

Generación de Código a partir de CSSL

- Sistemas Colaborativos
- CSSL: un lenguaje de modelado de sistemas colaborativos
- Desarrollo Dirigido por Modelos
- Lenguajes Específicos de Dominio

Sistemas Colaborativos

- CSCW y Groupware
- Definiciones: Rol, Objeto compartido, Workspace, Sesión, Herramienta, Protocolo, Vista, Acoplamiento, Awareness, Sincronismo, Permisividad
- Componentes de un Framework:
 - Arquitecturas de tiempo de ejecución
 - Abstracciones de programación
 - Widgets
 - Administradores de sesión

Frameworks

- Arquitecturas de tiempo de ejecución
 - Centralizada / Distribuida
 - Control de Concurrencia: consistencia, Esquemas optimistas, pesimistas, bloqueos, transacciones, Ejecución reversible, Transformaciones operacionales
 - Gestión de estado y sincronización, Comunicación, Tolerancia a fallos, seguridad..

Frameworks

- Abstracciones de programación:
 - Llamada a Procedimientos Remotos por Difusión
 - Eventos y notificadoros
 - Modelos Compartidos y vistas (MVC)
- Widgets
- Administración de Sesiones
 - Políticas, responsabilidades, latecoming

Desarrollo Dirigido por Modelos

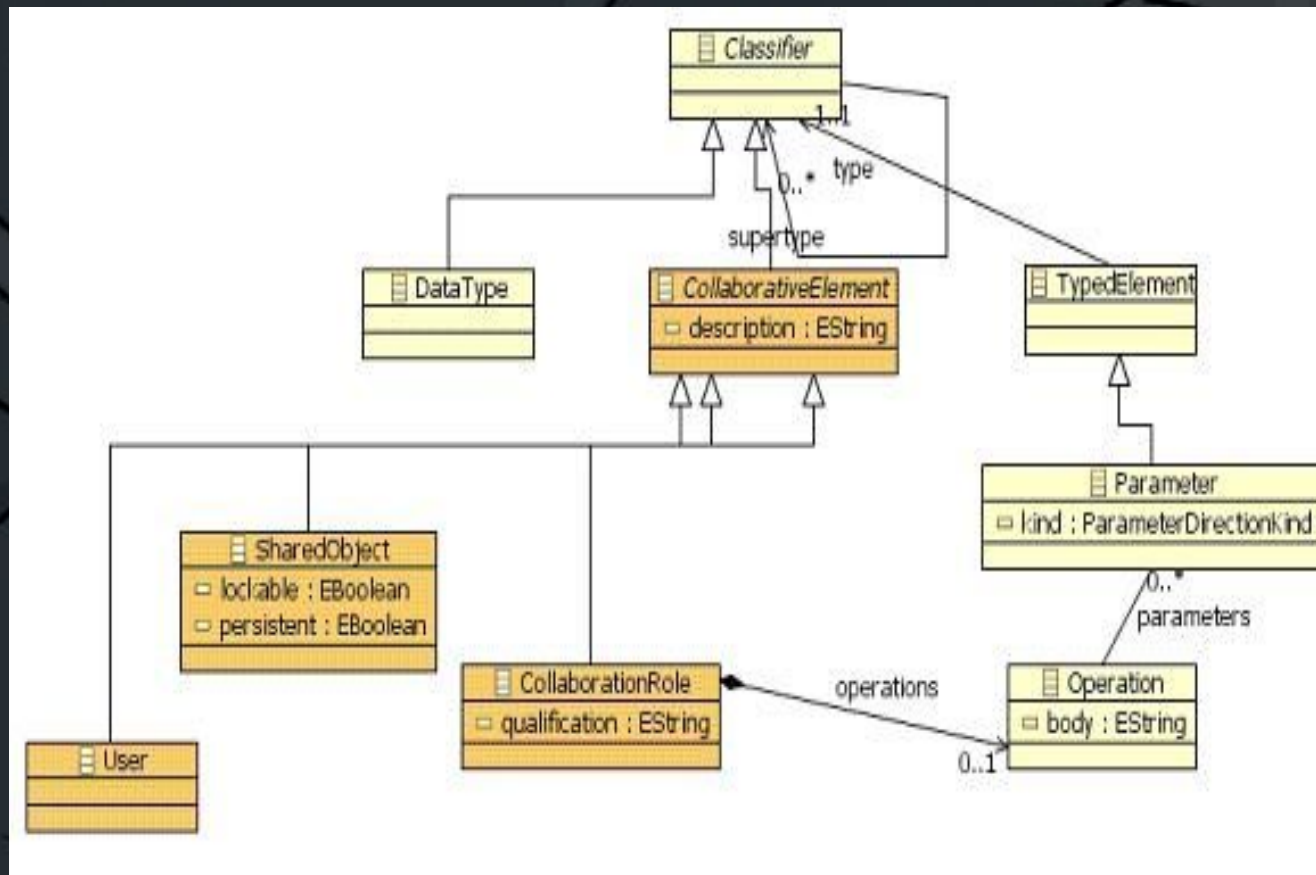
- MDA
 - PIM, PSM, Transformaciones
 - M0, M1, M2, M3
- DSM
 - Definición de un DSL
 - Nivel de Abstracción
 - Reducir el dominio
 - Generación Total
 - Arquitectura de DSM

