



ai design studio

unga unveiled

Version 1.0 - 10/2010

by ai design studio (<http://ai-designstudio.net>)

Contents license as indicated in the page footer

<http://ai-designstudio.net>



Introduction

It is hard to explain to a “non initiated” what unga is and why I am wasting my time with this project having more interesting things to do, for example... collaborating with opensim code base or improving my Autocad importer for Second Life/Opensim.

Well, I will try to explain why I think unga could improve the Opensim ecosystem and, overall, the “serious” or “business” opensim applications.



Definitions: Opensimulator

The starting point is to know what Opensim¹ is: a serie of “servers” or programs offering virtual world services (in general, whatever they are), currently compatible with Second Life² viewers.

The “services” provided for opensim are similar to a web hosting but oriented to 3D, that is, into one opensim server you will find “content”, in this case buildings, objects, programs, etc. and you can interact with them having a 3D view. You are able to navigate for the content, too, for example you can walk for the reproduction of a cathedral, a spaceship or a desert. These “servers” or “spaces” are called “regions”.

The 3D navigation is made with an “avatar”, the representation of your presence in the world. Other users in the same server can see the representation of your avatar as well as interact with it. Also, It is posible to customize your avatar, changing shape, textures, adding objects, etc.

You can jump from one opensim server to another to get different content. The jump can be made using “teleporting” (providing an address to the browser) or visually, doing what is called “region crossing”. Your “avatar” can walk from a server to another nearby server. Even more, you can have a peek of the content of the neighbour server.

This is an oversimplification, of course, and I'm not taking into account some other services pluggable in Opensim like voice chat and video reproduction.

1 <http://opensimulator.org/>

2 <http://secondlife.com/>

Definitions: Grid Services

If you have only one region (or server) on your personal computer you can use the *standalone* installation of opensim. It works very well. Your data is present only in your computer and you can control some “regions”, create stuff, allow other people to enter in your world, etc.

But think in the future. Imagine that your creations became famous and everybody wants to see it. Imagine that you need more “terrain”, or you want to create more than 4 or 5 spaces. Your personal computer will be very limited (memory, disk space, cpu use...) and your computer will have to control the user management, the content (asset) management, the inventories for each user, the communications between regions, etc... Sooner or later you will need a way to “scale”³.

Opensim allows to scale creating “grids” of regions. That is: you have some servers to control users, content, inventories, etc. and other servers to send the content to the users (interact with them), the regions servers. The first ones are common for each region, that is, for example, that you have a central user server which will authenticate the servers for every region.

This way of work is called the Grid Mode and, this is my personal definition, I call this “central”, or common servers, Grid Servers (or services, depends on the day :-P).

The services use to be:

- *User management*: controlling login, user profile, user relationships...
- *Grid* or region management: controls communication between regions.
- *Asset* management: serve content for each region server, for example textures or shapes).
- *Inventory*: stores the inventory of objects for each user.

There are money services or even group services available, but are not interesting for the core in this moment.

The Opensim code base has a basic implementation of these Grid Servers. It is called ROBUST⁴. The interesting thing, in this case, is the way ROBUST communicate with opensim servers: it uses http⁵, a standard protocol.

And now, here comes unga...

3 <http://en.wikipedia.org/wiki/Scalability>

4 <http://opensimulator.org/wiki/ROBUST>

5 http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

<http://ai-designstudio.net>



Definitions: unga

unga started as a challenge, an experiment, as a way to check if I can do it something useful.

Well, I did it :-P

I didn't like ROBUST (and the old UGAIM servers) too much. Don't misunderstand me, ROBUST is a valid solution, it works and does what is has to do, but I was thinking about scalability, among other things.

Maybe with 10 or even 50 regions there won't be any problem. Maybe the response (specially from the always needed asset server) will be correct. Sometimes it could be slow, but it can be fixed using more memory, more CPU or more bandwidth... These scares me, they mean money, much money

Imagine 100 or 1000 region servers...

Another thing that scares me is that grid servers run over .NET. Not only this, they are http servers (they don't do anything more than serving http), but BASIC http servers. You haven't got the power of an Apache or IIS server, for example for farming (horizontal scalability⁶) or event for security (is easier to prevent a DDoS attack banning IP's from the htaccess file of Apache, for example).

