Autonomic Mediation Middleware for Smart Manufacturing

The purpose of the Industry 4.0 initiative is to bring together IT technologies and production processes to enable the emergence of smart, connected manufacturing. Integration is, however, a complex activity that requires rethought in the context of manufacturing. Indeed, new, challenging requirements are emerging, such as dynamicity, heterogeneity, or autonomicity. Here, the authors present Cilia, a mediation middleware designed for smart manufacturing. Cilia is based on service-oriented components. It provides dynamic and autonomic features, allowing great flexibility at runtime with minimum human intervention. The authors also apply a real use case that involves supervision and maintenance of pumping stations.

Philippe Lalanda
University of Grenoble

Denis Morand
Schneider Electric

Stéphanie Chollet
Grenoble Institute of Technology

The purpose of the Industry 4.0 initiative is to bring together IT technologies and production processes to enable the emergence of smart, connected manufacturing. The term, coined in 2011 by a government-funded German project, refers to what could be the fourth industrial revolution after the following: the emergence of industrial production based on mechanization and steam power; the achievement of mass production through the use of electric engines; and the automation of production based on electronic control, computation, and communication. Industry 4.0 is based on the use of new production techniques, new materials, and the generalized adoption of digital technologies. Here, we concentrate on that last aspect, more specifically on the connectivity issue.

Manufacturing systems traditionally include two software systems with limited interactions, respectively located in production and IT facilities. On the plant floor, field devices control local operations, collect data, and monitor the local environment. Control systems are used for acquisition and supervision of high-level data. The IT level provides supporting software, including for instance business processes, enterprise information systems, and data analytics.

Industry 4.0 builds on generalized connectivity. Field devices and control systems are horizontally networked and vertically connected to supporting software. Such integration provides the opportunity to place specific orders, to individually manage items, to consider alternative ways to achieve manufacturing objectives, and to allow the systematic oversight and improvement of production activities. Enhanced connectivity brings great benefits but also raises formidable challenges, calling for new architectures and techniques.

Jay Lee and his colleagues have conducted seminal architectural work in this area.[1] They propose a unified architecture integrating data connectivity, data conversion, data storage and analysis, longer-term cognition, and self-management. Many others have based or built their research work on this architecture and focused on a single level. For instance, service-oriented technology[2] offers models that abstract from concrete interoperability protocols and shift synchronization and interaction patterns into higher levels. Middlewares such as Base[3] or RoSe[4] adapt to changes in connectivity by switching protocols on the fly.

We believe that integration work is also needed to improve connectivity between the different levels. Integration is a complex activity: software systems that can be integrated generally haven't been designed. As a result, data are represented in different ways, interfaces are mismatching, communication protocols are unlikely to be compatible, synchronization mechanisms are absent, and so on. Also, software systems have their own evolution pace, which requires frequent updates of the integration solution. Manufacturing systems are no exception to the rule. Quite the contrary, additional constraints must be considered such as the shortage of skilled administrators.

To contend with these issues and meet Industry 4.0 requirements, we present an integration middleware called Cilia. This middleware provides autonomic features[5,6] such as self-configuration and self-optimization to enhance throughput, data relevance, and energy usage.[7] It's lightweight and requires minimum human intervention. In the next section, we provide a fully implemented and operational use case, developed with Schneider Electric, to show Cilia's applicability.

A Running Example

Our research deals with generalized connectivity in smart manufacturing. The following use case, investigated with Schneider Electric, illustrates the benefits of integrating local and remote supervision. It also highlights stringent needs in terms of flexibility and autonomy.

Our use case is about the supervision of pumping stations. Pumping stations are facilities, including pumps and equipment, for moving fluids from one place to another. They're usually part of complex, distributed infrastructures managing water (for supply, drainage, and sewage). In our context, pumping stations are near Paris, France. They're part of an infrastructure for water supply using pumping but also transfer of water under gravity. The infrastructure includes 10 pumping stations with more than 100 kilovolt amperes (kVA), several lifting stations, and a major water treatment station.

