Relatório Técnico

Núcleo de Computação Eletrônica

A Responsive Client Architecture with Local Object Behavior Deployment

A. P. V. Pais
B. Brasil
C. E. T. Oliveira
G. Tavares

NCE -12/02

Universidade Federal do Rio de Janeiro

A Responsive Client Architecture with Local Object Behavior Deployment

Ana Paula V. Pais[1], Bárbara Brasil[1], Carlo E. T. Oliveira[1], Gilson Tavares[1]

[1] Universidade Federal do Rio de Janeiro, NCE-CCMN Bloco C, PO. Box 2324, Rio de Janeiro Brazil
{anapais, barbara, carlo, gilson}@nce.ufrj.br
http://tedmos.nce.ufrj.br/labase/index.html

Abstract. Information Systems with a large user base require thin client technology due to deployment issues. Configuration and version management, added to maintenance cost are major players on this decision. However, heavy traffic loads and network latency impose severe penalties on users. A candidate solution to this problem must optimize users ergonomics whilst keeping maintenance costs low. In a straightforward approach, client architectures should provide web deployable local processing facilities. This paper describes an architecture based on the MVC paradigm capable of porting portions of the system to client executable scripts. Required entities are encapsulated in XML carriers and exchanged on demand between client and server side systems. Numerous user interactions are executed on the client side, relieving the network and improving responsivity. This solution not only enhances ergonomics but is also highly scalable, delegating tasks to the greater number of client machines, whereas focusing server activity on more relevant operations.

1 Introduction

Web information systems can be hard on users that expect responsivity. Solving this problem should not impact on development or maintenance costs. A responsive client system must be capable of handling local legs of use case logic without interfering with architectural robustness of entire application. Most of the time, users are dealing with data that can be resolved locally or involve read only access to entity lists. Discretionary additions of client side scripting may solve the user problem at cost of system integrity flaws, interweaving of GUI and business logic, high increase of maintenance complexity. An engineering approach should commit to architectural soundness, preserving at client level the quality standards employed at the server side. A client scaled reproduction of the server design can live up to the paradigm advantages, running the local logic in conformance with business constraints and keeping layout design apart from control code. Keeping the MVC paradigm [1] the client architecture can operate safely on objects, transporting them back and forth

In web systems, the user interface resides in the client layer, while model and control modules reside in the server. This means that any user action is submitted to the server. The client layer does not execute any type of validation.
The access control is fundamental for the majority of currently developed systems. Users of a system possess different privileges. As an example, all users can know which are the available products of a store, but not all of them can modify the information of a product. This means that some actions in the use case are barred to some users. The stunt architecture currently does not contemplate this aspect.

2.2 Reverse Stunt Architecture (RSA)

The high benefit of the stunt architecture derives from tracking user specified use cases right down to its implementation. However, in the case of legacy systems, use case logic is tangled all over the code, requiring refined computational intelligence to extract it. A simpler approach to this case is to rely on the data model underlying the legacy system. In most cases, the data model can be extracted from database metadata information. This relevant information is not taken into account by the stunt architecture, since the data model is generated from use cases. The Reverse Stunt Architecture is a modification on SA to make use of existing data models.
In most of the times, the only existing documentation in legacy systems is the database design. However, the data model is a highly representative part of the system and cannot be dismissed. In fact, it is a promising starting point for the migration of system.
The Reverse Stunt Architecture was designed to tackle specifically this problem. It uses the relational schema to automatically generate various modules of the new system on the top of a robust and reliable architecture. The generated classes map entities extracted from the database tables. The RSA also generates a set of support classes, building a minimal functionality to the system. These classes prepare the system to carry through basic operations as modify, insert or exclude an entity. RSA integrate in its architecture mechanisms to improve system management and configuration, commonly found in many corporations. On the other hand, the business rules are engaged. In this aspect, the architecture stunt is well fuller. Table 1 presents a comparison of SA and RSA.

Table 1 - Comparison of SA and RSA

Feature	RSA	SA
MVC	Poor	Complete
Traceability	ER	UML
Customization	Easy	Hard
Maintenance	Hard	Moderate

The RSA embeds many other aspects for business applications. It supports access control based on user profiles; users profiles scope management; calendar based access control and declarative specification of entity visualization. As this