I know, Opensim region server run over .NET, but the region servers are specialized servers, they need to be a dedicated server, at least nowadays (it is not easy to create UDP⁷ sockets from a web server).

So, why not using a well tested http server for serving http content?

That is unga: Grid services published on a web server.

But there is still much more.

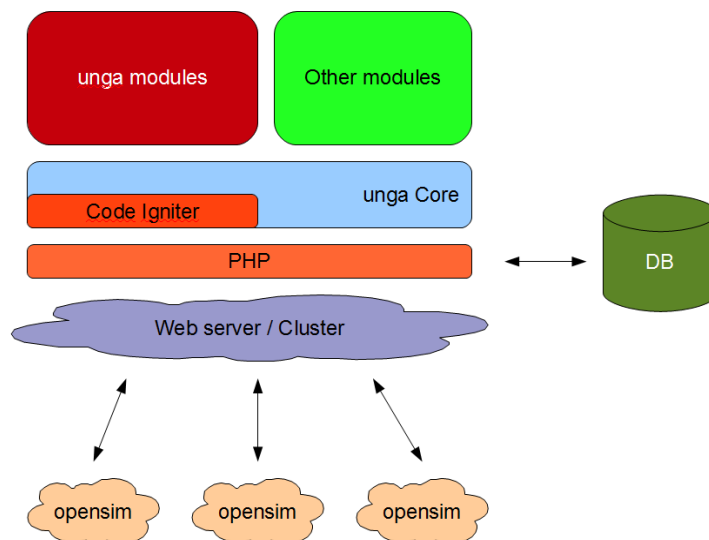


Figure 1: Overview of unga architecture

6 http://en.wikipedia.org/wiki/Scalability#Scale_horizontally_.28scale_out.29

7 http://en.wikipedia.org/wiki/User_Datagram_Protocol

Architecture

Once the decision had been taken I had to make it real.

It should be quick, it should be secure, it should be scalable, it should be well thought. The basic unga architecture is based, or running over, the Code Igniter⁸ PHP framework.

PHP? This is for losers. Well, I'm sure you use dozens of services programmed in PHP daily. From Facebook or Twitter to the online newspapers you read every day, and they work well, really well.

Of course I could have used another more "cool" language like Python, Ruby or Lua, but this was my personal decision. I've programmed in PHP for several years, I know the good and the ugly... and PHP has thousands of well tested libraries and hundreds of frameworks to help me.

Oh, this was another personal decision: I didn't want to reinvent the wheel; if it works, use it.

(I have plans to migrate parts of unga to Python... you know, another personal challenge).

If I want to scale, I can buy cheap equipment and scale web server horizontally (more cheap servers, instead of a huge hyper-expensive server). If I want security, I can activate well-tested security modules from the web server and it will be only a question of server administration.

The first operative version of unga, for opensim version 0.6.X was released on April 2010⁹. There were some attempts before, using another frameworks (CakePHP, too slow) but I consider this the first operative version.

I made a mistake: I've programmed unga only for ONE opensim version... I didn't realize that the Connectors¹⁰ exists (see below) so this version was very limited. The good part is that it was compatible with ROBUST services, that is, it will work just by changing the host addresses, no more configuration needed.

Anyway, with the existence of connectors I decided to rewrite some parts of unga core. Not only the core, all the modules, to adapt it to the new core... and, as I enjoy complicating my life, to rewrite the modules again and to add new features.

8 <http://codeigniter.com/>

9 <http://sourceforge.net/projects/unga/>

10 http://opensimulator.org/wiki/OpenSim_Services_and_Service_Connectors

<http://ai-designstudio.net>



Opensim Connectors saved my life

The opensim connectors are the way the Opensim architecture has to allow to define the way Opensim region server gets the data it needs from the “support servers”.

Opensim defines interfaces for functions that will be called when the region server needs data. For example GetAsset, StoreAsset, Login, etc.

Opensim connectors allow to join these interfaces with pluggable modules that will do... anything. They can do a “local” call to a database server or a call using “whatever protocol” to an external server. For example, a call to GetAsset could connect to a local database using ADO.NET or to connect with a http server in the address http://my_asset_server and get the XML representation of the Asset (more or less is the way ROBUST asset server works...).

So with the connector I have a new open horizon : I can use whatever I want. I'm not limited to the protocols and data defined by ROBUST (xml-rpc former and now some strange REST, and XML for data).

