Bus Inf Syst Eng 62(6):467–481 (2020)
https://doi.org/10.1007/s12599-019-00627-x

RESEARCH NOTE

A Case for a New IT Ecosystem: On-The-Fly Computing

Holger Karl · Dennis Kundisch · Friedhelm Meyer auf der Heide · Heike Wehrheim

Received: 18 April 2019 / Accepted: 10 October 2019 / Published online: 9 December 2019

Abstract The complexity of development and deployment in today's IT world is enormous. Despite the existence of so many pre-fabricated components, frameworks, cloud providers, etc., building IT systems still remains a major challenge and most likely overtaxes even a single ambitious developer. This results in spreading such development and deployment tasks over different team members with their own specialization. Nevertheless, not even highly competent IT personnel can easily succeed in developing and deploying a nontrivial application that comprises a multitude of different components running on different platforms (from frontend to backend). Current industry trends such as DevOps strive to keep development and deployment tasks tightly integrated. This, however, only partially addresses the underlying complexity of either of these two tasks. But would it not be desirable to simplify these tasks in the first place, enabling one person – maybe even a non-expert – to deal with all of them? Today's approaches to the development and deployment of complex IT applications are not up to this challenge. "On-The-Fly Computing" offers an approach to tackle this challenge by providing complex IT services through largely automated configuration and execution. The configuration of such services is based on simple, flexibly combinable services that are provided by different software providers and traded in a market. This constitutes a highly relevant challenge for research in many branches of computer science, information systems, business administration, and economics. In this research note, it is analyzed which pieces of this new "On-The-Fly Computing" ecosystem already exist and where additional, often significant research efforts are necessary.

Keywords IT ecosystem · Multi-sided market · Automation · Orchestration · Service-oriented architectures · Configuration · Deployment

Accepted after two revisions by Martin Bichler.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12599-019-00627-x) contains supplementary material, which is available to authorized users.

H. Karl · F. Meyer auf der Heide · H. Wehrheim
Department of Computer Science, Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany
e-mail: holger.karl@upb.de

F. Meyer auf der Heide
e-mail: fmadh@upb.de

H. Wehrheim
e-mail: wehrheim@upb.de

D. Kundisch (✉)
Department of Business Information Systems, Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany
e-mail: dennis.kundisch@upb.de

1 Introduction[1]

Are you an IT developer who recently spent days searching for appropriate libraries reusable for your new application? Or are you a knowledgeable IT user who wanted to build a web application for your sports team, but gave up soon? Then you are one of many who have experienced the

[1] Three related overview papers (Happe et al. 2013; Petrlic et al. 2014; Szopinski et al. 2017) are also concerned with On-The-Fly Computing. Despite some overlap, these contributions differ in their scope, each addressing different aspects of On-The-Fly Computing.

complexity of development and deployment in today's IT world: Despite the existence of so many pre-fabricated components, frameworks, cloud providers, etc., building IT systems still remains a major challenge.

When *developing* applications, a developer should be familiar with a vast range of libraries, frameworks and environments: Application libraries constituting user frontends running directly on smartphones differ from frontends running inside a web browser. Application components that implement an application's business logic might run inside a web framework on a server, requiring access to different kinds of databases or event processing frameworks. A backend application component might do complex machine-learning tasks, which require yet another set of frameworks. When *deploying* an application, a similar confusion is encountered: Preparing for deploying on smartphones, web browsers, cloud environments or bare metal setups is all very different, each with its own unique set of challenges. Knowing all this likely overtaxes even an ambitious developer. Today, this results in spreading such tasks over different team members, each with their own specialization. Nevertheless, not even highly competent IT personnel can easily succeed in developing and deploying a nontrivial application that comprises a multitude of different components running on different platforms (from frontend to backend).

Current industry trends such as *DevOps* strive to keep development and deployment tasks tightly integrated. This, however, only partially addresses the underlying complexity of either of these two tasks. But would it not be desirable to simplify these tasks in the first place, enabling one person – maybe even a non-expert – to deal with them?

Today's approaches to the development and deployment of complex IT applications are not up to this challenge. To put such an approach into effect, the selection of frameworks, choice of suitable libraries, acquisition of components, generation of code, and production of executable and deployable artifacts ("software") must all be much more automated than what is feasible today. With proper automation, it should even become possible to generate, deploy, and execute software on suitable hardware on very short term, possibly even *on-the-fly*, when the need for a particular new piece of software arises. We stipulate that this is a highly relevant challenge for research in many branches of computer science and adjacent disciplines such as information systems, business administration, and economics. This challenge is clearly formidable yet it also does not appear to be hopeless. In this research note, we analyze which pieces of this new "On-The-Fly Computing" (OTF Computing) ecosystem are in place and where additional, often significant research efforts are necessary. Addressing the arising multiple engineering challenges – software, infrastructure, and market engineering challenges – would contribute to accomplishing the grand challenge of information systems "*developing model-driven methods and tools for the full-scale automated generation of implementation-ready IS*" formulated in Becker et al. (2015) and it would partly contribute to accomplish the grand challenge "*enhancing reliability of software*" proposed by the German Informatics Society (Eymann et al. 2015).

2 Use Cases, Roles, and Definition

We use three example use cases to illustrate the concept of OTF computing. They cover typical on-the-fly scenarios, ranging from straightforward to forward-looking. This description will introduce a number of roles (see also Appendix A available online via http://link.springer.com) as well as some concepts and artifacts (see also Appendix B) and relationships between them.

2.1 Example Use Cases

2.1.1 Conventional Web Applications

Let us consider a typical web *application*, geared to serve many *users* – think of an online map application or a hotel booking website. In this use case, the idea for such an application was conceived and implemented by an individual or a company; they requested the creation of this web application in a more or less implicit form (as today, OTF Computing is not explicitly developed). Such a *requester* would (today) specify that its application consists of a couple of *components*, such as a web framework or a database. These components are made available from different sources, e.g., open-source initiatives or companies, which all assume the role of a *component provider*. In addition, the code that represents the actual application semantics had to be developed; this forms yet another component to be run, e.g., as code inside a web framework.

Once all components are available, they can be run either as a single *service* or as a collection of interacting microservices (Sill 2016). In case some of the components are not available as (source or binary) code, they might still be available as a service. One example is a payment service made available via a REST interface by a *service provider* – PayPal is the canonical example here; mapping and weather forecast services are other typical examples. In summary, an application itself is a service, composed of other services that are either already running and accessed or are started as an inherent part of the application.

Such an application is typically executed on different *infrastructures*, such as users' smartphones, and on servers in a cloud system. These infrastructures are made available, explicitly or implicitly, by *infrastructure providers*: say, an Amazon web service for the web servers and the users when running frontend code in their web browser.

In this simple, well-known use case, there are two steps involved that, today, are highly labour- and knowledge-intensive: (1) the composition of an application out of simpler components, along with additional code for application semantics (today typically provided by the *developer*); (2) identifying options for executing an application's components (or accessing constituting services) on suitable infrastructures. In the following sections we shall investigate how OTF Computing would make these steps more efficient. But before, let us consider two more complex use cases.

2.1.2 Big-Data Applications in Backend Systems

Consider a machine-learning application analyzing large amounts of data in a commercial context where data scientists assist an expert from an application domain. Often, such applications change frequently and are composed of more or less simple components (e.g., database access, data preprocessing, graph extraction, clustering, various learning schemes, classification, or regression algorithms). Such machine-learning libraries exist https://spark.apache.org/docs/latest/ml-guide.html and their components can be combined, today albeit with nontrivial manual effort. In OTF Computing, this effort should be substantially reduced.

