

Beyond Traditional Web Technologies for Locally Web-Services Migration

Clay Palmeira da Silva¹ ^a and Nizar Messai² ^b

¹University of Huddersfield, U.K.

²Université de Tours, France

Keywords: The CUBE, Web Services, Multiple-Devices, User-Side, Service Synchronization.

Abstract: COVID-19 raised our dependency on computers and mobile devices to perform daily tasks. Thus, we saw a user face the challenges of managing multiple services/device environments. Therefore, this multiple services/devices scenario becomes significant while we still face the lack of interoperability between operating systems, services, and applications on devices. Additionally, big tech companies are fighting to pursue new web technologies for their services. Additionally, they require more resources from devices to keep running. Beyond that, we had to learn and adapt to a new set of server-side services for web meetings, such as Microsoft Teams, Zoom, Google Meet, and Webex. However, we did not see any propositions on the user side for the same web services that could continuously and undistruptive provides a similar user experience of web-service migration in a multiple-devices scenario regardless of the operating system. Therefore, this paper focuses on web services based on REST, RESTful, or GraphQL to analyze their performance using the System Usability Scale. Additionally, we focus on a real-life experiment on multiple-device environments for synchronizing web services' user instances without continuously depending on a cloud-based system. We presented results from 35 users, where we measured various metrics over the previously mentioned web services through three different devices. Moreover, we revisited The CUBE architecture, enhancing features that allow us to obtain new results. The results demonstrate that when the users used the CUBE, they had a better QoS experience, low latency, and better response time. Moreover, the CUBE provides computational performance up to $\approx 69\%$ faster than the traditional cloud-based synchronization procedure.


1 INTRODUCTION


The year 2020 brings up upon us the complete necessity of adopting new technologies, and this behavior has become our new reality. We saw this new technological behavior in computer science users, usually familiar with these technologies, and also for those who do not dispose of previous training or adaptation, which we will call here non-experienced users. Moreover, the last couple of years proves that an entire generation has become more dependent on web services. In this context, a concept gained strength, the Everything-as-a-Service (XaaS) provides us with a broad set of services that make our days more comfortable. However, conscious or not, we increased our dependency on cloud-based systems to manage these web services, demonstrating how dependent we have

become.

In addition to the features of the cloud-based systems, we are living a reality where we have more powerful mobile devices than ever before (Suckling and Lee, 2017). However, we do not entirely explore these devices' capacities regarding processing, storage, and security measures to use web services locally. Moreover, for most people, dealing with an environment that requires managing multiple services or devices is challenging.

Additionally, It is also necessary to deal with the lack of interoperability between operating systems, services, and applications. Moreover, we live in a time of connectivity dependency (da Silva et al., 2020). That means we are sometimes unaware of our connection dependency on network systems or social media. Thus, if we consider the user-side approach, it is far more complicated to migrate a given web service from one device to another without a cloud-based system to ensure the conditions to keep the service

^a  <https://orcid.org/0000-0003-0438-581X>

^b  <https://orcid.org/0000-0002-3784-101X>

running.

On the server side, to keep up with the necessity of adapting web services regarding the scenario of COVID-19, companies such as Google, Twitter, Facebook, and Amazon already use REST and RESTful on their platforms and adopted new technologies such as gRPC or GraphQL. However, the redevelopment of services to the new paradigm is costly (Kus et al., 2020). Furthermore, we faced the rise of miscellaneous services with the same purposes, such as Zoom, Livestorm, Google meets, Zoho Meetings, Microsoft teams, and Amazon chime, among others.

In such a services conflict scenario, to give an example, suppose a user starts to use a given service, a chat on Facebook or LinkedIn on a given device. For any particular reason, it is necessary to change the device. The new device needs a cloud-based system to ensure the service can resume continuing the task. However, despite the cloud-based system, it is not guaranteed that the service will resume the user instance on the new device. We may have a service rupture due to interoperability issues in this case.

An alternative to server-side or cloud-based system approaches can rely on client-side management of multiple device service synchronization. In (da Silva et al., 2018a), the authors proposed The CUBE, a system-model architecture that addresses the challenge of web services at the user side for a multiple-devices and operating systems environment. The CUBE fully complies with Liquid Software principles described in 1996 by (Hartman et al., 1996). Most recently described in 2016 by (Galidabino et al., 2016) to allow for user-side services synchronization and migration over multiple-devices (da Silva et al., 2020).

