5. Simulación y resultados

5.1. Creación de una aplicación para ejecución de pruebas

La ejecución y el testeo del proceso en Eclipse se realizará en lenguaje de programación java. Por medio de la implementación y ejecución del archivo definido como ProcessTest.java se iniciará la secuencia definida para el proceso de modelado de vino. A través de la edición directa de este archivo, que define una clase java llamada ProcessTest, será posible especificar los parámetros principales de la ejecución además de dar comienzo al proceso BPMN referenciado en su código. Para el caso concreto comentado a lo largo de esta memoria, el código de ejecución correspondiente se presenta en el cuadro expuesto a continuación.

```
package com.sample;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Date;
import java.util.HashMap;
import org.drools.KnowledgeBase;
import org.drools.builder.KnowledgeBuilder;
import org.drools.builder.KnowledgeBuilderFactory;
import org.drools.builder.ResourceType;
import org.drools.io.ResourceFactory;
import org.drools.logger.KnowledgeRuntimeLogger;
import org.drools.logger.KnowledgeRuntimeLoggerFactory;
import org.drools.logger.KnowledgeRuntimeLoggerFactory;
import org.drools.runtime.StatefulKnowledgeSession;
import org.jbpm.process.instance.*;
import org.jbpm.process.workitem.wsht.WSHumanTaskHandler;
public class ProcessTest {
      public static final void main(String[] args) {
            KnowledgeBuilder kbuilder =
            KnowledgeBuilderFactory.newKnowledgeBuilder();
            kbuilder.add(ResourceFactory.newClassPathResource("general.bpmn")
            , ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("inspeccion.bpm
            n"), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("peticion.bpmn"
            ), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("recepcion.bpmn
            "), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("prensada.bpmn"
            ), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("busqueda.bpmn"
            ), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource("descubre.bpmn"
            ), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource(
            "llenado.bpmn"), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource(
            "calidad.bpmn"), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource(
            "embotellado.bpmn"), ResourceType.BPMN2);
            kbuilder.add(ResourceFactory.newClassPathResource(
            "marketing.bpmn"), ResourceType.BPMN2);
```

```
                kbuilder.add(ResourceFactory.newClassPathResource(
                "distribucion.bpmn"), ResourceType.BPMN2);
                kbuilder.add(ResourceFactory.newClassPathResource("
                configurar.bpmn"), ResourceType.BPMN2);
                KnowledgeBase kbase = kbuilder.newKnowledgeBase();
                StatefulKnowledgeSession ksession =
                kbase.newStatefulKnowledgeSession();
                KnowledgeRuntimeLogger logger =
                KnowledgeRuntimeLoggerFactory.newConsoleLogger(ksession);
                KnowledgeRuntimeLoggerFactory.newConsoleLogger(ksession);
                ksession.getWorkItemManager().registerWorkItemHandler("Human
                Task", new WSHumanTaskHandler());

                //CALCULO DE LA FECHA DE EJECUCIÓN
                java.util.Date utilDate = new java.util.Date();
                java.sql.Date sqlDate = new
                java.sql.Date(utilDate.getTime());

                //VARIABLES LOCALES AL PROCESO
                HashMap<String, Object> params = new HashMap<String,
                Object>();
                params.put("paso", true);
                params.put("x", false);
                params.put("db", true);
                params.put("fecha", sqlDate);
                params.put("id_empleado", 0);
                params.put("id_calidad", 0);
                params.put("s_empleado", 'A');
                params.put("id_pedido", 0);
                params.put("id_cuba", 0);
                params.put("id_lote", 0);
                params.put("id_inspeccion", 0);

                //ELEMENTOS MODIFICABLES EN NUESTRO SISTEMA
                params.put("driver_db", "org.hsqldb.jdbcDriver");
                params.put("url_db", "jdbc:hsqldb:file: C:/Users /Jesus/
                Desktop/PROYECTO/jbpm-installer/workspace/Modelado Vino/db");
                params.put("cantidad_lote", 500f);
                params.put("capacidad_barrica", 250f);
                params.put("capacidad_cuba", 250f);
                params.put("capacidad_botella", 0.75f);
                params.put("t_env_joven", 1000f);
                params.put("t_env_crianza", 2000f);
                params.put("t_env_reserva", 3000f);
                params.put("t_rep_joven", 1000f);
                params.put("t_rep_crianza", 2000f);
                params.put("t_rep_reserva", 3000f);
                params.put("t_camion", 300f);
                ksession.startProcess("com.sample.general",params);
        }
}
```