In this use case, the domain expert would be the *user*; the data scientist assumes the role of the *requester* on behalf of the user. *Components* such as Spark (Zaharia et al. 2016) are provided by corresponding *component providers* (here, the Apache consortium); *services* such as data visualization (e.g., Plotly) are provided by the corresponding *service provider* (plot.ly). Some of the *infrastructure* to run data analysis could be hosted by the data scientist's company, taking care of the *infrastructure provider* role as well.

The data scientist provides additional expertise here, which is a key contribution to the application: which components should be meaningfully composed into a big-data analysis *application*? It is a typical activity, composing components and services to form a new, useful component (or application), based on a semantic, domain-specific understanding of the problem at hand. Moreover, it could be necessary to orchestrate the execution of an application on one or multiple infrastructures, catering to the particular needs of the various services.

In today's software engineering approaches, this role is not explicitly visible; we do believe, however, that it is a key aspect for future development approaches. We hence highlight it explicitly and name it *broker* to point out its role between different other roles: The broker needs to understand requesters (and, implicitly, users) as well as the offer of component/service providers and infrastructure providers. It needs to be able to create a new component out of existing pieces plus, potentially, generate additional glue code between those pieces. This is a considerable challenge to be pursued and is, in fact, a key contribution of OTF computing: Rather than attempting to mandate (yet another) new API for composed services (a futile endeavor bound to fail), we adapt existing APIs, with their syntactic and partially semantic abilities, to integrate individual services into a new, composed service (or recursively further composing already composed services). For the example of microservices, we can generate glue code that, e.g., bridges the gap between a service intended to be used in a microservice pipeline and another one in a client-server-style orchestrated microservice.

2.1.3 User-Triggered Smartphone Application Generation

In the most forward-looking application, the role of the *broker* becomes even more important. Let us imagine that a broker is able to understand the needs of a user very well and can create an application to be executed on the user's smartphone directly, on-the-fly, as a user expressed his or her idea. This will require considerable understanding of ill-expressed, informal requests. It is far beyond today's capabilities in natural language understanding and software synthesis but serves as an interesting target scenario.

In particular, in this scenario, another issue becomes apparent: Even assuming that a broker were capable of all this semantic understanding, it is not obvious where such a broker should search for components or services to be used in generating such an application. Nor is it apparent where such services could be offered in the first place. Hence, we identify a last role necessary to implement OTF Computing: A *market provider* that creates a marketplace, operates it, and mandates and polices rules of who is allowed to offer and request what (components, services, infrastructure, ...) under which kinds of licences, etc. Again, this role is in place only for some kinds of IT systems today; for example, Apple or Google are market providers by means of their app stores. But these stores fall short of supporting the search functionality of components. Today, this role exists only in a rudimentary, highly manual form; it is shared over code websites such as GitHub, cloud providers such as Amazon or cloud consolidators such as HashiCorp. We expect an OTF computing ecosystem to be able to support single or multiple market providers and brokers, having access to automatically searchable component and service repositories.

2.2 Roles in an OTF World

In summary, OTF Computing comprises a number of familiar roles; it also points out the significance of a couple of roles that are present today, but are only filled implicitly or with a limited scope in comparison to an OTF implementation. First and most important among those is the *broker* role. This role will be crucial to create new components, services and applications and orchestrates the development of newly required functionalities. Second, the *market provider* will provide the organizational, economic, contractual and possibly legal framework in which an OTF economy can develop and flourish.

The key relationships of these roles is summarized in Fig. 1. It shows only the most important interactions between roles. We would like to emphasize that a participant on an OTF marketplace can assume several roles simultaneously (e.g., a developer might take on the roles of a service provider and a component provider at the same time) and that a role can be jointly assumed by several entities (e.g., all developers individually assuming the role of a component provider jointly assume the role of the market provider). Hence, OTF Computing is not limited to specific (centralized) market structures but also allows for a broad range of diverse market structures including decentralized settings (e.g., peer-to-peer).

2.3 Definition and Description of OTF Computing

OTF Computing refers to the approach of providing complex IT services through largely automated configuration and execution. The configuration of such services is based on simple, flexibly combinable services that are provided by different software providers and traded in a market. Several compute centers compete for execution. They have the know-how and technical prerequisites to efficiently execute composed services. In order to make this OTF Computing attractive for customers and providers, a variety of tasks must be performed. Examples are the user-friendly description of requested IT services, assurance of the quality of provided services, targeted further development of the underlying markets from customers and providers, protection of the market participants as well as the support of interaction in these dynamically changing markets. OTF Computing is characterized by the fact that the user need not create or configure the service, and this composed service is nevertheless made available promptly. Moreover, such composed services are adaptive and can be improved at runtime on the basis of explicit or implicit feedback.

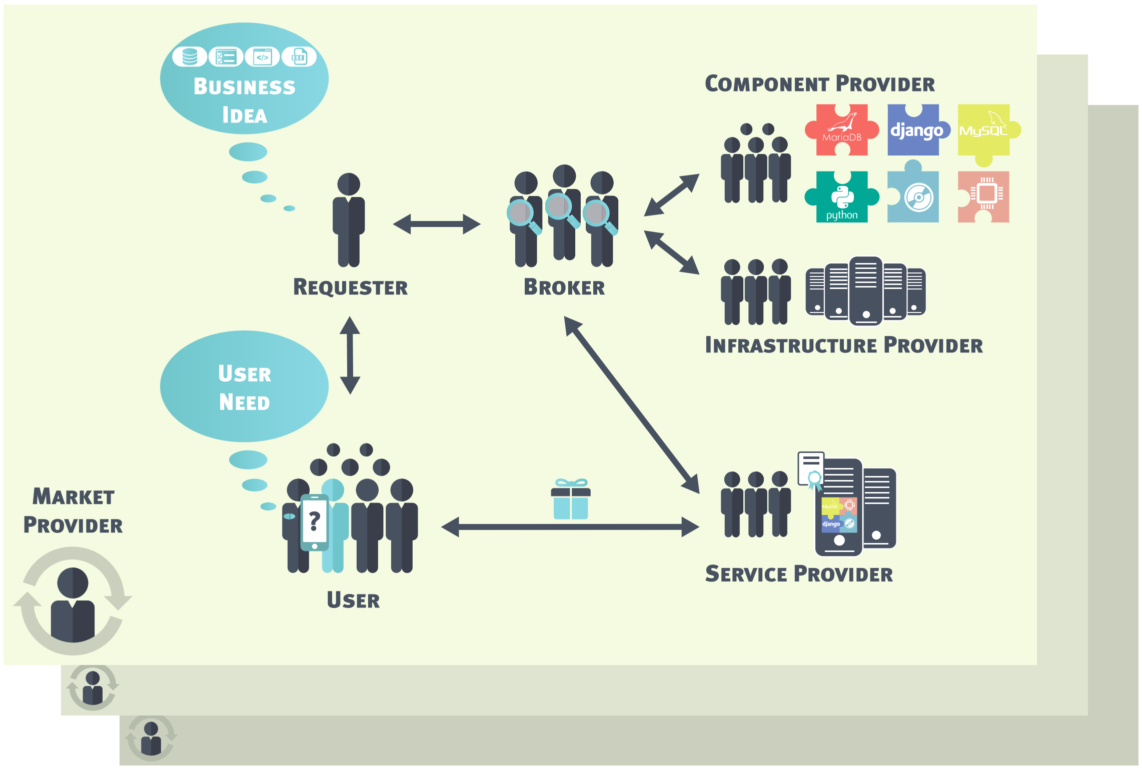

Fig. 1 Roles and their relationships in OTF computing

They adapt to a changing software landscape, for example through the availability of new components.[2]

Thus, the major innovation of OTF Computing is the integration of a variety of disciplines from computer science, information systems, business administration, and economics, rather than its contributions to its ingredients such as cloud computing, service-oriented architectures (SOA) or virtualization, for example.

