Optimally Storing the User Interaction in Mashup Interfaces within a Relational Database

Antonio Jesús Fernández-García[1], Luis Iribarne[1], Antonio Corral[1], Javier Criado[1], and James Z. Wang[2]

[1] Applied Computing Group, University of Almeria, Spain
[2] The Pennsylvania State University, USA
`{ajfernandez,luis.iribarne,acorral,javi.criado}@ual.es`
`jwang@ist.psu.edu`

Abstract. Cross-device applications that have user interfaces managed in multiple forms of interaction are prevalent. In particular, component-based (or *mashup*) applications are growing in popularity due to their easiness to build customized user interfaces with pieces of information from different sources. Since the user interaction on mashup interfaces can generate a large quantity of data, which can be useful to improving the interaction and usefulness of the application, it may involve the creation of cloud infrastructures to manage the dynamic distributed user interfaces within this context. Storing the generated data from the interaction performed over the user interface can be challenging. To achieve these goals, in this paper, a relational database for storing this interaction information generated on distributed user interfaces is proposed. Thus, user interaction over heterogeneous interfaces and devices described in detail, will be easily accessible for further analysis using machine learning and data mining techniques to offer a better user experience.

Keywords: Mashup, user interaction, multiforms of interaction, cross-device applications, relational database

1 Introduction

Today users consume information through heterogeneous devices such as computers, laptops, tablets or smartphones. Moreover, each device has a different way to interact with; some of them support classical forms of interaction by means of keyboard and mouse, others interact through touch interfaces, gestural interfaces or voice recognition (*Natural User Interaction*, NUI). Other interaction technologies are emerging, *e.g.*, virtual reality or wereables & IoT solutions.

Frequently, a same application needs to be available for multiple devices via different user interfaces (UIs). Users expect applications to be accessible via any device regardless of the screen size, the type of interaction, or the technologies involved in it. It becomes even more complicated when it concerns to the user configuration of the interface. Usually, UIs manage some configuration options and remember the behavior and the interactions performed by users and more features progressively.

Due to the increasing amount of services and APIs available, it is becoming a standard practice to use content from many sources in a Web application through a single UI. These UIs, commonly referred to as component-based UIs or *mashup*, allow users to easily customize their UI by employing different pieces of information or data creating their own tailored UI. Mashup interfaces [1] are typically used. Due to their granularity (coarse-grained) they facilitate the adaptation of their internal structure. A cloud infrastructure for the management of mashup UI can be a natural approach. In previous works a series of Web services, located in the platform-independent layer of a cloud infrastructure, have been created to support component-based architectures of mashup UI [2] [5]. These services include features such as managing users, component or sessions; and the administration of modules, controllers and databases that underlies below. This infrastructure provides a solid base to create dynamic UIs [4].

This paper focuses on the interaction of users over *mashup* interfaces. There is an extraordinary potential in analyzing the interaction performed in mashup interfaces to improve the user experience by adapting the interface at run-time to the users' requirements and even stepping to the users' needs. Using *machine learning* and *data mining* techniques over the interaction data acquired from users makes it possible to discover behavioral patterns and create prediction models. For that, it is necessary not only to acquire the data but also to know exactly the morphology of component-based UIs and to create an *optimized* relational database that can storage all the data for further analysis. Currently, there is no database schema proposal to store user interaction. The problem is not straightforward because there are many mashup UI and each one of them has a different purpose and their users have different domain knowledge, skills and expectations. Also, Web technologies are diverse and for that reason the data acquisition process that has to be implemented to store the interaction in the database should be independent of the technology used to develop the mashup UI, as well as not intrusive and totally transparent for users.

The rest of the paper is organized as follows. Section 2 describes the morphology of a basic mashup graphic user interface (GUI). Section 3 proposes a relational database to store the interaction produced over this type of interfaces. Section 4 shows a query to the information gathered in the database deployed in a real mashup. We conclude and provide future directions in Section 5.

2 Essential Mashup GUI Morphology

Mashup User Interfaces (mashup UI) are Web applications that integrate one or more components from one or more sources to create a unique UI that combines different components that might or might not have relationship among them. This section explains in detail how mashup UIs are composed, with focus on mashup GUIs. A standard interface that covers all the common aspects of mashups has been considered. There are many more features available in specific interfaces but all of them have some core elements and operations, which have been taken into consideration in this morphology definition.