El primer paso a la hora de implementar la ejecución del proceso vendrá descrito por la creación de una nueva instancia para KnowledgeBuilder. Este elemento se encargará de recoger todos los archivos bpmn2 suministrados mediante su método add y crear así un KnowledgePackage consumible por el repositorio de aplicaciones KnowledgeBase. Para lograr esta asociación entre KnowledgeBuilder y los distintos modelos definidos en bpmn se hace necesario el uso de elementos tipo Resource.

Una vez conseguida la asociación anterior se puede proceder a la creación de una instancia de KnowledgeBase. Este elemento representará el repositorio donde permanecerán recogidos los distintos elementos añadidos al KnowledgeBuilder, permitiendo además la creación de sesiones para la ejecución de los procesos. Como se puede observar, la implementación de una nueva sesión resulta bastante intuitiva ya que sólo será necesaria la invocación del método newStatefulKnowledgeSession.

La implementación del elemento StatefulKnowledgeSession permite la interacción iterativa con el engine de JBoss a través de una sesión definida para ello. Dentro de dicha sesión se conseguirá finalmente iniciar la ejecución del proceso a través del uso del método interno definido como startProcess. Entre los parámetros aceptados por este método se encuentran, por un lado, el identificador del proceso en ejecución para el establecimiento inequívoco del inicio de la secuencia y, por el otro, un objeto de tipo HashMap<String, Object> que fijará valores iniciales a las variables locales internas definidas en el proceso.

El último elemento a tener en cuenta en la implementación del archivo java definido anteriormente es el KnowledgeRuntimeLogger. Este elemento, usado a modo de asistente en la ejecución del proceso, irá creando secuencialmente un registro de incidencias a medida que el proceso completa las diferentes etapas establecidas. Tal y como cabe esperar, este registro no aportará por tanto ningún beneficio operacional al sistema, ya que su implementación sólo contempla utilidad al programador, reportando los posibles fallos asociados al mal funcionamiento de la ejecución.

5.2. Human task

Como elemento central de la interacción humana con el proceso, las tareas humanas se han convertido en un punto de interés trascendental en la realización de este proyecto. Por este motivo, y con objeto de mejorar su comprensión, resulta de vital importancia puntualizar con mayor profundidad sus principales propiedades o campos de definición. Entre estos campos se destacan los siguientes:

- ✓ ActorId: este campo hace referencia al participante del proyecto responsable de la ejecución de esta tarea. Para el caso concreto de que haya más de un actor responsable de la tarea se puede añadir más de un identificador, separando cada uno de ellos por comas.

- ✓ GroupId: esta potente propiedad permite asociar una tarea de ejecución a un grupo cerrado de usuarios. De este modo, cada uno de ellos competirá por la adjudicación de la tarea ya que sólo uno de ellos será el encargado de realizarla.
- ✓ Priority: para facilitar la organización del usuario, esta variedad de tareas permite la definición de prioridades específicas de ejecución. A partir de ellas se podrá dotar de mayor o menor importancia a una tarea según las características impuestas por el desarrollador.
- ✓ Skippable: este campo plantea la posibilidad de dejar al usuario la decisión de realizar o no la tarea programada.
- ✓ On entry and on exit actions: estos campos están asociados con la ejecución automática de scripts de código a la entrada y salida de la tarea humana.
- ✓ Parameter mapping: esta lista ordenada de variables permite obtener una relación directa entre los parámetros locales del proceso y los parámetros internos de la tarea. Toda esta asociación se realizará al inicio de la ejecución de la tarea, copiando la información de unos en los otros.
- ✓ Result mapping: esta asociación resulta análoga a la realizada por parameter mapping con la salvedad que, para este caso, la copia de datos se realiza a la inversa durante la finalización de la tarea.
- ✓ TaskName: hace referencia al nombre del formulario o plantilla definido en ftl para la interacción web con el usuario.
- ✓ Timer: este campo permite evitar posibles bloqueos durante la ejecución a través de la asignación de tiempos máximos para la realización de las tareas.