3 Software Engineering Techniques for OTF

OTF Computing provides complex IT services through largely automated configuration and execution. For software engineering, OTF Computing poses three key challenges, primarily related to the development of high quality applications and the interaction of requester and user with the OTF ecosystem as a whole.

Description	How to tell the broker what is to be generated or executed? And, equally important, how to describe what is available (components, services, infrastructure) and thus usable by the broker? The options range from natural language descriptions to full formal specifications.
Configuration	How to assemble which components and services? Where to deploy them? On an abstract level, this is a planning or optimization problem; for the final execution, this is a question of interfaces, platforms, technology standards and hardware.
Quality assurance	How to ensure high quality? Current software engineering methods for quality assurance (e.g., testing, analysis, monitoring, certification) can be leveraged, but need to be adapted to the OTF Computing context.

The challenges and hence the methods to tackle them are closely interconnected. We discuss them in more detail and explain, in particular, which roles face which challenge(s).

[2] A similar definition and description of OTF Computing can be found in, e.g., (Happe et al. 2013). We note that realizing the OTF Computing vision can be interpreted as the development of a novel and special type of service system contributing to the stream of research on service systems engineering (Böhmann et al. 2014) with a focus on the engineering services architectures.

3.1 Description

Within the OTF ecosystem, three roles need *languages* for describing requirements (on applications to be built and on single components or services to be found in the market) or guarantees (of the specific infrastructure provided): The first is the requester. Preferably, she would not be required to learn a specific language for writing requirements. Rather, she would like to fill in a form or directly write her requirements in her own mother tongue. The second role is the broker. For his task, he needs languages for directly asking providers or querying the market for a specific entity. The third sort of role is taken by all providers (of services, components or infrastructure). They need to precisely formulate the guarantees provided by their entities. For all of these tasks, domain-specific languages could be envisaged, but also general-purpose languages adequate for specifying all aspects of services, components and infrastructure.

3.1.1 Existing Approaches

Semantic descriptions of services are primarily employed in the area of web services. Such descriptions use standardized languages (e.g., WSDL or OWL-S standardized by W3C) for defining types, interfaces or formats for message exchange; some formalisms (e.g., WSRF) also give semantics about stateful vs. stateless components of a service that can be used to drive execution decisions. Ontologies serve as a way of formalizing web service descriptions in general and stating the concepts of an application domain and their relationships (Gruber 1993; Oberle et al. 2006). Based on these formalizations, (description) logics can be used to reason about service descriptions, e.g., when matching offered and requested services (Paolucci et al. 2002). While ontology-based approaches restrict such reasoning to concepts and their relationships, formal methods such as Z (for stating types and operations) or the process algebra CSP (for describing workflows) offer an even higher degree of precision by giving a semantics to concepts themselves. This allows for a formal proof of the matching between two given services or of the correctness of a service with respect to its stated interface. With respect to the formulation of natural language requirements, we currently see the rise of systems communicating with humans in natural language (Serban et al. 2016; Masche and Le 2018) (virtual assistants such as Apple's Siri or Amazon's Alexa). They (seem to) understand questions and can provide answers.

While all these approaches have brought considerable progress in describing services – also in an OTF context – the challenges in the area of description imposed by OTF Computing are still not fully solved. On the side of the human requester, current approaches fall short in flexibility: natural language queries about *unknown* entities (i.e., not yet assembled service compositions) cannot be looked up in the database and today's virtual assistants will not engage in interactions with requesters to make non-understood queries more precise. OTF Computing needs *content-wise enquiries* to close gaps in the knowledge, learn new knowledge and suggest alternatives when no direct answer is available (Geierhos and Bäumer 2017).

For the communication between broker and providers, the difficulties lie in (1) existing description languages not being used at all (lots of libraries describe their offered services simply in English), (2) non-standardized languages or languages without a formal semantics being used, and (3) a missing agreement on a single language – even in one application area. As a consequence of the lack of a universal, standardized description language, the challenge lies in matching requests and guarantees given in completely different languages against each other (Platenius et al. 2017).

3.1.2 Research Required

To achieve true OTF Computing systems, novel techniques for natural language processing, building dialogs with users and learning from past dialogs are required. On the specification language side, our prediction is that no universal language will emerge, but rather many domain-specific languages. The key requirement in all these areas is, however, not language design but willingness to converge, standardize and use a small set of languages.

3.2 Configuration

In OTF Computing, configuration means *assembling* an application that meets the requester's requirements. In addition, selected services might need to be *configured* (in the sense of setting a service's parameters). Both is the task of the broker.

3.2.1 Existing Approaches

A prerequisite for configuration is first of all the existence of services that can be freely combined into larger software. This idea of software composition has long been advocated by work on SOA (Erl 2005) and, more recently, on microservices (Shadija et al. 2017). A major number of works in these areas target (a) design principles of services and (b) technological concepts that allow the use and combination of services in SOA or a microservice architecture. For the design, issues such as abstraction, composability, isolation, reusability and, in general, separation of concerns play a major role. On the technological side, various protocols, interface description languages and container concepts have been developed (REST, Docker container, to name just a few).

Contrary to that, configuration in OTF Computing refers to the *conceptual* task of selecting and assembling a number of existing services into a more complex composition. Techniques such as microservices thus ideally complement the configuration process in OTF Computing. Current approaches to automated software and service composition (Mohr 2016) typically adopt one of the following forms: *Template*-based approaches assume the structure of the service composition to be given a priori and configuration consists of instantiating placeholders, setting parameters, or customizing variants (e.g., Berardi et al. 2005; Nair et al. 2018). These approaches are applied in web service composition, for software product lines or simply for configuring software. They often amount to solving an optimization problem, e.g., finding an optimal configuration taking preferences of the requester into account (Lamparter et al. 2007). *Free* configuration on the other hand builds on no such assumptions. The construction of the overall structure is part of the configuration process itself. Such approaches require formal specifications of both the requirements and the services [in propositional or first-order logic, e.g., (Hoffmann et al. 2008)]. Free configuration approaches rely on AI *planning* in one form or another.

While the drawback of template-based approaches is the necessity of knowing the template beforehand, the difficulty for free configuration lies in requiring formally specified services. Again, it has to be stated that such specifications are often not available. As another drawback of free configurations, planning often builds only sequential compositions of services/components: i.e., it only finds a very limited structure. As a consequence, neither template nor free techniques are applied on a wide scale today. An exception to this are domain-specific configurations, an example being automatic assembly of machine-learning applications (AutoML, e.g., Thornton et al. 2013). Domain-specific approaches can typically go beyond general approaches by leveraging expert knowledge on the application domain and its structure, and thus achieve a higher degree of automation.

3.2.2 Research Required

The template-based configuration approach needs more general ways of describing templates; free configuration needs techniques for building more complex structures. There are, however, also general limits to any progress in this area, due to composition (depending on the exact setting) being an NP-hard or even undecidable problem. The future might thus lie in numerous domain-specific techniques where generalization across application domains can only be achieved by setting up an abstract framework for configuration which needs to be instantiated to domains.

