

# **Service-Based Management System for Non-Profit Organizations**

Joshua Pena, Xang Xiong

5-17-2009

## **Introduction**

This paper serves as documentation for the design and use of the service-based management system for Non-Profit Organizations that was developed by Joshua Pena and Xang Xiong to satisfy the senior project requirement for the Department of Computer Science at California State University, Fresno. The analysis and reasoning for the decisions made concerning the design of the project will be discussed along with detailed descriptions of the core modules. In addition, a brief guide to getting started with the product will be attached to ease the process of beginning testing/use of the project.

## **Requirements Analysis**

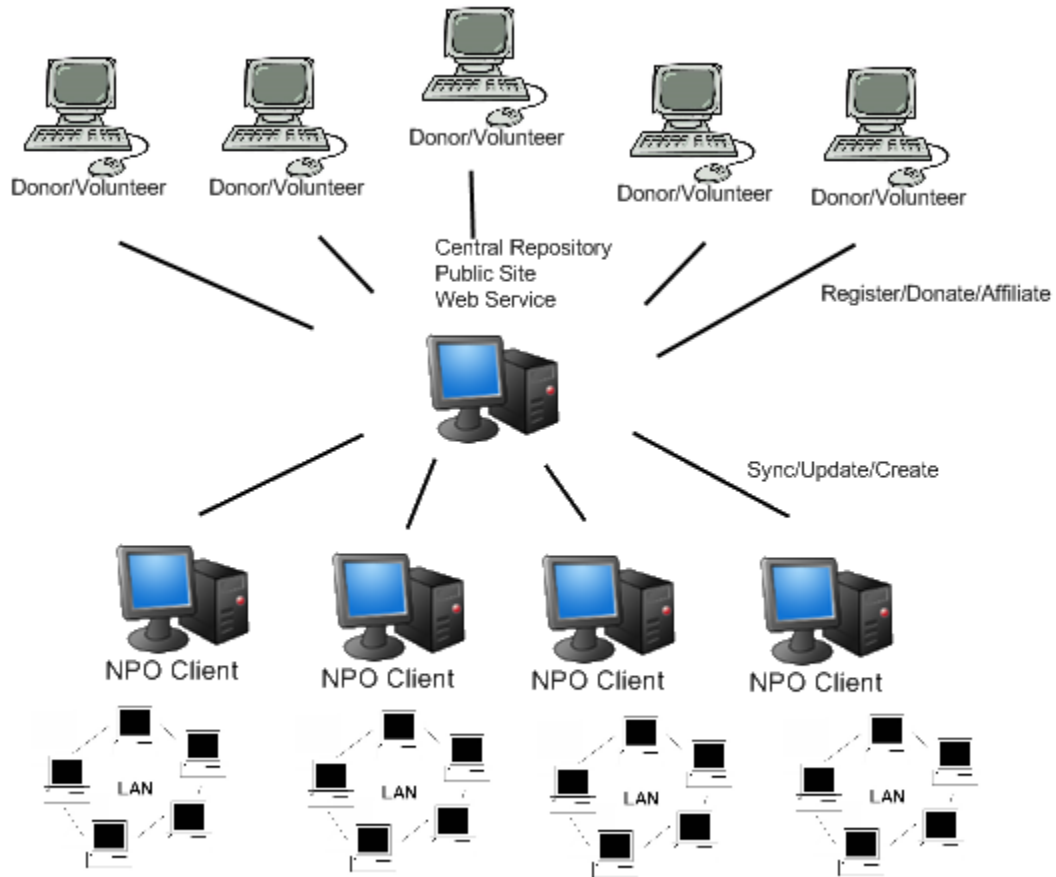
With a vague idea of what was to be done, requirements were drawn up and evaluated in a realistic manner. Since this project was to be finished over the course of one semester by two individuals, the requirements were refined down to something that was achievable. Our initial idea was to have a donation management system using Google Checkout. From there, adding PayPal as an option seemed logical, and was done. However, it is obvious that Non-Profit Organizations don't only rely on donors and donations, but also events and volunteers. This is why the project quickly evolved from a Donation Management System, to a complete solution for Non-Profit Organizations to handle the everyday tasks confronting them.

