MyDuke.com Technical Overview
Stefan Negritoiu

stefann@stefann.com

1. Introduction

Planning and prototyping for the MyDuke project started in fall 1999. The initial goal was to create web mail software for the Duke University community. The deliverable would be a web interface to the email services provided by the institution's backend servers. At the time, the University's Office of Information Technology provided no such service and the situation was frustrating to many users. We set out to create such a service and upon building a prototype and receiving feedback from potential users we realized there was a growing need within the community not only for more accessible email services, but also for an improved experience with electronic resources on a broader scale.

We concluded that users would have a much better interaction and they would be more productive if access to most electronic services provided by the University would be made more straightforward and more ubiquitous. Since then, our focus is not solely on ubiquitous access to email, but on building an extensible framework that would provide a complete user experience to the community. It is important to clarify that, at present, this project does not intend to create or replace any back office electronic resources provided by the institution, but its focus is on easier to use and more accessible user interfaces to such resources along with the middleware that connects the two ends.

2. Logical architecture

Upon starting this project we evaluated many technologies for building web applications, everything from simple CGI based solutions to complex, integrated frameworks. Due to our belief in objected oriented design we chose a solution based on components, which provides a good opportunity for black box code reuse[1].
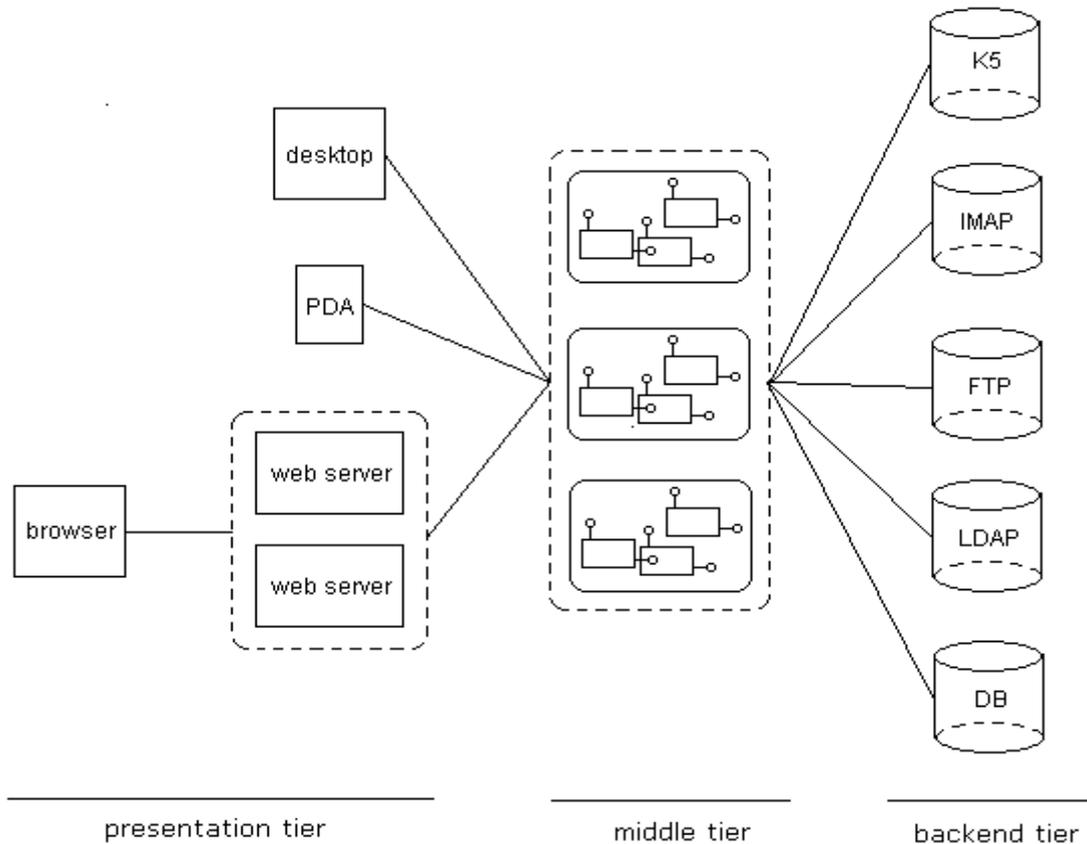
2.1. The middle tier

We decided to divide the web application into two tiers: a presentation tier and a middle tier. The middle tier is formed from several components each one acting as a client to one backend service. For instance, we wrote a component that acts as an IMAP client to the University's email servers. The component abstracts the message stores into a hierarchical object model, which feels more natural to developers of the presentation layer. That is, we turned the rather flat view of the TCP based protocol into a set of objects providing abstractions for message store, folder, message, etc.

