especially of communication protocols and network topologies), and traditional safety and defense approaches' ineffectiveness.[6]

SECURITY

IoT systems require interaction between numerous heterogeneous hardware and software elements. This aspect poses security challenges in terms of physical-level malicious attacks, system-level trust, privacy, and so on. IoT system security research has received much attention,[7] with solutions ranging from encryption mechanisms, to communication security, to sensor data protection, to encryption algorithms. Researchers have identified four main common challenges: security structures that combine control and information access, key management, security laws and regulations, and requirements for emerging applications.[8]

REAL-TIME SYSTEMS

In this context, an IoT system's correctness depends on the system components' functional behavior (for example, the output values produced by a sensor) and the timing of their actions (for example, how long a value sensed by a device is available in the cloud). Research in this area focuses largely on real-time data collection, on-the-fly data manipulation, efficient data discovery, and real-time data visualization. For example, the OpenIoT open source platform (www.openiot.eu) provides mechanisms for integrating IoT data and applications in cloud-computing infrastructures. It supports elastic real-time computation, deployment and secure access to semantically interoperable IoT apps, and high-performance real-time online data analysis.

References

1. L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, 2010, pp. 2787–2805.
2. I. Sommerville, *Software Engineering*, 9th ed., Addison-Wesley, 2010.
3. J. Rushby, "Critical System Properties: Survey and Taxonomy," *Reliability Eng. & System Safety*, vol. 43, no. 2, 1994, pp. 189–219.
4. I. Silva et al., "A Dependability Evaluation Tool for the Internet of Things," *Computers & Electrical Eng.*, vol. 39, no. 7, 2013, pp. 2005–2018.
5. K.S. Dar, A. Taherkordi, and F. Eliassen, "Enhancing Dependability of Cloud-Based IoT Services through Virtualization," *Proc. IEEE 1st Int'l Conf. Internet-of-Things Design and Implementation* (IoTDI 16), 2016, pp. 106–116.
6. C. Chen and S. Helal, "A Device-Centric Approach to a Safer Internet of Things," *Proc. 2011 Int'l Workshop Networking and Object Memories for the Internet of Things* (NOMe-IoT 11), 2011, pp. 1–6.
7. Y. Agarwal and D.K. Anind, "Toward Building a Safe, Secure, and Easy-to-Use Internet of Things Infrastructure," *Computer*, vol. 49, no. 4, 2016, pp. 88–91.
8. H. Suo et al., "Security in the Internet of Things: A Review," *Proc. 2012 Int'l Conf. Computer Science and Electronics Eng.* (ICCSEE 12), 2012, pp. 648–651.

stepper motors, LEDs, and switches that can change their state depending on the observed data. Storage systems store data related to the monitored physical entities. User interfaces interact with a specific physical entity by means of a given IoT application. Every device also has a context representing the location, other devices in the same area, the resources available nearby, and so on.

An Example MC-IoT System

Consider a smart surveillance system that aims to detect intruders entering a company's factory buildings. The system involves several drones patrolling the area, cameras moving along the building edges, fixed sensors at strategic places detecting suspicious behavior, and a human guard in one of the buildings observing the whole process.

If the system detects suspicious behavior indicating an intruder, one of the drones takes a picture of the intruder, delivers a verbal warning to the intruder, and sends a notification to the guard's smartphone. The number and types of entities in the system can vary according to the area's size, the number of buildings, and other company requirements.[5]

MC-IoT System Challenges

With the increased functionality that often must be performed in real time, MC-IoT systems are becoming more complex. Besides dealing with the challenges characteristic of any complex system (such as performance, reliability, maintainability, and evolution), mission-critical systems must cope with challenges that are specific to them.

Heterogeneity

Heterogeneity is common in MC-IoT systems, which differ in resources, protocols, hardware and software platforms, programming languages, and so on. This heterogeneity, together with the lack of standardized platform-agnostic software solutions, makes even cross-platform development intractable.

