Literature Review
Service Frameworks and Architectural Design Patterns in Web Development

Connor Patrick
ptrcon001@myuct.ac.za
Computer Science Honours
University of Cape Town

15 May 2014
Abstract

Organizing and managing of large scale events such as eSports tournaments is a challenging task. With potentially thousands of participants a Web-based service is the ideal tool for such a task. But for such a Web application to be effective it must be well designed. Performance, scalability and simplicity are just a few important aspects of a well-designed Web application. Web service frameworks and architectural design patterns exist to help the developer in the designing of an optimal Web application, however there are many options to choose from when considering a which to use. Web service frameworks are used to standardize the way in which Web applications communicate and invoke services. Architectural design patterns are solution templates used in the structuring of the software components of the application. Microsoft’s .NET framework has become a popular choice amongst developers for the building of interactive Web applications. As such the Web service frameworks (SOAP, RPC and REST) and architectural design patterns (Multilayer, PAC and MVC), which can be used with the .NET framework, are described and compared. The performance, scalability, security and simplicity of each these choices are evaluated with the intention of singling out the best all-round option. The evaluation of the Web service frameworks, as well as the results of previous studies, showed that REST far outmatched both SOAP and RPC in all of the above mentioned areas. The best choice for design pattern was MVC. Although it is similar to PAC it is slightly more efficient and the use of multiple view components would make it easier to design a Web application for multiple platforms. The compatibility of REST and MVC with Microsoft’s popular .NET framework provided further motivation towards their use.
Contents

1. Introduction and Motivation ...........................................................................................................................................2
2. Web Service Frameworks ..................................................................................................................................................2
   2.1 Simple Object Access Protocol (SOAP) .....................................................................................................................2
   2.2 Remote Procedure Call (RPC) ....................................................................................................................................3
   2.4 REpresentational State Transfer (REST) .....................................................................................................................3
3. Architectural Design Patterns .............................................................................................................................................3
   3.1 Multilayer .................................................................................................................................................................3
   3.2 Presentation-Abstraction-Control (PAC) ....................................................................................................................4
   3.3 Model-View-Controller (MVC) ................................................................................................................................4
4. Critical Analysis ...............................................................................................................................................................5
   4.1 Frameworks ...............................................................................................................................................................5
   4.2 Design Patterns ..........................................................................................................................................................5
5. Conclusion ........................................................................................................................................................................6
6. References ........................................................................................................................................................................7
1. Introduction and Motivation

With the recent rise in popularity of eSports many Gaming Leagues have come into existence. These leagues arrange and manage tournaments with thousands of competing gamers in short spaces of time at regular intervals. Most of them make use of a website upon which competitors register and sign-up for a desired tournament. Websites have proven in this regard to be a useful tool for the spreading of information and the organisation of such events. But for a website to be effective it needs to be well designed so as not to discourage the users.

Various software engineering tools and standards have been developed that are used in the design of professional or commercial Web applications. Examples of such standards would be the Web service frameworks and software design patterns. Microsoft’s .NET Web API framework has become popular over recent years, as such the standards that will be looked at in this review can be used alongside it (Nein, 2014).

A Web framework is made up of a collection of software components that aids developers in the creation and execution of Web-based user interfaces, (Vosloo & Kourie, 2008). The framework manages the content displayed on the Web interface as well as which pages to display and what actions are available to the user of the page. A Web service framework is used to standardise the ways in which communication takes place between Web applications, (Mohamed & Wijesekera, 2012).

Web services are software systems that are designed to support interoperable, machine interaction across networks, (Shi, 2006). These services are used by Web applications for communications between other systems in the form of requestors and providers, (Dezhgosha & Angara, 2005). They make use of both HTTP and XML, along with a minimum of one of the available open standard technologies, such as SOAP or REST.

Design patterns are solutions that have been designed to be reusable; they can be used to in any system with similar functionality requirements, (Thung, Ng, Thung, & Sulaiman, 2010). They assist the developer in the design and implementation of the system, as well as reducing production time.

Software architectural design patterns are design patterns that are used in the design of the structuring of software components in a system, (Wang & Burge, 2010). This works as a general foundation for more specific problems. Popular design patterns are designed such that the solutions they provide are as efficient as possible and that the software using the pattern is easily maintainable, (Thung, Ng, Thung, & Sulaiman, 2010).

This literature review looks at the popular frameworks and architectural design patterns that can be used in the development of Web-based applications alongside Microsoft’s .NET framework. These will be discussed and compared in order to determine the best choices available for the design of a website to be used as an eSports portal.

