Instituto Tecnológico y de Estudios Superiores de Monterrey.

Universidad Virtual.

Campus Guadalajara.

Maestría en Ciencias de la computación.

Reporte Técnico del proyecto practico:

0 4 MAY 2007

"Programación de juegos en Red, con X-CALIBER".

BIBLIOTECA

Que presenta:

Cristóbal Montes de oca Villarreal.

260310

Matricula 351865

19 de Noviembre del año 2001.

Capitulario.

Capitulo 1.- Introducción é Historia........ 1
1.1.¿Qué es X-CALIBER?........ 2
1.2. Este reporte y X-Caliber........ 3

Capitulo 2.- Primera etapa de Desarrollo........ 4

2.1. Introducción y descripción........ 4
2.2. Diseño para la primera etapa de la API cliente........ 4
2.3.¿Servidores Dummy?........ 9
2.4.Como usar del CD-ROM la demostración del código de la primera Etapa. 9
2.5 Logros según el "Milestone Gaheris" para la Primera Etapa........ 11
2.6. Conclusiones de la etapa........ 11

Capitulo 3.- Segunda etapa de Desarrollo.

3.1. Introducción y descripción........ 13
3.2. Refinamiento........ 13
3.2.1. Esta es mi versión de la clase queue – Prophet........ 13
3.2.2. Esta es la versión depurada de la clase queue – Prophet........ 15
3.3. Como ejecutar del código del CD-ROM........ 16
3.4. Logros según el "Milestone Gaheris" para la Segunda Etapa........ 16
3.5. Conclusiones de la etapa........ 17

Capitulo 4.- Tercera Etapa de desarrollo........ 18

4.1. Introducción y descripción........ 18
4.2. Diseño del API del cliente de esta etapa........ 18
4.2. ¿Cómo funciona la interacción del cliente y el servidor?........ 19
4.2.1. Descripción básica del "engine X-Caliber" en el API cliente.... 19
4.2.2. Descripción del servidor dummy OverLord........ 20
4.3. Como ejecutar del código del CD-ROM........ 22
4.4. Logros según el "Milestone Gaheris" para la Tercera Etapa........ 23
4.5. Conclusiones de la etapa........ 23

Apéndice A. Esta es la definición oficial del MileStone Gaheris........ 26

Apéndice B. Diseño global del Sistema X-Caliber........ 29

Apéndice C. Bibliografía........ 41

Apéndice D. Glosario y Referencias........ 42

Capitulo 1.- Introducción é Historia.

Antaño lo mas cercano a un ambiente "Multi – usuario" fueron los famosos BBS. Auque los comentarios que uno ponía en los BBS tardaban a veces minutos u horas en que alguien mas los contestara. En los tiempos mozos del BBS existió una plataforma conocida como C-NET, que era un sistema de BBS cuyo código permitía cierto grado de personalización, de tal manera que era posible que se usara para juegos en línea. Los juegos eran simples, pero divertidos, la mayoría trataba de construir una Civilización Mítica, por medio de la inclusión de otros jugadores y luego de ir a la guerra unos contra otros. Todos estos juegos estaban basados en texto plano y solo un jugador a la vez podía hacer sus movimientos en el juego.

Por su parte el genero RPG para juegos de mesa es detallado en cuanto al nivel estadístico del estado en que se encuentra cada uno de los participantes del juego.

Hace no mucho tiempo el genero RPG entro al mundo de los juegos de computadora de tal manera que los personajes con los que el usuario interactúa pretenden emular las acciones de otros jugadores reales. A estos se les conoce como personajes NPC. Sin embargo algunos juegos recientes del genero RPG han tenido un éxito impresionante al permitir que sean usuarios reales en línea, los que interactúen en ambientes de juegos virtuales. De tal manera que los personajes, los ambientes, las armas y las acciones en general que toman parte en lo juegos, ahora no tienen que ser imaginados por lo jugadores, si no que son representados con los algoritmos de graficas computacionales mas potentes, para aparentar realismo.

De esta forma juegos como Ultima - Online ó Everquest de Sony están pobladísimos de usuarios, de cualquier parte del mundo, tomando parte en las aventuras que estos ambientes plantean.

Detrás del éxito de estos juegos para "Multi – Usuarios" existe una muy bien diseñada plataforma Cliente / Servidor y de ambientes distribuidos, que permite el computo masivo que se requiere para permitir la interacción realista y en tiempo real, de los que se citan en los juegos.

X-CALIBER surge gracias al movimiento de Código Abierto que se ha suscitado en los últimos años, permitiendo que desarrolladores independientes trabajen en forma colaborativa, para que esta plataforma se realice y sea gratuita, estando disponible para desarrolladores amateurs o profesionales.

Yo fui invitado a ser parte del diseño de la plataforma desde su registro original a la comunidad de Código Abierto, y auxilie a generar el diseño global de la plataforma que se presenta en su versión original en el Apendice B.

1.1¿Qué es X-CALIBER?.

El proyecto X-CALIBER pretende producir un sistema que correrá en Linux en forma de servidor para Juegos RPG en línea que sirva para muchos jugadores a la vez. Esto permitirá que juegos RPG de tipos variados usen esta infraestructura. Los clientes o aplicaciones corriendo del lado del usuario, podrán implementarse de tal manera que muestren ya sea en formato de texto, gráficos 2d ó gráficos 3D lo que esta sucediendo en el ambiente masivo, desde la perspectiva personal de la posición virtual de cada usuario. Este sistema estará orientado a objetos pretendiendo que sea altamente escalable y muy flexible para que se incorporen módulos nuevos a los servidores.

Las primera etapa (Milestone) implementa Sockets en el código del cliente y Servidores, sin embargo posteriormente se utilizara CORBA para las versiones definitivas.

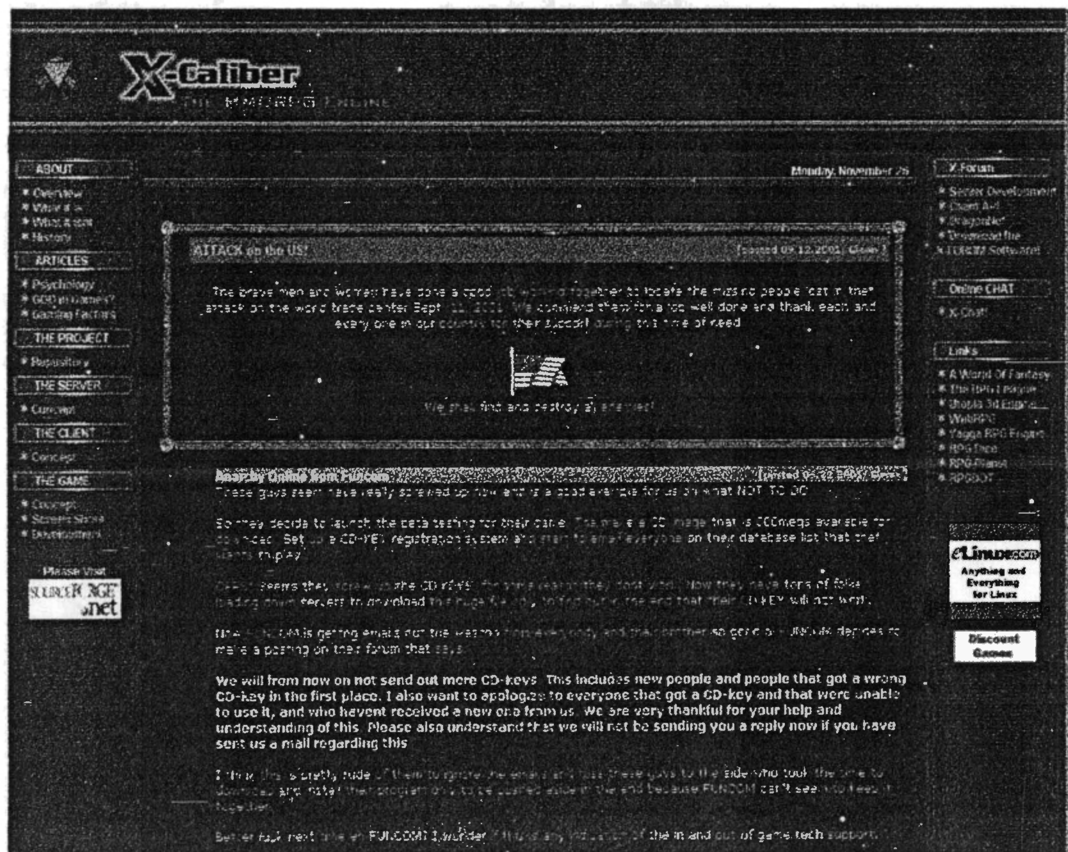


Figura 1.1. Sitio Web de X-Caliber (http://www.x-caliber.org).

Nota: X-Caliber en español puede leerse como "De cualquier calibre" y esto es por que se pretende que existan clientes para el sistema desde varios ambientes operativos. Sin embargo los servidores se desarrollan para Linux.

1.2. Este reporte y X-Caliber.

Al igual que otros proyectos de código abierto, X-Caliber, es una plataforma que se pretende construir basada en "Milestones", en pocas palabras una "Milestone" es una lista de las cosas que debe de hacer y/o poseer el código de determinada etapa de la constricción del software. O sea si se completan todos los puntos de un "Milestone" entonces puede liberarse una versión de determinado sistema.

El equipo de desarrollo del sistema X-Caliber se divide en dos partes: La primera esta integrada por los programadores voluntarios de los distintos servidores y la segunda por un equipo de desarrolladores del API cliente (hasta el momento con un solo voluntario, "Su servidor").