This calls for platform neutrality. The ideal way to reach this goal is through standardization of basic system functions, scalability to different platform variants, transferability throughout the network, integration from multiple suppliers, maintainability throughout the system lifecycle, and software updates and upgrades over the system's lifetime.

Large-Scale and Emergent Properties

MC-IoT systems can reach the size of tens or hundreds of distributed things. Such a system's large size hampers the manual handling of concurrency. So, it might not be feasible to manually identify critical sections or sequences of program instructions that only one subsystem might be executing at a particular time.

MC-IoT systems should be engineered to automatically scale to accommodate and take advantage of an arbitrary number of devices. These systems' level of concurrency and complexity might lead them to expose emerging properties that represent unexpected behaviors stemming from both interaction between system parts and the system's interaction with its context. Emergent properties might be beneficial but can also be harmful—for example, if they compromise system safety. So, MC-IoT systems should have orchestration mechanisms to suitably control concurrency.

Context Awareness and Uncertainty

MC-IoT systems need to adapt to changing context information, where the context can be physical, computational, or user-related. Context information is usually collected from sources that differ in quality and are often failure prone. Faulty context information might lead MC-IoT systems into runtime failure. To minimize this risk, the development, execution, maintenance, and evolution of MC-IoT systems should be supported by techniques that

- manage the variety of context information types and their relationships and
- deduce the actions MC-IoT systems should perform to meet the environmental constraints.

Additionally, MC-IoT systems live under uncertainty: they are intrinsically dynamic and might change according to the available resources, unpredictable contextual information, and so forth. They can also be subject to rigid development practices imposed by certifications and compliance with standards. So, MC-IoT system evolution and adaptation should be controlled by mechanisms that can prevent defective and malicious adaptations when possible; isolate faults; and maintain the desired security, performance, and dependability despite adaptations.

Dynamic Discoverability of Available Resources

In highly dynamic environments such as MC-IoT systems, new, unknown, or recovered devices can show up at any time. For the system to exploit them, it must have a mechanism that can dynamically discover the available resources and constraints. The system should be able to recognize, communicate with, and adjust to the devices and their characteristics.

Reusability

When developing from scratch an application that's very similar to previous ones, MC-IoT system developers often become frustrated because they don't have appropriate support for reusability. Indeed, divergent implementations of the same functionality in MC-IoT systems, such as communication protocols and controllers, exist. The lack of a systematic, disciplined, and quantifiable software engineering methodology, as well as comprehensive abstraction mechanisms for handling MC-IoT systems' increasing complexity, leads to countless similar, but not congruent, isolated solutions that can't be easily reused and combined. Systematic reusability is of paramount importance to make MC-IoT software development sustainable in the market, where expectations for new-generation devices grow at an incredible pace.

Security and Trust

As Figure 1 shows, IoT applications rely on many things connected together. Such systems' intrinsic complexity, given, for example, the multitude of protocols and APIs, exacerbates security issues, which must be dealt with. So, a key aspect is to secure MC-IoT systems' decentralized architecture—that is, all possible points vulnerable to attacks.

Additionally, with the increasing adoption of things that communicate on users' behalf, the need is growing for effective privacy and trust management mechanisms that are part of MC-IoT systems starting at design time. For example, this means that systems need to deal with data sharing and information transparency, while taking into account policies and enabling user control over personal, location, and movement information.

Applying MDE

Dealing with the challenges we just discussed is difficult. It requires abilities related to

- managing abstractions in IoT definitions,
- dealing with various degrees of automation in software development, and
- performing analyses related to different concerns of the system's intended mission, both at design time and runtime.

> Software and hardware heterogeneity is both a strength and a big challenge of IoT systems.

Abstraction, automation, and analysis are the specific aspects of MDE, which can effectively tame system complexity and heterogeneity, express domain-specific concepts, and support communication among stakeholders.

Models are first-class entities of MDE; they represent systems at an abstraction level that can enable system analysis, understanding, and manipulation in general. Models are described through a corresponding modeling language, which is defined by a metamodel describing the set of concepts and well-formedness rules to which models must conform. Model transformations are exploited to manipulate source models and eventually generate target artifacts (for example, source code or test cases).