Pump manufacturers have designed electronic devices to control and supervise the stations. Pumping stations have to maintain consistent pressure and water flow at any time. This is a complex process with constant adjustments. It requires closely monitoring pump faults, levels, or any other alarms and parameters to ensure and increase efficiency. In particular, if a water pump falls outside of the acceptable operating conditions provided by the manufacturer, it could cavitate. Cavitation is the formation of vapor cavities in the water within the pump. It can be destructive, entailing significant costs.

In most current installations, control and supervision are achieved by software close to the pumps and by local technicians through a specific human-machine interface (HMI). This approach suffers from many limitations. First, technicians aren't always on site and this might delay the maintenance operations. Also, they aren't experts on every piece of equipment and don't have all the necessary information and documentation about the pumps and the overall installation. Optimization in that context would be difficult, if not impossible, to achieve. Advanced analysis is often conducted offline, during post-incident reviews by experts in the US (Boston) headquarters.

Our purpose in this work is to integrate local and remote supervision and, in doing so, offer new added-value functions. In particular, the goal is to bring operational data from pumping stations to the supervising headquarters and, conversely, to provide data and instructions to the technicians (see Figure 1). We also routinely collect data on the plant floor and send it

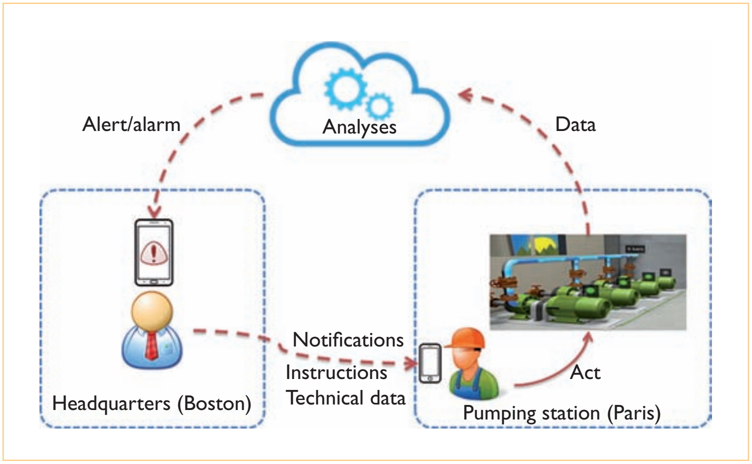

Figure 1. Use case description. The overall goal is to bring operational data from pumping stations to the supervising headquarters and, conversely, to provide data and instructions to the technicians at the pumping stations.

to a cloud infrastructure for analysis. When an abnormal situation is detected, an alert is sent to experts in Boston on their smartphone. Data must be presented in a common format that's easily understandable and usable. Experts are then expected to diagnose the problem and send instructions to local technicians. Instructions should come with all the useful information for an effective monitoring of the phenomenon, including data, images, or schemas.

This use case must meet stringent requirements. First, regarding performance, abnormal situations must be reported to the experts in less than 1 second. It should be noted here that our purpose is to identify relatively slow derivations that can be treated by a technician, with the help of remote experts. The control systems' purpose is to deal with more urgent situations.

A second major requirement is to optimize the use of the allocated bandwidth to cut costs and save energy. In practical terms, it means that, in a given situation, only relevant data must be transmitted to the cloud. Of course, relevant data and assumptions might change over time. On abnormal situations, specific data are needed to conduct appropriate analysis. Different data at lower frequencies are expected during nominal operations. This requires great flexibility at runtime to adapt the way data are collected, transformed, and transmitted.

Implementing this use case requires the timely integration of heterogeneous data sources, a domain known as mediation.[8,9] Direct connection between field devices and the cloud is certainly impractical. It's costly in term of needed bandwidth and implies code duplication to deal with security, data transformation, and so on. It's far more beneficial to rely on a dedicated communication middleware that provides an appropriate development model and technical services facilitating the integration code's development and deployment.

