Liquid Mail - A client mail based on CUBE model

Clay Palmeira da Silva, Nizar Messai, Yacine Sam, Thomas Devogele
Département Informatique
Université François Rabelais
Tours, France
clay.palmeiradasilva@etu.univ-tours.fr,
{nizar.messai,yacine.sam,thomas.devogele}@univ-tours.fr

***Abstract*—Nowadays we live with a large number of different connected devices with various operating systems. This is leading us to a lack of technological resources to deal with this multiple connected devices owned by the same user. Thus, a new way to create applications is required in order to enhance the user-experience without being a concern with the operating system of the device. In this work, we present a client mail, based on CUBE model principals, with allowing the user changing device while using a mail service without losing session/data or connection. Preliminary results demonstrate its feasibility moving from a windows system to Android keep mail service running.**

***Index Terms*—Multi-device, Web Services, Liquid Software, CUBE model.**

I. Introduction

In our daily lives, that we have at our disposal an important number of different types of computing devices. We try to bring for our daily tasks all these technologies. However, when we think of the user-side, what should be a good thing in order to help our tasks can, in fact, be the opposite. Actually, we spend too much time and effort in order to use the same services in the different devices we own.

When this perspective is expanded to the number of connected devices, e.g. tablets, watches, and so on, we realise the insufficient resources to deal with this amount of devices at the same time. This scenario meets the goals of Liquid Software, which proposes fluently moving applications and services between devices sharing their behaviour and complexities.

In order to position our contribution, we will not address the liquid contributions at the Server-side, such as Apple's Handoff, Nextbit's Baton or Google Docs, once they are limited to certain operating systems. Insted, based on the CUBE principals, our implementation make synchronization at **USER-SIDE** and respecting the user-experience and context regardless the operating system.

As an example of our implementation, the use case in Figure 1 shows how a user can start a service (e.g., a client e-mail) and be able to switch between devices while moving, and also can be able to take a previously used device (Retake Device 2), then, at the final destination finish the message in another device.

In this paper, we extends the work presented in [1] through a client mail addressing the following challenges: *(i)* Create an environment to share behavior/session/connection; *(ii)* Allowing the user to easily change devices as to her/his needs; *(iii)* Continue to use a given available server and services; *(iv)* Allowing liquid software principals and *(vi)* Improving user mobility across multi-platform.

Some works as seen in [2], [3] and [4], had tried to address the multi-device service migration - but not multi-OS (e.g., Android and iOS), and all used Android as a platform.

For summarizing, the existing work at the State-of-Art, try to bring together some aspects of the Liquid software principals (e.g. fluidity), for that, they use different techniques as interface mismatches [5], middleware, synchronization [6], cross-device [2], among others. However, their approaches don't bring together the concepts proposed at the CUBE, that means, both the multi-(device/OS) in order to move fluently from services and devices.

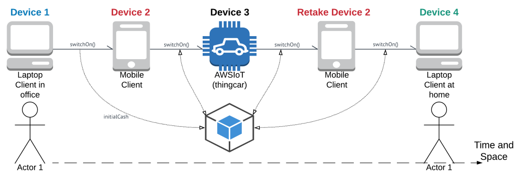


Fig. 1: A use case model describing our contribution.

Until today, we are not able to obtain a simple data-flow of web services without depending from the Server-side. To the best of our knowledge, there is no existing approaches which aims to solve the above mentioned problems at the User-side.

II. System Model

We briefly describe the CUBE model principals where our contribution is based on in order to deal with connected multi-devices owned by the same user.

A. Architecture

The content management system Figure 2 101 includes two application portions 103 and 105, two applications program interface (API) portions 102 and 106, developed in JavaScript, and a database portion 104.

From the portion 100, the content management system 101 will interact on two fronts, user's and services side. First, at 100, a device is specified as the first one. Its choice is made from the available devices and owned by the same user. Then, at 107, the CUBE will be able to receive and respond different

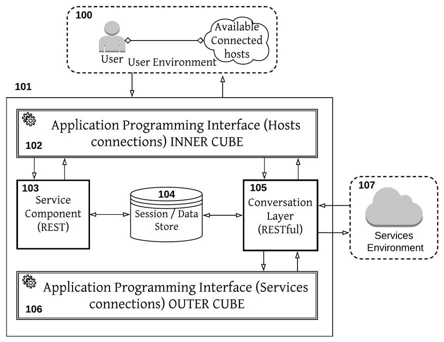

100
User
User Environment
Available Connected hosts
101
Application Programming Interface (Hosts connections) INNER CUBE
102
103
Service Component (REST)
104
Session / Data Store
105
Conversation Layer (RESTful)
107
Services Environment
Application Programming Interface (Services connections) OUTER CUBE
106

Fig. 2: CUBE works with multiple devices and services

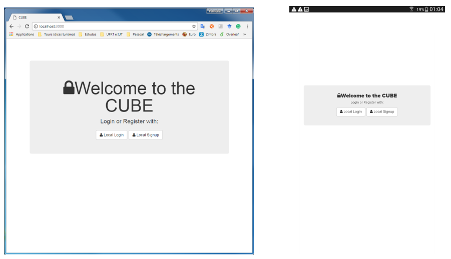
types of requests made by the 100. Meanwhile, services are synchronized in at API 106.

