Engineering the Transition of Interactive Collaborative Software from Cloud Computing to Edge Computing

GUILLAUME ORTEGAT, Université catholique de Louvain, EPL, Belgium
DONATIEN GROLAUX, ICHEC Brussels Management School & UCLouvain, LouRIM, Belgium
ETIENNE RIVIÈRE, Université catholique de Louvain, ICTeam, Belgium
JEAN VANDERDONCKT, Université catholique de Louvain, LouRIM & ICTeam, Belgium

The "Software as a Service" (SaaS) model of cloud computing popularized online multiuser collaborative software. Two famous examples of this class of software are Office 365 from Microsoft and Google Workspace. Cloud technology removes the need to install and update the software on end users' computers and provides the necessary underlying infrastructure for online collaboration. However, to provide a good end-user experience, cloud services require an infrastructure able to scale up to the task and allow low-latency interactions with a variety of users worldwide. This is a limiting factor for actors that do not possess such infrastructure. Unlike cloud computing which forgets the computational and interactional capabilities of end users' devices, the edge computing paradigm promises to exploit them as much as possible. To investigate the potential of edge computing over cloud computing, this paper presents a method for engineering interactive collaborative software supported by edge devices for the replacement of cloud computing resources. Our method is able to handle user interface aspects such as connection, execution, migration, and disconnection differently depending on the available technology. We exemplify our approach by developing a distributed Pictionary game deployed in two scenarios: a nonshared scenario where each participant interacts only with their own device and a shared scenario where participants also share a common device, including a TV. After a theoretical comparative study of edge *vs.* cloud computing, an experiment compares the two implementations to determine their effect on the end user's perceived experience and latency *vs.* real latency.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing design and evaluation methods**; **Graphical user interfaces**; *Web-based interaction*; *User interface programming*; • **Networks** → **Application layer protocols**; • **Computing methodologies** → **Distributed computing methodologies**; • **Computer systems organization** → **Cloud computing**.

Additional Key Words and Phrases: Cloud computing, Distributed User Interfaces, Edge computing, Peer-to-peer computing, Perceived latency, System latency, Traffic control, User interface migration.

ACM Reference Format:
Guillaume Ortegat, Donatien Grolaux, Etienne Rivière, and Jean Vanderdonckt. 2022. Engineering the Transition of Interactive Collaborative Software from Cloud Computing to Edge Computing. *Proc. ACM Hum.-Comput. Interact.* 6, EICS, Article 160 (June 2022), 31 pages. https://doi.org/10.1145/3532210

Authors' addresses: Guillaume Ortegat, Université catholique de Louvain, EPL, Place Sainte Barbe, 2, Louvain-la-Neuve, 1348, Belgium, guillaume.ortegat@gmail.com; Donatien Grolaux, ICHEC Brussels Management School & UCLouvain, LouRIM, Rue au Bois, 365a, Brussels/Louvain-la-Neuve, 1150/1348, Belgium, donatien.grolaux@{ichec,uclouvain}.be; Etienne Rivière, Université catholique de Louvain, ICTeam, Place Sainte Barbe, 2, Louvain-la-Neuve, 1348, Belgium, etienne.riviere@uclouvain.be; Jean Vanderdonckt, jean.vanderdonckt@uclouvain.be, Université catholique de Louvain, LouRIM & ICTeam, Place des Doyens, 1 & Place du Levant, 3, Louvain-la-Neuve, 1348, Belgium, jean.vanderdonckt@uclouvain.be.

2573-0142/2022/6-ART160 $15.00
https://doi.org/10.1145/3532210

1 INTRODUCTION AND MOTIVATION

Nowadays, *Cloud computing* provides remote services to many of our devices and platforms, such as smartwatches, smartphones, laptops, desktops, tabletops, and smart TVs [55, 58]. Despite its efficiency and continuous hardware and software improvements, some limitations tend to appear, mainly due to the growing diversity of users' devices, and to increasingly striking requirements in terms of interactivity and responsiveness in applications. This shift in requirements is driven in part by the expansion of the Internet of Things (IoT) and of its application domains, *e.g.*, healthcare, transportation, or ambient spaces [57], and by the need for collaboration between users, *e.g.*, in cyber-physical systems, augmented reality, or smart environments.

The availability of cloud computing infrastructure has played a key role in the advent of novel stateful collaborative and interactive applications [39]. In such applications, a constant flow of messages is exchanged between cloud servers and application clients to maintain state persistence and consistency, and to exchange the data structures required by its services [8]. When the volume of this data increases and/or when the distance between the server and its clients increases, the *latency*, *i.e.*, the time elapsed between the instruction executed to transfer this data and the actual moment it is received, also increases. This factor is often neglected [31] but can negatively affect the system response time [50] and the lag perception, especially on mobile platforms [13].

In addition to the latency aspects, we note that interactive applications developed purely in the client-server model of the cloud computing paradigm also have the disadvantage that they are unable to exploit the ecosystem of local devices that the interacting users have access to. First, users may have simultaneous access to multiple devices with different interactional or display capabilities. Second, and perhaps even more interestingly, such devices may be shared by multiple users. Interacting users may be physically close to one another and have mutual or alternate access to these shared devices. In both cases, the computational, interactional, and communication capabilities of these devices, whether they are sophisticated or constrained, are characteristics left unsupported by the pure client-server model of cloud computing [39, 55].

In contrast to the cloud computing paradigm, *edge computing* [42] advocates to migrate the primary hosting task on a local device and to establish Peer-to-Peer (P2P) communications among devices [55]. The edge computing paradigm is increasingly considered to address concerns such as interaction latency, bandwidth costs, security, and privacy [57]. However, this is a double-edged sword: on the one hand, migrating hosting tasks away from the cloud infrastructure can allow serving application clients with lower latency, on the other hand, the performance of the hosting tasks becomes dependent on the configuration of the device they are migrated to. In particular, the benefits of edge computing have never been explicitly investigated for multi-user collaborative applications where the migration is performed onto the end user devices themselves. In contrast to cloud servers whose computational and network performance can be tailored to the needs of the application, migrating to end user devices introduces a factor of uncertainty. Therefore, we do not know whether engineering the GUI (Graphical User Interface) of an interactive application according to the principles of edge computing can be an acceptable option with respect to the experience perceived by end users, or to the ability to seamlessly integrate a diversity of (shared) devices. Likewise, we are not aware of any method for engineering such a GUI according to these principles. Therefore, we want to investigate the following research question:

***RQ**: Can edge computing offer a viable alternative to cloud-based multi-user collaborative applications?*

While we cannot provide a definitive answer to this complex and multi-faceted question, this paper focuses on the preparatory work to answer it. First, we need to determine which method should be used to engineer a GUI according to the edge computing paradigm and to what extent

would it be possible to convert a GUI implemented according to cloud computing principles to use resources at the edge. Second, we need to validate this method on a simple project that focuses on these parts while keeping a functional core that is as simple as possible, and measure the impact on the users' perception of the product. To address this research question and its associated challenges, this paper makes the following contributions:

- A structured method for engineering the transition of a collaborative software from using cloud computing principles to edge computing ones.
- The enactment of this method on Edge Pictionary, an interactive application playing the famous game of Pictionary and its comparison with other implementations.
- An empirical evaluation comparing both cloud and edge implementations to evaluate their impact on the perceived user experience of the software.

The remainder of this paper is structured as follows: Section 3 compares cloud, edge, and related computing paradigms and discusses prior work on engineering GUIs based on their software architecture; Section 4 defines the method for engineering a cloud-supported *vs.* an edge-supported version of an interactive application and applies it to the game of Pictionary; Section 5 presents the results of our user evaluation. Section 6 concludes this paper and discusses future avenues based on the shortcomings observed in the previous section.

2 OPEN SCIENCE

The open source code of the application described in this paper, in both cloud-based and edge-based versions, along with its documentation is released at https://github.com/dewanetout/Pictionary.git. Some excerpt is available in "Supplementary files". To reproduce the different scenarios, the settings of the drawing device and display device must be set by the user. The instruction to launch the different versions is explained in the README file. The traffic controller is available in the tc.js file.

3 RELATED WORK

Since this paper is at the crossroads of cloud *vs.* edge computing, we first revisit their existing definitions for comparison. Then, we characterize the software architecture evolution of an interactive application and its UI towards the full use of edge computing principles.

3.1 Cloud, Edge, and Fog Computing

Cloud computing [2, 30, 55] is today the *de-facto* standard computing paradigm for designing and running the back end of web-based and mobile applications. Requests initiated by application users travel across the Internet to reach the location of application servers (available as virtual machines, containers, or so-called serverless functions [35]) hosted in one or several data center(s). Many authors have noted that the high latency between clients and these data centers can lead to inadequate response time, in particular for interactive applications [52]. Migrating cloud services closer to users is sometimes feasible due to the availability of multiple data centers for a given cloud provider [40, 58]. While some authors claim that low-latency cloud resources are available for most of the world's population [19], this is balanced by the costs of supporting enough instances of an application to serve a geographically spread user base or by the fact that connections remain poor for large regions of the world [21].

The term *Edge computing* is used by many researchers to refer to multiple architectural and software engineering patterns that share the goal of reducing the time and distance travelled by the network traffic between the server and the user's devices, by being aware of the user's location and hosting some or all the computation and control on these devices [27]. A foundation principle of edge computing is that the centralized function of cloud computing is shifted to edge devices [2].

Edge computing hopes to decrease latency of delivery, to increase data processing rates [55], and to support a wide diversity of users and devices.

There exists different options to migrate operations between the cloud and edge devices to support this decentralization: from the cloud to an edge server/device or between two devices/servers themselves. This migration should transfer data and state [63], either by *live migration* to move data from a machine to another while the service is running, or by *handover* when state is sufficiently lightweight [65]. The two main challenges with edge migration are to keep the live functions working and accessible by all users, and to handle shared state consistency [1, 60]. Migration is essential and the needs of a project can change the choice of technology depending on the performance of migration operations [25].

The term *Fog computing* [20] is a distributed computing paradigm that provides cloud-like services to the network edge [57]. It is specifically concerned with the installation of a smaller set of servers (*e.g.*, micro-data centers or "cloudlets" [23]) close to the users, but does not transfer any hosting role to the edge devices themselves. Fog computing is often associated with the advent of the IoT and the processing of the massive amounts of data that connected objects generate [20]. Some authors consider fog computing as a form of edge computing [57] while others make a clear distinction between the two [42]. *Multi-access Edge Computing* (MEC) [54] is a variant of Fog computing focusing on offering cloud-like and network access services to mobile users and appliances (*e.g.*, connected IoT objects) [2, 63].

3.2 Analytical Comparison

Platforms for collaborative applications should be able to scale for millions of concurrent users at the same time. However, the typical workload is split in many independent collaborative sessions between a very limited number of users. Realistically, 2 to 10 users can edit the same document or participate to the same virtual space at a given time. Each independent session can be supported by its own hardware infrastructure; with cloud computing this can be a dedicated server, application container, or set of serverless functions, while with edge computing this can be a piece of software deployed to one of the end user devices.

Globally, the performance of an interactive application depends on (1) the computational capability of the device hosting the backend, and on (2) the performance of the network (latency and bandwidth) between this host and connected clients. While the performance of servers in data centers can be selected according to their needs, in edge computing, one must split the computational task among multiple devices and evaluate the best equipment to run these tasks on available devices, *e.g.*, using a mathematical model [39]. When considering latency, fog computing has an advantage over classical cloud computing hosted in distant data centers, under the assumption that both use dedicated servers making differences in computational latency negligible (and that the necessary state is local to these computations). Edge computing has the *potential* to offer the best latencies for close-by collaborating users by adopting peer-to-peer technology to exchange messages directly between their devices [2], particularly when the UI is demanding [59]. However, in contrast with cloud and fog solutions we have no direct control on the hardware capabilities and configuration of end user device(s) where the host is migrated. For example, migrating to a slow mobile phone connected to an old-generation mobile network may just lower the global performance of the system in comparison to a cloud or fog hosting, and despite the additional latency. In practice, however, modern devices have significant available computing power and good network connectivity. Under such favorable circumstances and with the appropriate selection of the host system, we posit, therefore, that edge hosting is a strong candidate for hosting latency-sensitive collaborative applications.

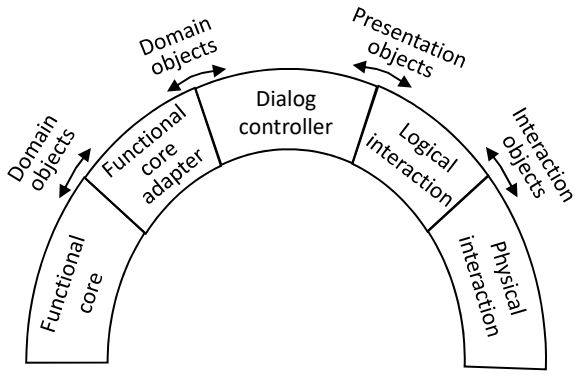


Fig. 1. The Arch metamodel for interactive application architecture.

