WEB APPLICATION FRAMEWORK FOR INFORMATION SYSTEMS DEVELOPMENT

Miroslav Raška
Bachelor Degree Programme (4), FIT BUT
E-mail: xraska09@stud.fit.vutbr.cz

Supervised by: Marek Rychlý
E-mail: rychly@fit.vutbr.cz

Abstract: Web applications development is experiencing specific difficulties in presentation layer which could be solved by a framework described in this article. Selected difficulties are introduced, including the desired solutions, possible outcomes and comparison with existing web application frameworks. Basic architecture with central message bus and general structure of components is explained, supplemented by examples of usage. Finally, possible effects on devepolment process of applications based on this framework are discussed.

Keywords: Web application framework, Software architecture, Rich Internet applications, RIA, Single-page applications, Information systems, Front-end, Dart, JavaScript

1 INTRODUCTION

The development process of web applications is experiencing several technical difficulties in presentation layer that complicate and delay projects. Emerging web technologies are designed to solve the basic issues present in currently used languages and libraries. While technicalities are being covered, an approach focused on application-wide topics that work with broader context is still missing. The purpose of this work is to create web application framework oriented on proper architecture, simplification of complex interactions and internal security. These are some of the issues arising with the boom of single-page cloud applications, e.g. highly interactive enterprise information systems.

2 MOTIVATION AND CHALLENGES

The existing frameworks are usually built around UI components and remaining non-visual functionality is made to fit the UI paradigms. Major frameworks have MV* architecture[1] [2] that is a software pattern for specific usage but here it constitutes the whole framework architecture, neglecting other suitable design patterns. The new **framework architecture should be designed around non-visual functionality**, leaving the implementation of UI components to external libraries. After analysing frameworks additional challenges were found unresolved:

- **Simplify implementation of user interactions.** Application business logic could be very complicated, one user input may result in triggering actions in many related components, some of which might not yet exist, be rendered or visible. In combination with many conditional statements the logic is complex and framework should contain simple yet error-proof solution.
- **Introduce internal security mechanisms [2][4].** By default components have access to every DOM element and every object. Errors in code can therefore cause unexpected behavior in any

[1] Also called MVW (Model-View-Whatever); term covers e.g. MVC, MVVM and MVP design patterns. [1]
[2] Comparison of frameworks is available at http://todomvc.com/.

part of the application or create serious memory-leaks. Both cases are usually hard to detect and locate. To prevent this behavior components will have limited access within application.

- **Add extension points wherever possible.** Although the framework should be suitable for most of the target applications, it should allow users to provide own implementation of any independent part without modifying framework source code. Additionaly, this will also facilitate the use of UI components from any existing library.
- **Create component hierarchy based on their funcionality.** Web frameworks work with the basic DOM hierarchy, some of them also create hierarchy of UI components which is, however, related to component position in the DOM and limited to visual components. New type of hierarchy will combine visual and non-visual components and will be independent on the DOM structure. This will help with internal security and business logic implementation.
- **Make application structure and source code easy to understand [2].** Framework should contain well chosen directory structure template for new applications. Clarity within components can be enforced by specific structure of the base class. In addition to technical documentation, set of general rules and recommendations for every application will be included.

The description of a solution to these issues follows. Two sections describe new web framework structure through its architecture and structure of every application component.

3 ARCHITECTURE

The main part of the framework architecture is a *message bus* responsible for the communication between components and external environment. These basic types of communication are included by default, other types of communication can be user-defined:

- **Events**: are dispatched using publish-subscribe pattern [3]. Events with the same name can be published by multiple components and subscribers do not have access to the source component. Events are asynchronous and they are not cached or stored for later access.
- **Services**: only one component in application can provide service with specific name. Subscription is one-time only, functionality is synchronous therefore result is immediately available. Compared to events, the component providing service must be present at the subscription time.
- **Security checks**: before every previously mentioned message is sent, special messages are dispatched to verify whether the normal message is allowed to enter the bus.