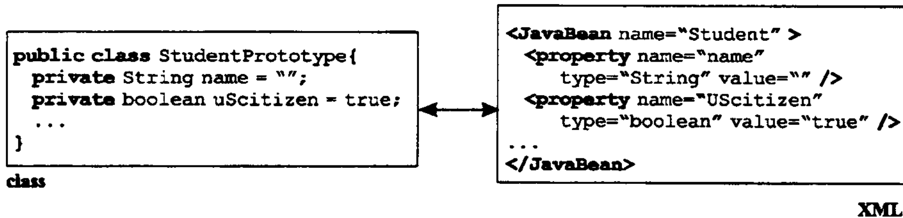

Figure 1 – prototype-carrier translation

In some cases there is a need for prototypes that do no correspond to any entity class of the domain model. This requirement appears in some use cases in which intermediate entities or pseudo-entities represent a transient combination of attributes. The use case controller is assigned to assemble these prototypes. The functionalities to include, to bring up to date and to exclude are not part of this variation. Even if the prototype is created in this variant form, it still allows the information contained within to be passed to other parts of the system through XML. This mechanism offers advantages as decoupling between the system layers and adaptation of the model to the necessities of the use case. Operations can be carried through without compromising the integrity of the entities or performance of the system. The representation in XML guarantees an efficient carriage of information.

2.4 Visualization Widgets

The entities cross the system borders in the form of prototypes. When arriving at the client layer, the entity assumes a form with which the user can interact. Widgets constitute the bridge between the user and the entity.
The state of an entity is represented by the value of its attributes. These values can be classified in some categories, which can be associated with one or more widgets. The more common widgets are textboxes, checkboxes and combo-boxes.

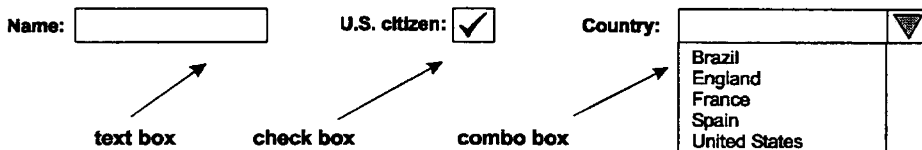

Figure 2 - common widgets

A textbox can be used to represent attributes that store a text, character or a number. A checkbox represents boolean values, as true/false or 1/0. A combo box shows a set of options, where a single choice can be made. This can be useful for attributes whose value must be chosen from a fixed set of values, as the day of the week.
The form that the attribute will take in the user interface is stored in the database. Each attribute is associated with a set of features:

- Visual – identification of the widget that represents the entity attribute in the edition screen;

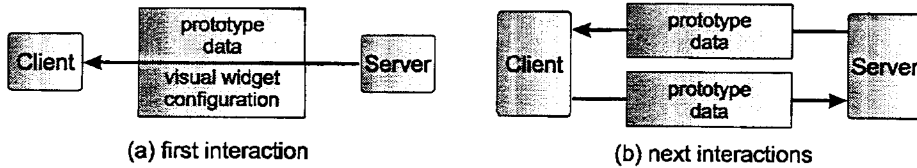

Figure 4 - XML carrier operation

When the use case initiates, the client receives the XML carrier containing both view and model elements. The XML carrier is translated into objects responding to local user interaction in the client layer. The objects that implement prototypes are initiated with information proceeding from the server. These objects are locally modified due to user interaction with visual widgets. Widgets wrappers implement the widgets listed in the XML carrier. These wrappers are the bridge between the visual interface and prototypes. Widgets wrappers are associated to the attributes of a prototype. When the user modifies the value shown for one widget, the wrapper is notified and modifies the corresponding prototype. On the other hand, when the prototype suffers some modification, all wrapper listeners are notified, updating the visual interface.

When user action implies in data transfer to the server, the prototype is translated into XML. Only the model part of the carrier is created from the current state of the client application and submitted to the server. The server receives the XML carrier, and executes the required operations. Client/Server interchange carries on by passing the XML back and forth, synchronizing model and view representations.

3 The Enhanced Client Architecture

The stunt architecture is tailored for system construction originating from the elaboration of UML modeling diagrams. To simplify automatic code generation, SA is restrictive about user interface customization and does not support reverse engineering of metadata.

The RSA was developed to supply some deficiencies of the SA. The current development of the RSA centers on the construction of web systems. Moreover, one of the objectives of the RSA is the construction of a client layer capable of local use case operation.

Although SA requires an abstract GUI representation, RSA can live up with a hardcode implementation. The client layer of the RSA encloses HTML pages and scripts responsible for local operations. The HTML Page possesses a standard format that can be constructed from the information contained in the database. The developer can also create new forms to suit use case requirements, observing the established standards.

Moreover, a single HTML page can contain all the required code covering the full set of screens needed in a use case. That is made through a set of layers, where a single layer is visible at a time. Each layer consists of a block of HTML code that becomes visible when renderized in a specific place of the screen. The client executes a set of scripts written in Javascript [11], capable of managing layers as well as modifying the content of fields in the corresponding HTML forms.
View is committed to a coherent standardized layout. Overlaid panes are built in advance to achieve fast response against user interaction. The use case is represented in a HTML page using a standardized format, as shown in Figure 6. Three containers form the HTML page: a menu bar, one tabbed notebook and a button bar.