In the work of Palmeira et al. (da Silva et al., 2020), the authors presented a study of the CUBE's formal aspects, describing its technical improvements with a feasibility test on tight-coupling web service interaction from YouTube, providing some results from a set of 10 users. However, the authors did not provide further details regarding some aspects of their architecture.

This paper focused on services based on REST, RESTful, and GraphQL, such as Facebook, Google Maps, Linked In, and Twitter, to analyze their performance, considering a better user experience. Moreover, revisiting the CUBE, we also discuss how it works this user-instance approach inside The CUBE architecture (da Silva et al., 2018a). However, we focus only on the Pool Area $\{P_a\}$, where the relation $\{d_n \leftrightarrow s_n\}$ between devices $\{d_1, d_2, d_3\}$ and services $\{s_1, s_2, s_3, s_4\}$ occurs. Using the System Usability Scale (SUS) for user experience, we presented results

from 35 users, where we measured various metrics over the previously mentioned web services through three different devices.

The remainder of this paper presents the following structure: Section 2 brings our motivations and contributions. Section 3 brings our perspective in revisiting the CUBE architecture with the proposed improvements. In the following, Section 4 presents service integration and technical challenges, and Section 5 presents the evaluation with commented results. Section 6 discusses the related works from REST and GraphQL perspectives. Finally, section 7 gives concluding remarks highlighting our future research directions.

2 MOTIVATIONS AND CONTRIBUTIONS

Nowadays, web services are the central use of mobile devices. Therefore, we are facing a real problem within a multiple-devices/operating systems environment. Moreover, we are thinking about services running over multiple devices/platforms and how this challenge has inspired researchers and industries. Besides, we also have noted that all proposed solutions rely on the server or cloud-based systems to synchronize services data and state (da Silva et al., 2020).

Today, with all available technology and depending on the cloud-based system to make synchronization, we can reproduce the scenario proposed in Figure 1, which is a migration of a given service across a multiple-devices environment. However, to achieve this scenario, all devices must have an internet connection supported by a cloud-based system with an account of the user on each platform to ensure the service.

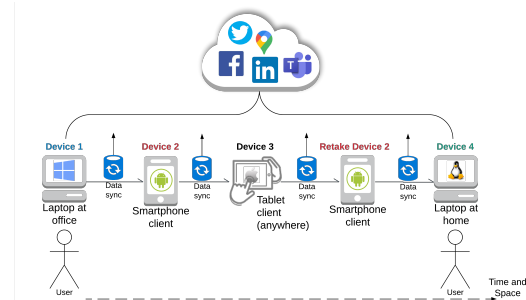


Figure 1: Data/State synchronization is managed by the cloud-based system.

From a resource perspective, some devices are too small or too big for a given service, e.g., a keyboard on a smartphone needs to be bigger to write an email for some users, requiring a device with an external

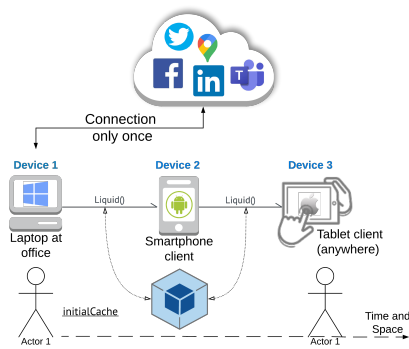


Figure 2: Use case model describes our motivation scenario.

keyboard or a bigger screen. In this case, the user should choose a more suitable "device candidate" to reproduce such a service. In this context, the CUBE architecture (da Silva et al., 2018a) deals with these challenges (da Silva et al., 2020). We want to ensure one specific functionality, the user-instance migration at the user side of a given service towards multiple devices.

However, this user-instance migration on the user side is challenging since we know that most web services use APIs to enable all types of different distributed applications. All in cloud-oriented frameworks, such as Google Cloud Platform. In this case, REST-based approaches present some issues for interface design, such as over-and-under fetching, as was described by Dominik Kus et al. (Kus et al., 2020). These shortcomings triggered the quest for new API-related technologies, such as GraphQL. Despite the flexibility of GraphQL, it presents some challenges as a lack for integration capabilities (Stünkel et al., 2020).