CSSL

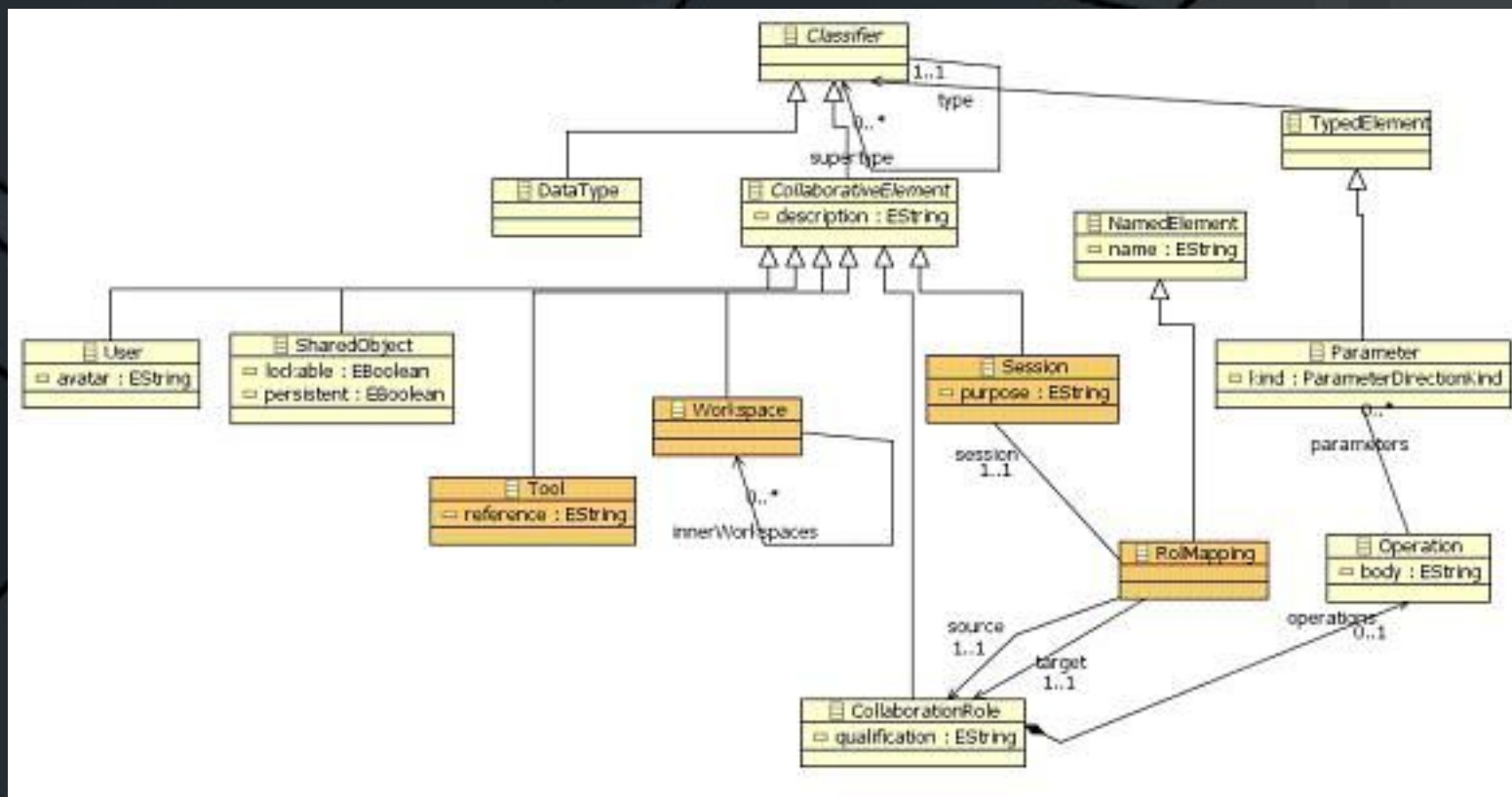
Collaborative Software Systems Language

- Kernel
- Workspaces
- Protocols

Kernel



Workspaces

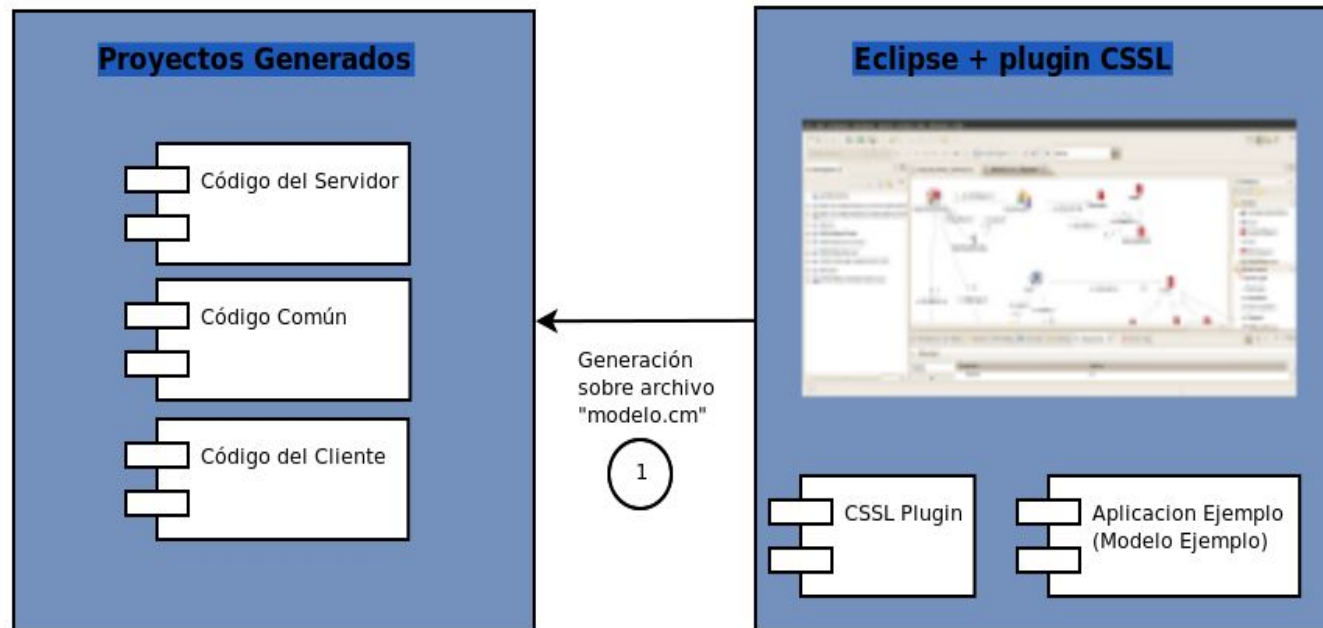


Presentación del plugin

- Qué hace el plugin
- Cómo probar que el código generado está funcionando
- Cómo está escrito el plugin

