The web interface is becoming the programmatic interface of new-generation applications. This interface can unite the edge, the fog and the cloud seamlessly, permitting applications to be developed, deployed and operated as an orchestration of independent parts that reside and operate wherever they best meet the user requirements.

The continuum of computing: enabling technologies

By TULLIO VARDANEGA

User-oriented and business computing is becoming ubiquitous, and spans from the cloud to the edge via fog nodes in between. The spectrum across such platforms is becoming the natural host environment for an increasing number of value-added applications that can be deployed at various points along it, without necessarily having a single fixed position. Most such applications are independent of one another, often very specialized, sometimes equivalent in function. Transforming their delivery mode into an as-a-service style favours their "servitization", which, by not requiring local installation, alleviates the burden on the end-user platform, thereby increasing their economic value. That transformation is comparatively easy to achieve since the web has become the programmatic interface of new-generation applications. This in turn allows us to regard the spectrum across the cloud and the edge via the fog as the "continuum of computing". That continuum is the perfect habitat for meta-applications that orchestrate selected sets of independent applications into user-defined workflows that single out the service providers via the schema-based interface contracts that they publish; decide on deployment; and obey the execution preferences that are most opportune to meet user requirements. Such preferences also extend quality of service to include privacy, confidentiality, energy, social and ethical concerns.

Key insights

- New-generation browsers, with a thoroughly streamlined protocol stack, will be the operating systems of the future. The web is going to be their API.
- New-generation browsers will allow industrial and consumer applications to run on resource-scarce edge nodes, in addition to more resourceful cloud and fog nodes. This will cause the "continuum of computing" to emerge as a seamless execution platform.
- Value-added meta-services will be realized as user-defined orchestrations of independent, specialized applications running on the continuum.
- Application orchestrators will be programmed with natural interfaces such as use voice command, textual natural-language specifications, and learning-by-example. Orchestration engines will serve the user's best interests in privacy, confidentiality, energy, social and ethical preferences.

Key recommendations

- Support the development of browsers that guarantee efficiency and fitness for direct execution on edge devices, and can sandbox portable value-added applications.
- Support the development of trusted orchestrators, loyal to their users, which can run on any mobile device and appliance.
- Support the development of "natural interfaces" for users to command the orchestrators running on their edge devices.