The migration of a REST-based interface to GraphQL or keeping interfaces in both styles is costly (Kus et al., 2020; Stünkel et al., 2020; Brito and Valente, 2020). Therefore, it is required to adopt middlewares such as the OpenAPI-to-GraphQL introduced by Wittern et al. (Wittern et al., 2018) to create a corresponding schema between REST and GraphQL. However, how can users, experimented or not, easily handle the front end with these web services without being concerned with such technologies? We want to go beyond REST or GraphQL-based APIs and deal with their issues in a run-time context that is not bound to such technologies.

We aim to present web-services migration in a multiple-devices environment regardless of the operating system. We graphically present our proposition in Figure 2, where the user makes a connection with the cloud-based system only once. Then they can switch between devices while moving without service disruption.

We addressed the user-instance migration challenge using the user-centric principle to provide a simple solution for the user choosing a device according to a personal interest in a run-time environment. In the CUBE, all state/data/certificate control goes locally. We noted that this transfer is not a copy of the service or something like that. The CUBE operates within different layers and system modules to work less dependent on the cloud-based system. For this purpose, we revisited the CUBE model (da Silva et al., 2018b), (da Silva et al., 2018a), (da Silva et al., 2020), adding new features described hereafter.

- We refined the relation between devices and services $\{d_n \leftrightarrow s_n\}$ on the Pool Area $\{P_a\}$ to overlap technological issues as REST-based or GraphQL-based.
- We abstracted the CUBE architecture to an application deployed as a callback procedure inside the operating system.
- We removed the necessity to have the services applications inside the CUBE for further use by the user.

3 REVISITING THE CUBE MODEL

First, it is necessary to recall some definitions of the CUBE architecture briefly. Our focus here is on the "space" between devices and services where acts the Pool Area $\{P_a\}$.

3.1 CUBE Model Essentials

The CUBE requires some essential elements for its construction. We will focus on the fifth element, the "one service," which we will define as $\{OS\}$. We noted that the formal description of the "one service" or the process migration was not previously discussed.

The $\{OS\}$ occurs when the end user U starts to access a Service s_1 using a device d_1 . This relation is expressed by $d_1 \overrightarrow{OS} s_1$.

We assumed that the end user U has more than one mobile device at disposal d_1 and d_2 . However, service migration is not already available. Let us represent the mobile devices and service for U as: $\mathbb{D}_u = \{d_1, d_2\}$, $\mathbb{S}_u = \{s_1\}$, and $d_1 \overrightarrow{OS} s_1$, $d_2 \overrightarrow{OS} s_1$ but, for now, there is no communication between the devices d_1 and d_2 . That means, the process migration $\{Mig \rightarrow\}$ is not available yet, $d_1 \neg \{Mig \rightarrow\} d_2$. In this case, service migration on the user side is not possible.

We also recalled the description of the INNER CUBE (da Silva et al., 2020), described as a tuple in Definition 3.1:

Definition 3.1. $\mathbb{I}_c := \{d_p, \mathbb{D}_u \setminus \{d_p\}, \mathcal{R}_d\}$ where:

- d_p is the current device being used by the end user U .
- $\mathbb{D}_u \setminus \{d_p\}$ is the set of discovered devices owned by the user end U ready to use.
- \mathcal{R}_d is the research mechanism for devices.

Once the INNER CUBE is built, the set of discovered devices can communicate with each other. Therefore, the process migration $\{Mig \rightarrow\}$ is online, $\{d_1 Mig \rightarrow d_2, Mig \rightarrow d_3, \dots, Mig \rightarrow d_n\}$ is true.

Now, let us denote by " \leftrightarrow " a connection between any device and any service: that is $d \leftrightarrow s$ means device d has established a connection to service s (da Silva et al., 2020).

We also recalled the definition of the OUTER CUBE (Definition 3.1), denoted by \mathbb{O}_c , as the set of Services \mathbb{S} for which a connection has been established with a device in the INNER CUBE, \mathbb{I}_c (da Silva et al., 2020).

