CUP: A Formalism for Expressing Cloud Usage Patterns for Experts and Nonexperts

The proliferation of cloud services, from infrastructure servers to software, has led to new patterns in service deployment and provisioning practices, but not to standards for expressing and communicating such patterns to a broad-based audience. To enable the formal description and pattern classification of scenarios where cloud services are combined and provisioned to end users, we propose the Cloud Usage Patterns (CUP) formalism. With CUP, both general end users and cloud experts can express patterns, textually and visually. By expressing patterns seen in practice, we demonstrate that CUP is practically useful and makes the use of lengthy prose descriptions obsolete, which is currently common and often results in misunderstandings.

Cloud computing enables the on-demand provisioning of services, leading to time and cost efficiency. Spurred by technology advances and by the emergence of major cloud service providers, such as Amazon, Microsoft, and Alibaba, a variety of cloud services and processes for service deployment and provisioning to end users have emerged. Cloud service deployment and provisioning involve different types of service delivery models, providers, and stakeholders. Such services, and the processes where cloud services are combined in service...
chains to provide value to end users, are ripe to be described in patterns,\textsuperscript{1,2} which we refer to as \textit{cloud usage patterns}.

Cloud services are provisioned under service delivery models that bundle and abstract a set of service functionalities, or \textit{abstraction levels}. NIST differentiates between three abstraction levels: \textit{infrastructure as a service} (IaaS), \textit{platform as a service} (PaaS), and \textit{software as a service} (SaaS).\textsuperscript{3} IaaS enables the provisioning of basic infrastructure resources (e.g., servers, networks, and storage) such that an end user may deploy arbitrary software, starting with the OS. PaaS enables end users to deploy and host applications developed using libraries, services, languages, or tools provided by a cloud provider; a PaaS end user typically does not access directly the underlying infrastructure. SaaS enables end users to use applications but excludes control over infrastructure and application-hosting environments.

We distinguish between native and nonnative cloud service providers. A native cloud provider is an administered entity (i.e., an entity whose boundaries are defined with the boundaries or jurisdiction of given operation management policies) that owns cloud infrastructure (e.g., a datacenter). A nonnative cloud provider is an administered entity that does not own cloud infrastructure, but relies on provisioned resources from native cloud service providers in order to provide services to consumers. Thus, a nonnative cloud provider has the role of both a service provider and consumer. A provider offers services to consumers and may be a native or a nonnative cloud provider that operates at any of the abstraction levels. A consumer may be either a cloud provider operating at one of the abstraction levels and consuming resources at the same time, or an end user.

We also distinguish between two types of stakeholders: \textit{end users} and \textit{organizations}. An end user (an individual or a system) consumes cloud services provided by a native or nonnative cloud provider. An organization is an entity that may be either the organization to which an end user belongs or a cloud service provider.

Enabling debates about cloud service deployment and provisioning practices using intuitive descriptions and classifications is crucial, as such debates facilitate, among other things, sharing best practices and designing new cloud services. We propose in this article \textit{CUP} (Cloud Usage Patterns), a textual and visual formalism for expressing cloud usage patterns. CUP natively offers textual- and visual-language constructs enabling the expression of service chains involving multiple stakeholders and of relationships between the stakeholders (e.g., consumer and provider). The stakeholders consume or provide resources (at orders of magnitude) and operate at a given abstraction level (i.e., IaaS, PaaS, or SaaS). CUP also offers a mechanism allowing the expression of arbitrary, user-defined aspects of service deployment and provisioning (e.g., authorization policies, network properties, and so on).

In contrast to previous work, CUP is designed to service a general audience at the intersection of the research, industry, and user communities, by providing a different tradeoff between pronounceability, expressiveness, and complexity.

CUP is currently the only formalism designed to be pronounceable, which allows for fast verbal discussions about cloud service deployment practices and approaches.