3.3 Quality Assurance

Quality assurance refers to achieving high-quality applications, with respect to functional as well as non-functional properties. Three roles are responsible for or may influence the quality of a service composition. The first is the service or component provider, whose interest may, however, just be in achieving a high price for his service/component or high sales figures (Sect. 5). The second role is the broker system. As it has only limited influence on the entities plugged into an application (it selects components/services, but cannot influence their internal operation), it either needs to be supplied with techniques for checking service/component properties or has to rely on user reviews or ratings. The third role is the infrastructure provider responsible for meeting service-level agreements; this is discussed in Sect. 4.

3.3.1 Existing Approaches

Numerous software analysis and testing methods exist for ensuring the safety and security of services and software (see e.g., Dwyer et al. 2007; Orso and Rothermel 2014 for overviews). The drawback of these techniques in an OTF context is the fact that they are typically not aligned to an "on-the-fly" usage. Notable exceptions are specific *certification* techniques that target safety properties (Necula 1997; Shao 2010; Jakobs and Wehrheim 2014; Beyer et al. 2015). Software certification attaches correctness *proofs* to software and builds proof checkers inspecting the validity of proofs. Such approaches already work towards the concept of "on-the-fly" quality checking as proof checking is typically faster than proof construction.

For the second role (the broker relying on user ratings) the key challenge is the fact that applications are rated by users as a whole; in fact, it is the *application execution* which is typically rated. The broker cannot easily see which parts of an application are the cause of a bad or good rating; it might rest in some application component or in an unsuitable infrastructure. This question is known as the *disaggregation* problem (which is also broached in Sect. 5).

3.3.2 Research Required

A truly "on-the-fly" quality assurance requires major improvements in the speed of software analysis techniques, whether they involve certification or not. For certification, more analysis approaches need to be able to construct "proofs" and be supplied with proof validators. Disaggregation techniques for user ratings must be developed and put into practice.

4 IT Infrastructure for OTF

Obviously, an application becomes useful only once it is executed. To execute it, some form of execution machinery is necessary – we summarized this under the umbrella term of *infrastructure*. The question here is to deal with the infrastructure as such as well as with the *mapping* (or *scheduling*) of components to specific instances of the infrastructure for execution.

For infrastructures and for executing components on top of them, OTF computing builds upon well established approaches from (distributed) cloud and grid computing (e.g., virtualization, infrastructure as a service/infrastructure-as-code Morris 2016, scaling in/out as well as up/down, continuous integration, delivery and deployment of microservices-based applications as well as their infrastructure Shahin et al. 2017; Chen 2015). As pointed out above, a significant contribution of OTF Computing is how these microservices are produced and annotated with additional information (about performance aspects, etc.). Leveraging such information for execution of OTF services provides challenges in the same three categories (*Description*, *Configuration*, and *Quality Assurance*) as discussed in Sect. 3. We discuss these challenges in turn.

4.1 Description

4.1.1 Existing Approaches

As already pointed out in Sect. 3, a description of the capabilities of an infrastructure is an essential precondition. Foundations for that are currently being laid by data center description languages such as the Data Center Markup Language (DCML, dcml.org), hopefully resulting in standard interfaces for data centers in the sense of the Software-Defined Data Center concept. Obviously, this goes along with virtualization of the infrastructure, putting data centers under the control of cluster management systems (e.g., Mesos, mesos.apache.org) along with suitable interfaces. The idiosyncrasies of different infrastructure providers – today, usually synonymous with cloud providers – can be hidden by abstract layers such as the one provided by Terraform, terraform.io. Nonetheless, standardized description languages with clear semantics are still largely absent.

More generally, the notion of *infrastructure as code* is currently gaining momentum: It allows to describe an application's specific needs and to provision it on-the-fly, when an application starts up. This includes, for example, provisioning a suitable number of (usually virtual) servers along with a required operating system, library, and network/storage setup.

While infrastructure as code does constitute an excellent basis for OTF Computing, it is nevertheless insufficient. For example, the current focus of data-center-oriented descriptions is understandable given the economic drive of cloud computing and large setups. But it does not do justice to the needs of OTF Computing where we foresee a much wider range and much a richer versatility of infrastructure. In practice, there is currently no consistent approach to describe execution options for a given application that might be run on either a smartphone or a backend system (or where functionality might even be dynamically distributed between these two execution environments). As another example, an application might be runnable on a standard CPU architecture or on a General Purpose Computation on Graphics Processing Unit (GPGPU) or a Field-Programmable Gate Array (FPGA) (with different binaries included in the application description, of course). This requires a description of the corresponding capabilities of real systems, bypassing many virtualization layers that explicitly try to abstract away such differences. Also, for large applications with a geographically distributed user base, a single data center might not be the right solution – distributed cloud computing might rather be preferable. This necessitates knowledge about multiple data centers and their interconnectivity.

4.1.2 *Research Required*

To address these scenarios and needs, our description techniques need to support *infrastructure polymorphism*: vastly heterogeneous and vastly distributed infrastructures. While a lot of such information is available in more or less implicit form (e.g., geographic distribution of data centers), none of that is available in a formalized, machine-readable, standardized way.

This leads to the second issue: Which infrastructures are at hand? Today, the choice where to execute a service is extremely limited as no consistent description is available, nor is there any standardized way of finding such information. We need an *infrastructure discovery* mechanism that can provide opportunities from the small to the very big.

The ensuing question is what to include in these descriptions. Obviously, quantitative descriptions of capabilities are relevant, but also cost information, offered service level agreements (SLAs), etc. Without that, mapping components to infrastructure would boil down to guesswork as is done today when little guidance exists on which cloud provider to use. Ideas in this context are legion, usually tracing back to the rich literature on SLAs, but also on grid computing. OTF Computing goes beyond that by requiring a good understanding of load: Some types of OTF applications will be characterized by large user populations, distributed world-wide, with possibly widely differing requests (think of a next-generation, video-on-demand streaming provider).

To address these needs, existing formalisms [e.g., ETSI's NFV description formalisms (ETSI NFV ISG 2014)] need to be formalized towards *geographically distributed load profiles*, constituting a rich research field, in particular when incorporating time-varying load predictions.

Obviously, all these description approaches need proper standardization and sufficient buy-in to get an OTF ecosystem started. This is discussed in more detail in Sect. 5.

4.2 Configuration

4.2.1 *Existing Approaches*

With proper qualitative and quantitative descriptions of both infrastructure and applications available, the task at hand is configuration. In this context, that means figuring out which particular version of an application (or a service) to deploy, possibly choosing not only which but also which kind of infrastructure, where to serve which particular demands, and how many resources to assign. Typically, this boils down to rather complex optimization problems, often to be solved with real-time constraints or in an online setting.

For relatively simple applications, the closest research fields here are Distributed Cloud Computing and Network Function Virtualization. Both concern themselves with such placement and automatic scaling problems, along with lifecycle management of such composed applications. This work is typically focused on wide-area infrastructures and ignores internal organization of "compute nodes", which might in turn be entire data centers. Inside data centers, on the other hand, very little explicit information about an application is taken into account. A first example in this direction, focused on data-parallel applications such as Map/Reduce jobs, is the notion of a coflow (Chowdhury and Stoica 2012), where flows of a single application are synchronized. However, there is very little support available for scheduling such applications according to complex SLA requirements.