Next, we show how MDE techniques and tools can help tackle the challenges we discussed earlier.

Heterogeneity

Software and hardware heterogeneity is both a strength and a big challenge of IoT systems.[1] For example, in the surveillance system we described earlier, the drones, cameras, and sensors are manifold and can be based on different software and hardware platforms.

Thanks to modeling languages—specifically, domain-specific modeling languages (DSMLs)—MDE can provide a unique way to represent in one place heterogeneous systems' many aspects. For instance, you can use the same abstract modeling language to configure the surveillance system's drones and sensors and specify their behavior, no matter who made them. Models defined through these languages are much more human-oriented than common code artifacts, which are naturally machine-oriented. So, software can be defined with concepts that don't necessarily depend on the underlying platform or technology.

Modeling heterogeneity in the development and runtime of complex software-intensive systems is the main goal of the GEMOC (gemoc.org) approach. Florent Latombe and his colleagues provided a metamodeling approach in GEMOC for defining DSMLs with rich concurrency semantics (a characteristic unique to MC-IoT systems).[6] Despite the promising features of current approaches such as GEMOC, they're

still rudimentary regarding analyzing the relations between logical, physical, and domain time.

Moreover, when heterogeneity is constantly present, models can become complex and hard to grasp, even for experts. MDE offers powerful instruments to support separation of concerns in terms of multiview modeling—defining and managing models from different design viewpoints. Nevertheless, there's currently no precise way to verify the consistency of the implicit (not explicitly modeled) assumptions made in the different view-specific models. Moreover, view-specific analysis results are currently difficult to leverage at the system level to verify interesting global properties. Research in this area is necessary because verification of an MC-IoT system's global properties is crucial for the successful execution of its mission.

Large-Scale and Emergent Properties

MC-IoT applications' runtime evolution is a challenge. This is particularly difficult if the application operates on code-based artifacts. For example, imagine that a specific functionality of our surveillance system (such as management of suspicious behavior) is implemented for a specific physical device that becomes unavailable at a certain point. Reallocating the functionality to a different type of device would be difficult without modifying the functionality itself.

Models@runtime techniques use models and abstractions of the runtime system and environment to effectively manage the complexity of evolving behaviors during execution.[7] You can view MC-IoT systems as a specialization of dynamically adaptive systems (DASs). Researchers have combined MDE with aspect-oriented techniques to specify and execute DASs, as in the paper "Models@run.time to Support Dynamic Adaptation,"[8] which initiated the theory behind the Kevoree Modeling Framework (modeling.kevoree.org). This approach uses MDE-related techniques for modeling and for automatically generating code. In this context, smarter, but still safe, dynamic MAPE-K-style self-adaptation in critical conditions would be interesting to pursue and would strongly advance the state of the art and practice. (MAPE-K stands for *m*onitor, *a*nalyze, *p*lan, and *e*xecute over shared *k*nowledge.)

Context Awareness and Uncertainty

As we mentioned before, MC-IoT systems are characterized by uncertainty and unexpected changes in their context. In our surveillance system example, drones can suddenly crash or drain the available batteries. To be able to adapt to these changes and thereby cope with uncertainty, things in the MC-IoT system must be designed as adaptive systems.

MDE proposes various ways to define adaptive systems and support adaptation under uncertainty. Researchers have proposed mechanisms that automatically generate alternative models to cope with different context conditions. Such mechanisms will enable identification of functional and nonfunctional tradeoffs between the models, thereby dealing with functional and nonfunctional uncertainty. In the last 10 years, researchers have employed MDE techniques to enable functional and nonfunctional adaptation of DASs to context changes in uncertain operational conditions. For instance, Heather Goldsby and Betty Cheng proposed a generative evolution-based approach that semiautomatically provides users with a set of possible configurations with different functional and nonfunctional tradeoffs, suitable for different context conditions.[9] To benefit from these approaches, MC-IoT systems will need more powerful, fully automated mechanisms for selecting the most suitable configuration, as well as self-reconfiguration capabilities.