Uso y Test del Plugin

USO DEL PLUGIN



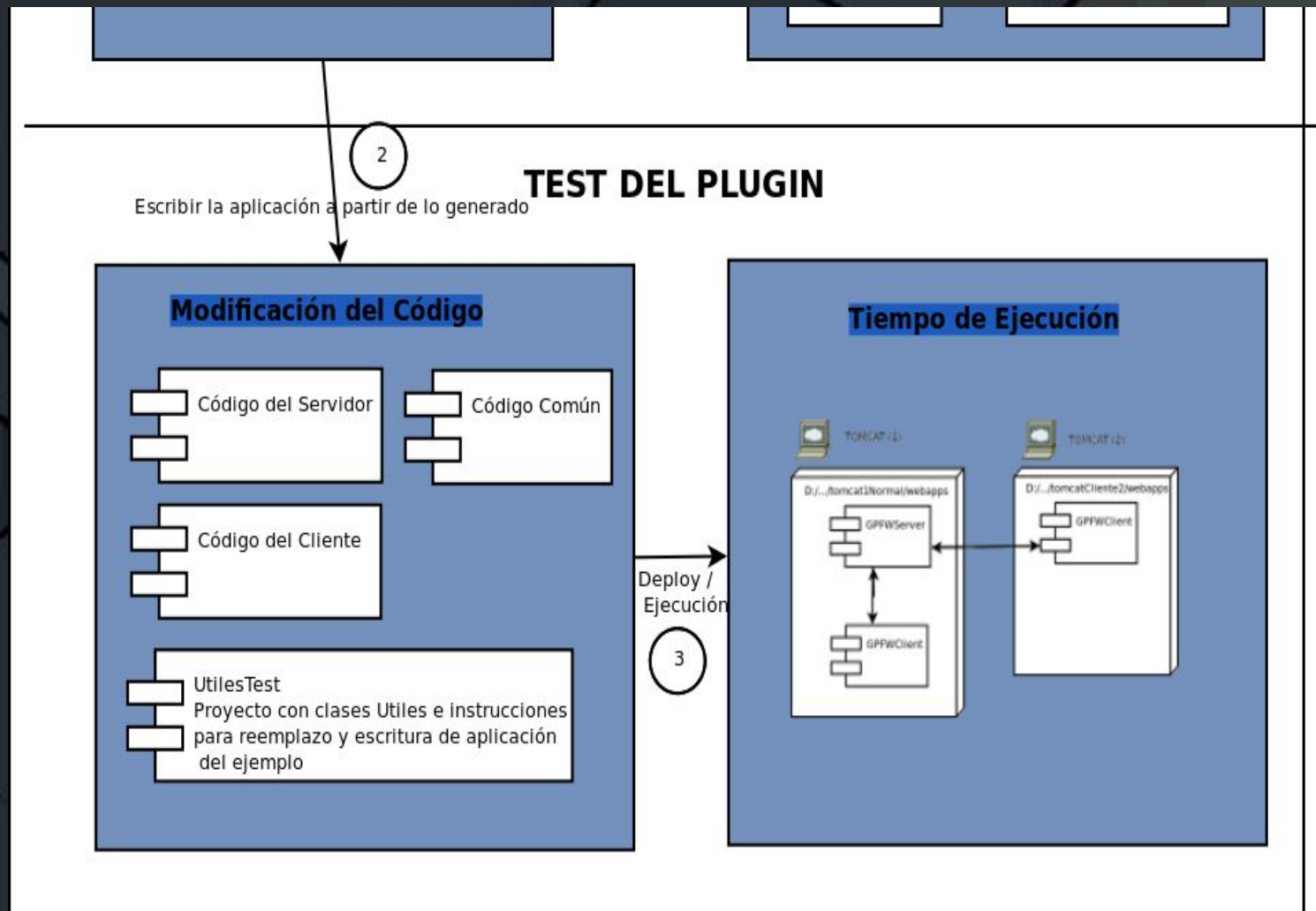
TEST DEL PLUGIN

Escribir la aplicación a partir de lo generado

Modificación del Código

Tiempo de Ejecución

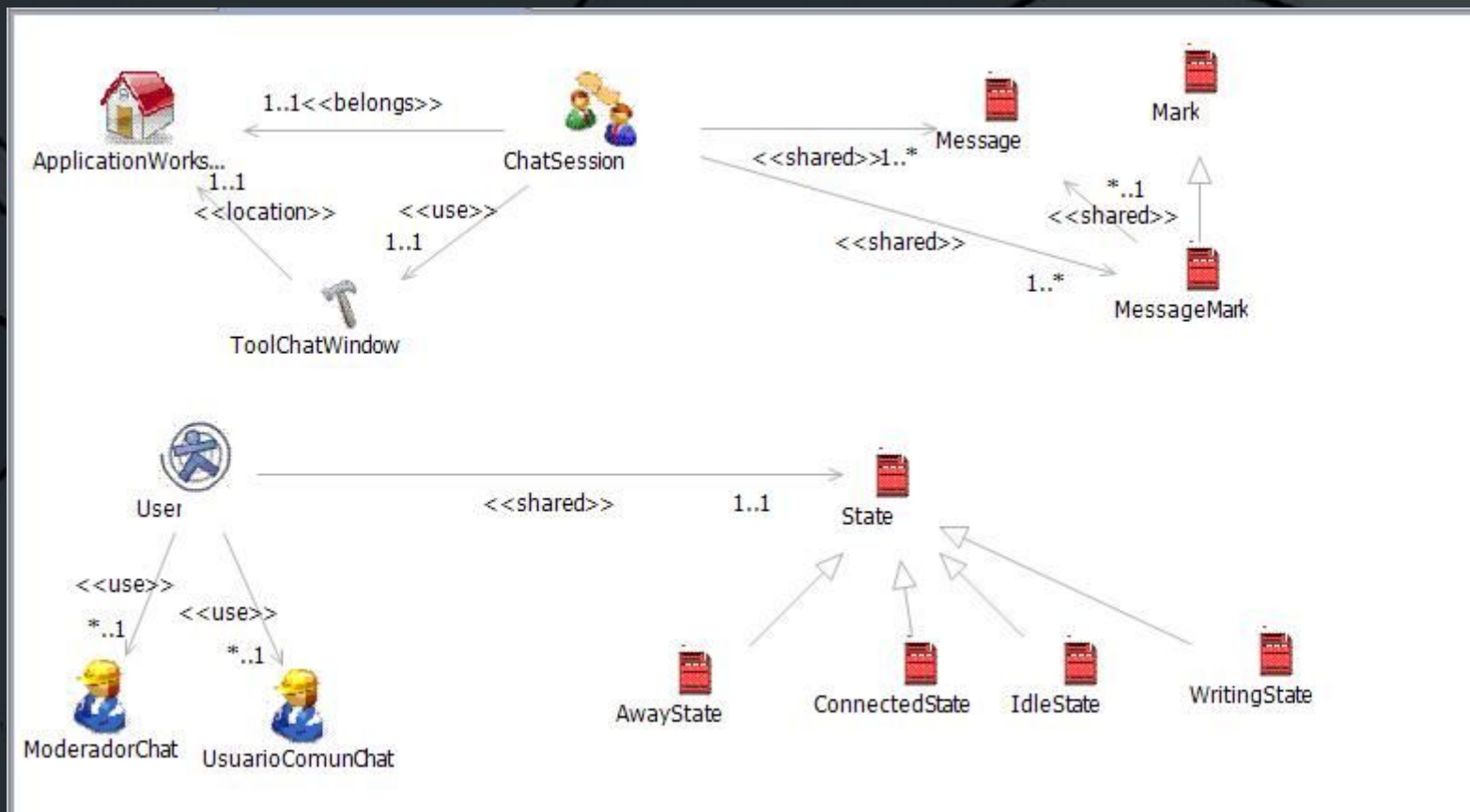
Uso y Test del Plugin



Qué hace el plugin

- Ejemplo de generación de código
 - Introducción del modelo de ejemplo
 - Operación de ejemplo
- Ejemplo de regeneración
- Descripción del código resultante
- Descripción del funcionamiento del código resultante (Arquitectura)

Modelo del Ejemplo



Nueva Operación

Definimos la operación *NewMessageOp2*

The screenshot shows an IDE interface with a tree view on the left and a properties view at the bottom. The tree view shows a package named 'Session ChatSession' containing several elements: 'Shared Relationship messages', 'Shared Relationship messageMarks', 'Use Relationship ChatSession -> ToolChatWindow', 'Belongs Relationship ChatSession -> ApplicationWorkspace', 'Operation NewMessageOp2' (highlighted), and 'Parameter mensaje'. The properties view at the bottom shows the following details for the selected operation:

Property	Value
Body	///Implementar operacion de envio de mensaje
Classifier	Session ChatSession
Name	NewMessageOp2
Parameters	Parameter mensaje

Lanzar la generación

The screenshot shows an IDE interface with a UML class diagram. A context menu is open over the diagram, listing various actions. The diagram includes classes like ChatSession, User, and PoolChatWindow, with relationships such as <<belongs>>, <<use>>, and <<shared>>. The menu options are:

- New
- Open
- Open With
- Copy
- Paste
- Delete
- Move...
- Rename...
- Import...
- Export...
- Refresh
- Initialize cm_diagram diagram file
- Run As
- Debug As
- Team
- Compare With
- Replace With
- CSSL

A button labeled "Generar codigo" is located at the bottom right of the menu.

Instrucciones_demo.txt default.cm_diagram

tests/Aplicacion

ApplicationV

01/11/09 22:31

gram 89 19/10/

91 01/11/09 22:

/04/10 00:48 pa

4/10 00:48 pabl

06 12/04/10 00:

ancia1]