Enterprise service buses (ESBs) are middleware used to integrate service-based applications in IT. In most cases, ESBs require lots of memory and processing power, and they're highly technical and require deep expertise.[10] In that regard, they aren't adapted to meet Industry 4.0 requirements (to meet such requirements, we believe that mediation middlewares must be easy to use by non-IT experts, lightweight, flexible, and autonomic).[11]

One other important thing to note is that there aren't enough skilled software administrators to update mediation solutions, as opposed to IT installations where teams are dedicated to those operations. Thus, autonomic solutions are needed. Programming and runtime models must be simple. Once again, the point is to avoid complex solutions where integration code is difficult to understand, fix, and evolve.

Cilia Middleware

Now that we outlined our use case, here we provide details on how Cilia works, exploring the core concepts and features.

Core Concepts

The Cilia framework is an autonomic integration framework designed to meet the requirements of cyber-physical systems. Its purpose is to collect, synchronize, transform, and transfer data in a highly flexible way with limited human intervention. Cilia offers a set of abstractions to support design, deployment, and updates of integration solutions. Cilia is named in reference to cilium, sensory organelles found in eukaryotic cells.

In Cilia, an integration solution is a composition of domain-specific components called mediators. Each mediator implements a single mediation operation and is structured into three elements to deal with synchronization, processing, and routing issues in a modular way. Specific mediators, called adapters, implement communication protocols and handle the dynamicity of resources (devices in most cases).

Mediators are connected via bindings. A binding describes a connection between an output port and an input port. At execution time, it's realized by a communication protocol transferring data from one mediator to another. This protocol can be specified at development or deployment time, and dynamically changed at runtime.

The Cilia platform is built on top of the Open Service Gateway Initiative (OSGi), injected Plain Old Java Object (iPOJO),[12,13] and RoSe for dynamic service import/export. As Figure 2 shows, an integration solution can be distributed over several platforms through the use of diverse message-oriented middleware, including IBM Message Queuing Telemetry Transport (MQTT). This is a major feature for scaling purposes. Mediators can be moved dynamically from one platform to another to deal with load fluctuation.

Cilia provides a Domain-Specific Language (DSL). Domain-specific concepts are transformed into iPOJO components for execution. The execution framework maintains two levels. A meta-level is made of interrelated Java objects representing the specified domain-specific concepts, such as chains, mediators, or bindings. A base-level contains the iPOJO components, implementing and executing the mediation chains de facto.

The meta-level represents abstract domain concepts and links them to the code corresponding to their implementation. This level also provides a means to observe and manipulate the concepts, and subsequently their implementation. The meta-level is a causal model: modifications to the meta-level are reflected in the base level and vice versa. This architecture is key to implementing autonomic features.

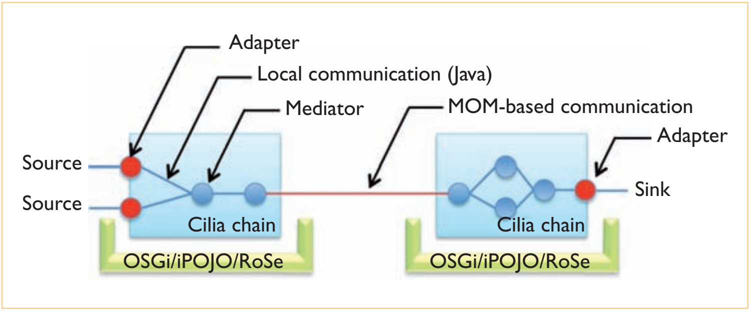

Figure 2. Cilia platform. The integration solution is distributed over several platforms through the use of diverse message-oriented middleware. Here iPOJO = injected Plain Old Java Object, MOM = message-oriented middleware, and OSGi = Open Service Gateway Initiative.

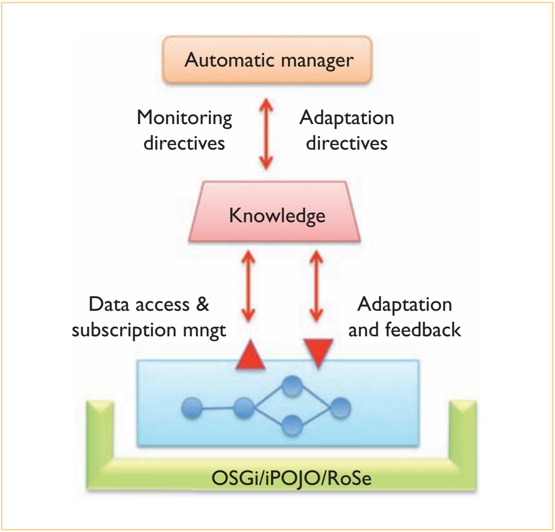

Figure 3. Autonomic aspects in Cilia. Cilia provides touchpoints to dynamically monitor and adapt mediation chains under execution.