An interesting aspect of edge computing is its potential for symbiosis with adaptation mechanisms leveraging the diversity, proximity, and co-ownership of devices in the local deployment environment of applications. This comes in contrast with the purely client-server model of cloud computing, whereby two interacting clients sitting in the same room would still interact through applications on their phone, laptop, or tablet sending request response with a distant cloud server unaware of this local context. The hosting of (part of) the application backend logic on local devices in the edge computing paradigm makes it more easily amenable to the implementation of adaptation mechanisms for the application to the local context, such as the availability of multiple devices. Examples of systems assisting with such an adaptation include the Input Adapter Tool (IAT) [16], allowing the adaptation of desktop applications to use alternative interaction devices to the classical keyboard and mouse combo, or programming tools such as UIWear [64] that streamline the offload of parts of a GUI to a secondary device like a wearable or smartwatch and XDStudio [49] that simplifies the construction of multi-device adaptive GUIs. Such adaptation may, furthermore, consider the availability of shared devices, as in the Stanford's interactive workspace project [34].

3.3 Evolution of the Software Architecture

For more than thirty years, the Arch metamodel [6, 7] has become a reference model for describing the software architecture of an interactive application and has been extensively used in many UI architectures [5, 9, 11, 12, 22, 41, 48]. To address the UI independence with respect to the rest of the application, the Arch metamodel is decomposed as follows into five components (Fig. 1):

(1) The Physical interaction handles the interaction with the end user at the lexical level. On the one hand, any action performed by the end user on input devices is sent as an event, such as pressing a push button or issuing a gesture. On the other hand, it ensures rendering the UI depending on the target platform. Building this component is typically addressed by graphical toolkits or UI Management Systems (UIMS).
(2) The Logical interaction handles the interaction with the end user at the syntactical level to ensure application independence with respect to the physical interaction. On the one hand, any information sent by the physical interaction is converted into abstract information, to be exchanged with the next component. On the other hand, the abstract actions are sent to the physical interaction to be enacted.
(3) The Dialog controller ensures sequencing the tasks by mapping the abstract information coming from the interaction to the domain-specific information. It defines how the abstract information (*e.g.* sequences of events) should change the internal state of the application and how the application should react.

(4) The Functional core adapter handles the communication with the application domain to ensure application independence with respect to the functional core. On one hand, it translates calls from the dialog controller into instructions sent to the functional core (*e.g.* call of semantic functions, execution of domain tasks). On the other hand, it translates the callbacks from the functional core into dialog-compliant instructions (*e.g.* result of a semantic function, status of a task).
(5) The Functional core executes the instructions from the functional core adapter in terms of semantic functions (*e.g.* methods) performed on domain objects (*e.g.* execute a search task on a database, calculate a model).

Fig. 2 depicts how the software architecture of interactive applications evolved by instantiating the Arch metamodel for some significant cases. This figure is structured into two columns: *uni-target* when the architecture represents an application for a single user interface at a time, and *multi-target* when the architecture represents an application targeting multiple contexts of use, such as for different devices and platforms [15].

For a *stand-alone application*, such as one running on a desktop, the Arch metamodel is simply instantiated for its five components to one UI (Fig. 2a), while the Physical interaction is decomposed into as many UIs as needed for the multiple targets (Fig. 2b). For example, PlastiXML [18] accommodates different screen resolutions with an Interaction adapter structuring the plasticity domain into zones where adaptation rules are applied to produce UI variations.

For a *cloud-based application*, the server typically covers the left part of the Arch metamodel (represented in brown in Fig. 2c), leaving the physical interaction to take place at the client-side. The client size can vary from a thin client to a thick client. When this application targets many platforms (Fig. 2d), say a tablet, a laptop, and a desktop, a proxy adapter transforms the information from the Dialog controller into meaningful information for a proxy to adapt the UI corresponding to the platform used. For example, Zorrila *et al.* [66, 67] developed an environment for distributing a cloud-based application on multiple platforms. Vandervelpen *et al.* [61] distribute web services on the fly to any platform to obtain a GUI corresponding to the available services.

In the above cases, the resulting UIs are always considered as a single entity. When *Distributed User Interfaces* (DUIs) appeared [26], this condition was no longer satisfied: the resulting UI could be decomposed into pieces that are widespread either on one platform (Fig. 2e) or on several (Fig. 2f). For example, Grolaux *et al.* [29] presented an engine enabling the end user to detach any GUI portion from its main application, to migrate it, and to reattach it when the task is completed. Migratable UIs [28] can migrate from one platform to another at run-time. Frosini *et al.* [24] presented a framework for developing DUIs for uni-target, then for multi-target [25], where the DUI is distributed for multiple users using multiple devices, via a dynamically migrating engine.

This paves the way to introduce *Peer-to-Peer (P2P) distribution* of UIs where the Interaction adapter and the Physical interaction are ensured by the P2P distribution either for a single target (Fig. 2g) or for several (Fig. 2h). For example, Melchior *et al.* [44] developed a toolkit for P2P DUIs based on the concept of distribution graph [46] whose transitions are executed by calling distribution primitives. Manca *et al.* [43] made this distribution dynamically customizable.

The next step in progressively encompassing more Arch components from right to left occurs with *edge computing*: it can encompass components up to the Functional core adapter by introducing separate archs, one for each UI on the same platform (Fig. 2i) or on different platforms (Fig. 2j). For example, Metis [39] allocates tasks coming from the Functional core to end users depending on their role by edge computing. Seo *et al.* [56] modularize a GUI to distribute it based on a module clustering technique borrowed from edge computing. Apart from these works, we are not aware of any method for systematically engineering a GUI according to the principles of

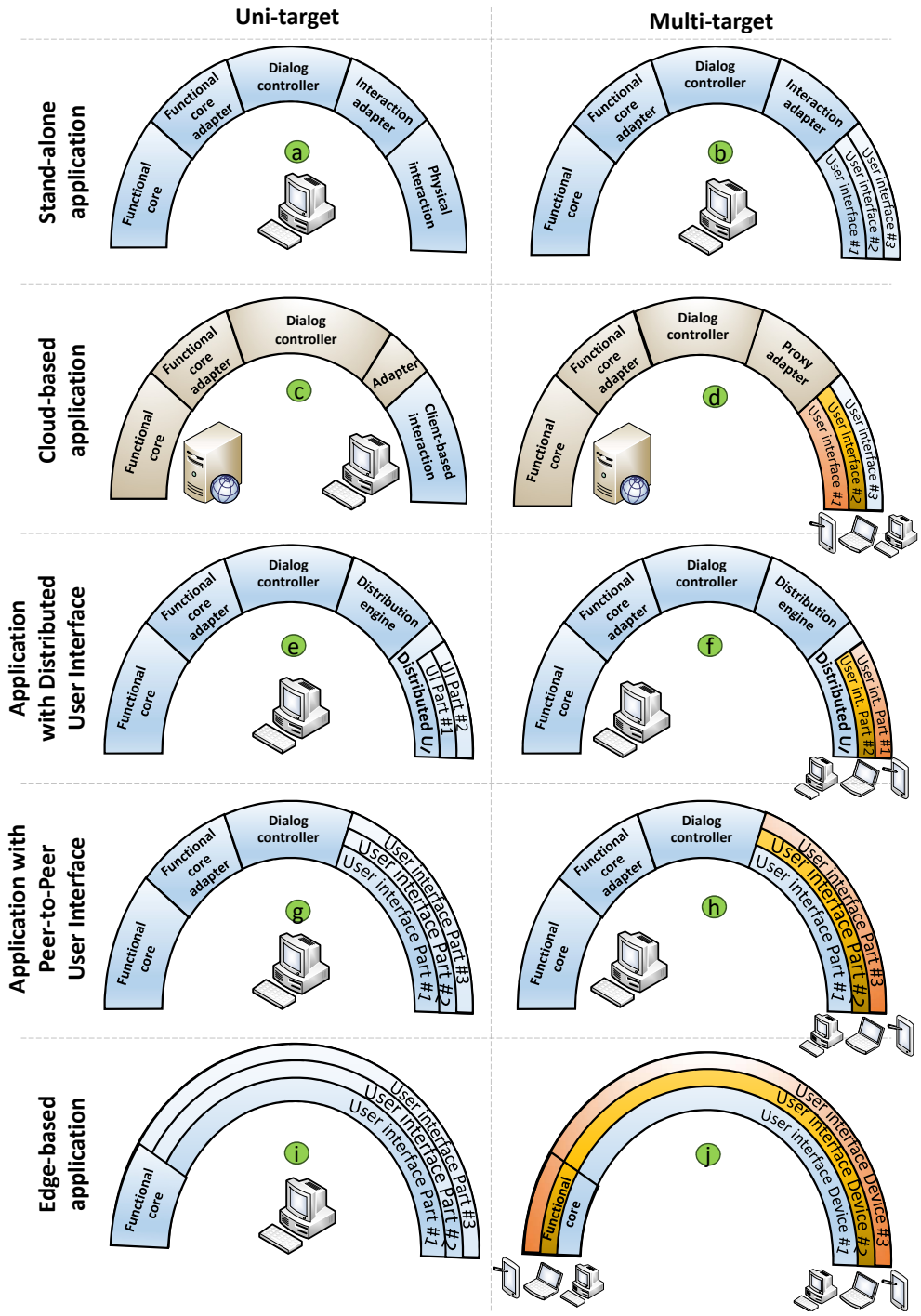


Fig. 2. The Arch metamodel instantiated for various architecture styles.

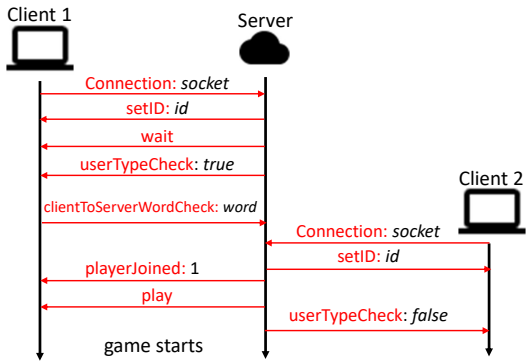


Fig. 3. Connection of two clients to the application. Client 1 is the first connected. The arrow label is structured by its event name in red followed by the variable type of content in black.

edge computing, which is the focus of our research question. More recently, Adaptive Gateways for dIverse muLtiple Environments (AGILE) [62] delivered a modular hardware and software framework for proof-of-concept implementation and rapid prototyping methodologies of interactive applications based on edge computing. While this initiative represents a significant step towards engineering applications, it does not cover the GUI *per se*.

Historically speaking, computer technology started with a host computer holding all computational capabilities with terminals connected to it with limited interaction. When this paradigm was exhausted, a shift towards pushing personal computers was observed where all computational capabilities were sent back to the end users, with little or no connection to a central host. With the advances of Internet-based technologies, some computational capabilities were delegated again to a server, with its culminating point occurring with cloud computing. Again, when this paradigm became exhausted because of too many users, some computational capabilities, including interactional capabilities, were sent back to the clients. Today's paradigm is therefore applicable with edge computing, trying to take benefit of all local devices and computer-based systems. There is a constant back and forth in the distribution of capabilities between a central system and end users.

4 METHOD FOR ENGINEERING A CLOUD- *VS.* EDGE COMPUTING INTERACTIVE APPLICATION

In this section, we describe a method for engineering a cloud computing-based multi-user collaborative application and its transition to an edge computing-based version. To properly validate this method, we apply it to a use case that respects these constraints:

(1) *Be browser based*: as is common for modern collaborative applications, the application UI is browser based, hence runs in JavaScript.
(2) *Be as simple as possible*: the focus of the use case is on the engineering method related to edge computing, and its functional aspects should be kept as simple as possible.
(3) *Satisfy network performance*: the use case should provide useful feedback on this approach for a wide range of network latency.
(4) *Satisfy computational performance*: as said in the state of art, edge network can scale computational tasks by distributing them between the edges. Consequently, our use case does not need to test this dimension, and we can afford light workloads.

Although the performance constraints seem generous, they still work with a wide range of existing collaborative applications. For example, an edge version of Word online could work with different degrees of latency, which would result in different levels of lag between the action of a user and

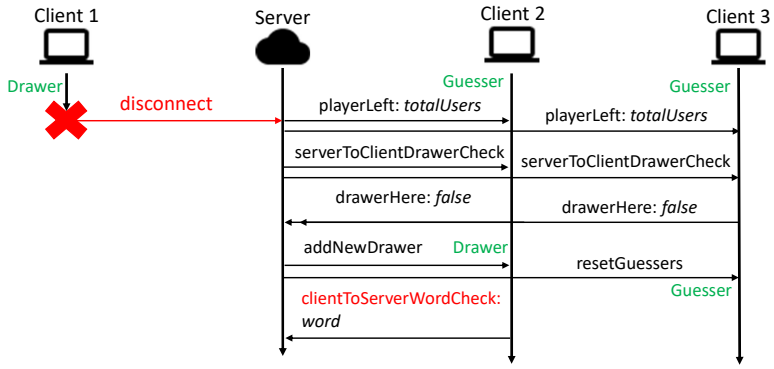


Fig. 4. Disconnection of clients from the application.

when other users see the change, with little impact on the global user experience. For computational aspects, all peers must update their own view of the edited document when they are notified of changes from others. For the edge hosting the server, this task stays the same, and it now must also broadcast the changes to other peers which is negligible in term of computational performance.