Progress Information

Mark

CSSL- Generación

Proceso finalizado exitosamente

OK

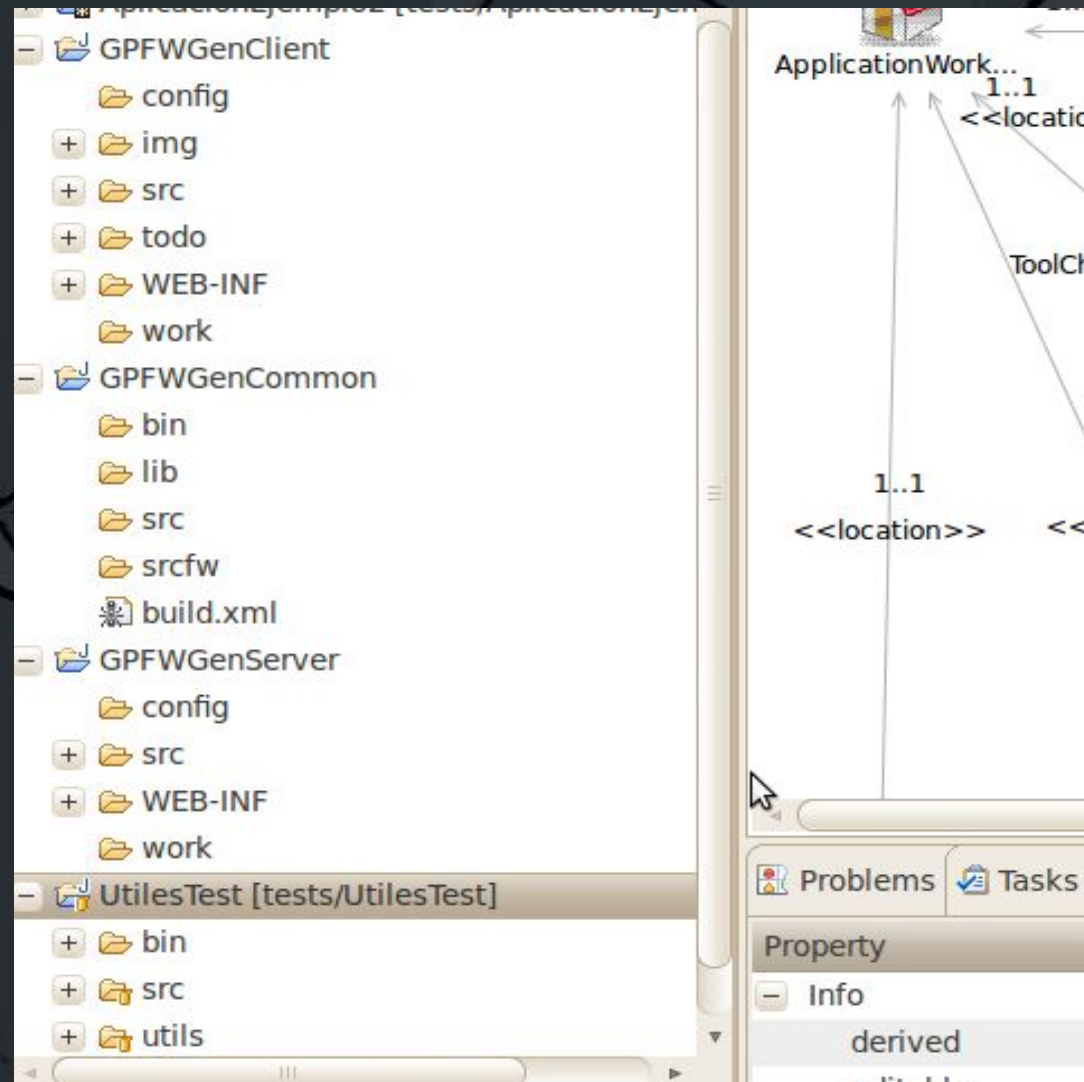
1..1 1..1 User <<shared>> 1..1 Stat

<<location>> <<belongs>>

<<use>> <<use>>

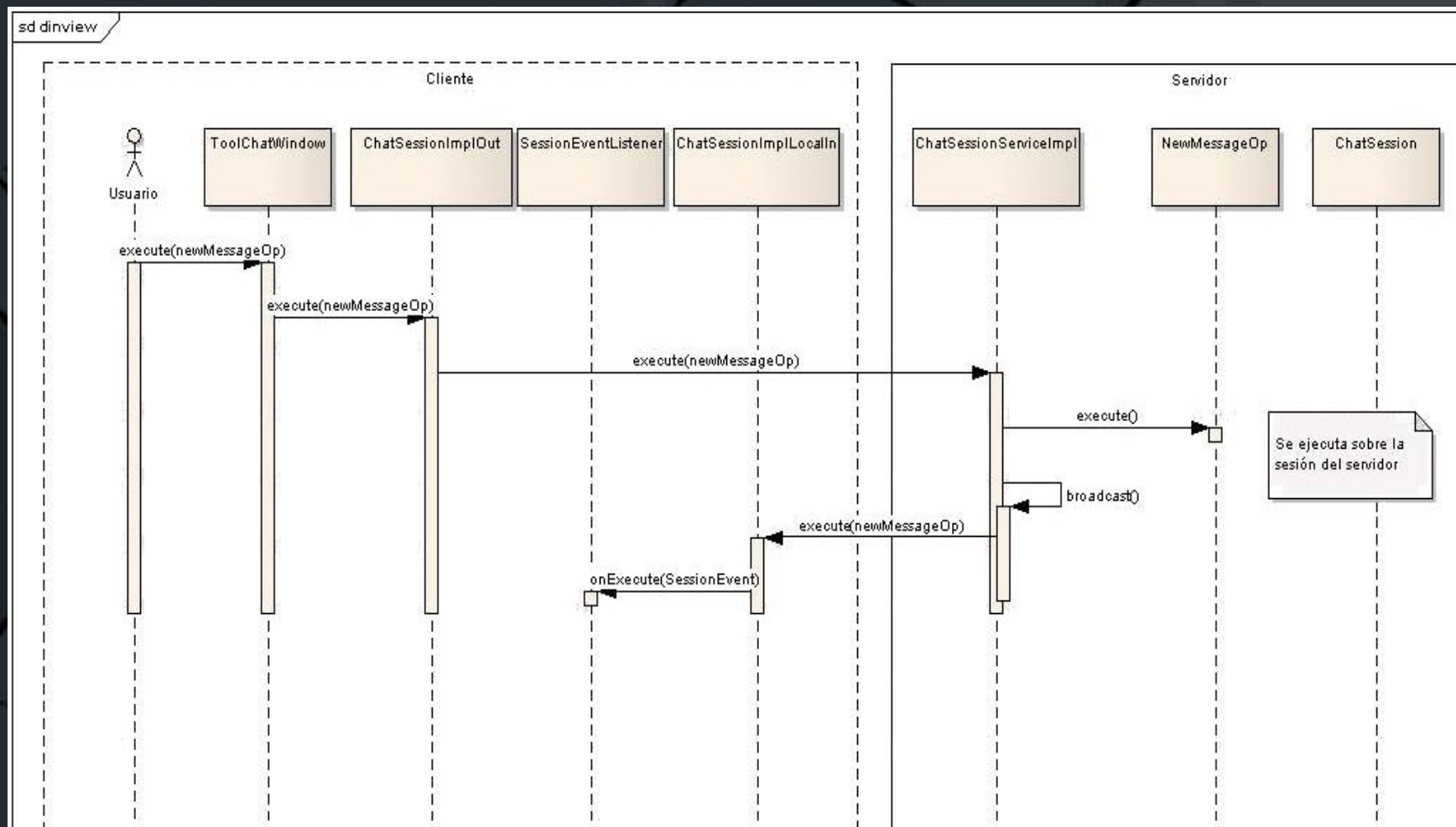
*..1 *..1

Código Generado



- **GPFWGenClient:** contiene las clases usadas exclusivamente por el cliente
- **GPFWGenServer:** contiene las clases usadas únicamente por el servidor
- **GPFWGenCommon:** contiene las clases usadas tanto por el servidor como por el cliente

