

AddonsForLife Project

So, I'm enjoying my summer vacation, playing World of Warcraft (WoW). But this is not an ordinary game: This game allows users to install and use "addons," which are essentially small applications used to extend the functionality of UI elements in the game. They don't interact with the 3D world directly (to prevent cheating and avoid bots that play the game automatically), however they do offer very rich functionality that extends the game's interface in very interesting ways.

So, I'm sitting here with a few widgets on my desktop for monitoring things like free space on the drive, showing my next appointment on my calendar, my task list, and the current weather. It's then that it struck me – these widgets are mostly passive. Their focus is very much on giving the user information, rather than acting on it. Sure, the task list allows me to mark tasks as finished, but I'd need to open up the application that publishes the task list in order to modify tasks.

However, in a game like WoW, most addons have the ability to both display and manipulate data, and are very task oriented. Most addons exist for the purpose of helping a player with a task or guiding the player towards a goal. What would happen if we took those concepts out of the virtual world of a game, and applied them to the real, physical world? Applications that guide people towards reaching real world goals could easily translate to real productivity gains and real economic benefits.

The basic platform of WoW's addon system is the scripting language Lua. Lua is a scripting language that is gaining popularity, especially in the gaming industry, for the following reasons:

- The basic concepts are simple, and there only eight basic data types.
- The syntax is forgiving. For example, it allows but does not require a semicolon at the end of a statement.
- Functions are a basic data type, and the language itself is loosely typed (variables do not have types – only values do). This gives it the power to make addons that are easy to extend in new and interesting ways.
- The most important data structure in Lua is the table. Tables in Lua act much like associative arrays, being able to be indexed by most of the data types, and being able to hold any of the data types (except nil).
- Tables make Lua easy to understand, and since most indexing is implemented by hash tables, they're also very fast, making Lua one of the fastest running interpreted languages.
- It's easy to secure. There are only a small number of base libraries, and the ones that could pose a security risk can easily be excluded from being used. Being an open source (MIT license) project, it's also easy to modify the Lua interpreter itself.

Thus, Lua makes an ideal base for creating a development environment that is easy to use and secure. This should work well even when not dealing with games.

As a scripting language, Lua needs a base language. C# provides an ideal base language for Lua: It provides access to WPF, a library well suited to custom UIs, but still has basic widgets that can be used. C# is a garbage collected language and relatively safe, so we can focus on the scripting rather than

memory management. Lastly, C# can be used in the future to create Internet based widgets with similar functionality via Silverlight.

The following base functionality needs to be implemented for the concept:

- Basic APIs between Lua and C# need to be created. There are already libraries available to simplify this process, so it should certainly be possible to use C# with Lua.
 - Three major APIs need to be created:
 - An API to allow Lua scripts to create and manipulate WPF widgets. This is central to the Lua addon concept.
 - An API to handle events between Lua and C#, and between Lua scripts. This is desired for Lua scripts to be able to handle real time events.
 - An API to allow Lua scripts to save state to the drive. This is not necessarily a file system – this could be done by letting addons specify variables to save to the drive, which can be done on demand or when the addon is shut down by the user. Saving a large amount of data can be done by selecting a table as a variable to save to the drive.
 - Saving state can be dropped if there isn't enough time.
- The C# code needs to be able to run Lua scripts or Lua functions inside Lua scripts.
 - The loading process can be straightforward: A folder on the drive can be set as the addon folder, with each addon having its own folder in the addon folder. The loader can simply iterate through the path loading Lua scripts as it encounters them.
 - Another file can be used to store metadata for the application (version number, files to be loaded, required and optional libraries)
- At least one sample Lua application needs to be made.

Possible applications implemented using Lua include:

- A basic contact list application
 - Manages contact information
 - Very visible to other applications, as contact information should be shared.
 - Also stores groups each contact belongs to.
 - The idea of individuals belonging to groups is stronger in a game than in a standard application: Groups are often called “guilds” and the groups themselves have a lot of power and impact in many games.
 - Since this is used with other parts of the project, it would be high priority.
- A basic calendar application
 - Not just a calendar for the individual – also has calendars for the groups.
 - Medium priority.
- A basic messaging application
 - Many instant messaging applications have individual chat and chat “rooms.”
 - Chat rooms have not been very popular from what I've seen in standard IM usage. They were popular in the early days of the Internet, but have been largely replaced by social applications.

- However, in games “guild” chats have been very popular.
 - Therefore, I would make group chats a part of the standard experience.
 - Can be cut out of this project if isn’t enough time.
 - *Requires a server of some type.
- A basic “email” application
 - Send directly to contact(s) by selecting them from a list.
 - Try to avoid the extra step of using a different identifier, such as an email address. The user just selects the contacts to send to, the application takes care of routing the messages.
 - Contacts can also include groups.
 - Low priority.
 - *Requires a server of some type.
- A basic TODO application
 - Would be handy for most people, although low priority for this project.
- A “reward” application
 - I’ve got some very interesting ideas for this – essentially a three section application.
 - This is where the groups in the contact list could be very powerful, as the group can decide on goals and awards for the individuals.
 - The first section is a point based section for one time goals that anybody in a group can accomplish. Individuals gather points towards a goal, eventually reaching 100%. Could be useful for training and education.
 - The second section is a system without points, for notable accomplishments.
 - The third section is a system for repeatable goals, which award a set number of points, but without any cap on the number of points that can be accumulated. This could also be used as a currency system for buying virtual or physical items.
 - Medium priority for this project, although I’d like to see at least part of it functioning.
- An automotive application
 - For keeping track of fuel usage, service dates, trip logs.
 - Something I thought of, but am unlikely to implement for the project unless there is time.

*Although most of the functionality of most applications can be done with information stored offline, the chat and email applications would likely require a more complete server setup. In order to keep the scope of the project within a feasible time frame, I’m unlikely to implement them for the semester project.