ArchCollect: An architecture for collecting, transforming, loading and displaying web users interactions

Joubert de Castro Lima, Tiago Garcia de Senna Carneiro, Rodrigo Martins Pagliares, Júlio César Ferreira and João Bosco Mangueira Sobral.

joubert@facic.fuom.br - FACIC/FUOM
tiago@dpi.inpe.br
pagliares@yahoo.com.br - FAS/UNIPAC
bosco@inf.ufsc.br - Federal University of Santa Catarina

Abstract

This paper describes the design of an architecture called ArchCollect, a set of software components used for collecting, transforming, loading and displaying web users interactions on web applications. Seven Java components gather information coming only from the user, independent of the web application that will be monitored and of the web server used to support it. This improves the portability of this architecture and its capacity to deal with many web applications at the same time. The ArchCollect relational model provides analyses, regarding factors such as purchases, business results, the length of time spent to serve each interaction, user, process, service or product. In this architecture, data extraction and the data analysis are performed either by personalization mechanisms provided by itself, or by commercial decision making tools, such as, OLAP, Data Mining and Statistics, or by both.

***Keywords*:** Interaction pattern, web users interactions, components

1. INTRODUCTION

Interaction, in our scope, is a general term used for classifying specific events that were emitted by users in any sort of application. These events are classified by clicks on elements on a page of an application. These elements are buttons, links and banners, the last one used particularly for commerce applications.

In the electronic commerce, the users interactions analysis area has been largely studied [1, 2, 3, 4, 5, 6, 7, 17]. There are many commercial tools available for this area [8, 9, 10, 11, 12, 13, 14, 15, 16]. All these tools extract the initial data from the web server log files and, as [2] explains, the majority of the commercial tools typically keep count of hits, rank the most popular pages requested and tell where the user came from, the length of time each page was viewed and the page from which the user entered the site and from which the user exited.

The purpose of this article is to show an example of an architecture that has low coupling to the monitored application and an architecture that possibilities future experiments, once the basics ideas has been implemented and tested.

We obtain low coupling once we collect all the information necessary to the ArchCollect direct from the web client. The ArchCollect ***user component,*** inserted into the HTML or XML code of the existent application, collects all the necessary information and sends it to the ArchCollect ***collecting component.*** This last component loads this information in an intermediate text file called semantic log file.

Once the necessary information has been stored into the text file, the ArchCollect ***transformer component*** filter them and call the ArchCollect ***loader component*** to store this filtered information into a set of tables called the ArchCollect relational model. In these relational model we have the final interaction pattern, ready to be used.

The data extraction and the data analysis are performed either by personalization mechanisms provided by the architecture itself, or by commercial decision making tools, such as, OLAP, Data Mining and Statistics, or by both.

The rest of this article is structured as follows: In section 2 the related works are emphasized and compared to the developed architecture. Section 3 explains, with details, the ArchCollect architecture components, their low coupling to the existent application and the Archcollect architecture relational model. In section 4 the experiments are characterized. Section 5 presents the experiments results. Finally, in section 6 the conclusions are made and the future works are proposed.

2. RELATED WORK

Some works, such as [1, 2, 3, 4, 5, 6, 7, 17], were proposed to deal with data extraction under many different perspectives, in other words, to illustrate analyses offered to administrators (*Sites Modification*, *Systems Improvement*, *Business Intelligence*) and offered to the application users (personalization).

Such projects possess as mechanism or internal logic, the possibility of storage of the data collected in data structures, typically graphs, or in relational models consisting of some tables. Projects as ECI, WebLogMiner use relational models and tools with OLAP services, Data Mining and Statistics for extraction of the stored data. Projects as Shahabi use graphs where each node is a page and each track (graph arrow) corresponds to one link clicked by the user. The overlapping of the graphs generates the similarities. Projects as WUM bring in each node of the graph a set with the users who had passed in the page in question. Similar paths are identified from such mechanism.

A procedure for analyzing and describing the difference among the works [1, 2, 3, 4, 5, 6, 7, 17] is described by six parameters proposed in [3].

1. Number of users they stand: Many users at the same time. This characteristic is found in all referred projects.

2. Number of applications they monitor: again, the projects quoted above analyze a unique monitored application, allowing a unique analisys architecture for each monitored application. High coupling is a consequence in all works on table 1, except on *Shahabi*[6] project.

3. Application focus: this analysis parameter is unique. Only projects that have general purposes were analyzed. It is necessary to make clear that there are projects with more specific purposes divided in areas such as personalization, sites modification, systems improvement and business intelligence.

4. Data source: data for analysis may come from many different sources such as the server, the client or the proxy. Projects such as WebSifit, SpeedTracer and WUM obtain their data from the servers. The Shahabi project obtains its data directly from the client and the ECI architecture analyzes data coming from both sources.

5. Usage data: which data categories were obtained for analiys? Projects such as the WebSifit works with data called usage data, content data and structure data. The SpeedTracer project works only with usage data. The Shahabi project, as well as the ECI architecture and the WUM project work with usage data and structure data.

6. Emphasized subjects: the projects above emphasize points such as the user, page, links, page layout, time and session.The ECI architecture emphasizes besides these points, business results and purchases, allowing a model that is richer in information.