To respect these constraints, we selected to illustrate the cloud to edge transition on Pictionary, a famous game that led to several implementations and free applications, in particular in DUIs [45–47]. For this purpose, the rules are adapted as follows: a player can be a drawer or a guesser, the host randomly chooses a drawer that will have a word to draw to make it a guess to the other players, who are guessers. The drawer is the only player allowed to draw on the drawing area. High latency introduces a lag between the drawing and its showing on other devices connected to the game. This is unfortunate but not game breaking. When the latency is low, interesting combinations of device usage become possible: a user can sit on a couch drawing on his mobile phone while watching the television onto which it is reflected. To be comfortable, the delay between the action on the mobile phone and the reaction on the television should be less than 100 ms. Indeed, Beznosyk *et al.*'s studies [10] show that network delays up to 100 ms are acceptable for a collaborative game and that jitter higher than 50 ms introduces some discrepancy between the responses given by impaired and unimpaired players.

4.1 Cloud Application

A standard browser application in a cloud architecture is composed of a server and its clients. The server, located in a data center, manages the hosting part of the application, sends the required files to the client, and exchanges messages to ensure the interaction between users. The client is the device, which exchanges messages with the server. The front end of the application is composed of all the features and functions for user interaction, such as GUI elements and local functions. Typically, clients perform HTTP requests to the server which decodes, processes, and responds to them. Unfortunately, HTTP requests can only be performed between clients and servers and do not allow direct peer to peer communication between clients. The only browser supported protocol that allows direct communication between clients is WebRTC, whose purpose is to support direct video conferencing. However, this protocol is rich enough that it also supports pure data exchange between clients without the need of a video feed. For our purpose, HTTP requests are used for static resources (HTML, CSS, JavaScript files), and for the initial connection of each client. We then rely on the Socket.io[1] library for the rest of the data exchange between the server and the client. This library provides an event-based approach to bidirectional data communication.

[1]socket.io is a library that enables real-time, bidirectional and event-based communication between the browser and the server. See https://socket.io/docs/v3/index.html

The cloud application, to be further adapted to edge computing, should be composed of multiple mechanisms, where a *mechanism* consists of a set of processes to perform a task[42]. Each mechanism is composed of a set of events exchanged and functions applied upon their reception [58]. In our example, the main mechanisms are: drawing, guessing, connecting, and disconnecting.

The connection mechanism (Fig. 3) is different whether the client is the first to be connected or not. In the former, the user has the role of the drawer while in the later, he has the role of a guesser. In both situations, the client uses socket.io to open a socket and send it to the server, which responds with a unique ID assigned to this client. Then, the server assigns its role to the client by means of the event `UserTypeCheck`: **`true`** means a drawer while **`false`** means a guesser. The first connected client is designated as the drawer, who informs the server of the word to guess with the event `clientToServerWordCheck`. The server will send a `wait` event to hold the game until another player connects and makes the server send a `play` event to the first player. The game starts as soon as the second player joins. If the player is not the first connected, the server designates the player as a guesser and send to all other players the number of players in the game. The global structure of the cloud application is depicted in the left part of Fig. 5.

4.2 Edge Application

According to our definition of edge computing, the server parts are migrated to a device of an end user to become aware of the local ecosystem of devices and platforms, to reduce the latency, and to propose multiple models to fit the device characteristics. Edge computing is an alternative to cloud computing but is not server-less, even the hosting part is moved on the end user device. A server still sends the application files to the client and helps clients get connected in a peer-to-peer way. If the context of use becomes more constrained or less favorable, the server can migrate from one device to another one more accommodating. Consequently, the hosting functions can be split to take advantage of the power and diversity of edge devices.

The migration of the hosting part is a central place [65]. For multiple reasons, the hosting parts could be migrated at run-time, thus posing the question of their quality of service [17]. The cloud server functions could be moved to other devices in different ways. The other client parts involved in cloud computing remain present in edge computing. The back end and front end could be grouped on the same user device or split across different devices (Fig. 6a). The global structure of the edge application is described in the right part of Fig. 5.

4.3 Transition from Cloud Application to Edge Application

To ensure a smooth transition from cloud computing to edge computing, we need to migrate the hosting part to a client device in a way that the resulting code of the new application fits the constraints and the requirements imposed by edge computing.

4.3.1 Main Constraints. Since the application is an interactive stateful application used in a browser, several constraints should be satisfied. First, the runtime environment of a server is different from the runtime environment provided by browsers. To achieve migration, the server needs to be able to run in both environments. As browsers natively support JavaScript, using this technology on the server side enables using the same code in both environments. However, browsers do not support directly external libraries as cloud servers would do. For instance, Require and Import commands will not work in a browser and need adaptation. A cloud server uses multiple libraries to implement an HTTP server to send and receive packets. Browsers cannot run a local server and, even if they could, this option would not be efficient. Thus, an alternative must be implemented. Second, the classical scheme of a user typing the URL of an application to connect the device to the application is no longer affordable since URLs dynamically change. Third, the event-based communication

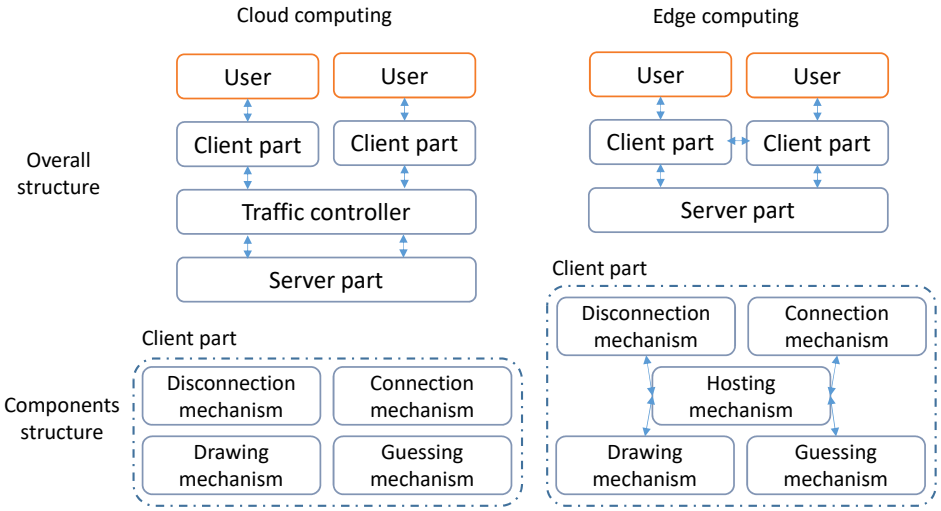


Fig. 5. Structure of the Pictionary application: overall (top) *vs.* components (bottom), cloud-based (left) *vs.* edge-based (right).

architecture is popular by developers trained for event-based interaction styles such as REST. Thus, this should remain consistent in edge computing. The last constraint should preserve the end user's habits: the new behavior in the edge application should be consistent with the one found in cloud computing.

4.3.2 Migration of the Host. The hosting part(s) can be migrated in different ways [63]:

- *Start the hosting part on the edge device*, either on the first connection or on initialization of a new room. The starting state is empty or set to default, the server-side is initiated on the client device by merely warning the user device that the host is already set to no state. This migration is similar to static UI migration.
- *Lively migrate the hosting part from the server to the edge user.* The state variables should be transferred without service interruption and information loss. Information loss could happen if a new information modifying state is sent to the old host and this one has already transferred the states to the new one. Host synchronization ensures a successful migration.
- *Lively migrate the hosting part from an edge device to another one.* Similar to the previous one, but in another configuration.

The two last migration types will be used for our Pictionary application.

4.3.3 Host Function Division. The migration of host functions or services uses different mechanisms [65]. Since edge devices vary in terms of available resources and computational power, a function cannot simply be transferred to any edge device and could be subject to function division. For example, only powerful edge devices can welcome a server and heavy functions should be preferably loaded on non-mobile devices, such as a desktop or a laptop, to preserve battery. For this purpose, we differentiate four divisions of functions (Fig. 6b):

(1) *Split-once*: many devices host each a part of the application. To apply this division, the function should be able to split into several smaller tasks. This division delegates part of the hosting role to many edge devices. To preserve their persistence, the states should be shared among functions. Whether their size permits, they can be concentrated on a single device.
(2) *Split-many*: many devices host each part of the application, and each part can be hosted at the same time on many devices. To apply this division, the application server should be able

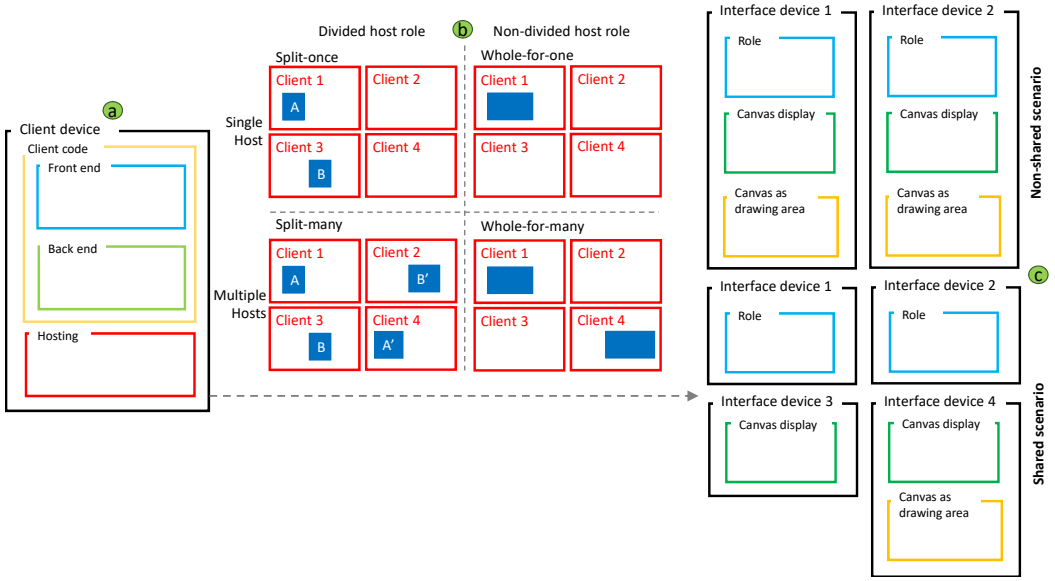


Fig. 6. Transition: (a) initial cloud configuration, (b) division mechanisms, and (c) resulting edge configuration. Client devices are depicted as red rectangles, hosting tasks, as blue rectangles. When the hosting task is split, letters in the blue rectangles represent the different parts. The same letters show the same parts.

to split, such as using multiple rooms of users. The same sub-tasks can be hosted many times on many devices. When the size of the function is large, the state must be shared among devices hosting the state variables. When the size of the function is moderate, using multiple rooms is applicable as it compares to having many independent instances in the split-once division.

(3) *Whole-for-one*: one device hosts the whole application. It consists of not sharing the hosting task and replacing the server with only one edge device. For example, applications with low computing requirements could favor this scheme.

(4) *Whole-for-many*: many devices host the whole application. This division is decomposed into two cases: when the hosting part is hosted by many devices to be fault-tolerant and to ensure the function (again, the states must be shared among the different hosts) or when many application instances are running. For example, the states of each playroom of an on-line game are independent and hosts are independent of each other. The whole application is hosted many times, but the state of each instance does not have to be shared with other instances.

Each application should identify the most relevant split mechanism depending on its function requirements.

4.3.4 Service State Consistency. Stateful applications always rely on state variables. Without them, application failure or interruption will occur. The server used for discovery could be used to save the current state in case of massive failure or disconnection. The quickest solution to recover from a failure consists in sharing the state with all devices, even if they are not hosting. In case of failure, another device can host the service without interruption. For simple applications using the same state variables during a short time, with fault tolerance, the state does not have to be shared with another device than the host. While this solution is simple, it is not protected against failure and the state recovery is harder.

4.3.5 Mechanisms Adapted. All mechanisms of the Pictionary application are structured according to an event-based communication model, which is the typical architecture for multi-threaded dialogue [33]. Fig. 7 illustrates how this event-based communication should be transformed to support the transition from cloud to edge with the classical "Hello World!" example. The blue rectangle holds the server code that is left unaltered. All transformations for event exchange are located in the client code.

The client should host the game and ensure its local behavior at the same time. The event exchange begins with an event sent when the user validates the name entered in an edit box. The first event sent is `Hello` with this name. In the edge implementation, the local behavior and the server behavior should be preserved if the client is hosting the application. The client cannot send an event to itself. Thus, it will directly broadcast the server event to the other clients. If it is hosting the server, the Boolean **`this`**`-client-host` will be set to true. Furthermore, when the client hosting the game receives an event `hello` from another client, the server should broadcast the message to the other clients, but also apply it to its local behavior as if it has received the server event `hello world`. To avoid code duplication, the server behavior and the client behavior code are moved from the event receiver function to an independent function.

The four functions used for the Pictionary, *i.e.*, guessing, connection, disconnection, and drawing (Fig. 5, right) are adapted according to the aforementioned method. Connections and disconnections must have architecture adaptation due to the library constraint. While the Peer-to-peer library is not able to detect the disconnection by itself, the small server used will detect it, ensuring that there is still a hosting device and inform the hosting device of the disconnection. For the connection, we keep the initialization done by the server because the client takes a small time to establish the peer-to-peer connection with other client devices (Fig. 8). The server starts the game initialization and then the hosting device will manage all interactions between clients without intervention of the server. If the client is first connected, the server will ask to host the game. Devices connected in peer-to-peer will be informed of the connection of a device by the event `Goprivate`.