Dynamic Discoverability of Resources

Discoverability of resources and services provided by newly available devices in the IoT is fundamental, especially for MC-IoT systems whose goals depend on such capability. In our surveillance system example, new drones and sensors might be added to increase surveillance possibilities.

Service-oriented modeling and service-oriented architectures (SOAs) define the use of models and model transformations for identifying, specifying, realizing, composing, and orchestrating services.[10] MC-IoT systems can benefit from this mature discipline to exploit models for dynamic discoverability and realization of new resources and services. For example, the WADE (Workflows and Agents Development Environment) platform supports the development of mission-critical applications based on a mix of SOAs, multiagent systems, and workflows of mission tasks.[11] The leveraging of such approaches in the IoT, especially at a large scale, will require global standards for service and agent integration, as well as for security, privacy, architecture, and communications.

Reusability

MDE is often combined with component-based software engineering to define reusable and replaceable self-contained model entities that can be properly integrated through architectures, connectors, and integration

patterns to describe complex systems.[12] In our surveillance system example, the involved entities can be modeled as different components, which can be manipulated through dedicated model transformation and analysis tools.

Researchers have aimed to exploit the power of models and model transformations to guarantee the runtime preservation of quality attributes both in isolation and in combination.[13] MC-IoT systems have an inherent issue in trading off the use of components, which is common in other domains, with the system's high variability and the need for self-adaptation. Achieving robustness in contexts with pervasive criticality and uncertainty will require appropriate mechanisms for component replication and dynamic disassembly and reassembly of components with guarantees of functional and nonfunctional preservation.

Security and Trust

Security has been largely addressed in MDE. Particularly interesting is model-driven security,[14] which defines system models together with their security requirements and uses both to generate system architectures and complete, configured access control infrastructures. Model-driven mechanisms for enforcing trust and managing privacy have been studied since MDE's birth. Particularly mature are the mechanisms for trust negotiation and management for web services.[15] These mechanisms grant access on the basis of trust established through negotiation between the service requester and the service provider. They're suitable for MC-IoT systems, in which service providers might not know the service requesters' identity in advance owing to ubiquitous and emergent services.

In our surveillance system example, MDE facilitates the definition and enforcement of user-friendly, precise, and effective security and privacy specification policies. It also facilitates the understanding of how violations of such policies might affect the system's parts.

Addressing the technical challenges we described here should occur in continuous synergy with the various possible stakeholders and others interested in the development, use, and proliferation of MC-IoT systems. In addition, a cultural shift must occur regarding how people are connected and share their personal lives and data.