Table 1 summarizes each work's characteristics by using the criteria proposed by [14]

Project	**User**	**Coupling (App. Monitor)**	**Applic. Focus**	**Data source**	**Usage data**	**Emphasized subjects**
WebSifit	Multi	High (unique application)	General	WEB Server	Usage/ Content/ Structure	-----
ECI IBM	Multi	High (unique application)	General	WEBServer/ User	Usage/ Structure	Purchases and business results
SpeedTracer	Multi	High (unique application)	General	WEB Server	Usage/ Structure	----
WUM	Multi	High (unique application)	General	WEB Server	Usage/ Structure	-----
Shahabi	Multi	Medium (unique application)	General	User	Usage/ Structure	-----
ArchCollect	Multi	Low (multi application)	General	User	Usage/ Structure	Purchases, business results and performance.

Table 1. Related works and its characteristics.

3. ARCHCOLLECT AND ITS COMPONENTS

The ArchCollect is composed by seven components as shown in Figure 1. On each user interaction, the ***user component***, that is inserted (cut-and-paste) into the HTML or XML code of the existent application, collects user and the interaction relevant information and sends them, in a HTTP request, to the ***duplication component***. This latter component must be installed in the port of the host whose IP address or DNS name is known by the web users. This component purpose is to receive each HTTP request from the users, send it to the application that is been monitored, and if it is a relevant interaction, it must

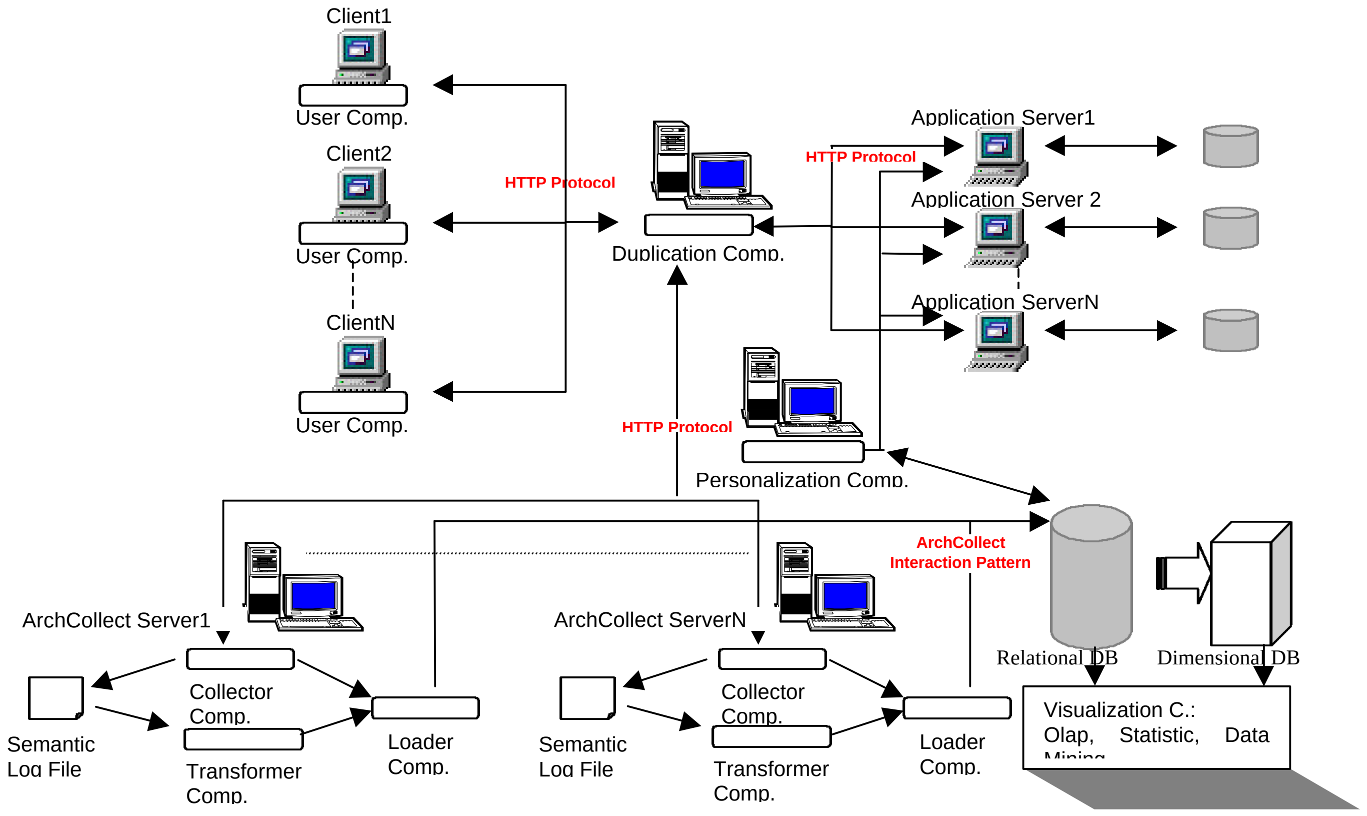


Figure 1 :Operational environment showing one possible deployment of the ArchCollect architecture.

duplicate the request and send it to the ***collectig component***. If there are no ArchCollect cookies in the requests, the ***collecting component*** presumes that the interaction is coming from a new and unknown web user, then it adds this user into the database, creates and assigns new cookies to him/her, and sends a HTTP response to the duplication component. In all interactions the ***collecting component*** stores the received information into the semantic log file. The duplication component finishes its work when it joins, into a unique HTTP response, the HTTP response that has come from the monitored application with the one that has come from the ***collecting component***, and finally sends it to the user browser. Neither the client side of the application, nor the server side needs to be modified for the application to be monitored, and this way the ArchCollect portability is improved.

