A Simple MVC Framework for Widget Development

Behnam Taraghi[1], Martin Ebner[1]

[1]Graz University of Technology,
Social Learning, Computer and Information Services,
Graz, Austria
b.taraghi@tugraz.at
martin.ebner@tugraz.at

Abstract. The great advantage of a Rich Internet Application (RIA) is the improved performance, since a great part of the processing can be done on the client-side rather than the server-side. For many web-based RIAs JavaScript (JS) is the most common programming language as it is popular as a script language for browsers. While server-side programming languages have the advantages of object oriented programming paradigm, JS bases on objects with a specific object literal notation. Using design architectures such as Model View Controller (MVC) in JS reduces the code complexity and allows semi-parallel application development. It causes an easier and much less time-consuming development on further extensions of RIAs. Web-based widgets are actually RIAs. In this paper a very simple design pattern for widget development is introduced that bases on MVC design architecture. The pattern has been applied during the development phase of widgets for the Personal Learning Environment (PLE) at Graz University of Technology (TU Graz) and is mostly appropriate for students and RIA developers who have beginner knowledge/experience in JS programming.

Keywords: PLE, Personal Learning Environment, Widget, MVC, Rich Internet Application, RIA

1. Introduction

Nowadays there are many Web 2.0 applications and information systems on the World Wide Web. Since the very first beginning of the so called ReadWriteWeb [1] and following the first attempts on e-Learning 2.0 [2], educational organizations in particular universities provide various services to the students online. Former used rigid Learning Management Systems (LMS) are enhanced by different online applications [3] that may be used by learners on the WWW such as social networking systems, blogospheres, social bookmarking, YouTube, flickr, slideshare etc [4]. Due to the enormous and rapid growth of different applications for different goals on the WWW teachers as well as learners, especially novice users are overwhelmed by the huge number of possibilities on the Web and to some extent cannot find the ones they need and would be best suitable for them [5].

To overcome the challenge of managing the distributed and potentially unknown resources and applications, a first prototype of a Personal Learning Environment (PLE) has been developed at Graz University of Technology (TU Graz) since 2009 [6]. The basic architecture of PLE is a mashup [7] of widgets. For each service a widget is provided that follows an extension of the W3C widget specifications [8]. The PLE, its requirements and its technological concept are described in detail in a former publication at MUPPLE09 workshop [9] for further reading.

Users should be able to access the PLE (and widgets) by desktop PCs, laptops or other devices such as mobile phones. Furthermore the PLE and therefore each widget must be easily extendable in sense of features they provide and adaptations to new technologies. To meet this goal a suitable framework must be used in the initial design phase to guarantee the greater scalability and less complexity. In this publication we introduce a design pattern based on Model View Controller (MVC) design architecture that is used for the development of widgets in PLE. It can also be applied actually in any other Rich Internet Application (RIA) developed in JavaScript (JS). The biggest advantage of the introduced framework, in comparison with other conventional frameworks, is its simplicity and resemblance to the Object Oriented (OO) programming in server-side languages. Especially for the developers who are not advanced JS programmers or have beginner knowledge/experience in client-side programming, but are familiar with OO paradigm this framework is an appropriate tool to start.

2. MVC frameworks

RIAs are revolutionizing users' experiences. They offer a more interactive experience to users. The great advantage of a RIA is an improved performance, since a great part of the processing can be done on the client-side rather than the server-side. For many web-based RIAs JS is the most common programming language because of its popularity as a script language for browsers. While server-side programming languages have the advantages of object oriented programming paradigm, JS bases on objects with a specific object literal notation. There are some different ways to implement inheritance and class-like structures in JS as described in [10]. Using design architectures such as MVC in JS reduces the code complexity and allows semi-parallel application development.

MVC design architecture bases on the separation of three main distinctive components of an application: (i) the presentation layer (view) that deals with the visual display of the application and the whole Graphical User Interface (GUI), (ii) the logic layer (controller) that describes the behavior of the application and connecting components, presentation and data layer to each other. (iii) The data layer (model) that is responsible for the data collected on the client-side. Figure 1 displays the structure of an MVC framework.

