Diamond
A Cube Model Proposal based on a Centric Architecture Approach to Enhance Liquid Software Model Approaches

Clay Palmeira da Silva, Nizar Messai, Yacine Sam and Thomas Devogele
Departement Informatique, Université François Rabelais, 30 Avenue du Monge, Tours, France

Keywords: RESTful, API, Web Services, HTTP, Liquid Software, CUBE Model.

Abstract: The adoption of multiple connected devices in our lives is a reality which the available technology is not able to deal with. The concept of Liquid Software emerged in the end of the 90s, however, its full potential of a unified interface which can drift between different connected devices and bring with its behavior and complexities is still not fully applied. Thus, enhancements of current Web application architecture, in other words, a new approach able to deal with our technology requirements is required. In this context, we propose a centric-basic architecture to deal with Liquid Software principles and constraints. The CUBE, once built, should be able to deal with all these requirements, making use of best practices from different technologies.

1 INTRODUCTION

In the last few years, the concept of RESTful has been widely adopted, and a large number of APIs or frameworks have been created and defined as so, although, several studies indicate that many APIs are not fully RESTful (Florian Haupt and Pautasso, 2015).

RESTful principles require the use of hypermedia. Thus, API or Frameworks which apply XML or JSON on their codes do not make use of hypermedia handling, since these languages are not able to deal with hypermedia. Furthermore, to be defined as a Level 3 RESTful API means to have the ability to change the current state of a resource through hypermedia. Thus, according to (Pautasso, 2014), APIs which do not deal with hypermedia are not fully mature and cannot be defined as RESTful.

This leads us to another insight: Is the HTTP protocol used completely and correctly? (Richardson and Ruby, 2007) Is the REST architecture respected? In other words, are the required design constraints defined by (Pautasso, 2014) followed in order to achieve RESTful principles?

Initially in our research, these unanswered questions still require more investigation. Meanwhile, other related issues drive us in different directions, such as: i) Maturity Software; ii) Instantiation Resources; iii) HTTP Link Header; iv) API Conversation and v) Liquid Software. (Pautasso, 2014), (Tommi Mikkonen and Pautasso, 2015), (Florian Haupt and Pautasso, 2015), (Andrea Gallidabino et al., 2016), (Panziera and Paoli, 2013), (Mahdi Bennara and Amghar, 2014), (Savas Parastatidis and Robinson, 2010), (Richardson and Ruby, 2007).

These directions lead us to assess how Web Services are built. While they exchange representations of their resources with consumers, they never provide direct access to the actual state of the underlying resources (Savas Parastatidis and Robinson, 2010). Therefore, the Web does not support pointers (Savas Parastatidis and Robinson, 2010), and URIs are used to construct all representations with their resources on the Web, relating, connecting and associating them.

All these different approaches and techniques are driven towards a common interest, of attempting to find a way to enhance communications through API, Web Services and Frameworks, using or not the concepts of Liquid Software, detailed later in this paper. However, an approach or methodology which can combine these techniques to achieve results is still unavailable.

It is in this scenario that we present our CUBE model based on a centric architecture approach, in order to support multiple connected devices owned by the same user. Once the information is acquired from a Server, it is possible to share data with all the connected devices in the same user-network through a trigger. This makes the not only a technology integrator, but also a way of saving resources.

This paper describes our CUBE model concept for the design of Web applications and how and to which extent current technologies can support its novel requirements. This paper is structured as follows: Section 2 provides a background regarding the previously mentioned main constraints. Section 3 discusses engineering in light of present Web technologies. Section 4 provides extended discussions regarding our findings and Section 5 gives concluding remarks highlighting future research directions.

2 CUBE MODEL APPROACH MOTIVATIONS

This section describes how different techniques and perceptions led us to our proposal, and how we extracted the best state-of-art practices and built them into our CUBE model.

2.1 RESTful Principals and Constraints

We can assume that, while service-oriented computing becomes adopted every day, SOAP-based APIs is more and more giving place for REST-based approaches. According to (Mahdi Bennara and Amghar, 2014) in 2014, 2125 SOAP-based APIs were developed, against 6833 REST-based ones.

To be defined as RESTful, the platform should follow some principles, as described in (Pautasso, 2014), which, in general, lead to high system performance, reduced consumption, high interoperability and security in all devices, as well as encapsulated legacy systems.