Apart from the IMAP component, which solved the problem of accessing email servers, we also wrote components that implement the client side of the IMSP protocol for centrally stored configuration and address book data and the FTP protocol to leverage the institution provided storage space for work related and personal data storage. In the near future we plan to write more components to leverage even more of the back end electronic resources offered by the University such as an LDAP server and potentially a calendar and scheduling server.

Although all components are written to allow black box reusability either in desktop or web applications, we focused on their primary usage scenario, our web application. Thus, we abstracted another three concepts by building them into components: a transport pool for TCP based protocols,

---

[1] It is beyond the scope of this presentation to discuss the benefits and drawbacks of component based software design. There is much literature on the subject and I do not wish to duplicate it here.

a data encryption aware socket, and an authentication module. The transport pool is used on the middle tier to mange TCP connections to backend servers. It is a known fact that setting up and tearing down TCP connections is expensive, especially if encryption and authentication need to occur every time.

Instead of following a truly disconnected model at every tier, we chose to pool the connections such that one is established only upon a first request of the user. The middle tier will keep the connection open and will reuse it for every subsequent request of the same user. Connections are torn down upon a user explicitly ending his or her session, upon the server closing it due to inactivity or at the decision of the transport pool based on configurable internal policies. Using a transport pool does pose some scalability concerns because of operating system imposed limitations on the number of simultaneous open connections. In practice however, we found that a machine is saturated[2] due to other load factors, long before reaching this limit. Intelligent and thoughtful programming practices, like asynchronous (I/O) calls, become important in this scenario.

It is equally important to mention that using such techniques as transport pools makes state management in a cluster configuration more difficult. A socket connection is a hard piece of session state[3] and a user's session needs to be tied to a certain cluster machine to benefit from this type of pooling. Again, in practice we found that the benefits of using pooling at the connection level far outweigh the additional work for state management especially considering there might be other reasons in the overall design of the web application that might mandate tying a user's session to a certain cluster machine.

---

[2] By saturated I mean being maxed out due to user requests in terms of memory usage, processor load or any other metric which would prevent it to serve additional requests without a significant degradation in service level.

[3] By hard piece of session state I mean an immovable entity that is part of session state. For example, a soft piece of session state would be data sets indicating which books a user has ordered so far. They can be stored on a central session state server, which can be shared among cluster members, or one can imagine a session state replication scheme among cluster machines. Doing so with socket connections is neither desirable nor feasible.

The other two useful components we wrote allow us to design the main components in a transport agnostic fashion and allow easy switching between various authentication schemes. By programming against predefined interfaces we can have each component use normal sockets or sockets that encapsulate the traffic using SSL by modifying a single flag. The authentication module gives each component the ability to use custom authentication schemes like Kerberos instead of authentication directly to its corresponding server. All these features become crucial when considering that physical placement of the middle tier machines might put them on unsafe networks.

2.2. The presentation tier

Considering our initial goals, the principal presentation tier was a HTML-based website. The more we worked on the project and the more we understood the user scenarios, we realized that presentation tiers in the form of a desktop application and a mobile application were feasible in addition to the website. We reached an important milestone in the design of the project when we realized that as long as there existed a common, platform independent mechanism for performing calls on remote objects (aka RPC) we could have virtually any type of client running on any type of platform interact with our middleware.

Therefore, we designed our components not only with object oriented designs in mind, but also taking into account the possibility of them being used remotely by any number of clients. The standard that makes this possible today is SOAP, an XML-based RPC protocol. The presentation tier will eventually consist of a website targeted to desktops, a website targeted to wireless PDAs and mobile phones, a desktop application, and a PDA application.

