Applied Computer Science, vol. 16, no. 4, pp. 48–55
doi:10.23743/acs-2020-28

Submitted: 2020-10-17
Revised: 2020-12-02
Accepted: 2020-12-17

architectural paradigm, pattern, module, cloud technology

Denis RATOV [0000-0003-4326-3030]*

ARCHITECTURAL PARADIGM OF THE INTERACTIVE INTERFACE MODULE IN THE CLOUD TECHNOLOGY MODEL

Abstract

The article discusses an architectural template for building a module for organizing the work of a multiuser windowed information web-system. To solve this problem, JavaScript objects have been created: a window manager object and a window interactive interface class, which allow a web application to function when organizing cloud technologies. The software implementation is considered and the results of the practical use of the developed module are presented.

1. INTRODUCTION

Today, when developing information systems, cloud technologies are often used for remote computing and data processing (Medvedev, 2013). Cloud computing refers to the provision of computer resources and capacities to the user in the form of Internet services. Cloud computing is a distributed data processing process in which computer resources and network capacity are provided to the user as an Internet service (Papadopoulos & Katsaros, 2011). Cloud technology inherently implements the processes of creating cloud applications and organizes work with them, without the introduction of additional software. Typically, for such applications, functionality is created in a web browser environment. Such a software product is a client-server application with a Web interface that provides the user with the ability to access data from any

* Volodymyr Dahl East Ukrainian University, Faculty of Information Technology and Electronics, Department of Programming and Mathematics, Tsentralnyi Ave, 59A Severodonetsk, Luhansk Oblast, Ukraine, 93400, denis831102@gmail.com

active point, provided that they are connected to the Internet. Cloud data processing or computing is not provided on the clients' personal computers, but on powerful server computers. For effective interaction of the client with remote data without completely reloading the current page, consider the user interface template, which is put into the structure of a module that implements controls, input, sending and receiving data in the form of windowed web forms with their inherent functionality in the browser context.

By a web form we mean an independent fragment of the user interface with its own logic of behavior, for the display of which the template objects of the module being developed are used. One of the purposes of such a module is to reuse it. This allows you to define the functionality of objects once and use them in different contexts and information systems.

When developing the module solves the following problem: you need a reliable encapsulated namespace in which you can define the data and functionality of objects. This makes it possible to make some of this data available and to limit the functionality of others.

Today there are web technologies, and libraries and frameworks developed on their basis for creating web applications and user interfaces designed for information systems to work in browsers. The processes of standardization of HTML (HTML 4.01 Specification, n.d.), CSS (Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, n.d.) and JavaScript (ECMAScript Language Specification – ECMA-262 Edition 5.1, n.d.) languages allowed to achieve not only a high degree of cross-platform user interfaces, but also a fairly good degree of cross-browser compatibility, so the use of appropriate standards when building Web applications has become the dominant approach.

When developing modular information systems, standards alone are not enough: design patterns, libraries of standard controls, support for presentation logic (for example, Presenter in the MVP model (MVP architecture, n.d.)) and much more are needed. The corresponding tools are still in their infancy. Examples of free products are (MediaWiki, n.d.), Drupal (Drupal – Open Source CMS, n.d.), WordPress. The inevitable payback for such systems is the binding to server technologies, which limits their application in situations where the server environment is fixed for the developer. When considering client libraries that do not depend on server technologies, their specialization is visible: on manipulating the DOM model (jQuery (jQuery, n.d.), Zepto.js (Zepto.js: the aerogel-weight jQuery-compatible JavaScript library, n.d.)), styling pages and controls (Bootstrap, n.d.; jQuery UI, n.d.; w2ui (w2ui: Home, n.d.)), building application frameworks (AngularJS (AngularJS – Superheroic JavaScript MVW Framework, n.d.), Backbone.js (Backbone.js, n.d.), Knockout (Knockout: Home, n.d.)).

Despite the rich set among the existing tools, there are tasks that are relevant in the development of information systems: the presence of a dispatcher of interface models according to a given specification, data integrity control with the possibility of multi-user access.