Definition 3.2. $\mathbb{O}_c := \{s_i \in \mathbb{S} \text{ such that } \exists d_i \in \mathbb{D}_u \text{ and } d_i \leftrightarrow s_i\}$. where:

- s_i is any current service being used by the user end U .
- d_i is any current device being used by the user end U .
- \mathbb{D}_u is the set of discovered devices owned by the user end U ready to use.

Now it is possible to focus on describing how the elements of the Pool Area acts. Therefore, in the following subsection, we tackle the first contribution of this paper.

3.2 Contributions

Since we have described the *one service* \overleftrightarrow{OS} , we are improving new concepts to the previous definition of the Pool Area (da Silva et al., 2020). Therefore, by Definition 3.2, we are redefining the Pool Area as \mathbb{P}_a , as the set of relationships established between any device d_i in \mathbb{I}_c and any Service s_i in \mathbb{O}_c .

Definition 3.3.

$\mathbb{P}_a = \{d_i \overleftrightarrow{OS} s_i \text{ such that } d_i \in \mathbb{I}_c \text{ and } s_i \in \mathbb{O}_c\}$. where:

1. \overleftrightarrow{OS} that establishes the connection between device and service, is revisited as:
2. $\overleftrightarrow{OS}_\beta \{d_1 Mig \rightarrow d_2\}$ for connections between devices and services with the CUBE.

3. $\overleftrightarrow{OS}_\alpha d_1 \neg \{Mig \rightarrow\} d_2$ for connections between devices and services without the CUBE.

Let us now describe how is expressed the complexity of item 3 on the Definition 3.3 since there is no service migration nor communication between devices. It is expressed by $\sum_{i=1}^n d_i \sum_{j=1}^m s_j = (d_1 + d_2 + \dots + d_n) \cdot (S_1 + S_2 + \dots + S_m)$. On the other hand, item 2 of the same definition 3.2 presents less complexity, expressed by

$$\sum_{i=1}^n d_i = d_1 + d_2 + \dots + d_n.$$

3.3 Discussing the Model-Scenario Representation

The following use-case, shown in Figure 3, uses different web services based on REST or GraphQL. We provided a detailed perspective of these elements regarding the positioning of the CUBE in this new context.

We revisited the scenario representation presented in (da Silva et al., 2020) to discuss how our proposed improvements to the CUBE architecture allow for a better user experience. We consider this point a key feature of our contribution.

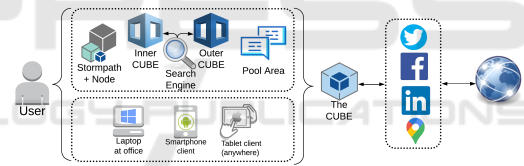


Figure 3: Our second feasibility test scenario of the CUBE.

To start, we must assume three conditions:

1. The user-end chooses a connected mobile as their first device;
2. Users decided to run any service from the set (Facebook, Twitter, LinkedIn, or Google Maps);
3. The user launches a given service from a web browser or a native application

Adding new devices to the CUBE is still the same as before. The user manually selects the devices to the INNER CUBE (\mathbb{I}_c) (Definition 3.1). Then, the synchronization process creates the CUBE session. We note that these are the only similarities between the underlying processes for the work presented in (da Silva et al., 2020).

The runtime instance of the CUBE runs as a callback inside the operating system. Thus, using device d_p , the user requests a service $s_i \in \mathbb{S}$, e.g., start a chat from Facebook or LinkedIn.

Once the user starts the service on the (d_1), she/he can migrate the service instance to another device (d_2 or d_3) using the button present on the browser or inside the native app. Since the process migration is available $\{Mig \rightarrow\}$, the user can retrieve the exact moment of their instance without service disruption.

Thus, callback procedures will automatically update the CUBE elements, the *INNER CUBE* (\mathbb{I}_c), Pool Area (\mathbb{P}_a), and *OUTER CUBE* (\mathbb{O}_c). Therefore, allowing the user to freely handle the migration of services towards any device that belongs to them.