Autonomic Features

The Cilia framework has been designed with autonomicity in mind (see Figure 3). For that purpose, it provides touchpoints to dynamically monitor and adapt mediation chains under execution.

Monitoring allows the construction of a configurable knowledge base, storing design and runtime information about the mediation chains and supporting platform. State variables are used to model the dynamics of the running chains. State variables are attached to global mediation chains and their constituents. Their values, called measures, are kept in circular lists to keep records of the past. Specifically, the state variables attached to each mediator include a scheduler start time and incoming data; processor start and end time; processor for incoming and outgoing data; mediator execution time; and a number of messages. State variables also include values of processor fields annotated by developers.

Monitoring can be activated or deactivated globally. Also, elements to be monitored and the way they're monitored can be configured without service interruption. Similarly, the knowledge base can be loaded or not, used or not, executed on a different machine (through REST interfaces) or not, and so on.

Several adaptations can be dynamically performed: mediation chains can be added or removed; configuration parameters can be updated; and

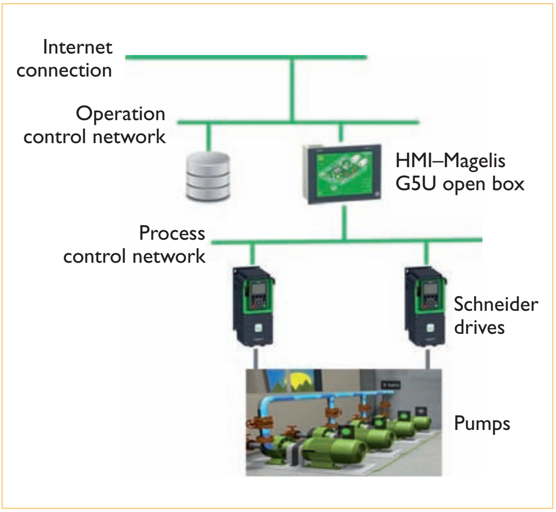

Figure 4. Control architecture. We added a database to store all the messages sent to the cloud. HMI = human-machine interface.

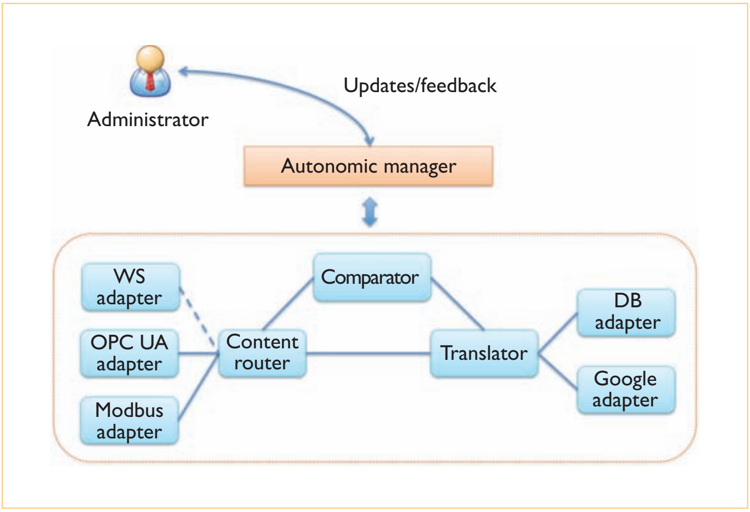

Figure 5. Mediation chain integrating drives and the cloud. US = Unified Architecture and WS = Web services.

mediators can be added, removed or replaced. Adaptation is however a complex operation, requiring preserving control flows and saving the data being processed. To do so, the Cilia framework implements a quiescence mechanism (see elsewhere[14] for details).