2. MODULE ARCHITECTURE

The module being developed consists of two relatively independent, interacting with each other components, which are implemented as JavaScript objects: a window manager object and a window interactive interface class. Let's use JavaScript's mechanism for accessing objects using the new operator, which is used to create objects using the function of our own constructor, thereby creating an analogue of the class. Such a constructor stores an instance of the object in the closure. This prevents changes to the object outside of the constructor function. This uses an object creation template called a "module" and an isolated namespace template (Stoyan Stefanov, 2011). In order to change not only the appearance of the displayed form components, but also their behavior without modifying the main objects of the module, a design principle called the template method was applied during development (Gamma, Helm, Johnson & Vlissides, 2001). The library code contains a method for setting callback functions where the developer needs to supplement the standard data and event handling with his own actions.

The mechanism of the user interactive interface, in addition to the functions of working with interface components, must meet the following requirements:

1. The web form object must have a method for sending an ajax request to the server and be able to process server responses.
2. Web form object can be either a simple set of fields or contain subsections, tabs or tables.
3. The appearance of web forms must be customizable.
4. There can be several web forms on the page that can interact with each other.

```
var ListWin = function (){
    let Forms = [],      // array of forms
        numNewForm = 0, // another unique form identifier
        curObj = 0,      // number of the current element (form)
        zIndex = 98;     // initial level of forms
    this.getDocumentHeight = function (){ }// height of the whole page including the invisible part
    this.getDocumentWidth = function(){ }; // the width of the entire page including the invisible part
    this.createNewForm = function(arrProp){ }; // create and load a new form
    this.getForm = function (num){ };  //link to the form by its number
    this.setZIndex = function (curNum){ };  //move to the top level of the form
    this.delForm = function (){ }; //delete the current form
    this.setTimeoutWaitScreen = function (timeOut){ }; //close the current form
    this.loadWait = function (modeL, modeS){ }; // preloid of the operation
    this.emptyForm = function (cap, len){ }; // check for the existence of a form with a given title
    this.mousedownDAD = function(object, funcMouseUp, hideClone) { } // function drag and drop
};
```

Fig. 1. Dispatcher object architecture

The developed window manager (ListWin) is a JavaScript object that:

1. Stores a collection of generated web forms with their components.
2. Provides a high-level API for manipulating web forms and data from client controls.
3. Provides interaction of the web form with the DOM model of the web document.
4. Performs preliminary visualization of running processes on forms.
5. Provides a drag and drop mechanism for controls.

The ListWin dispatcher implementation consists of its own constructor with privat fields and public methods. Figure 1 shows the object architecture of the dispatcher.

The JavaScript object of the windowed interactive interface is responsible for the operation of the application, handling the events of the form component and includes:

1. Methods for rendering and manipulating the web form: init(), changeCaption(), changeVisible(), setWidth(), WaitLoad(), addObj().
2. Web form event trigger constructor: initializationEvent().
3. Constructor of event handlers for web form controls: addEvent().

Let's consider the implementation of the window interface object (Fig. 2). The functions of an object are implemented as its public methods.

```
var winObject = function () {
    let id = 0,    // form id
        idContainer;  // form container id
    this.collObj = {};
    this.init = function (p_num, p_caption, p_visible, title, w, h, l, t, zIn){ };//initialize
    this.WaitLoad = function(flag){ }
    this.changeCaption = function (p_name){ };  // change the title of the form
    this.initializationEvent = function(){}; // initialization of form movement events
    this.changeVisible = function (p_visible){}; //change the visibility of the form
    this.setWidth = function(w){}; //change the width of the form
    this.setHeight = function(h){}; //change the height of the form
    this.addObj = function(name_obj, arrProp){}; //add a new object with events to the form
    this.addEvent = function(mask, event, strFunc){}; //assigning events to form objects by mask
    this.wXHR = new XHRClass();
};
```