The ***transformer component*** converts all the information stored in the semantic log file into a relational model that has granularity defined as users unique interactions. This component also identifies the interactions belonging to the same session of an user.

The ***loader component*** implements the persistence layer between the ArchCollect logical model and the relational database. The stored data may be presented by ***visualization components*** such as, OLAP, data mining and statistics tools, and may be used by the ***personalization component*** responsible for establishing the user profile and customizing the web content according to it.

Figure 1 shows that the duplication component may interact with many ArchCollect servers and many monitored applications servers, what improves the architecture scalability. If there are too many interactions per seconds, a cluster of ArchCollect servers may be used to support this workload and the duplication component may share the workload among the servers using a round-robin schedule algorithm. Remember that a ***duplication component*** can be installed regardless of the number of necessary machines for the simulation.

3.1. USER COMPONENT

Every element in a web page has its name and its identifier. The user component defines these two properties or, if necessary, other properties to give semantic meaning to the interactions. This last situation can happen in monitored applications implemented with XML elements. The element name is the same name that already exists in the application and will be used as the interaction name. The element identifier is associated to a product, a service or a process. It is optional, in the case where the element is not associated to any of these items.

Besides these page elements properties, the user component collects the interaction complete date and time so that the local time where the interaction took place can be obtained. For specific elements in commerce applications, the user component also collects the price and the quantity related to the product, service or process that has been acquired. The page where the element has been clicked is also collected. At last, we collect the client cookies and some semantic issues, if they exist.

The information is stored in a text element in each page and sent together with each request in the ArchCollect interaction pattern showed in Figure 2.

```
pattern ::= date“+”page“+”element[“+”product]“+”session”+”semantic
date    ::= day“+”month“+”year“+”hour“+”minute“+”second
element ::= name“+”id
product ::= quantity“+”price
session ::= userIP[[“+”sessioncookie]“+”persistentcookie]“+”entranceID
semantic ::= age association”+”rent association”+”layout association”+”etc.
```

Figure 2. ArchCollect interaction pattern.

3.2. DUPLICATION COMPONENT

The ***duplication component*** must be operating on the same host(s) and port(s) where the original web server should be running. Thus, all HTTP requests sent to the original application are firstly received by this component and replicated to the web server which should have been moved to another host(s) or port(s). In other words, the duplication component implements the *Proxy* pattern[18].

The first task of this component is listening to the port(s), already predefined, looking for users interactions. When a request arrives at the duplication component, it sends an identical request to the original web server and a request with its first line modified to the ArchCollect server. This line is modified so that the collector component is instantiated in the ArchCollect server. The duplication component can deal with applications that use both, GET and POST HTTP methods.

Unfortunately, this component will also receive all the requests from all the users and not only the interactions that are relevant to the ArchCollect. Because of this, the component first filters and duplicates only requests related to the application services, for example, ASP, JSP, CGI, PHP pages.

The ***duplication component*** also registers the elapsed time between the moment when a request arrives and: 1) the moment when the application response arrives, 2) the moment when the collector component response arrives, and 3) moment when the HTTP response has just been sent to the user browser.

This structure has enabled measuring the time an interaction has spent on each server, and thus, it also enabled to relating each interaction aggregated value to its cost in terms of consumed computational resources. This information is stored in the relational model.

3.3. COLLECTING COMPONENT

The tasks proposed for the ***collecting component*** are:

1) Identifying the user, i.e., verifying if the ArchCollect cookies do exist on the HTTP request
2) If the ArchCollect persistent cookie does not exist:
   a. the collector component assumes that this request came from a new user, whose information is added to the relational database, as well as the HTTP request header information, via the *loader component*;
   b. the session and persistent cookies are created, and the *session.*`entranceID` field of the interaction pattern is set to “1”
3) If the ArchCollect session cookie does not exist, but the persistent cookie does exist:
   a. the collector component assumes that this request came from a new session, from a known user, and then the session cookie is created, and the session.entranceID field of the interaction pattern is set to “1”
4) If the ArchCollect session and persistent cookies does exist:
   a. the collector assumes that this request came from a known session and known user, and the session.entranceID field of the interaction pattern is set to “0”
5) storing the interaction pattern in the semantic log file
6) sending the current cookies to the duplication component.

3.4. TRANSFORMER COMPONENT

Because user information needs to be stored in a relational database, a specialized component is required for some specific activities such as extracting, transforming and loading data from the semantic log file. This component has to identify the data about unique users sessions so that they can be stored into the ArchCollect relational model.

The first task is to process the data from semantic log file, storing each line of the file into the relational database using the *loader component*. When the end of the file is reached, a mark is stored, so the transforming process can continue from that mark after a period of time, sufficient for the addition of new registers to the log file.

The second task is to identify the session. Using a simple rule, it is possible to establish the information of the user entrance in the application: if the field *session.id* from a line in the interaction pattern is set to '1' then it is that line that corresponds to the entrance page of the user session.

To find the last interaction of each user session, i.e., the point where the user exits the monitored application, it is necessary to process the whole semantic log file, looking for the register that has the most recent date in the session. This task of finding the exit register, which stops at the end of the log file, stores the register that has the most recent date and waits for a predefined fraction of the timeout value. This task will happen again as many times as necessary to end the session. All these values (waiting value and timeout value) depends on the monitored application workload.