4.2.2 *Research Required*

There is a large range of research that is needed here. Even assuming that all information about infrastructure would be available, there is still a wide range of resulting optimization issues. We do need fast yet reasonably precise heuristics or, ideally, approximations for complex scheduling, scaling, and placement problems, especially across heterogeneous infrastructures for versatile services, capable of running on different execution environments. Inside data centers, this problem is aggravated by the many system layers and the resource competitions among applications; on the other hand, it is simplified by the complete control over all these layers. In wide-area networks, many simplifications that are acceptable in data centers no longer work (e.g., equal delay along all paths), but application structure is typically much simpler than in complex data-parallel applications.

In total, configuring complex applications and a multi-tiered, heterogeneous infrastructure will stay a considerable challenge for the foreseeable future.

4.3 Quality Assurance

4.3.1 Existing Approaches

With respect to execution, OTF Computing shares many of the challenges of generic cloud/grid computing: Policing SLAs, ensuring privacy of execution, and making cost claims transparent are all issues shared across many approaches. The OTF Computing case, however, has some additional challenges: attributing any violations of an SLA to the right instance. In both conventional and OTF Computing, a data center might have failed to live up to its promises, or the software was not up to the task.

4.3.2 Research Required

Quality assurance for distributed computing has a long history. The composed nature of typical OTF software and the role of the broker add another level of complexity here. As the software engineering Sect. 3 already outlined, a mechanism for *disaggregation of responsibility* is needed.

5 Markets for OTF

“A market is a set of humanly devised rules that structure the interaction and exchange of information by self-interested participants in order to carry out exchange transactions at a relatively low cost.” (Gimpel et al. 2008). To refer to the app store example (see 2.1), the app store provider (i.e., the market provider) must define who is allowed to participate, who may offer an app, if there is a commission to be paid to the market provider for every sold app, and so on. A successful marketplace is organized in a way that the marketplace grows and increases the shareholder value of the owner(s) of the marketplace.

Apparently, even if all previously described technological challenges (see Sects. 3 and 4) were mastered, an OTF marketplace’s chances of success are not only determined by technical aspects; they also hinge on these rules from a business perspective and on applying legal constraints. Hence, there is a need to structurally think about and develop such a marketplace, otherwise the likelihood of wasting invested resources is high. This includes, amongst others, setting the right incentives for all the different groups of participants to join and haunt the marketplace as well as aligning these rules on the business layer with the implementation of these rules on the application and infrastructure layer of the marketplace itself. Hence, not only must who may offer something in the marketplace be defined, the actual processes also have to be implemented to materialize this offer in the marketplace – and these implemented processes must run on some suitable infrastructure. OTF Computing poses challenges not just for software engineers or researchers focusing on the necessary infrastructure but also for the agents setting the rules of such a socio-technical system.

This interdisciplinary challenge of designing an electronic marketplace is not new as reflected, for instance, in the *reference model for electronic markets* (see, e.g., Schmid and Lindemann 1998) developed in the 1990s or in contributions in the discipline of *market engineering* (see, e.g., Weinhardt et al. 2003; Neumann 2004; Gimpel et al. 2008) including the efforts of the *grid economics* community (as manifested in the annual GECON conferences since 2004, http://www.gecon-conference.org). In the CATNET project, for instance, an economic self-organization approach for electronic services brokerage, which can be implemented for realizing service markets within service-oriented grid computing infrastructures – markets that are similar to OTF markets – is proposed and analyzed (Eymann et al. 2003, 2007). This approach is a feasible solution to the service allocation problem. Likewise, the importance and impact of electronic markets and electronic marketplaces has been discussed intensely in the information systems literature since the end of the 1980s (see, e.g., Malone et al. 1987; Bakos 1991).

Despite these extensive efforts, there is still a lot to discover (see, e.g., de Reuver et al. 2018 for a research agenda on digital platforms in general). With respect to designing markets for OTF Computing there remain at least three key challenges that relate to the information asymmetry, the multi-sidedness, and the enterprise architecture.

Information asymmetry	How to avoid an OTF marketplace from failing from a business perspective given the manifold information asymmetries present between the many market participants with respect to individual behavior and the actual quality of services and applications?

Multi-sidedness	How to best exploit the strong cross-side network effects in an OTF marketplace? How to overcome the mutual baiting problem?
Architecture	How to best support the build-up of an OTF marketplace spanning all three layers: business, application, and infrastructure? And how to cope with the fundamental dynamics of an OTF marketplace and its necessary iterative development?

We will discuss these challenges in turn.

5.1 Information Asymmetry

Developing a "good" set of rules is not easy, due to the presence of information asymmetries between the involved parties. In comparison to the requester, let alone the user, a broker in an OTF marketplace, for instance, has a lot more information about the available services and their respective quality that may be used to compose an application. Ever since the seminal contribution by Akerlof (1970) it is established in the literature that in the presence of information asymmetry between buyers and sellers, an adverse selection problem may emerge that drives higher quality out of the market and may even lead to a market collapse. The traded objects in an OTF market are unique, user-specific – or, strictly speaking, requester-specific – service compositions. Hence, they are dominated by experience (Nelson 1970) and credence attributes (Darby and Karni 1973). Experience attributes can be known only after using a product/service, while credence attributes cannot be evaluated by a consumer even after consumption but have a perceived value. For decades a doctor's visit has been a typical example of a service that is dominated by credence attributes. Today, many complex digital service compositions share this characteristic, for instance in the area of machine learning. Obviously, this makes a quality inference extremely difficult ex ante for the user. At the same time, the broker and all further participants on the supply side of the marketplace have a hard time figuring out the user's actual willingness to pay if it is not already provided in the request.

5.1.1 Existing Approaches

One promising way to solve this problem of information asymmetry lies in designing incentive compatible mechanisms. In the market engineering literature there exist already a couple of very valuable mechanisms to coordinate distributed activities via multidimensional auctions in so-called *Service Value Networks* (see, e.g., Blau et al. 2009, 2010). Service Value Networks constitute a more general class of markets where complex services – also referred to as *on-demand services* (Blau et al. 2010) – that are composed of single services are traded. An OTF marketplace is special case of such a Service Value Network. Due to the complexity of the composed services and the (potentially) high number of involved parties in composing these services as well as the fact that several brokers may compete for a requester's order, the necessary mechanism may not completely solve the problem of information asymmetry in an OTF marketplace. In addition, it is not clear whether such necessarily complex mechanisms will be accepted by the market participants.

Complementing these already existing auction mechanisms, two very generic means can be used to mitigate information asymmetries (Riley 2001): signaling (e.g., advertising, granting warranties) and screening (e.g., performing market research, imitating reference customers). The literature already provides valuable insights into the mechanisms to mitigate information asymmetries, for instance, by soliciting electronic word-of-mouth communication in the form of online reviews (Burtch et al. 2017) and learning about the product quality from these reviews (Kwark et al. 2014; Zimmermann et al. 2018). The current state of knowledge about online reviews has already been synthesized regarding three aspects: first, the impact of these reviews on economic outcomes such as prices and sales (see, e.g., Cheung and Thadani 2012; Babić Rosario et al. 2016), second, the factors that drive review generation such as reviewing motivation or reviewer self-selection (see, e.g., De Matos and Rossi 2008; Hong et al. 2017), and, third, the moderating effect of review system design (Gutt et al. 2019). Almost all of the extant research in this area is conducted in a "classical" B2C setting with stationary devices being used to write and read reviews (Gutt et al. 2019). The distinct features of OTF marketplaces (i.e., unique service compositions, a vast amount of possible service compositions for a specific request, and low or even negligible marginal costs of producing and distributing a service composition, not necessarily a B2C setting) require further research in this area.