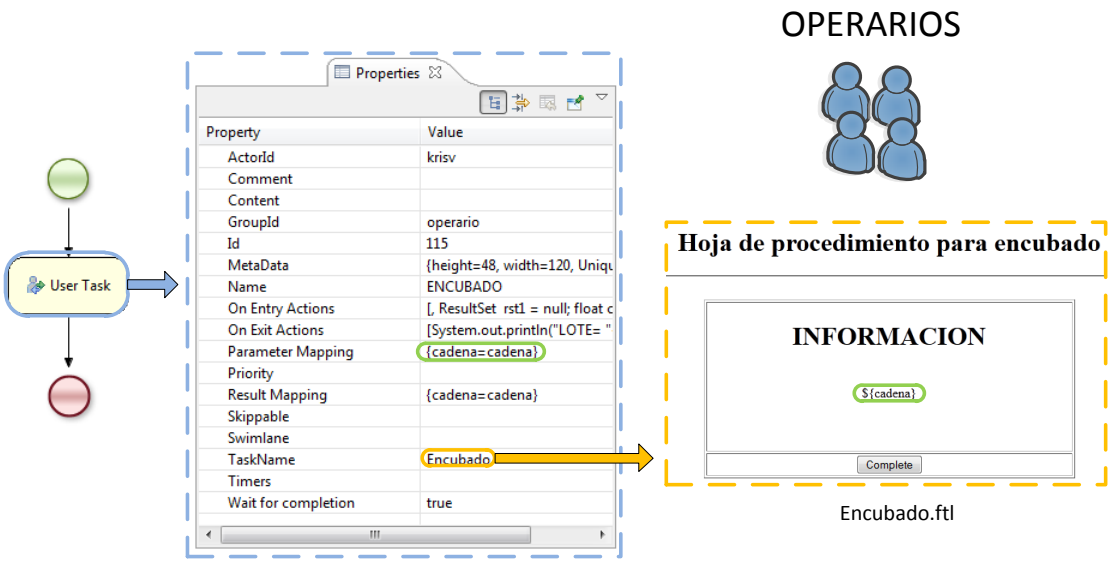

Figura 45: Asociación entre las tareas humanas y su interacción con el usuario

Uno de los aspectos más representativos configurables para estas tareas de usuario lo representa la asignación del actor o actores asociados al nodo. Esta asignación podrá ser realizada según dos puntos de vista completamente independientes. El primero de ellos vendrá determinado por la asignación directa de la tarea a un usuario o conjunto de éstos mediante la utilización del campo ActorId. A raíz del registro interno para los usuarios del servidor de aplicación se podrá implementar esta asignación directa de forma clara y unívoca. Para el caso particular de una asignación múltiple, es decir una tarea asignada a varios usuarios distintos, la tarea en cuestión quedará recogida dentro del cubo de tareas accesible a todos ellos, siendo necesario completar todas ellas para continuar con el proceso.

Respecto al segundo método de asignación, referenciado bajo el campo GroupId, el tratamiento de las tareas se plantea bien distinto. En este caso, la tarea es asignada a un grupo fijo de usuarios definidos según su rol específico en el sistema. Una vez lanzada la tarea, todos los usuarios tendrán visible la actividad a realizar con la salvedad de que para este caso los usuarios competirán por ella. Según esta estructura sólo un usuario de todos los que compartan el mismo rol podrá ser capaz de seleccionar y completar la tarea, implementando así una actuación bajo demanda.