CUP is less verbose than the existing specification languages for describing cloud service deployment and provisioning practices, such as OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications).\textsuperscript{4} A highly technical description, such as OASIS TOSCA, excludes common users and inevitably lengthens discussions. CUP has been specifically designed for the purpose of describing a given cloud service deployment and provisioning practice or scenario. In summary, CUP is to full-blown specification languages what domain-specific languages (DSLs—e.g., Hadoop Pig) are to full-blown programming languages (e.g., Hadoop code written in Java). We believe that by using CUP, potential and actual cloud users can easily collaborate in the specification of service-provisioning requirements, cloud system designers can identify and share best practices, and researchers and consultants can engage in pattern classification and comparison.

The formalism presented in this article has been developed by the Cloud Working Group\textsuperscript{5} of the Standard Performance Evaluation Corporation (SPEC) and officially endorsed by the Research
Group of SPEC. In this article, we begin with an example where CUP is not used, to show the practical need for CUP. We then introduce CUP; we compare CUP with popular specification languages, such as OASIS TOSCA; and we show how CUP can be used in practice. Finally, we discuss the future of CUP.

WITHOUT THE CUP FORMALISM

Consider the following cloud-service-provisioning scenario. StartupSaaS is a provider of popular social-game applications located in the US. Since its inception, StartupSaaS has used infrastructure resources leased from ExcellentIaaS in order to deploy and run gaming services. However, owing to the need for finer granularity in resource management and customization, StartupSaaS eventually built its own IaaS cloud. In addition, StartupSaaS still leases additional infrastructure resources from ExcellentIaaS when the capacity of its IaaS cloud is saturated.

Infrastructure resources are provisioned to StartupSaaS in the unit of a virtual machine (VM) of size L (large), which is how ExcellentIaaS names VMs with a 4-GHz CPU, 4 Gbytes of main memory (RAM), and 60-Gbyte hard disk space (HDD). ExcellentIaaS delivers resources to StartupSaaS only when StartupSaaS reports CPU and memory utilization higher than 80 percent of all infrastructure resources that it owns. The amount of resources leased to StartupSaaS increases exponentially while StartupSaaS’s resources are saturated and decreases linearly if StartupSaaS reports that the capacity of its infrastructure is not saturated anymore. ExcellentIaaS owns datacenters in both Europe and the US, and it provides infrastructure resources to StartupSaaS from its datacenter in the US for the sake of efficiency.

How to describe this scenario with a simple depiction and/or a string of just several characters? Would such a description ease discussions between SaaS and IaaS experts from StartupSaaS and ExcellentIaaS when combining expertise to jointly provide better services to gamers?

Sometimes StartupSaaS cannot offer a good-quality experience to its players. In such cases, explaining to the players what went wrong is part of the commercial strategy of StartupSaaS, which must be done convincingly yet without excessive technical detail. Furthermore, to propose improvements and to assess StartupSaaS’s service-provisioning practice, researchers and independent assessors need to exchange relevant information with the engineers at StartupSaaS without using lengthy prose. Would the simple formalism be helpful here?

As mentioned earlier, StartupSaaS has changed its service-provisioning practice from exclusively renting resources from ExcellentIaaS to building an in-house cloud while continuing to rent resources from ExcellentIaaS. How can the simple formalism enable the different business units within StartupSaaS to efficiently identify a promising practice, compare it with best practices in its own and other industries, and identify the changes needed to improve the service provided to their customers?

Answering these questions is the essence of our CUP formalism.

WITH THE CUP FORMALISM

CUP enables the specification of each participant in a given service chain, from the owner of virtualized or nonvirtualized hardware, to the value-adding cloud service providers operating at the different abstraction levels, to end users. CUP supports the expression of elementary patterns, where only a single provider at any abstraction level provisions resources to a consumer, and extended patterns involving, for example, hybrid resource provisioning or mediators, which we discuss later. CUP supports the expression of cloud usage patterns in both a textual and visual form.

We designed CUP to satisfy the following requirements:

- **Expressiveness.** The formalism should be expressive enough to enable the expression of any pattern seen in practice.
- **Comprehensibility.** A pattern expression should be intuitive and easy to understand.
• **Nonambiguity.** The process for expressing a cloud usage pattern should be clearly defined and produce distinct results for different cloud usage patterns (i.e., a single pattern expression should express only a single, distinct cloud usage pattern).

