Exploiting client logs to support the construction of adaptive e-commerce applications

Leandro G. Vasconcelos
National Institute for Space Research
1758 Av. dos Astronautas
Sao Jose dos Campos, Brazil
Email: le.guarino@gmail.com

Rafael D. C. Santos
National Institute for Space Research
1758 Av. dos Astronautas
Sao Jose dos Campos, Brazil
Email: rafael.santos@inpe.br

Laercio A. Baldochi
Federal University of Itajuba
1303 Av. BPS
Itajuba, Brazil
Email: baldochi@unifei.edu.br

***Abstract*—An essential feature of successful e-commerce applications is the ability to provide the right content at the right time for the user. Therefore, personalization techniques have been exploited to build adaptive applications in which the user interfaces change according to the user needs and preferences. In this work, we advocate that it is possible to achieve personalization by analyzing the behavior of the user when browsing an e-commerce application. In order to prove our hypothesis, we built a toolkit that allows the automatic gathering and analysis of client logs in real-time. Moreover, our solution provides a web service that exposes the analysis outcome also in real-time, thus allowing the client application to adapt on-the-fly according to the results provided by the toolkit. This paper presents a case study that demonstrate the effectiveness of our approach to support the construction of adaptive e-commerce applications.**

I. Introduction

An important feature of today's e-commerce applications is the fact that they must satisfy the requirements of thousands of customers, which present different needs, attitudes and idiosyncrasies. Underestimating the importance of satisfying the customer may affect the success of the application, leading to the loss of revenues.

Therefore, the success or failure of an e-commerce application is due to its potential to attract and retain visitors. Thus, studies to understand and predict user's behavior in Web applications has become increasingly important for e-commerce [5].

An essential feature of successful websites is the ability to provide the right content at the right time for the user [12]. In this sense, research on adaptive web applications have also become increasingly frequent. According to Velasquez and Palade [12], adaptive web applications are the next generation of web development.

Adaptive web applications usually analyzes the user behavior in order to identify her needs and interests and, based on this analysis, provide customized content. An efficient way to perform this analysis is using Web Usage Mining (WUM) techniques, as patterns extracted by analyzing logs usually provide useful information about the application usage [7].

The analysis of the user behavior when interacting with a web application provides insights that may lead to the customization and personalization of the user's experience. Therefore, e-marketing and e-commerce professionals have great interest in WUM [2]. For instance, clustering techniques have been used to group customers who behave similarly during navigation, making it possible to extract browsing patterns [3], [6].

The literature reports on several works in WUM that exploits logs collected on the server. Server logs present information regarding the resources accessed by the user such as pages, images and other files. Client logs, on the other hand, are collected on the user's browser, and provide more detailed information about the user interaction such as mouse movements, use the scroll bar and keyboard.

The volume of the client logs data is significantly larger than the volume of server logs, which discourages the analysis of client logs in the work reported in the literature. However, we advocate that client logs may be exploited to identify the user behavior, allowing applications to adapt in real-time.

In this work, we present an approach to identify the user behavior in real-time using client logs. Our approach focuses on the paths that a user navigates when executing tasks on e-commerce applications. We compare these paths to "optimal paths" recorded by applications experts. Therefore, we are able, for instance, to identify in real-time if a user is having trouble to perform a task. This way, adaptive applications may be built to support the user according to her personal needs and preferences.

Our approach relies on a web service in order to provide real-time information which e-commerce applications can consume in order to understand the behavior of the current user. Therefore, the application developer may exploit our services in order to write specific code that allows the desired adaptation. In order to support the construction of adaptive web applications based on our approach, we developed a toolkit called RUB – Real-time User Behavior.

Similarly to the RUB Toolkit, Google Analytics provides an API [8] that allows a web application to consume data in real time about the active user, the browser, the viewed pages, etc. Besides providing this same data, our toolkit also allows the web application to consume information concerning the usability of the application, such as user's wrong actions and the usability index of tasks. Abbar et al. [1] also presented an approach to real-time analysis, that is directed only for the

content, and it aims to recommend relevant articles to the user in real-time.

This paper is organized as follows: Section II presents the research that contributed to the development of the RUB toolkit; Section III details the toolkit architecture; Section IV presents a case study that demonstrates how e-commerce applications can take advantage of the features of our toolkit. Finally, Section V presents the conclusions and future work.