Knowing a complete solution for Non-Profit Organizations was the goal, a more challenging task was present. Since our application was to be service based, the technologies that were available to use narrowed down significantly. A technology that was flexible, efficient, and well-known was ideal for the situation, which is where the .NET Framework came in. The .NET Framework is capable of creating web-based, forms-based, and service-based applications with ease, along with a plethora of others. Since .NET is built and maintained by Microsoft, there would be certain conflicts with machines that are based on Non-Windows based machines. For this reason, web applications were the best option to accomplish the task at hand. Still, a central, Windows-based server was necessary to host our application.

### General Architecture

Arguably the most important part of the project was to have a solid idea of the architecture. Many ideas were considered, such as having a central, public website that could accomplish all the tasks laid out in the requirements. This could have been done, but it caused a conflict with our initial goal of having a service-based applications. So, we wanted to take advantage of this approach. Knowing that many Non-Profit Organizations like to keep their internal data within the Organization, the ideal application would be one that would be run on a local network. Each of the clients would then register and sync with a public service that would be hosted on a completely separate server, and would serve as a repository for organizations, donors, volunteers, and events. In addition, a public website where donors and volunteers could register and donate or volunteer for events, which is also connected to the same database as the

web service, was to be constructed to allow easy donations to be made. To get a visual idea, please review the following diagram.



1. Client/Service architecture of the non-profit organization management system

From the above diagram, it can be seen that the following items will be needed: a redistributable client-side management package, a central public website, and a Web-Service to coordinate between each module. Using this design model with the .NET Framework, a deeper look into each component will now be described.

### Service Architecture

The architecture of the Web-Service has only one purpose and that is to be an interface between the Client Side Management Package and the Central Public Website's database. Since the entire project was implemented in ASP.NET and C#, the Web-Service is based on Microsoft's WCF based Web-Service. WCF gives much more capabilities and ease of use over the traditional SOAP approach.

The Web-Service architecture offered and exposed methods that would give the Client Side Management Package the ability to input data into the Central Public Database and at the same time retrieve data from the Central Public Database to update the Client Side Management database. Though this seems like a simple task, achieving this one purpose proved to be much more complex than it may seem.

Since the intention of the Web-Service is to expose methods to any registered Client Side Management Package, there arises the issue of any registered Client Side Management Package having access to information from the Central Public Database that doesn't pertain to that certain Client Side Management Package. With just a simple design of the Web-Service architecture, any registered Client Side Management Package can retrieve any information from the database that was exposed through any methods on the Web-Service. Say for example if A is a registered Client Side Management Package and B is also another registered Client Side Management Package; with a simple Web-Service design, anything that A inputs into the Central Public Database can be deleted or altered by B.

Another issue that needed to be addressed for the design of the Web-Service architecture is that any given Client Side Management Package can have more than one Non-Profit Organization. Since one Non-Profit Organization's data should be separate from another Non-

Profit Organization, the Web-Service architecture had to be design to be able to handle such instances. Say for example if A is a Client Side Management Package and A has two register Non-Profit Organization, AB and AC. With a simple design, when A makes a request to the Web-Service endpoint, the Web-Service would not be able to distinguish what request was for what Non-Profit Organization. This will result in data inconsistency among register Non-Profit Organization under a single client side account.

To resolve these two issues, the first step taken in the architectural design was to distinguish any given register Client Side Management Package from another. The approach was to assign a random generated string to a register Client Side Management Package. This random generated string, let's call it RSTRING, will distinguish one register client side package from another and also serves as a authentication step to make sure there's no unpermitted outside access to the Public Central Database by a foreign web script. The process of how this work is that in order for a user who has installed a Client Side Management Package onto his/her system to make request to the Public Central Database, the user must first register for a member's account using the Client Side Management Package. Once the account is register, the user will receive a RSTRING and that will be automatically store into the database by the Client Side Management Package. Now every time the user makes specific calls to the Web-Service, the Client Side Management Package will send the RSTRING with the rest of the request.