We summarize in the following the main features of CUP; for more details, we refer the reader to *Cloud Usage Patterns: A Formalism for Description of Cloud Usage Scenarios.*

### The Textual CUP Formalism

The expression of a cloud usage pattern with the textual CUP formalism is a string, which we refer to as a **CUP string**. A CUP string consists of the sections **Hardware resources**, **IaaS**, **PaaS**, **SaaS**, **End user**, and **Keyword descriptions**, some of which correspond to the different abstraction levels (see this article’s introduction).

Figure 1 depicts the expanded format of a CUP string for expressing elementary patterns (read from left to right; optional characters are depicted). Descriptions of the textual-language constructs depicted in Figure 1 are presented in Table 1. In Figure 2a, we present several example CUP strings that we refer to in this section and in the section “The Visual CUP Formalism.”

![Figure 1. Expanded format of a CUP (Cloud Usage Patterns) string for expressing elementary patterns. (ε is an empty string; ∗ is the Kleene operator; \( N^+ \) is the set of positive natural numbers; \( A_p \) is the set of ASCII printable characters; and the sections “IaaS,” “PaaS,” and “SaaS” have the same format as “Hardware resources.”)](image-url)
<table>
<thead>
<tr>
<th>Textual-language construct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Denotes hardware provisioning without the use of virtualization technology</td>
</tr>
<tr>
<td>v</td>
<td>Denotes hardware provisioning with the use of virtualization technology</td>
</tr>
<tr>
<td>i</td>
<td>Denotes a cloud provider at the IaaS abstraction level (optional)</td>
</tr>
<tr>
<td>p</td>
<td>Denotes a cloud provider at the PaaS abstraction level (optional)</td>
</tr>
<tr>
<td>s</td>
<td>Denotes a cloud provider at the SaaS abstraction level (optional)</td>
</tr>
<tr>
<td>e</td>
<td>Denotes an end user</td>
</tr>
<tr>
<td>.</td>
<td>Denotes the crossing of a boundary between two separate stakeholders and administered entities in a service-provisioning relationship—i.e., an external service-level agreement (optional)</td>
</tr>
<tr>
<td>v^p</td>
<td>Used for specifying an amount of resources provisioned by a provider—i.e., provisioning volume (optional; v^p is a single value or a range of values)</td>
</tr>
<tr>
<td>[ ]</td>
<td>Used for encapsulating an expression specifying replicated resource-provisioning capabilities—i.e., a replica (optional)</td>
</tr>
<tr>
<td>x v^r</td>
<td>Used for specifying a number of replicas (optional; v^r is a single value or a range of values)</td>
</tr>
<tr>
<td>-</td>
<td>Used for specifying v^p and/or v^r as value ranges (optional)</td>
</tr>
<tr>
<td>cst:</td>
<td>Custom; used for defining keywords (e.g., unit acronyms) not deemed standard by communicating parties (optional)</td>
</tr>
<tr>
<td>KeyA/B/ ...</td>
<td>User-defined keywords—keys (optional)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{ }</td>
<td>Used for encapsulating the value of a key (optional)</td>
</tr>
<tr>
<td>()</td>
<td>Used for encapsulating pattern expressions (optional)</td>
</tr>
</tbody>
</table>
A CUP string consists of at least three sections, including the sections “Hardware resources” and “End user,” and at least one of the sections “IaaS,” “PaaS,” and “SaaS.” The order of the letters denotes the provider–consumer pairs that exist in a given pattern. For instance, the CUP string CUP A (see Figure 2a) denotes the following provider–consumer pairs: (Hardware resources, IaaS), (IaaS, PaaS), (PaaS, SaaS), and (SaaS, End user).
A dot (".") marks the crossing of a boundary between a provider and consumer that are two separate stakeholders and administered entities (i.e., cloud service providers, or a cloud service provider and an end user; see the introduction). In practice, this is regulated by a service-level agreement (SLA) in which quality-of-service (QoS) requirements and customer contracts are defined. In case a boundary between two separate administered entities is crossed, we refer to the SLA between the provider and consumer as an external SLA. In case such a boundary is not crossed, we refer to the SLA between the provider and consumer as an internal SLA.