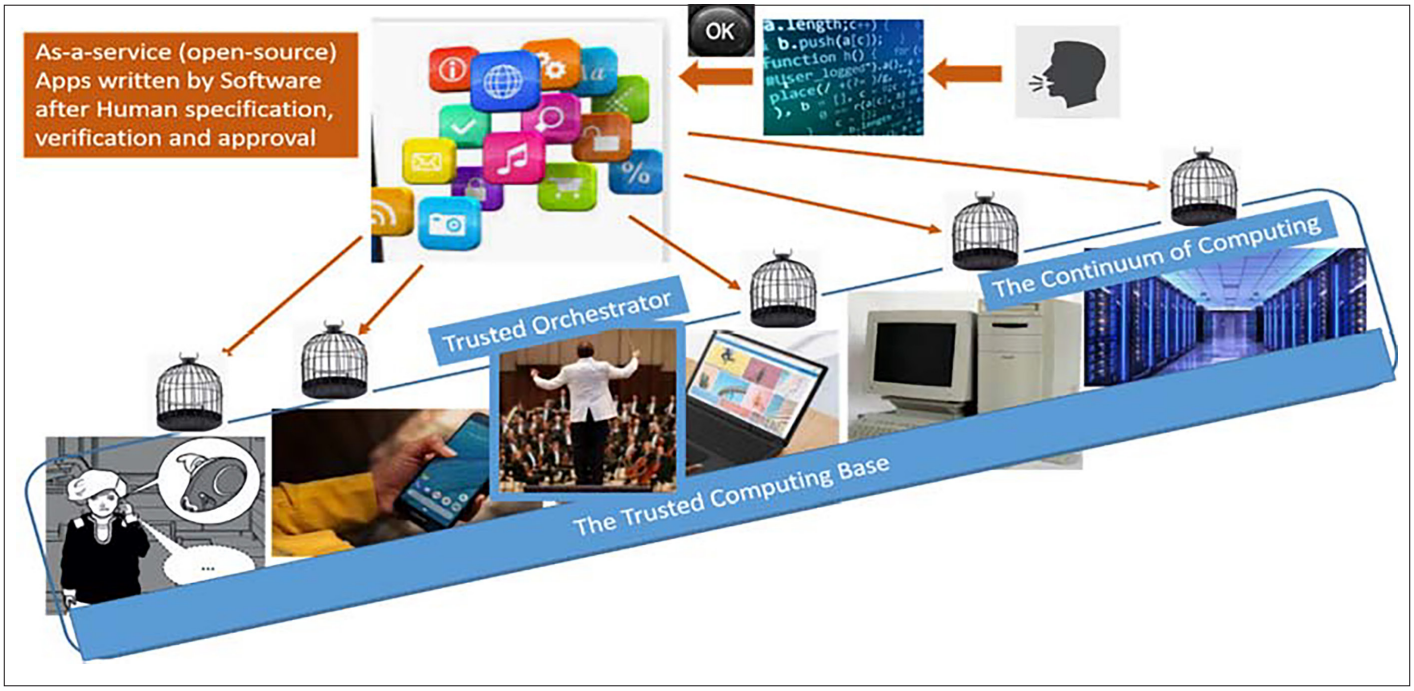

Figure 1: A pictorial view of the "continuum of computing"

Introduction

The notion of "continuum of computing" reflects the observation that edge, fog and cloud computing platforms are being pulled close together into what will likely become a seamless execution environment (depicted in the bottom part of Figure 1). This happens under the push exerted by a massively increasing number of value-added applications targeting mobile, handheld, wearable and unattended devices, which can all run **over one and the same base programmable interface**: the internet. Those applications serve either human users or industrial apparatuses, as in the IoT, with a wealth of functionalities (depicted in the top-center part of Figure 1). This trend is made even more prolific – and the value-added higher – by the fact that most such applications need not be deeply embedded in the target device. This trait accelerates their lifecycle many times over that of traditional device-specific deeply embedded applications.

Whereas this vision may be more immediately associated with the consumer market, it applies equally well to industrial scenarios. Such a notion reflects the opportunity to extend monitoring control of IoT applications to mobile users and the desire to seek low-latency use of mission- or business-critical services (e.g., deep learning) by deploying them at the edge.

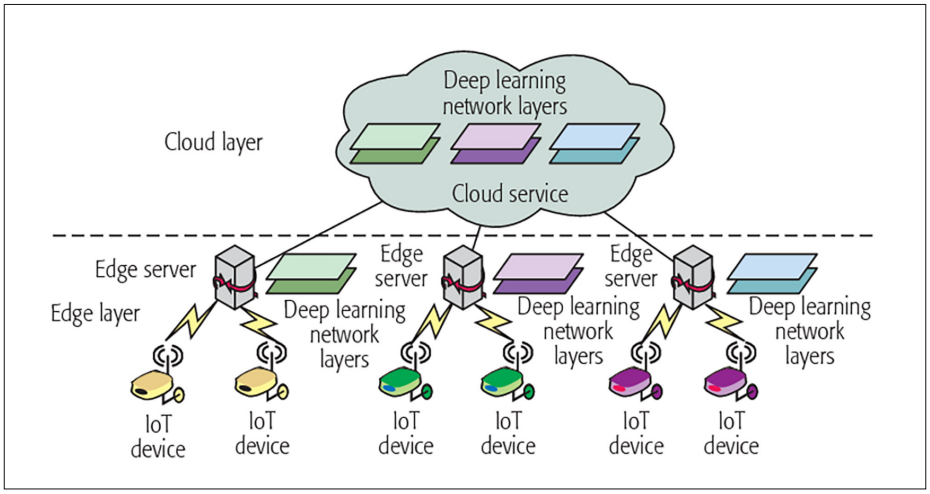

Figure 2: A view of the continuum applied to the IoT (Source: DOI:10.1109/MNET.2018.1700202)