2. Web Service Frameworks

2.1 Simple Object Access Protocol (SOAP)

SOAP is a Web service framework protocol developed by Microsoft which is designed to work with HTTP messages. The underlying format of the framework is Extensible Markup Language (XML), (Mohamed & Wijesekera, 2012), (Nein, 2014). A SOAP-based system is designed primarily around the transmission of XML messages between a provider and a requestor. The service sets offered by SOAP as well as the interface are defined in XML files that adhere to the grammar specified by the World Wide Web Consortium (W3C), this language is known as a Web Services Description Language (WSDL) and the files are referred to as WSDL files, (Mulligan & Gracanin, 2009). Transmitted messages can be either synchronous or asynchronous, in the case of synchronous messages Remote Procedure Call (RPC) will be
used. In some cases the service providers register their services in a public or private registry known as a service broker. Universal Discovery, Description and Integration (UDDI) is an open standard technology that can be used to provide such a registry. The service requestor accesses this registry to view the information in the WSDL files stored there, it then uses this information to construct a SOAP request which is sent to the service provider to invoke a service, (Dezhgosha & Angara, 2005).

2.2 Remote Procedure Call (RPC)
RPC is based on the distributed objects movement which focuses on network object communications in object-oriented systems. Most Service-Oriented Architectures (SOAs), which aims to standardise the way in which applications access services, make use of RPC.
RPC views the server as a container of procedures which the client can call upon. RPC, like SOAP, makes use of UDDI repositories to store information regarding available services. It also makes use of SOAP messages to call procedures. Procedures are in verbal form. (Feng, Shen, & Fan, 2009).
RPC elements in SOAP were gradually replaced with the exchanging of XML files as discussed previously, (Mohamed & Wijesekera, 2012).

2.4 REpresentational State Transfer (REST)
REST is a much more lightweight Web service framework than SOAP and RPC; it has a more concise style in terms of implementation as well as operation, (Song, Xu, & Liu, 2010). The framework uses the CRUD principle which has only four different behaviours: Create, Read, Update and Delete. REST works through the Hypertext Transfer Protocol (HTTP) and the four CRUD operations map easily to HTTP methods: GET, POST, PUT and DELETE. These operations are atomic; they cannot be decomposed further. REST development reduces the complexity of the system and increases the scalability. In REST every resource has a unique URL to identify it, these URLs can operated upon by any or all of the HTTP commands, (Mulligan & Gracanin, 2009). REST clients are able to easily retrieve and work with very specific resources on any server it is connecting to.
REST uses a stateless communication protocol, which means that all client requests are independent of other requests and each contains all the information required for the server to fully understand the request, (Mohamed & Wijesekera, 2012). REST requires the following of the architecture:

- The system is a client-server model
- Requests are stateless
- Responses are cacheable
- A layered system is used to improve scalability, performance and security
- Resources have unique names and valid access points
- All resources are accessed through a uniform HTTP interface
- Resource representations are interconnected with URLs

A RESTful system is one in which Web services make use of HTTP and REST principles, (ibid).

3. Architectural Design Patterns

3.1 Multilayer
In a multi-layered architecture the system is separated into logical groupings of software components called layers. Layers can communicate with each other, as well as with other applications. Layers are divided such that each of them specialises in a specific kind of task. Tasks can be divided up further and assigned to sub-layers, (Wang & Burge, 2010).
There can be any number of layers, however the typical approach is to use a three layered model. The three typical layers are:
- **Presentation Layer**: This layer handles user interactions and is responsible for what the user sees.
- **Business Layer**: The core functionality exists on this layer. System logic is split into various components.
- **Data Layer**: This layer handles the accessing of data, both local data (locally stored databases) and data available through the network.

Layers can be split across various numbers of machines, referred to as tiers. In a single tier system all layers exist on one machine. In a two-tier system the data layer are database is usually on the server while the other two layers on the client side. In a three-tier system, which is the standard approach, the presentation layer is on the client side, the business and data layers are on the server side and the database is on a separate database server, (Microsoft, 2014).

### 3.2 Presentation-Abstraction-Control (PAC)

PAC is an architectural pattern that was designed for interactive systems. In a system there would exist a hierarchy of agents, each of which would be responsible for a single function, (Thung, Ng, Thung, & Sulaiman, 2010). The presentation, abstraction and control layers are split such that the user interface of an agent is completely separate from the main functionality of the agent. Agents are operated upon independently and develop as such, but they cooperate to generate views. Agents are split into three levels (top, intermediate and bottom) allowing the data model and user interface of the system to develop independently of the other. The levels are able to collaborate with each other though. Changes to the presentation and abstraction layers do not necessarily have an effect on other agents. The use of multiple independently developing agents allows for PAC to make use of multitasking as each agent can easily be run on separate threads. As such PAC works well with multi-user systems. Gathering information from all the various agents does however present a control problem, (ibid).