II. Previous Work

The World Wide Web presents a clear structural pattern in which websites are composed of a collection of pages that, in turn, are composed of elements such as hyperlinks, tables, forms, etc, which are usually grouped by special elements such as DIV and SPAN. By exploiting this pattern, and considering that interface elements are usually shared among several pages, we proposed COP [10], an interface model that aims at facilitating the definition of tasks.

The main concepts in COP are Container, Object and Page. An object is any page element that the user may interact with, such as hyperlinks, text fields, images, buttons, etc. A container is any page element that contains one or more objects. Finally, a page is an interface that contains one or more containers.

Besides exploiting the fact that containers and objects may appear in several pages, the COP model also exploits the similarities of objects and containers within a single page. In any given page, an object may be unique (using its id) or similar to other objects in terms of formatting and/or in terms of content. The same applies to containers: a container may be identified in a unique way, or it may be classified as similar to other containers, but only in terms of formatting.

The COP model was the foundation for the development of USABILICS [10], a task-oriented remote usability evaluation system. USABILICS evaluates the execution of tasks by calculating the similarity among the sequence of events produced by users and those previously captured by evaluators. By using USABILICS, evaluators may benefit from the COP model to define generic tasks, thus saving time and effort to evaluate tasks. The usability evaluation approach provided by USABILICS is composed by four main activities: task definition, logging, task analysis, and recommendations.

Task definiton. USABILICS provides a task definition tool called UsaTasker [11], which allows developers to define tasks by simply interacting with the application's GUI. UsaTasker provides a user-friendly interface for the management of tasks, where the evaluator can create (record), view, update and delete tasks. For recording a task, all that is required is to use the application as it is expected from the end user. While the evaluator surfs the application interface, she is prompted with generalization/specialization options, as specified by the COP model.

Logging. Our solution exploits a Javascript client application that recognizes all page elements using the Document Object Model (DOM) and binds events to these elements, allowing the gathering of user interactions such as mouse movements, scrolling, window resizing, among others. Events generated by the pages of the application, such as load and unload are also captured. Periodically, the client application compresses the logs and send them to a server application.

Task analysis. We perform task analysis by comparing the sequence of events recorded for a given task and the corresponding sequence captured from the end users' interactions. The similarity between these sequences provides a metric of efficiency. The percentage of completeness of a task provides a metric of effectiveness. Based on these parameters, we proposed a metric for evaluating the usability of tasks called the usability index [9].

Recommendations. USABILICS is able to identify wrong actions performed by end users and the interface components associated to them. By analyzing a set of different tasks presenting low usability, we found out that wrong actions are mainly related to hyperlink clicks, to the opening of pages, to the scrolling in pages and to the interaction with forms. We defined 6 recommendations for fixing these issues. An experiment [10] showed that, by following our recommendations, developers were able to improve the usability of applications significantly.

A restriction of the USABILICS tool was the inability to define and analyze tasks on mobile devices, such as smartphones and tablets. To fight this limitation, we developed a tool called MOBILICS [4], which extends the main modules of USABILICS in order to support the usability evaluation of mobile web applications.

The RUB toolkit leverages the results of our research in remote usability evaluation in order to provide real-time information regarding the user behavior in web applications.

III. The RUB Toolkit

An important challenge for the development of adaptive web applications is the fact that it is hard, if not impossible, to understand the interaction requirements of a large amount of users at design time, as user's needs and preferences evolve constantly.

Therefore, the approach of developing adaptive web applications that tries to foresee the user behavior is not an adequate approach. Instead, we advocate that a better approach is to allow the developer to monitor the user, adapting the application according to her behavior.

In order to support developers monitoring the user behavior, we propose a system called RUB – Real-time User Behavior. RUB logs and analysis users interactions in real-time, providing information concerning the user behavior back to the application. Therefore, it is possible to adapt the application to satisfy the requirements of the current user, while she is interacting with the application.

The RUB system extends USABILICS in order to analyze logs in real-time. Moreover, it provides a web service that allows developers to write applications that consume the resulting log analytics as well as the raw data logs generated by the current users. Therefore, it is possible to code adaptations triggered both by the analysis of a set of logs – for instance,

a low usability index for a given task – and by the occurrence of an specific event, such as the visit to a particular page.