Fig. 2. Implementing the window interface object

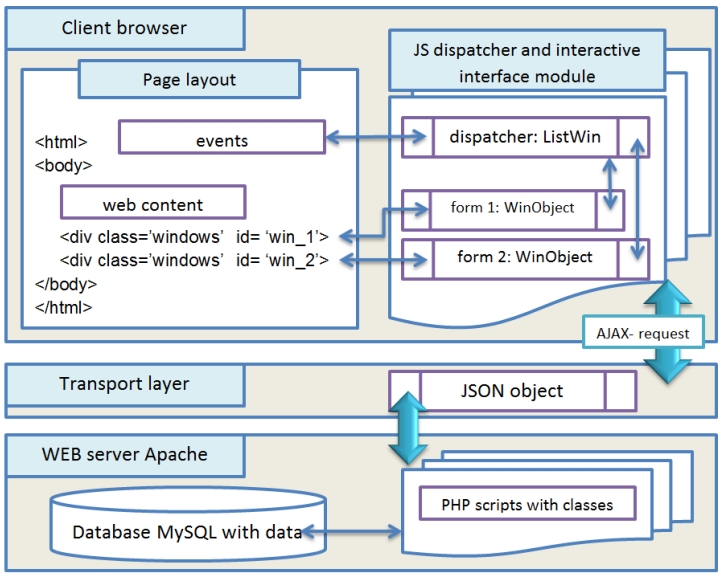

Fig. 3. Module components interaction scheme

Using an object XHRClass the transport layer of interaction between the client and the server is implemented, namely, loading data for web forms from the server, saving data to the server, asynchronous AJAX requests to methods of php objects on the server. All data transfer takes place in the background without reloading the page (Crane & Pascarello, 2006). The JSON format is used to transfer structured data between client and server.

The interaction scheme of the module components is shown in Fig. 3.

When constructing a separate application module based on the described objects of the dispatcher and the interactive window interface, the basic principle of creating objects can be a structural-hierarchical relationship. This approach allows you to design individual interactive interfaces in the form of application modules.

At the level of interaction of web-forms with each other, structural approaches are no longer effective enough, since only a small number of forms are in fixed master-subordinate relationships. Therefore, at this level, a transition was made to the network model of the organization (Fig. 3): forms win_1, win_2 are independent acting objects that react to the events of their components using callback functions assigned during construction by the ListWin dispatcher.

3. MODULE IMPLEMENTATION RESULTS

Consider this model and an architectural template using the example of implementing the module for creating certificates Modul_Sertifikat (Fig. 4). Using the loadFormBaza() method, the object provides manipulations with the DOM model of the web document. This uses the createNewForm() constructor of the ListWin dispatcher and its addObj() method to add controls with event handlers.

```
var Modul_Sertifikat = {name : 'Modul_Sertifikat', timer : undefined, XHR : new XHRClass(),
    formBaza : null, arOrderBaza : {order : 'FIO', dir : 'ASC'},
    arSymbolDir : {'DESC' : '&#9650;', 'ASC' : '&#9660;'};
Modul_Sertifikat.loadFormBaza = function(){
let optOglad = this.optionOfArray(this.arOglad);
    this.formBaza = ListWin.createNewForm({caption : 'Certificate of preventive examination',
                                           height: 235, width : 880, visible : 1, WaitScreen : 0});
    if (this.formBaza) { with (this.formBaza){
        addObj('div', {data : 'NAME: ', pos : [439, 38], style:'font-size:14pt; color:#132F76;'});
        addObj('input', {id : 'INAME', type:'text', pos : [480, 35], wh : [288, 22], class : 'defaultInp',
                         style : 'font-size:10pt; border:2px solid #92C8ED;',
                         events :{oninput : this.name+'.inputData(this, true);' } });
        addObj('table', {id : 'kolontitul_tabl', data : kolontitul_tabl,  class : 'tab_class', wh : [835, 30],
                         style : 'margin-left:-2px; margin-top:88px; border: solid 2px #759cdd; '} );
        addObj('div', {id : 'containerTabl', wh : [835, 460], pos : [13, 165],
                       style : 'overflow: hidden auto; border: solid 2px #759cdd; visibility: hidden;'} );
        addObj('table', {id : 'tabl', parent : 'containerTabl', data : '<tbody></tbody>', class : 'tab_class',
                         style : 'position: relative; width:100%;' } );
        addObj('img', {id : 'WaitLoad', src: "img/loader.gif", wh : [70, 70], style : 'left:45%; top:25%;'});
        addObj('button', {data : '<span style="font-weight:bold; font-size: 12pt;">&#9672; Add new client</span>',
                         style : 'right:190px; height:50px; width:145px;', id : 'buttonIn',
                         events : {onmouseup : this.name+'.cartaPacient();'} });
        addObj('button', {data : 'Close', class : 'buttonform',  style : 'right:20px; font-size: 12pt;',
                         events : {onmouseup : 'onClick_CloseForm()'} });
        }
        Modul_Sertifikat.refreshBaza();
    }
};
```