There are two very contrasting possible versions of the continuum. One is captive and vendor-specific, and defers to the big giants of the internet, the only actors in that context that have the technical and financial resources required to cover the whole span of the continuum, horizontally (toward the user and the application developer) and vertically (toward the on-target runtime). The other is open and vendor-neutral; in line with the intended nature of the internet, it allows **interoperation** across all parties. It is this latter version on which we focus.

The intrinsic resource limitations of the near-user hosting devices (battery, storage, bandwidth, processing) require the applications delivering such functionalities to be designed so that, while becoming available on the target device, they should be able to offload the heavy-duty work to "collaborating" computing nodes. These nodes can be located opportunistically anywhere appropriate, from a fog node near the user to the deep centre of the cloud, as long as the user requirements continue to be met. Such a scenario has interesting, interwoven ramifications.

- The parts of the application nearer to the user, on the target device, will have an **as-a-service delivery mode**. The host environment of such a delivery mode can be very different from a normal commodity computer: it requires virtually nothing in the way of the traditional, file-system-based local installation; likewise, it does not need a large proportion of the operating system infrastructure. File systems have no practical value for resource-constrained devices, which can have other ways to arrange isolated storage for executing processes. Similarly, general-purpose operating systems, with their massive overhead and evolutionary clutter, are not very fit for purpose in this scenario.
- The collaboration between local and remote parts of the application, for delocalized data as well as distributed computation, will occur at the highest possible level of abstraction, which means at **the highest levels of the internet protocol stack** (HTTP/3 [1]). That choice reaps the largest benefits in terms of language-neutral expressive power (with architectural approaches that can be resource-centric with REST [2], data-centric with GraphQL [3], collaboration-centric with gRPC [4], real-time streaming with WebRTC [5], etc.), combined with portability and interoperability (stemming from relying on HTTP derivatives, the most standard and ubiquitous of APIs, outside of the boundaries of any given programming language). Moreover, as the endpoints of all such protocols are defined as schema documents, they can be the basis of **contract-based assertions and specifications**, set on the service interface exposed by the individual app parts. This prospect enables the construction of very advanced component aggregations, in which the application provider declares what the app can offer (aka "guarantees" in functional and non-functional terms) and under what conditions (aka "assumptions"). The app user can decide which app to choose from those that provide the same functional API.

The as-a-service delivery mode and the mobility of the computation across the edge-fog-cloud continuum extends the opportunity for deploying new services. The former requires app composition and integration to adopt open and efficient means for service discovery and service registry. The latter requires lightweight containerization, and advanced hosting and deployment engines. Interestingly, **base solutions to most such needs are already in place, for example in Kubernetes' ecosystem, originally born in the cloud but now being "miniaturized" to operate within smaller compute confines. Those solutions need to be evolved towards increased openness, agility and capabilities**.

Application hosting

The scenario described comes with distinct implications for the execution platform that hosts end-user applications. A most natural internet-enabled candidate for host for the on-target part of the application is a web browser, as opposed to a more classic general-purpose operating system, whose excess of evolutionary clutter makes it fatally unfit for purpose [6]. The prime reason for that candidacy is that modern browsers, most notably Chromium and its derivative Chromium OS [7], have learned very well how to:

- Efficiently separate their various activities, of which visual rendering is just one (extraordinarily sophisticated, but progressively less central as natural-interaction comes to the foreground – depicted in the top-right part of Figure 1). For obvious reasons of efficiency, all of this concurrent operation – where not flat-out parallelism – requires **self-served scheduling**, which does not have to be deferred to the untenably costly services of an underlying kernel-space operating system;
- Sandbox the web-hosted apps and plugins that the user may wish to use; realizing such sandboxing requires the host to expose standard APIs (which in turn facilitates the much-desired mobility of hosted apps, and amplifies their economic value) and to self-provide the most efficient form of **application-level containerization** that does not need running directly on a hypervisor;
- Natively **communicate with internet protocols over and above HTTP**, so that service invocation and service composition can all be uniformly expressed in terms of HTTP-based requests, independent of programming languages (hence maximally portable), attached to contract-based interfaces that augment their schema-based endpoint specifications.