3.5. LOADER COMPONENT

The ***loader component*** has the purpose of receiving the information from the ***transformer component***, from the ***duplication component*** and, in some cases, from the ***collecting component*** and store it, according to the ArchCollect architecture final pattern of interactions, in the relational database.

Considered a persistence layer, the ***loader component*** allows the integration between the ArchCollect logical model and the relational model. The architecture business layer is separated from the transaction layer resulting in low coupling among the architecture modules, achieving better performance, once such component is instantiated on demand and with better transactional integrity.

3.6. VISUALIZATION COMPONENT

Once the interactions are correctly stored in the relational models, it is necessary to define some extration tools that answers business questions.

The relational model was translated into a dimensional model. The solution is based on the separation of the data that is used for decision making, called dimensions, from the operational data, called fact table. This kind of data is stored at data webhouses.

The data webhouse is responsible for data storage and management, while the OLAP services, for example, convert the stored data into useful information for decision making [19].

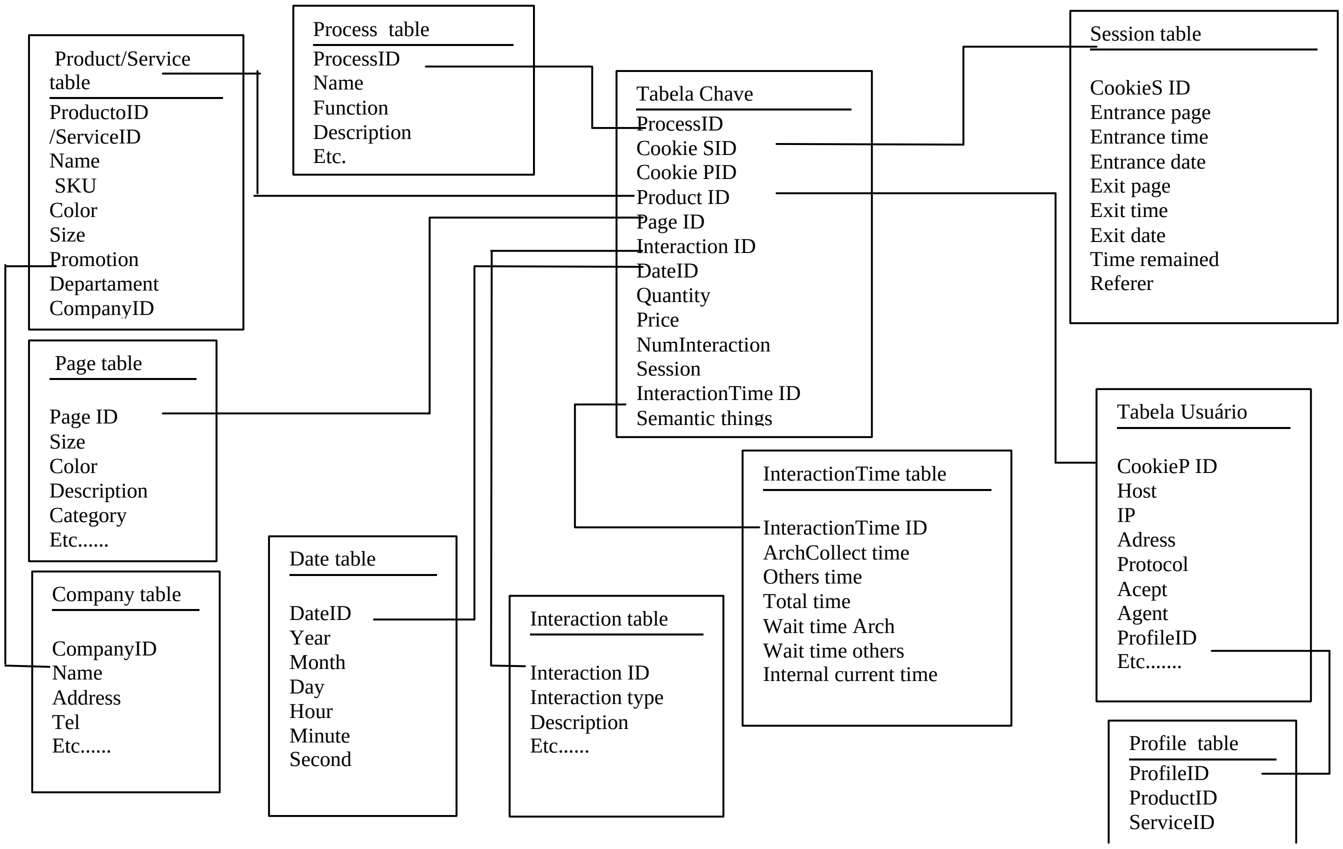
The ***visualization component*** can be implemented in any data webhouse tool and OLAP or Data Mining or Statistic services off-the-shelf.

3.7. PERSONALIZATION COMPONENT

Web commerce applications deal with anonymous users who arrive, emit interactions and leave the application. Important information with grain of granularity in the order of the users interactions are obtained by the components that were described until now. This strategic and behavioral information have been passed online to the application managers, and then re-passed to the users through modifications in the application contents. Also, in order to re-pass these modifications online to the commerce application user, it was proposed the ***personalization component***.

The ***personalization component*** task is to generate, on the begining of it´s execution in background and periodically, the inicial profiles. The interaction pattern stored into the relational model is used to establish the inicial profiles.Once this task is done, the online process begins.