Adapters are also autonomic. They can handle device volatility and evolutions regarding the needed quality of service, without service interruption. This feature relies on the iPOJO ability to change service bindings when better candidates are available. The selection function is expressed with Lightweight Directory Access Protocol (LDAP). Filters can concern address ranges, devices version, and device location.[15]

It's important to recall here that system developers are responsible for the development of the autonomic managers. The purpose of the framework is to provide facilities (including touchpoints and knowledge) to ease their work.

Experiments

After setting up the framework, we implemented and carefully evaluated our use case in real settings.

Settings

As we mentioned, pumping stations are deployed around the city of Paris while domain experts are located in Boston. Pumping stations are equipped with electronic pumps controlled by independent Schneider drives. A specific HMI, connected to the drives through a Modbus network, collects about 15 measures per second for each pump and displays observed data in the form of summarized synoptic tables. Collected data are used to supervise the process but also to control the drives' proper functioning. Depending on the operational situation, different datasets must be collected at various periodicities.

We integrated the Cilia framework within the HMI. Precisely, the HMI is a Schneider Electric Magelis G5U Open Box, a highly configurable hardware based on an Intel X86 processor (1.3 GHz) with a 16-gigabit data storage (compact flash card) and a 2-Gbyte memory (RAM). We also integrated the Cilia platform in an IT server on the cloud side. As Figure 4 shows, we also have added a database to store all the messages sent to the cloud.

Finally, for the communication between the cloud and the experts' smartphones, we used Google Cloud Messaging. This mobile service allows developers to send notification data or information from IT servers to Android-based smartphones.

Implementation

We developed a Cilia mediation chain to connect the drives to the local database and cloud infrastructure. This chain is presented in Figure 5.

The mediation chain is made up of the following elements:

- *A Modbus adapter*, whose purpose is to connect to the drives, collect data, and put them in the mediation chain.

- *An OPC Unified Architecture (UA) adapter*, whose purpose is to receive alarms as publish/subscribe events.
- *A Web service adapter*, whose purpose is to get data from other HMIs. This adapter is optional and dynamically added.
- *A content router mediator*, whose purpose is to route the message as a function of its nature (alarm or data).
- *A comparator mediator*, whose purpose is to analyze the collected data against several specific thresholds and generate a message on rising edge filters.
- *A translator mediator*, whose purpose is to generate a message in a pivot format and multicasts it all outputs.
- *Database (DB) SQL* and *Google Messaging Service adapters*, whose purposes are to connect to local databases and the global management system.

Let us note that the different adapters that have been used (OPC UA, Modbus, DB SQL, and Google Messaging Service) were already part of the Cilia library and have been reused.

The nominal behavior is rather straightforward. The Modbus adapter collects data from the drives to make sure the pumps are functioning well (acting as a comparator). The OPC UA adapter receives possible alarms from other devices. Current messages are then transformed into the correct format, stored in the local storage facility, and sent to the cloud. In parallel with the mediation chain, we developed an autonomic manager. The autonomic manager can take decision and plan actions to adapt various parameters of the mediators and the mediation chain. Adaptations can be triggered in two different ways: by the administrator or by the autonomic manager itself.

In the first case, the administrator logs onto Cilia with a Web service interface through a secure virtual private network (VPN) connection, and can specify new configurations. Adaptations must not disturb the supervised process, nor lead to data losses. In general, adjustments are made when the water flow is slow or, on the contrary, when the pumps perform at peak efficiency. In both cases, cavitation risks are weak. The administrator can also send new administration goals. For instance, he might ask new devices to be supervised. Here, the autonomic manager must modify the mediation chain using Cilia facilities. For instance, he might dynamically add the Web service adapter.

In the second case, adaptations are decided upon and performed by the autonomic manager depending on the runtime conditions. In particular, when abnormal conditions are detected, data to be collected and associated periods might be changed by the autonomic manager. As before, adaptations are carefully controlled and performed during some predefined temporal windows. In nominal functioning, process data are collected every 3 seconds. When cavitation risk grows, the polling period is autonomically increased by the manager. The manager might also change the mediation chain, to get additional data, for instance. This is the case when the Web services adapter is introduced in the chain. We note, however, that the mediation chain has been designed so that it's capable of being dynamically extended. In this example, the following mediators are able to handle data coming from the Web service adapter.