There are many examples of commercial component-based interfaces. Nowadays, mashup interfaces (or component-based interfaces) are widespread in commercial software, particularly in Web applications [7]. Geckoboard is a KPI dashboard surface where users can visualize and work with their most important business data in real-time focusing on sales, marketing or operations among other features [8]. Cyfe allows users to build their own dashboard adding pieces of information through social media, analytics, sales, finance or project management components among others [9]. ENIA (Environmental Information Agent) is a mashup component-based GUI for environmental management used by the Andalusian Environmental Information Network (REDIAM) [11], a public organization that belongs to the Andalusian Regional Government (Spain) [10].

Figure 1 conceptually presents a component-based interface where the elements that form it are shown. Obviously, there may be many more elements and they could be positioned differently. All mashup UIs studied share some core elements and that is what is represented in Figure 1.

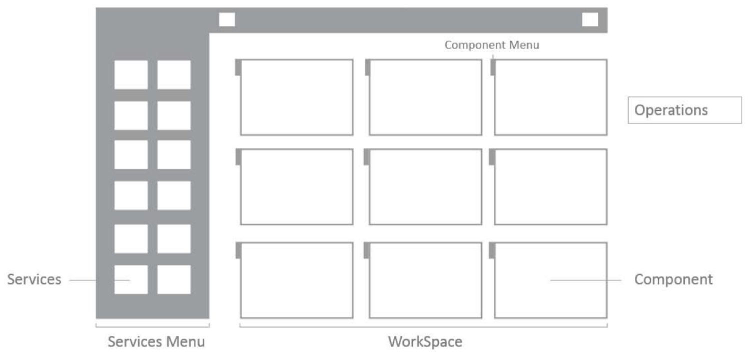


Fig. 1. Conceptual design of a component-based Web application.

Services. The capacities that the mashup application offers. They are available to users in order to operate with them. An instance of a *Service* is a *Component*.

Services menu. In this menu a list of all the *Services* available in the mashup application can be found. Users can navigate through this menu to find the services they may need. Usually, this menu is categorized and grouped by types of services and it has some search tools to locate them directly.

Component. When a user adds a Service to the *workspace* it is automatically transformed into a *Component*. A *Component* is a *Service* that is being used by a user at a certain moment in time. When a *Component* is instantiated, a set of attributes like width, height or position are assigned to it.

Workspace. The *Workspace* is the work area where users have all the *Components* (*Services* instantiated) they are working with.

Operations. All possible actions that are able to be applied over the *Components* such as resize, move or delete, among others.

Therefore, a mashup GUI ($\mathcal{M}$) is defined in the following manner: $\mathcal{M} = \{\mathcal{S}, \overline{\mathcal{S}}, \mathcal{C}, \mathcal{W}, \mathcal{O}\}$. Thus, M is comprised of a set of services $\mathcal{S}$, a service menu $\overline{\mathcal{S}}$, a set of components $\mathcal{C}$, a workspace $\mathcal{W}$ and a set of operations $\mathcal{O}$. The set of *services* $\mathcal{S}$ is defined as $\mathcal{S}$=$\{S_1, S_2, .., S_N\}$ where N is the number of *services* registered in the information system. The set of *components* $\mathcal{C}$ is defined as $\mathcal{C}$=$\{C_1, C_2, .., C_L\}$ where L is the number of *components* instantiated in the workspace $\mathcal{W}$. A concrete component C_i has some properties so it could be defined as $C_i = \{PosX, PosY, Width, Height\}$. Finally, the set of operations is defined as $\mathcal{O}$=$\{Add, Delete, Move, Resize\}$ and they are described below:

Add. Consists in adding a service to the workspace from the services menu, so it is instantiated into a component. When instantiating, some properties such as position in the x-axis, position in the y-axis, width and height are assigned to the component.

Delete. Consists in removing a component from the workspace. That happens mostly because it is of no use and users decide to dispense of it.

Resize. Consists in changing the size assigned to a component. It modifies the 'width' (w) and 'height' (h) properties. Sometimes the Resize operation can be decomposed in several operations such as:

$$Resize(x) = \begin{cases} x = ResizeBigger \mid (w_i * h_i) < (w_{i+1} * h_{i+1}) \\ x = ResizeSmaller \mid (w_i * h_i) > (w_{i+1} * h_{i+1}) \\ x = ResizeShape \mid (w_i * h_i) = (w_{i+1} * h_{i+1}) \\ \quad \wedge ((w_i \neq w_{i+1}) \vee (h_i \neq h_{i+1})) \end{cases},$$

where ResizeBigger operation is considered when the area covered after the operation is bigger; ResizeSmaller, when the area covered after the operation is smaller; and ResizeShape when the area covered is the same but the values of the properties are differents.