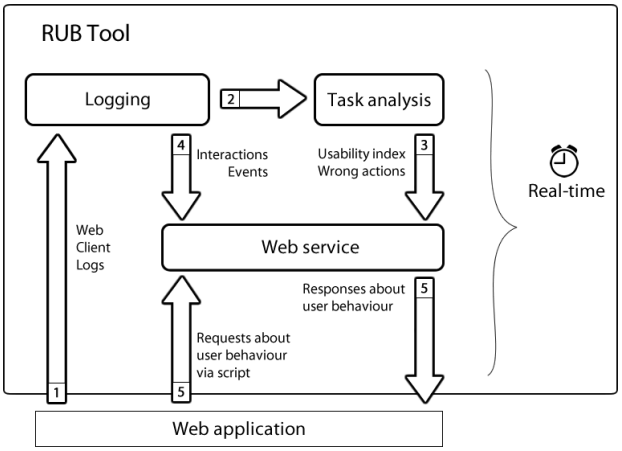

Fig. 1. The RUB Toolkit Architecture

Figure 1 presents RUB's architecture, which is composed by three main modules: Logging, Task Analysis and Web Service. The numbered arrows represent the flow of information among modules and between the web application and our system.

As it happens in USABILICS, the approach starts with the gathering of client logs in the web application, which are sent to the Logging module (flow 1). In order to support the affordances of real-time processing, the Logging module has been redesigned to store the logs in a graph database, which speeds up the log analysis. As soon as the logs are available in server-side, the recorded interaction events are made available both to the Web Server (flow 2) and to the Task Analysis modules (flow 3).

The Web Server module provides services that group the interaction events by user, by browser, by device screen size, etc. This way, it is possible to analyze the recorded events according to different criteria. The Task Analysis module process the available logs comparing them to the logs of pre-recorded tasks. When a match is found, i.e., the user data presents a sequence of logs that resemble the optimal path of a pre-recorded task, the usability index for that task is calculated and made available to the Web Service module (flow 4). The wrong actions found during the execution of tasks are also made available to the Web Server.

The Web Service module provides methods for programmatically accessing the services provided. Flow 5 illustrates the communication between the application and the RUB System. The following subsections detail the three main modules of our architecture.

A. Logging Module

The logging approach implemented in USABILICS exploits a Javascript application that collects user logs, compresses them and sends the compressed data to the server. In the server-side, this data is uncompressed and stored in a relational DBMS. This approach presents two drawbacks: (i) the latency to have the logs available for real time processing is prohibitive and (ii) a relational database presents scalability issues.

To fight the latency issue, our logging approach has been completely redesigned, both in client and in server sides. The client-side application was reimplemented using JQuery and JSON, making it even lighter than before. In the server-side, the usage of Node.js has allowed to process the incoming JSON much faster using its event-driven, non-blocking I/O model.

Considering the structure of the client logs gathered by the logging module, we decided to use a NoSQL database for graphs, called Neo4J. Our approach to fight the scalability problem consists in using two databases: one for storing data to be used in real-time and another to store the data generated by finished sessions. As soon as logs are made available by the Node.js application, they are kept in the real-time database. The data regarding a user session is kept in the real-time database until the end of the user interaction. A process that acts similarly as a garbage collector filters the old data periodically, moving them to the offline repository database. We plan to use log mining algorithms to find usage patterns in the offline repository.

B. Task Analysis Module

The analysis of end-users logs in USABILICS is based on tasks. The developer previously define the tasks for a web application and records the optimal path for their execution. After the deployment of the application, logs start to be collected and stored for further analysis.

This approach is effective in order to discover and fix usability problems. However, this is an asynchronous procedure that usually takes time – logs need to be collected during several days or even several weeks before a good evaluation is possible. After the evaluation, a low usability index and the wrong actions that caused the detected problems help the developer to fix the application.

We advocate that these inputs are also valuable to support adaptive web applications. For instance, the usability index may be used to detect novice users, i.e. users that are not familiar with the application. These users tend to present an exploratory behavior, moving the mouse and scrolling the page more times than it is usual for regular users. Whenever a novice user is detected, the application may, for instance, adapt its interfaces in order to provide help.

In order to leverage our task analysis procedure so as to support the analysis of logs in real-time, we developed an algorithm that calculates the usability index iteratively, pointing out wrong actions as soon as they are detected during the task analysis. Thus, our approach allows applications to inquire about the usability index at anytime, even before the end of a task.