References

1. K. Ashton, "That 'Internet of Things' Thing," *RFID J.*, vol. 22, no. 7, 2009, pp. 97–114.
2. P. Patel and D. Cassou, "Enabling High-Level Application Development for the Internet of Things," *J. Systems and Software*, May 2015, pp. 62–84.
3. "The Changing Face of Mission-Critical IT in an Always-On World," Hewlett Packard Enterprise; resources.idgenterprise.com/original/AST-0166749_HP_MC_wp.fin2.pdf.
4. M. Bauer et al., "IoT Reference Model," *Enabling Things to Talk*, Springer, 2013, pp. 113–162.
5. D. Bozhinoski et al., "Leveraging Collective Runtime Adaptation for UAV-Based Systems," to be published in *Proc. 42nd Euromicro Conf. Software Eng. and Advanced Applications* (SEAA 16), 2016.
6. F. Latombe et al., "Weaving Concurrency in Executable Domain-Specific Modeling Languages," *Proc. 2015 ACM SIGPLAN Int'l Conf. Software Language Eng.*, 2015, pp. 125–136.
7. N. Bencomo et al., eds., *Models@run.time: Foundations, Applications, and Roadmaps*, Springer, 2012.
8. B. Morin et al., "Models@run.time to Support Dynamic Adaptation," *Computer*, vol. 42, no. 10, 2009, pp. 44–51.
9. H.J. Goldsby and B.H.C. Cheng, "Automatically Generating Behavioral Models of Adaptive Systems to Address Uncertainty," *Model Driven Engineering Languages and Systems*, LNCS 5301, Springer, 2008, pp. 568–583.
10. A. Arsanjani, "Service-Oriented Modeling and Architecture," IBM Developer Works, 2004; www.ibm.com/developerworks/library/ws-soa-design1.
11. G. Caire et al., "WADE: A Software Platform to Develop Mission Critical Applications Exploiting Agents and Workflows," *Proc. 7th Int'l Joint Conf. Autonomous Agents and Multiagent Systems: Industrial Track*, 2008, pp. 29–36.
12. A. Cicchetti et al., "CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems," *Proc. 27th IEEE/ACM Int'l Conf. Automated Software Eng.* (ASE 12), 2012, pp. 362–365.
13. F. Ciccozzi, A. Cicchetti, and M. Sjödin, "Round-Trip Support for Extra-functional Property Management in Model-Driven Engineering of Embedded Systems," *Information and Software Technology*, vol. 55, no. 6, 2013, pp. 1085–1100.
14. D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security: From UML Models to Access Control Infrastructures," *ACM Trans. Software Eng. and Methodology*, vol. 15, no. 1, 2006, pp. 39–91.
15. H. Skogsrud, B. Benatallah, and F. Casati, "Model-Driven Trust Negotiation for Web Services," *IEEE Internet Computing*, vol. 7, no. 6, 2003, pp. 45–52.

ABOUT THE AUTHORS

FEDERICO CICCOZZI is an assistant professor at Mälardalen University's School of Innovation, Design and Engineering. His research focuses on defining metamodels and model transformations for automation aspects in the model-driven development of component-based embedded real-time systems. Ciccozzi received his PhD in computer science and engineering from Mälardalen University. He's a member of IEEE. Contact him at federico.ciccozzi@mdh.se; www.es.mdh.se/staff/266-Federico_Ciccozzi.

IVANO MALAVOLTA is an assistant professor in the Vrije Universiteit Amsterdam Department of Computer Science. His research focuses on software architecture, model-driven engineering, and mobile-enabled systems. Malavolta received a PhD in computer science from the University of L'Aquila. He's a member of ACM and IEEE. Contact him at i.malavolta@vu.nl; www.ivanomalavolta.com.

IVICA CRNKOVIC is a professor of software engineering at Chalmers University and Mälardalen University and a guest professor at the University of Osijek. He's also the director of Chalmers University's Information and Communication Technology Area of Advance. His research interests include component-based software engineering, software architecture, software development processes, and software engineering for large complex systems. Crnkovic received a PhD in computer science from the University of Zagreb. Contact him at crnkovic@chalmers.se; www.ivicacrnkovic.net.

PATRIZIO PELLICCIONE is an associate professor in the Department of Computer Science and Engineering at Chalmers University of Technology and the University of Gothenburg. His research interests include software engineering, software architecture modeling and verification, autonomous systems, and formal methods. Pelliccione received a PhD in computer science from the University of L'Aquila. Contact him at patrizio.pelliccione@gu.se; www.patriziopelliccione.com.

DAVIDE DI RUSCIO is an assistant professor in the University of L'Aquila's Department of Information Engineering, Computer Science and Mathematics. His main research interest is model-driven engineering, including domain-specific modeling languages, model transformation, model differencing, model evolution, and coupled evolution. Di Ruscio received a PhD in computer science from the University of L'Aquila. Contact him at davide.diruscio@univaq.it; www.di.univaq.it/diruscio.

ROMINA SPALAZZESE is a senior lecturer in computer science in Malmö University's Department of Computer Science. She's also a project leader and senior researcher at the Internet of Things and People Research Center (iotap.mah.se). Her research primarily concerns the engineering of complex distributed and self-adaptive systems by exploiting formal methods. Spalazzese received a PhD in computer science from the University of L'Aquila. Contact her at romina.spalazzese@mah.se; www.rominaspalazzese.com