Modern web browsers of this kind are the operating systems of the future, especially in the segment of the continuum exposed to the user, because they do everything that really matters to respond to the user application needs, viz., safe componentization (aka sandboxing), scheduling, high-level inter-process communication, and life-cycle management (including installation, update, removal, all without the need for local file systems, but merely using simple storage management).

Interesting advancements toward the secure and efficient deployment of sandboxed mobile code within such browsers are being made [8]**: their base technology needs to be trialled more extensively, hopefully with moonshot projects, to determine how to bring them to maturity.**

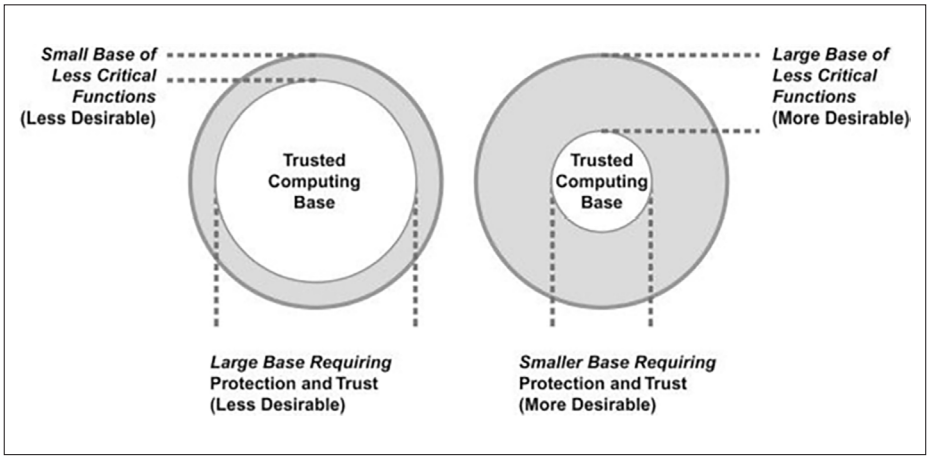

Figure 3: The change of landscape in the software computing stack.

Deployment engine

The observations above posit a radically different landscape in terms of software computing stack from the application to the deployment and execution engine at run time, as Figure 3 attempts to portray. The new landscape (on the right) determined by the previously described scenario will not push out and entirely replace the old, traditional one (seen on the left), but rather will coexist with it. The former will be fitter for the near-to-user part of the continuum, where there is more room for disruptive innovation; the latter will constitute the legacy, closer to the periphery of the cloud, where the ecosystem is more stable and tradition is a helpful convenience. The cloud itself uses a different structure, which employs virtualized or native containerization to modularize the application space, and strives to separate infrastructure management, application management and service delivery.

- The traditional structure on the left has a large base – the operating system, with its own vulnerabilities, and the applications with theirs – requiring protection and needing to be assessed for trust when used in critical settings. This verification is very difficult and costly to achieve, because of the highly-coupled composition of the software stack (e.g. general-purpose operating system with traditional process-based isolation, resident applications that scarcely separate data from code, ad-hoc inter-process communication).
- The wholly novel structure on the right is the opposite: it has a much smaller, leaner, sleeker base that has to be made trusted, which we call here Trusted Computing Base (TCB) and liken to the core of a modern web browser. It also has a large base of less critical functions (sandboxed applications – depicted in Figure 1 as the cage that encapsulates the apps, above the interface to the continuum platform), which can be very user-need-specific, short-lived, and free to fail, incurring local service disruption, but without causing ripple effects.