There are already some works in filed of MVC design architecture for JS. JavaScriptMVC[1] is one of the open-source frameworks. It provides additionally to

[1] http://javascriptmvc.com/ (last visit: 2010-06-24)

MVC architecture, features such as concatenation, compression, testing modules, error reporting etc. TrimJunction[2] is a clone of the Ruby On Rails[3] web MVC framework for JS. PureMVC[4] provides a lightweight implementation of MVC design architecture in different programming languages including JS. Sproutcore[5] is an HTML5 application framework for building responsive client applications in modern browsers and bases on MVC design architecture.

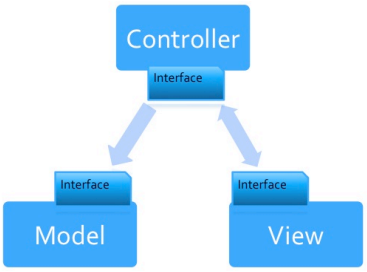

Fig. 1. MVC structure

3. The simple MVC framework

At TU Graz students of informatics are implementing widgets for the PLE as shown in [6]. They are mostly experienced programmers in server-side programming languages, know the OO paradigm very well and are familiar with different MVC frameworks in server-side programming languages such as Java or PHP. On the other hand not all have very much experience in client-side programming such as necessary for JS. The conventional MVC frameworks provide (advanced) developers with features that are beneficial for implementing advanced RIAs. The support for these features increases the complexity of the frameworks, which is on the other hand time-consuming to learn and hence a disadvantage for beginners or less experienced JS developers. In order to make the widget development for students as simple as possible we applied a very simple and lightweight MVC architecture that aims to get very easy to start with. It is realized through module pattern. Each three components of MVC is actually a module in JS. Modules use a singleton paradigm, support private data and stay out of the global namespace. These features suit module pattern to be applied for model, view and controller in projects that may grow in the course of time and become more complex. The modules themselves are implemented using closure functions in JS. In Douglas Crockford's book [10] modules, closures and singleton paradigm are very well described for further reading.

The low experienced students who are actually advanced OO developers in server-side programming languages need to consider the modules as three static classes with

[2] http://code.google.com/p/trimpath/wiki/TrimJunction (last visit: 2010-06-24)
[3] http://rubyonrails.org/ (last visit: 2010-08-19)
[4] http://trac.puremvc.org/PureMVC_JS/ (last visit: 2010-06-24)
[5] http://www.sproutcore.com/ (last visit: 2010-06-24)

the ability to contain private and public methods or variables. They can then start implementing the three modules as they are used to in server-side OO programming languages. They can extend the framework by adding additional sub modules to support features that are provided by conventional MVC frameworks such as templates, event handling etc.

Figure 2 demonstrates a very simple example of the MVC architecture based on the described structure above for deeper comprehension.