4 SERVICE INTEGRATION AND TECHNICAL CHALLENGES

The results we presented in (da Silva et al., 2018b; da Silva et al., 2020) encouraged us to go further and tackle the challenges we found. Thus, we decided to explore the boundaries and limitations of the REST-based and GraphQL-based approaches to enhance our proposal for a scenario where the technology behind a given web service cannot interfere with the end-user experience.

For this purpose, it was necessary to change the underlying code and project the CUBE as a repository. Thus, we comply with our second contribution to this paper. To recall, we proposed abstracting the CUBE architecture to an application and deploying it as a callback inside the operating system. We presented in Figure 4 the context of how the user end can directly interact with different web services easily and efficiently through the CUBE.

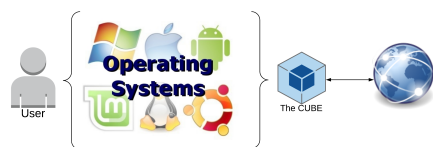


Figure 4: The representation of the second feature of this paper shows how the user end interacts with web services through the CUBE.

The improvement mentioned in the previous section regarding the CUBE in the run-time environment made us realize that the logical dock, once used to deploy the services presented in (LiquidMail (da Silva et al., 2018b) and LiquidTube (da Silva et al., 2020)) became deprecated. However, we retain all of the functionalities already mentioned in the author’s previous works. Since we no longer need the logical dock to deploy services, we realized that it makes the CUBE more efficient for managing services, especially for the non-experienced user.

We reached some successful migrations regard-

ing the meeting services, such as Microsoft Teams. We successfully migrated an ongoing meeting to another device in the same scenario. Issues regarding protocols TCP and UDP associated with WebRTC maybe jeopardize the simulation. For now, we cannot precisely describe the unstable error that we found. Therefore, since we could not consistently reproduce such migration in our tests, we preferred, for now, to see it as a work in progress.

5 EVALUATION

For the evaluation, we followed the same as the authors mentioned in (da Silva et al., 2020)). We sent the test through the CUBE. The end user randomly chose the user instance without a third party synchronizing it. In this feasibility test, we used three different devices. Device one (d_1), a desktop HP Pentium Dual-Core 3GHz, 8GB Ram, and a Windows 7 64bits Professional. The device two (d_2), a Tablet Samsung A6 Chipset Exynos 7870 octa-core 1.6GHz, Wi-Fi 802.11 a/b/g/n, and Wi-Fi Direct, Bluetooth v4.2 LE-A2DP with Android 8.0 Oreo. The device three (d_3), an iPad 32GB, Chipset 1Ghz Single-Core ARM Cortex-A8, Wi-Fi 802.11 a/b/g/n (2.4Ghz), Bluetooth 2.1-A2DP with iOS 4.0.

With a set of thirty-five persons with no prior training with the CUBE, the test consists of starting any service from the set (Facebook, Twitter, Linked In, Google Maps or Microsoft Teams) on the device one d_1 (Desktop Windows). Then, migrate by clicking on the extension icon in the browser on d_1 to migrate the user-instance to d_2 (Tablet Android) with the actual content (context/interface/data). In the sequence d_2 sends the content to d_3 (Tablet iOS). This sequence can repeat towards any other device until returning to the first device, d_1 (Desktop Windows). We note that the user must migrate the service to three different devices. Figure 5 shows the steps on different devices. The considered interactions are those only between d_1 until d_3 .

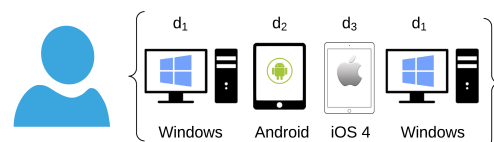


Figure 5: The service’s content is fluently moved around a different set of devices \mathbb{D} owned by the user-end.

Regarding data integration challenges due to requests from services with different bases (REST or GraphQL), we evaluate each device group’s latency by minimum, maximum, average, median,

and jitter as a metric inside the underlying latency. Since we had online interactions, we provided a sample of the user-instance migration for the mentioned services in the following link *https://www.youtube.com/watch?v=9en1auesqU*.

5.1 Results

Here we decided to separate our analysis due to the inconsistency found on Microsoft Teams. Therefore, we have two sub-sections with the respective information.