Código Generado



• Clases Comunes

gen.model: entidades

gen.model.factory: enumeradores e instancias únicas para algunas entidades

gen.remoting.api: interfaces de los servicios de sesión.

gen.remoting.session: utilidades para manejar las operaciones

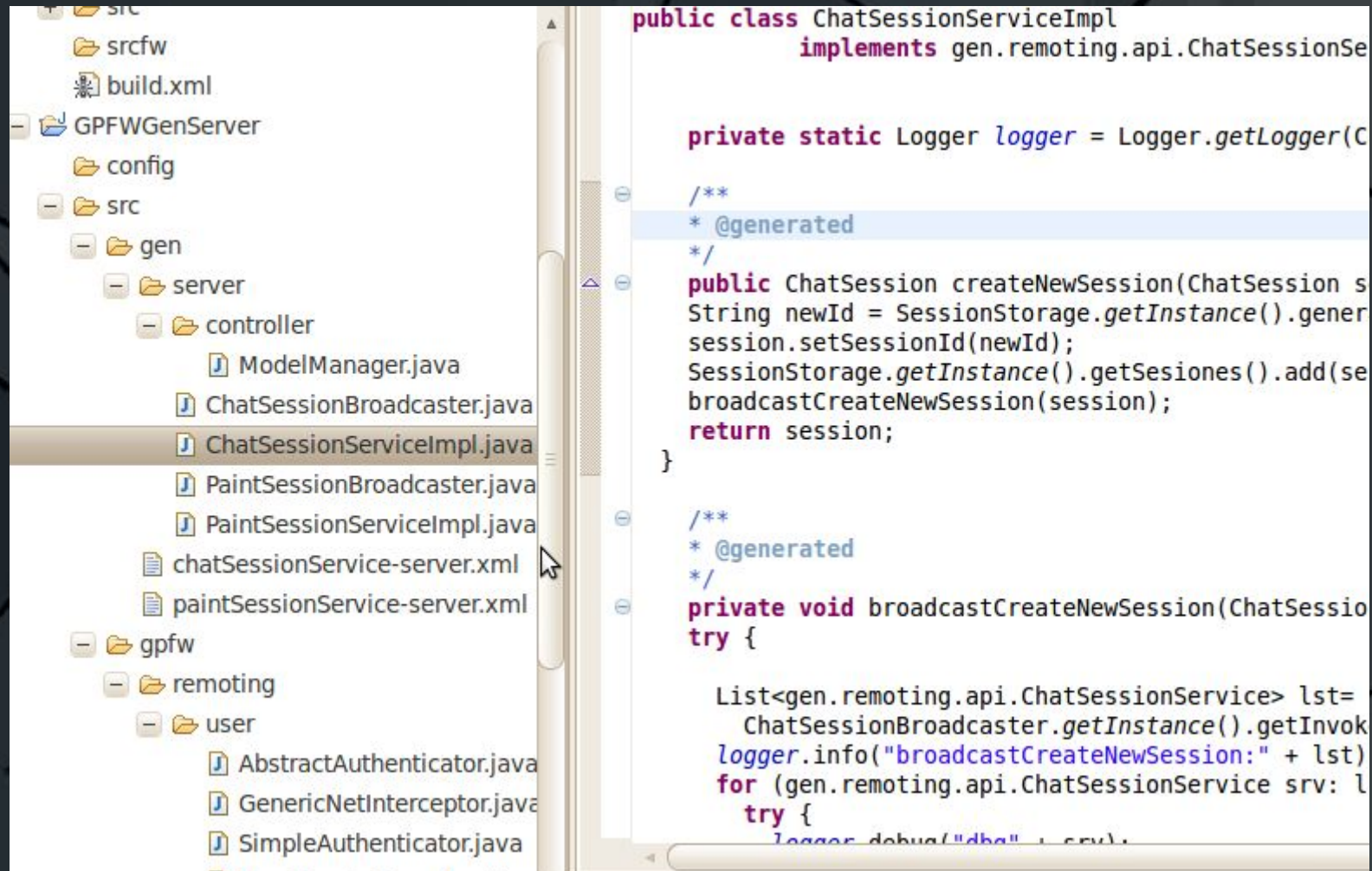
*gpfw.**: clases útiles del framework, no son generadas, no tienen relación con el modelo

gpfw.model: clases abstractas, desde donde extenderán las clases generadas

*gpfw.remoting.session.**: útiles para el manejo de los servicios de sesión y sus operaciones

gpfw.remoting.user: útiles para el manejo de usuarios. Interfaz del servicio de usuarios.

Clases del Servidor



```
public class ChatSessionServiceImpl
    implements gen.remoting.api.ChatSessionSe

    private static Logger logger = Logger.getLogger(C

    /**
     * @generated
     */
    public ChatSession createNewSession(ChatSession s
String newId = SessionStorage.getInstance().gener
session.setSessionId(newId);
SessionStorage.getInstance().getSesiones().add(se
broadcastCreateNewSession(session);
    return session;
}

    /**
     * @generated
     */
    private void broadcastCreateNewSession(ChatSessio
try {

        List<gen.remoting.api.ChatSessionService> lst=
            ChatSessionBroadcaster.getInstance().getInvok
        logger.info("broadcastCreateNewSession:" + lst)
        for (gen.remoting.api.ChatSessionService srv: l
            try {
                logger.debug("dba" + srv);
```


Ejemplo: una operación

```
package gen.remoting.session.ops;

import gen.model.Message;
....
public class NewMessageOp2 extends AbstractSessionOperation implements IOperation {
    private static final long serialVersionUID = 1L;
    /**
     * @generated
     */
    public NewMessageOp2(gen.model.Message mensaje) {
        this.mensaje = mensaje;
    }
    // inicio Parametros
    /**
     * @generated
     */
    private gen.model.Message mensaje;
    /**
     * @generated
     */
    public Message getMensaje() {
        return mensaje;
    }
}
```

Continuación..

```
.....  
/**  
 * @generated  
 */  
public void setMensaje(gen.model.Message myVar) {  
    this.mensaje = myVar;  
}  
// fin Parametros  
  
/**  
 * @generated  
 */  
public boolean execute() throws GPFWException {  
    // TODO: implementar  
    return true;  
}  
}
```