CUP supports the specification of volumes of resources provisioned by providers (single values or ranges, depicted as \(v^p\) in Figure 1). By enabling the specification of \(v^p\) as a volume range, CUP supports the specification of elastic properties of cloud providers. That is, we express elasticity, a feature for provisioning an arbitrary amount of resources when needed, by specifying the lowest and the highest amount of resources that may be provisioned by a provider at a given point in time. For instance, the CUP string \(CUP B\) (see Figure 2a) denotes that the infrastructure provider may provide between 5 and 1,000 units of infrastructure resources to the SaaS provider when needed.

CUP supports the specification of replicas (i.e., replicated resource-provisioning capabilities) and their number with the constructs \([\_]\) and \(\times v^r\) (see Table 1). The latter is important since replicas are commonly used in practice—for example, for ensuring service reliability through redundancy or providing for an order-of-magnitude better service performance to users.

We assume that a given amount of resources are provisioned from a provider to a consumer in a unit. A unit is normally specified with an acronym, which may or may not be deemed standard by the different communicating parties. For instance, one normally easily relates "VM" to "virtual machine" as a unit of resources provisioned by an IaaS provider. However, some acronyms might not be known to everyone discussing a service-deployment-and-provisioning scenario. For instance, in the example provided in the section "Without the CUP Formalism," service designers and developers at StartupSaaS might not know what exactly ExcellentIaaS understands regarding "a VM of size L."

To address the above issue, we introduced a key–value mechanism that enables the specification of keywords and associated descriptions as a writer of a CUP string sees fit. This includes unit acronyms and descriptions as well as keywords for specifying various cloud provider properties, such as geographical location and elasticity implementation details. The latter is useful since currently there are many different understandings of what is elasticity.7

As opposed to using keywords known by the communicating parties (see, for example, VM in \(CUP B\) in Figure 2a), we require that any keywords that need to be described are preceded by the keyword \(cst:\). \(CUP C\) (see Figure 2a) demonstrates the definition and description of a unit "virtual machine of size L." (see the section "Without the CUP Formalism").

CUP supports the expression of extended cloud usage patterns involving hybrid resource provisioning or mediators. Hybrid resource provisioning is provisioning of resources to a consumer from multiple native or nonnative cloud providers at the same abstraction level (see, for example, the section "Without the CUP Formalism"). For example, in \(CUP D\) (see Figure 2a), the pattern expressions (i.e., parts of a complete CUP string) \(vi\) and \(vi.\) encapsulated in parentheses express the patterns of the two different IaaS providers that provide infrastructure resources to the SaaS provider.

CUP supports the expression of value chains with mediators. In the context of cloud computing, a value chain is a network of service providers that cooperate in order to add or generate value for end users.8 An example of a mediator is an organization that leases platform resources from one or more PaaS providers, adds value to the leased services (e.g., by adding functionalities), and offers them to consumers as its own product.

When expressing value chains with mediators, pattern expressions describing service provisioning to mediators are encapsulated in parentheses (see \(CUP E\) and \(CUP F\) in Figure 2a). For instance, \(CUP E\) expresses a pattern where a mediator provides infrastructure resources to end users. The infrastructure resources are leased from an IaaS provider. \(CUP F\) expresses a pattern involving both hybrid resource provisioning and a mediator.
The Visual CUP Formalism

The visual CUP formalism is a recommendation for visualizing CUP strings in an intuitive manner and in a form easy to understand by a wide audience. CUP strings can be visualized using the visual-language constructs presented in Figure 2b. In Figure 2b, we also depict four examples of visualized CUP strings.