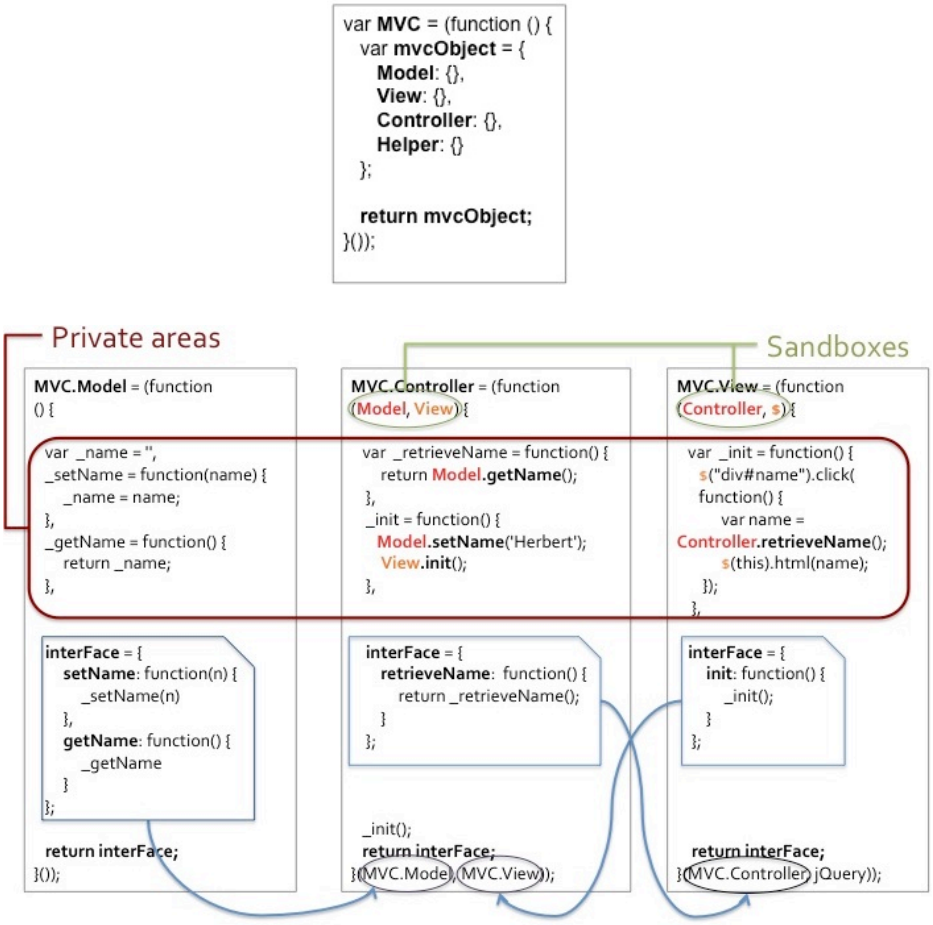

Fig. 2. Simple MVC framework in JS used for developing web-based widgets

The outlined private area in figure 2 contains private members that are valid only within the scope of the corresponding module. The "interFace" objects contain the public members of corresponding modules. The public members can access to the private members as expected. The view and model modules are in the sandbox of controller (passed as arguments of the closure function: "MVC.Model" and "MVC.View"). Consequently the controller can call the public members of view and model through the local objects "Model" and "View". Correspondingly the view can

access the public members of controller for event handling and the methods of jQuery[6] framework for DOM manipulation.

It is possible to define some default public members in each module directly in the definition of MVC closure. If a module contains already such default members, its "interFace" object must not be initialized, but be extended in order to avoid unsetting the default-defined members. An example of such case can be seen in figure 3. The "interface" object of the view module that is defined in that example is extended so that the default members, if set any, be maintained.

In case of widgets we need a special type of functionality such as distribution of user requirements in multiple pages and navigation between pages in UI. This could be realized e.g. with the help of a "page" module as additional module or a sub module in views. Sub modules can be used as helpers in all three main modules and have exactly the same structure and benefits as the main modules.

With the help of such architecture a very simple framework is realized, which resembles the classes and their public and private members in server-side programming languages. The developers have the possibility to control whether (sub) module A is allowed to access (sub) module B by setting the (sub) module B in the sandbox of (sub) module A. Through this approach all (sub) modules and their members stay out of the global namespace, the code is readable, maintainable and easily extendable.

3.1 Extendibility examples

As mentioned above through the use of module pattern the MVC architecture is easily extendable. The students can use their experience from server-side programming to add additional (sub) modules and refine the JS code in detail. We consider here some extensions in view and model and try to hold the examples very simple and primitive for better understanding.

3.1.1 View extension examples

As an example of extension possibilities we can consider the view module and extend it by using templates. View is the actual presentation layer in MVC design architecture. The public methods in view are normally called by the controller to display the retrieved data in the UI. In the shown example in figure 2, View.init() is called. When the user clicks on an html element, some data (here the "name") is retrieved by the controller and put into a node in Document Object Model[7] (DOM). Templates can be used to reduce the complexity of view and separate the HTML layout from the actual view's functionality. The web designers are able to create the templates independent of the programmers who are responsible for three main units (model, view and controller). Figure 3 shows this extension in a very simple style for deeper understanding.