Desarrollo de la aplicación

Pasos

- **Escribir modelo y generar código** (ya hecho)
- **Escribir el cuerpo de las operaciones**
- **Escribir código que interactúa con las clases del modelo:** Por ejemplo la GUI, el módulo de autenticación, otras acciones que deba realizar el cliente al recibir una operación, el código para inicializar y enviarle a la herramienta la operación.
- **Otras modificaciones:** por ejemplo configurar gpfw.properties, limitar el broadcast de la operación
- **Build / Deploy / Test:** compilar las clases y generar los paquetes instalables (.war), instalar dos Tomcats, copiar los paquetes, ejecutar, revisar los logs de ejecución

Modificar la operación

```
package gen.remoting.session.ops;
..
import gen.model.ChatSession.ChatSessionSharedObjects;
..
public class NewMessageOp extends AbstractSessionOperation implements IOperation {
    private Message msg ;
    public NewMessageOp(Message message) {
        this.msg = message;
    }(... )
    /**
     * @generated NOT
     */
    public boolean execute() throws GPFWEException {
        ((ChatSessionSharedObjects)getSession().getSharedObjects()).getMessages().add(getMsg(
));
        return false;
    }
}
```

Interactuando con las clases generadas

```
package util.test;
```

```
.....
```

```
public class TestUtilUser2 implements SessionEventListener<ChatSession>{
```

```
.....
```

```
    private ChatSessionService getChatSessionService() {  
        return (ChatSessionService) SpringUtil.getApplicationContext().getBean("chatSessionImplOut");  
    }
```

```
    public void loginUsuario2(){  
        SessionEventManager.getInstance().getListeners().add(this);
```

```
        .....
```

```
        this.getUserService().login(user);  
        UserManager.getInstance().setUserLoggedIn(user);  
        boolean f = this.verSiHayOtroYUnirse();
```

```
        ....
```

```
    }
```

```

public void envioMensajeUsuario2(){
    this.tool = (ToolChatWindow) session.getToolByClassId
( ToolFactory.TOOL_CHAT_WINDOW);
    AbstractUser user = UserManager.getInstance().getUserLoggedIn();
    Message msg = new Message();
    ....
    msg.setUser((User)user);
    NewMessageOp op = new NewMessageOp(msg);
    logger.info("envio mensaje de Usuario 2. execute");
    this.tool.execute(op);
}
.....
public void onExecute (SessionEvent<ChatSession> e) throws GPFWEException {
    AbstractSessionOperation opSession = ((AbstractSessionOperation)e.getOperation());
    if (!opSession.getSessionId().equals( this.session.getSessionId()))
        return;
    if (e.getOperation() instanceof NewMessageOp) {
        NewMessageOp op = (NewMessageOp) e.getOperation();
        logger.info("MOSTRAR en pantalla NewMessageOp:" + op.getMsg().getMessage());
        //responder
        if (op.getMsg().getMessage().startsWith(comienzoMensajePropio )){
            threadPool.execute(new Responder());
        }
    }
}
}
}