Por lo anterior este reporte técnico se concentra en el primer "Milestone" para el desarrollo de X-Caliber llamado Gaheris (Vea el Apéndice A, para la descripción completa del mismo y el Apendice B, para ver el contenido del diseño global de X-Caliber). Y también se concentra en la parte del diseño e implementación del API del cliente para la mencionada plataforma. Esto debido a que el trabajo tanto de diseño y programación que presento (en el CD-ROM adjunto), son referentes al API del cliente.

El resto de los capítulos del presente reporte especifica entonces, las tres etapas por las que se atravesó para el desarrollo del API cliente de X-Caliber.

Capitulo 2.- Primera etapa de Desarrollo.

2.1. Introducción y descripción:

La primera etapa del desarrollo del diseño del API del cliente para X-Caliber y su "Milestone Gaheris" se desarrollo para el ambiente Linux. El primer inconveniente que existía para desarrollar el diseño de este API, fue que los primeros programadores que trabajaron para X-Caliber únicamente trabajaron, sobre el código de los servidores que el diseño básico de X-Caliber especifica, sin tomar en cuenta como se desarrollaría un cliente global para la plataforma.

Además el código de los servidores requiere, no solo que exista una interfaz de comunicación con el cliente, si no también entre los servidores mismos para poder trasladar a los clientes entre unos y otros (que es la razón de que existan servidores múltiples) .

En resumen existían versiones de tres de los servidores, pero estos funcionaban solo con sus clientes particulares, y además no eran capaces de comunicarse unos a otros.

Keir Davis (El líder de diseño del los servidores de X-Caliber) me solicito entonces que para el diseño de la primera etapa del API del cliente, diseñara también, como deberían de comunicarse el cliente, con una forma estándar, con cada uno de los servidores y también, como debería de ser la comunicación entre servidores.

Debido a lo anterior genere la siguiente especificación:

2.2. Diseño para la primera etapa de la API cliente para el "Milestone Gaheris":

Metáfora. El modelo metafórico empleado para desarrollar la API del cliente para X-Caliber se llama: Relación "Señor – Profeta" o "Lord – Prophet", la cual es análoga a la relación "Cliente – Servidor", donde un Profeta es un cliente y un Lord es un servidor, cada profeta posee un "Oracle" o Oráculo (analogía a socket) y usando su Oráculo puede realizar las siguientes acciones: 1.- "Cast – Lord" o Invocar a su Señor (conectarse al servidor), 2.- "Speak to Lord" o Hablar al señor (equivalente a SEND), 3.- "Listen to Lord" o Escuchar al señor (equivalente a RECV). De esta forma cada clase profeta que posee el cliente, puede comunicarse únicamente con su respectivo Lord, pero puede comunicarse con los otros profetas para compartir información. Como puede observarse el formato de comunicación es una envoltura especial de la interfaz de los "Sockets BSD".

Por lo tanto la comunicación deberá definirse para los siguientes elementos.

- GateKepper dummy Lord & a GateKepper prophet.
- Queue dummy Lord & Queue prophet.

- Lobby dummy Lord & Lobby prophet

Cada mensaje entre los elementos de mencionados, se representa en el diagrama con una flecha de este tipo: -Strc#→ donde # es el numero de la estructura de datos que se envía y recibe.

El siguiente es el diagrama que define el orden de envío y recepción de mensajes entre los Profetas y los servidores "Dummy" o (servidores de prueba):

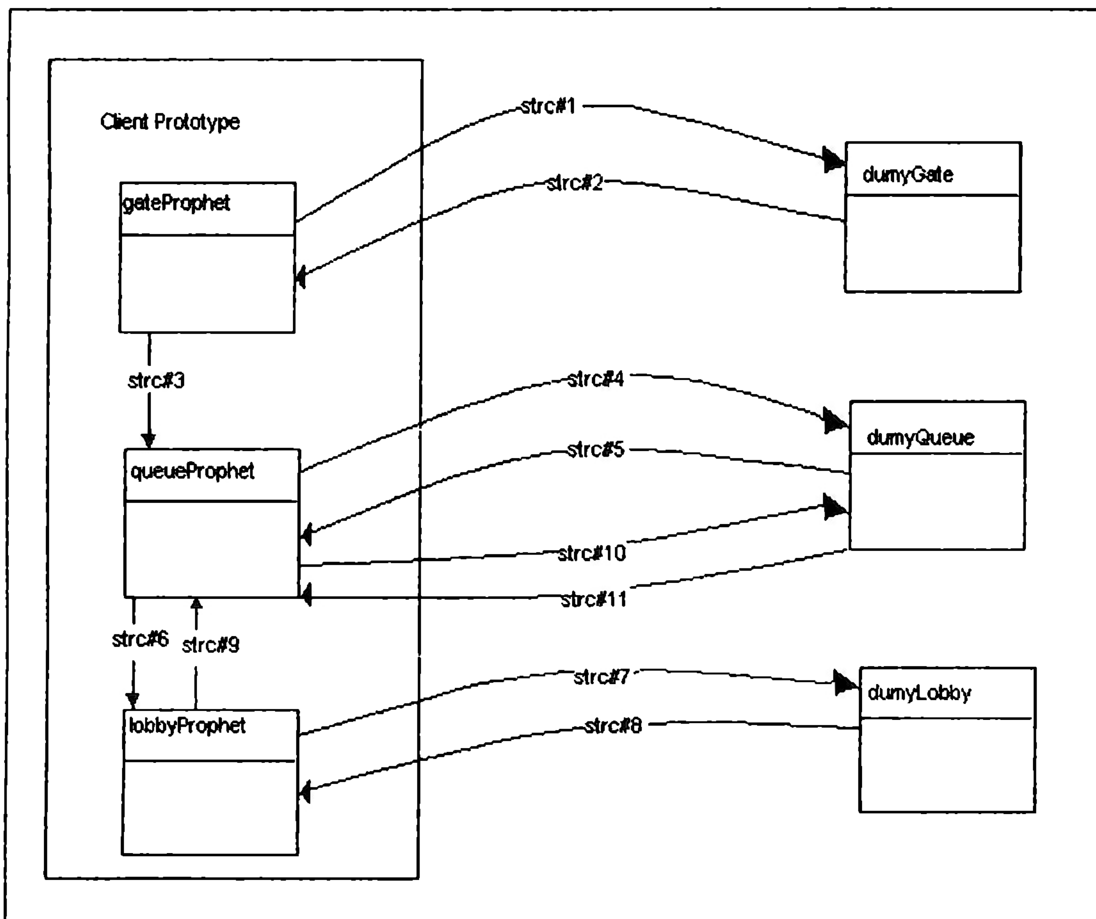


Figura 2.1. Diagramas de comunicación para el caso de uso que los servidores de prueba soportan en esta etapa.

Las siguientes tablas muestran el contenido de las estructuras de comunicación del diagrama anterior. Este contenido es precisamente el que se maneja en el código de ejemplo que se incluye en el CD-ROM de demostración para la primera etapa:

Strc#1: gateProphet --> gateLord : **loginStruct**	
Struct:	gateKepperLoginStruct

```
cmd = CMD_GATE_LOGIN  //gateKeeper server Command
parm1 =LOBBY_SRV //Destination server type
parm2 = 0 // Not required
parm3 = 0 // Not required
userName[16] ="xtobal" //username
passWord[16] ="supercool"//password
```

Strc#2 gateLord --> gateProphet : **output4QueueProfet**

Struct: cmdStruct

```
cmd = CMD_GATE_OK; //gateKepper status replay.
parm1 = // Binary IP of  Queue Lord;
parm2 =// Port of Queue Lord;
parm3 =// Queue Lord Session ID.
```

Strc#3 gateProphet --> queueProphet : **fromXProfet**

Struct: cmdStruct

```
cmd = CMD_GATE_OK; //gateKepper status replay.
parm1 = // Binary IP of  Queue Lord;
parm2 = 2001 // Port of Queue Lord;
parm3 =// Queue Lord Session ID.
```

Strc#4 queueProphet --> queueLord : **input4XLord**

Struct: cmdStruct

```
cmd = CMD_QUEUE_CLIENT; //queueServer command
parm1 = LOBBY_SRV //Destination Server Type
parm2 = 0 // Not required
parm3 = // Queue Lord Session ID.
```

Strc#5 queueLord --> queueProphet: **output4XProfet**

Struct: cmdStruct

```
cmd = CMD_QUEUE_OK; //queueServer status reply
parm1 = //Lobby Server BINARY IP
parm2 = 2003 // Lobby Lord Port
parm3 = // Lobby Lord Session ID.
```

Strc#6 queueProphet --> lobbyProphet:**inputFromQueueProfet**

Struct: cmdStruct

cmd = CMD_QUEUE_OK; //queueServer status reply
parm1 = //Lobby Server BINARY IP
parm2 = 2003 // Lobby Server Port
parm3 = // Lobby Lord Session ID.

Strc#7 lobbyProphet --> lobbyLord: **input4LobbyLord**

Struct: cmdStruct

cmd = CMD_LOBBY_CLIENT // Lobby Lord command
parm1 = OVERLORD_SRV // Destination Server
parm2 = 0 // Not required
parm3 = // Lobby Lord Session ID.

Strc#8 lobbyLord --> lobbyProphet: **output4QueueProfet**

Struct: cmdStruct

cmd = CMD_LOBBY_OK // Lobby Lord status replay
parm1 = // Binary IP of Queue Lord;
parm2 = 2001 // Port of Queue Lord;
parm3 =// Queue Lord Session ID..

Strc#9 lobbyProphet --> queueProphet: **fromXProfet**

Struct: cmdStruct

cmd = CMD_LOBBY_OK // Lobby Lord status replay
parm1 = // Binary IP of Queue Lord;
parm2 = 2001 // Port of Queue Lord;
parm3 =// Queue Lord Session ID..