The second step was to distinguish any given Non-Profit Organization from another. This approach is similar to the above approach of using an RSTRING, but then we will call it an API\_KEY for this case. The API\_KEY will also be use as an authentication method and a way to distinguish any Non-Profit Organization from another. The process of how this works is that when a Client Side Management Package wishes to register a new Non-Profit Organization, the

client package will make a request to the Web-Service and the Web-Service will register the new Non-Profit Organization. When the registration is done, the Web-Service will return a unique API\_KEY for that Non-Profit Organization back to the Client Side Management Package. The Client Side Management Package will store this API\_KEY and every time the Client Side Management Package needs to make a request to the Web-Service on behalf of this register Non-Profit Organization, it will send the API\_KEY along with the rest of the request. Keep in mind that each individual Non-Profit Organization wishing to interact with the Public Central Database must be register with the Web-Service and assign an API\_KEY.

The overall process of how the Web-Service architecture works is that every Client Side Management Package wishing to use the Web-Service will need to be register and receive an RSTRING. This RSTRING will be use to register multiple Non-Profit Organization under that Client Side Management Package. Each time a Client Side Management Package registers a Non-Profit Organization, that Non-Profit Organization will receive an API\_KEY to be use specifically for that Non-Profit Organization.

### Public Site Architecture

The Public Site architecture is parallel to any website's architecture. The architecture implemented on the Public Site is based on the Model View and Controller (MVC) design pattern. Since most of the methods and classes design to facilitate the Model component of MVC was designed and use in the Web-Service component, most of those methods and classes were reuse in the Public Site. The remaining design of the Public Site architecture fell mainly onto the View and Controller components with some minor addition to the Model component.

The main goals and functions of the Public Site are Member's registrations; Donor and Volunteer's registration; donation; volunteering for events; searching for register Non-Profit Organizations, Members, Volunteers, and donors; and lastly, other miscellaneous functionality that enhances the user's experience using this Public Site.

The Member's registration serves as a way to organize an individual's information and authentication. This process is very straight forward with a normal form submission and database insertion. After the registration, a Member has the ability to log into their account, giving them access to sign up for a Donor or Volunteer's account; if they haven't specify and done so in the Member's registration.

The Donor's registration is a way to specify if a Member wants to become a Donor or not. This will also enable the Member to be recognized as a Donor by any registered Non-Profit Organizations and other Donors. With a register Member account registered as a Donor, the Member can make donations to any Non-Profit Organizations, keep track of the donations they have made, affiliate themselves with any Non-Profit Organizations and if they would like, unregister themselves from being a Donor.

The volunteer's registration is a way to specify if a Member wants to become a Volunteer or not. This will also enable the Member to be recognized as a potential volunteer by any registered Non-Profit Organizations and other Volunteers. With a register Member account registered as a Volunteer, the Member can volunteer for events and keep track of the events they've volunteer for. They will also be able to track any upcoming events, affiliate with Non-Profit Organizations or it they choose to; unregister themselves from being a Volunteer.

Making donation is a very useful functionality that the Public Site offers. The Public Site currently offers two ways for any Non-Profit Organization to receive donations. The first is by PayPal and the second is by Google Checkout. In order for a Non-Profit Organization to receive a donation, they must first have a PayPal or Google Checkout account. Then they must create the donation buttons from either PayPal or Google Checkout or if they would like, they can create both PayPal and Google Checkout buttons. After they have created the buttons from PayPal and Google Checkout, both PayPal and Google Checkout will offer ways to generate the HTML code to be use to receive the donations. What the Non-Profit Organization would have to do is take this HTML code and save it into the public central database using the Client Side Management Package. After the HTML code for the button or buttons has been saved, whenever a Donor browse to that particular Non-Profit organization, the Donor will see those donation buttons. If a Donor clicks on the donation button, the Donor will be able to donate to that Non-Profit Organization that had created that donation button. Once the Donor has chosen to donate, they will fill out the short form specifying the donation amount. Once they submit the form, the information will be inserted into the Public Central Database and they will be redirected to PayPal or Google Checkout; depending on the button they've clicked. The information regarding what Non-Profit Organization the donation is for and how much was donated will also be forward to PayPal or Google Checkout too. This approach to donation management and processing offers a simple way to interface with PayPal and Google Checkout and also giving a powerful way handle donations on any particular site.