Table 1. Cilia's footprint.

Modules	Footprint (kilobytes)
OSGi	483
OSGi bundles (JSON, Web service server)	4,449
iPOJO	399
Cilia core	165
Cilia runtime	459
Adapters (mediators)	55
Modbus and Web services protocols	92
OPC Unified Architecture (UA) protocol	8,900

Results

The aim of implementing this use case was multipurpose. The first goal was to explore and validate new added-value services and, at the same time, to highlight the great potential of Industry 4.0. This last aspect was highly successful, as a similar service is about to be commercialized.

The second goal was to demonstrate the scientific and technological interest of Cilia. On that last point, our purpose was to show that Cilia could run in an embedded industrial HMI, usable by non-experts, able to meet real-time constraints, and able to adapt itself to varying needs in terms of the provided data.

First of all, embedding Cilia in the Magelis was quite easy. Table 1 summarizes Cilia's footprint, which shows that it fits Magelis' capabilities.

Table 2. Update processing time.

Operation	Time (ms)
Mediator creation	130
Binding creation	60
Binding removal	50
Mediator removal	115
Mediator removal	200

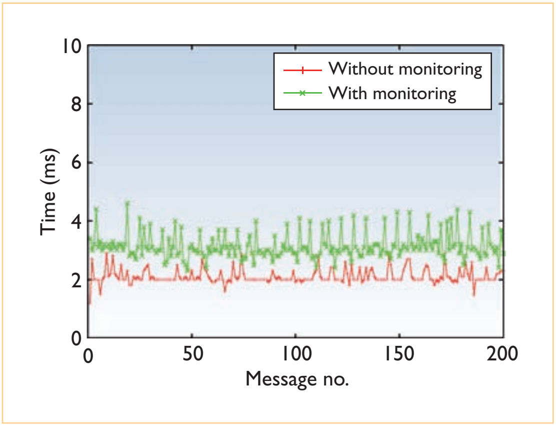

Figure 6. Monitoring cost in terms of the time needed for data to traverse a mediation chain. Overhead appears to remain stable when the number of messages increases.

Interestingly enough, it appears that protocol modules are by far the most costly elements.

Regarding performance, the use case has been entirely implemented and evaluated for several weeks. Performance-related requirements have been met as expected. In particular, all messages were delivered in less than 100 milliseconds, no message was lost, and all messages were saved in the database. A major point is that no advanced skills were necessary to implement the use case. The mediation solution has been developed by a Java programmer in 10 days and supervised by an administrator unfamiliar with Cilia intricacies. Recently, a JavaScript-based solution appeared in the industry domain. For instance, with Node-Red (see nodered.org), where the notion of a mediator is replaced by the notion of a node, dynamic updates of mediation chains are possible. However, its usage requires advanced skills in JavaScript, which is hard to find in the industrial domain.

As highlighted by Figure 5, an autonomic manager has also been developed to perform dynamic updates. Developing the autonomic manager was essentially a domain issue. It's based on simple Java programming, because Cilia provides facilities to deal with complex issues regarding monitoring and adaptation.

Finally, let's consider the overhead brought by the autonomic features. The processing time engendered by updates actually varies a lot depending on the nature of the modifications. Reconfigurations and bindings manipulations are rather cheap (see Table 2). Conversely, creations and replacements of mediators are much more costly, which seems understandable, given the cost of the quiescence mechanism. However, compare these measures with the time spent to restart the mediation chain: a full restart takes several seconds, and can reach minutes for complex configurations.

Figure 6 presents the time needed for some selected data to traverse a mediation chain, with and without monitoring. To obtain these figures, we sent up to 200 messages in the mediation chains. Monitoring overhead appears to remain stable when the number of messages increases. For our use case, the cost is reasonable because a difference of a few milliseconds has little impact.

Industry 4.0 builds on generalized connectivity and paves the way for multiple new added-value services. In this article, we described such a service, which is about to be available on the market. We also highlighted that integration is a complex issue that can't be handled by current IT solutions. New models must be provided to meet specific requirements of manufacturing.