All these incarnations of the user interface will perform remote calls into the middle tier. In addition, the components can be exposed to public consumption through web services enabling peer websites to make use of them or to synchronize with the content of the site.

In what the organization of the user interface is concerned, we designed a framework that allows for individual features to be developed and plugged into the client independently. For instance, the email client is one feature, the FTP storage is another one, and the IMSP based address book yet another one. We have also implemented a couple of database-backed features like classifieds or PeerNet, whose purpose is to help undergraduates with their academic decisions. We have functionality as diverse as the ability of users to perform queries against files exposed on the Windows network on campus and we can easily add more as they are developed.

3. Physical architecture

In its first implementation the project was built on the Windows 2000 platform[4]. We wrote the middle tier as COM (Component Object Model) components in C++ and the presentation tier as an ASP (Active Server Pages) web application in VBScript. Both tiers were physically hosted on one machine and we made no use of clustering. We refer to this setup as MyDuke.com version 1. It serves over 300,000 hits/day, or, in a more relevant metric, an average of 9,500 logins each day. The machine has a single Pentium III 500 processor with a SCSI hard disk and 512 MB RAM.

As the service gained popularity in the past year, as the load increased to considerable levels and as users started to depend on the website, there was a major need to address reliability and performance. Although providing snappy responses to users' requests under light loads, during peak times the request queue waiting to be served was certainly greater than zero and sometime, albeit rarely, in the low teens. Reliability was also an issue as from time to time we needed to patch the server software or our own software, which required temporary interruptions of the service. This,

---

[4] Although an important decision, it is beyond the scope of this presentation to discuss the choice of platform. There are many case studies from vendors and third parties, which outline the pros and cons of each platform.

coupled with bugs in our own code, which caused the web server software to crash every so often, was causing considerable dissatisfaction among users.

We decided to rewrite the entire project. We arrived at this crucial decision upon seeking a good long-term solution to the above problems and upon evaluating new technologies that came to market after our first implementation. The new implementation, to which I refer as MyDuke.com version 2, features the same design discussed in the previous section, but incorporates more maturity in coding technique and a more in depth understanding of the protocols involved.

The components are now designed for Microsoft's .Net platform and are written in C#. The web application is now an ASP.Net application written in C# and is designed to adapt itself to the type of device that's using it. We were able to build a single core framework to address the needs of clients from mobile phones to desktop browsers. We are currently prototyping desktop and PDA native interfaces (as opposed to browser based interfaces) for multiple platforms.

To address the reliability and performance issues at the operations level as well, we added a second machine and the entire setup consists of a two-node cluster.

4. Usage scenarios

The ultimate goal for this project is to aggregate back end electronic services provided by the University and present them as programmatic and user interfaces. As discussed in a previous section, user interfaces will range from the classic web browser solution to desktop clients for various operating systems as well as trimmed down versions of such applications for PDAs and mobile phones.

Programmatic interfaces are exposed as web services and can be leveraged by other websites either to extend or automate various tasks or perform data exchanges with MyDuke. As an example of this technology, we are currently working with Chronicle Online and Devilnet to integrate both sites' classifieds functionality. The reasoning behind this effort is that Chronicle would like to extend their classifieds offering from newspaper-only to include online features. MyDuke also has a classifieds-like section, which is quite popular due to the growing number of users. We don't believe anyone should have to enlist his or her ad in both sites to get the maximum exposure. Therefore, when queries against the classifieds database will be performed at either site, behind the scenes, each site will query not only the local database, but also the other site's database using a web service.

5. Conclusion

With this project we hope to bring to everyone's attention the immense potential offered by current day technology for growing user productivity and better user experience with electronic resources. This is not an opinion we hold in isolation from the community. The excellent feedback we received over the past year and the incredible growth rate only proves we are going in the right direction. We believe in listening to the community's needs and intelligently employing technology to address those needs. We also hope to set an example of how one can build component based, loosely coupled, distributed applications by designing software in ways that allow programmatic interactions with other software entities.