```

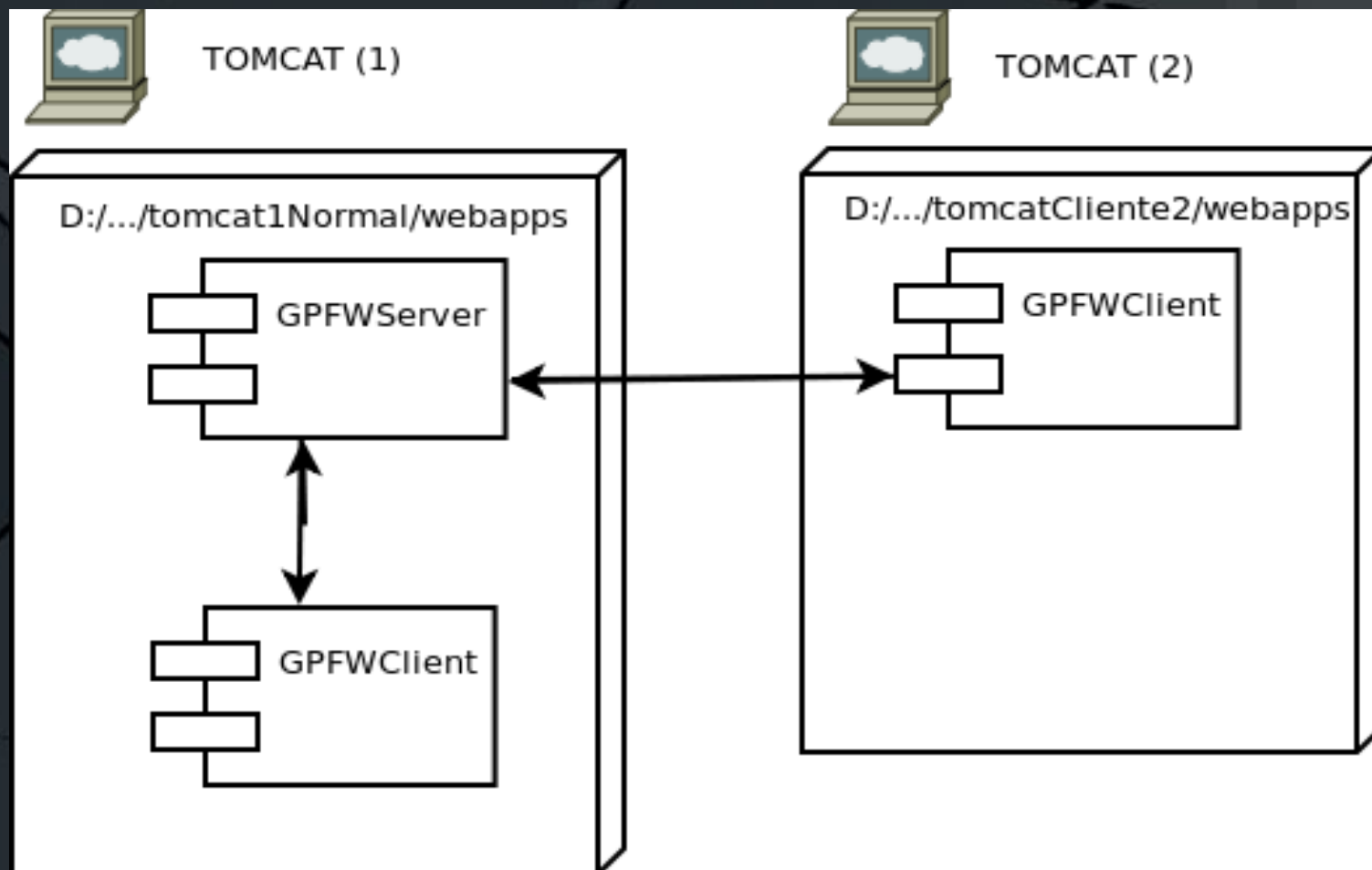
Probando la aplicación

Configurar el entorno

Estas tareas se realizan a mano:

- Compilar el código
- Cambiar `gpfw.properties` (cada cliente debe apuntar hacia el servidor)
- Instalar dos tomcats
- Copiar la aplicación compilada a cada tomcat (cliente1, cliente2, server)
- Ejemplo: ver `UtilesTest/build.xml`

Arquitectura del Test



Fin del Test

- Levantar ambos tomcats
- Revisar los logs de ejecución: allí se puede ver como se enviaron y recibieron los mensajes

Modificar el modelo

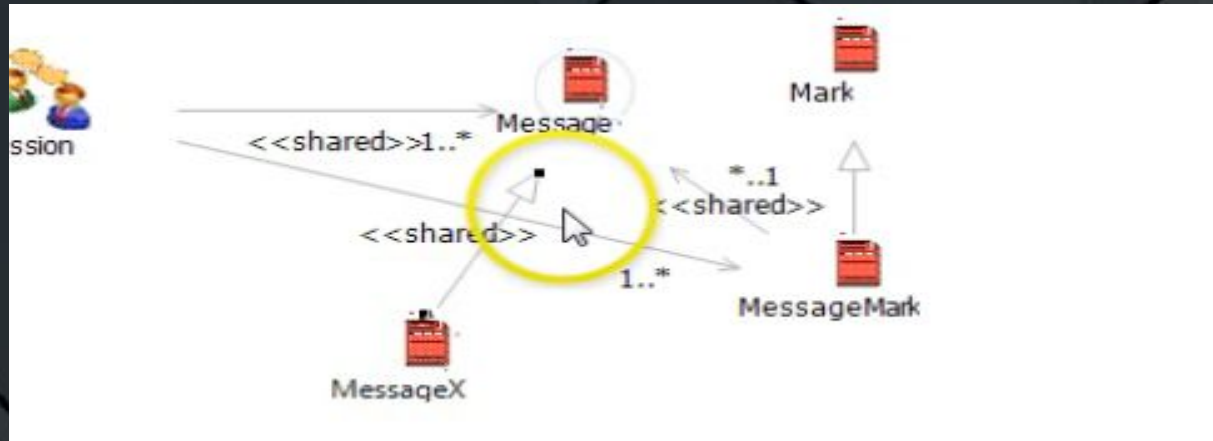
```
    } // finParametro

    // finParametros

    /**
     * @generated NOT
     */
    public boolean execute() throws GPFWException {
        ((ChatSessionSharedObjects)getSession().getSharedObjects()).getMessages().add(this.getMensaje);
        return false;
    }
}

Console History Properties Error Log
```

```
/**
 * @generated
 */
private void broadcastCreateNewSession(ChatSession session) {
    try {
        //Como modifico el @generated este comentario se eliminara en la regeneracion
        List<gen.remoting.api.ChatSessionService> lst=
            ChatSessionBroadcaster.getInstance().getInvokersAll(gen.remoting.api.ChatSessionService);
        logger.info("broadcastCreateNewSession:" + lst);
        for (gen.remoting.api.ChatSessionService srv: lst){
            try {
                logger.debug("dbg" + srv);
                srv.createNewSession(session);
            } catch (RuntimeException e) {
                logger.error("broadcastCreateNewSession. error broadcasting",e);
            }
        }
    }
}
```



A screenshot of an IDE's context menu. The menu items are: Debug As, Profile As, Team, Compare With, Replace With, CSSL, and Properties. The **CSSL** item is selected, and a sub-menu is open. The sub-menu items are: Error Log, Value, false, true, August 27, 2010 12:4:29 AM, **Generar código**, and D:\pablo\GanyMede-modeling\runtime-Ecl default.cm. The **Generar código** option is highlighted with a yellow circle.

Regeneración

```
package gen.model;  
  
public class MessageX extends gen.model.Message impl  
  
    public MessageX() {  
    }  
  
}
```

Nuevo elemento

```
/**  
 * @generated NOT  
 */  
public boolean execute() throws GPFWEException {  
    ((ChatSessionSharedObjects)getSession()).getSharedObjects  
    return false;  
}
```

No Sobreescrito

```
/**  
 * @generated  
 */  
private void broadcastCreateNewSession(ChatSession session) {  
    try {  
        I  
        List<gen.remoting.api.ChatSessionService> lst=  
            ChatSessionBroadcaster.getInstance().getInvokersAll(gen.re  
        logger.info("broadcastCreateNewSession:" + lst);  
        for (gen.remoting.api.ChatSessionService srv: lst){
```

Sobreescrito

En resumen..

- Se genera automáticamente la comunicación entre los pares
- Se generan los objetos que representan a los conceptos de dominio y sus relaciones
- Se provee una abstracción de programación
- Se facilita el acceso a los objetos equivalentes en distintas máquinas
- Se acelera el desarrollo inicial del programa

Conclusiones

- Dominio demasiado amplio
- Generación total
- Aumento de productividad
- Regeneración
- Centrarse en conceptos del dominio
- Explorando posibles continuaciones
- Reemplazar framework (¿Google Wave?)
- Reducir dominio / mayor expresividad

FIN