The CUP string $\text{CUP G}$ (see Figure 2a) expresses a pattern where a native IaaS cloud provider delivers infrastructure resources to an end user. The native cloud provider, depicted as a box with a solid line in Figure 2b, and the end user are separate administered entities. Thus, the QoS requirements and customer contracts between the provider and the end user are defined as part of an external SLA visualized with a solid line. The underlying hardware resources and the IaaS abstraction level belong to the same administered entity (i.e., the IaaS provider). Therefore, the QoS requirements and customer contracts for the provisioning of hardware resources are defined as part of an internal SLA visualized with a dashed line. The hardware resources are provisioned with the use of virtualization technology. This is reflected by placing the graphical symbol of a provider inside the box depicting the virtualization layer.

The visual CUP formalism matches its textual counterpart to only a certain extent. Definitions and descriptions of user-defined keywords (done with a key–value mechanism when writing CUP strings, see the section “The Textual CUP Formalism”) and specifications of values or value ranges ($v^r$ and $v^r$; see Figure 1) obviously cannot be visualized in a standardized manner using visual-language constructs for that purpose. Therefore, we recommend the annotation of CUP string depictions with keywords, keyword descriptions, values, and value ranges.

In Figure 2b, the depictions of the CUP strings $\text{CUP I}$ and $\text{CUP J}$ (see Figure 2a) demonstrate the use of the graphical element $[x]$ for depicting a user-defined keyword (i.e., VM; see Table 1). They also demonstrate annotating CUP string depictions with keywords, keyword descriptions, and value ranges, which is required when $[x]$ is used. Finally, the depictions of $\text{CUP I}$ and $\text{CUP J}$ demonstrate visualizing CUP strings that express patterns involving hybrid resource provisioning and mediators, respectively. For clear depiction, the graphical element visualizing hybrid resource provisioning can be placed horizontally ($\text{CUP I}$) or vertically ($\text{CUP J}$).

AN EMPIRICAL COMPARISON OF CUP WITH OTHER SPECIFICATION LANGUAGES

We compare here CUP with OASIS TOSCA in the XML and YAML (Yet Another Multicolumn Layout) formats (see the introduction). We also compare CUP with the commercial specification language Amazon Web Services CF (CloudFormation) in the JSON (JavaScript Object Notation) format.

We compare the sizes of text snippets used to express cloud usage patterns. Cloud usage pattern expressions of smaller sizes are preferable, not only for preserving space and messaging bandwidth, but also for quicker parsing by human eyes. In addition, pattern expressions with sizes in the order of a few tens of characters, or fewer, can be pronounced quickly. In order to compose expressions of minimal sizes when expressing patterns with OASIS TOSCA and Amazon CF, we applied the best practices listed in the TOSCA Primer and the documentation of Amazon CF.

The results of our study are summarized in Table 2. We consider three scenarios: a single server hosted by an IaaS cloud (“1 server”), 10 servers hosted by an IaaS cloud (“10 servers”), and two servers provisioned by two separate IaaS providers (“2 servers, hybrid provisioning”). We consider two cases for each scenario, where a server configuration is specified (“configuration”) and not specified (“no configuration”). In Table 2, the column “CUP string” presents the CUP strings for the considered scenarios, whereas the column “CUP” presents the sizes of these strings. We do not present here the cloud usage pattern expressions in the OASIS TOSCA and Amazon CF languages owing to their sizes, some of which are up to thousands of characters (see Table 2). These expressions are available online for reference at http://tinyurl.com/orw72jq.
Table 2. A comparison of CUP with other specification languages.