In addition, RESTful services should also respect some constraints, such as: i) Stateless interaction, ii) Resource linking, iii) Identification or addressability of resources, iv) Uniform interface, v) Hypermedia as a mechanism for decentralized resources and vi) Self-describing messages (Mahdi Bennara and Amghar, 2014) (Pautasso, 2014) (Panziera and Paoli, 2013).

Thus, when all these principles and constraints are put together as best practices to create a REST-based application (API or Framework), another problem emerges: Which language can be used to support a level 3 RESTful characteristic? What should be the choice in order to support hypermedia controls? These questions lead to the concept of maturity software addressed in the next subsection.

2.2 Maturity Software

A level 3 RESTful characteristic relays in the ability of a service to change the set of links that are given to a client based on the current state of a resource (Pautasso, 2014) (Lanthaler and Gtl, 2011). In this context hypermedia support controls XML and JSON (Tuan-Dat Trinh et al., 2015) are not able to achieve a real Level 3. On the other hand Atom, XHTML or JSON-L are able to do so (Pautasso, 2014).

However, what exactly does this mean? An API to be defined as RESTful should be fully mature, thus, making use of hypermedia in order to model the relationship between resources. As put simply by (Savas Parastatidis and Robinson, 2010) that hypermedia drives systems to transform their application state.

Still according to (Savas Parastatidis and Robinson, 2010), the hypermedia system is described by participants as a transferring resource representation that contains links according to an application protocol.

The maturity level of a service also affects the quality attributes of the architecture in which the service is embedded (Pautasso, 2014). That is why a standardized and uniform interface was applied to each resource in order to remove unnecessary variations and to enabling all services to interact with all resources within the architecture, thus promoting interoperability and serendipitous reuse (Vinoski, 2008), (Pautasso, 2014).

2.3 Instantiation Resources

In a scenario of multiple devices, the instantiation resource feature is significantly important. This is a key feature of most RESTful Web Services, which enables clients to create new resource identifiers and set the corresponding state to an initial value (Pautasso, 2014).

Consequently, a resource can either be set by the service or by the client. Thus, it is possible to assume that the URI created by the service is unique, while it is possible that multiple clients generate the same identifier (Pautasso, 2014). This feature can be used in order to provide a new set of services delivered by different servers to the same client. But, in order to do this, the approach proposal described by (Tommi Mikkonen and Pautasso, 2015) should be modified for an inside perspective, where the services are shared by the connected devices.

In this aspect, we allow for the use of POST, GET, SET and DELETE to be managed by the API in order to achieve the result expected by the client. Thus, it seems necessary to establish a better appliance of semantics concepts, not only in the Instantiation Resources but also in the Link Header, which changes significantly according to client behavior.

However, when accessing a social-media platform such as Linkedln through Facebook or Google, authorization from the client is required, and these platforms often accuse a security error when attempting integration. It is noteworthy that the certificate to access the different resources is not provided by the Servers/Services, but by the client through an encrypted password.

2.4 Link Header

A Link header is a powerful HTML resource, although not usually correctly applied. It gains significant value in REST-based approaches. Some interesting research has been conducted in this regard, as seen in (John and M.S.,), who propose to discover REST resources as the user navigates. Their work seems promising, but depends too much on perfect assumptions in order to obtain results.

Another similar approach is composed of RESTful-linked services on the Web, as proposed by (Mahdi Bennara and Amghar, 2014). Despite their effort and interesting proposal, there is a lack of concern with the use of Semantics in order to achieve better results.

In fact, when the link header is used correctly it is possible to include one or more hyper-links. Thus, multiple targets are shown to be selected by the client. This gives rise to other issues, e.g. which URI should the client follow and does its selection imply the expected result by the client? If so, how should the consumer's choice of the URI be conducted in order to obtain the expected outcome?

These issues still require attention in order to enhance the Link Header. It is important not only to flood the consumer with all URIs, but also to give a well-defined set of useful URIs.

2.5 API Conversation

We can define an API Conversation as a set of communication activities between two or more participants (Florian Haupt and Pautasso, 2015). Thus, the focus is on communication between a consumer and a RESTful Web API. From the client's perspective, this means an interaction with a certain API to fulfill a goal. Thus, the resources are the building blocks of each RESTful Web API (Florian Haupt and Pautasso, 2015).