Strc#10 queueProphet --> queueLord: **input4XLord**

Struct: cmdStruct

cmd = CMD_QUEUE_CLIENT; //queueServer command
parm1 = OVERLORD_SRV //Destination Server Type
parm2 = 0 // Not required
parm3 = // Queue Lord Session ID.

Strc#11 queueLord --> queueProphet: **output4XProfet**	
Struct:	cmdStruct

cmd = CMD_QUEUE_OK; //queueServer status reply
parml = //OVERLORD BINARY IP
parm2 = 2004 // OVERLORD Port
parm3 = // OVERLORD Session ID.	

A continuación muestro un extracto de código del archivo xcProphets.h (del código de demostración para la primera parte) que define las estructuras usadas por las tablas anteriores:

```
//Server ID's
#define QUEUE_SRV      0
#define GATEKEEPER_SRV 1
#define LOBBY_SRV      2
#define OVERLORD_SRV   3

//Queue Server Commands
#define CMD_QUEUE_TRANSFER 0
#define CMD_QUEUE_POLL     1
#define CMD_QUEUE_OK       2
#define CMD_QUEUE_ERROR    3
#define CMD_QUEUE_CLIENT   4

//Gatekeeper Server Commands
#define CMD_GATE_OK          100
#define CMD_GATE_ERR         101
#define CMD_GATE_LOGIN       102

//Lobby Server Commands
#define CMD_LOBBY_OK         200
#define CMD_LOBBY_ERR        201
#define CMD_LOBBY_CLIENT   202

//Overlord Server Commands
#define CMD_OVERLORD_OK    300
#define CMD_OVERLORD_ERR   301
#define CMD_OVERLORD_LOGIN 302

// Communication Structures
struct gateKepperLoginStruct{

        unsigned long cmd;
        unsigned long parml;
        unsigned long parm2;
        unsigned long parm3;
        char userName[16];
        char passWord[16];
```

```
};

struct cmdStruct{

        unsigned long cmd;
        unsigned long parm1;
        unsigned long parm2;
        unsigned long parm3;

};
```

2.3.¿Servidores Dummy?.

Como habrá notado empleo la terminología "Servidor Dummy", la razón es que los servidores que construí, tienen funcionalidad parcial, o sea tienen la funcionalidad mínima que se requiere para poder probar a un cliente en distintos casos de uso. Este mecanismo se empleo así, debido a que no existían en X-Caliber servidores compatibles unos con otros ni con un cliente en general. Estos servidores dummy sirvieron después para definir como se desarrollarían los servidores de la segunda etapa.

2.4.Como usar del CD-ROM la demostración del código de la primera Etapa.

Si su unidad CD – ROM por defecto se encuentra en la unidad D, el archivo comprimido del código será D:\Primera-Etapa\x-caliber-client-api-FirstStage.zip.

Lo primero que deberá hacer es descomprimir este archivo en un ambiente Linux,

Abra una consola y diríjase a :
cd x-caliber-client/client/dumyServers/
compile con :
g++ -o dGate dumyGate.cpp
Ejecute con:
dGate
Abra otra consola y dirijase a :
cd x-caliber-client/client/dumyServers/
compile con:
g++ -o dQueue dumyQueue.cpp
ejecute con:
dQueue
Luego abra otra consola y dirijase a:
cd x-caliber-client/client/dumyServers/
compile con:
g++ -o dLobby dumyLobby.cpp
ejecute con:
dLobby
Por ultimo abra una ultima consola para el cliente y diríjase a:
cd x-caliber-client/client/

compile con:
g++ -o xc xcMain.xx

Ejecute:
    xc
Ejecute varias veces para simular a varios clientes:
    xc
    xc
Si todo funciona correctamente, los servidores dummy le mostraran en sus consolas lo que han recibido y enviado a cada cliente.

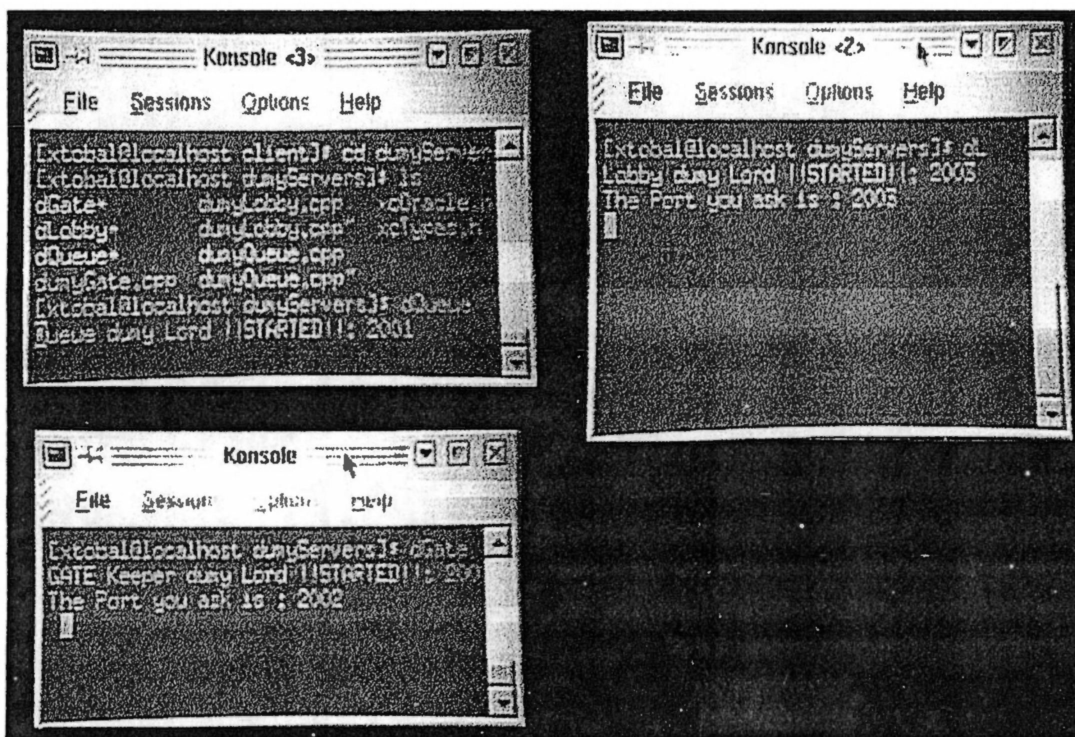


Figura 2.2. Servidores esperando la ejecución de un cliente en un ambiente Linux.

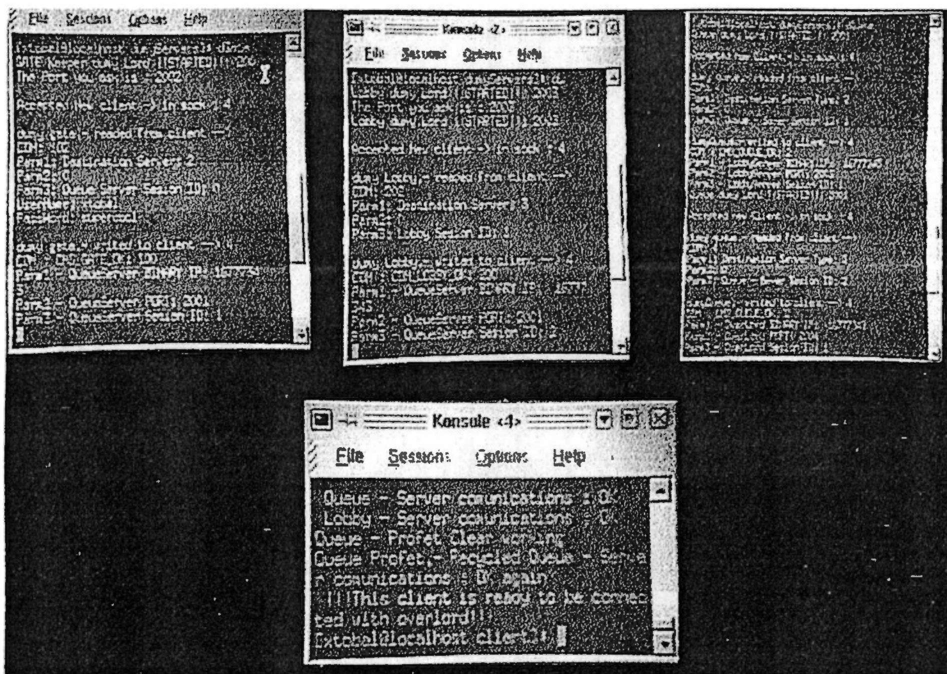


Figura 2.3. Cliente y Servidores. Una vez que el cliente se ha ejecutado.

2.5 Logros según el "Milestone Gaheris" para el cliente de esta Primera Etapa:

El cliente deberá:

- Conectarse al Servidor GateKepper. **"Se logro".**
- Poder atender el manejo que los servidores le dan al cliente para trasladarlo de servidor en servidor. **"Se logro".**
- Interactuar con el servidor para permitir que los usuarios se puedan mover en un mundo. **"No se logro".** En esta etapa no existía ningún servidor OverLord aun.
- Manejar envió de mensajes de tipo Chat entre los jugadores. **"No se logro",** al igual que el punto anterior requería de un servidor OverLord.

Nota para ver el la definición del "Milestone Gaheris" completa diríjase al Apéndice A.

2.6. Conclusiones de la etapa.

Una vez definida la forma en el cliente debería interactuar con cada servidor, Keir y yo nos reunimos usando la aplicación "Mirc". Durante esta junta el decidió que podía refinar el código que yo le entregue, de tal manera que se redusca altamente el numero de líneas de código, utilizando algunos artificios de la orientación a objetos, pero respetando el concepto original de los "Lords" y los "Prophets".