Move. Consists in changing the position of a component. It modifies the $PosX$ and $PosY$ properties. When $(PosX_i \neq PosX_{i+1}) \wedge (PosY_i = PosY_{i+1})$ the component has been displaced horizontally, when $(PosX_i = PosX_{i+1}) \wedge (PosY_i \neq PosY_{i+1})$ the component has been displaced vertically and finally, when $(PosX_i \neq PosX_{i+1}) \wedge (PosY_i \neq PosY_{i+1})$ the component has been displaced both horizontally and vertically.

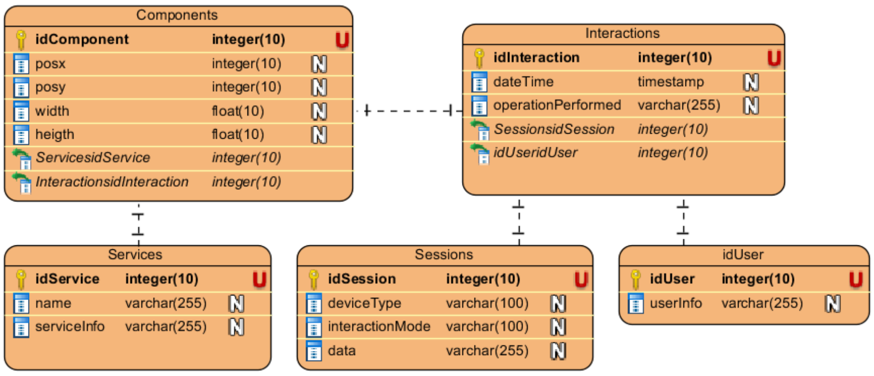
3 Database Design for Storing Interactions

When an interaction occurs in the mashup application, a data acquisition process will start and save all the information regarding the operation by the interaction. Together with the operation it is convenient to save the information about the user that generates the interaction as well as the component that is affected. It is also advisable to save the state that remains in the workspace after the operation. Although storing all the workspace might seem rather costly, it would make it possible to rebuild all the users' behavior step by step throughout the interfaces in case further analysis, not considered at design time, is required. That is why this option is viewed in this proposal.

In order to store data of the interactions that have been performed by users in the mashup UI, it is necessary to define a relational database model that would be able to store all the relevant information of the interaction. This relational database should be as complete as possible to have a good understanding of the interaction itself and the circumstances that surround that interaction. Figure 2 shows a proposal relational database schema that represents the interaction performed as well as the situation of the UI after it.

Each row of the *Interactions* table corresponds to an interaction taken by the user and the field *operationPerformed* saves the kind of operation performed. The *Interactions* table is related to the *Sessions* table, thus all the operations performed in the same session are grouped. The *Sessions* table has two important fields *deviceType* and *interactionType*. The first one storages the kind of device used in the session where the operations are performed; it can be a Tablet, a Laptop, a SmartPhone, Home Automation Systems or Smart Watches, among others. The second one storages the type of interaction used when performing the operation: mouse, keyboard, gesture, voice or presence, among other. Note that there can be many sessions working currently because one application can be use at the same time through more than one UI from the same or different devices or systems.

The *Users* table, which is related to the *Interactions* one, has information of all users registered in the application. Usually, there are a lot of users registered in most of applications, therefore, it is important to distinguish the operation performed by each one of them. Some information systems allow guest users to access to the system and, for those kind of users, there is normally a specific row in the *Users* table. Note that it would be necessary to obtain extra information about users that do not come with the interaction, hence it would be useful a request to Web services provided by the service, if any. In case the mashup UI has no users registered the *Users* and *Sessions* tables can be thrown out.



Components		
idComponent	**integer(10)**	U
posx	integer(10)	N
posy	integer(10)	N
width	float(10)	N
heigth	float(10)	N
ServicesidService	*integer(10)*	
InteractionsidInteraction	*integer(10)*	

Interactions		
idInteraction	**integer(10)**	U
dateTime	timestamp	N
operationPerformed	varchar(255)	N
SessionsidSession	*integer(10)*	
idUseridUser	*integer(10)*	

Services		
idService	**integer(10)**	U
name	varchar(255)	N
serviceInfo	varchar(255)	N

Sessions		
idSession	**integer(10)**	U
deviceType	varchar(100)	N
interactionMode	varchar(100)	N
data	varchar(255)	N

idUser		
idUser	**integer(10)**	U
userInfo	varchar(255)	N

Fig. 2. Database schema to storage interaction in a standard mashup UI

The *Components* table includes all components that populate the workspace after and interaction has been performed. With the information gathered in this table, it is possible to rebuild the workspace exactly as it was when the interaction was performed. The *posx, posy, width* and *height* attributes are enough to set each component in the workspace. Finally, the *Services* table, related to the *Components* table, has information about all the services that are registered in the Information Systems. As in the *users* table, it could be necessary, but not mandatory, to access to external Web services to obtain more relevant information about services that do not come with the interaction.