---

[6] http://jquery.com/ (last visit: 2010-08-19)

[7] http://www.w3.org/DOM/ (last visit: 2010-08-19)

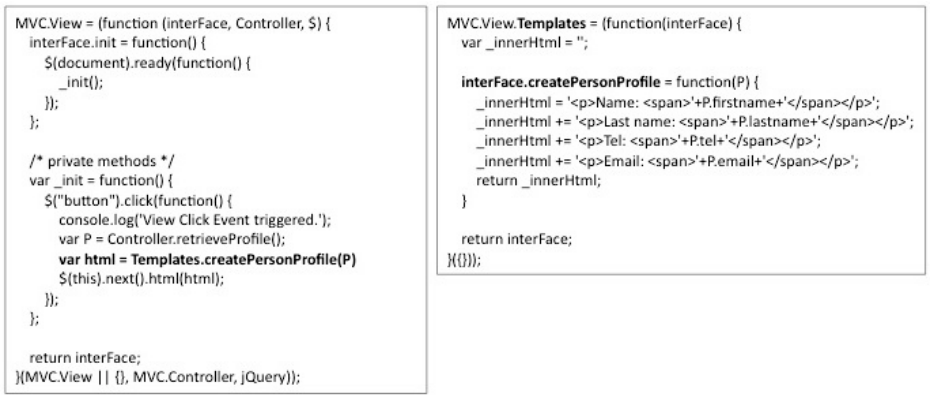

Fig. 3. Extension of the view to use HTML templates by adding a new sub module "Templates"

Different templates can be put in one sub module for view. For the case of deeper understanding this example is held as simple as possible. In real applications different template engines can be applied such as JavaScript Templates (JST)[8], PURE JavaScript Template Engine[9], Closure Templates[10] and jQuery template plugin[11].

3.1.2 Model extension examples

As another example the model can be extended. Model acts as the data layer in MVC framework and can be responsible for data retrieval from data resources (from same domain or remote servers) through XML HTTP Requests (XHR). Creating a sub module for XHR requests can bring some benefits. Different XHR sub modules (ajax, cross domain requests, etc.) can stand in model for data retrieval from a remote server. The sub module can be configured for different widget specifications. For instance, in the first case the developed widget is applied as a gadget in iGoogle and therefore the widget must be adapted correspondingly. In this case no total refactoring is actually needed in model. The XHR sub module must only be extended to use proxies specified for iGoogle gadgets. For W3C widgets and other specifications the same approach must be applied.

Another example is the use of one of HTML 5 features for caching in model. Caching is an efficient approach to increase performance in RIAs. Local storage can be used to cache data on browser side. This approach can be applied for instance by using jStorage[12] in a sub module. jStorage is a simple plugin for Prototype[13],

[8] http://code.google.com/p/trimpath/wiki/JavaScriptTemplates (last visit: 2010-08-19)
[9] http://beebole.com/pure/ (last visit: 2010-08-19)
[10] http://code.google.com/intl/de-DE/closure/templates/ (last visit: 2010-08-19)
[11] http://github.com/nje/jquery-tmpl (last visit: 2010-08-19)
[12] http://www.jstorage.info/ (last visit: 2010-08-19)
[13] http://www.prototypejs.org/ (last visit: 2010-08-19)

MooTools[14] and jQuery to cache data (string, numbers, objects, even XML nodes) on browser side, making use of HTML 5 local storage.

Figure 4 shows the model extensions with XHR and localStorage. In this example the data is retrieved from remote resources through XHR sub module and saved on local storage if the data is not already saved before. Otherwise the saved data is returned directly from local storage. The XHR sub module is extended to support W3C widget specification. In the case of W3C widget the specified proxy is used. Otherwise (i.e. for the case of a normal web-based widget) it is assumed that the remote service is on the same domain and hence a normal ajax request is sent.

```
MVC.Model = (function (interFace) {
  interFace.getListOfCourses = function(addr) {
    return _getCourses(addr);
  };
  /* private methods */
  var requestIsActive = false,
  _getCourses = function(addr) {
    var value = LocalStorage.get(addr);
    if(!value){
      // if not - load the data from the server
      if(!requestIsActive) {
        requestIsActive = true;
        XHR.fetchCourses ('http://example/service', addr,
        function(data) {
          value = data;
          // and save it
          LocalStorage.set(addr,value);
          requestIsActive = false;
        });
      }
      return _getCourses(addr);
    }
    else return value;
  };
  return interFace;
}(MVC.Model || {}));
```

```
MVC.Model.XHR = (function ($, spec) {
  var interFace = {
    fetchCourses: function(url, user_id, callback){
      _sendRequest(url, {'L': user_id}, callback);
    }
  },

  _sendRequest = function(service_url, param, callback){
    switch(spec) {
      case 'W3C':
        proxy.send(service_url, param, callback);
      break;
      default:
        $.getJSON(service_url, param, callback);
      break;
    }
  };

  return interFace;
}(jQuery, Config.specification));
```

```
MVC.Model.LocalStorage = jQuery.jStorage;
```