Por otro lado también decidimos que el código de los servidores que se utilizaría en la segunda etapa no estaría diseñado pensando en la orientación a objetos, si

no mas bien en un esquema basado en procedimientos. Esto para agilizar su funcionamiento y sobre todo para reducir la complejidad de los primeros intentos de los servidores que ya habíamos construido.

En el siguiente capitulo demuestro con mas detalle lo que se logro en la segunda etapa de desarrollo.

Capitulo 3.- Segunda etapa de Desarrollo.

3.1. Introducción y descripción:

En esta etapa del "MileStone Gaheris" se desarrollaron al fin, versiones estables de tres de los servidores que conforman el diseño básico de X – Caliber estos servidores son: "GatteKeeper – Server", "Queue – Server", "Lobby – Server". El desarrollo de estos servidores se logro en gran medida, gracias a que el diseño del cliente de la primera etapa, definía la forma de comunicación que deberían tener los servidores para poder atender a un cliente en particular, y poder trasladarlo de servidor en servidor hasta llegar al servidor "OverLord" como destino final.

Por otra parte los compañeros programadores de X-Caliber hicieron un excelente trabajo de refinamiento sobre mi código original del API cliente:

3.2. Refinamiento:

El punto mas fuerte que el desarrollo de "Código Abierto" proporciona al diseñador, es la "Depuración masiva" que el código de diseño puede sufrir al pasar por la mano experta de varios programadores. A continuación tenemos dos muestras de código, la primera muestra mi implementación de la clase "queue - Prophet" y la segunda muestra la versión altamente refinada de la misma clase, después de haber sido depurada por varios programadores.

3.2.1. Esta es mi versión de la clase queue - Prophet y sus funciones miembro:

```
//Definition for a Queue - Profet Class
class xcQueueProfet {

private:
        char host[16];
        char port[16];

        struct cmdStruct input4XLord;
        struct cmdStruct output4XProfet;

        void clear(void){
         input4XLord.cmd = 0;
         input4XLord.parm1= 0;
         input4XLord.parm2= 0;
         input4XLord.parm3= 0;

         output4XProfet.cmd = 0;
         output4XProfet.parm1= 0;
         output4XProfet.parm2= 0;
         output4XProfet.parm3= 0;

         host[0]='\0';
         port[0]='\0';
```

```
            cout <<"Queue - Profet Clear working "<<'\n';
          }

public:
          xcQueueProfet(){
            cout<<"Queue - Profet Constructor Working "<<'\n';
            clear();
          }

          cmdStruct GetQueueProfetOutput(cmdStruct fromXProfet){
            return output4XProfet;
          }

//        void ~xcQueueProfet(/*destroy()*/
          void recycle(){
            clear();
      cout<<"Queue Profet.- Recycled";
          }

          void setQueueProfetInput(cmdStruct inputFromXProfet){

           struct in_addr inet_address;

           inet_address.s_addr=inputFromXProfet.parm1;
     strcpy(host,inet_ntoa(inet_address));

           sprintf(port,"%u",inputFromXProfet.parm2);
           input4XLord.cmd = CMD_QUEUE_CLIENT;    //Queue Command

           if (inputFromXProfet.cmd==CMD_GATE_OK){
                  input4XLord.parm1 = LOBBY_SRV;     //Destination Server Type: LOBBY_SRV
           }
           else{
             input4XLord.parm1 = OVERLORD_SRV;   //Destination Server Type: OVERLORD_SRV
           }

           input4XLord.parm2 = 0;
           input4XLord.parm3 = inputFromXProfet.parm3; //Queue-Server Sesion ID

          }

          int useOracle(){

            int s,status;
            xcOracle Oracle;

            s = Oracle.castLord(host, port);
            if(s < 0)cout<<"Can't create or connect a socket for QueueServer";
            else {
              Oracle.writeToOracle(s,(unsigned char*)&input4XLord,sizeof(cmdStruct));
              status=Oracle.readFromOracle(s,(unsigned char*)&output4XProfet,sizeof(cmdStruct));
              close(s);
              if(status!= 0)s = -1;
```

```
            }
            return s;
          }

          cmdStruct getQueueProfetOutput(){
            return output4XProfet;
          }
};
```

3.2.2. Esta es la versión depurada de la clase queue - Prophet y sus funciones miembro para la segunda etapa de desarrollo.

```
class cQueueProphet
{
public:
          cQueueProphet();
   cQueueProphet(cQueueProphet &copy);
   ~cQueueProphet();

public:
   queueServStruct* connect(string host, int port, unsigned long sessionID);

private:
   cOracle oracle;
};

cQueueProphet::cQueueProphet()
{
}

cQueueProphet::cQueueProphet(cQueueProphet &copy)
{
}

cQueueProphet::~cQueueProphet()
{
}

queueServStruct* cQueueProphet::connect(string host, int port, unsigned long sessionID)
{
   queueServStruct *queueCmd = new queueServStruct;
   int status;

   status = oracle.connectToLord(host, port);
   if (status <= 0) {
      delete queueCmd;
      return NULL;
   }

   memset(queueCmd, 0, sizeof(queueServStruct));
   queueCmd->cmd = CMD_QUEUE_CLIENT;
   queueCmd->parm1 = sessionID;
```

```
  status = oracle.speakToLord((unsigned char *) queueCmd, sizeof(queueServStruct));
  if (status != SOCK_OK) {
    delete queueCmd;
    return NULL;
  }

  status = oracle.listenToLord((unsigned char *) queueCmd, sizeof(queueServStruct));
  if (status != SOCK_OK) {
    delete queueCmd;
    return NULL;
  }

  status = oracle.listenToLord((unsigned char *) queueCmd, sizeof(queueServStruct));
  if (status != SOCK_OK) {
    delete queueCmd;
    return NULL;
  }

  return queueCmd;
}
```

3.3. Como ejecutar del código del CD-ROM la demostración del código de la segunda Etapa.

Si su unidad CD – ROM por defecto se encuentra en la unidad D, el archivo comprimido del código será D:\Primera-Etapa\ x-caliber_20010930.tar

Este código requiere también de un ambiente Linux.

1. Abra una ventana consola.
2. Luego ejecute : tar zxvf x-caliber_20010930.tar.gz , para descomprimir el archivo del codigo.
3. Diríjase a : cd x-caliber
4. Ejecute : make
5. Abra otra tres ventanas de consola.
6. En la primera ejecute : ./bin/queueServ
7. En la segunda ejecute : ./bin/gateServ
8. En la tercera ejecute : ./bin/lobbyServ
9. Y por ultimo en la cuarta ejecute : ./bin/testclient .

Si todo resulta bien deberá ver que el cliente de pruebas se conecta al servidor gateKeeper y luego el cliente es transferido al servidor Lobby.

3.4. Logros según el "Milestone Gaheris" para la Segunda Etapa:

Queue Server

El servidor Queue deberá tener su funcionalidad completa, permitiendo que varios servidores transfieran clientes entre ellos. **"Se logro".**

Gatekeeper Server

El servidor gateKeeper deberá de ser capaz de:

- Aceptar conecciones de los clientes. **"Se logro".**
- Transferir clientes al servidor Lobby usando el servidor Queue. **"Se logro".**
-

Lobby Server

El servidor lobby deberá ser capaz de :

- Preguntar al servidor <u>Queue</u>, si existen nuevos clientes. **"Se Logro"**
- Transferir a los clientes al servidor OverLord. "**Se logro,** pero no existía aun en esta etapa un servidor OverLord".

3.5. Conclusiones de la etapa.

Como puede verse en los logros de "Milestone" que se obtuvieron, era necesario diseñar la tercera parte del API del cliente, ahora pensando como se comunicaría el servidor OverLord (en esta parte aun no diseñado) con el cliente.

En esta segunda etapa mi labor fue de soporte y no de desarrollo directo, debido a que consistió básicamente en el refinamiento de lo que ya se había realizado en la primera etapa.

Capitulo 4.- Tercera Etapa de desarrollo.

4.1. Introducción y descripción:

Lo que estaba pendiente del "MileStone Gaheris" al principio de esta etapa, era definir el diseño del servidor mas importante del sistema X-Caliber : "El servidor OverLord" y además diseñar una nueva clase para el API del cliente que interactuara con este servidor. Al igual que en las otras etapas, decidí llamar a la clase : "overLord – Prophet". Esta clase se torno mucho mas compleja que el resto del las clases profeta que había diseñado hasta ahora.

Por otro parte una nueva complicación se dio en este punto : No existía ningún servidor previo del tipo "overLord", por lo que la única forma de especificar la interacción entre la parte del API cliente y el servidor OverLord era de nuevo crear un servidor de tipo dummy (pero esta vez mucho mas complejo que los anteriores también), ambos, tanto el cliente como el servidor deberían tener características similares, para monitorear el estado del juego y de los jugadores a nivel local y remoto, los clientes y el servidor OverLord deberían de tener sincronizaciones periódicas.

4.2. Diseño del API del cliente de esta etapa:

El siguiente es el diagrama de clase para la clase profeta OverLord generado por una aplicación de diseño case.

Figura 4.1 Diagrama de clase del OverLord Prophet.

4.2. ¿Cómo funciona la interacción del cliente y el servidor?.

Tanto el servidor dummy overLord como el profeta, poseen características similares para poder mantener el estado del juego, de hecho podemos visualizar a ambos como un "engine de juegos" completo pero sin gráficos. Donde todo lo que ocurre en un juego determinado se lleva al cabo basado en la manipulación de las estructuras de datos internas.