Como aspecto singular asociado a la integración de servicios externos mediante esta tecnología, el manejador de tareas para la interacción humana deberá también ser integrado al motor interno de jBPM. Para realizar esta integración será necesario proceder al registro del manejador a través de la propia sesión definida en java. Para el caso particular implementado

en este proyecto, el manejador escogido es el elemento WSHumanTaskHandler y quedará registrado en la sesión mediante la sentencia:

```
ksession.getWorkItemManager().registerWorkItemHandler("Human Task", new
WSHumanTaskHandler());
```

Bajo esta elección, la comunicación entre el motor jBPM y el manejador de tareas seguirá una arquitectura básica de mensajes servidor-cliente. A través de ella, y utilizando la consola de usuario habilitada en el servidor, se realiza la gestión autónoma de las tareas para los distintos usuarios del sistema.

5.3. Interfaz de trabajo

La interfaz de trabajo establecida para los usuarios del proceso aparece definida por una agradable consola fácilmente accesible por cualquier navegador web. Tras alcanzar la ubicación de la consola, el primer paso para acceder a ella será completar la autenticación como usuario admitido en el proceso. Estos usuarios permanecerán definidos en ficheros locales de autentificación, donde quedarán registrados mediante combinación de nombre de usuario y contraseña. Para la configuración en red local, la ubicación fijada para alcanzar la consola de trabajo responde a la siguiente URL: localhost:8080/jbpm-console. La ventana del navegador para la autenticación del usuario se representa en la captura de pantalla expuesta a continuación.

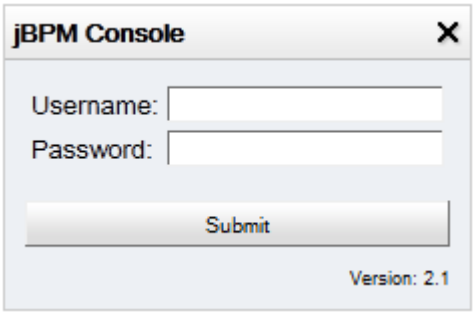

Figura 46: Ventana inicial para la autenticación del usuario

Una vez superado este proceso de autenticación, el usuario tendrá pleno acceso a la interfaz de gestión de tareas asociada a su cuenta. Dentro de esta interfaz existirán una serie de pestañas de elección así como diferentes sub-ventanas encargadas de mostrar la información interna relacionada con cada una de ellas. La interfaz comentada se recoge en la siguiente captura de pantalla.