And then there was REST

REST¹¹ is not a framework, it is a way to define an architecture to access web services.

In its ideal form it should allow to do CRUD operations over data, using the http defined methods¹² GET, POST, PUT and DELETE. But the world is not always ideal.

Unga tries to use “the CRUD¹³ way of life”. In the core system, a base class to automatically detects the operation and redirects to the appropriated module and function is defined.

In the core system, a base class is defined. It automatically detects the operation and redirects to the appropriated module and function

For example, if the http operation is GET unga will redirect to a function called “retrieve” and will send the parameters defined in the url. If you use a POST, the redirection will be to “create” and the parameters are recovered from the post data of the request.

11 http://en.wikipedia.org/wiki/Representational_State_Transfer

12 http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

13 http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

<http://ai-designstudio.net>



The "syntax" unga follows for REST is something like this:

http://HOST_AND_PATH/MODULE/ENTITY/filter_field/filter_data/filter_field_X/filter_data_X/field_to_recover.

HOST_AND_PATH indicates where is unga published and Code Igniter is the one who treats this data.

MODULE defines wich service are we calling. For example inventory, users, grid...

ENTITY is the object we are working with. For example, inside inventory we can work with folders or items, in the grid, with regions and presences, and so on...

filter_field and **filter_data** are filters to locate the data you want to work with. For example, if you want all the folders with type=2 and with permission type=5 the filter should be something like this:

HOST/INVENTORY/folders/type/2/permission/5

Of course operators for >, <, like and so on are allowed using a special form for filter_data.

field_to_recover means that you don't want to recover all the data present in an object, for example, you could want only the binary data of an asset with id 25. Then the url will be something like:

HOST/WAREHOUSE/assets/id/25/data.

Resuming the filter and field part: for unga a pair of values means a filter, one value alone means a field to recover.

And of course there was the http operation:

GET, will need a full url indicating what you want to get (for example asset/id/25).

POST won't take the filter into account and will need POST data for creating a new object (for example a new folder).

PUT will modify the entity defined in the filter (for example asset/id/25) with the POST data sent in the request (for example a change in the asset description).

DELETE will erase the data specified in the filter, no less and no more.

As I said all these operations and redirections are controlled into unga core. The only thing needed is to define which type of data (asset, folder...) are we working.

Even more, unless for special or delicated operations the unga core has defined both the controller (url face) and the model (database operation) for the CRUD operations. There is no need for creating new methods, only to define the name, the table (or storage entity for operation) and the internal object for working.

Data formats: whatever

I like freedom, I like it very much. And I understand that if I don't feel comfortable with XML¹⁴ and want to use another format, some other people like XML.

In the PHP world is a (bad) practice to work directly with the received data instead with internal objects. In the old days, where the object creation is expensive for the interpreter could be understandable, but nowadays...

So, as in the .Net or Java world, the best way to work is using objects inside the application and serializing to "whatever format" when communicating with external services. This will allow us to work with, for example, XML, JSON, strings, apples, stone-writing data, etc.

unga provides 2 serializers: for XML and for JSON¹⁵. The first one is more descriptive but heavier, the last one is less descriptive but lightweight. You can use whatever you want or need only specifying the "Accept" and "Content-Type" headers¹⁶ in the request. It is even possible to send XML data and receive JSON...

How do we define the object we want to use in a service? Easy, using annotations¹⁷ (attributes in the .NET world). Each controller should define the base or default object it wants to work with. For example the Assets service will work, by default, with AssetBase objects.

These PHP objects will be serialized to and from XML/JSON in a way similar to .NET.

In unga the library used for allowing PHP annotations (not native in the language) is Addendum¹⁸.

You are free to implement whatever serializer you want.

Objects, objects, objects

The data objects, that is, the objects used internally by unga to work, are defined using annotations too. It will define the type of each field or (this is a future improvement) the name of the field in the database.

Each controller will work, by default, with only one type of entity (for example asset, folder, item...) and these entity has a 1-1 relation with a database table. So we know now that when we launch a DELETE to the entity Asset we are deleting a register in the Assets table.

Of course unga is flexible enough to work with table joins or only in some fields. If anyone does not like the default implementation of the CRUD operations, just override them. These serves for controllers and for models.