Cilia has been specifically designed to meet the requirements of smart manufacturing and cyber-physical systems. Based on recent advances in software engineering and autonomic computing, it's easy to use, embeddable in manufacturing HMIs, effective, and capable of self-management. Internally, it's more complex than an equivalent system without autonomic features. Indeed, absorbing complexity can't be achieved without any impact on the complexity of the software itself.

As explained, the implementation of predictive maintenance functions has been successful in pumping stations. Currently, we're working to extend our solution with learning capabilities to keep information on incidents, issues, and solutions. In this way, problems that have already arisen could be handled locally and efficiently. ◧

References

1. J. Lee, B. Bagheri, and H.A Kao, "A Cyber-Physical Systems Architecture for Industry 4.0-Based Manufacturing Systems," *Manufacturing Letters*, vol. 3, Jan. 2015, pp. 18–23.
2. M. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," *Proc. 4th Int'l Conf. Web Information Systems Eng.*, 2003, pp. 3–12.
3. C. Becker et al., "Base-A Microbroker-Based Middleware for Pervasive Computing," *Proc. IEEE Int'l Conf. Pervasive Computing and Comm.*, 2003, pp. 443–451.
4. J. Bardin, P. Lalanda, and C. Escoffier, "Towards an Automatic Integration of Heterogeneous Services and Devices," *Proc. IEEE Asia-Pacific Services Computing Conf.*, 2010, pp. 171–178.
5. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
6. P. Lalanda, J.A. McCann, and A. Diaconescu, *Autonomic Computing Principles, Design and Implementation*, Springer, 2013.
7. H. Koziolek et al., "Measuring Architecture Sustainability," *IEEE Software*, vol. 30, no. 6, 2013, pp. 54–62.
8. G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *Computer*, vol. 25, no. 3, 1992, pp. 38–49.
9. G. Wiederhold and M. Genesereth, "The Conceptual Basis for Mediation Services," *IEEE Expert*, vol. 12, no. 5, 1997, pp. 38–47.
10. C. Hérault, G. Thomas, and P. Lalanda, "Mediation and Enterprise Service Bus: A Position Paper," *Proc. 1st Int'l Workshop on Mediation in Semantic Web Services*, 2005, pp. 67–80.
11. D. Morand, I. Garcia, and P. Lalanda, "Autonomic Enterprise Service Bus," *Proc. 16th IEEE Conf. Emerging Technologies & Factory Automation*, 2011, pp. 1–8.
12. R. Hall et al., *OSGi in Action: Creating Modular Applications in Java*, Manning Publications, 2011
13. C. Escoffier, R.S. Hall, and P. Lalanda, "iPOJO: An Extensible Service-Oriented Component Framework," *Proc. IEEE Int'l Conf. Services Computing*, 2007, pp. 474–481.
14. P. Lalanda, C. Escoffier, and C. Hamon, "Cilia: An Autonomic Service Bus for Pervasive Environments," *Proc. IEEE Int'l Conf. Services Computing*, 2014, pp. 488–495.
15. S. Chollet, P. Lalanda, and C. Escoffier, "Extension of Service-Oriented Component Models for Dynamic Environment," *Proc. IEEE Int'l Conf. Services Computing*, 2015, pp. 648–655.

Philippe Lalanda is a professor in computer science at Grenoble University. His research interests include autonomic computing and pervasive computing, with recent work focusing on smart factories and smart homes. Lalanda has a PhD in artificial intelligence from Nancy University, France. Contact him at philippe.lalanda@imag.fr.

Denis Morand is a software architect in the Industrial Control and Drive unit at Schneider Electric. His research interests include the Internet of Things and augmented realty for smart factories. Morand has a PhD in informatics from Grenoble University. Contact him at denis.morand@schneider-electric.com.

Stéphanie Chollet is an associate professor in computer science at the Grenoble Institute of Technology. Her research interests include software engineering, pervasive computing, and autonomic computing; in particular, a model-driven approach to facilitate the development and administration of dynamic applications. Chollet has a PhD in computer science from Grenoble University. Contact her at stephanie.chollet@grenoble-inp.fr.