Our algorithm exploits the premise that the first event of an optimal path of any given task is unique, i.e., it does not belong to the optimal path of other tasks. Therefore, when a task is being performed and the user executes an event that

Algorithm 1 Real-time task analysis

currentTask is the current task that user is performing
lastAccomplishedEvent is the last event of optimal path accomplished by the user
T is the list of tasks defined by the webmaster
usabilityIndex is the usability index of each task performed by the user
listWrongActions is the list of wrong actions detected for each task performed by the user

Initialization: When a new session is started in the web application

Input: the events gathered by Logging module
Output: the usability index and the wrong actions of each task performed by the user

```
1: usabilityIndex ← 0
2: listWrongActions ← ∅
3: currentTask ← null
4: lastAccomplishedEvent ← null
5: Initialize the event gathering
6: for each event e' gathered by Logging module do
7:    AnalyzeEvent(e')
8: end for
   =0
```

Procedure AnalyzeEvent(*e'*)

```
1: for each t' in T do
2:    if e' is the initial event of the optimal path of t' then
3:       if currentTask != null then
4:          SaveResultsOfTask(currentTask)
5:          Abort currentTask
6:       end if
7:       currentTask ← t'
8:       lastAccomplishedEvent ← e'
9:       break
10:   end if
11: end for
12: if currentTask != null then
13:   Calculate the similarity measure between e' and the next
      event of optimal path of currentTask
14:   Update usabilityIndex with the value of similarity mea-
      sure
15:   if e' is the next event of optimal path then
16:      lastAccomplishedEvent ← e'
17:      if e' is the final event of optimal path then
18:         SaveResultsOfTask(currentTask)
19:         Finalize currentTask
20:      end if
21:   else
22:      if e' is a wrong action then
23:         Append e' in listWrongActions
24:      end if
25:   end if
26: end if
    =0
```

Procedure SaveResultsOfTask(*currentTask*)

```
1: Save usabilityIndex of currentTask in the current interac-
   tion
2: Save listWrongActions related to task
   currentTask in the current interaction
3: usabilityIndex ← 0
4: listWrongActions ← ∅
5: lastAccomplishedEvent ← null
6: currentTask ← null =0
```

corresponds to the first event of another task, the current task is terminated and the analysis of a new task begin.

Our algorithm takes as input the events collected by the Logging Module while the end user surfs the application. Therefore, the algorithm starts whenever a new session is initiated.

The algorithm presents three main parts: (i) the main program, which executes everytime a new session is initiated; (ii) the procedure *AnalyzeEvent*, which analyses the events gathered by the Logging Module; and (iii) the procedure *SaveResultsOfTask*, which stores the usability index and the wrong actions performed during the execution of the task.

When a new session is initiated, the variables are adjusted to default values (lines 1 to 4). The gathering of events is then initiated by the Logging Module (line 5). The procedure *AnalyzeEvent* is called whenever a new event *e'* is logged (lines 6 and 7). This procedure checks whether the new event belongs to the task under evaluation, or is the first event of a new task. In the first case, the procedure calculates the similarity between this event and the expected event in the optimal path of the task (line 13 in *AnalyzeEvent*).

In the *AnalyzeEvent* procedure, if the event *e'* corresponds to the first event of a task and there is a current task under evaluation (*currentTask != null* on line 3), then it is necessary to finish the current task, as it was aborted by the user (line 5). In this case, it is important to save the user behavior while she was trying to perform the task, so the procedure *SaveResultsOfTask* is called on line 4 before aborting the task on line 5.

On line 13, the event *e'* is compared to the next event in the optimal path of the task. This comparison results in a similarity measure between 0 and 1. The usability index is then update with this measure in line 14. If *e'* corresponds to the expected event in the optimal path, it means that the user acomplished a step of the task, so we make *lastAccomplishedEvent = e'* (line 16). Otherwise, it may the case that *e'* is a wrong action (line 22). In this case, it must be added to the list of wrong actions (line 23). If *e'* is the last event of the task, then the procedure *SaveResultsOfTask* is called to store the usability index and the detected wrong actions (line 18).

C. Web Service Module

The Web Service Module provides endpoints to support the construction of adaptive web applications using RUB. As can

be noticed in Figure 1, this module is fed by the Logging Module (flow 3) and by the Task Analysis Module (flow 4). The Logging Module provides information regarding recent events and interactions performed by online users, while the Task Analysis Module supplies the analysis about the tasks performed by those users.

In order to reduce the number of requests to the web service, we implemented a library called *JUsabilics* which encapsulates the calls to the service endpoints. By using this library, non-volatile data may be stored in local objects, avoiding unnecessary remote calls. To benefit from JUsabilics, all the developer needs to do is to include calls to the library methods inside her code. When a method call is made requesting data that is not available locally, an asynchronous request is made to the web service, which responds sending data in the JSON format. Following, we present examples of the usage of our library.