4.2.1. Descripción básica del "engine X-Caliber" en el API cliente.

El núcleo central de todo "engine de juegos" esta basado en la estructura de datos que contenga a los objetos dinámicos de un juego. Estos objetos dinámicos pueden ser los personajes del juego, las balas, los llamados "items" y todo aquello que no sea estático en un juego. Por lo tanto el conjunto de los algoritmos básicos que se requieren para manipular a esta estructura de datos conforman precisamente al "engine del juego". A continuación muestro la estructura de uno de los nodos de la lista de elementos dinámicos para el API X-Caliber:

```
struct PLAYER_INFORMATION
{
	bool		bActive;
	long		lPlayerID;
	float		fActualX;
	float		fActualY;
	float		fActualZ;
	float		fLastX;
	float		fLastY;
	float		fLastZ;
	float		fVelocity;
	int		iFrame;
	float		fRot;
	char		szPlayerName[32];
};
```

Esta estructura esta pensada para únicamente personajes, no para balas o "Items" o otros objetos, sin embargo puede definirse una estructura similar para manejar con X-Caliber a elementos de este tipo.

En el cliente deberá existir una definición de un arreglo de elementos con el máximo numero jugadores que una escena pueda contemplar. Por ejemplo si se desea que existan 100 jugadores como máximo. Un arreglo quedaría de esta manera.

```
PLAYER_INFORMATION Lista_De_Jugadores[100];
```

El API del cliente posee entonces una abstracción orientada a objetos, que permite acceder a los datos de esta lista para obtener información de determinado jugador, pero solo pude modificar los datos del jugador local.
Por ejemplo.
Si se desea saber la posición en X, Y y Z del jugador local no se accederá a ellos directamente de la lista si no se utilizaría el siguiente código:

```
Long  X,Y,Z;

X  =  myOverLordProphet.getLocalActualX();
Y =   myOverLordProphet.getLocalActualX();
Z  =  myOverLordProphet.getLocalActualX();
```

En el codigo anterior el objeto myOverLordProphet es una instancia de la clase overlordProphet que engloba la funcionalidad de todo el API del cliente X-Caliber para esta etapa.

Por su parte los objetos no locales no se actualizan por medio del API de X-Caliber, si no por medio de una función en un Hilo de Windows que esta constante mente monitoreando los mensajes que llegan del servidor. Por lo tanto los accesos a la lista de jugadores deberán estar protegidas con las llamadas "Critical Sections" del API de windows, para evitar problemas entre el hilo principal y el hilo de lectura de información del servidor.

Esta función deberá de ser declarada de tipo estática y deberá llamarse

```
s_g_oPh_vProphetMessageHandler(void *arg);
```

Esto para que la pueda identificar el objeto overLord, esta limitación se removerá posteriormente utilizando funciones de tipo Callback para dar mas seguridad al programador.

Nota1: En el siguiente "Milestone" de X-Caliber se planea que la lista no posea un numero de elementos finito, si no que se utilice un sistema de listas dinámicas.

Nota2: Cabe señalar que el programador que utilice X-Caliber no debe preocuparse por los problemas de las "Secciones criticas" del concepto de hilos múltiples, ya que los accesos a la Lista de jugadores se hacen por medio del API de X-caliber.

4.2.2. Descripción del servidor dummy OverLord.

Como había enunciado anteriormente el servidor OverLord representa la parte medular del sistema X-Caliber. Para poder diseñar la interacción entre el cliente X-Caliber y este servidor me vi forzado a realizar una versión Dummy de este servidor, por decirlo de alguna forma este servidor no es tan "Dummy" como los anteriores de hecho es casi completamente funcional, pero posee muchas limitaciones debido a que se desarrollo pensando en ser una aplicación de consola

BIBLIOTECA
289370
...OLOGICO Y DE ESTUDIOS SUPERIORES
CAMPUS
ESTADO

de windows, lo que limita en gran medida la capacidad de usuarios del mismo (De hecho el limite es 4 aproximadamente)._ No he generado una versión de este servidor dummy como aplicación de windows para darle mas poder, debido a que la implementación oficial será sobre Linux, y la versión de consola del mismo es suficiente para guiar al equipo de desarrollo de servidores de X-Caliber.

El servidor deberá mantener una lista de jugadores de la misma longitud que el máximo permitido en el cliente y viceversa, sin embargo el servidor posee dos copias de esta lista una para escrituras y una para lecturas.

Dicho de otro modo el servidor esta constituido por 3 hilos de ejecución el primero es el principal y su trabajo después de iniciar a los otros dos hilos, es el de copiar la información que existe en una primer lista a una segunda lista, esto cada vez que las "Critical Sections" de Windows se lo permiten.

Uno de los hilos del servidor esta constantemente escuchando por los mensajes de los clientes y actualizando a una primer lista, luego el otro hilo del servidor lee en una segunda lista la información de cada cliente y entonces informa a todos los clientes del estado global del juego, esto cada 10/segundo.

Lo anterior se hace de esta forma para poder informar a todos los clientes de el estado del juego, al mismo tiempo de que el servidor esta siendo actualizado con datos de los distintos clientes, si no se hiciera en forma concurrente, existiría un gran retraso entre cada envió de información a los clientes. Por otra parte es necesario mantener dos listas, debido a que los mensajes que lleguen mientras se esta informando del estado del juego, simplemente se perderían.

Las siguiente figura muestra una consola con el servidor dummyServer y otras dos con clientes interactuando:

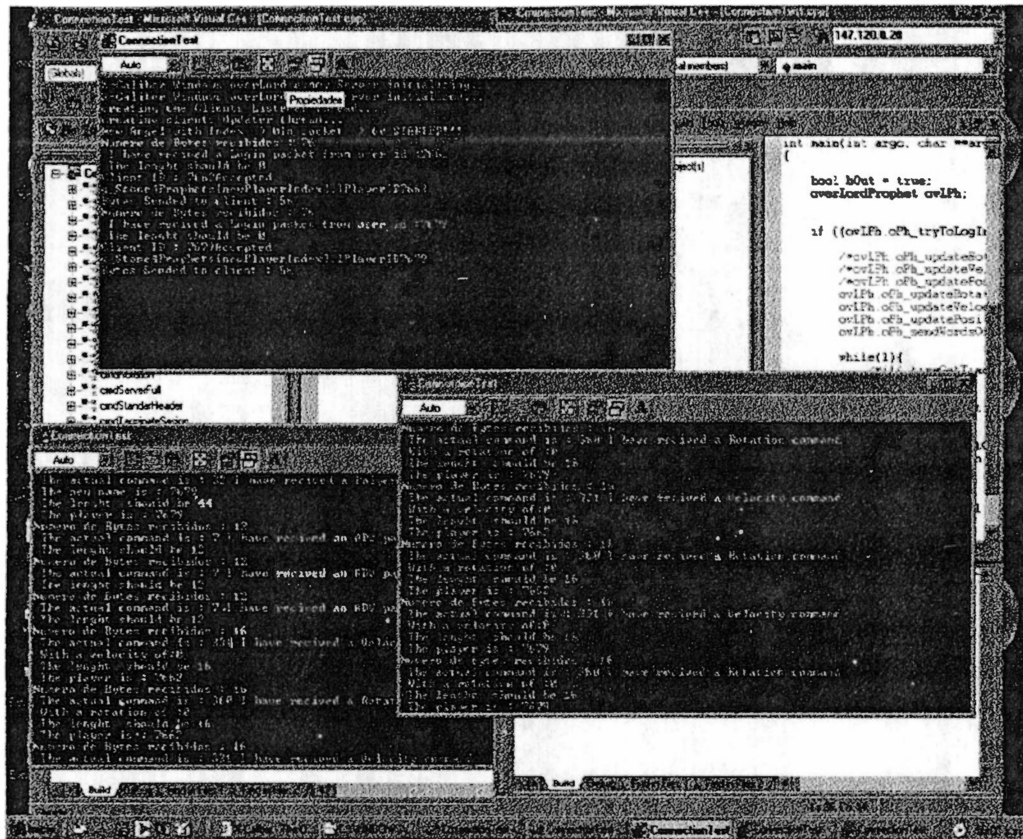


Figura 4.2. Servidor "dummy – overLord" interactuando con dos clientes a la vez.

4.3. Como ejecutar del código del CD-ROM la demostración del código de la Tercera Etapa.

Las siguientes instrucciones toman en cuenta que su unidad CD-ROM es la unidad D.

Para poder ejecutar un cliente y un servidor simplemente deberá correr los dos programas siguientes del CD-ROM anexo al reporte:

Esto ejecutaría al servidor OverLord dummy:
D:\Tercera–Etapa\X-Caliber\DummyServer\Release\ConnectionTest.exe

Y esto ejecutaría a un cliente:
D:\Tercera–Etapa\X-Caliber\OverLordProphet\Debug\ConnectionTest.exe

Sin embargo esto no proporciona mucha información de lo que acontece con el "engine".

Lo ideal es copiar las siguientes carpetas a un su Disco duro local para posteriormente cambiar los atributos de solo lectura a sus todos sus elementos.

D:\Tercera–Etapa\X-Caliber\DummyServer

D:\Tercera–Etapa\X-Caliber\OverLordProphet

Luego con su Visual C++ 6.0 podrá explorar ambos sistemas tanto el servidor como el cliente, de tal manera que puada realizar diversas pruebas y determinar el funcionamiento del API del cliente.

El nombre del archivo proyecto para visual C++ de ambos proyectos se llama ConnectionTest.dsw.