The second task is the profile sofistication, assuming new ones and increasing the existing ones. To do this the



personalization component operates in conjuction with the monitored application. The request received is repassed to the personalization component. The interaction pattern ilustred in figure 2 has a subpart called semantic. Based on this subpart the online profile is sofisticated.

It is not the intent of this article to detail this component, once the ideas are not yet implemented and tested. Problems with performance in the second task persists. We based our ideas in the ***Yoda***[20] project. Once this component is ready it will require an individual work.

3.8. RELATIONAL MODEL

The relational model in Figure 3 is considered the ArchCollect architecture core, wich reflects the ArchCollect architecture final pattern of interactions.

It was proposed a model for two types of applications that exist in the market. One type reflects e-commerce and e-business applications. In the model features such as purchases, business results and the elapsed time related to each interaction are emphasized. For the other type of corporate interactions it is relevant to know features such as process, sales, purchases, data updates and elapsed time related to each interaction. So, the architecture is adequate to innumerous contexts. For the general context of interactions it is relevant to know subjects such as date, user, page, session and at last the interaction itself.

The pourpose of this article is to describe an example of an interaction analisys architecture and no metrics was used to establish the relational model. We just emphase general subjects that all corporates will be interested on. On future works a detailed study will be done classifing the set of metrics that the ArchCollect arquitecture will adopt.

4. EXPERIMENTS CHARACTERIZATION

The system was analyzed with an experiment composed by three computers that are interconnected by a hub Ethernet 10/100Mbps. One of the computers executes the benchmark Microsoft Web Stress Tool that implements many threads that continuously, send POST requisitions on ASP pages – Active Server Pages of a web application stored in another computer that executes the IIS server – Microsoft Internet Information Server 5.0.

The third computer executes all the ArchCollect architecture components and the JWS server – Java Web Server 2.0 used by the architecture ***collecting component*** for servlet instantiation. The Table 2 shows the configuration of each one of the computers.

COMPUTERS	Web Stress Tool	IIS	ArchCollect
Processor	Pentiun II, 350 MHz	Pentiun II, 350 MHz	Athlon , K7 650 MHz
Memory	128 MB SDRAM	128 MB SDRAM	256 MB SDRAM
Hard Disk	IDE,7200 rpm	IDE,7200 rpm	IDE, 7200 rpm
Network	Ethernet 10/100 Mbps	Ethernet 10 Mbps	Ethernet 10/100 Mbps
Operational System	Windows NT 4.0	Windows 2000 Professional	Windows NT 4.0

Table 2. Computers used in the experiments.

The experiments where made in two scenarios. In the first scenario, only two computers are connected to the hub and the Microsoft Web Stress Tool generates HTTP requests directly to the IIS server with no interference of the ArchCollect architecture. In the second scenario, the three computers are used and the Microsoft Web Stress Tool generates HTTP requests to the ArchCollect architecture server that collects and analyzes the requests and then, re-pass them to the computer that executes the IIS server.

4.1. IDENTIFICATION OF THE SERVICES

The services that were implemented by the ArchCollect architecture are identified below:

- Service 1. Collecting Interactions: this service consists of collecting performance and semantic information about each web user interaction. The total time elapsed for an interaction to be generated, collected and for the response page to be presented to the user includes the *think time*[1] plus the duplication response time[2] plus network delay. The duplication response time is composed by the monitored application and collector component response times.
- Service 2. Transforming interactions: This service is defined as a continuous reading of the semantic log file. The aim is to get each user interaction, identify the beginning and the end of each user session and the subsequent insertion of the information related to each session or interaction in the relational database of the ArchCollect.
- Service 3. Establishing user profile: This service consists in the establishment of the profile for each user, based on the interactions of this user with the system. This service has not been analyzed in our experiments.
- Service 4. Building response page according to the user profile: This service includes the configuration and the dynamic generation of the response page requested by the user based on its profile. This service has not been analyzed in our experiments.
- Service 5. Presenting the OLAP or Data Mining services: This service consists in the execution of one commercial extraction tool. This service has also not been analyzed in our experiments.
- Service 6. Statistics analysis presentation: The ArchCollect architecture collects, for each interaction, its total response time, its waiting time and its service time, and also the service and the waiting time of the monitored application. This way, the same performance analysis methodologies and capacity planning used in this section are used for analyzing the behavior of the monitored application and of the architecture itself. This information is presented online to the administrator of the site by the visualization component.

4.2. SERVICE LEVEL DEFINITION AND PERFORMANCE METRICS CHOICE

The ArchCollect architecture must be able to collect the information related to an interaction in a finite and shortest time interval. Every time this interval is exceeded, the information is rejected and a timeout is raised. The greater is this timeout value, the greater will be the number of simultaneous users, the greater it will be the number of active threads and the greater it will be the use of resources as memory and CPU. The smaller it is this timeout value, the greater it will be the number of timeouts and the smaller it will be the quantity of collected information.

[1] Think time: Time between the moment that the user receives a page and the moment that the user requests another page. This variable molds the length of time that it takes for a user to read or understand the information in a page.

[2] Response time: Time between the moment that the service is requested and the moment that this service is completed.