1) Recommending links that have not been visited: Consider a website that sells online courses where each course is presented in a different page. To promote a specific Java course, it may be interesting to present a link to its page in the heading of the current page. However, it only makes sense to present this link if the current user has not already visited this page. The following script exploits the *isVisitedPageLike* method to provide this functionality.

```
jUsabilics.isVisitedPageLike('courses/soccer',
     function(data){
         if (data.visited == false){
               ('#divRecommend').show();
         }
     })
```

2) Checking the usability index of a task: The method *getCurrentUsabilityIndexByTask* provides the usability index for a task under execution. The following script executes an arbitrary procedure to deal with the problem that the usability for the current task is low. The number 123 is the ID of a task predefined by the evaluator.

```
jUsabilics.getCurrentUsabilityIndexByTask(123,
function(data){
    if (data.usabilityIndex < 0.5){
          //do something to improve the usability
    }
})
```

IV. Case Study

In order to demonstrate the effectiveness of our approach, we used the RUB Toolkit to improve the functionality of an e-commerce application that sells sports training courses. Our aim was to investigate if the toolkit was able to provide relevant information so that the website could be adapted in real-time, i.e., while the end user is performing a task.

Thus, we analyzed the main task of the application: buying an online course. This task consists of the following steps:

1) Select a course from the main menu to see its details.
2) Click the button *Buy*.
3) Select the payment method.
4) Fill a form containing the fields: name, address, e-mail, phone number, date of birth and additional information.

After filling out the form, a confirmation page is displayed to the user indicating the procedure for payment (PayPal or bank deposit). Payment is done outside of the website.

By analyzing this task, specifically the interface layout, we identified two possible usability problems:

- **Problem 1:** the position of the *Buy* button on the course details page is located at the bottom of the page and, thus, may be hard to find in order to perform the step 2 of the taks; and
- **Problem 2:** the lack of guidance in filling the form in step 4.

Therefore, we implemented interface adaptation rules that consume information in real-time in order to detect each of the presented usability problems.

For *Problem 1*, the implemented rule checks the usage of the scroll bar alternating the vertical direction (up/down). If so, the *Buy* button is repositioned in order to become easier to find. This was done exploiting the method *getEventsByPage* of jUsabilics Library, which was invoked programmatically at intervals of 500 ms. The script below illustrates this method call, which requires as a parameter the URL where the user is browsing and the type of event that should appear in the logs of the current user interaction. In the example, scrolling events were selected in order to identify the position of the bar in the various events.

```
jUsabilics.getEventsByPage(
     window.location.href, "scroll",
     jUsabilics.interaction.id, function(data){
        if (data.length > 0){
           //if the user is changing the scroll direction
        }
})
```

For *Problem 2*, the implemented adaptation rule checks the usage of the scroll bar while the user is filling up the form. If the user goes to the bottom of the page and then returns to the top, a help message is presented, guiding the user in the filling of the form.

A. Recruitment of users

For this experiment, we recruited 10 inexperienced users on the evaluated e-commerce application, i.e., users who have not ever used the application. The age of the users are between 24-39 years and they have different levels of education, from high school to the graduate level.

All users received the same instruction: use the website in order to buy a specific course. The choice of a specific course was made so that users have a target in performing the task. Users were not seen presentially and they performed the tasks on their personal computers. Using the RUB Toolkit, we collected the logs of user interactions for later analysis.

B. Discussion

During the the execution of the ten tasks by the recruited users, the usability index was calculated in real-time and the wrong actions were identified. The hypothesis of *Problem 2* was not found in any of the ten executions, i.e., no user had difficulty to fill out the form. However, the *Problem*

1 was identified in six task executions and, consequently, the adaptation rule was triggered in real-time, displaying the *Buy* button more properly. In four tasks, users completed the purchase without the need of interface adaptation.

The aim of adapting the interface in real-time is to assist the user during the execution of the task. Therefore, the adaptation of the interface occurs after the identification of any difficulty in interaction. Thus, the adaptation must improve the usability of the interface for the current user.

In order to verify the effectiveness of the adaptation rules for improving the usability of the application, it is worth comparing the amount of wrong actions and the value of the usability index in the tasks in which the adaptation rule was triggered and in those in which no adaptation occurred. In our experiment, six of the ten users found difficulties in executing the task, thus triggering the adaptation.