Fig. 4. Certificate creation module

After the form is generated, the refreshBaza() method is called (Fig. 5). In it, an ajax request is sent to the server to the getSertifikat.php script, to the listFIO method of the class, which prepares the necessary data and organizes logic for the client side. The prepared information in the JSON object is delivered to the client browser, where it is converted to control parameters using an anonymous callback function. Form elements are accessed through the collObj collection of form objects.

```
Modul_Sertifikat.refreshBaza = function(){
let kodMed = this.formBaza.collObj['selectMed'].value;
    this.formBaza.WaitLoad(true);
    this.formBaza.wXHR.query({param:'listFIO', idustanova : kodMed,
                        FIO : formBaza.collObj['FIO'].value,
                        order : this.arOrderBaza.order, dir : this.arOrderBaza.dir}, 'php/getSertifikat.php');
    this.formBaza.wXHR.request.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200){
            Modul_Sertifikat.formBaza.setHeight(700);
            Modul_Sertifikat.formBaza.WaitLoad(false);
            Modul_Sertifikat.formBaza.collObj['containerTabl'].style.visibility = 'visible';
            answer = this.responseText.split('|');
            Modul_Sertifikat.formBaza.collObj['tabl'].querySelector('tbody').innerHTML = answer[0];
            System.changeWidthTabl(formBaza.collObj['containerTabl'], formBaza.collObj['tabl'], 850, 831);
        }
    }
};
```

Fig. 5. Form data revision method

In Fig. 6 shows the result of generating a certificate base form with filled data. The JSON object received from the server was converted into form parameters: into a two-level table with a client base and a list of certificates belonging to individuals; to the assigned callback functions for the context menu events for selecting clients and working with certificates; to general information about the number of certificates for a given filter. When the client card editing item is selected, a new form with AJAX loading of client data from the server is generated.

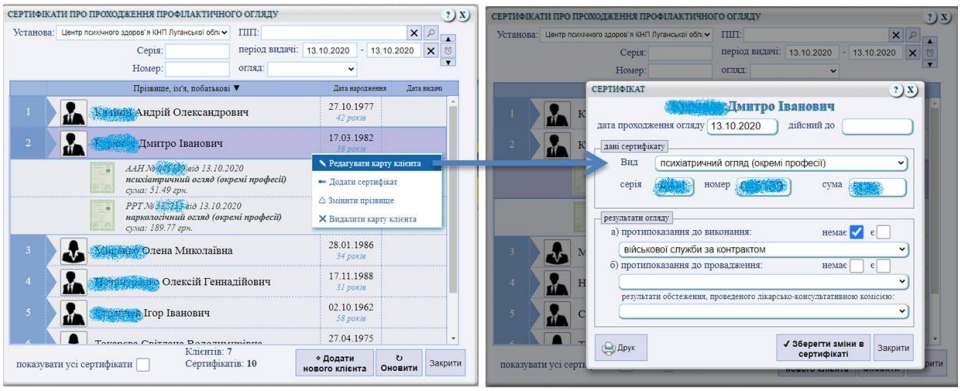

Fig. 6. Result of generating a certificate database form with filled data

To organize cloud computing and build an information system on the server side, the following software was used:

1. Server operating system – FreeBSD version 11.2.
2. Apache web server.
3. Hypertext preprocessor PHP version 7.4.
4. Database management system – MySQL server version 5.7.31.

To work on the client side of the software modules of an information system developed on the basis of JavaScript and Standard ECMA-262, any operating system is required that supports a www navigator with a JavaScript interpreter.

The practical implementation of the module model with the implementation of the dispatcher and the interactive window interface is represented by the installation in the form of the medical information system MedSystem being developed, in which the application modules are implemented in accordance with the considered interaction mechanism (Fig. 7).