5.1.2 Research Required

Complementing the signaling capacity of software certification (see Sect. 3: Quality Assurance), new efficient and effective signaling and screening mechanisms on the business layer must be developed that mitigate existing information asymmetries on an OTF marketplace. It seems promising to expand research on electronic word-of-mouth communication in general and online reviews in particular, as they have become the de facto standard of reputation systems on virtually any platform market. Important aspects here are to extend the existing knowledge by focusing on the moderating effects of specific design features of reputation systems, varying business environments (e.g., B2B, multi-sided reviewing), diverse devices (e.g., mobile phone) by writers and readers of online ratings (Gutt et al. 2019), and the challenge of disaggregation (see Sect. 3: Quality Assurance).

5.2 Multi-sidedness

Developing a "good" set of rules is not easy, because an OTF marketplace is a multi-sided platform market (Parker et al. 2016). Multi-sided platform markets are economic platforms that have two or more distinct groups of participants (for an illustrative example see Fig. 1) that provide each other with network benefits (referred to as two-sided markets if there are exactly two groups of participants). Network effects can emerge on one side of the platform and across sides. Cross-side network effects give rise to the chicken-and-egg dilemma (also referred to as the mutual baiting problem) of early-stage, multi-sided platform markets (Stummer et al. 2018). This dilemma describes the need for a critical number of participants (e.g., service providers) on one side of the marketplace to attract participants on another side (e.g., requesters); however, service providers will adopt the marketplace and only invest if they observe a sufficient number of requesters on the other side – or at least expect them to join. Once a multi-sided platform market reaches the critical user mass on each side, network externalities stimulate self-reinforced platform growth.

5.2.1 Existing Approaches

Network effects put a much stronger focus on dynamic aspects of economic interactions. Hence, it is not enough to design a "good" set of rules; these rules must be modified on an ongoing basis by the platform owner (in our case the market provider), depending on the market environment. Research has just begun to understand the dynamics and consequences of the presence of strong network effects on aspects such as the price setting, platform growth and competition of multi-sided platform markets. A couple of platform launch strategies have already been proposed (see, e.g., Parker et al. 2016; Stummer et al. 2018; Evans and Schmalensee 2010; Veiga et al. 2017), but these strategies so far focus almost exclusively on two-sided settings. An OTF marketplace, in contrast, may have up to six distinct groups of participants (see Fig. 1).

5.2.2 Research Required

The scope of the analysis of two-sided platform markets has to be extended to cover multi-sided platform markets with three and more distinct groups of participants. Analytical modeling will be one method of choice here and the necessary extensions are straightforward, but the models are getting intractable rather quickly and interpretation of closed-form results is getting increasingly difficult. Simulating the establishment of an OTF marketplace that employs some launching strategy (or a combination of several launching strategies) therefore also seems to be a promising approach as demonstrated, e.g., in Ruutu et al. (2017); Stummer and Haurand (2018).

5.3 Architecture

To build up an OTF marketplace, system architects – the person "responsible for the whole-system view" (Mills 1985) – apply an integrated view spanning three layers: business, application and infrastructure. Ever since the seminal contribution by John Zachman in 1987 (Zachman 1987), enterprise architectures have been in the focus of research and business practice alike. Despite recent advances, comparably little design knowledge is available when it comes to the design of digital marketplaces instead of enterprises – a notable exception is the contributions by the market engineering community. In fact this makes a huge difference, as the focus of enterprises is more on organizing the production of the service or good within the supply chain, while the focus of marketplaces is on facilitating economic interaction between market participants. Hence, the incentives of individuals or firms to participate in the marketplace must be taken into account when designing the marketplace on a much broader scale. In addition, one should note that there are many flavors or variants of an OTF marketplace. An OTF marketplace could take the form of a public marketplace across firms or an "in-house" marketplace within a firm, as a domain-specific (e.g., machine-learning applications, office applications) or a domain-independent marketplace, as a B2B or B2C marketplace, as a marketplace with competing brokers or just one broker, and so on. Consequently, for each variant a specific set of rules as part of the business architecture has to be found that make this OTF marketplace successful. And these rules then must be translated into requirements for the application and the infrastructure layers.

5.3.1 Existing Approaches

Designing an OTF marketplace is a domain-specific engineering activity that is an instance of a more generic activity of market engineering. Market engineering is defined as the use of legal frameworks, economic mechanisms, management science models, and information and communication technologies for the purpose of designing and constructing places where goods and services can be bought and sold and providing services associated with buying and selling (Gimpel et al. 2008). In this discipline a (generic) process that structures the procedure of engineering a market institution has been suggested (see, e.g., Gimpel et al. 2008; Weinhardt et al. 2003; Neumann 2004) that already covers many relevant aspects (including suggested tool support for some activities) for designing an OTF marketplace.

Designing an OTF marketplace also includes designing the business architecture, which involves designing the business strategy, governance, organization, and business processes. When it comes to the governance, organization, and especially business processes, there is both ample modeling support and design knowledge available, including, for instance, reference process libraries (e.g., Becker et al. 2013; Scheer and Nüttgens 2000). Developing a business strategy and – as a pivotal part of that – a viable business model are, however, clearly underdeveloped in terms of design knowledge, process, tool support, and modeling support. A business model describes the mechanisms of how a firm creates, delivers and captures value (Teece 2010), and as such can be understood as a detailed description of a firm's strategy (Casadesus-Masanell and Ricart 2010). Business models are important because firm performance depends not only on the characteristics of the products or services a firm offers, but also on the business model employed for commercializing these products. Akin to process modeling, creating and innovating a business model is a creative and collaborative process. As with many creative processes, the outcome of this process cannot be definitely judged with respect to the quality upfront (i.e., "Will this business model be successful?"); it has to be tested and typically adapted in an iterative fashion (applying, for instance, the so-called "build-measure-learn feedback loop" Ries 2011)[3]. Due to the existing information asymmetries and the multi-sidedness of an OTF market, the iterative and dynamic fashion of an OTF business model development is especially pronounced. Integrated multi-level modeling spanning the business, application and infrastructure layer of an OTF market is already a challenge if the business model is rather stable. The Open Group Architecture Framework (TOGAF) could be used as a starting point for this endeavor as it is the de facto industry standard for designing, planning, implementing, and governing an enterprise information technology architecture.

5.3.2 *Research Required*

A meta-model for OTF markets – or, on an even broader scale, for IT service markets – has to be developed including a domain-specific language, a variability model to account for the different variants of an OTF marketplace, a method to build up an OTF marketplace and a governance framework comprising conformance checks. Such a meta-model would help in designing a specific OTF marketplace much faster and in higher quality. Special emphasis should be put on: (a) the method that should allow for a hypothesis-driven dynamic development of the market and (b) the impact of these dynamics on the application and infrastructure layers, both in terms of modeling (one might even refer to it as on-the-fly modeling) and (on-the-fly) implementation.