Therefore, there is a clear relation between the desired service quality and the computational performance (internal resources) necessary to reach this quality. This parameter can be optimized as the system workload increases or diminishes, but in our experiments it was used the value of 60 seconds. The performance metrics[21, 22] chosen to analyze the collected interactions service are: the total response time observed by the user and the system throughput[3]

4.3. PARAMETERS THAT INFLUENCE THE EXPERIMENTS

Many parameters influence the experiments, some of these parameters are characteristics of the system itself, such as the quantity of the servers RAM memory or the processor speed and the transmission speed of the connection between the clients and the server. Other parameters depend exclusively on the workload, such as the number of simultaneous users, the number of new users that arrive at the system per time unit and the think time. The experiments were made with the purpose of knowing only the impact of changes in the parameters: number of simultaneous users and number of new users per time interval. The values used for these parameters in the experiments were: number of simultaneous users using the system, which was equal to 50, 100, 125, 150, 175 and 200; and number of new users arriving at the system in a time interval of 10 minutes, which was equal to 150, 200 and 300. The think time was established as 10 seconds, an average time for an ASP page used in the experiment to be received by a user in the Internet.

All the experiments were made with a Microsoft Web Stress Tool, configured to simulate 56 Kbps connections between the clients and the web server.

4.4. CHARACTERIZATION OF THE WORKLOAD

The workload was characterized by a set of HTTP requests using the POST method on 10 ASP pages stored in the IIS server. Each one of the pages was associated to a different product and each request that was sent to the IIS server has informed the amount of this product that was bought. It was supposed a site were 20% of the users access at least, until the third page of the site, 30% until the seventh page of the site and 50% of the users access until the tenth page. All the pages had the size of 12885 bytes. The table 3 shows the requests that characterize the workload.

For each one of the proposed scenarios, experiments were made where the number of simultaneous users had varied and the rate of new users remained constant and experiments where the opposite had occurred, resulting in a total of 2x(6x1+1x3)=18 experiments. Each experiment lasted, on average, 10 minutes and was repeated 10 times, so that the results average could be collected. The total time used in the execution of the experiments was, at least, equal to 30 hours.

[3] Throughput - the *throughput* measures the number of services that the system is able to carry out per time unit, in other words, the number of requests collected per second.

#	**Method**	**Page**	**Content**
1	POST	\pag1.asp	28+10+1998+21+34+19+pagina1+button1+CD+2+20
2	POST	\pag2.asp	28+10+1998+21+34+19+pagina2+button1+Livro+2+30
3	POST	\pag3.asp	28+10+1998+21+34+19+pagina1+button1+Computador+2+2000
4	POST	\pag4.asp	28+10+1998+21+34+19+pagina1+button1+Arroz+2+5
5	POST	\pag5.asp	28+10+1998+21+34+19+pagina1+button1+Feijao+2+5
6	POST	\pag6.asp	28+10+1998+21+34+19+pagina6+button1+Macarrao+2+2
7	POST	\pag7.asp	28+10+1998+21+34+19+pagina7+button1+Oleo+2+1
8	POST	\pag8.asp	28+10+1998+21+34+19+pagina8+button1+CD+2+20+1
9	POST	\pag9.asp	28+10+1998+21+34+19+pagina9+button1+Livro+2+30+1
10	POST	\pag10.asp	28+10+1998+21+34+19+pagina10+button1+Computador+2+2000 +1

Table 3. Workload characterization

5. PRESENTATION AND ANALYSIS OF THE RESULTS

Figure 4 presents some samples of the response time observed by a web user, when the number of simultaneous users varies from 50 to 200 users and the rate of new users arriving at the system remains in 20 users/minute. The ArchCollect architecture had its best performance when the number of simultaneous users was equal to 50 users and the response time was equal to 3,9 seconds. Up to 100 users, the response time increases linearly as the number of users increases, but from this limit the response time seems to have an exponential growth as the number of users increases.

The IIS server response time remains relatively the same for all the experiments. This figure also shows that, in the second scenario, the response time observed by the user increases by an order of magnitude when compared to the response time observed in the first scenario. The ArchCollect architecture has caused the increase of 3.2 seconds on the average response time observed by the user, and at most 15.6. seconds

For the same number of simultaneous users, the response time does not show a significant growth when the rate of new users doubles – Figure 5.

Figure 6 shows how the service and waiting time of the monitored application and of the ArchCollect architecture behave when the number of simultaneous users increases and the rate of new users arriving at the system remains constant and equal to 20 users/minute. When the number of simultaneous users is over 150, it is observed that the waiting time of the ArchCollect architecture becomes bigger than its service time and starts to grow quicker, confirming the saturation state of the architecture.

Figure 7 shows that the throughput observed by the web user grows linearly as the number of simultaneous users increases up to a limit of 100 users. Over this value, the system comes to a saturation state where the ArchCollect throughput tends to a limit of 7 requisitions answered per second, approximately 604.800 interactions a day, and the IIS server throughput continues growing linearly. This discrepancy can be justified by the fact that the ".asp" pages that compose the workload do not have any ASP code.

6. CONCLUSIONS AND FURTHER WORKS

This article presented an architecture for collecting and analyzing users interactions. The low coupling to the application that is going to be monitored enables the ArchCollect to deal with many web applications at the same time.

Components with specific functions allowed the development of a tool that keeps sufficient information to answer question about sales, business results and performance results. We implemented and tested the inicial ideas of this architecture.

The work can be extended in many directions. First, the semantic log file should be eliminated. The ***collecting component*** should send the collected information directely to the ***transformer component***, improving then the total scalability of the ArchCollect. Second, some experiments, using different numbers of ArchCollect servers, must be made to observe the architecture scalability. Third, duplication components should not make disk access, because they should, as the ***collecting component,*** send it´s collected time directly to the ***transformer component.*** In this new model, the transformer component will be the only one to instance loader component objects.