4.4. Logros según el "Milestone Gaheris" para la Tercera Etapa:

Si recordamos de la primera etapa nos quedaba pendientes los siguientes puntos:

- Interactuar con el servidor para permitir que los usuarios se puedan mover en un mundo. **"Se logro"** aunque no existe un mundo 3D, el API es capaz ahora de incorporarse a un "engine de gráficos" y dar soporte necesario para la interacción de redes. Lo anterior con ayuda del servidor dummy OverLord.
- Manejar envió de mensajes de tipo Chat entre los jugadores. **"Se logro"**, el API del servidor y el OverLord dummy server poseen la funcionalidad necesaria para implementar este punto.

4.5. Conclusiones de la etapa.

Existirá una cuarta etapa de desarrollo que no se ha documentado aquí, debido a que, es en la que me encuentro trabajando en este momento, básicamente esta parte final del Milestone Gaheris pretende utilizar el "engine Nebula" de Random Labs para hacer un prototipo con despliegue grafico de lo que ocurre internamente en el "engine X-Caliber".

La siguiente es una imagen de cómo podría verse el proceso de ingreso a los servidores con un "engine grafico":

Figura 4.3. Proceso de ingreso al sistema X-Caliber desde un ambiente grafico.

Esta también es una imagen conceptual de lo que seria el sistema cliente, interactuado en un ambiente tridimensional con el servidor overLord.

Figura 4.4. Proceso de interacción del servidor y el cliente overLord ahora con formato grafico.

Futuro:

Una vez terminado el "Milestone Gaheris", mi intención es dejar delegado a alguien mas el concepto cliente - servidor, y entonces concentrarme en un "engine de gráficos" para juegos hecho con OpenGL o Direct3D que se capaz de entenderse en forma muy transparente con lo hasta ahora logrado con X-Caliber.

Apéndice A.
Esta es la definición oficial del MileStone Gaheris:

The MileStone Goal

This milestone will be concerned with providing the basic services of a minimal server framework. Once this milestone is reached, a simple client application should be able to be written that allows a user to log into the system, move around, and chat with other users. There will be no characters or NPC's.

Queue Server

The Queue server should be fully functional and allow the various servers to transfer clients between themselves.

Benefits

- Server does not need to know how many or the address of the server that it is handing-off to.
- Each server type can have any number of instances to provide load balancing.
- Client initiates all connections which eliminates most firewall issues.

Procedure

1. Current Server connects to the Queue Server.
2. Queue Server is informed of the Client's IP address and the Destination Server type.
3. Queue Server returns a session ID to the Current Server.
4. Current Server disconnects from the Queue Server.
5. Current Server relays the Queue Server's address/port and the session ID to the Client.
6. Client disconnects from Current Server.
7. Client connects to the Queue Server.
8. Client submits the session ID and Queue Server verifies that the ID is valid and that the Client's IP matches

the one provided by the Current Server.

9. Destination Server connects to the Queue Server and checks for a pending Client.

10. Queue Server informs Destination Server of Client's IP.

11. Destination Server returns session ID to the Queue Server.

12. Queue Server returns Destination Server's IP/port and session ID to the Client.

13. Client disconnects from Queue Server.

14. Client connects to Destination Server.

15. Client submits the session ID and Destination Server verifies that the ID is valid and that the Client's IP matches the one provided by the Queue Server.

Gatekeeper Server

The Gatekeeper server should be able to:

- Accept client connections.
- Transfer client to Lobby Server using the Queue Server.

Lobby Server

The Lobby server should be able to:

- Poll the Queue server for new clients.
- Transfer clients to an Overlord server.

Overlord Server

The Overlord server should be able to:

- Poll the Queue server for new clients.
- Track character movements through the world.
- Enable characters to chat if they are in range of each other.

WorldGraph Repository

The WorldGraph server should be able to:

- Check out a map section to an Overlord server.

Client API

The Client API should be able to:

- Connect to the Gatekeeper server.
- Handle the server hand-off described HERE.
- Interact with the Overlord server to permit users to move around in the world.
- Handle chatting with other users.

Apéndice B.
Diseño global del Sistema X-Caliber.

X-Caliber Server Design Document

Introduction

The X-Caliber Server is a client-server architecture that collects information from thousands of clients, processes the game world, and sends each client an update specific to their view of the world. The idea is to make the server as data driven as possible to allow the server to handle different types of games, anything from fantasy to a sci-fi world and allow the GM to update the universe rules and change things quickly (and hopefully on the fly). The server code will be broken down so that it may be run on several machines, or on just one without even knowing the difference.

Resources

Several machines with fast internet access to act as servers, several machines for repositories, should be fairly local to servers.

Server Overview

Gatekeeper Login Server

This will process the initial login request, validate the user, and send them to either the patch server or the lobby server. This server should probably have the Patch server as part of it due to it's lighter work load with just the logins.

Lobby Server

The Lobby is where the client creates or selects the character that he wants to play. They are also able to chat with other players while they wait for any patches to download and install in the background. They can also view the news bulletin boards here while they wait for any patches to download. The lobby server sends the client information about the master Overlord server to get the client in the game. From there the Overlord servers will dynamically balance out the load between the available Overlords.

Patch Server

The Patch server is responsible for checking and updating the client with the most current version of the game world. The Server is responsible for managing which clients need which information nd sending it to them.

Puppeteer NPC Server

Puppeteer NPC Server - This will handle all of the AI for NBC's and control their movement, interaction, etc. As far as the Overlord Area Server is concerned NBC's are no different than player characters. This may become a server farm if there is enough NPC or the AI becomes complex.

Overlord Area Sever

This is the actual game server. There can be many of these active at any time. Basically instances of this server will be started and assigned a piece of the world to be responsible for. That way as load increases we can simply increase the number of instances of the server by decreasing the amount of the world each watches over. The Master Overlord server dynamically distributes which clients and NPC the other Overlords watch.

Repository Overview

WorldGraph Map Repository

This is the repository for the world map. Multiple Overlord servers can be active at any given time and each would be responsible for a particular area of the world. At startup the server will checkout its piece of the map from the repository. WorldGraph will probably be an intelligent fronted to a PostgreSQL database.

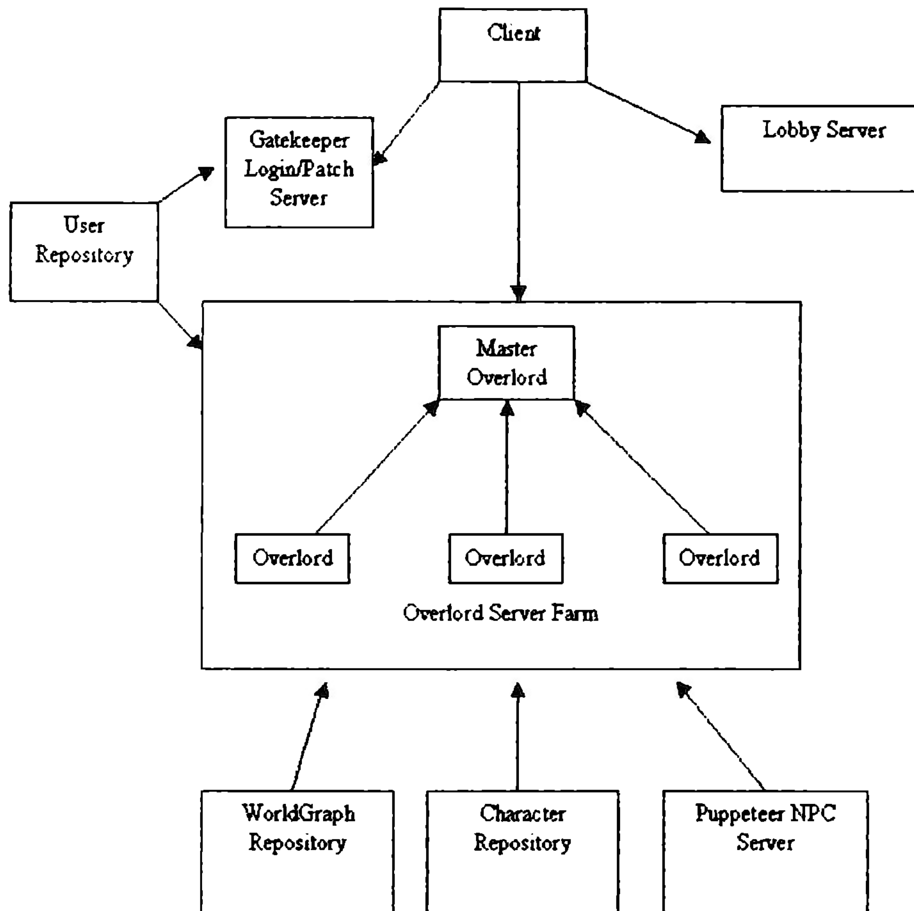
User Account Repository

This will hold user(not character) info.

Character Repository

This will hold character info.

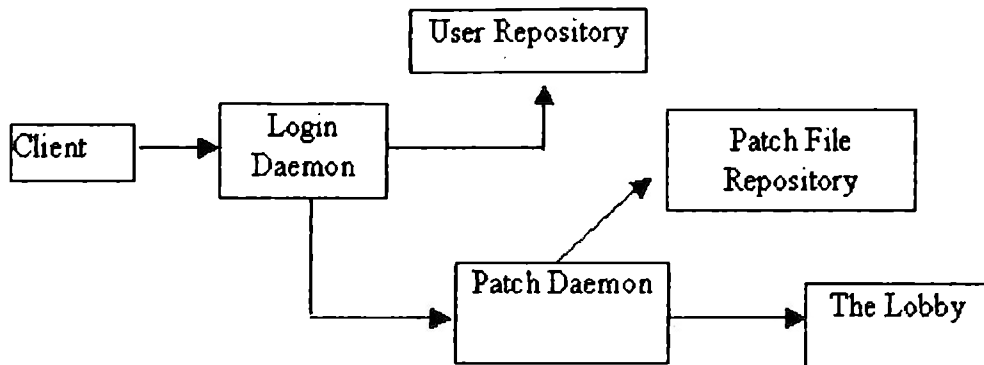
General X-Caliber Diagram



Gatekeeper Login Server

The Login server is made up of three parts: the login daemon, the User repository and the patch daemon.