<table>
<thead>
<tr>
<th>Considered scenarios and CUP strings</th>
<th>CUP string</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 server (no configuration)</td>
<td>vi.e</td>
</tr>
<tr>
<td>1 server (configuration)</td>
<td>vi_{cstL}.e</td>
</tr>
<tr>
<td>10 servers (no configuration)</td>
<td>vi_{10}.e</td>
</tr>
<tr>
<td>10 servers (configuration)</td>
<td>vi_{10cstL}.e</td>
</tr>
<tr>
<td>2 servers, hybrid provisioning (no configuration)</td>
<td>(vi.)(vi.)e</td>
</tr>
<tr>
<td>2 servers, hybrid provisioning (configuration)</td>
<td>(vi_{1cstL.})(vi_{1cstL.})e</td>
</tr>
</tbody>
</table>

| Sizes (in number of characters) of expressions of cloud usage patterns |
|---------------------------------|----------------|----------------|----------------|
| Scenario                        | CUP           | Amazon CF (JSON) | OASIS TOSCA (YAML) | OASIS TOSCA (XML) |
| 1 server (no configuration)     | 4             | 1,009           | 111             | 469             |
| 1 server (configuration)        | 36            | 522             | 188             | 554             |
| 10 servers (no configuration)   | 6             | 1,569           | 543             | N/A             |
| 10 servers (configuration)      | 38            | 1,082           | 1,342           | N/A             |
| 2 servers, hybrid provisioning (no configuration) | 11         | 2,286           | 350             | N/A             |
| 2 servers, hybrid provisioning (configuration) | 50       | 1,562           | 503             | N/A             |

We find that CUP is the most compact representation and the only one that is pronounceable when simple patterns are expressed (see, for example, the scenario “1 server (no configuration)” in Table 2). In comparison to CUP, Amazon CF and OASIS TOSCA are much more verbose, which becomes evident by looking at the number of characters needed to express the considered cloud usage patterns (see Table 2). Furthermore, the OASIS TOSCA XML-based format is less expressive than CUP for complex cloud usage patterns, such as patterns describing hybrid resource provisioning, for which there is no standard approach in the format. In addition, we observed that the OASIS TOSCA YAML-based format is repetitive. For example, adding more servers (even identical; see for example, the “10 servers” scenarios in Table 2) increases significantly the size of the pattern expressions in this format.

To demonstrate the compactness, pronounceability, and practical usefulness of CUP, we surveyed real-world cloud usage patterns and expressed them using CUP. For descriptions of the
applied method for selecting these patterns and more information on them, we refer the reader to “Cloud Usage Patterns: A Formalism for Description of Cloud Usage Scenarios.”

We depict in Figure 3 the CUP strings that express the considered patterns, and their visual forms. Each pattern expressed with CUP bares a code name based on the full name of the provider that provides resources to end users—i.e., AWS (Amazon Web Services), FBK (Facebook), GAN (GoAnimate), EJT (easyJet), EZS (EZasset), FRC (Force.com), SFR (Salesforce.com), DNB (DenizBank), ZNG (Zynga), and DTO (Dito). For instance, Force.com is a PaaS provider and provides platform resources to end users conforming to the pattern $v_{ps.e}$ (the FRC scenario). Conforming to the pattern $v_{ps.e}$ (the SFR scenario), Salesforce.com offers a customer-relationship-management application to end users, whose development and hosting is supported by platform resources provisioned by Force.com.

We find that CUP fills a missing space among languages offered by current cloud providers, such as Amazon CF. Commercial cloud providers working with broad consumer bases (e.g., Amazon, Google, and Microsoft) have simplified their formulation of services and SLAs by breaking them down into individual components. For example, Amazon Web Services offers individual, unit-sized services, such as leasing a single VM or an integral multiple thereof. However, this approach leaves all the difficulty of composing a complete service to the user and does not facilitate dialogue between service providers and consumers.

On the other hand, when cloud providers work with a narrow set of customers, they use detailed contracts agreed upon in lengthy meetings. For example, financial institutions are now increasingly using cloud services with specific terms of contract negotiated extensively.

CUP aims to simplify the discussion and agreement in both cases by proposing a formal language allowing for standardized, brief, and often pronounceable descriptions of cloud service deployment and provisioning practices.