Do you remember the "field_to_recover" component of a REST url? You only have to implement a method in the model to deal with it. For example, in the asset server is no field is requested by default will return an AssetBase object. If you request the field "data" in a "retrieve" (GET) operation unga will find for a method called "retrieve_data" in the the assets_model and will return whatever type of data is defined there.

Cool, isn't it?

14 <http://en.wikipedia.org/wiki/XML>

15 <http://en.wikipedia.org/wiki/JSON>

16 http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

17 http://en.wikipedia.org/wiki/Java_annotation

18 <http://code.google.com/p/addendum/>

Putting all together

So, we had connectors defined by Opensim Core, we have services published as REST, defined by unga... and we have to put them together.

Quite simple, just create a new connector that implements the interfaces Opensim needs and change the names in the configuration file.

[Actually I am creating the connectors... it is an ungrateful, repetitive and hard task].

But wait, there is more

unga is intended to provide services both for small grids as for huge-thousand-regions grids, so there are things I had to design (or I am designing right now) to improve scalability or best-administration-practices to follow.

Installation

The unga installation is of the type "*next->next->next*".

There is a graphical web assistant and the only thing unga needs to know is where to store data and who is the master administrator user. The installer will create all the basic unga structure. You can have unga working in less than one minute (if you can type quickly :-P).

The next plan in the installation process is implementing telepathy into the core base and a big red button for installing "a la Apple style". It will require some weeks.

Integrated user management

unga provides, by default, integrated user management. It has the necessary methods to create, modify and delete users and to authenticate and authorize them. This allows unga not only to be a complete Opensim full server but also to use the same user management for backend purposes (who can access to backend¹⁹?)

Also, unga provides a roles and permission-assigned-to-roles system. That is, each user has a role, and this role will allow the user to do some tasks. For example: a "user role" and a "super user role" can be defined. The first one is denied terraforming permissions and the second one is granted them.

Unga has a profile management system too. That means that there are groups of properties attached to each user. For example we could need opensim properties (position, creation date...), money properties (credit card, bank account, available money) or administrative properties like comments or favourite color.

¹⁹ http://en.wikipedia.org/wiki/Front_and_back_ends

Backend management

The backend allows to control the configuration of the unga system. It is possible to install/uninstall modules, to configure modules, to create users, etc...

The backend is programmed with Javascript and uses the jquery²⁰ libraries. But, the main advantage is that you can use AJAX²¹ to communicate with existing REST services. For example, you could make a backend extension to ban and logoff users and the call will be done to the "logoff" method of the unga users server, for example.

It should be possible too the access to the inventory service for allowing an inventory management.

The backend is under heavy rethinking and sure it will change in the next months.

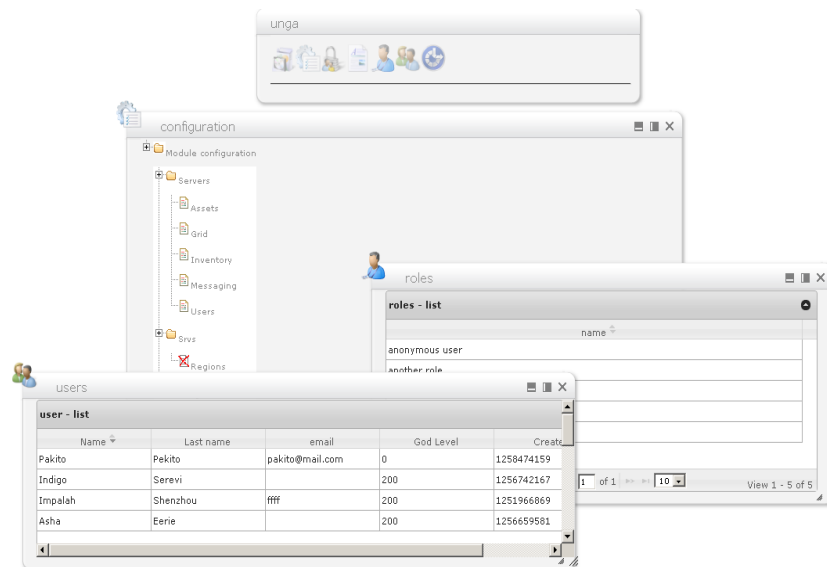


Figure 2: A view of the unga backend

Web integration

How to integrate unga with you existing web? Easy, use REST calls.