When a consumer requires multiple request/response interactions, she/he wishes to exchange published resources with one or more consumers. Therefore, it is possible to have multiple RESTful APIs emerging from the navigation within a Web through hypermedia relationships sponsors by consumers (Florian Haupt and Pautasso, 2015), (Tommi Mikkonen and Pautasso, 2015), (Pautasso, 2014),(Mahdi Bennara and Amghar, 2014). Figure 1 demonstrates how multiple requests can occur at the same time.

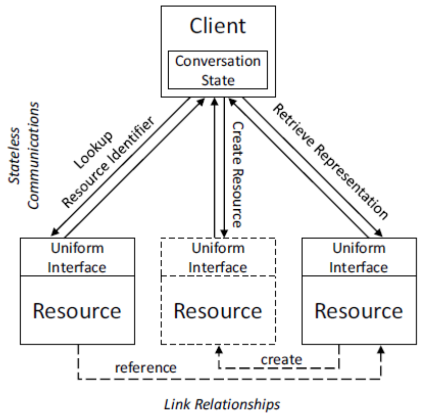

Figure 1: How a client can use data from different services. Font (Florian Haupt and Pautasso, 2015).

However, in this scenario, still according to (Florian Haupt and Pautasso, 2015), four examples of RESTful conversation types were collected: i) Redirect, ii) Accessing Resources Collections iii) Try-Confirm-Cancel, and iv) Long Running Requests. Each example has a different way to deal with the RESTful conversation.

In this context, another issue was identified: A model API which combines multiple basic conversation types. This means, in our perspective, how semantics can be used as a mediation support (repository) for multiple REST API conversations. Furthermore, this includes how semantics can be used to: i) Build, ii) Read, iii) Evaluate, iv) Compare and v) Enhance multiple REST APIs conversations.

2.6 Liquid Software

Despite being a technology mentioned at the end of the 90s (Hartman J.H et al., 1999), over two decades ago, we have seen no significantly enhancements in this method. However, one thing has changed, our behavior, since most of the time, we use at least two Internet-connected devices at same time.

When we expand this perspective for the available possibilities, such as laptops, smart-phones, tablets, phablets, game consoles, smart TVs, car display, watches, augmented reality glasses, digital cameras and photo frames, home appliances and so on (Andrea Gallidabino et al., 2016), we obtain a perspective of insufficient resources to deal with this amount of devices at the same time.

Despite the huge amount of connected devices, there is another constraint concerning the user-device experience. According to (Tommi Mikkonen and Pautasso, 2015), there are 3 main categories: 1) Sequential Screening, 2) Simultaneous Screening and 3) Collaboration Scenario.

Until today, despite all the available technologies, we are not able to obtain a simple data-flow without starting from zero each time. An example is while a person leaves a certain place, she/he decides to begin writing a message on a mobile device, but enters the car, so continues the action using speech resources through the car, then, when arriving at the final destination finishes the message on a laptop, desktop or a smart TV.

Currently, there is no fluid exchange of information through all platforms, connected devices and so forth. Protocols, messages and certificate standardization in order to achieve full data-flow through connected devices are still lacking.

3 ENGINEERING OF PRESENT IN WEB TECHNOLOGIES

The Web promises to change significantly in the next years regarding all the connected devices at our disposal. This requires not only changing the way how softwares will be built, but also how the Internet itself and its resources will deal with these great numbers of devices connecting with same or different behaviors (Pautasso, 2014).

On this subject, one point should be clarified: connected devices require an Internet connection to work, while IoT devices,do not need any Internet connection in some cases. Regarding the future of connected devices without an Internet connection, no assumptions can be made for now.

3.1 Layers Model

The architecture of current Web applications is currently not living up to the expectation of multiple devices with content, lacking new perspectives. It is known how the evolution brought us from OSI to TCP/IP, as displayed in the Figure 2. Therefore, a platform to deal with all connected devices and their behavior is necessary.

4 DISCUSSION

It is a fact that multiple devices are a growing trend. However, despite all efforts in the last years, much

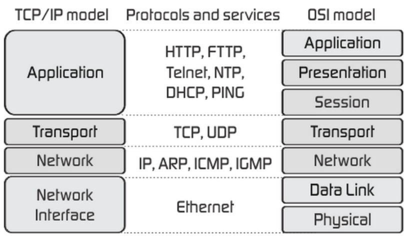

Figure 2: These structures could be enhanced in order to support multiple connected devices. Image source (http://www.whatisnetworking.net/tag/advantages-and-disadvantages-of-osi model/, 2017).

must still be developed to deal with this issue in a synchronized way.