6 Conclusions

Even in today's IT world with numerous sophisticated frameworks, libraries and environments, developing and deploying IT applications remains a challenge. This article has advocated OTF Computing as a novel IT ecosystem, foreseeing a great deal of *automization* in configuration and deployment as the key driver towards solving this challenge. This article has furthermore identified the *organization of markets* as the core ingredient of successful OTF Computing, to give incentives to all stakeholders in OTF Computing and to achieve alignment of technical and business needs. Though much research is still required for this vision, we already see aspects of OTF Computing being existent or coming into existence today within specific application areas (see for example https://sfb901.upb.de/poc). This makes us confident that OTF Computing is not a mere research idea – it is a vision that is soon to become reality.

As with all research, this research note has limitations. First, one may argue that the OTF Computing vision comes with a limited novelty and originality as there have been substantial advancements in recent years in most of the concepts (e.g, cloud computing, grid computing, continuous delivery, market engineering, etc.) that form the basis for OTF Computing. We argue that the OTF Computing vision centers on fundamentally interdisciplinary challenges that must be overcome. This results in (a) the need for interdisciplinary collaboration spanning at least

[3] In the same vein (Weinhardt and Gimpel 2007) note for Internet market platforms, such as eBay, Amazon or Google: "after the initial introduction of the electronic market platform, there is no clear cut distinction between design-time and runtime any more. [...] These service operators can continuously experiment with subsets of their user groups [...] and the real-time feedback allows continuous improvement in the design of their online businesses."

computer science, information systems, economics, and business administration, with BISE researchers being in the excellent position of acting as boundary spanners and (b) different research questions and outcomes in comparison to concept- or domain-specific research endeavors. Second, the selection of key domains (software engineering, IT infrastructure, and markets) in general and of key challenges within these domains in particular to structure this research note is to a certain degree subjective. The OTF Computing paradigm is a vast concept that spans several disciplines. In fact, within the scope of one research note it is not possible to deal with – let alone comprehensively – all relevant topics that must be researched for this vision to become true. Hence, we had to make decisions based not only on the existing literature but also on discussions with other scholars and own experience. Obvious candidates for facets that have not been covered extensively here include semantic matching or security. Reasons include, amongst others, already existing contributions such as Mohr et al. (2018); Huma et al. (2015); Petrlic et al. (2014) which already deal with these aspects.

Acknowledgements We thank the German Research Council (DFG) for generously funding our research within the collaborative research center SFB 901 "On-The-Fly Computing – Individualized IT-services in dynamic markets". We also deeply thank all colleagues, research assistants, all students and the administrative staff of the SFB 901 for making OTF "fly".

References

Akerlof GA (1970) The market for "lemons": quality uncertainty and the market mechanism. The Q J Econ 84(3):488–500
Babić Rosario A, Sotgiu F, De Valck K, Bijmolt TH (2016) The effect of electronic word of mouth on sales: a meta-analytic review of platform, product, and metric factors. J Mark Res 53(3):297–318
Bakos JY (1991) A strategic analysis of electronic marketplaces. MIS Q 15(3):295–310
Becker J, Kugeler M, Rosemann M (2013) Process management: a guide for the design of business processes. Springer, Heidelberg
Becker J, vom Brocke J, Heddier M, Seidel S (2015) In search of information systems (grand) challenges. Bus Inf Syst Eng 57(6):377–390
Berardi D, Calvanese D, De Giacomo G, Lenzerini M, Mecella M (2005) Automatic service composition based on behavioral descriptions. Int J Coop Inf Syst 14(4):333–376
Beyer D, Dangl M, Dietsch D, Heizmann M, Stahlbauer A (2015) Witness validation and stepwise testification across software verifiers. In: Nitto ED, Harman M, Heymans P (eds) Proceedings of the 2015 10th joint meeting on foundations of software engineering, ESEC/FSE 2015. ACM, Bergamo, pp 721–733
Blau B, van Dinther C, Conte T, Xu Y, Weinhardt C (2009) How to coordinate value generation in service networks. Bus Inf Syst Eng 1(5):343
Blau B, Conte T, van Dinther C (2010) A multidimensional procurement auction for trading composite services. Electron Commer Res Appl 9(5):460–472
Böhmann T, Leimeister JM, Möslein K (2014) Service systems engineering. Bus Inf Syst Eng 6(2):73–79
Burtch G, Hong Y, Bapna R, Griskevicius V (2017) Stimulating online reviews by combining financial incentives and social norms. Manag Sci 64(5):2065–2082
Casadesus-Masanell R, Ricart JE (2010) From strategy to business models and onto tactics. Long Range Plan 43(2):195–215
Chen L (2015) Continuous delivery: huge benefits, but challenges too. IEEE Softw 32(2):50–54
Cheung CM, Thadani DR (2012) The impact of electronic word-of-mouth communication: a literature analysis and integrative model. Decis Support Syst 54(1):461–470
Chowdhury M, Stoica I (2012) Coflow: a networking abstraction for cluster applications. In: Proceedings of the 11th ACM workshop on hot topics in networks, HotNets-XI. ACM, New York, pp 31–36
Darby MR, Karni E (1973) Free competition and the optimal amount of fraud. J Law Econ 16(1):67–88
De Matos CA, Rossi CAV (2008) Word-of-mouth communications in marketing: a meta-analytic review of the antecedents and moderators. J Acad Mark Sci 36(4):578–596
de Reuver M, Sørensen C, Basole RC (2018) The digital platform: a research agenda. J Inf Technol 33(2):124–135
Dwyer M. B, Robby J. Hatcliff, Pasareanu C. S, Visser W (2007) Formal software analysis emerging trends in software model checking. In: Briand LC, Wolf AL (eds) International conference on software engineering, ISCE 2007, Workshop on the future of software engineering, FOSE 2007, May 23–25, 2007. IEEE Computer Society, Minneapolis, pp 120–136
Erl T (2005) Service-oriented architecture: concepts, technology, and design. Prentice Hall PTR, Upper Saddle River
ETSI NFV ISG (2014) Network functions virtualisation (nfv): management and orchestration. Group Specification ETSI GS NFV-MAN 001 V1.1.1, ETSI
Evans DS, Schmalensee R (2010) Failure to launch: critical mass in platform businesses. Rev Netw Econ. https://doi.org/10.2202/1446-9022.1256
Eymann T, Legner C, Prenzel M, Krcmar H, Müller G, Liggesmeyer P (2015) Addressing grand challenges. Bus Inf Syst Eng 57(6):409–416
Eymann T, Reinicke M, Ardaiz O, Artigas P, de Cerio L. D, Freitag F, Messeguer R, Navarro L, Royo D, Sanjeevan K (2003) Decentralized vs. centralized economic coordination of resource allocation in grids. In: European across grids conference, Springer, pp 9–16
Eymann T, Streitberger W, Hudert S (2007) Catnets–open market approaches for self-organizing grid resource allocation. In: International workshop on grid economics and business models, Springer, pp 176–181
Geierhos M, Bäumer FS (2017) Guesswork? Resolving vagueness in user-generated software requirements. Cambridge Scholars Publishing, Berlin, pp 65–107
Gimpel H, Jennings NR, Kersten GE, Ockenfels A, Weinhardt C (2008) Market engineering: a research agenda. In: Gimpel H et al (ed) Negotiation, auctions, and market engineering, Springer, Heidelberg, pp 1–15
Gruber TR (1993) A translation approach to portable ontology specifications. Knowl Acquis 5(2):199–220
Gutt D, Neumann J, Zimmermann S, Kundisch D, Chen J (2019) Design of review systems: a strategic instrument to shape online reviewing behavior and economic outcomes. J Strateg Inf Syst 28(2):104–117
Happe M, Meyer F, der Heide auf auf, Kling P, Platzner M, Plessl C (2013) On-the-fly computing: a novel paradigm for individualized it services. In: 2013 IEEE 16th international symposium on

object/component/service-oriented real-time distributed computing (ISORC), IEEE, pp 1–10