Mentioning data extraction, the data mining and OLAP services must be added to the statistics visualization, ending the possibility of data extraction from a unique relational model.

The personalization components are just one path for the understanding of the collected interactions. Web recommendation algorithms or communities creation algorithms will be able to bring a huge contribution for the understanding of the interactions, and then providing a more sophisticated user behavior profile. When establishing bigger sets called communities, we improve crucial questions such as a better performance in the obtainment of the profiles.

Nowadays, only the waiting time and the service time for each interaction on the ArchCollect servers and on the web application servers are analyzed. These times show how much it it costs to carry out a certain service or process to the user. A better architecture internal performance analysis allows knowing which component, specifically, behaviors as the bottleneck of the whole architecture.

The ArchCollect implementation may be modified to support a bigger number of simultaneous users, to decrease its response time and to increase its throughput so that these values may be compared to the values presented by the IIS web server. Two alternatives to obtain this speedup are to implement more efficient starategies in our component algorithms or attempt to distribute these components among many hosts connected by a communication network.

Another point that can be studied is the adequacy of the current ArchCollect architecture to the interactive TV models on the web. Information about the target public, their interactions and these interactions relation to a service or product, has to be established. We believe this adequacy is not difficult to obtain, because if we think about the interactive TV as a purchase or sales channel, it will be possible to associate them to the commerce applications already existent on the web.

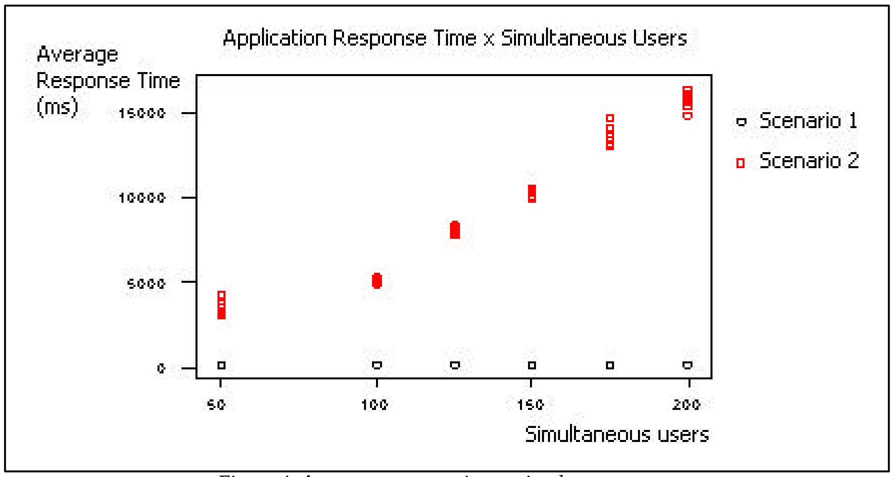


Figure 4. Average response time x simultaneous users.

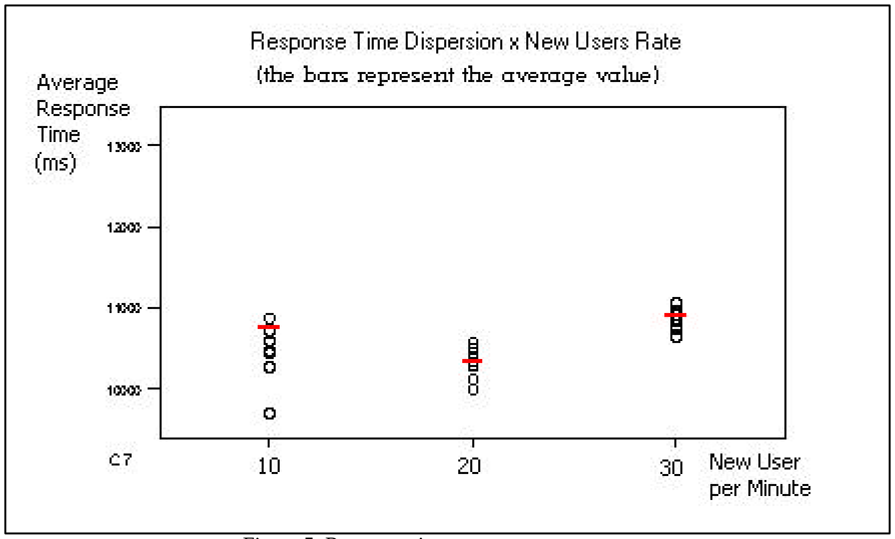


Figure 5. Response time versus new users rate.