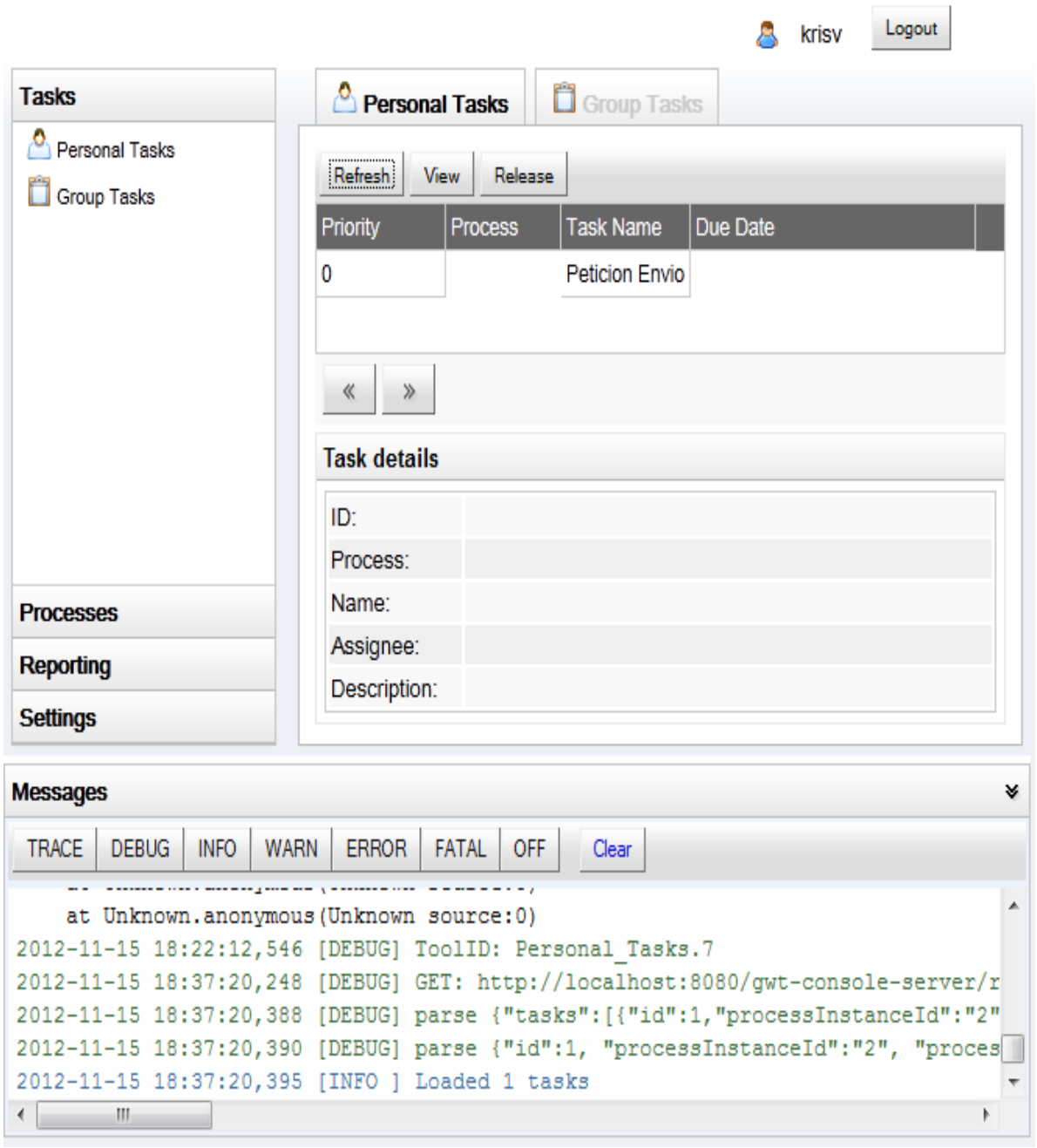

Figura 47: Interfaz de usuario para la gestión de tareas activas

En el índice de la izquierda se puede observar las diferentes ventanas accesibles por el usuario a través de las distintas pestañas definidas en la interfaz. La primera de ellas hace referencia a la ventana de gestión de tareas asignadas al usuario. Como se comentó anteriormente, la clasificación de estas tareas se encontrará segmentada según la naturaleza propia de la tarea, de forma que se podrán encontrar tareas tanto personales como de grupo. Para el caso concreto de las tareas de grupo, la adjudicación de la misma se realizará bajo demanda, siendo únicamente un usuario el encargado de completarla.

Otra de las pestañas incluidas en la interfaz es la que aparece bajo el sobrenombre de “procesos”. Esta nueva ventana, representada en la captura de pantalla de la figura 50, muestra un historial de ejecución de todos los procesos implementados en el servidor. En este caso no sólo mostrará los procesos que fueron ejecutados, sino también los que aún permanecen en ejecución así como algunas características interesantes asociadas a todos ellos. De forma adicional, en esta pestaña también se incluye la posibilidad de visualizar todos los procesos instalados en el sistema. A partir de ellos, y con una sencilla ventana de inicialización, se podrán iniciar desde cero nuevas instancias de los mismos, garantizando así su cómoda ejecución desde el inicio.