On its inside, the TCB may be regarded as a message-based micro (but not minimal) kernel that is oriented to supporting web services and apps. All software execution within it obtains sandbox isolation in four ways as described below, without using costly privilege levels:

- The kernel being written in memory-safe programming languages (e.g., Rust [9], Ada SPARK [10]), whose strict memory model uses ownership tags to control the lifecycle of and the access to program data, without requiring garbage collection;
- The extensive use of static verification tools that accompany such languages (rendered unprecedentedly easier to produce by the safeness traits of the language itself);
- The execution by interpretation or just-in-time compilation using languages such as Web Assembly or WASM8. WASM is attractive in this particular context for various reasons. Firstly, it warrants rigid memory separation between executing modules and strict separation between code and data to prevent execution breaches via corrupted data, and it solely allows structured-control-flow instructions, to prevent uncontrolled jumps. Secondly, it defines a Web Assembly System Interface (WASI) to support calls across heterogeneous platforms. Thirdly, it uses a capability-based mechanism to control execution access to external resources;
- The sandboxing of WASM applications within the WASM interpreter, and its hosting in a kernel-level process that acts as resource broker to it. The execution of the hosted (aka target) process follows the principle of least privilege, to reduce the surface of attack to the maximum possible extent, and uses message-based communication, in keeping with the share-nothing principle, prerequisite to robustness, scalability and parallelism at various levels of software system infrastructures.

Two notable conceptual precursors to the TCB described here are especially pertinent. Singularity [11] was established as a proof of concept at Microsoft Research in 2003, built with proprietary technology, and experimentally released between 2007 and 2013, to explore brand-new innovation in the architecture, build and execution of operating systems, and fits the vision we describe in this article. Fuchsia [12] is an open-source project launched by Google, which has not yet reached an official release date but has resonated well with the public given its inspiring principles, which are very much in line with the four points in the list above. **Neither of these technologies constitutes, per se, a self-sufficient stepping stone; they should be seen as**

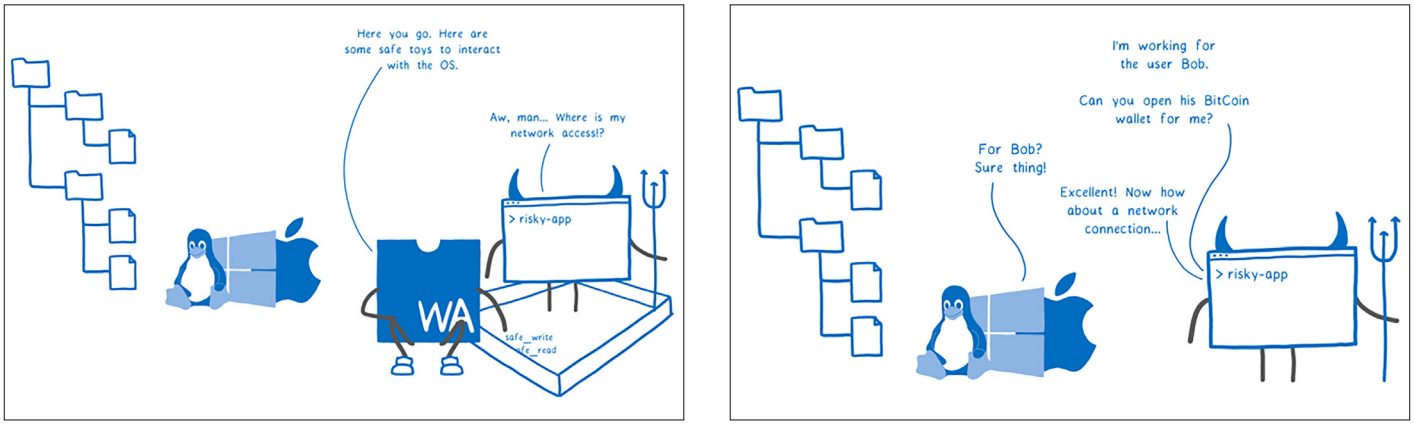

Figure 4: Sandboxing (right) made easy, contrasted with normal hosting (left).
(Source: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/.)

evidence that the required direction can be pursued. Development from scratch of a demonstrator TCB, no larger than a compact and streamlined mini-kernel, would be a useful tool on the path towards the continuum.