Volunteering for events is also another powerful functionality that the Public Site offers. Any Member registered as a Volunteer can search for any event and volunteer for that event. Once they've Volunteer for an event, the site will save that information into the database and the

Client Side Management Package in charge of the Non-Profit Organization that offered that event will be able to download the volunteers that have signed up for the event. This gives the Non-Profit Organization a wide array of audience on the web and also the ability to process and save the data locally onto their person computer or local network.

As mention above, the site also offer ways for the register member to search for Non-Profit Organizations, Members, Volunteers, and Events. Search capabilities are always important on web applications to give the user the ability to find just what they need. This simple search is straight forward with a single search field. Using the given keywords, the small search engine will retrieve any match in the database.

Overall the site offers useful functionalities intended to enhance and improve the register member's ability to interact with other registered member and Non-Profit Organizations. One of the special components built into the Public Site is a profile page for the register Member. This page gives the member the entire layout of their activities and information. With a profile page, registered Member will have a place that is dedicated to them and also can be use as their little workspace.

### Client Architecture

The chief idea behind the Client Application is to provide a safe, easy, and compatible solution that can be used by the many members of a Non-Profit Organization. Instead of having to go through the trouble of having one application per computer within the Organization, it was decided that a web application that can be hosted on a local machine and can be accessed easily by anyone within the network would be a much better alternative. There will only be initial setup

phase to get the site up and running, which simplifies the process a bit. Also, using a web application eliminates the conflict of internal users running different operating systems. The only Windows-based system needed is the one hosting the site, since it is an ASP.NET application. A SQL server database is also required, which is available in an express edition from Microsoft free of charge.

The software architecture of the client application took advantage of a popular design pattern in the .NET framework. With any web application, one of the most important areas of concern is data access. A safe, efficient means to access data from a database and manipulate it on the user-end is something that should come easily. The .NET framework provides a series of libraries that allow the temporary storage of data from a database into memory. From these datasets, data can be easily bound to ASP.NET modules and displayed on a web page with ease. However, since these datasets depend on their data source, they are dynamic in nature. Being dynamic in nature, Microsoft's intellisense, a valuable component of Visual Studio, cannot be taken full advantage of. This is where the iterator design pattern came into play. This design pattern provides a means to access elements of an aggregate object sequentially by exposing a simple iterator based on the .NET ArrayList class. Using the ArrayList, a custom collection can be created which implements the IEnumerator interface. Using this idea, objects are created to represent donors, volunteers, donations, and other elements of the project and are thrown into a collection of other objects that are related. Then from these collections, the .NET framework allows the direct binding to ASP.NET modules, just like they were one of the .NET data libraries. Because of this, we can now create meaningful collections of objects and take advantage of Visual Studio's intellisense to the fullest.

It should be noted that using this collection based approach is slower than directly binding to the existing datasets. One of the reasons for this is because of the extra layer that is required to convert the datasets pulled directly from the database into its representative object. Each of the tuples in the dataset must be iterated and converted to its object, then added to its respective collection. Because of this, many feel this to be unnecessary and choose not to implement an application in this way. Binding to collections also presents other problems such as implementing paging and sorting of a collection, which the .NET framework accomplishes easily with existing datasets. In order to do this with custom collections, other interfaces must be implemented, which is a small amount of extra coding. Since this was the first time either of us have experimented with implementing an application in this way, it is difficult for us to provide a constructive opinion on which way is more preferred. Both have their advantages and disadvantages, but both methods provide a nice way of binding data.