Table II shows the average amount of *mouseover* and *scroll* events detected on user interactions. These events are commonly performed by users when looking for an element in the interface. The table also shows the usability index of the recorded task. For the task executions without interface adaptation (four cases), the usability index was 76%, and the average number of *scroll* events was 20,7. For the six task executions with interface adaptation, the average amount of *mouseover* events and the average amount of *scroll* events were higher, however, due to the adaptation, the usability index was 75% – only 1% less than the executions without adaptation.

These results show that the adaptation rule implemented for the first problem identified the wrong actions (*mouseover* and *scroll* events) in sufficient time to trigger a relevant adaptation in real-time. Task executions with adaptation presented about 45% more *mouseover* and *scroll* events. However, as soon as the usability problem was identified and the adaptation was performed, the number of wrong actions declines significantly. More importantly, the value of the usability index is almost equal in the two cases, showing that the adaption is able to improve the usability when the user is having trouble to perform a task.

TABLE I
COMPARISON OF INTERACTIONS

Without adaptation			**With adaptation**		
MouseOver	Scroll	Index	MouseOver	Scroll	Index
79,2	20,7	76%	114,6	30,3	75%

V. CONCLUSION

This paper presents the RUB Toolkit, which gathers web client logs, analyzes them in real-time and provides information about the current user behavior in order to support the construction of adaptive Web applications.

We detailed the RUB Toolkit architecture and also presented the algorithm that executes the analysis of user interaction logs in real-time, which makes it possible to perform remote and automatic usability evaluation based on task analysis.

In order to provide web applications with real-time data concerning the user behavior, RUB provides a JavaScript library called jUsabilics. This library contains predefined functions that send requests to a web service, making it possible for applications to consume (i) the wrong actions identified during the task executions; (ii) the usability index of a particular task performed by the user, and (iii) the user actions.

In order to demonstrate the effectiveness of our approach, we conducted a case study on an e-commerce application. Our study presents the results of an adaptation triggered in real-time in order to assist users to perform the task of buying an online course. The study shows that this adaptation reduces the number of wrong actions and improves the usability index when the user is having trouble to perform the task.

The RUB toolkit contributes to the efforts of understanding the user behavior by analyzing client logs. Therefore, it is a valuable tool to improve the user experience in e-commerce applications.

In future work, we intend to incorporate Web Usage Mining algorithms to the RUB Toolkit in order to detect patterns of behavior based on the usage history.

REFERENCES

[1] S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Real-time recommendation of diverse related articles. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1–12, New York, NY, USA, 2013. ACM.

[2] S. R. Aghabozorgi and T. Y. Wah. Recommender systems: Incremental clustering on web log data. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 812–818, New York, NY, USA, 2009. ACM.

[3] M. Eirinaki and M. Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, Feb. 2003.

[4] L. F. Gonçalves, L. G. Vasconcelos, E. V. Munson, and L. A. Baldochi. Supporting adaptation of web applications to the mobile environment with automated usability evaluation. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 787–794, New York, NY, USA, 2016. ACM.

[5] S. Gunduz and M. Ozsu. A poisson model for user accesses to web pages. In *Computer and Information Sciences - ISCIS 2003*, volume 2869 of *Lecture Notes in Computer Science*, pages 332–339. Springer Berlin Heidelberg, 2003.

[6] A. Joshi, K. Joshi, and R. Krishnapuram. On mining web access logs. In *In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 63–69, 2000.

[7] D. Pierrakos, G. Paliouras, C. Papatheodorou, and C. D. Spyropoulos. Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13(4):311–372, Nov. 2003.

[8] G. A. API. Google analytics API. https://developers.google.com/analytics/devguides/reporting/realtime/dimsmets/?hl=pt-br, 2016.

[9] L. G. Vasconcelos and L. A. Baldochi. USABILICS: remote usability evaluation and metrics based on task analysis (in portuguese). In *Proceedings of the 10th Brazilian Symposyum on Human Factors in Computer Systems & 5th Latin American Conference on Human-Computer Interaction*, pages 303–312, 2011.

[10] L. G. Vasconcelos and L. A. Baldochi, Jr. Towards an automatic evaluation of web applications. In *SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 709–716, New York, NY, USA, 2012. ACM.

[11] L. G. Vasconcelos and L. A. Baldochi, Jr. Usatasker: a task definition tool for supporting the usability evaluation of web applications. In *Proceedings of the IADIS International Conference on WWW/Internet 2012*, pages 307–314, Madri, Spain, 2012. IADIS.

[12] J. D. Velasquez and V. Palade. *Adaptive Web Sites: A Knowledge Extraction from Web Data Approach*, volume 170. IOS Press, 2008.