Fig. 4. Extension of the Model to make use of a separate XHR sub module and HTML 5 local storage. The XHR sub module supports data retrieve for W3C widgets.

Conclusion

In summary the framework introduced in this paper provides the widget developers with some benefits. The biggest advantage of this framework, in comparison with other conventional frameworks, is its simplicity and resemblance to the OO programming in server-side languages. The separation of modules allows a semi-parallel application development. UI designers can implement the view templates while logic programmers build the model and controller and view's functionality. This would definitely lead to a faster widget development. Complexity is reduced and the scalability is supported. Breaking down the code to distinctive separate sub

[14] http://mootools.net/ (last visit: 2010-08-19)

modules make further developments on extensions much easier and less time consuming. For instance to extend a widget to HTML 5 features such as local storage the developers would merely need to add a sub module “localStorage” to the model. As usually necessary, some lines of code must be added to the model in order to call corresponding methods of “localStorage”. Once the module is implemented it can be applied in other widgets as well. The widgets can also be configured to run under different specifications, assumed that the corresponding sub modules are implemented respectively. The pattern described in this paper has been already applied for widgets that are developed for the Personal Learning Environment at Graz University of Technology. Our experience with applying this framework in widget development showed that the students who have beginner knowledge/experience in JS development could start developing widgets (and actually any JS based RIA) very quickly. They were so creative that they refined the code in sub modules that are mentioned partially above.

References

1. O’Reilly, T.: What is Web 2.0 – Design Pattern and Business Models for the next Generation of Software (2005), http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (last visited in December 2009)
2. Downes, S.: e-Learning 2.0, ACM e-Learn Magazine (2005), October 2005 (10)
3. Ebner, M., Scerbakov, N., Taraghi, B., Nagler, W. & Kamrat, I.: Teaching and Learning in Higher Education – An Integral Approach. In C. Crawford et al. (Eds.), Proceedings of Society for Information Technology & Teacher Education International Conference 2010 (pp. 428-436). Chesapeake, VA: AACE (2010).
4. Schaffert, S., Ebner, M.: New Forms of and Tools for Cooperative Learning with Social Software in Higher Education. In: Brayden A. Morris & George M. Ferguson (Ed.), Computer-Assisted Teaching: New Developments. Nova Science Pub, p. 151-165 (2010).
5. Nagler, W., Ebner, M.: Is Your University Ready For the Ne(x)t-Generation?, Proceedings of 21st ED-Media Conference, pp. 4344-4351; World Conference on Educational Multimedia, Hypermedia and Telecommunications (2009)
6. Ebner, M., Taraghi, B.: Personal Learning Environment for Higher Education – A First Prototype. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010 (pp. 1158-1166). Chesapeake, VA: AACE (2010).
7. Tuchinda, R., Szekely, P., Knoblock, C.: Building Mashups By Example. ACM Proceedings of IUI 2008, Maspalomas, Spain (2008)
8. Widgets 1.0 Packaging and Configuration - W3C Working Draft, http://www.w3.org/TR/widgets/ (2008)
9. Taraghi, B., Ebner, M., Schaffert, S.: Personal Learning Environments for Higher Education: A Mashup Based Widget Concept, Proceedings of the Second International Workshop on Mashup Personal Learning Environments (MUPPLE09), Nice, France (2009), ISSN 1613-0073, Vol-506 http://ceur-ws.org/Vol-506/
10. Crockford, D.: JavaScript: The Good Parts. O'Reilly Media / Yahoo Press (2008)