Panda3D Distributed Networking

This paper will describe how to use the Panda3D Distributed Networking system. You should have at least basic experiences with the Panda3D game engine as well as knowledge of network development.

1. Server

Server Repository

The Server Repository is the first thing you will need for the distributed system. The repository knows all clients which are connected to it and will listen at the given port for client requests so it can handle them appropriate.

2. Client

Client Repository

The Client Repository keeps track on the distributed nodes of an application.

NOTE: The client repository can only be set up after showbase is instantiated due to the use of DirectNotify within the Distributed Network code.

3. Zones

A zone in a Distributed Network is like a room. You can put many things in it and can have as many as you want. For example you can put general distributed objects like the timeManager in zone one to have all those things together. In other Zones you could have full levels and the clients in this level. Each Client can listen to one or more zones and will react to changes made in these zones.

4. DC Files

Distributed Class (DC) Files contains the definitions of your client classes so that the distributed system knows, how to handle the functions, what they should do, to whom they should go and so on.

Most important type in a DC file is the dclass. Every dclass has a name that not always, but most of the time corresponds with a pyhton class. For example a dclass of an Actor could look like this.

```
import Avatar

dclass Avatar {

        setHeight(uint16 h);

        setName(string n) required;

        setPos(int x, int y, int z);

}
```

Usable types to store variables

int8/16/32/64	**uint8/16/32/64**	**Float64**
char	**string**	**blob**

Hint on floats: There only exists float64. For single precision floating point numbers you can use integers like:

int16 foo/100

This divides the int value in foo by 100 so you get a range of -3.27 - +3.27

Arrays

Each variable can be an array by appending [#] on the end of a value where # is the size of the array. You can also leave the value between the columns empty and it will become a dynamic array. So it could look like this for a fixed:

int8[16] foo

and a dynamic array:

int8[] foo

Allowed range and list of ranges

If you want to only have a specified range of numbers that is allowed to be send or set on a value, you can use it like this:

```
dclass Foo{
        setHam(int16(1-1000,2001-3000)
}
```

This will only allow numbers from 1-1000 and 2001-3000. This can also be used in array declarations.

Required

The required statement at the end of a field determines that this field has to be set at the generation of the object. These fields have to be called set* as a convention as in the python representation of that class the set gets taken away at generation time and will be replaced with a get to call the set value with the get functions return value.

For example, taking the Avatar class as defined in above, we get this pyhton class:

```
class Avatar:
        def getName(self):
                return self.name
        def setName(self, name):
                self.name = name
        def dsetName(self, name):
                self.sendUpdate("setName", [name])
        def bsetName(self, name):
                self.setName(name)
                self.dsetName(name)
```

Note if, for example, the name value gets changed after generation of the DO, it doesn't change the value automatically of the DO on the server. This has to be done manually. Tho, calling the distributed versions of the functions (as defined in the dc file) the corresponding functions of the representing python class will be called on the client.

Further informations about DC files and it's content could already be found at the Panda3D Manual: http://www.panda3d.org/manual/index.php/DC_File

5. Distributed Objects

DistributedObject Class

The Distributed object (DO) class is the base class of all distributed classes. Each DO will have a doID which is just a basic integer variable.

DistributedNode Class

The DistributedNode class is like an extension to a normal PandaNode. They have only set functions which have to be called to update the respective property of the node on the clients.

DistributedSmoothNode Class

The smooth node class can be used for objects, which have to be smoothly updated on all clients at the same time.

DistributedActor Class

The DistributedActor is an extension to the “normal” Actor class and only reimplement a few actor specific functions to publicize the actor behavior over to the clients. They inherit DistributedNode so they will be able to updated the properties of the node on all clients.

NOTE: They are set to be cacheable

AI Classes

Of all the above Distributed classes, specialized AI classes do exist. These AI classes represent clients which manage all of the server-created objects in the world. AI clients do have some special privileges like creating distributed objects and sending specialized messages that the other clients normally are not allowed to.

TimeManager

Taken from the API documentation:

This DistributedObject lives on the AI and on the client side, and serves to synchronize the time between them so they both agree, to within a few hundred milliseconds at least, what time it is.

It uses a pull model where the client can request a synchronization check from time to time. It also employs a round-trip measurement to minimize the effect of latency.

Sending Messages between Server and Clients

sendUpdate function

6. Possible Errors and how to fix them

In a Server

In a Client

In the DC Files

The Error:

StandardError: Symbol [ModuleNameAI] not defined in module [ModuleName].

Possible Occurrences:

This means that the Module ModuleNameAI could not be found in the module ModuleName. It could be an import error if you write

From ModuleA import ModuleA/AI

by mistake and the ModuleAI class is not inside the ModuleA for instance, or if you forgot to write a function, variable or anything else and try to import it.

How to Fix:

If you write the wrong import statement, like above the

From ModuleA import ModuleA/AI

then this could be fixed by simply changing it to the following line

import ModuleA/AI