As a brief conclusion to the client side package, here are some of the features that it provides. This package provides a means to manage multiple organizations, not just one. So if someone is an administrator to more than one Non-Profit organization, using the same installation package and database, another organization can be registered and managed. This eliminates any problem of having to have a side-by-side configuration if managing multiple organizations. This package provides a means to enter, update, and manage donor, volunteer, donation, organization, and event information. Some of this information can be posted to the public repository for easy access to others, and events can also be posted to a Zvents account. A basic advanced search through donors or volunteers is available, along with a means to generate data and export it to a Microsoft Excel spreadsheet. Moderator accounts can also be created, which have access to all but the most important areas of the applications. Finally, and most

importantly, organizations can use the donation buttons that are easily generated by their PayPal and Google Checkout accounts and submit them to our repository. Doing so allows Donors who have registered through our public website to donate to the organization with a simple click of a button.

### Areas of Improvement

With any project that has been constructed for non-commercial use, there are areas that can be extended or improved. Much like the layout of this paper, each module will be looked at and some of the areas that can use some more attention will be brought to light. It should be noted that most of these areas were left like this on purpose for a simple lack of time. Given our resources and the amount of time available to complete the project, not everything could be perfected.

The Web-Service will never be fully complete. There are always rooms to grow and ways to extend the functionalities the Web-Service currently have. There are better methods to handle the Client Side Management Package authentication and managing Non-Profit organizations information. As technology grows, there will probably be improvement on WCF, which in terms will give better ways for the Web-Service to be implemented.

The actual design and methods made available by the Web-Service can also be improved, meaning in terms of efficiency or portability. Say for example there might be a better way to do database connection and query other than Store Procedure, which this current system is using. There might also be other ways to return data from the Web-Service, like in a Json file for example, which can make the Web-Service application more portable across other applications

that utilizes Json. There are more functionalities that could be added to the Web-Service. Say for example volunteer service hour tracking, announcements for Non-Profit organizations, and maybe even advertisement for Non-Profit organizations, which they can use to advertise upcoming events, dinners, or just fundraising opportunities.

The Public Site can also use more improvement on both the user interface and functionality. On any web application the user interface is as important as the functionality. The look and feel of the site will help enhance the user's experience by making navigation much simpler. Though the current design of the Public Site is sufficient for a school project, it will need a huge improvement for a commercial release. There is more style sheet that needs to be applied to make the interface a much more professional look and feel. The data being display will need to be better organized so that it will be easier for the average user to understand. The functionalities would need to be extended to give the user much more capabilities that are unmatched elsewhere. For a commercial release, the search engine would need to be improve as for the database will grow to a huge size and a simple search algorithm will take minutes to search, which in the commercial realm is unacceptable. Extra functionalities like custom email messaging support, RSS feeds, and advertisements should be added to give the user more control over the information on the Public Site.

There are several areas of the client package that were not fully implemented for lack of time. The first of which is the pledge section, which was part of the original requirements. We wanted to provide a means of keeping track not just of donations, but also pledges that were received. Since this was viewed as one of lesser items of importance, it was something that was left till the end to finish. The underlying logic and design is present both in the database and at the object level, but various interfaces still need to be constructed. The other section that was not

finished was the messaging system. Initially, we wanted to have a way for the moderators/administrators to send periodic e-mails to its donors and volunteers, since their e-mail addresses are being stored. Achieving something like this would require a different kind of application, ideally a windows service, which the .NET framework does provide. However, this in of itself requires a fair amount of coding.

### Conclusion

The application that we have built can be used to manage the everyday tasks that Non-Profit organizations face. Using the .NET framework along with a SOA approach, a dynamic multi-tiered application was built that gave us a good look into what it takes to build a software product from scratch into something that can be used for practical means. A client-side package, WCF based web service, and public web site were built to allow organizations and its affiliates to make their basic business process easier. It certainly was a challenging task, especially when responsible for each phase of the software design process, but it also provides a sense of appreciation and satisfaction.

Overall this project as a whole gave us the opportunity to exercise many aspect of Computer Science, from core Software Engineering to database design and new Web-Service technology like Microsoft WCF Web-Services. This project not only serves as a bundle of software that can be used to make day to day task for Non-Profit Organizations much more efficient, but also a platform for a commercial or open source project that one day can the communities around the globe can benefit from.