The Internet was built to allow for caching information across its infrastructure, which can be helpful in many aspects. For example, a subsequent request for the same page along the same network path may be satisfied by a cached representation (Savas Parastatidis and Robinson, 2010). However, when we think of 1 or N connected devices to a user which needs to access her/his personal email, all this caching becomes useless.

When we think about computer-to-computer systems, a cost in terms of transactional atomicity and also scalability is present (Savas Parastatidis and Robinson, 2010) (Florian Haupt and Pautasso, 2015). Despite the efforts to build a system using Liquid Software principles (Hartman J.H et al., 1999) (Tommi Mikkonen and Pautasso, 2015) (Andrea Gallidabino et al., 2016), there is a lack of an architecture or framework able to deal with all the involved requirements, principles and constraints. Thus, its in this scenario where we propose our centric-based CUBE model, Figure 3.

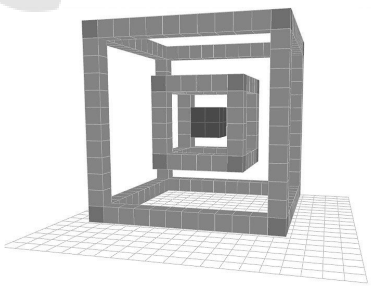

Figure 3: The user is in the center surrounded by devices (inner CUBE) and services (outer CUBE).

The model description follows this sequence: A Server (S1) requests an ordinary service (So). It needs to wait for the sum of the response time (t). The time can be influenced by the sum of the distance to each service (d), which can change according to the availability of routers. Once the Server (S1) has received the service (So), it keeps all data as cache. A second Server (S2), which needs the same data once requested by (S1), does not need to follow the entire path that (S1) followed, since all that (S2) requires is the data.

After the first request from (S2), if other Servers receive on-line requests of the same data from (S1), all resources used in the initial (S1) search are saved. This means lower energy consumption and throughput, among others. A comparison with a TCP-IP model, as displayed in the Figure 4, can project how CUBE would act.

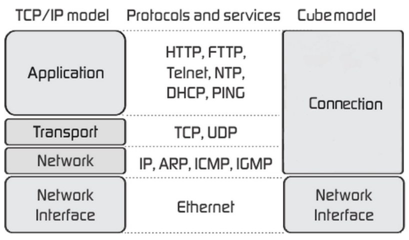

Figure 4: All devices could use the same data after acquiring.

It is important to realize that each Server (S1, S2,...Sn) is in fact a connected device owned by the same user. Since all the credentials, certifiers, headers and data in each layer of the TCP is already owned by the first Server (S1) there is no need to perform all these steps again, since the owner is the same.

In order to activate any new device in its own network, the user will need a trigger, such as a fingerprint or an encrypted password. To add this device, two methods can be used, combined. First, the graph methodology (Bulitko et al., 2008) of level search, which can begin at any arbitrary node to all adjacent nodes. Second, using the Dijkstra algorithm (Brandes, 2001) to search for the minimal distance from a node to another.

As for choosing a CUBE as the adequate model for the approach, a quadtree is applied (Samet, 1988), generally used to describe a class of hierarchical data structures whose common property, is thus based on the principle of recursive decomposition of space. According to (Samet, 1988), they can be differentiated by the following: 1) The type of data that they represent, 2) The guiding principle of the decomposition process, and 3) The resolution (variable or not).

Regarding the geometrical limitation of the CUBE, as an example, 8 different devices using 8 servers at the same time, two ways of dealing with this type of organization are available. The first is in the inner CUBE (devices), as the connection between vertices can provide a new device - volume approach, while the same principal can be applied to the outer CUBE (services). If technology limitations are present, another way to manage too many devices in the same user-network is duplicating the CUBE, which is possible using the Tridimensional approach of Architas of Tarento (Boyer, 1996) and (O'Connor and Robertson,).

We understand that the CUBE can be a model which could share restrictions or also act as a proxy in order to filter all Internet requests in the same network. This brings forward the concept of domestic networks and their profiles sharing the same services but at different levels of request.

As discussed, there are many current constraints which require attention, such as increasingly connected devices attempt to gain our attention. However, following the data-flow of these devices still requires a significant effort in order to bring forward a unique platform.