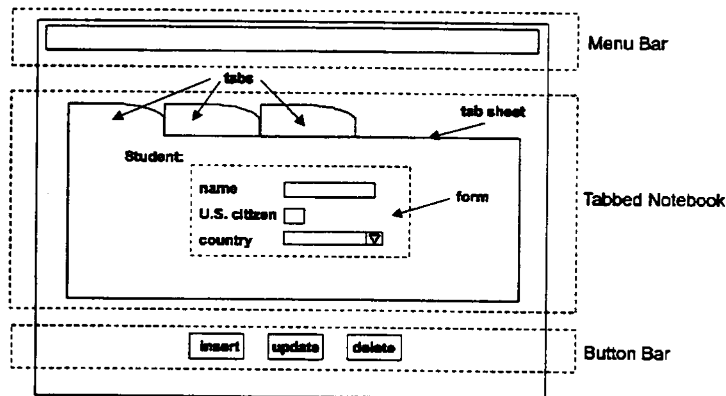

Figure 6 – standardized layout

Menu Bar. Is the entry point for the execution of the main system use cases. It corresponds to the set of accessible options to an user.

Tabbed Notebook. Contains the set of screens (tab sheets) for a use case. A single tab sheet is visible at a time. A tab sheet becomes visible when its corresponding tab is selected by the user. The tab sheet content consists of a HTML form, that is specified in one DHTML [10] layer. A tabbed notebook corresponds to a Javascript component that manages which layer must be visible. It also answers to the events provoked by the user interaction with the visible tabs.

Button Bar. The set of actions executable by the user is represented as buttons in the button bar. These actions are capable of modifying the use case current state.

Widget Mapping. The form shown in a tab sheet contains a set of input fields, with which the user can interact. These fields constitute the graphical representation of attributes pertaining to one or more prototypes. The form allows the user to visualize and modify the content of prototypes. The prototypes are transmitted to the server,

sheet is always associated to at least one state. The selected tab sheet represents the current state of the controller. Tabs presented in the screen represent events originating state transitions. Not all the states will be visible in the screen. In the state configuration it is necessary to specify the acknowledged actions with the corresponding target states. Actions can take the form of button in the button bar or be associated to events on any other widget. Links and figures are examples of components that do not appear in the button bar, but represent an action for the controller. The controller change state or execute an action without accessing the server. Local operations reduce communication overhead between server and client and HTML renderization traffic. Controller construction defines the initial state and the collection of valid states and actions. However, the client controller configuration can change in the course of the use case. When this occurs, the server updates the client controller sending a new object wrapped in a XML carrier.

3.4 Model

Once loaded from server, entities reside on a local repository, responding to local operations.
Entities reach the client wrapped in a XML carrier. They are translated back into a set of objects, which act as models to wrapper widgets, and constitute the prototypes in the client side layer. The user interaction on visual widgets reflects in the prototype properties. The prototype is translated into XML when an user action needs to be executed on server side.

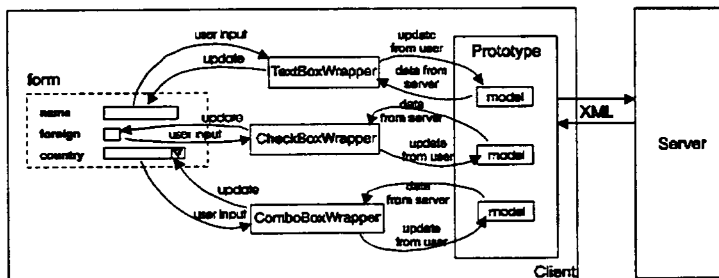

Figure 9 – widget-attribute binding

Prototypes, in the client context, are formed by a set of objects. There are three types of objects: JavaBean, Composite and Element. The JavaBean object corresponds to an entity and aggregates a set of Elements. The Element object corresponds to an attribute of the entity. Elements, generally, are used as models for textboxes and checkboxes. A Composite is a collection of JavaBeans. It represents aggregations between entities. Thus, a JavaBean is also composed of Composites.

The second stage consists of developing a mechanism capable to analyze the content of the original structure and to create corresponding objects. This mechanism is implemented by the patterns Visitor and Builder. The Builder is programmed to create objects in the destination structure from the information collected by the Visitor. The Visitor scans the intermediate structure analyzing the content of each element and invoking the Builder accordingly.
Within the active client context, the projector attributions are:

- Construct the prototype from the information contained in the XML carrier originated from the server (Figure 10.a).
- Construct the corresponding XML prototype carrier that it will be submitted the server (Figure 10.b).
- Construct wrappers widgets from the information contained in the XML carrier (Figure 10.c).