General Gatekeeper Login Server Overview



Login Daemon

The login daemon listens on a given port for the client to contact it. The daemon has 4 functions: create new accounts, verify current accounts, modify accounts, and email out forgotten passwords. The Login Daemon should communicate via TAP to ensure that the client and the server receive all packets and also because speed is not essential.

Creating a New Account

The client sends a ?request? message for a new username. The client also sends the requested username and an E-mail address. The daemon then checks the user repository for the username. If it is not available then the client is sent a ?Username already being used? message. If it is available then the client is sent a ?Username accepted? message. A username/password message is also sent to the user?s E-mail account. Once the user gets the E-mail, he can login. The User can change this password in the modify account area, once the user is logged in.

Verifying a Current Account

If the client already has an account then the client sends the daemon its username and password. If the username and password are valid, the client information is then handed over to the patch daemon to get started; otherwise it receives a ?user not found? or ?password incorrect? error. The client can then try again to log in. Once logged in, the Client will then be in the account maintenance area.

Account Maintenance Section

The client will receive patch status updates during this time (as well as in the lobby) in the background. The client has the option to change the password, delete the account, log off, or enter the lobby. Changing the password requires the password and a new password, typed twice to verify. The user information is also sent to the users by E-mail. To delete the account, the client must enter the password and then verify a second time that they wish to delete the account. This will remove the account from the User Repository and any character

information from the Character Repository. The user may not enter the Lobby after deleting the account and any Patch updated that were being transferred will stop. If the client chooses to log off any patch update that is being downloaded is stopped. The user may also choose to enter the Lobby. Once in the Lobby the user information is forwarded to the Lobby Server.

User Repository

The user repository stores the data for each user. It stores:

User Login Name: String
User Password: String
User E-mail address: String
User Identification Number: Integer
10 Character identifiers: array of characters identifiers in character repository

Patch Daemon

When the Patch daemon receives the user information from the login server, when the client successfully logs in, it sends an ?ask? message to the client, which will respond with the client?s update number. This number is compared to the current update number that is on the patch server. The update number is incremented every time the world repository changes. The patch server can reset the client a number if it needs to by sending it a ?reset message.? If there is no difference between update numbers the client has the most current world and can join the game with no consequences and is sent the ?ready to join game? message. If the update numbers are different, then the patch server needs to search the update log, which is a historical log of every update to the world. The update log keeps track of which files are changed or added and associates them with a unique update number. The patch server constructs a ?patch? by placing all the files that need to be updated and a script to places the files in the correct place in a .zip file. This file is then sent to the client as a background thread. After the client has received the file, it is unpacked and installed. The client then makes a list of all the game files and their dates in a text file, zips it, and sends this list to the patch server. The patch server then compares this file to the current file list it has. If the two lists do not match, then the patch daemon will repeat the process with the mismatched files. If the two lists match, the patch server then sends the client a ?set update number? message to set the client?s update number to the current number and the client is patched. The patch server then sends the client a ?ready to join game? message and they may leave the lobby. The client may not continue past the Lobby until this file is done downloading and has been installed, but may enter the chat rooms and the news forum. The download status is shown in a window so that the client knows how much longer it will take.

Patch File Repository

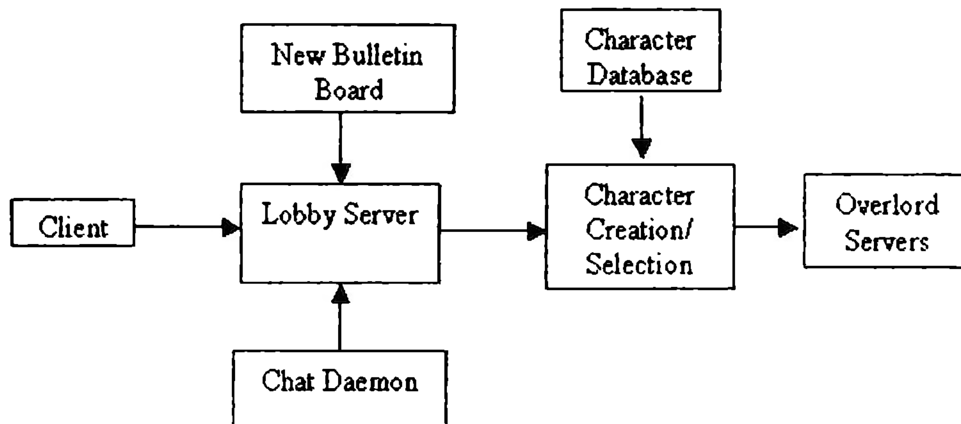
The Patch File Repository is a log file of which files have changed on which dates

and associates each change with a unique identifier number.

Lobby Server

The Lobby server is made up of several parts: a Chat daemon, the News Bulletin Board, and the character selection/creation server.

General Lobby Server Diagram



Chat Daemon

When the client selects the chat room, the client?s data is sent to the chat daemon. The chat daemon adds the client?s address to the list of clients that will receive. The chat daemon listens on a certain port for incoming data. Every time a client sends the daemon a ?text? message, the daemon sends all the clients the same ?text? message. There will be a number of chat rooms that can be created on the fly by the individual users. Each chat room is given a unique identifier. To determine which messages appear, the identifier is sent as part of the ?text? message. The daemon will only send the clients the messages that apply to the room that they're in.

News Bulletin boards

The News bulletin boards are accessed through the Lobby so that the clients can leave messages for others and read important updates put out by the GM?s. [implementation needed]

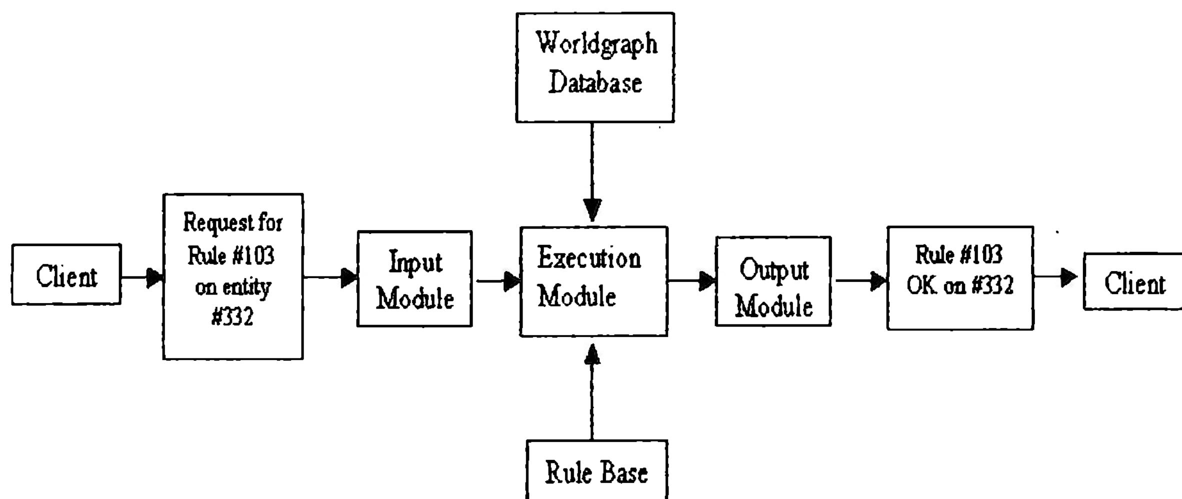
Character Selection/Creation

The Character selection module gets from the user database [implementation needed]

Overlord Servers

The Overlord server rules the game universe. The Overlord server needs to ensure synchronization between the all the clients and the game world. The Overlord consists of several servers working together to manage all the objects in the world. They are each made up of several modules: the input module, the rule base, the execution module, and the output module. It has access to the character and the world repositories. The overlord server receives client updates from the user clients and from the Puppeteer Server for the AI entities. The Overlord server dynamically balances the load of each server between themselves to be able to run the world more efficiently. The Overlord server consists of several modules that perform all it?s functions. They are the Master module, Load Balancing Module, Input module, Rule Base module, Execution Module, and output module.

General Overlord Server Overview



Dynamic Load Balancing

Each Overlord server would have to keep a dynamic list of the other overlord servers so that they can pass entities (players/objects/NPC?s) to each other to manage as the entity moves through the world. The Master Overlord server is the first overlord server to come up on the game world. It has the additional responsibilities of managing the other Overlord servers if there is only one. Any Overlord server can perform these responsibilities and will in the event the master server goes off-line. The responsibilities includes checking to ensure that the other overlords are still up, and updating the Overlords? dynamic list of each other and their load, which will facilitate each Overlord being able to dynamically pass the clients to each other (need to think of a way to prevent thrashing-maybe a ?time on server? criteria)

Rule Base

The rule base is where the rules of the universe are stored. They are referenced by the execution module to determine the outcome of the world events. Each rule is given a unique number. The client askes to execute a certain rule, the server validates each rule, and replys to the client with whether or not the client can execute that rule. Certain rules will require arguments that need to be passed in from the client, such as a target.