Cloud version
Server code

```
Socket-name = []
Socket.react_to_event('hello', name):
        broadcast('hello world',name)
        socket-name.append(name)
```

Client code

```
Click_on_validate_name():
        sendserver('hello',box.get_text())
        print('you said hello')

Socket.react_to_event('hello world',name):
        print(name + ' say hello')
```

Edge version
Server code

```
Socket-name = []
Socket.react_to_event('hello', name):
        broadcast('hello world',name)
        socket-name.append(name)
```

Client code

```
Socket-name = []

%client part
Click_on_validate_name():
        sendserver('hello',box.get_text())
        print('you said hello')
        server-behaviour()

client-behaviour(name):
        print(name + ' say hello')

%server part
server-behaviour():
        if this-client-host:
                broadcast('hello world',name)
                socket-name.append(name)

Socket.react_to_event('hello', name):
        server-behaviour()
        client-behaviour(name)
```

Fig. 7. Event exchange pseudo code transformation for the "Hello World" application. The cloud computing version is on the left and the edge computing transformed code version is on the right.

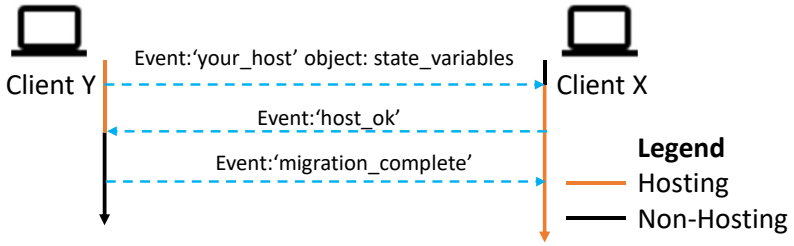


Fig. 8. Edge adapted connection mechanism.

4.3.6 Additional Mechanisms. Our definition of edge computing uses host migration. Migrating stateful applications is a delicate operation. The state must be migrated without information loss or modification of the application behavior from the user's point of view. There are two ways to migrate the hosting parts: migration at initialization time or live migration.

Migration at initialization time is comparable to migrating the hosting part of a static application. The states are not set (or are set to default values) before clients start using the application. The migration does not have to transfer these states. In the Pictionary application, we use this kind of migration when the first client is connected. The server will advertise that he has to host the game by a simple event named `youhost`. The client will start the host with the default value of the states.

The *live migration* is a migration performed when the application is in use. The consistency of the state is important for multiple applications. Multiple state consistency protection can be implemented. All connected devices can keep the state and in case of failure or disconnection of the hosting device, another device can host the game without transfer of the states. Another solution is to save the state on the server used for the connection and on the hosting device. In case of failure of the hosting device, the server can designate a new hosting device by transferring it the state of the application. The Pictionary application being state loss tolerant, we do not implement a state loss protection. Indeed, the game must be interrupted and restarted if a player is disconnected. However, we still implement live migration between two clients, which is an exchange of three events. The first event is named `youhost`. All the state variables are linked to this event. The new hosting client will respond with an event `host_ok` to inform the hosting device that he begins to host the game. The old hosting device will stop hosting the game and inform the new host with an event `migration_complete`.

Due to the shortcomings of the socket.io-p2p library, two protections are added as follows:

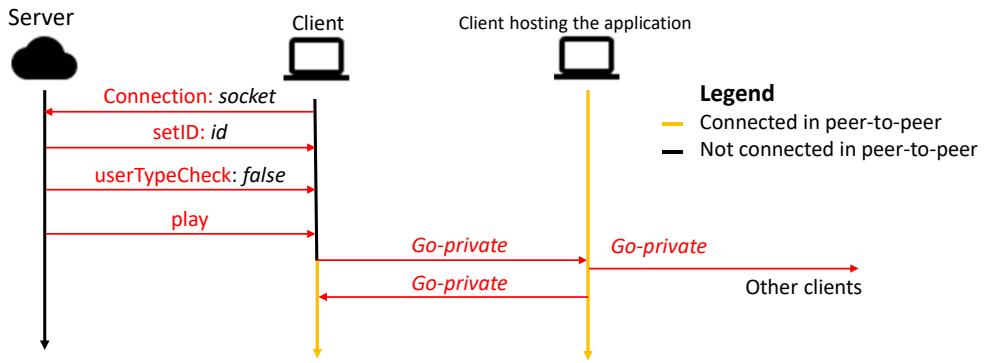
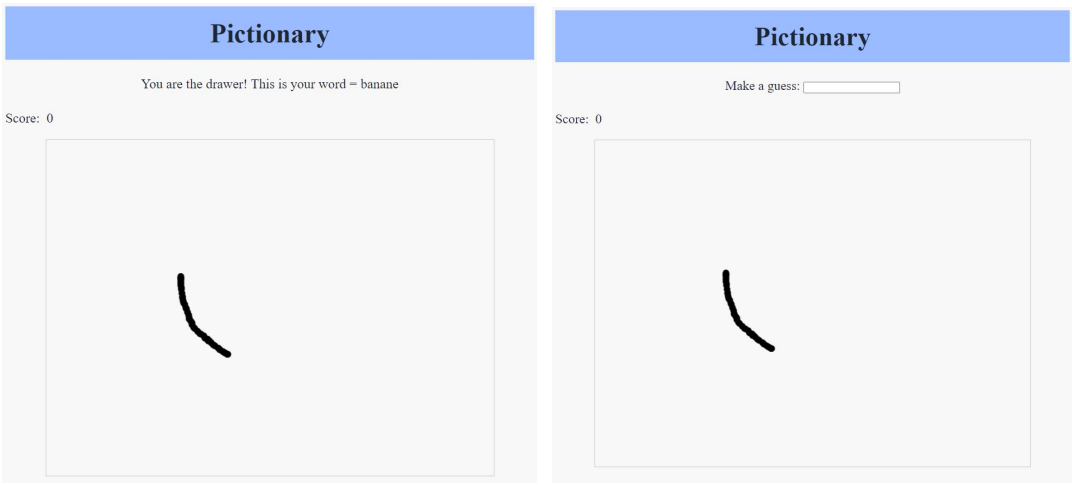


Fig. 9. Illustration of the live migration between two clients.

(1) Sending to the right host is more complicated than with cloud computing. In a peer-to-peer network, the devices are all connected, and they send to all devices. Messages are sent to all users and the structure must be adapted. For all messages which are not broadcast, the receiver must be identified. In our Pictionary, the socket id is used as the device id. The variable or object joined with the event is adapted with this id. If there are no variables joining the event, the id will be placed as the variable. If data is already included within the event, a dictionary is created with two fields, one with the id ("ar") and another with the object (the name can change to make more sense). The server must also be adapted to use the same functions as the client code.
(2) An Idempotency mechanism is needed because the socket.io-p2p library can send the same message multiple times. Since this can impact the behavior of the system, a protection mechanism is added so that applying multiple times the same message is the same as applying it only once, which is useful for at-least-once delivery models. We use an incremental message-id to avoid applying two times the same event content. Each user will keep in memory the last received message-ids and will only apply the received message if the message-id is unregistered.

4.4 Separation of the User Interface

After developing an operational edge application, a UI interface design addressing the separation of concerns is composed of two elements: the role of the device (*i.e.*, the word to guess or a field to input a guess) and the drawing canvas displaying the current drawing. Functionalities or mechanisms can be linked to elements of the graphical user interface. We can split the interface and the functionalities linked to it on multiple devices. We add options and propose two scenarios. They are adapted in two versions, cloud and edge computing. Options allow organizing different blocks of the interface. There are two propositions of possible scenarios.



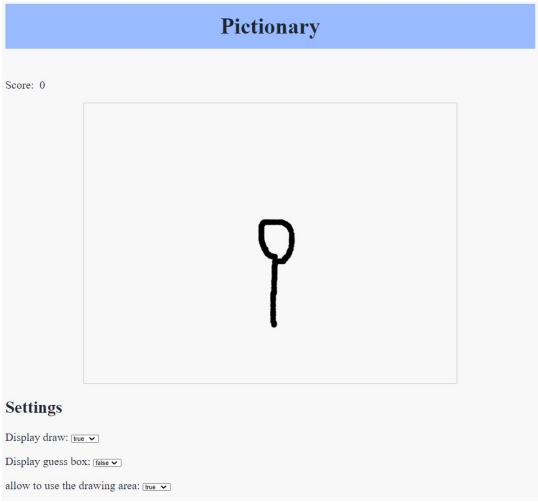
(a) Non-shared scenario drawer interface. (b) Non-shared scenario guesser interface.

Fig. 10. Non-shared scenario interfaces.

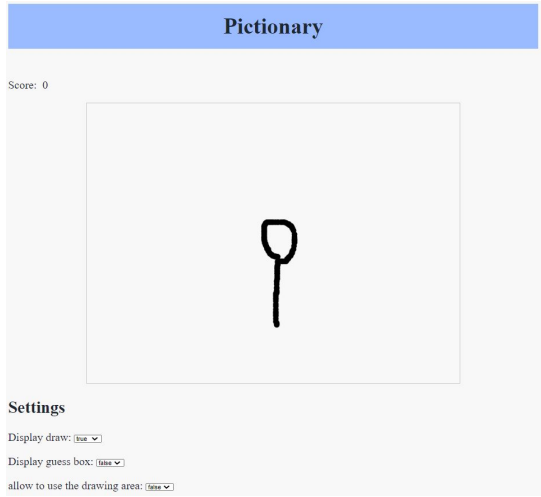
4.4.1 Non-shared Scenario Interface. In this typical scenario proposed by the cloud, all the functionalities are grouped on the client device, which is a straightforward way to play. The interface is not shared between multiple devices. No more modifications are needed. Fig. 10 shows the role of the drawer interface on the left and the guesser interface on the right.

(a) Shared scenario user drawer interface

(b) Shared scenario user guesser interface



(c) Non-shared scenario drawing role interface.



(d) Shared scenario display role interface.

Fig. 11. Interfaces involved in the shared and non-shared scenarios.

For example, let us consider a group of five people located in different rooms in a hotel. They all use a personal device: a smartphone, a laptop, or a tablet. Each device displays the drawing, the word for the person drawing or the guess box for the guessers, and the drawer draws using her own device. They can be in the same hotel hall or in their respective rooms.

4.4.2 Shared Scenario Interface. This scenario is favorable when players are all in the same room. The physical proximity results in the devices being on the same local network, with low latency and huge available bandwidth. The user interface is split into three different interface functions: the user's role, the drawing device, and the display device. The user's role is limited to the area that displays the word or the guess box to enter a proposition. The drawer and guesser roles are illustrated on the top of Fig. 11.

The second function is the drawing device: a touchscreen device, or a device where it is easy to draw such as a tablet or a touchscreen smart television. This role is illustrated on the left bottom of Fig. 11. The third function is the display device. This device will display the drawing for all users. This role is illustrated on the bottom right of Fig. 11. Our group will play this version with user interface separation. The first player starts by connecting a smart television to the game and setting it to display the drawing. The second player connects a tablet to the game and sets it to the draw function. He also connects his mobile phone and sets it to display the first word to draw. The other users connect their mobile phones and set them to display the input field where they can type their guesses for the word. They will all see the drawing on the TV. In particular, the low latency enables a near real-time feedback on the TV for the person drawing on the table. Once a player guesses the current word, his role changes and the tablet is given to him to draw the next word.

5 EVALUATION

In the previous section, two different versions of the Pictionary application were developed, *i.e.*, a cloud computing-based version and an edge computing-based version, both supporting the non-shared scenario and the shared scenario, which is assumed to leverage the low latency and diversity of edge devices. In this chapter, we evaluate these two versions with their respective scenarios to determine their potential effect on the usability and latency perceived by the end user.

For on-line games, two aspects of lag are widely recognized as affecting the end-user experience [51, 53]: latency (a constant delay) and jitter (variations of this delay). Quax *et al.* [53] analyzed these two values depending on the game genre, *i.e.*, for action, puzzle, strategy, and racing games. If the Pictionary is considered as a strategy game, their study reveals that any negative influence caused by the latency perceived by the participants received a lower value than for other genres like action and racing games, thus indicating that such a strategy game is less demanding than action and racing games. Beznosyk *et al.* [10] observed that a game with a delay higher than 100 ms lasted noticeably longer and that a noticeable drop in the game score occurs when the delay exceeds 60 ms. Jörg *et al.* [36] showed that an average delay of 150 ms already affects the end-user experience in several ways, such as in the difficulty to control the game. Normoyle *et al.* [51] found that delays up to 300 ms do not impact the players' experience as long as they remain constant.

Inspired by these studies, our evaluation is aimed at determining the difference between the cloud and edge conditions in terms of perceived vs. real latency.

