Bringing Order to Chaos

An attempt to impart a design on UOX3

Summary

The purpose of this document isn't to delineate a be all and end all solution but to provoke discussion and ideas in an attempt to bring out greater influence by others and to determine a course of action for future work. Ideas in this document may be discarded entirely or embraced wholeheartedly, or manipulated to suit a purpose.

It will detail some ideas and thoughts that may be useful for the future development of UOX3. This is not an exhaustive list of ideas, nor should they be taken as gospel. Please note that just because it is being discussed, does **not** mean that it will end up being used. It is purely designed for conjecture and ideas, a way of stimulating developers and users alike, with the ability to provide feedback and input. Not all parts will be applicable to developers, not all parts will be applicable to users. It is just a mechanism by which people can ponder and debate the merits of systems and potential future implementations.

Categories

Details

Source Code Restructuring

Status	Concept
References	

The nature of UOX3 has changed over the years, away from a single source file to many source files. There now exist 161 different files in the project, all sitting together in a single directory. This is not the most efficient mechanism for storage of files, even though the VC workspaces are separated into folders. What is proposed is a mechanism for segregating these files into subfolders based on their nature. A naïve solution is to separate CPP and H files into their own directories, but this would still leave a large amount in each folder. While it would require a restructuring of the headers to a small degree, this would improve maintenance to some degree. What a valid structure would be, I don't know, though it could be split into major subsystems. Something like this may be appropriate:

Folder	**Subfolder**	**Description**
source		Main source folder
	combat	Combat related routines here
	objects	Major object classes here. Cchar, Citem, CmultiObj immediately come to mind
	network	Network related – packet classes, networking subsystems
	dfns	Parsing and loading of dfn systems. Script, ScriptSection, cServerDefinitions would be here
	script	JS engine files would go here
	magic	Magic related files here
	command	Command related files would go here
	Races	Race related files would go here
	Region	Region related files would go here. This includes both town regions and map regions
	Fileio	File related files here. This includes all routines to read/write from files, including the MULs
	Guild	Guild files would go here
	Misc	All other files would end up here
	Account	Account related files would end up here

One thing that has not been mentioned so far is the use of third party libraries. There is the idea of reinventing the wheel with a lot of the work that is done in UOX, including parsing of strings, directory traversal and other means. Some of these things could be offloaded into well-tested third party libraries, which would also remove some of the platform specific issues that we seem to stumble over. One such library that could prove useful is the Boost library (http://www.boost.org). It involves a series of libraries, upwards of 50, which are well tested and portable. Amongst other things, we could use it to simplify our tokenisation of strings, date time systems, directory traversal systems and provide safer versions of functions such as sprintf.

The biggest issue with using third party libraries is the fact that it makes life more complex when it comes to compilation and development. There is also the issue of licensing but given the GPL nature of UOX3, few licenses should prove to be a problem.

Improved Class Hierarchy

Status	Concept
References	

The current class hierarchy is a haphazard collection, with some routines thrown into a class purely for the sake of them being there. There is also plenty of freedom left for improving the class hierarchy, where it only travels at most 3 deep in packet classes, and 2 deep otherwise. The hierarchy could be heavily improved though it would require a large investment in time and debugging. Most notably, the character class could be subclassed into an NPC and PC class, and the multi class could be subclassed further for housing, providing specific features. However, some of these could also be manipulated with the idea of Generic Properties mentioned further along.

The biggest link of this would be to the object factories and ease of maintenance. A large part of the SERIAL spectrum is completely unused, with characters and items only taking up a small part of that. We could reserve other parts of the spectrum for other classes of objects such as regions, towns, sockets and so forth, but this would rely on an uber base class that provides a serial and an objType for each.

A possible, non-exhaustive hierarchy could be something along the lines of

Uber base				
	CBaseObject			
		Citem		
			Cmulti	
				Chouse
		Cchar		
			CPC	
			CNPC	
	CSocket			
	CRegion			
	CPBuffer			
		Outgoing packets		
	CPInputBuffer			
		Incoming packets		
	Cguild			

Object Factories

Status	Concept
References	

This ties heavily into the previous section, detailing an improved class hierarchy. As it stands, we have separate routines and handlers for creation of objects. By use of an object factory, we can centralise the routines required for creation of objects. For instance, the cItems class is responsible for creation of items and multis while cCharacterStuff is responsible for creation of characters. By creating a single routine, we can make one place responsible for the creation and destruction of all our in game objects. In this instance, it would be a single routine, which is passed an enum detailing the object type we wish to create and returning a pointer of the appropriate type. Another routine would exist that would be responsible for the destruction of objects.