Hoffmann J, Weber I, Scicluna J, Kaczmarek T, Ankolekar A (2008) Combining scalability and expressivity in the automatic composition of semantic web services. In: Schwabe D, Curbera F, Dantzig P (eds) Proceedings of the eighth international conference on web engineering, ICWE, IEEE Computer Society, pp 98–107

Hong H, Xu D, Wang GA, Fan W (2017) Understanding the determinants of online review helpfulness: a meta-analytic investigation. Decis Support Syst 102:1–11

Huma Z, Gerth C, Engels G (2015) On-the-fly computing: automatic service discovery and composition in heterogeneous domains. Comput Sci Res Dev 30(3–4):333–361

Jakobs M, Wehrheim H (2014) Certification for configurable program analysis. In: Rungta N, Tkachuk O (eds) International symposium on model checking of software, SPIN, ACM, pp 30–39

Kwark Y, Chen J, Raghunathan S (2014) Online product reviews: implications for retailers and competing manufacturers. Inf Syst Res 25(1):93–110

Lamparter S, Ankolekar A, Studer R, Grimm S (2007) Preference-based selection of highly configurable web services. In: Williamson CL, Zurko ME, Patel-Schneider PF, Shenoy PJ (eds) Proceedings of the 16th international conference on world wide web, WWW 2007. ACM, Banff, pp 1013–1022

Malone TW, Yates J, Benjamin RI (1987) Electronic markets and electronic hierarchies. Commun ACM 30(6):484–497

Masche J, Le N-T (2018) A review of technologies for conversational systems. In: Le N-T, van Do T, Nguyen NT, Thie HAL (eds) Advanced computational methods for knowledge engineering: proceedings of the 5th international conference on computer science, applied mathematics and applications, ICCSAMA 2017, Springer, Cham, pp 212–225

Mills JA (1985) A pragmatic view of the system architect. Commun ACM 28(7):708–717

Machine learning library (mllib) guide. https://spark.apache.org/docs/latest/ml-guide.html. Accessed August 2019

Mohr F (2016) Automated software and service composition: a survey and evaluating review. Springer briefs in computer science. Springer, Heidelberg

Mohr F, Wever M, Hüllermeier E (2018) On-the-fly service construction with prototypes. In: 2018 IEEE international conference on services computing (SCC), IEEE, pp 225–232

Morris K (2016) Infrastructure as code: managing servers in the cloud. O'Reilly, Sebastopol

Nair V, Menzies T, Siegmund N, Apel S (2018) Faster discovery of faster system configurations with spectral learning. Autom Softw Eng 25(2):247–277

Necula G. C (1997) Proof-carrying code. In: Lee P, Henglein F, Jones ND (eds) Conference record of POPL'97: the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages, ACM Press, pp 106–119

Nelson P (1970) Information and consumer behavior. J Political Econ 78(2):311–329

Neumann DG (2004) Market engineering: a structured design process for electronic markets. Univ.-Verlag Karlsruhe, Düsseldorf

Oberle D, Lamparter S, Grimm S, Vrandecic D, Staab S, Gangemi A (2006) Towards ontologies for formalizing modularization and communication in large software systems. Appl Ontol 1(2):163–202

Orso A, Rothermel G (2014) Software testing: a research travelogue (2000–2014). In: Herbsleb JD, Dwyer MB (eds) Proceedings of the international conference of software engineering, ACM, pp 117–132

Paolucci M, Kawamura T, Payne TR, Sycara KP (2002) Semantic matching of web services capabilities. In: Horrocks I, Hendler JA (eds) The semantic web - ISWC 2002, first international semantic web conference. Springer, Sardinia, pp 333–347

Parker GG, Van Alstyne MW, Choudary SP (2016) Platform revolution: how networke markets are transforming the economy-and how to make them work for you. Norton, New York

Petrlic R, Jungmann A, Platenius M. C, Schäfer W, Sorge C (2014) Security and privacy challenges in on-the-fly computing. In: Tagungsband der 4. Konferenz Software-Technologien und -Prozesse (STeP 2014), pp 131–142

Platenius MC, Shaker A, Becker M, Hüllermeier E, Schäfer W (2017) Imprecise matching of requirements specifications for software services using fuzzy logic. IEEE Trans Softw Eng 43(8):739–759

Ries E (2011) The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses. Crown, New York

Riley JG (2001) Silver signals: twenty-five years of screening and signaling. J Econ Lit 39(2):432–478

Ruutu S, Casey T, Kotovirta V (2017) Development and competition of digital service platforms: A system dynamics approach. Technol Forecast Soc Change 117:119–130

Scheer A-W, Nüttgens M (2000) ARIS architecture and reference models for business process management. In: van der Aalst W et al (eds) Business process management. Springer, Heidelberg, pp 301–304

Schmid BF, Lindemann MA (1998) Elements of a reference model for electronic markets. In: Proceedings of the thirty-first Hawaii international conference on system sciences, vol 4, IEEE, pp 193–201

Serban IV, Sordoni A, Bengio Y, Courville AC, Pineau J (2016) Building end-to-end dialogue systems using generative hierarchical neural network models. In: Schuurmans D, Wellman MP (eds) 30th conference on artificial intelligence, AAAI, pp 3776–3784

Shadija D, Rezai M, Hill R (2017) Microservices: Granularity vs. performance. In: Companion proceedings of the 10th international conference on utility and cloud computing, UCC '17 Companion. ACM, New York, pp 215–220

Shahin M, Babar MA, Zhu L (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5:3909–3943

Shao Z (2010) Certified software. Commun ACM 53(12):56–66

Sill A (2016) The design and architecture of microservices. IEEE Cloud Comput 3(5):76–80

Stummer C, Haurand M. D (2018) The early-stage development of two-sided digital platforms: a simulation approach. In: Proceedings of the European conference on information systems (ECIS 2018)

Stummer C, Kundisch D, Decker R (2018) Platform launch strategies. Bus Inf Syst Eng 60(2):167–173

Szopinski D, Jazayeri B, Engels G, Kundisch D (2017) On-the-fly computing. Informatik 2017

Teece DJ (2010) Business models, business strategy and innovation. Long Range Plan 43(2):172–194

Thornton C, Hutter F, Hoos H. H, Leyton-Brown K (2013) Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon IS, Koren Y, Ghani R, Senator TE, Bradley P, Parekh R, He J, Grossman RL, Uthurusamy R (eds) The 19th ACM SIGKDD international conference on knowledge discovery and data mining, KDD, ACM, pp 847–855

Veiga A, Weyl EG, White A (2017) Multidimensional platform design. Am Econ Rev 107(5):191–95

Weinhardt C, Gimpel H (2007) Market engineering: an interdisciplinary research challenge. In: Dagstuhl seminar proceedings, Schloss Dagstuhl - Leibniz-Zentrum für Informatik

Weinhardt C, Holtmann C, Neumann D (2003) Market-engineering. Wirtschaftsinformatik 45(6):635–640

Zachman JA (1987) A framework for information systems architecture. IBM Syst J 26(3):276–292

Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65

Zimmermann S, Herrmann P, Kundisch D, Nault BR (2018) Decomposing the variance of consumer ratings and the impact on price and demand. Inf Syst Res 29(4):984–1002