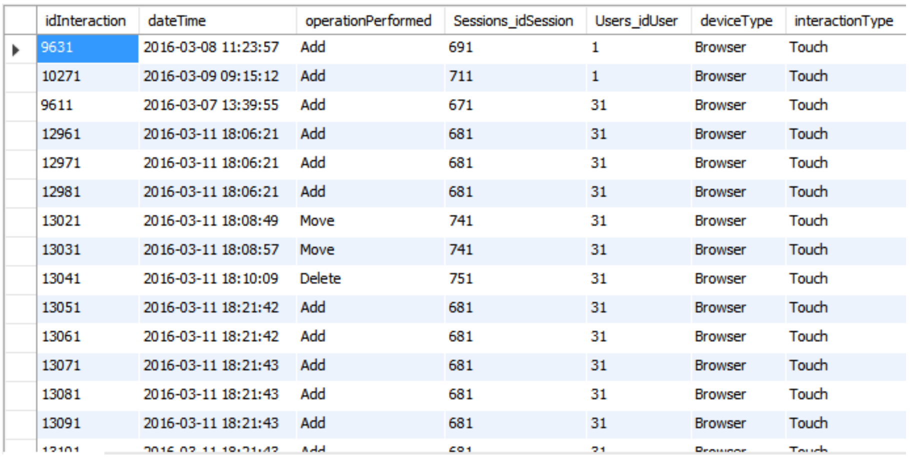
This database schema could seem rather costly due to the extensive resources consumption that may involve to store the workspace with all the components contained within. However, it is optimized in the sense that the database is expressive enough not only to generate datasets with rich data for further analysis, but for recreate user interaction step by step in case that more information about any aspect of the interaction could be detected as needed to infer a knowledge not contemplated when designing the database schema.

4 Database Behavior in a Real Mashup

Once the relational database schema proposed has been deployed in a database over a real environment, it is possible to access to all the interactions that have occurred with a great detail. In this case, we deployed the relational database in ENIA, the mashup interface previously described, which focuses on the management of environmental information. The real implementation of the database has more tables and the mashup interface has more operations compared to the set of fields and operations we have discussed previously in this paper. But, as a matter of fact, the operations and tables described are present. The next piece of SQL code queries a MySQL database deployed in a platform as a service cloud infrastructure provided by Azure. ClearDB provides the MySQL databases in Azure as database as a service. This query extracts all the operations that have been performed by users and sessions specifying in each case the kind of UI upon which the interaction was performed (browser, mobile browser, tablet app, smartphone app...) as well as the type of interaction used to perform the operation (mouse, keyboard, touch, gesture, voice...).

```
SELECT interactions.idInteraction, interactions.dateTime,
   interactions.operationPerformed, interactions.Sessions_idSession,
   interactions.Users_idUser, sessions.deviceType,
   sessions.interactionType
FROM interactions, sessions, users
WHERE interactions.Users_idUser=users.idUserClient AND
   interactions.Sessions_idSession=sessions.idSession
```

Figure 3 presents the data obtained from the SQL code shown before. We can be distinguish between add, move and delete operations. All of them have been performed in a desktop or laptop browser and the form of interaction has been made by touching the laptop or computer screen.



	idInteraction	dateTime	operationPerformed	Sessions_idSession	Users_idUser	deviceType	interactionType
▶	9631	2016-03-08 11:23:57	Add	691	1	Browser	Touch
	10271	2016-03-09 09:15:12	Add	711	1	Browser	Touch
	9611	2016-03-07 13:39:55	Add	671	31	Browser	Touch
	12961	2016-03-11 18:06:21	Add	681	31	Browser	Touch
	12971	2016-03-11 18:06:21	Add	681	31	Browser	Touch
	12981	2016-03-11 18:06:21	Add	681	31	Browser	Touch
	13021	2016-03-11 18:08:49	Move	741	31	Browser	Touch
	13031	2016-03-11 18:08:57	Move	741	31	Browser	Touch
	13041	2016-03-11 18:10:09	Delete	751	31	Browser	Touch
	13051	2016-03-11 18:21:42	Add	681	31	Browser	Touch
	13061	2016-03-11 18:21:42	Add	681	31	Browser	Touch
	13071	2016-03-11 18:21:43	Add	681	31	Browser	Touch
	13081	2016-03-11 18:21:43	Add	681	31	Browser	Touch
	13091	2016-03-11 18:21:43	Add	681	31	Browser	Touch
	13101	2016-03-11 18:21:43	Add	681	31	Browser	Touch

