Descripción:

GLoad V 1.0 es una librería específicamente creada para generar carga transaccional respectiva a un comando u operación, sobre la estructura que el usuario necesite. Algunos ejemplos pueden ser: Generar consultas masivas sobre una base de datos, enviar mensajes concurrentes a un servidor a través de un socket, acceso a una estructura de datos de forma masiva, entre otros.

La arquitectura del framework se puede observar en la figura 1, sin embargo no se entrará en detalles sobre esta ya que este documento tiene como objetivo explicar su utilización y no su funcionamiento interno.

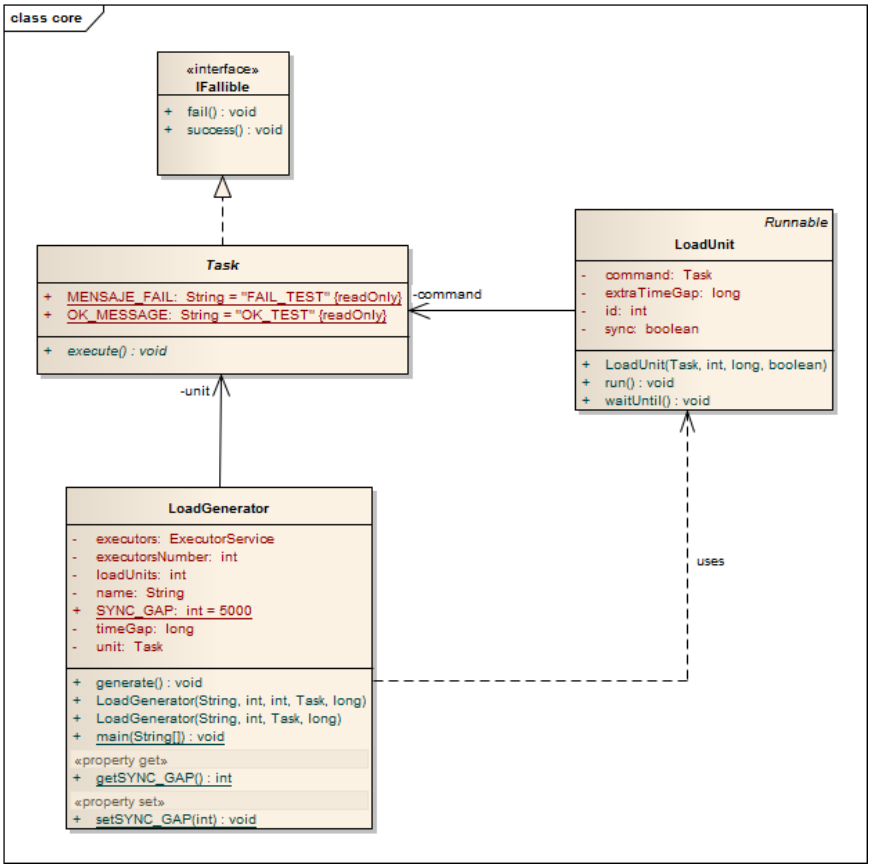

Figura 1: Arquitectura General Generador de Carga GLoad 1.0

Actualmente existen otros frameworks de generación de carga tales como JUnitPerf, el cual (como su nombre lo indica) está estrechamente ligado al framework de prueba JUnit (específicamente versión 3). Sin embargo el anterior framework tiene 3 desventajas principalmente: la primera es que está anclado a JUnit y no es fácil o natural su utilización fuera de este contexto, en segunda instancia no garantiza que al querer lanzar transacciones estrictamente simultáneas el framework lo haga, y tercero que solamente ejecuta concurrentemente un método dado de una clase de prueba, lo que es en ocasiones precario si queremos lanzar concurrentemente transacciones complejas.

GLoad permite:

- Lanzar transacciones concurrentemente no importa su tamaño, peso, o complejidad. Las transacciones pueden estar encapsuladas en una aplicación entera cuyas operaciones se quieran replicar.