CONCLUSION

Addressing the need for a simple and compact, yet expressive formalism for expressing patterns in cloud service deployment and provisioning practices (i.e., cloud usage patterns), we introduced in this article the CUP formalism. CUP is designed for communicating cloud usage patterns in an intuitive manner and in a form easy to understand by a broad-based audience.
We foresee many practical applications of CUP, some of which are the following:

- **Basic agreement between a cloud provider and end users.** A brief specification (i.e., a pattern expression) can be used to define a pre-agreement of service.
- **Researchers describing their problem or experimental setup.** CUPs could be gainfully employed in studies about cloud service deployment and provisioning approaches, especially if there is a sufficient population of examples with correlated data (e.g., cost or performance). For instance, by correlating cost data with a variety of CUPs, the community could learn about and quantify savings of one pattern over another.
- **Acquisition process and selection of cloud offerings.** For service deployment managers and government procurement procedures, patterns expressed with CUP can be used for the automated screening of cloud service offerings. For instance, managers could use a cloud broker to select among many hundreds of offerings, especially in the well-defined cloud-infrastructure-service domain.

There are a number of ways in which this work could be continued. For instance, the tradeoff between the expressive power and length of CUP expressions could be investigated through case studies, which may lead to extending and/or adapting the existing formalism. Further, the use of CUP for identifying antipatterns in industrial use cases could be investigated. Finally, with the involvement of the community, a taxonomy for expressing patterns containing, for example, definitions of machine sizes, SLAs, and elasticity policies could be assembled. This effectively would be an effort to build a cloud-usage-pattern knowledge base.

**REFERENCES**


**ABOUT THE AUTHORS**

**Aleksandar Milenkoski** is an IT security analyst at ERNW. His research interests are in the evaluation of cloud-based systems, with a focus on intrusion detection systems. Milenkoski received his doctorate from the University of Würzburg. Contact him at amilenkoski@ernw.de.

**Alexandru Iosup** is a tenured full professor and University Research Chair at Vrije Universiteit Amsterdam, and an associate professor at the Delft University of Technology. His research focuses on massivizing (distributed) computing systems—e.g., cloud computing and

May/June 2018
big data systems. Applications include big science, business-critical workloads, online gaming, and large-scale education. Iosup received a PhD in computer science from the Delft University of Technology. He’s a member of IEEE and ACM. Contact him at a.iosup@vu.nl.

Samuel Kounev is a full professor and the chair of software engineering at the University of Würzburg. His research focuses on the engineering of dependable and efficient software systems, including software design, modeling, and architecture-based analysis; systems benchmarking and experimental analysis; and autonomic and self-aware computing. Kounev received a PhD in computer science from TU Darmstadt. He’s a member of IEEE and ACM. Contact him at samuel.kounev@uni-wuerzburg.de.

Kai Sachs is an engineering manager at Careem. His research interests include software engineering, cloud computing, and performance modeling and evaluation. Sachs received a PhD in computer science from TU Darmstadt. Contact him at kai.sachs@careem.com.

Diane E. Mularz is a principal software systems engineer at the MITRE Corporation. Her research interests are in performance and software engineering, patterns, and complex systems. Contact her at mularz@mitre.org.

Jonathan A. Curtiss is a performance and modeling engineer at the MITRE Corporation. He’s active in cloud performance modeling and system performance engineering. Curtiss received his BSc in electrical engineering from the State University of New York. He’s a Senior Member of IEEE. Contact him at jcurtiss@mitre.org.

Jason J. Ding is a senior director of the Performance Engineering Department at Salesforce.com, where he leads performance-engineering teams working on the performance and scalability of cloud-based enterprise applications and search solutions. Ding received a PhD in computer science from Texas A&M University. Contact him at jding@salesforce.com.

Florian Rosenberg is a manager and research staff member at the IBM Thomas J. Watson Research Center. His research interests are in DevOps and cloud computing, particularly software configuration and continuous deployment. Rosenberg received a PhD from the Vienna University of Technology. He’s a member of IEEE and ACM. Contact him at rosenberg@us.ibm.com.

Piotr Rygielski is a developer at the SAP Innovation Center. His research interests are in modeling and performance analysis of virtualized network infrastructures in cloud datacenters. Rygielski received a PhD from the University of Würzburg. He’s a member of IEEE. Contact him at p.rygielski@sap.com.