Fig. 3. Results from the query to the interaction db

This database allows to access to every operation performed in the UI from heterogeneous devices; it also enables to recreate the user behavior step by step by analyzing the *workspace*, as it has been later each operation performed, for a better understanding of the user's behavior.

5 Conclusions and Future Work

This paper proposes a relational database to allow mashup UIs to store the interaction performed by users over them. The database is valid even when the interface runs in distributed heterogeneous devices that support different interactions modes. A clear definition of a mashup GUI morphology has been made in order to study it and suggest a relational database that can save the interaction with accuracy.

The creation of a data acquisition process is proposed as future work. This data acquisition process could be a microservice than runs in the cloud and is continuously listening to the request from different mashup UIs distributed in multiple devices. It can just receive all the data directly from the client and store it or even make some requests to the mashup UI services, if any, to obtain extra information about users or components. Moreover, it can make a request to third party services that can provide valuable context information.

The information stored in the database proposed contains valuable information about the users' behavior. Machine learning experiments can be performed for the creation of an automatic learning system that can be used to offer a better user experience. The discovery of behavioral patterns gives the opportunity to create prediction models that assist users, providing them with the components they are most likely to need, including the shape, size and layout configuration properties they expects.

A future deployment of the data acquisition process that storage the interaction in the relational database proposed and the machine learning experiments can be used over the work shown at Criado *et al.* [3], where component-based interfaces are adapted at run time using model transformation according to a set of rules. The new rules generated can update the rules repository making the application autonomously evolve over time [6].

Acknowledgments. This work was funded by the EU ERDF and the Spanish Ministry of Economy and Competitiveness (MINECO) under Project TIN2013-41579-R and under a FPI Grant BES-2014-067974 and by the Andalusian Regional Government (Spain) under Project P10-TIC-6114. Wang has been funded by the US National Science Foundation.

References

1. Daniel F., Matera M. (2014) Mashups: Concepts, Models and Architectures.
2. Vallecillos J., Criado J., Padilla N., Iribarne L. (2016) A Cloud service for COTS component-based architectures. *Computer Standards & Interfaces*, Elsevier.
3. Criado C., Rodriguez-Gracia D., Iribarne L., Padilla N. (2015) Toward the adaptation of component-based architectures by model transformation: Behind smart user interfaces. *Software: Practice & Experience Journal* 45:1677–1718, 2015
4. Roscher C., Lehmann G., Schwartze V., Blumendorf M., Albayrak S. (2011) Dynamic Distribution and Layouting of Model-Based User Interfaces in Smarts. In: Hussmann H., Meixner G., Zuehlke D. (eds.) *Model-Driven Development of Advanced User Interfaces*. LNCS 340, pp. 171–197. Springer, Heidelberg.
5. Fernandez-Garcia A.J., Iribarne L. (2010) TDTrader: A methodology for the interoperability of DT-Web Services based on MHPCOTS software components, repositories and trading models. *2nd Int. Workshop of Ambient Assisted Living*, (IWAAL2010), pp. 83–88.
6. Fernandez-Garcia A.J., Iribarne L. Corral A., Wang J.Z. (2015) Evolving mashup Interfaces Using a Distributed Machine Learning and Model Transformation Methodology, On the move to meaningful internet systems, (OTM2015), LNCS 9416, pp. 401–410, Springer International Publishing.
7. Hoyer V., Fischer M. Market overview of enterprise mashup tools. *Service-Oriented Computing* (ICSOC'2008), LNCS 5364, pp. 708–721, Springer Berlin Heidelberg.
8. Geckoboard. Commercial mashup KPI dashboard, `https://www.geckoboard.com/`
9. Cyfe. Commercial mashup business dashboard, `https://www.cyfe.com/`
10. ENIA. Environmental Inf. Agent, `http://acg.ual.es/projects/enia/ui/`
11. The Andalusian Environmental Information Network (REDIAM), `http://www.juntadeandalucia.es/medioambiente/site/rediam`