WorldGraph Repository

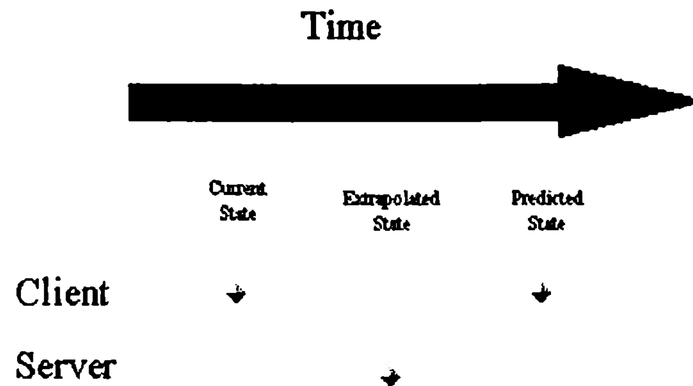
The WorldGraph Repository stores all the data in the world, to include the actual world itself. Each object in the world has an entry into the database, and is given a unique identifer.

Execution Module

The Execution Module is the heart of the server. It is responsible for updating the Worldgraph repository and the character Repository according the rules that are defined for the game universe by the game base. The execution module looks up each event in the rule base and follows the rules that it finds there for each type of event. In order to get rid of the illusion of lag, should use prediction/extrapolation and synchronized random number tables.

Extrapolation/Prediction

Certain Messages, especially movement ones, should use an extrapolation/prediction scheme to help overcome lag. By extrapolating the upcoming position of a character using the current position and velocity and sending the extrapolated data to the server, it should reache the server the same point in time when the data is true, then the server verifies the data and then predicts the upcoming position based on this data and verifies that. If that is ok, then the server sends back the ACK for the predicted data. The client should recieve this predicted data at the time the data would be true (see diagram).



Synchronized Random Number Tables

The client and the server should keep a large array of random numbers, which are the same on both client and server, to speed up the result of certain actions, such as registering whether or not a character lands a hit or not. The client and the server both maintain where the client is in thier list, and update to the next number after every time the list is accessed. In the event of a difference between the server and the client, the server wins and the client asks the server where he should be in the list and updates accordingly. This should allow for the client to not have to wait for the server to verify every manuever because they both "are thinking the same."

Message Design

Assumptions: The client, on average, is going to get 4k/s throughput with today?s 56k modems on todays networks
We want to have an update at least 15 times a second
TCP/UDP Header data is 6/34 bytes per message

This gives us 4096/15=273 bytes per update for somewhat real time updates.

Now we have to consider that this is a 2 way street and have to consider the server updates that have to tell the client about the whole world, whereas the client only has to tell the server about itself. The server has to tell the client about new entites coming into the clients area of perception (about a 100m radius of the player). The server should try to wrap as many messages as possible into each data gram.

Example Client Rule Request Message (13 Bytes)

Client Number 2 bytes	Rule Requested 1 Byte	Parameter 1 2 bytes	Parameter 2 2 bytes	Check Code 2 bytes	Client time 4 bytes

Example Server Response Message (8 Bytes)

Rule Requested 1 Byte	OK 1 bit	Server Time 4 bytes	Check Code 2 bytes

Example Client Response Message (7 Bytes)

Client Number 2 Bytes	Rule Executed 2 Bytes	OK 1 Bit	Check Code 2 Bytes

Example Server Update Message (12 Bytes)

Entity Number 2 Bytes	Rule to execute 2 Bytes	Parameter 1 2 Bytes	Parameter 2 2 Bytes	Server Time 4 Bytes

Example Server Message for New Entity (19 Bytes)

Entity Type 2 Bytes	Entity State 1 Byte	Entity Position 12 Bytes (x, y, z)	Server Time 4 Bytes

Total Throughput

Using these messages, on average the user can expect to handle:

1 client update (13+6 Bytes=19 Bytes)
1 sever response (8+6 Bytes=14 Bytes)
11 server entity updates and 3 new entites((10*5)+(3*19)+6=153 Bytes)
14 Client Responses ((13*7)+6=97 Bytes)

total: 19+14+153+97=283 Bytes

To get more information per update we could better design the messages to make them smaller, decrease the amount of updates per second, compressing the messages before sending them, and not acknowledge every message we get. By simply dropping the number of updates from 15 to 10 updates per second we get 409 bytes to use per update. This allows for 10 entiy updates, and 3 new enties with acknowledments per update. Because not all entites are going to change state or position every update, this may be acceptable message throughput; however, more is always better. Also, we need to keep in mind that we need to throw in a ping somewhere to keep current ping data up for prediction purposes.

Input Module

The input module is an independent thread that collects the data, checks it for errors, decompresses it, and then places it in a FIFO queue for the server to process. The input module receives all the messages. The module can receive the following messages:

From Clients:

Client execute Rule####
Client ACK
Client request status
Client Chat (handled by chat thread)

From Master Overlord Server:

Master ACK
Master take control of entity ####
Master New Overlord in pool

From other Overlord Server:

Overlord ACK
Overlord entity limit reached
Overlord entity number changed
Overlord Take entity ####
Overlord Give entity ####

Output module

The output module packages up the data and sends out all the messages to the other servers and clients. The server can send the following messages:

Server time

Server object update
Server ACK
Server chat
Server set attribute
Server change stat
Server Modify skill/talent
Notify Master Server New server in Overlord Pool
Update Overlord Server Load
Client Switch to this server
Server expect this client
Which Server to send to

Apéndice C.
Bibliografía.

Multiplayer Game Programming
Todd Barron
Prima Tech Game Development.
ISBN 0 – 7615-3298-6

UML y C++.
A practical Guide to object – Oriented Development.
Ricahrd C. Lee.
Prentice Hall.
ISBN. 0-13-619719-1

Programación en Internet.
Kris Jamsa. Ken Cope
Mc Graw Hill
ISBN: 970-10-0989-4

Game Programming Gems.
Mark DeLaura.
Charles River Media.
ISBN: 1-58450 –049-2

C++ para Linux.
Jesé liberty y David B. Hovath
SAMS and Prentice Hall
ISBN : 970 – 26- 0012-X

La biblia Red Hat Linux 6.
Arman Danesh
Anaya Multimedia
ISBN: 84 – 415 – 0987- 5

Unix System Programming
Using C++
Terrence Chan
ISBN: 1-58450 –049-2

Apéndice D.
Glosario y Referencias.

Juego Multi – usuario.
Originalmente el concepto de juegos de varios usuarios se limitaba a compartir una sola PC o terminal por no mas de 4 jugadores. Gracias a la tecnología cliente / servidor y al éxito que han tenido las versiones en - línea de juegos como Quake Arena o Unreal Tournament el concepto de juegos Multi - usuarios o en - línea se esta convirtiendo en una característica que la mayoría de los juegos modernos desea ofrecer.

Código Abierto.
Sistema por medio del cual, se desarrolla un sistema pero la distribución incluye los fuentes de la aplicación, también se le denomina así al movimiento de desarrollo de código voluntario y gratuito.

Everquest.
El sitio oficial de este juego RPG de sony es: http://everquest.station.sony.com/
Es el juego RPG con gráficos de 3D, multiusuario en línea por excelencia.

Game Engine.
Permite que un juego determinado utilice un modelo genérico para desarrollar un juego determinado en la computadora sin que los desarrolladores de dicho juego tengan que codificar de nuevo funcionalidad que otros desarrolladores de juegos han hecho previamente. Hoy día existen muchos "engines" para juegos que pueden adaptarse a la necesidad de cada producto que se desee construir. Para encontrar una buena lista de "engines" de juegos visite el sitio GDSE:
http://developer.dungeon-crawl.com/

Nota: X-CALIBER es parte de la lista de Engines para multijugadores de esta base de datos encabezando la lista con 5 estrellas (caritas felices).

Milestone.
Estos documentos definen el alcance hacia la perspectiva global de un producto de una versión determinada. Detallan distintas tareas que se reparten entre los colaboradores y una vez que se completan se puede terminar una versión de un producto.

Nebula Engine.
Este es un "engine" de juegos profesionales que es capaz de utilizarse en Windows y Linux es gratuito y sin problemas de licencias su sitio oficial es:
http://www.radonlabs.de/

NPC.-
Los Non Player Carácter son personajes con distintos niveles de Inteligencia Artificial que simulan ser adversarios humanos en los juegos RPG por computadora.

RPG.-
Role Play Game.- Es un genero de juego de mesa en el que cada participante escoge un papel a desempeñar durante el transcurso del juego,
Este genero se ha migrado con éxito a los juegos de computadora.

Sourceforge.
Sourceforge proporcional servicio gratuito de alojamiento para proyectos de código abierto. Sourceforge.net es el lugar donde la mayoría de proyectos de código abierto se realiza por medio de voluntarios. Sourceforge es una marca registrada de VA Linux Systems, Inc. http://sourceforge.net

Mi nombre de usuario en Sorceforge es: xtobal y mi pagina personal de desarrollador es: https://sourceforge.net/users/xtobal/

Ultima – Online.
Este es un juego RPG al que se accede por medio de una conexión web.
puede elegirse un a vida o role, y luego sumergirse en un mundo medieval en el que se puede interactuar con otros jugadores que representan a sus usuarios humanos. Para un definición mas extensa visite:http://www.uo.com/visitor/whatisuo.html

X-CALIBER.
El sitio web oficial de X-CALIBER es: http://www.x-caliber.org, la pagina de desarrollo oficial de X-CALIBER dentro de la comunidad SourceForge es:
http://sourceforge.net/projects/x-caliber/ en esta se encuentra el código fuente de nuestra aplicación así como los documentos de diseño.

Critical Section.-
Mecanismo que emplea la API de Windows para evitar problemas de sincronización al acceder a los datos en hilos múltiples.

Hilos múltiples.-
Tanto Windows como los diversos UNIX emplean a los hilos múltiples para permitir que una aplicación ejecute procesos en forma concurrente.