All components can work with events and request services. Access to security checks and provision of services is limited to special components – application singletons [3] called *managers* (sometimes also *providers* or *adaptors* to clarify their role [3]). Figure 1 shows communication flows in a sample application. Following two scenarios demonstrate some of the advantages of this architecture.

Domain Data Provider provides access to the database at server side when Internet connection is present. When entering offline mode it is replaced with provider working with browser local storage. This dynamic change of managers is done without affecting the rest of the application because the API of both managers is identical and components have only indirect access to manager services.

DOM Provider restricts access to DOM elements, so that every UI component can only see one subtree of DOM. This subtree is automatically determined by component position in UI components hirarchy stored in **UI Hierarchy Manager**. Because all elements are manipulated through this provider, no component can accidentaly change the DOM of other component.

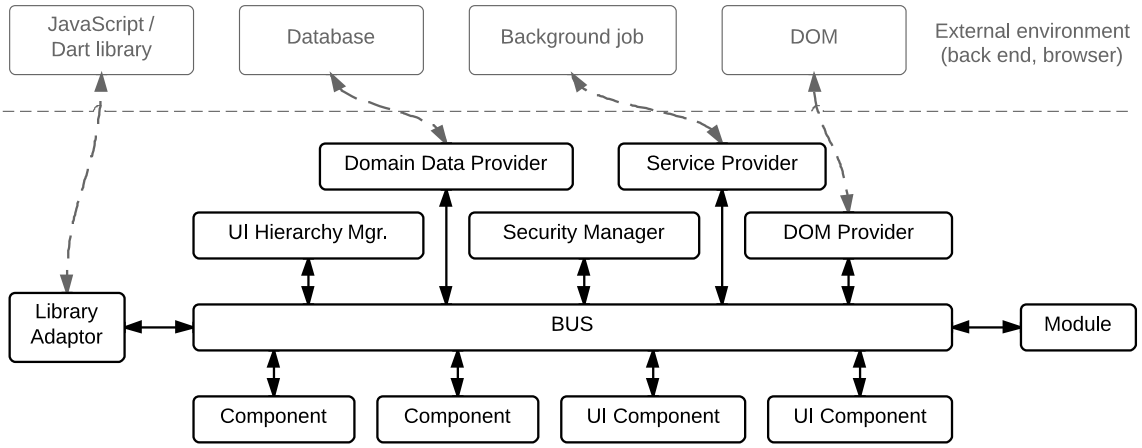

Figure 1: Communication flows in sample application with one central message bus.

4 STRUCTURE OF COMPONENTS

Every component is configured by special object that is locked for changes when component is fully initialized, therefore components cannot be dynamically reconfigured and the behavior remains consistent. Dynamically changing behavior can be done using decorator pattern. When components expose their API (events, services) on the bus, it is also locked for changes. The extensibility is made possible by internal hooks that can be used by any subclass or mix-in. The clarity of code is improved by separate configuration that removes many fields from class definitions. Observe that UI components do not differ from other components, they just use services of DOM Provider more often.

5 FINAL REMARKS

Dart language and its libraries were chosen as the base for the implementation. Dart solves many of the language issues present in JavaScript and it is already available in a stable version. This base technology combined with well-designed architecture and internal security should prevent programmers from making mistakes that are common to today's web applications.

Another vital point is that it is more difficult to design an application complying to framework principles. Therefore more effort on analysis of component behavior and structure will be made and the application will be better designed. Higher initial effort is benefitial in long term in less errors to fix. Extensibility options allow this framework to power most types of web applications.

ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-S-14-2299.

REFERENCES

[1] FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003. ISBN 978-0-321-12742-6.

[2] ZAKAS, Nicholas. *Maintainable JavaScript*. Sebastopol: O'Reilly, 2012. ISBN 14-493-2768-0.

[3] GAMMA, Erich, R. HELM, R. JOHNSON and J. VLISSIDES. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston: Addison-Wesley, 1995. ISBN 02-016-3361-2.

[4] ZAKAS, Nicholas. *Scalable JavaScript Application Architecture* [online]. 2009 [cit. 2014-03-02]. Available from: http://www.slideshare.net/nzakas/scalable-javascript-application-architecture