5.1 Experiment

To determine the similarities and differences between the cloud computing-based version and the edge computing-based version, a controlled experiment was conducted based on two conditions: a control condition represented by the cloud computing-based version and a new treatment represented by the edge computing-based version. We made the following hypotheses:

H_{11} = *The Pictionary application resulting from applying the engineering transition (Section 4.3) has a sufficient usability for the experiment.* The end user does not notice any significant usability difference between the Pictionary application developed for the purpose of testing cloud *vs.* edge computing with respect to interactive applications with average usability.

H_{21} = *The latency perceived by end users in the edge computing-based version is different than the one perceived in the cloud computing-based version.* In the shared scenario of the Pictionary, when there is a wide diversity of devices in terms of system response time, the edge computing-based version offers a superior perceived latency than the cloud computing-based version.

5.1.1 Participants. By word of mouth, we recruited a convenience sampling of 10 participants (2 females, 8 males) aged between 22 and 60 years (M=30.3, SD=13.9) coming from various disciplines and backgrounds: 3 from civil engineering, 2 from computer science, 1 from linguistics, 1 from marketing, 1 from economic sciences, 1 from political sciences, and 1 from history of art. Only one participant did not know the game of Pictionary beforehand while four participants regularly play the game. All participants declared using daily a computer or a smartphone, but less frequently a tablet, except for one participant.

5.1.2 Apparatus. Nowadays, the computational power and capabilities of portable devices like smartphones and tablets are improving so much that they almost match those of desktop computers. However, this is not always the case: device diversity prevails when various end users come with their own devices, exhibiting a wide range of computational power. Device diversity and portability

Device category	Desktop	Tablet	Smartphone
Device	Alienware M15	Apple iPad pro	Xiaomi MI9t
Year of production	2015	2020	2019
Pen based	No	Yes	No
CPU	Intel I7-4720HQ	M1	Qualcomm Adreno 618
Amount of cores	4	8	8
Frequency	2.6 GHz	3.2 GHz	2.2 GHz
Screen resolution	1920 × 1080	2732 × 2048	2340 ×1080
Touchscreen	No	Yes	Yes
Screen size	15.4 inch	12.5 inch	6.39 inch
Keyboard	Yes	No	No

Table 1. Characterization of devices analyzed.

Delay added [ms]	Device latency [ms]
0	8
100	120
150	170

Table 2. Latency measured depending on the delay added.

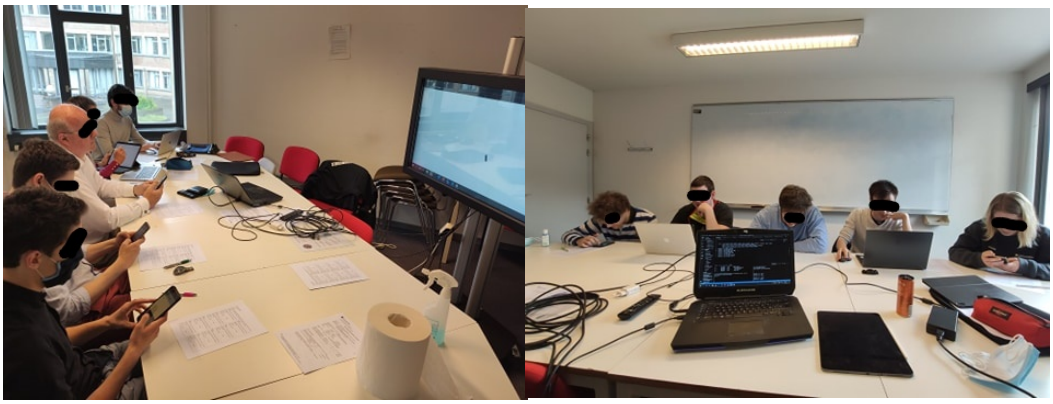
are exactly two factors that are specifically addressed in edge computing. Therefore, to reflect this diversity, we tested several devices brought or mentioned by the participants in three categories, *i.e.*, a smartphone, a tablet, and a desktop computer, to select three cases considered as the most frequently used platforms (Table 1).

These devices were tested to measure their respective latency, defined as the time elapsed between when the drawer starts creating a new symbol and when it is displayed on a guesser canvas. We measured this latency for each device hosting the server role. Without any delay added to the edge version, it is difficult to notice any difference between the two versions since our application is light to host on the three devices selected. The average latency is about a few milliseconds on all devices due to the performance of the application. Adding a delay to the cloud version with a traffic controller results in a latency ranging from 8 ms to 20 ms (Table 2). Consequently, we defined 2 scenarios (S=shared *vs.* N=non-shared) × four possible configurations = 8 conditions (Table 3):

- Cloud computing-based version with no added delay (S0/N0): this configuration represents an ideal cloud-based interactive application with a user connected directly to the server. This configuration is used as a baseline.
- Cloud computing-based version with an added delay of 80 ms (S80/N80): this configuration represents a cloud-based application with good latency.
- Cloud computing-based version with an added delay of 300 ms (S300/N300): this configuration represents an application with a latency for a server far away from the end user.
- Edge computing-based version with no added delay (SE/NE): this configuration represents a typical usage of edge computing.

	Cloud 0 ms	Cloud 80 ms	Cloud 300 ms	Edge 0 ms
Shared scenario	S0	S80	S300	SE
Non-shared scenario	N0	N80	N300	NE

Table 3. The eight conditions of the experiment with their code.



(a) First group of the experiment. (b) Second group of the experiment.

Fig. 12. Setup for the two groups of five participants.

5.1.3 Task and Stimuli. The experiment took place in two separate sessions of about 25 minutes, one session for each group of five members, a constraint induced by the sanitary crisis at the time of the experiment. Each group was invited to play the Pictionary game with the following rules. Each game is played until a player wins. To minimize impatience due to long games, the winning score is set to 2 points. A list of simple words to draw was prepared and randomly sorted. As per the stimuli, we selected simple words which do not require any special understanding or drawing capabilities: car, flower, house, cloud, tree, light bulb, mobile phone, shoes, dishes, foot, balloon, beach, candy, gun, plane, chair, table, shower, bed, money, mouse, knife, boat, heart, square, hand, bird, sheep, elephant, fork, eyes, banana, cow, hammer, turtle, arm, CD, motorbike, lollipop, and sun.

5.1.4 Setup and Procedure. Before the experiment, participants were introduced to the procedure and informed that they could leave the experiment at any time. They were then invited to sign a GDPR-compliant consent form and filled in a demographic questionnaire on their age, highest education level, current occupation, etc. Participants were then invited to enter a quiet meeting room and to take a seat in front of a table where to place their device flat on the desk. To preserve the ecological validity of the experiment, participants were allowed to bring and use their own device: 3 different smartphones and 2 computers per group. They were then aligned along a table in front of a large vertical display (Fig. 12).

The course of the full experiment was explained, and it was only mentioned that the participants will evaluate different versions of the Pictionary game through an application described to them. Participants were then given a few minutes to try the application out and look at the various screens before continuing the experiment. The rest was divided into five tasks:

(1) *Configuration*: the participants configure the application for their device.
(2) *Game*: the participants play 8 games corresponding to the eight conditions summarized in Table 3, *i.e.*, four in the shared scenario and four in the non-shared scenario, corresponding to the eight conditions. These conditions were randomly assigned based on an integer generator making random numbers in the [1..8] interval[2], without participants being notified about which condition was running by using a random generator.
(3) *SUS Questionnaire*: after the games, the participants were asked to fill out the System Usability Scale (SUS) questionnaire [14], which comprises ten questions evaluating the usability of a

[2]See https://www.random.org/integers/

system (in this case, the Pictionary application). Each question is stated on a 5-point Likert scale (1=strongly disagree to 5=strongly agree). Five questions are positively stated while the five others are negatively stated to avoid repeating entries. This questionnaire was selected for its proven reliability [4], for its high correlation with usability ($\alpha = 0.92$) [3], and for its efficient administration.

(4) *Open-Ended Questions, perceived latency, and comments*: after each game, the participants answered two open-ended questions (What are the strengths of this version and what are the aspects that need some improvement?) to provide insights on their overall feeling, free form comment, and the perceived latency of the version (what is the latency of the application on a 7-point rating scale, 1=very small to 7=very long).

(5) *Ranking and Criteria*: after completing all games, participants were asked to rank the versions in decreasing order of perceived usability (from 1=the most usable to 8=the least usable) and to state the three most favorable criteria (ranked from 1=the most important to 3=the least important) and the three least favorable ones.

5.1.5 Quantitative Measurement. While time was not a constraint during each game, participants were timed and their actions were recorded in a log file with a timestamp, along with the real system latency measured for each significant action. This measurement makes it possible to compare the latency perceived by participants with the real latency measured by the system.

5.1.6 Design. Our study was therefore a within-factor design (since all participants evaluated the eight conditions) with one independent variable representing the eight conditions (Table 3), and with five dependent variables:

(1) SUS Question score: integer variable with 5 values representing the individual score assigned by participants to each SUS question (1=lowest value, 5=highest value).
(2) SUS Overall score: real variable representing the overall SUS score computed for all questions [3].
(3) Perceived latency: rating integer variable with 7 values representing the latency perceived by participants (1=slowest, 7=fastest).
(4) Real latency: real variable representing the average system latency measured at run-time for each participant.
(5) Game ranking: ranking integer variable with 8 values representing the subjective order of usability perceived for each condition (1=most preferred, 8=least preferred).

5.2 Results and Discussion

5.2.1 Pictionary Application Usability. We computed d'Agostino-Pearson and Anderson-Darling statistical tests to determine whether the answers follow a normal distribution. Since at least one test failed for each question, we computed a Wilcoxon signed-ranked test for a single sample to determine whether the average answer for each question is significantly above (in case of a positive statement) or below (in case of a negative statement) the median value of 3. The statistics computed on the answers are presented in Table 4 and its distribution in Fig. 13.

The question Q1="I think that I would like to use this system frequently" (M=2.44, SD=1.13) is below the median value of 3, but not significantly, which suggests that participants are less eager to use this application in a frequent way, probably because of the distributed setup.

The question Q2="I found the system unnecessarily complex" (M=1.44, SD=1.01) received a score significantly below 3 (p=.004**) with large effect size (r=0.85) [38], which suggests that participants did not perceived the application as complex to use. This observation was uniform since only one participant scored '4'. This is confirmed by Q3="I thought the system was easy to use" (M=4.00,

Question	M	SD	z-score	p	Sig. $\neq 3$?	r	Interpretation
Q1	2.44	1.13	1.32	0.11	*n.s.*	0.44	
Q2	1.44	1.01	2.56	0.004	**	0.85	Large
Q3	4.00	0.71	2.37	0.008	**	0.79	Large
Q4	3.33	1.87	0.38	0.37	*n.s.*	0.13	
Q5	3.44	1.01	1.17	0.15	*n.s.*	0.39	
Q6	2.33	1.12	1.52	0.06	*n.s.*	0.51	
Q7	4.22	1.09	2.24	0.011	*	0.75	Large
Q8	2.40	0.87	2.14	0.015	*	0.71	Large
Q9	4.11	0.78	2.34	0.008	**	0.78	Large
Q10	1.22	0.44	2.75	0.002	**	0.92	Large

Table 4. Statistics computed for the SUS questions.

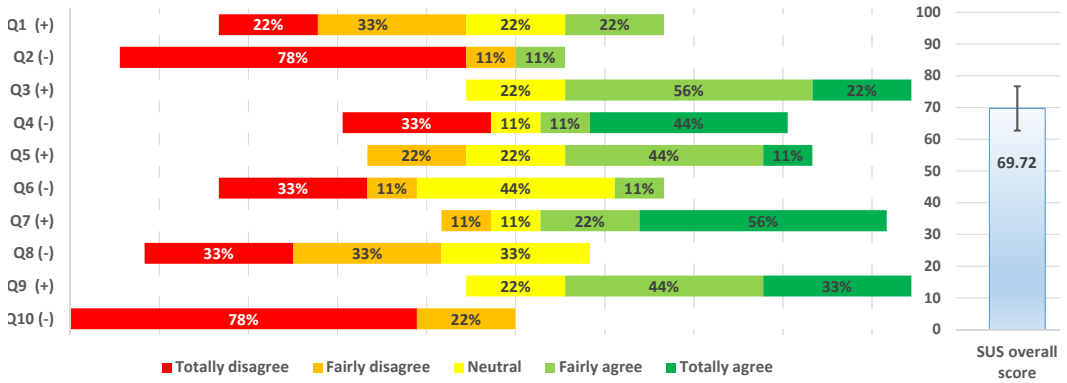


Fig. 13. Distribution of participants' answers to SUS questions and SUS overall score. (+) depicts a positive statement and (-) depicts a negative statement. Error bars show a confidence interval of 95%.

SD=0.71), which is also significantly above the median value (p=.008**) with a large effect size (r=0.79) [38].