5.1.1 Social Media Services

Since we have considered enhancing the migration process of the CUBE, it was necessary to measure the end user sentiment about using the CUBE as the orchestrator of web services. Moreover, we needed objective metrics about how services required network infrastructure with and without the CUBE was necessary since this was also the scope of our contributions.

From the perspective of a SUS metric for user experience, we present a set of graphics in Figure 6, each with a different metric. In (a) the analysis regarding usability. In (b), the perceived satisfaction (PSSUQ) with three subconstructs. In (c), the usability analysis from the SUMI model within three subconstructs. In (d), the QUIS model analyzes the interaction satisfaction also within three subconstructs. Finally, in (e), the website quality from WEBQUAL is within three subconstructs.

Considering the latency with the following metrics: minimum, maximum, average, median, and jitter, we achieved an enhancement an average on the following results with the synchronization on the client side (CliSi) over the server side (SerSi): 3,09ms (CliSi) against 6,19ms (SerSi) represents 50% faster for the minimum latency, 8,86ms (CliSi) against 28,83ms (SerSi) represents 69,26% faster for the maximum latency, 5,74ms (CliSi) against 18,33ms (SerSi) 68,68% faster on average, and 5,68ms (CliSi) against 19,69ms (SerSi) represents 71,15% faster on the median average. Figure 7 (a) shows the metrics obtained with the synchronization by the Server-side. On the other hand, (b) presents the synchronization at the Client-side provided, which CUBE orchestrates the migration.

Regarding this latency aspect over the devices, it was possible to identify that client-side synchronization allows for a lower average latency between devices, which is the complete opposite on the server side since there is a higher dependency on the network infrastructure necessary to migrate the services.

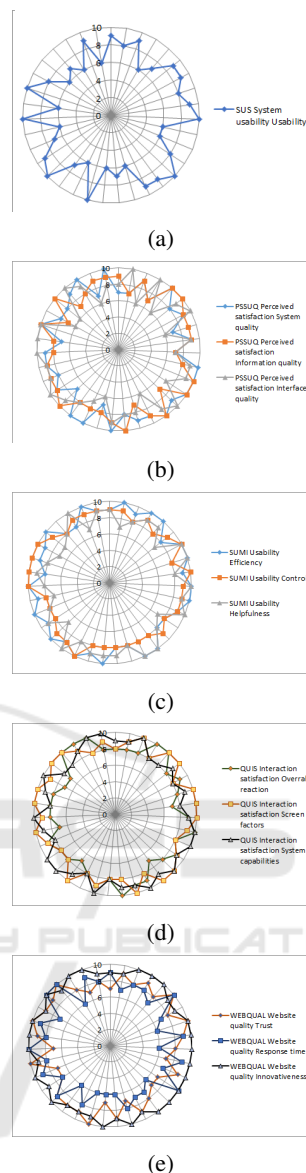


Figure 6: We present the evaluation from a SUS perspective in five different measurements.

The achieved abstraction of service migration with both tight and loose-coupling services within different server-side implementations, such as REST, RESTful, or Graph-QL, allows our contribution to exploring user-instance orchestration in multiple device scenarios without concerns regarding programming languages or frameworks.

5.1.2 Online Meeting Services

As previously mentioned, we have managed some migration of online meetings using Microsoft Teams. At some level, most of the users from the test set achieved the migration, but with some disruptive fail-

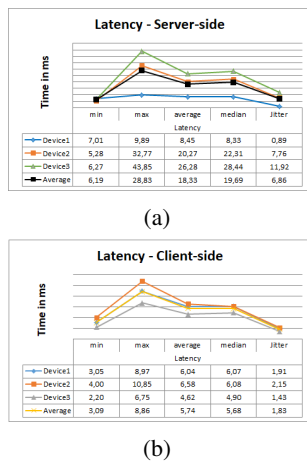


Figure 7: We presented the latency from network analysis comparing server-side vs. user-side synchronization.

ures. We looked back at the improvements proposed at the Pool Area to understand if was some modification in our proposition. However, we realized the disruptive condition was previous to our modifications. However, since we achieved such functionality, even if it was not stable enough during the tests, it is feasible for future research topics on service migration orchestrated on the user side.