Next, the *INNER CUBE* 102 interact with the user's devices, allowing change it during the service execution. While services and devices changes, all the information, and sessions retrieved are stored at 104 via module 103. Thus, when any change of device is required, the user does not need to start from scratch in order to retrieve or fill in her/his information.

Working as a pool area, the application portion 105 its also an API built under RESTful principles and is responsible for changing information across the multi-platform environment. Moreover, this part of the module provides a secure connection for the user devices through a token created in the *INNER CUBE* 102 by the join of User ID from device's and service which are provided by the *OUTER CUBE* 106. This type of addressability through a resource identification allows the pool to ensure a stateless interaction between services. Also relies at pool area make requests at external services, portion 107.

III. Implementation Insights

Were considered SQL or NoSQL model to authenticate and synchronization, which are deployed as a callback procedure for good performance. Regarding the Internet connection, we manage a local (sign up/login in), also allows synchronizing this authentication step when a connection is available. In addition, we got IMEI and MAC addresses of the device and associate them to the user login.



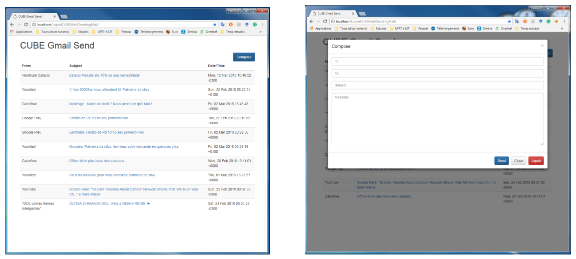
(a) Windows (b) Android

Fig. 3: CUBE deploys in a multi-OS without changing its code

Using a hybrid programming model, our proposal connects through a desktop using web services principals in a Windows and iOS platform. We also achieve connection in the Android platform, through a tablet Samsung Galaxy tab4 with Android 4.4.2, as shown in Figure 3 (b).

The first connection in the CUBE may take a while as the user must specify her/his own device among all devices displayed. Then, other callback functions create the user-centric environment shown available devices ready to be used.

The feasibility test is shown in Figure 4. We developed a small client email, Figure 4 (a), integrated with the Gmail API with the following fields: to, cc, subject and body, Figure 4 (b). Thus, we were able to start the CUBE with our email service in one device and successfully move to other devices retrieving the data already typed and just add new information until send the mail. We notice was no Server-side synchronization to ensure data fluidity between devices.



(a) Local Inbox (b) Local Sending

Fig. 4: Created e-mail deployed locally using Gmail API

IV. Conclusions

Throughout a User-Centric approach and combining principles of REST and RESTful models, our application, based on CUBE model principals, is able to converge multiple and heterogeneous environments having different behavior and complex systems. Thus, it displays the ability required in dealing with the challenges of the Internet platform in a multi-device environment allowing fluently moving from both services and devices regardless the operating system.

References

[1] C. P. da Silva, N. Messai, Y. Sam, and T. Devogele, "Diamond - a cube model proposal based on a centric architecture approach to enhance liquid software model approaches," in *Proceedings of the 13th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST,*, INSTICC. ScitePress, 2017, pp. 382–387.

[2] P. Hamilton and D. J. Wigdor, "Conductor: Enabling and understanding cross-device interaction," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 2773–2782.

[3] D. Wolters, J. Kirchhoff, C. Gerth, and G. Engels, *Cross-Device Integration of Android Apps*. Cham: Springer International Publishing, 2016, pp. 171–185.

[4] A. N. Iyer and R. T., "Extending android application programming framework for seamless cloud integration," in *Proceedings of the 2012 IEEE First International Conference on Mobile Services*, ser. MS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 96–104.

[5] M. Autili, P. Inverardi, F. Mignosi, R. Spalazzese, and M. Tivoli, *Automated Synthesis of Application-Layer Connectors from Automata-Based Specifications*. Cham: Springer International Publishing, 2015, pp. 3–24.

[6] J. Lee, S. J. Lee, and P. F. Wang, "A framework for composing soap, non-soap and non-web services," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 240–250, March 2015.