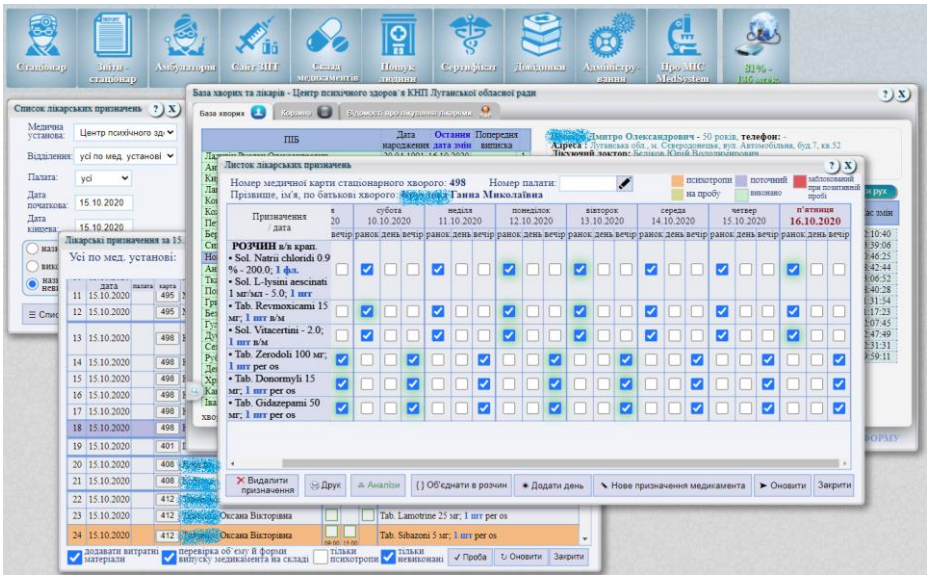

Fig. 7. Medical information system MedSystem

4. CONCLUSIONS

The proposed module architecture template for the implementation of the dispatcher and the functionality of the user window interface made it possible to create the modules of the MedSystem medical information system. The modular approach had a positive effect on the responsiveness of the user interface and the ability to scale the functionality of the system itself when implementing cloud technologies.

The results of the use showed that the mechanism of modular creation of a dispatcher and a windowed interactive interface of web-forms not only organically fits into existing technologies for building web applications, but also itself has sufficient potential to become the core of cloud technologies for the development of multi-user information systems and web services.

REFERENCES

AngularJS – Superheroic JavaScript MVW Framework. (n.d.). Retrieved October 15, 2020 from http://angularjs.org

Backbone.js. (n.d.). Retrieved October 15, 2020 from http://backbonejs.org

Bootstrap. (n.d.). Retrieved October 15, 2020 from http://getbootstrap.com

Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. (n.d.). Retrieved October 15, 2020 from https://www.w3.org/TR/CSS22/

Crane, D., & Pascarello, E. (2006). *Ajax in action.* Moscow: Ed. house "Williams".

Drupal – Open Source CMS. (n.d.). *drupal.org*. Retrieved October 15, 2020 from https://drupal.org

ECMAScript Language Specification – ECMA-262 Edition 5.1. (n.d.). Retrieved October 15, 2020 from http://www.ecma-international.org/ecma-262/5.1

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2001). *Techniques for Object-Oriented Design. Design patterns.* SPb.: Peter.

HTML 4.01 Specification. (n.d.). Retrieved October 15, 2020 from https://www.w3.org/TR/html401

jQuery. (n.d.). Retrieved October 15, 2020 from http://jquery.com

jQuery UI. (n.d.). Retrieved October 15, 2020 from http://jqueryui.com

Knockout: Home. (n.d.). Retrieved October 15, 2020 from http://knockoutjs.com

MediaWiki. (n.d.). Retrieved October 15, 2020 from http://www.mediawiki.org/wiki/MediaWiki

Medvedev, A. (2013). Cloud technologies: development trends, examples of execution. *Modern automation technologies*, *2*, 6–9.

MVP architecture. (n.d.). Retrieved October 15, 2020 from http://www.gwtproject.org/articles/mvp-architecture.html

Papadopoulos, A., & Katsaros, D. (2011). Distributed Indexing of Multidimensional Data for Cloud Computing Environments. *Third IEEE Intl Conf. on Cloud Computing Technology and Science* (pp. 407–414). IEEE.

Stefanov, S. (2011). *Javascript. Patterns.* St. Petersburg: publishing house Symbol-Plus.

w2ui: Home (n.d.) JavaScript UI. Retrieved October 15, 2020 from http://w2ui.com/web

Zepto.js: the aerogel-weight jQuery-compatible JavaScript library. (n.d.). Retrieved October 15, 2020 from http://zeptojs.com