### 3.3 Model-View-Controller (MVC)

The MVC pattern is one of the most widely used design patterns in software development, (Prajapati & Dabhi, 2009). It is an object-oriented design pattern that consists of three layers; the model, the view and the controller. The model encapsulates the system logic and data. The view is the presentation layer of the system (the interface). The controller controls the behaviour of the system through user and system input; it interprets view-generated events, (Thung, Ng, Thung, & Sulaiman, 2010). This particular pattern has been designed such that it is easy to separate the user interface from the application data and from the functionality of the system. This allows developers to easily change any of these layers without affecting the others. This allows for a long-term, easily maintainable system, (Worrall & Chaussealet, 2011). The following figure depicts how a standard MVC pattern behaves.

[Diagram of MVC pattern]

Adapted from Figure 1. MVC Design Pattern | Prajapati & Dabhi, 2009|
4. Critical Analysis

4.1 Service Frameworks
The RPC aspects of SOAP protocols have been gradually phased out, either in preference of XML file exchange or in preference of REST, (Mulligan & Gracanin, 2009). Each RPC interface defines its own services with its own semantics, drastically limiting the scalability of the framework, (Feng, Shen, & Fan, 2009). This in contrast to REST which has a uniform interface because of its use of HTTP and CRUD. This combined with its use of stateless interactions makes REST highly scalable.

SOAP messages require XML wrappers on all messages, whereas REST does not. This means that SOAP messages can be up to ten times the size of REST messages, which is a significant performance issue. Overall REST has better performance, particularly on mobile devices, than SOAP, (Mulligan & Gracanin, 2009).

When SOAP/RPC messages are sent across a network they are wrapped in an HTTP header, this allows the messages to easily get through a firewall as the server will only understand the messages intent once it has been parsed. This allows for malicious commands to get through the firewall undetected. With REST everything is abstracted into a resource and all resources are assigned a unique URI. Specific HTTP commands for each specific resource can be release separately to a client and all this is handled on the HTTP firewall making REST more secure. (Feng, Shen, & Fan, 2009).

REST’s direct use of HTTP for data exchange means that the system does not need to parse and package the messages as it does for SOAP. It also makes use of caching to prevent unnecessary calculations and improving performance, (ibid).

A large number of RESTful Web services have been specifically designed as alternatives to inefficient SOAP services. These new services have been improved such that they are more scalable, are more interoperable and have simpler and more flexible interfaces, (Mulligan & Gracanin, 2009).

4.2 Design Patterns
The multi-layered approach is quite popular in most Web applications, with the number of layers depending on the particular application. Both PAC and MVC can be, and often are, implemented on top of a multi-layered architecture. The approach helps with system modularity; by splitting up the system into layers each layer can be developed independently of the others. It also allows the system to extend across multiple machines. Ultimately the use of a layered architecture is dependent on the system being developed.

MVC and PAC do have a number of similar properties. They both split the system into three sections, they each have a control layer and a layer that handles the presentation of the system. The main difference between the two is that they handle input and output differently as well as synchronisation, (Thung, Ng, Thung, & Sulaiman, 2010). In MVC output is handled independently from both the model and view layers whereas in PAC output is handled in the presentation layer alone. The presentation layer also handles the input.

PAC is more suited to multitasking due to the independency of its agents, however the hierarchy of agents does add to the complexity of the system. The communication between the numerous agents can result in overhead, thus lacking efficiency. (ibid)
5. Conclusion

From the above information it seems evident that of the frameworks discussed REST, with its superior performance and scalability, would be the most appropriate framework available. This notion is backed by studies done by Mulligan et al and Feng et al in which REST is compared with RPC and. Given that an eSports website could easily be accessed by a mobile device, or even a mobile application, an improved performance is very important. A third study by Mohamed et al which looked at using REST as a lightweight framework on mobile devices states that “...RESTful Web services are easy to deploy, publish, find and invoke...”. Together this seems sufficient evidence to conclude that REST is the most useful of the above mentioned Web service frameworks.

The most appropriate design pattern is not as clearly available however. The multi-layered architecture is often used alongside other design patterns and is not always a sufficient solution on its own. PAC and MVC are similar in design with neither significantly better than the other. However MVC does have a slight efficiency advantage over PAC which might make a noticeable difference in large systems. MVC is generally the more popular choice and with it allows for the treating of mobile interfaces as alternative View objects, making it easier to develop for multiple platforms. This makes it the more preferable choice in this particular case.

As both REST and MVC are compatible with the .NET framework, (Worral & Chaussalet, 2011), it would be ideal to build an interactive website such as an eSports portal with REST and MVC implementations.
6. References