However, in order to use the best practices, a perspective of business process is required. While this approach deals with produced and consumed resources, a limited and linear vision of multi-connected systems is still prevalent. This scenario brings us to an edge which requires not depleting our resources, while, at the same time, providing all technology features.

5 CONCLUSIONS

Working with the same data-flow independently of the connected device and platforms is still an issue which must combine different technologies. It is under this concept that lie some principles of Liquid Software, which should be a tendency in a world which is more and more connected every day.

Thus, an effort is required in dealing with all the challenges of the Internet platform. In this context, a model able to converge multiple and heterogeneous environments with different behavior and complex systems is necessary.

This position paper has presented our CUBE model, applying the best practices of different technologies described herein. Once the concept is mature enough in our perspective, future steps will allow us to evaluate the enhancement of each technology in the cube vertices's and edges.

To the best of our knowledge, the main challenge is to assess whether the CUBE will be a model to a framework, an API or a new protocol able to deal with the increasing requests of connected devices.

REFERENCES

Andrea Gallidabino, C. P., Ilvonen, V., Mikkonen, T., Syst, K., Voutilainen, J.-P., and Taivalsaari, A. (2016). On the architecture of liquid software: Technology alternatives and design space. In *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE.

Boyer, C. B. (1996). *History of Mathematics*. Edgard Blcher ltda, So Paulo.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177.

Bulitko, V., Lustrek, M., Schaeffer, J., Bjornsson, Y., and Sigmundarson, S. (2008). Dynamic control in real-time heuristic search. *Journal of Artificial Intelligence Research*, 32:419–452.

Florian Haupt, F. L. and Pautasso, C. (2015). A conversation based approach for modeling rest apis. In *12th Working IEEE/IFIP Conference on Sotware Architecture*. IEEE.

Hartman J.H, B. P., P.G., B., A.B., M., R., P., O., S., T.A., P., L.L., P., and A.C, B. (1999). Joust: A platform for liquid software. In *IEEE Computer 32*. IEEE.

http://www.whatisnetworking.net/tag/advantages-and-disadvantages-of-osi model/ (2017). Accessed 26-01-2017. In *advantages-and-disadvantages-of-osi-model*. http://www.whatisnetworking.net/tag/advantages-and-disadvantages-of-osi-model/.

John, D. and M.S., R. Restdoc: Describe, discover and compose restful semantic web services using annotated documentations. In *International Journal of Web and Semantic Technology Vol 4, N 1, January, pages 37 to 49*.

Lanthaler, M. and Gtl, C. (2011). A semantic description language for restfull data services to combat semaphobia. In *5th IEEE International Conference on Digital Ecossystems and Technologies*. IEEE.

Mahdi Bennara, M. M. and Amghar, Y. (2014). Composing restful linked services on the web. In *WS-REST 2014*. Companion.

O'Connor, J. and Robertson, E. F. Archytas of tarentum. In *The MacTutor History of Mathematics archive*. http://www-history.mcs.st-andrews.ac.uk/Biographies/Archytas.html.

Panziera, L. and Paoli, F. D. (2013). A framework for self-descriptive restful services. In *International World Wide Web Conference IW3C2*. Companion.

Pautasso, C. (2014). Restful web services: Principles, patterns, emerging technologies. In *Web Services Foundation*. Springer Science+Business Media.

Richardson, L. and Ruby, S. (2007). *RESTful Web Services*. OReilly media and Associates, Sebastopol, 1nd edition.

Samet, H. (1988). *Theoretical Foundations of Computer Graphics and CAD - An overview of Quadtrees, Octress and Related Hierarchical Data Structures*. Springer-Verlag Berlin Heidelberg.

Savas Parastatidis, Jim Webber, G. S. and Robinson, I. S. (2010). The role of hypermedia in distributed system development. In *WS-REST - Proceedings of the First International Workshop on RESTful Design Pages 16-22*. ACM New York.

Tommi Mikkonen, K. S. and Pautasso, C. (2015). Towards liquid web applications. In *ICWE*. Springer International Publishing Switzerland.

Tuan-Dat Trinh, P. W., Do, B.-L., Kiesling, E., and Tjoa, A. M. (2015). Semantic mashup composition from natural language expressions: Preliminary results. In *iiWAS, Brussels, Belgium*. ACM.

Vinoski, S. (2008). Serendipitous reuse. In *IEEE Internet Comput. 12(1), 8487*. IEEE.