As an aside, we would also put the lookup routines for objects in this space as well. Because of this, the object factory would be responsible for the creation, deletion and lookup of our object types. This relies heavily on the idea of a refactored class hierarchy, if the object factory is to be responsible for the creation of objects like sockets and regions as well.

Because of this, we're also likely to remove the items[] and chars[] array from a global scope, making the object factory the recipient of these containers. This allows us to hide away the way the pointers are allocated,

Generic Properties

Status	Concept
References	[10]

Recreation of Guild Systems

Status	Concept
References	[1]

Recreation of Town Systems

Status	Concept
References	[5]

Player Housing

Status	Concept
References	[2]

Map Handling

Status	Concept
References	

Versioning

Status	Concept
References	

Versioning in UOX3 is rather ad hoc, advancing in a fashion that depends entirely on the coder. To my knowledge, there is no set heuristic or algorithm for the incrementing of a version number. As it currently stands, UOX3 is at version 0.97.06 Build 1m. There are 4, arguably 5, parts to that version number. The 0 meaning the major version number, the 97 indicating major subversion, the 06 indicating the minor version and then 1, possibly 2 parts: major build and minor build.

Assuming we see it as 5 parts, then there's a lot of possibilities for increment. But there are no hard and fast rules for incrementing this, except to say that version 1.0 will be the developer's ideal version, and as the code gets closer, so does the version number. A possible list of rules for incrementing may prove useful to help clarify the versioning system. One possible heuristic (and remember, it's only a suggestion) is:

Version Part	**Rule**
Major	Revolutionary leap in behaviour, practice or functionality
Major Subversion	Significant updates to major subsystems
Minor Subversion	Major updates to single subsystem, or collection of relatively updates to group of subsystems
Major Build	For every commit that's worth going into the cvs. Any bug fix would do it. Upon completion of a bug fix, the minor build is reset to 0 (or a) and this is incremented.
Minor Build	Every single build attempt increments this. This number may increase more than once between commits. So if fixing a bug takes 4 compiles and tests, then this would go up by 4.

Databases

Status	Concept
References	

Grouping

Status	Concept
References	

Game Balance

Status	Concept
References	[7], [8], [9]

Testing

Status	Concept
References	[6]

Maintenance

Status	Concept
References	

Reputation

Status	Concept
References	[4]

Game Rewards

Status	Concept
References	[3]

Combat Systems

Status	Concept
References	[7], [8], [9]

Chat Systems

Status	Concept
References	[1]

References

[1] Olsen, J. (2003). Designing a Flexible Guild Creation and Management Command Set. *Massively Multiplayer Game Development. pp. 442 – 453*

[2] Sage, P. (2003). Player Housing – My House Is Your House. *Massively Multiplayer Game Development. pp. 421-426*

[3] Pizer, P. (2003). Social Game Systems: Cultivating Player Socialization and Providing Alternate Routes to Game Rewards. *Massively Multiplayer Game Development. pp. 427 - 441*

[4] Brockington, M. (2003). Building a Reputation System: Hatred, Forgiveness and Surrender in *Neverwinter Nights*. *Massively Multiplayer Game Development. pp. 454 - 463*

[5] Rogers, A. (2003). City-State Governments – Their Roles in Online Communities. *Massively Multiplayer Game Development. pp. 464 – 476*

[6] Walker, M. (2003). Unit Testing for Massively Multiplayer Games. *Massively Multiplayer Game Development. pp. 137 - 150*

[7] Sanderson, D. (2003). Everybody Needs Somebody: Practical Advice for Encouraging Cooperative Play in Online Virtual Worlds. *Massively Multiplayer Game Development. pp. 20 - 29*

[8] Hanson, B. (2003). Game Balance for Massively Multiplayer Games. *Massively Multiplayer Game Development. pp. 30 - 37*

[9] Olsen, J. (2003). Game Balance and AI Using Payoff Matrices. *Massively Multiplayer Game Development. pp. 38 - 48*

[10] Cafrelli, C. (2001). A Property Class for Generic C++ Member Access. *Game Programming Gems 2. pp. 46 - 50*

Contributors

Maarc	darkangelab@users.sourceforge.net

Changelog

Version	Date	Change	Editor
1.0	13 Aug. 03	Initial Revision	Maarc