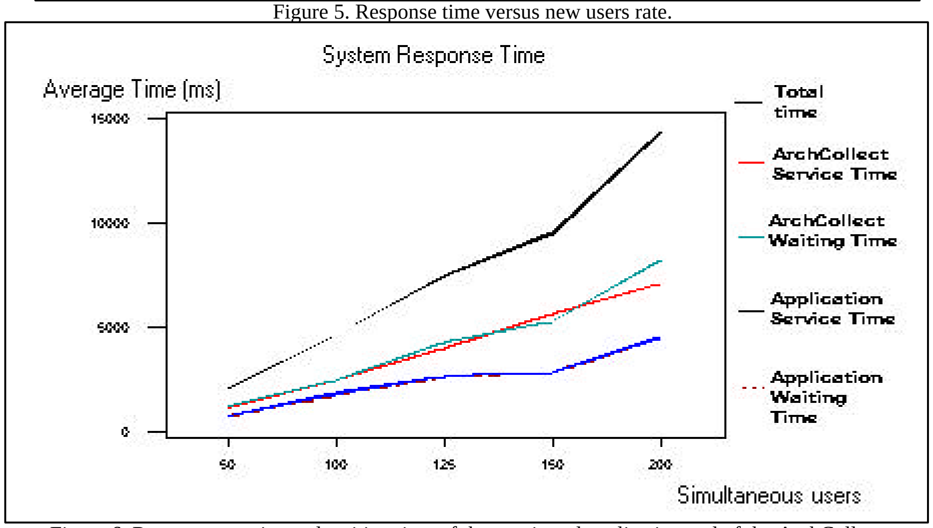


Figure 6. Response, service and waiting time of the monitored application and of the ArchCollect

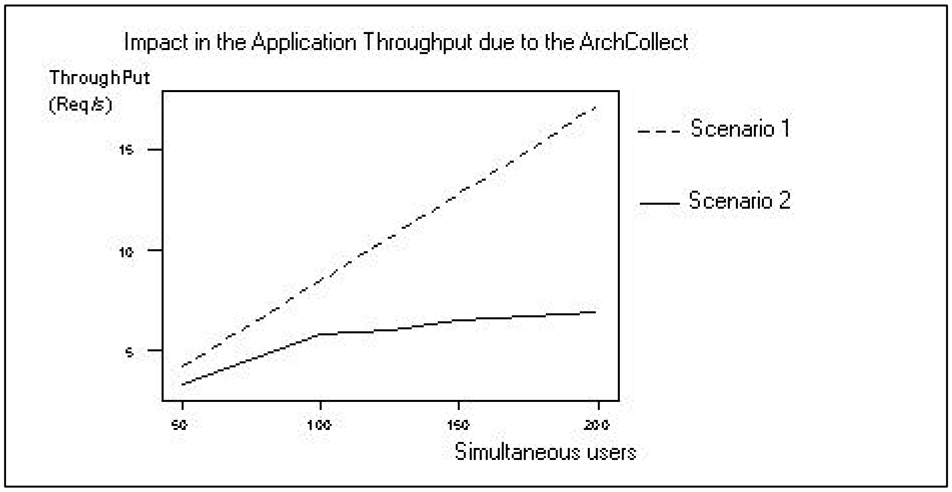


Figure 7. Impact of the ArchCollect on the rate of answered requisitions observed by the server.

7. References

[1] M.S.Chen, J.S.Park, P.S.Yu, " Data Mining for Transversal Patterns in a Web Environment", Proc. of 16th International Conference on Distributed Computing Systems, 1996.

[2] S.Gomory, R.Hoch, J.Lee, M.Poldlaseck, E.Schonberg, "Ecommerce Intelligence : Measuring, Analyzing, and Reporting on Merchandising Effectiveness of Online Stores", IBM Watson Research Center, 1999.

[3] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, Pang-Ning Tan, "Web usage minig : Discovery and applications of usage patterns from web data", SIGKDD, January, 2000.

[4] Myra Spiliopoulou and Lukas C Faukstich. WUM: A web utlilization miner, EDBT Workshop WebDB98, Valencia, Spain, 1998.

[5] Kun-Lung Wu, Philip S Yu, and Allen Ballman. SpeedTracer: A web usage mining and analysis tool, IBM Systems Journal, 37(1), 1998.

[6] Cyros Shahabi, Amir M Zarkesh, Jafar Adibi, and Vishal Shah. Knowledge discovery from users web-page navigation, Workshop on Research Issues in Data Engeneering, Birmingham, England, 1997.

[7] O.R.Zaiane, M.Xin, J.Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs, Proc. of Advances in Digital Libraries Conference, 1998.

[8] Andromedia Inc´s Aria, http://www.andromedia.com

[9] DoubleClick Inc, http://www.doubleclick.com

[10] Engage Technologies Inc´s, http://engagetechnologies.com

[11] IBM Corp´s SurfAid, http://surfaid.dfw.ibm.com

[12] Marketwave Corp´s Hit List, http://www.marketwave.com

[13] Media Metrix, http://www.mediametrix.com

[14] net.Genesis´net.Analysis, http://www.netgenesis.com

[15] NetRating Inc., http://www.netratings.com

[16] Straight UP!, http://www.straightup.com

[17] Kun-Lung Wu, Philip S Yu, and Allen Ballman. SpeedTracer: A web usage mining and analysis tool, IBM Systems Journal, 37(1), 1998.

[18] Gamma, E. et.al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 1995.

[19] Sakhr Youness, "Professional Data Warehousing with SQL Server7.0 and OLAP Services", 2000.

[20] Cyrus Shahabi, Farnoush Banaei-Kashani, Yi-Shin Chen, and Dennis McLeod, "Yoda: An Accurate and Sacalable Web-based Recommendation System", In the preceedings of the Sixth Internacional Conference on Cooperative Information Systems, Trento, Italy, September 2001.

[21] Menascé, Daniel A. & Almeida, Virgilio A.F., "Capacity Planning for WEB Performance - Metrics, Models & Methods", Prentice Hall, PTR, 1998.

[22] Jain, R.. "The Art of Computer Systems Performance Analysis - Thechniques for Experimental Design, Measurement, Simulation, and Modeling", John Wiley & Sons,Inc., 1991