4.2 Visitor

The visitor pattern role in the projector operation is to renderize the information into the required representation. The visitor pattern is used to carry through operations in the structures of data used in the client module. The main function of the visitor is to interchange the representation of the information between the forms of prototype and XML.
The codified information arrives at the client encoded in XML. The receiving projector scans the prototype using a visitor. This visitor gets the corresponding information to each element of the model from the XML input. This operation results in the update of the local model. The transmitting projector applies the visitor in the model to collect the local information. This visitor collaborates with a builder to construct the XML output. The resultant information is sent to update the server.
The visitor pattern normally scans all the branches and leaves of a nested structure. In the used client projectors, the visitor can also be programmed to ignore some nodes of the structure, allowing that only selected parts of the date structure to be exchanged or synchronized with the server.

4.3 Operation Scenarios

The use of MVC architecture in the client layer allows the local processing of simple and complex use cases. The renderization of a use case involves the generation of a XML carrier and a client side controller. While the carrier is just the encoding of view and model data, the controller logic must be entirely developed in Javascript. The development of a fully fleshed information system requires a large amount of dedicated work. However, automatization of use case development could accelerate this process. Factoring similar functionalities on usual scenarios leads to a set of

some peculiarities of explorer have been used to speed up the development, these will have to be neglected by forms that are more adaptable to other browsers.
The main objective of this implementation is to reuse the data proceeding from server requests, managing its access locally in the client layer. These data remain in the client, preventing the submission of successive identical requests to the server. Previously loaded screens with requested data that are common to several use case extensions are kept for posterior use. This technique reduces the number of client/server interactions.

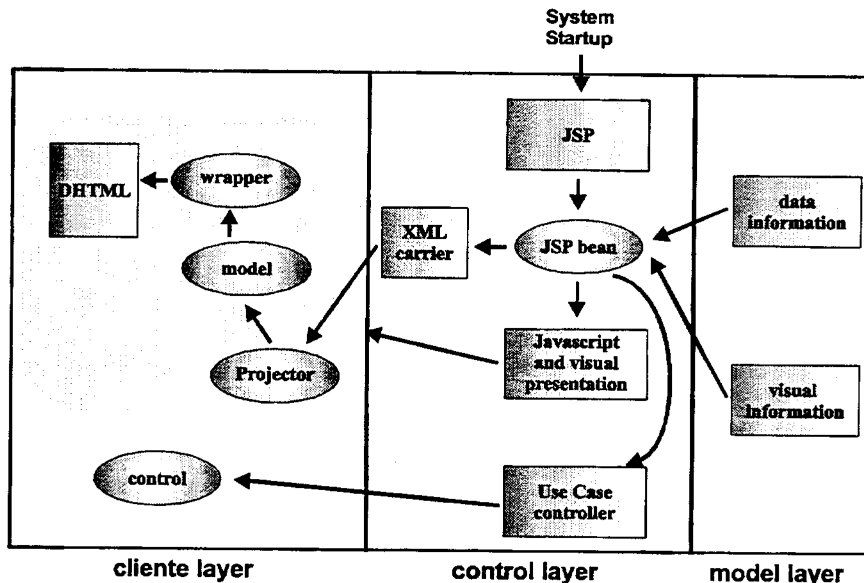

Figure 11 – client initialization flow

6 Assessments

The RSA is being used in a university academic management system called SIGA. This system involves all academy features with personalized user interaction. Teachers, students and academic managers work together in the same environment with different access privileges configured by declarative clauses.
The SIGA system is appropriate to evidence the features proposed by RSA since its first implementation was developed in Cobol language, hosted on a mainframe. In a first attempt of system migration, the development schedule was overdue. The critical points on this process pointed out the requirements to conceive the RSA. At the current version, RSA is ready to present better results for this process.
The addition of specific functionalities beyond the basic RSA controller is still a hard task. Due to generation by the diagram models the SA can build complete use case

5. Gamma, E., Helm, R.; Johnson, R., Vlissides, J., Design Patterns – Elements of Reusable Object-Orient Software. Addison-Wesley, 1998.
6. Monson-Haefel, R., Enterprise JavaBeans. O'Reilly & Associates, 3rd edition
7. Harold, E. R., XML Bible. IDG Books Worldwide, Inc.
8. Hall, M., Core Servlets and JavaServer Pages. Sun Microsystems.
9. Shannon, B., Hapner M., Matena V., Davidson J., Pelegri-Llopart E., Cable L., Java 2 Platform, Enterprise Edition: Platform and Component Specifications. Sun Microsystems.
10. DHTML References. Available at: http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/dhtmlrefs.asp. Access on march/2002.
11. Wagner, R., et al., JavaScript Second Edition. Sams Net.