The question Q4="I think that I would need the support of a technical person to be able to use this system" (M=3.33, SD=1.87), although being averaged above 3, is not significantly above this value (p=.37, *n.s.*). Similarly, the question Q5="I found the various functions in this system were well integrated" (M=3.44, SD=1.01) is not significantly superior to the median (p=.15, *n.s.*). The question Q6="I thought there was too much inconsistency in this system" (M=2.33, SD=1.12) is not significantly inferior to the median (p=.06, *n.s.*). Yet, this suggests that the various GUI components were perceived throughout the experiment with consistency, one of the most important usability factors. The question Q7="I would imagine that most people would learn to use this system very quickly" (M=4.22, SD=1.09) is significantly superior to the median (p=.011*) with a large effect size (r=0.71) [38]. This question received the highest average among all questions, thus suggesting that learnability of the system was the most appreciated. This is confirmed by the question Q8="I found the system very cumbersome to use" (M=2.00, SD=0.87) is significantly below the median (p=.015*) with a large effect size (r=0.71). The question Q9="I felt very confident using the system" (M=4.11, SD=0.78) is also significantly above the median (p=.008**) with a large effect size (r=0.78). The question Q10="I needed to learn a lot of things before I could get going with this system" (M=1.22, SD=0.44) is highly significantly below the median (p=.004**) with an effect size that is the largest among questions (r=.92). This high correlation also reinforces Q7 about learnability, probably the most appreciated usability factor.

Condition	Perceived lat.		Real latency		Any significant difference between perceived and real latencies?						
	M	*SD*	*M*	*SD*	*MR*	*p*	Sig.?	Cohen's *d*	Glass's Δ	Hedges's *g*	Interpret.
S80	5.90	1.10	138.42	81.59	707.8	≤ 0.0001	****	2.29	1.62	1.67	Large
S0	5.56	1.42	60.86	52.38	441.1	= 0.066	*n.s.*				
N0	4.90	1.97	64.14	38.30	487.3	= 0.0068	**	2.18	1.55	1.59	Large
S300	4.90	1.73	439.13	303.09	1031	≤ 0.0001	****	2.02	1.43	1.45	Large
N80	4.67	1.50	117.01	42.54	716.0	≤ 0.0001	****	3.73	2.64	2.69	Large
N300	4.30	1.16	504.61	543.70	991.2	≤ 0.0001	****	1.30	0.92	0.95	Large
NE	4.22	2.17	12.25	9.41	155.0	≥ 0.99	*n.s.*				
SE	3.40	1.52	30.11	27.77	347.6	≥ 0.99	*n.s.*				

Table 5. Correlation between perceived latency and real latency. Conditions are sorted in decreasing order of their averaged perceived latency (*M*=mean, *SD*=standard deviation, *H*=Kruskall-Wallis mean rank, *p*=p value, Sig?=significance, $p \leq .01^{**}$, $p \leq .001^{***}$, $p \leq .0001^{****}$, *n.s.*=not significant).

The SUS overall score averaged on all participants is 69.72% (*SD*=10.70), which represents a good score that is just above the threshold of 68% (a SUS score above a 68 would be considered above the average score of the 500 studies and anything below 68 is below average[3]). Guttman's $\lambda 2 = 0.92$, which represents a very high score confirming the reliability of the answers. Kendall's coefficient of concordance (W=0.31, $p = .003^{**}$) indicates a fair agreement by participants when appraising the conditions. In conclusion, the hypothesis H_{11} is considered supported. In this way, it is expected that the usability of the tested application should not be considered as a negative factor.

5.2.2 Effect on Perceived vs. Real Latency. In this section, we first examine the two individual latencies separately, then their potential correlation.

Regarding the **perceived latency**, the condition inducing the longest perceived latency is the shared cloud version with a delay of 80 msec (S80: *M*=5.90, *SD*=1.10), followed by the shared cloud version without any delay (S0: *M*=5.56, *SD*=1.42). Surprisingly, the two worst conditions with a delay of 300 ms, *i.e.*, S300 and N300, were not perceived as such since they appear after less constrained conditions (S80, S0, and N0). The two last conditions are the two edge conditions in the non-shared scenario (NE: *M*=4.22, *SD*=2.17) and in the shared scenario (SE: *M*=3.40, *SD*=1.52), respectively. Since the perceived latency does not follow any normal distribution (d'Agostino-Pearson and Anderson-Darling tests both failed), we computed three Kruskall-Wallis statistical tests with Dunn's multiple comparisons (n=8, α=.05). Within the four shared conditions, there was only one significant difference (*MR*=15.50, p=.0203*) between S80 and SE with a small effect size (Cohen's d=.0191, Glass's Δ=.0197, Hedges's g=.0193) [32, 38]. Within the four non-shared conditions, no significant difference was observed (*MR*=1.290, p=.7315, *n.s.*). Within all conditions, no other difference was observed (*MR*=12.78, p=.0795, *n.s.*). In conclusion, these results suggest that the two edge-based conditions have a perceived latency that is better than in the cloud computing-based versions, thus supporting the hypothesis H_{21}.

Regarding the **real latency**, the middle large column of Table 5 shows the average real latency measured by the system throughout the experiment corresponding to the conditions sorted in decreasing order of their perceived latency. As expected, the two slowest conditions are those with a an additional delay of 300 msec., *i.e.*, N300 and S300, respectively. Apparently, the non-shared version N300 was slower than its shared counterpart S300. Then, various conditions all come as related pairs: S80 (*M*=138.42, *SD*=81.59) and N80 (*M*=117.01, *SD*=42.54), S0 (*M*=60.86, *SD*=52.38) and N0 (*M*=64.14, *SD*=38.30), and finally the edge versions SE (*M*=30.11, *SD*=27.77) and NE (*M*=12.25, *SD*=9.41). We observe that the fastest version is the non-shared edge version, which benefits from the lowest averaged latency (*M*=12.25), but also the narrowest standard deviation (*SD*=9.41) among all conditions, which reinforces the position of this edge version as the first one.

Similarly to the perceived latency, we computed a series of Kruskall-Wallis statistical tests with Dunn's multiple comparisons (n=8, α=0.05). Only three comparisons were not statistically different in all 28 possible comparisons: S0 *vs.* N0 (MR=−24.50, p>.05, *n.s.*), S80 *vs.* N80 (MR=34.72, p>.05, *n.s.*), and S300 *vs.* N300 (MR=62.50, p>.05, *n.s.*), which is expected since these conditions are comparable in the two scenarios, shared *vs.* non-shared. The significant difference existing for the 25 remaining comparisons confirms that the real latency was well different.

Consequently, we reported significant differences in the right part of Fig. 14 by scenario only, *i.e.*, within shared and within non-shared, and with the edge version as a baseline, to preserve the legibility of the figure. For example in the shared scenario, the SE condition is significantly inferior to SO (MR=157.7, p=.0271*) with a large effect size (Cohen's d=1.29, Glass's Δ=5.16, Hedges's g=1.90) [32, 38], inferior to S80 (MR=434.6, $p\leq$.0001****) with a large effect size (Cohen's d=2.16, Glass's Δ=13.39, Hedges's g=2.80) and S300 (MR=347.09, $p\leq$.0001****). In the non-shared scenario, the NE condition is always statistically ($p\leq$.0001****) inferior to N0 (MR=337.9, $p\leq$.0001****) with a large effect size (Cohen's d=1.86, Glass's Δ=5.51, Hedges's g=2.46), N80 (MR=321.9), and N300 (MR=450.3) also with a large effect size. These statistical results confirm the prescribed order of conditions depending on the scenario and the added delay, if any: the scenarios with a delay of 300 msec are slower than those with a delay of 80 msec, without any delay, and in the edge version.

To determine whether there is any effect of the scenario on latency, we further computed a series of Kruskall-Wallis tests with Dunn's multiple comparisons by pairs (each shared version *vs.* its corresponding nonshared version). With respect to the real latency, the non-shared version NE is significantly better than its shared counterpart SE (MR=155.7, p=.0002****) with a large effect size (Cohen's d=0.86, Glass's Δ=1.89, Hedges's g=1.01). With respect to the perceived latency, no significant difference was found between the scenarios. These results suggest that, apart for the edge-based version, the scenario did not influence the perceived and real latencies.

Finally, to determine whether participants perceived the same latency as the real one, we computed a last series of Kruskall-Wallis statistical tests with Dunn's multiple comparisons (n=8, α=0.05), which are reported in the rightmost column of Table 2. Five conditions out of eight, *i.e.*, S80, S300, N0, N80, and N300, revealed a significant difference between the latency perceived by participants and the real latency measured by the system. In these cases, these results suggest that the participants did not perceive the real latency and were wrong in their perception. On the contrary, there is no significant difference between the perceived latency and the real one for the three conditions: S0 (probably because this condition represents the typical cloud-based configuration), NE, and SE (which are exactly our two challenging conditions for edge-based computing).

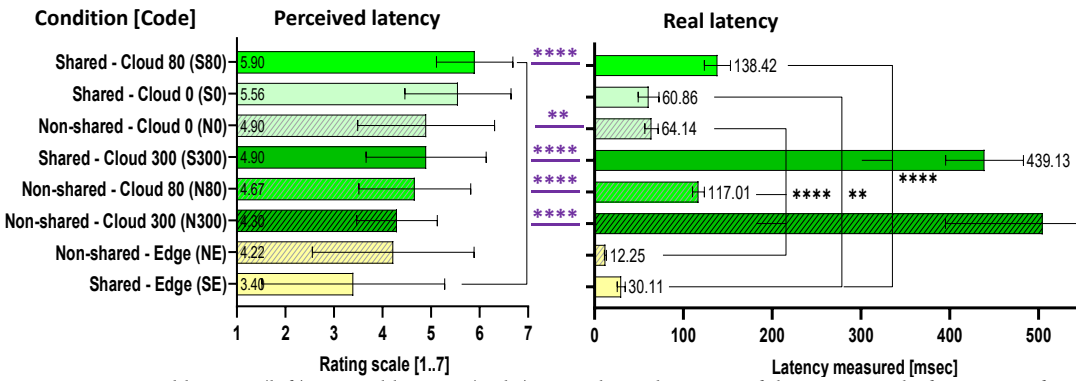


Fig. 14. Perceived latency (left) *vs.* Real latency (right). Error bars show a confidence interval of 95%. Significance level is indicated as follows: $p\leq$.05*, $p\leq$.01**, $p\leq$.0001****.

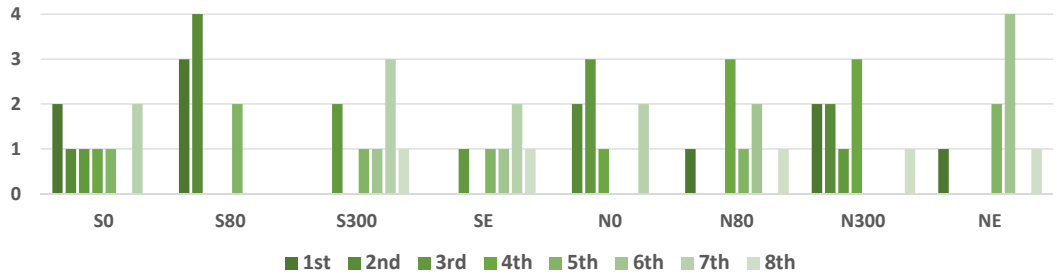


Fig. 15. Distribution of conditions ranking.

5.3 Games Ranking

In this section, we removed one outlier out of the ten participants as he expressed his preference not as a ranking of conditions but as a preference for the nonshared scenario over the shared scenario. Fig. 15 depicts the distribution of all ranks expressed by participants for the eight conditions. Fig. 16 compares these distributions to identify for each place (ranging from the first place to the eighth place) which condition is expressed as the best ranked. The S80 condition is ranked the most frequently for the first, second, and fifth places, thus representing the condition that is the best placed overall. The N0 condition is ranked the most frequently for the third place, while N80 and N300 are *ex aequo* for the fourth place. The NE condition is ranked the most frequently for the sixth place and the S300 condition for the seventh place. For the last place, most conditions received the same low ranking, thus suggesting that they did not identify any preference for this place. Overall, participants reported that this ranking was very challenging to them as they did not identify any particular strength or weakness of a condition over another one, apart from the perceived latency that was discussed in the previous section.

We believe that the classification done by participants, which is very varying, cannot express a precise representative preference. Due to the small population sampling (n=9), statistical tests were not significant for the ranking of the games. However, the notes and remarks expressed by the participants help to precise some aspects. One of the recurring notes was that they experienced difficulties in differentiating the external aspects of the games. This could suggest that the Pictionary application did not modify their user habits. They were not able to see the difference between the edge and cloud versions.

5.4 Informal Comments and Observations

Scenarios and Devices. We tested two scenarios. To induce some device diversity, each participant brought their own device. We can distinguish two main categories: mobile platforms (such as smartphones and tablets) and stationary platforms (such as laptops and desktops). We found some correlation between the favorite scenario and the device category: 83.3% of the mobile participants prefer the shared scenario that leverages the low latency of the network. All participants prefer the non-shared scenario in general when no constraint is imposed on the network. The GUI is considered as a benefit for the device diversity and its characteristics. By relying on the devices participants were familiar with, it was expected to minimize the learning curve induced by manipulating a new, potentially unknown, device, although belonging to the same family. We did not investigate whether this device familiarity could influence their preferences and their perceived latencies, but we investigated whether the device frequency of usage could influence these preferences and latencies. For this purpose, we ran a series of two-tailed Spearman rank tests (Table 6) with a 95% confidence interval (α=0.05), but we found only one negative correlation between the tablet

Fig. 16. Best ranked conditions per place (from 1st to 8th place).