- Lanzar transacciones o ejecutar operaciones de forma estrictamente simultánea. (Todas las transacciones salen al tiempo en el orden de milisegundos)

- Lanzar transacciones o ejecutar operaciones concurrentemente con un tiempo de espera entre cada una dado en milisegundos.

Utilización Básica Ejemplos:

Problema Simple Cliente – Servidor:

En este ejemplo vamos a explorar la utilización de GLoad 1.0 a través de la generación de carga de un cliente simple sobre un servidor que escucha sus peticiones a través de un socket.

En primera instancia a continuación se muestra la arquitectura del ejemplo (figura 2):

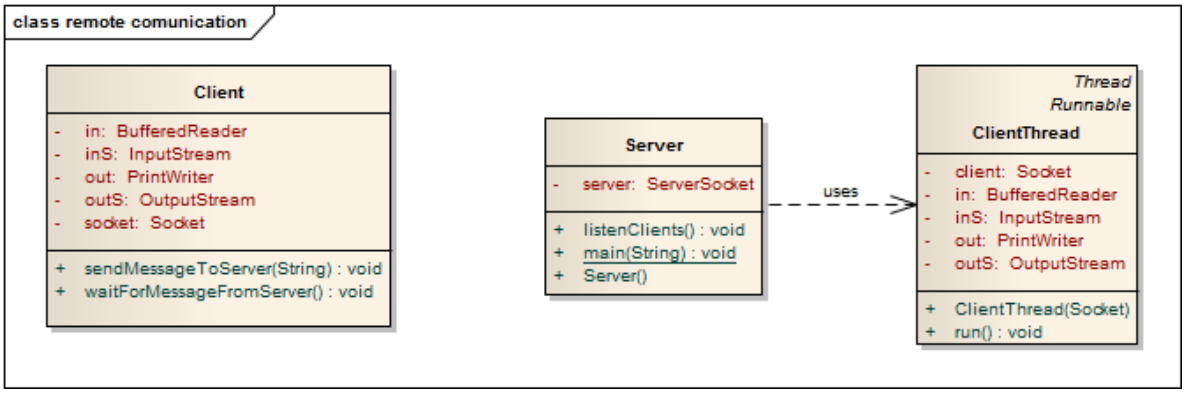

Figura 2: Ejemplo Client - Server.

En este caso tenemos un Cliente que se comunica a través de un socket con un Servidor que escucha sus peticiones. Este último cada vez que recibe una conexión de un cliente crea un nuevo ClientThread que se encarga de procesar el mensaje y responderle al cliente respectivo.

La aplicación la puede encontrar dentro de GLoad 1.0 en:

"/GLoad/source/uniandes/gload/examples/clientserver"

Generando Carga Sobre el Anterior Ejemplo:

Con base en el anterior ejemplo vamos a generar transacciones de carga sobre el servidor utilizando la estructura existente del cliente que se comunica con él.

Utilizando el Framerok GLoad 1.0 necesitamos básicamente 2 clases que apoyan la generación de carga respectiva.

- Creando el Controlador:

En primera instancia vamos a crear el controlador quien es el que crea: el LoadGenerator, la tarea o transacción que queremos que se ejecute, y configura e inicia la generación de carga.

A continuación se presenta el código que se analizará detenidamente:

```
 * GLoad Core Class - Task
public class Generator
{
    /**
     * Load Generator Service (From GLoad 1.0)
     */
    private LoadGenerator generator;

    /**
     * Constructs a new Generator
     */
    public Generator ()
    {
        Task work = createTask();
        int numberOfTasks = 100;
        int gapBetweenTasks = 1000;
        generator = new LoadGenerator("Client - Server Load Test", numberOfTasks, work, gapBetweenTasks);
        generator.generate();
    }

    /**
     * Helper that Constructs a Task
     */
    private Task createTask()
    {
        return new ClientServerTask();
    }

    /**
     * Starts the Application
     * @param args
     */
    public static void main (String ... args)
    {
        @SuppressWarnings("unused")
        Generator gen = new Generator();
    }

}
```

En general se recomienda la creación de una clase aparte de la del problema o aplicación original para que controle la generación de carga, esta se encargará de ver con perspectiva la aplicación sobre la cual queremos generar carga, en este caso la llamamos Generator, a continuación su explicación detallada.