If you want, for example, to allow your visitors to create an user from your web just create a services (with PHP, ASP, javascript...) in your web to call the unga users module.

Another way to do it is inserting an IFRAME²² and call to a unga controller... but it is under construction.

Unga is not a CMS²³, so if you pretend to give access to sensible information (like user management, inventories, etc) you have to create the services on your own CMS.

20 <http://jquery.com/>

21 http://en.wikipedia.org/wiki/Ajax_%28programming%29

22 http://en.wikipedia.org/wiki/HTML_element#Frames

23 http://en.wikipedia.org/wiki/Content_management_system

Cache

Unga has 3 operative caches²⁴ right now: file cache (simple serializing into file system), memcache²⁵ (distributed cache) and APC²⁶ cache (present if PHP-APC module active). The bad news are that they aren't implemented in the services right now... it is a task for the future.

There is another cache planned: database cache, similar to the one used in Drupal²⁷.

No-SQL databases

unga does not provide a common architecture for working with no-SQL²⁸ databases but is easy to work with them, just substituting the model we want and implementing the calls.

The near future plans include providing models for the asset module and to use MongoDB²⁹ and Cassandra³⁰.

Signals, extensibility made easy.

Unga has the ability to send signals to the world and to capture those signals. These signals often are called "hooks³¹".

A controller, a library, a model or a function can send a signal and another object can capture the information and work with it. The advantage is that the model that sends the signal hasn't worry about who is receiving it.

For example, you can send a signal of "logged in" each time a user has logged and have a module to capture the "logged in" signals and create statistics. So the responsibility of the statistics can be sent to another module, an not to the authentication one.

24 <http://en.wikipedia.org/wiki/Cache>

25 <http://en.wikipedia.org/wiki/Memcached>

26 <http://www.php.net/manual/en/book.apc.php>

27 <http://drupal.org/>

28 <http://en.wikipedia.org/wiki/NoSQL>

29 <http://www.mongodb.org/>

30 <http://cassandra.apache.org/>

31 http://en.wikipedia.org/wiki/Hook_%28programming%29

<http://ai-designstudio.net>



Development curiosities

I tried to use TDD³² (Test Driven Development) for unga... I couldn't, at least not at 100%. There are some implemented tests (mostly in controllers and libraries), but they are not all completed yet because I am very disorganized. My intention is completing ALL the tests and achieving a coverage of, at least, an 80% of the code.

Tests are made using PHPUnit³³, which provides, using xdebug³⁴ too, code coverage. There is a example of some code coverage graphics below.

The API documentation is made with PHPDocumentor³⁵, so it is available in a easily browsable form.

Phing³⁶ is used for controlling all the building tasks. It launches tests, documentation, and even publishing. This part is well tested and should work well.



Figure 3: Cathedral (+15000 primitives) built in Opensim and using unga as grid server. No lag detected... (this image is here for no reason at all)

32 http://en.wikipedia.org/wiki/Test-driven_development

33 <http://www.phpunit.de/>

34 <http://www.xdebug.org/>

35 <http://www.phpdoc.org/>

36 <http://phing.info/trac/>

Status of unga

Nowadays unga is totally unstable and does not work with Opensim servers. The core has been rewritten and the modules need adjustments.

Unga can be downloaded and installed and it will work. Even the Assets and Inventory servers will work correctly (assets, folders and items) when using REST calls, but the Opensim Connectors aren't ready yet so no region can't connect to it.

Actually there is documentation except for the comments inside the source. I expect some day I will complete tutorials, videos and so on.

The final

unga is intended to be a grid server system. It promises high performance, security and scalability, but the responsibility, in most cases, is in the field of the web server administrator. For high "populated" grids could be a good choice for saving money.

unga is flexible enough to allow whatever configuration you want. If you don't want all the servers in the same machine, split them into several. If you don't want to use a SQL database, use your own data access overriding the existing models.

And of course each day I think in new stuff to add to unga... in the future, when I have some time available :-P

Lino Figueroa (a.k.a. Impalah Shenzhou)

Download unga

Stable, old and no maintained version:

ZIP from <https://sourceforge.net/projects/unga/>

Unstable, development version, from SVN:

```
svn co https://unga.svn.sourceforge.net/svnroot/unga unga
```