6 RELATED WORK

We present in this section the contributions regarding the challenges of REST, RESTful, and migration to GraphQL. Technically speaking, we managed to avoid the challenges presented by the authors with the abstraction of the CUBE architecture, as we proposed in our contributions.

The work of Palmeira et al. (da Silva et al., 2018a; da Silva et al., 2018b; da Silva et al., 2020) requires some user manipulation, directly impacting the usability. Moreover, they need a dock to anchor the services (e.g., LiquidMail and LiquidTube). Thus, we could abstract this docker constraint and make the procedure more intuitive with buttons on browsers and native apps. Additionally, we revisited the Pool Area proposed by the authors to enhance their functionality of service migration to help further development requirements on our client-side approach.

The OpenAPI-to-GraphQL was introduced by Wittern et al. (Wittern et al., 2018) as a language for web applications at the server side, promoting a new perspective for developers and applications that make use of the REST-based platform. Their results demonstrate a trend with challenges up ahead. Regarding the challenges that address the migration or trans-

lation process from REST-based to GraphQL-based, the study conducted in (Kus et al., 2020) presents a link generator for increasing the utility of OpenAPI-to-GraphQL translations by creating a wrapper based on its documentation.

Challenges regarding migration from REST to GraphQL are also presented by Brito and Valente (Brito and Valente, 2020), where their results demonstrate that GraphQL is more comfortable to implement over REST, with measured performances between participants.

Another approach to enhancing GraphQL-based solutions is introduced in (Díaz et al., 2020), where authors propose an application GrapCoQL, which enhances the techniques presented in (Hartig and Pérez, 2018).

An exciting approach in (Luscher, 2016) deals with GraphQL at the client side, proposing a solution to inject a custom relay-local schema. However, since we have higher network demand for the underlying REST API, the author suggests moving the schema from the client side to the server side to reduce latency issues.

The work presented by Stünkel et al. explains the main differences and challenges between SOA systems with integration on the service level and Microservice architecture that encourages decoupling, where data integration is a common issue. During the running test phase, we noticed no data integration problems. Therefore, our achievement makes us optimistic about future directions.

7 CONCLUSIONS

During the pandemic of the COVID-19, we saw many web services become more familiar to a non-experimeted user. In this context, Social Media and online meeting services have become a trend. However, most of the services proposed by different companies are similar regarding online meetings. Therefore, the user needs to learn or still does to use them.

We can add to this chaotic scenario the lack of compatibility and interoperability of the most recent technologies, such as REST, RESTful and GraphQL. We realized that the transition would take a while due to data integration and query complexities in both senses. To tackle this from a more abstract perspective, we presented an enhancement of the CUBE architecture to go beyond the technological limitations, which impose constraints on developers, retailers, and industry.

In this paper, we described some of the main challenges regarding the approaches to convert REST-

based to GraphQL-based schemas. They mainly converge to solutions that require additional effort or content transfer to a server-side due to the high request over the network infrastructure. Moreover, the approaches deal with migration that not involves the non-experimented user or, even more, any optimal synchronization on the client side.

We focus our approach on two main objectives. First, propose an enhancement on the Pool Area of the CUBE architecture, with a more clear mathematical formalism to make it easy to develop and integrate solutions on the already defined architecture. After that, we focused on evaluating the user experience and network latency, mainly on the non-experimented user.

We successfully migrated social media services in a multiple devices environment with positive results regarding the latency in a network system while orchestration runs on the user side. On average, the client-side orchestration provided by the CUBE allows for a faster service migration experience, with 68,68% faster over the server side. Moreover, it is possible to notice on the client-side graphic that latency is more balanced, opposite to the latency of the server side, which demonstrated a higher variation due to network infrastructures.

We partially achieved online meeting service migration in the same environment. Thus, we saw that achievement not as a failure but as a positive achievement that required more investigation and perspectives on the user side development. Therefore, we advocate synchronization on the client side as an alternative to the classic server-side synchronization. This approach's main advantages rely on having REST and GraphQL running simultaneously without interoperability or data integration issues, as we found in our test with thirty-five users. Furthermore, infrastructure failures could lead to fewer network requests without service disruption. Consequently, we can assume a reduction in energy consumption of network servers infrastructure.