Value-added end-user applications: service orchestration

With the large amount of value-added, prevalently independent, individual applications that are candidate for running on the TCB in the novel scenario, the opportunity arises to promote and sustain "non-anticipative evolution" [13] at the runtime platform level. This concept suggests that the complexity of the modern world (in terms of needs, expectations and potential) is better served by allowing adaptive openness that follows second-order user requirements (those that reflect actual user experience as opposed to prescriptive anticipation of it) than by platforms that anticipate them placing arbitrary constraints on the user (be it human or application). One illustration of how non-anticipative value added can be obtained, elegantly and agilely, from the software stack on the right in Figure 3 is the provision of an orchestration engine. This engine would allow individual users (humans or applications) to specify and launch a flow (presumably captured as a directed acyclic graph) that executes selected apps in some defined order and determines how the corresponding output of one should become the input of another.

Such an orchestration engine would have two complementary parts:

One, at the user side, should support "natural interfaces" (e.g. voice-commanded specification – depicted in the top-right part of Figure 1) for the simple yet compelling reason that producing (as opposed to using) such orchestration flows should be within the reach of every individual and not only of trained programmers. This is currently not the case, which is hardly justifiable, either technically or conceptually. An excellent example of this striking deficit is the current radical separation of coding and use of the so-called "skills" of Amazon Alexa (now renamed Echo Dot). With this device, no matter how attractive and satisfactory the deployment of such skills would be to the user, their programming is still a specialized human job, rendered so by an awkward low-level programming interface (to produce very trivial programming directives, in fact). The reason for this is not technical; it is more likely related to market economics (presumably, creating the technology that enabled voice-commanded specification of skills would have delayed the launch of the product. What was – and is – important in the consumer market for a producer is to be first wherever innovation is, which frequently causes corners to be cut in the race) and **should be overcome thanks to the maturation of speech-to-code technology** [14] **or spec-to-code or by-example-code and all possible instances of "natural interfacing" between humans and computers**. These things favour freedom from the obligation of writing arbitrarily idiomatic program text. Such a component of the envisioned orchestration engine should help the user manage the lifecycle of "flows", enabling them to be created both offline (without requiring immediate execution) and online (with immediate execution, perhaps even in place of or in addition to other executions), as well as to be inter-

rupted, modified, resumed or stored. All of that should be equally doable with new-generation natural interfaces (primary) and with more traditional programmatic solutions (secondary, back-office style). **Seen at a wider angle, the envisioned form of application coding, which can be dubbed "software-writing-software", is becoming increasingly attractive and rendered possible – at boosted capacity – by deep learning compute infrastructures. Such infrastructures can be trained to understand an enormous base of example code, and directed to generate code artefacts that have the required traits, functional and non-functional, regardless of the target programming language of choice or necessity**.