Freq./Cond.	S0	S80	S300	SE	N0	N80	N300	NE
Computer	-0.64 (0.06)	0.11 (0.77)	-0.32 (0.37)	-0.18 (0.62)	-0.62 (0.06)	-0.54 (0.13)	-0.13 (0.71)	-0.57 (0.11)
Smartphone	-0.40 (0.29)	-0.33 (0.36)	0.38 (0.27)	0.12 (0.75)	-0.34 (0.34)	0.03 (0.95)	-0.52 (0.13)	-0.40 (0.29)
Tablet	0.19 (0.63)	0.18 (0.63)	-0.12 (0.73)	0.29 (0.42)	-0.45 (0.20)	-0.67 (0.048*)	-0.04 (0.92)	-0.41 (0.27)

Table 6. Spearman rank correlation test (two-tailed, α=0.05) between device frequency of usage and condition. Conditions are sorted in appearing order: Spearman ρ (p value, $p \leq .05^{*}$).

frequency of usage and the Non-shared - Cloud 80 (N80) condition. In general, our participants had a very low frequency of tablet usage (M=1.9, SD=1.76) compared to computer frequency (M=6.8, SD = 0.40) and smartphone frequency (M=6.3, SD=1.42), which could suggest that their very low frequency of usage is inversely correlated with the most traditional condition.

Impact of Server Disconnection. We finished a game with an edge computing-based version and we shut down the server. At that time, participants reported that they even did not notice it and

were still playing. In a discussion with them after the game, all participants confessed that they did not notice that the server was disconnected. This suggests confirming an advantage of edge computing over the cloud computing paradigm. The edge version works well in case of server disconnection and it is not noticed by end users.

Fun with the Game. Multiple players seem to enjoy playing and sometimes winning Pictionary. Some of them were focused on victory and said that they enjoy playing.

5.5 Limitations and Perspectives

This experiment shows that a collaborative multi-user application created under our design constraints is a viable solution with respect to the end user experience. This section explores the limitations of these constraints and paths for further research.

First, we avoided the problem of the long-term persistence of information. Typically, collaborative applications are tied to online storage solutions, for example, One Drive for Office 365 or Google Drive for Google Workspace. A shared online storage drive cannot reasonably be migrated to an edge device. However, the collaborative edition of a single document can still benefit from an edge solution by having the edge server act as a proxy to the persistence of the document. In that sense, the cloud application is partially migrated to an edge, where the collaboration tasks are migrated while the persistence tasks are not.

Second, the computational workload can be an issue. Cloud applications commonly use content delivery networks scaled to their needs. When using the random devices of the connected users, there is no performance guarantee. If the responsibility of the cloud application is mainly to ensure the proper exchange of messages between the clients (a broadcaster), we assume any modern device connected to the Internet is powerful enough for this task. For more computationally intensive workloads, solutions must be worked out. A broad strategy could be to perform a rough benchmark of an edge node before accepting it as a migration target. The same rough benchmark applied to the cloud server serves as a baseline and only edge nodes with performance close enough or higher than the baseline can be migrated to. Another strategy could be used for applications where the workload can be distributed among peers. By applying a partial migration strategy and splitting the workload among several edge devices, it becomes possible to gather their computational workload and even achieve better performance than cloud solutions.

Third, the network bandwidth can be an issue. The bandwidth available to a mobile phone on GPRS is very different than the same mobile phone on 4G, which is also very different than the bandwidth of a cloud server. On the other hand, applications that are naturally used by people in close proximity, like the Pictionary game, benefit greatly by having all participants on the same local network. For other cases, solutions should be worked out on a use case basis. For applications where high latency is acceptable, a broad solution is to create a p2p mesh between the edge devices for efficient broadcast [37]. By splitting the broadcasting of messages between several edge devices, their bandwidth is compounded, at the cost of greater latency.

Fourth, the network latency can itself be an issue. This is close to the network bandwidth problem but can occur independently of it, *e.g.* a satellite connection with large bandwidth but poor latency. A fast action-oriented game may require latency as close to real time as possible. However, many applications can tolerate a bit of latency with little real-world consequences. For example, players of the Pictionary game could suffer a one-second delay over other players and not notice the delay because they are in different locations. Once again, migrations that bring all peers to the same local network yield the best results. In the general case, the latency of edge devices cannot be lowered, and the best we can do is to ensure the migration occurs only on edge devices with good enough latency.

In conclusion of this discussion, each cloud application is impacted differently depending on these four constraints. The Pictionary application is a best-case example: no persistence is required, the computational workload is very low, bandwidth and latency requirements are low, and the game is often played with all participants on the same network anyway. Other types of cloud applications should be evaluated before deciding to introduce the migration mechanism. There is little sense in attempting to migrate the persistence functions of cloud applications as they are intrinsically tied to physical devices like hard drives. Similarly, the migration of computation heavy functions should probably be avoided. As for network requirements, heavy bandwidth and low latency requirements should probably be avoided, unless the application naturally invites all participants to sit on the same network. This is the case for a party game like Pictionary, but also the case for many collaborative work applications where participants are on the same company network. When the (computational and network) performance is critical, mechanisms should be put in place to ensure the migration only occurs on edge devices powerful enough. Moreover, the performance capability may evolve over time: a mobile phone may switch from a fast WIFI to a slow GPRS connection, another CPU intensive process may suddenly run, and so forth. The mechanism should ensure to regularly check that the expected level of performance is maintained.

Despite these limitations, for the right type of applications, edge migration brings a large scalability potential over cloud applications. As illustrated by the shutting down of the server during one of the Pictionary sessions, once a game is set up in the edge scenario, the cloud server is not needed at all anymore. Consequently, a single small cloud server can potentially host an infinite number of games. This is a huge advantage over typical cloud solutions.

6 CONCLUSION

In this paper, we showed that the pictionary interactive application developed for testing edge computing is not less usable than average interactive applications (SUS OVERALL SCORE=69.72), with a fair agreement among participants and a very good reliability among them. The two edge computing-based versions, with shared scenario and non-shared scenarios, were subjectively perceived as the versions benefiting from the best latencies among all conditions, including all cloud computing-based versions with or without delay. These two perceived latencies did not reveal any significant difference with the real latencies measured by the system throughout the experiment. This was also the case for the cloud computing-based version with the shared scenario without any delay. In this evaluation, we designed an experiment controlling a maximum of variables while keeping a realistic scenario to preserve ecological validity. The result analysis shows that new scenarios leveraging the low latency of edge computing to detach part of GUI are favorable to the device diversity and their varying characteristics. The assumptions made at the beginning of this paper are validated.

Users show difficulty to differentiate if the application is using cloud or edge computing. They favor graphical user interface splitting on some, but not all, device types. Edge computing is a valid alternative to cloud computing as it effectively addresses many of the the issues usually linked to cloud computing (such as latency, high network usage, and scalability). The latency allows to detach graphical interface parts, which helps leveraging the characteristics of each device.

ACKNOWLEDGMENTS

The authors of this paper are grateful to the anonymous EICS reviewers, especially the expert in latency, and the associate chairs for their constructive comments on earlier versions of this manuscript. They also thank the participants of the experiment reported in the paper for their involvement.

REFERENCES

[1] Anshul Ahuja, Geetesh Gupta, and Subhajit Sidhanta. 2019. Edge Applications: Just Right Consistency. In *Proc. of the 38th IEEE Symposium on Reliable Distributed Systems* (Lyon, France) *(SRDS '19)*. IEEE, 351–3512. https://doi.org/10.1109/SRDS47363.2019.00047

[2] Yuan Ai, Mugen Peng, and Kecheng Zhang. 2018. Edge computing technologies for Internet of Things: a Primer. *Digital Communications and Networks* 4, 2 (2018), 77 – 86. https://doi.org/10.1016/j.dcan.2017.07.001

[3] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies* 4, 3 (May 2009), 114–123. https://uxpajournal.org/determining-what-individual-sus-scores-mean-adding-an-adjective-rating-scale/

[4] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction* 24, 6 (2008), 574–594. https://doi.org/10.1080/10447310802205776

[5] Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, and Marco Winckler. 2014. Bridging the gap between a behavioural formal description technique and a user interface description language: Enhancing ICO with a graphical user interface markup language. *Science of Computer Programming* 86, 15 (2014), 3–29. https://doi.org/10.1016/j.scico.2013.04.001

[6] Len Bass, Ross Faneuf, Reed Little, Niels Mayer, Bob Pellegrino, Scott Reed, Robert Seacord, Sylvia Sheppard, and Martha R. Szczur. 1992. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *SIGCHI Bulletin* 24, 1 (1992), 32–37. https://doi.org/10.1145/142394.142401

[7] Len Bass, Reed Little, Bob Pellegrino, Scott Reed, Robert Seacord, Sylvia Sheppard, and Martha R. Szczur. 1991. The Arch Model: Seeheim Revisited. In *Paper presented for the User Interface Developers' Workshop at CHI'91,*. Association for Computing Machinery, New York, NY, USA, 2–26. http://dui-workshop.org/2013/

[8] Federico Bellucci, Giuseppe Ghiani, Fabio Paternò, and Carmen Santoro. [n.d.]. Engineering JavaScript state persistence of web applications migrating across multiple devices. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Pisa, Italy) *(EICS '11)*, Fabio Paternò, Kris Luyten, and Frank Maurer (Eds.).

[9] Samir Benbelkacem, Nadia Zenati-Henda, Djamel Aouam, Yousra Izountar, and Samir Otmane. 2020. MVC-3DC: Software architecture model for designing collaborative augmented reality and virtual reality systems. *Journal of King Saud University - Computer and Information Sciences* 32, 4 (2020), 433–446. https://doi.org/10.1016/j.jksuci.2019.11.010 Emerging Software Systems.

[10] Anastasiia Beznosyk, Peter Quax, Karin Coninx, and Wim Lamotte. 2011. Influence of Network Delay and Jitter on Cooperation in Multiplayer Games. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry* (Hong Kong, China) *(VRCAI '11)*. Association for Computing Machinery, New York, NY, USA, 351–354. https://doi.org/10.1145/2087756.2087812

[11] François Bodart, Anne-Marie Hennebert, Jean-Marie Leheureux, Isabelle Provot, Benoît Sacré, and Jean Vanderdonckt. 1995. Towards a Systematic Building of Software Architecture: the TRIDENT Methodological Guide. In *Proceedings of 2nd Int. Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS '95)*, Philippe Palanque and Rémi Bastide (Eds.). Springer, Vienna, 262–278. https://doi.org/10.1007/978-3-7091-9437-9_16

[12] Jullien Bouchet, Laurence Nigay, and Thierry Ganille. 2004. ICARE Software Components for Rapidly Developing Multimodal Interfaces. In *Proceedings of the 6th International Conference on Multimodal Interfaces* (State College, PA, USA) *(ICMI '04)*. Association for Computing Machinery, New York, NY, USA, 251–258. https://doi.org/10.1145/1027933.1027975

[13] Ugo Braga Sangiorgi, Vivian Genaro Motti, François Beuvens, and Jean Vanderdonckt. 2012. Assessing Lag Perception in Electronic Sketching. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design* (Copenhagen, Denmark) *(NordiCHI '12)*. Association for Computing Machinery, New York, NY, USA, 153–161. https://doi.org/10.1145/2399016.2399040

[14] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* (1996), 189–194. https://www.taylorfrancis.com/chapters/edit/10.1201/9781498710411-35/sus-quick-dirty-usability-scale-john-brooke

[15] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interact. Comput.* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[16] Scott Carter, Amy Hurst, Jennifer Mankoff, and Jack Li. 2006. Dynamically Adapting GUIs to Diverse Input Devices. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility* (Portland, Oregon, USA) *(Assets '06)*. Association for Computing Machinery, New York, NY, USA, 63–70. https://doi.org/10.1145/1168987.1169000

[17] Priscila Cedillo, Emilio Insfrán, Silvia Abrahão, and Jean Vanderdonckt. 2021. Empirical Evaluation of a Method for Monitoring Cloud Services Based on Models at Runtime. *IEEE Access* 9 (2021), 55898–55919. https://doi.org/10.1109/ACCESS.2021.3071417

[18] Benoît Collignon, Jean Vanderdonckt, and Gaëlle Calvary. 2008. Model-Driven Engineering of Multi-target Plastic User Interfaces. In *Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, 16-21 March*

2008, Gosier, Guadeloupe. IEEE Computer Society, 7–14. https://doi.org/10.1109/ICAS.2008.37

[19] Lorenzo Corneo, Maximilian Eder, Nitinder Mohan, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, Per Gunningberg, Jussi Kangasharju, and Jörg Ott. 2021. *Surrounded by the Clouds: A Comprehensive Cloud Reachability Study*. Association for Computing Machinery, New York, NY, USA, 295–304. https://doi.org/10.1145/3442381.3449854

[20] Amir Vahid Dastjerdi and Rajkumar Buyya. 2016. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer* 49, 8 (2016), 112–116. https://doi.org/10.1109/MC.2016.245

[21] Eustace M. Dogo, Abdulazeez Femi Salami, Clinton O. Aigbavboa, and Thembinkosi Nkonyana. 2019. *Taking Cloud Computing to the Extreme Edge: A Review of Mist Computing for Smart Cities and Industry 4.0 in Africa*. Springer International Publishing, Cham, 107–132. https://doi.org/10.1007/978-3-319-99061-3_7

[22] Emmanuel Dubois, Christophe Bortolaso, Damien Appert, and Guillaume Gauffre. 2014. An MDE-based framework to support the development of Mixed Interactive Systems. *Science of Computer Programming* 89 (2014), 199–221. https://doi.org/10.1016/j.scico.2013.03.007 Special issue on Success Stories in Model Driven Engineering.