We also realized that The CUBE could be scalable, where a higher number of devices can be added to its microsystem to perform service migration without compromising performance. Moreover, since we successfully added it to the operating system, its features can be used for a more extensive set of devices from different users. It will require some assessments of the architecture to provide such a feature.

Finally, as future works, we are considering tackling challenges raised during the migration on the online meeting, such as those we performed in Teams. The CUBE architecture worked with some limitations. Thus, we considered starting to notate services and devices to create an ontology-based notation to

propose to the non-experimented user a better migration to a given device of a given service. Additionally, we can enhance user devices to provide some services, particularly those using IA. Therefore, the federated learning area raises a challenge when the synchronization goes on the user side. Moreover, scaling The CUBE over devices from different users opens the possibility of applying The CUBE architecture on Smart City applications spread over the FOG and EDGE infrastructure.

REFERENCES

- Brito, G. and Valente, M. T. (2020). REST vs graphql: A controlled experiment. In *2020 IEEE International Conference on Software Architecture, ICOSA 2020, Salvador, Brazil, March 16-20, 2020*, pages 81–91, Salvador, Brazil. IEEE.
- da Silva, C. P., Messai, N., Sam, Y., and Devogele, T. (2018a). Cube system: A rest and restful based platform for liquid software approaches. In Majchrzak, T. A., Traverso, P., Krempels, K.-H., and Monfort, V., editors, *Web Information Systems and Technologies*, pages 115–131, Cham. Springer International Publishing.
- da Silva, C. P., Messai, N., Sam, Y., and Devogele, T. (2018b). Liquid mail - A client mail based on CUBE model. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS, Vienna, Austria, July 2-6*, pages 1539–1540, Vienna, Austria. Institute of Electrical and Electronics Engineers (IEEE).
- da Silva, C. P., Messai, N., Sam, Y., and Devogele, T. (2020). User-side service synchronization in multiple devices environment. In Bieliková, M., Mikkonen, T., and Pautasso, C., editors, *Web Engineering - 20th International Conference, ICWE 2020, Helsinki, Finland, June 9-12, 2020, Proceedings*, volume 12128 of *Lecture Notes in Computer Science*, pages 451–466, Helsinki, Finland. Springer.
- Díaz, T., Olmedo, F., and Tanter, É. (2020). A mechanized formalization of graphql. In Blanchette, J. and Hritcu, C., editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 201–214, New Orleans, LA, USA. ACM.
- Gallidabino, A., Pautasso, C., Ilvonen, V., Mikkonen, T., Systä, K., Voutilainen, J.-P., and Taivalsaari, A. (2016). On the architecture of liquid software: Technology alternatives and design space. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 122–127, Venice, Italy. IEEE.
- Hartig, O. and Pérez, J. (2018). Semantics and complexity of graphql. In Champin, P., Gandon, F. L., Lalmas, M., and Ipeirotis, P. G., editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1155–1164, Lyon, France. ACM.

- Hartman, J., Manber, U., Peterson, L. L., and Proebsting, T. (1996). Liquid software: A new paradigm for networked systems. Technical report, University of Arizona, Tucson, AZ, USA.
- Kus, D., Koren, I., and Klamma, R. (2020). A link generator for increasing the utility of openapi-to-graphql translations. *CoRR*, abs/2005.08708.
- Luscher, S. (2016). Wrapping a rest api in graphql.
- Stüinkel, P., von Bargen, O., Rutle, A., and Lamo, Y. (2020). GraphQL federation: A model-based approach. *J. Object Technol.*, 19(2):18:1–21.
- Suckling, J. and Lee, J. (2017). Integrating environmental and social life cycle assessment: Asking the right question. *Journal for Industrial Ecology*, 1.
- Wittern, E., Cha, A., and Laredo, J. A. (2018). Generating graphql-wrappers for rest(-like) apis. In Mikkonen, T., Klamma, R., and Hernández, J., editors, *Web Engineering - 18th International Conference, ICWE 2018, Cáceres, Spain, June 5-8, 2018, Proceedings*, volume 10845 of *Lecture Notes in Computer Science*, pages 65–83, Cáceres, Spain. Springer.