Creando Clase Controladora Generator

1. Como primer paso se inicializa el elemento principal del GLoad 1.0 para la generación de carga (LoadGenerator), es con este el que va a controlar la generación de todas las transacciones que queramos generar.

```
/**
 * Load Generator Service (From GLoad 1.0)
 */
private LoadGenerator generator;
```

2. El siguiente paso es la construcción del constructor de la clase controladora (Generator), en esta se realiza el Set Up, de la generación de carga que queremos realizar. En este caso la idea es generar 100 transacciones de comunicación con el servidor, cada una con una separación entre sí de 1 segundo.

El primer paso es la creación de la tarea o Task que queremos que se ejecute concurrentemente, mas adelante veremos cómo se realiza su creación, por el momento sólo vamos a utilizar el método que la crea:

```
Task work = createTask();
```

Posteriormente creamos dos variables que indican: cuantas transacciones queremos que se ejecuten, y cada cuanto queremos que lo hagan (separación entre una y otra). Si hubiéramos deseado que todas las transacciones se ejecutaran simultáneamente el valor de "gapBetweenTaks" debería ser de 0.

```
int numberOfTasks = 100;
int gapBetweenTasks = 1000;
```

Por último inicializamos el generator (declarado en el paso 1) con los parámetros correspondientes:

```
generator = new LoadGenerator("Client - Server Load Test", numberOfTasks, work, gapBetweenTasks);
generator.generate();
```

Para que el generador comience a funcionar debemos llamar el método generate(), el cual inicia la producción de carga.

Adicionalmente esta clase cuenta con un método Helper el cual se encarga de instanciar el Task que queremos se ejecute concurrentemente.

Creando Clase Task Client Server Task:

Como segunda clase que debemos crear para la generación de carga encontramos la tarea que queremos que se ejecute concurrentemente.

Esta clase debe encapsular que operaciones se desea que se ejecuten iterativa o concurrentemente, para este fin usted tiene dos opciones.

1. Crear una clase que extienda de Task (clase del framework) e implemente el método execute() para que cree el objeto originador de las operaciones, y posteriormente las ejecute. (opción implementada en la figura a continuación).

2. Crear una clase que extienda de Task (clase del framework) que tenga un constructor que reciba el o los objetos que originan las operaciones, los asigne a atributos, e implemente el método execute() para que llame la operación que se desee de él o los originadores deseados. Esta última opción puede ser particularmente útil cuando se desea crear escenarios complejos.

```
 * GLoad Core Class - Task
public class ClientServerTask extends Task
{

    @Override
    public void execute()
    {
        // TODO Auto-generated method stub
        Client client = new Client();
        client.sendMessageToServer("Hi! i'm a client");
        client.waitForMessageFromServer();

    }

    @Override
    public void fail()
    {
        // TODO Auto-generated method stub
        System.out.println(Task.MENSAJE_FAIL);

    }

    @Override
    public void success()
    {
        // TODO Auto-generated method stub
        System.out.println(Task.OK_MESSAGE);

    }

}
```

En este caso solamente deseamos que se creen diferentes clientes que envíen un mensaje al servidor y esperen por su respuesta. En el caso que dada la lógica de negocio queramos que sea el mismo cliente quien envíe varios mensajes deberíamos hacer un constructor que reciba el cliente y lo inicialice en un atributo de la clase. Y posteriormente implementar el método execute() para que envíe los mensajes correspondientes.

Los métodos fail() y succes() son utilizados en caso de que las operaciones que se generen sean susceptibles a excepciones, caso en el cual (por mecanismos de control) se aconseja llamarlos respectivamente para ver el avance de la generación en el log de consola. En este ejemplo básico no los utilizaremos.