[23] Yehia Elkhatib, Barry Porter, Heverson B. Ribeiro, Mohamed Faten Zhani, Junaid Qadir, and Etienne Rivière. 2017. On Using Micro-Clouds to Deliver the Fog. *IEEE Internet Computing* 21, 2 (2017), 8–15. https://doi.org/10.1109/MIC.2017.35

[24] Luca Frosini, Marco Manca, and Fabio Paternò. 2013. A framework for the development of distributed interactive applications. In *Proc. of ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*, Peter Forbrig, Prasun Dewan, Michael Harrison, and Kris Luyten (Eds.). ACM, 249–254. https://dl.acm.org/citation.cfm?id=2480328

[25] Luca Frosini and Fabio Paternò. 2014. User interface distribution in multi-device and multi-user environments with dynamically migrating engines. In *ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Rome, Italy) *(EICS '14)*, Fabio Paternò, Carmen Santoro, and Jürgen Ziegler (Eds.). ACM, 55–64. https://doi.org/10.1145/2607023.2607032

[26] Jose A. Gallud, Ricardo Tesoriero, Jean Vanderdonckt, María Lozano, Victor Penichet, and Federico Botella. 2011. Distributed User Interfaces. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI EA '11)*. Association for Computing Machinery, New York, NY, USA, 2429–2432. https://doi.org/10.1145/1979742.1979576

[27] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. 2015. Edge-Centric Computing: Vision and Challenges. *ACM SIGCOMM Computer Communication Review* 45, 5 (sep 2015), 37–42. https://doi.org/10.1145/2831347.2831354

[28] Donatien Grolaux, Peter Van Roy, and Jean Vanderdonckt. 2004. Migratable User Interfaces: Beyond Migratory Interfaces. In *Proc. of 1st Annual International Conference on Mobile and Ubiquitous Systems, Networking and Services, 22-25 August 2004, Cambridge, MA, USA (MobiQuitous '04)*. IEEE Computer Society, 422–430. https://doi.org/10.1109/MOBIQ.2004.1331749

[29] Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. 2005. Attach Me, Detach Me, Assemble Me Like You Work. In *Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction - INTERACT 2005, Rome, Italy, September 12-16, 2005 (Lecture Notes in Computer Science, Vol. 3585)*, Maria Francesca Costabile and Fabio Paternò (Eds.). Springer, 198–212. https://doi.org/10.1007/11555261_19

[30] Kaylie Gyarmathy. [n.d.]. *Edge Computing vs. Cloud Computing: What You Need to Know*. https://www.vxchnge.com/blog/edge-computing-vs-cloud-computing

[31] Ilija Hadžić, Yoshihisa Abe, and Hans C. Woithe. 2017. Edge Computing in the EPC: A Reality Check. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (San Jose, California) *(SEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3132211.3134449

[32] Larry V. Hedges. 1981. Distribution Theory for Glass's Estimator of Effect Size and Related Estimators. *Journal of Educational Statistics* 6, 2 (1981), 107–128. http://www.jstor.org/stable/1164588

[33] Ralph D. Hill. 1986. Event-Response Systems: A Technique for Specifying Multi-Threaded Dialogues. *SIGCHI Bulletin* 17, SI (May 1986), 241–248. https://doi.org/10.1145/30851.275637

[34] Brad Johanson, Armando Fox, and Terry Winograd. 2002. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE pervasive computing* 1, 2 (2002), 67–74. https://doi.org/10.1109/MPRV.2002.1012339

[35] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *CoRR* abs/1902.03383 (2019). arXiv:1902.03383 http://arxiv.org/abs/1902.03383

[36] Sophie Jörg, Aline Normoyle, and Alla Safonova. 2012. How Responsiveness Affects Players' Perception in Digital Games. In *Proceedings of the ACM Symposium on Applied Perception* (Los Angeles, California) *(SAP '12)*. Association for Computing Machinery, New York, NY, USA, 33–38. https://doi.org/10.1145/2338676.2338683

[37] M. Frans Kaashoek and Ion Stoica (Eds.). 2003. *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*. Lecture Notes in Computer Science, Vol. 2735. Springer.

https://doi.org/10.1007/b11823
[38] Grady Klein and Alan Dabney. 2013. *The Cartoon Introduction to Statistics* (1 ed.). Hill and Wang, New York, NY, USA.
[39] Andreas Kouloumpris, Theo Theocharides, and Maria Michael. 2019. Metis: Optimal Task Allocation Framework for the Edge/Hub/Cloud Paradigm. (05 2019), 128–133. https://doi.org/10.1145/3312614.3312643
[40] Minseok Kwon, Zuochao Dou, Wendi Heinzelman, Tolga Soyata, He Ba, and Jiye Shi. 2014. Use of Network Latency Profiling and Redundancy for Cloud Server Selection. In *Proc. of IEEE 7th International Conference on Cloud Computing* (Anchorage, AK, USA) *(Cloud '14)*. IEEE, 826–832. https://doi.org/10.1109/CLOUD.2014.114
[41] Frédéric Lemoine, Noëmie Simoni, and Tatiana Aubonnet. 2017. Self-Controlled Components for Human-Machine Interaction Services. In *Proceedings of the 29th Conference on l'Interaction Homme-Machine* (Poitiers, France) *(IHM '17)*. Association for Computing Machinery, New York, NY, USA, 233–241. https://doi.org/10.1145/3132129.3132153
[42] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. 2018. Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management. *ACM Computing Survey* 51, 2, Article 39 (April 2018), 34 pages. https://doi.org/10.1145/3154815
[43] Marco Manca and Fabio Paternò. 2016. Customizable dynamic user interface distribution. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Brussels, Belgium, June 21-24, 2016 (EICS '16)*, Kris Luyten and Philippe A. Palanque (Eds.). ACM, 27–37. https://doi.org/10.1145/2933242.2933259
[44] Jérémie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. 2009. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering Interactive Computing System , Pittsburgh, PA, USA, July 15-17, 2009 (EICS '09)*, T. C. Nicholas Graham, Gaëlle Calvary, and Philip D. Gray (Eds.). ACM, 69–78. https://doi.org/10.1145/1570433.1570449
[45] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. 2011. A model-based approach for distributed user interfaces. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System, EICS 2011, Pisa, Italy, June 13-16, 2011*, Fabio Paternò, Kris Luyten, and Frank Maurer (Eds.). ACM, 11–20. https://doi.org/10.1145/1996461.1996488
[46] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. 2012. Modelling and developing distributed user interfaces based on distribution graph. In *Proc. of 6th International Conference on Research Challenges in Information Science, Valencia, Spain, May 16-18 2012 (RCIS '12)*, Colette Rolland, Jaelson Castro, and Oscar Pastor (Eds.). IEEE, 1–10. https://doi.org/10.1109/RCIS.2012.6240450
[47] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. 2011. Distribution Primitives for Distributed User Interfaces. (11 2011). https://doi.org/10.1007/978-1-4471-2271-5_3
[48] David Navarre, Philippe Palanque, Jean-Francois Ladry, and Eric Barboni. 2009. ICOs: A Model-Based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. *ACM Trans. Comput.-Hum. Interact.* 16, 4, Article 18 (Nov. 2009), 56 pages. https://doi.org/10.1145/1614390.1614393
[49] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. 2014. Interactive Development of Cross-Device User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 2793–2802. https://doi.org/10.1145/2556288.2556980
[50] Jakob Nielsen. 1993. *Usability Engineering* (1 ed.). Morgan Kaufmann.
[51] Aline Normoyle, Gina Guerrero, and Sophie Jörg. 2014. Player Perception of Delays and Jitter in Character Responsiveness. In *Proceedings of the ACM Symposium on Applied Perception* (Vancouver, British Columbia, Canada, August 8-9, 2014) *(SAP '14)*. Association for Computing Machinery, New York, NY, USA, 117–124. https://doi.org/10.1145/2628257.2628263
[52] Diana Popescu, Noa Zilberman, and Andrew Moore. 2017. *Characterizing the impact of network latency on cloud-based applications' performance.* Technical Report UCAM-CL-TR-914. University of Cambridge Computer Laboratory. https://doi.org/10.17863/CAM.17588
[53] Peter Quax, Anastasiia Beznosyk, Wouter Vanmontfort, Robin Marx, and Wim Lamotte. 2013. An evaluation of the impact of game genre on user experience in cloud gaming. In *Proc. of IEEE International Games Innovation Conference* (Vancouver, BC, Canada, 11 November 2013) *(IGIC '13)*. 216–221. https://doi.org/10.1109/IGIC.2013.6659141
[54] Andreas Reiter, Bernd Prünster, and Thomas Zefferer. 2017. Hybrid Mobile Edge Computing: Unleashing the Full Potential of Edge Computing in Mobile Device Use Cases. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Madrid, Spain) *(CCGrid '17)*. IEEE Press, 935–944. https://doi.org/10.1109/CCGRID.2017.125
[55] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. 2019. A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms: Transparent Computing, Mobile Edge Computing, Fog Computing, and Cloudlet. *ACM Comput. Surv.* 52, 6, Article 125 (Oct. 2019), 36 pages. https://doi.org/10.1145/3362031
[56] Yeong-Seok Seo and Jun-Ho Huh. 2019. GUI-based software modularization through module clustering in edge computing based IoT environments. *Journal of Ambient Intelligence and Humanized Computing* (09 2019), 1–15. https://doi.org/10.1007/s12652-019-01455-3

[57] Weisong Shi and Schahram Dustdar. 2016. The Promise of Edge Computing. *Computer* 49, 5 (2016), 78–81. https://doi.org/10.1109/MC.2016.145

[58] Tarik Taleb, Adlen Ksentini, and Pantelis A. Frangoudis. 2019. Follow-Me Cloud: When Cloud Services Follow Mobile Users. *IEEE Transactions on Cloud Computing* 7, 2 (2019), 369–382. https://doi.org/10.1109/TCC.2016.2525987

[59] Genc Tato, Marin Bertier, Etienne Rivière, and Cédric Tedeschi. 2018. ShareLatex on the Edge: Evaluation of the Hybrid Core/Edge Deployment of a Microservices-Based Application. In *Proceedings of the 3rd Workshop on Middleware for Edge Clouds and Cloudlets* (Rennes, France) *(MECC'18)*. Association for Computing Machinery, New York, NY, USA, 8–15. https://doi.org/10.1145/3286685.3286687

[60] Genc Tato, Marin Bertier, Etienne Rivière, and Cédric Tedeschi. 2019. Split and Migrate: Resource-Driven Placement and Discovery of Microservices at the Edge. In *Proc. of 23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland (LIPIcs, Vol. 153)*, Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:16. https://doi.org/10.4230/LIPIcs.OPODIS.2019.9

[61] Chris Vandervelpen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. 2005. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *Proc. of 5th International Conference on Web Engineering, ICWE '05, Sydney, Australia, July 27-29, 2005 (Lecture Notes in Computer Science, Vol. 3579)*, David B. Lowe and Martin Gaedke (Eds.). Springer, 197–202. https://doi.org/10.1007/11531371_28

[62] Massimo Vecchio, Paolo Azzoni, Andreas Menychtas, Ilias Maglogiannis, and Alexander Felfernig. 2021. A Fully Open-Source Approach to Intelligent Edge Computing: AGILE's Lesson. *Sensors* 21, 4 (2021). https://doi.org/10.3390/s21041309

[63] Shangguang Wang, Jinliang Xu, Ning Zhang, and Liu Yujiong. 2018. A Survey on Service Migration in Mobile Edge Computing. *IEEE Access* PP (04 2018), 1–1. https://doi.org/10.1109/ACCESS.2018.2828102

[64] Jian Xu, Qingqing Cao, Aditya Prakash, Aruna Balasubramanian, and Donald E. Porter. 2017. UIWear: Easily Adapting User Interfaces for Wearable Devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (Snowbird, Utah, USA) *(MobiCom '17)*. Association for Computing Machinery, New York, NY, USA, 369–382. https://doi.org/10.1145/3117811.3117819

[65] Zhe Zhou, Xintong Li, and Guangyu Sun. 2019. Accelerate Service Live Migration in Resource-Limited Edge Computing Systems. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (Arlington, Virginia) *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 354–355. https://doi.org/10.1145/3318216.3363456

[66] Mikel Zorrilla, Njål T. Borch, François Daoust, Alexander Erk, Julián Flórez, and Alberto Lafuente. 2015. A Web-based distributed architecture for multi-device adaptation in media applications. *Pers. Ubiquitous Comput.* 19, 5-6 (2015), 803–820. https://doi.org/10.1007/s00779-015-0864-x

[67] Mikel Zorrilla, Iñigo Tamayo, Angel Martin, and Ana Dominguez. 2015. User interface adaptation for multi-device Web-based media applications. In *2015 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*. 1–7. https://doi.org/10.1109/BMSB.2015.7177251