The second part, on the execution platform side, is a trusted orchestrator, a prominent element of the envisioned TCB, situated on its upper interface (depicted in Figure 1 as the picture box showing an orchestra director, positioned at the centre of the TCB interface exposed to the user side), which executes service calls directed to independent service apps, aggregated according a user-specified flow. The term "orchestrator" is used here to evoke the conductor of an orchestra that obtains the desired musical effects from commanding into action individual instruments that act independently according to their own music sheet. The particular orchestrator that we envision executes such workflows in a manner that is loyal to the user, for choice of applications (where multiple alternatives can provide the same service), preservation of data privacy (by controlling what data is extracted, directly and indirectly, from the execution and where they are stored), and social or ethical preferences (e.g. open-source versus proprietary; closer versus anywhere; run on green computing versus carbon-based). Workflow execution engines are commonplace in business process management [15]; this vision statement takes them to the next level in two ways: one is making them the prominent way of deploying applications at the outermost level of the software computing stack, recognizing that *digitalization is fundamentally about the composition of third-party functionalities across the network*; the other is adding "loyal intelligence" to their operation, which is one essential ingredient of the "ethical computing" that must accompany the digitalizing-of-everything in order for it to be human-centred. We are seeing the arrival of **programming languages that operate at the level of abstraction as described above (i.e. over the internet, which effectively takes over the role previously played by the middleware), and which are at a stage of their development that allows them still to be augmented with the missing features**. Such languages are often called cloud-native, to signify that their technology contemplates all of the steps entailed in going from traditional compilation and local execution to enabling containerization, deployment in a container orchestration framework, and distributed non-local execution. The most distinctive features of such languages are that they (1) are designed to be integration languages as opposed to system ones; (2) allow for multiple implementations according to the runtime platform that is to host them; (3) support network-friendly types for data, commands and style of interactions; (4) contemplate security abstractions; (5) favour static verification; and (6) enable powerful error treatment at run time. **Example languages that go in this direction include Ballerina** [16] **and Jolie** [17].

Conclusion

The "continuum of computing" is emerging as the confluence of several concurrent phenomena. The most prominent of them is that supporting the web (meaning its internet protocols) as the programmatic interface of modern applications means that it can be so thoroughly streamlined as to become fit even for resource-scarce edge nodes alongside the more resourceful cloud and fog nodes. In that sense, therefore, the continuum of computing emerges as a seamless platform where a multitude of user- and business-oriented web-based applications can be deployed and executed at will. The as-a-service delivery mode of most such applications favours their composition into user-defined orchestrations that operate as value-added meta-services. Such orchestrators may be very attractive vehicles of innovation in at least two respects. Their user side should allow the production of preference-based workflows of service calls with the lowest possible cognitive load for the user, with a shift from a programmatic to a natural-interface model that can use voice command, textual natural-language specifications, and learning-by-example support. Their execution engine should instead animate user-defined workflows singling out service providers via their schema-based interface contracts, deciding on their deployments, and obeying specified preferences that augment quality of service and take into account privacy, confidentiality, energy, social and ethical concerns. All of this is very much within reach from a technology perspective, and holds potential for significant innovation and advancement.

References

[1] Catalin Cimpanu, "Cloudflare, Google Chrome, and Firefox add HTTP/3 support", https://www.zdnet.com/article/cloudflare-google-chrome-and-firefox-add-http3-support/

[2] REpresentational State Transfer, https://restfulapi.net/

[3] GraphQL, https://graphql.org/

[4] gRPC, https://grpc.io/

[5] WebRTC, https://webrtc.org/

[6] "What is Google Chrome OS?", https://www.youtube.com/watch?v=0QRO3gKj3qw

[7] Chromiom, https://www.chromium.org

[8] WebAssembly, https://webassembly.org/

[9] Rust Programming Language, https://www.rust-lang.org/learn

[10] SPARK, https://www.adacore.com/about-spark

[11] Microsoft, 'Singularity', https://www.microsoft.com/en-us/research/project/singularity/

[12] Fuchsia, https://fuchsia.dev/fuchsia-src/concepts

[13] David Weinberger, "Everyday Chaos', https://www.everydaychaosbook.com/"

[14] VoiceCode, https://voicecode.io/

[15] Aleksei Kornev, "Why there are Cloud Functions and Service Mesh & whats next", https://itnext.io/the-concept-of-workflow-engines-c14e8088283

[16] Ballerina, https://ballerina.io/

[17] Jolie, https://www.jolie-lang.org/

Tullio Vardanega is associate professor in the Department of Mathematics of the University of Padua, Italy.

This document is part of the HiPEAC Vision available at hipeac.net/vision.

This is release v.1, January 2021.

Cite as: T. Vardanega. The continuum of computing: enabling technologies. In M. Duranton et al., editors, HiPEAC Vision 2021, pages 56-61, Jan 2021.

DOI: 10.5281/zenodo.4719380

The HiPEAC project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 871174.