Como se puede observar la implementación del método execute() para este ejemplo es: la inicialización de un nuevo cliente, luego el envío de un mensaje, y posterior espera de respuesta.

Corriendo el Ejemplo:

- Para ver el funcionamiento básico del ejemplo (SIN generación de carga) usted debe:

1. Ejecutar la clase server:

"/GLoad/source/uniandes/gload/examples/clientserver/Server.java"

De este modo el servidor queda inicializado y esperando solicitudes:

```
Server [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (26/04/2010 12:07:56)
Server Running ...
```

2. Ejecutar la Clase

"/GLoad/source/uniandes/gload/examples/clientserver/Client.java"

De este modo el cliente enviará los mensajes predeterminados, el servidor los recibirá, y posteriormente le enviará al cliente el ACK del mensaje respectivo.

```
<terminated> Client [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (26/04/2010 12:14:39)
Client - Message: ACK
Client - Message: ACK
```

- Para ver el funcionamiento básico del ejemplo (CON generación de carga) usted debe:

1. Ejecutar la clase server:

"/GLoad/source/uniandes/gload/examples/clientserver/Server.java"

De este modo el servidor queda inicializado y esperando solicitudes:

```
Server [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (26/04/2010 12:07:56)
Server Running ...
```

2. Ejecutar la clase Generator, que generará 100 transacciones con un intervalo de 1000 milisegundos (1 segundo) entre sí.

De este modo 100 clientes diferentes enviarán los mensajes predeterminados, el servidor los recibirá, y posteriormente le enviará a cada cliente el ACK del mensaje respectivo.

En este punto podemos ver dos logs en la consola, el primero de la clase Generator, que indica el estado de la generación, de carga, y el segundo que muestra el proceder normal de las actividades del servidor bajo carga.

```
Generator [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (26/04/2010 12:20:20)
Client - Message: ACK
[LoadUnit 52] [Executed at: Mon Apr 26 12:21:12 COT 2010]
Client - Message: ACK
[LoadUnit 53] [Executed at: Mon Apr 26 12:21:13 COT 2010]
Client - Message: ACK
[LoadUnit 54] [Executed at: Mon Apr 26 12:21:14 COT 2010]
Client - Message: ACK
[LoadUnit 55] [Executed at: Mon Apr 26 12:21:15 COT 2010]
Client - Message: ACK
[LoadUnit 56] [Executed at: Mon Apr 26 12:21:16 COT 2010]
Client - Message: ACK
[LoadUnit 57] [Executed at: Mon Apr 26 12:21:17 COT 2010]
Client - Message: ACK
[LoadUnit 58] [Executed at: Mon Apr 26 12:21:18 COT 2010]
Client - Message: ACK
[LoadUnit 59] [Executed at: Mon Apr 26 12:21:19 COT 2010]
Client - Message: ACK
[LoadUnit 60] [Executed at: Mon Apr 26 12:21:20 COT 2010]
Client - Message: ACK
[LoadUnit 61] [Executed at: Mon Apr 26 12:21:21 COT 2010]
Client - Message: ACK
[LoadUnit 62] [Executed at: Mon Apr 26 12:21:22 COT 2010]
Client - Message: ACK
[LoadUnit 63] [Executed at: Mon Apr 26 12:21:23 COT 2010]
```

Figura 3: Consola del Generator - Ejemplo Simple Client - Server

```
Server [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (26/04/2010 12:20:15)
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
Conection Accepted!
Message From Client Recived: Hi! i'm a client
Message From Client Recived: EOT
```

Figura 4: Consola del Server - Ejemplo Simple Client – Server (Bajo Carga)

- La anterior estructura de implementación es sugerida, usted podrá utilizar el generador a su gusto conociendo su funcionamiento básico, y sus estructuras principales LoadGenerator y Task. De igual forma usted podrá hacer uso de los elementos internos de la librería.