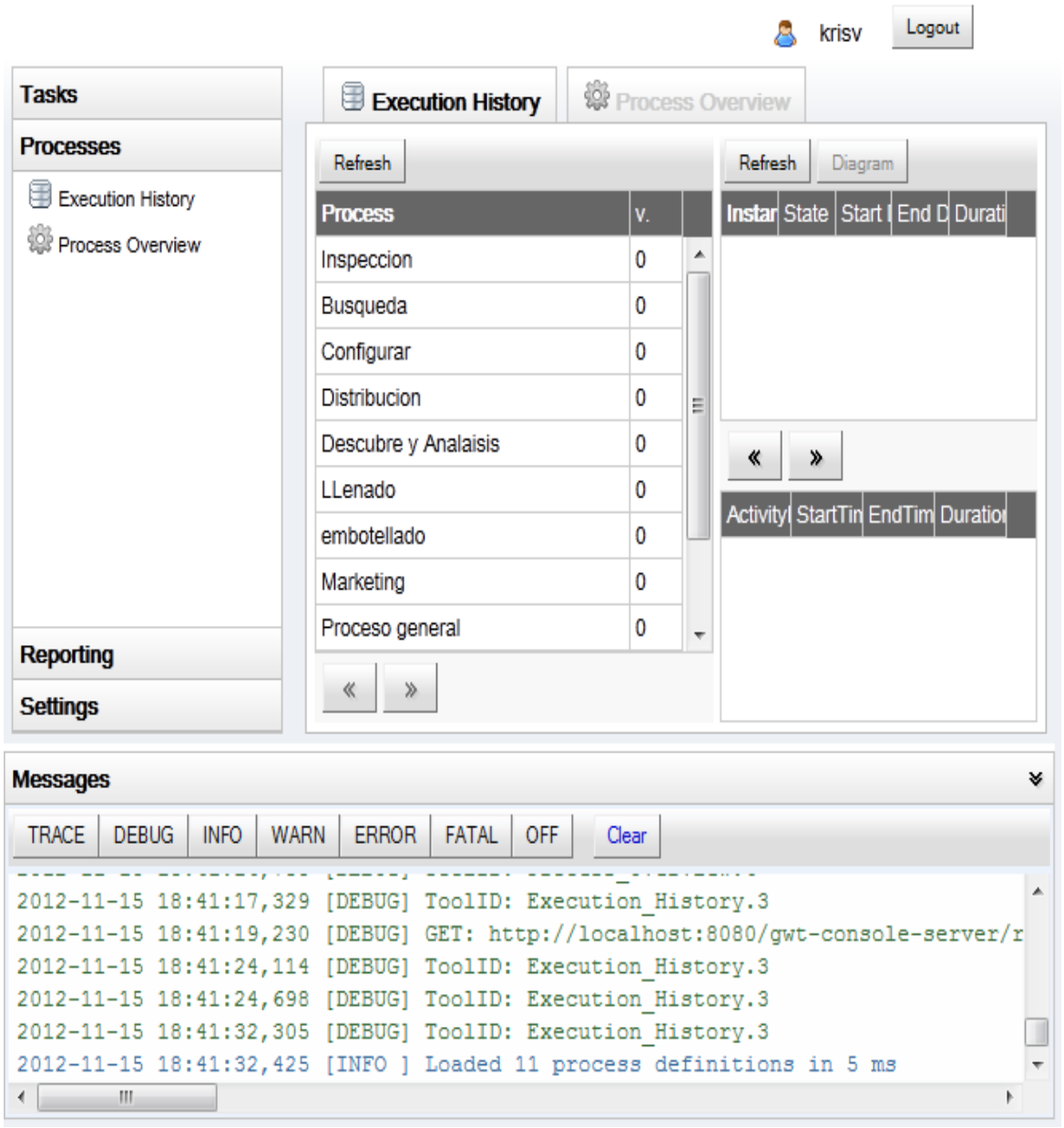

Figura 48: Ventana de gestión de procesos para el historial de ejecuciones

La última pestaña de esta interfaz está orientada principalmente a la gestión general del proceso de negocio por medio de un registro de las incidencias y ejecuciones asociadas al sistema. De este modo, y a través de una representación gráfica de monitorización de la actividad, se podrá realizar un seguimiento pseudo-controlado de las ejecuciones detallando además aspectos adicionales como su ubicación temporal o su duración.

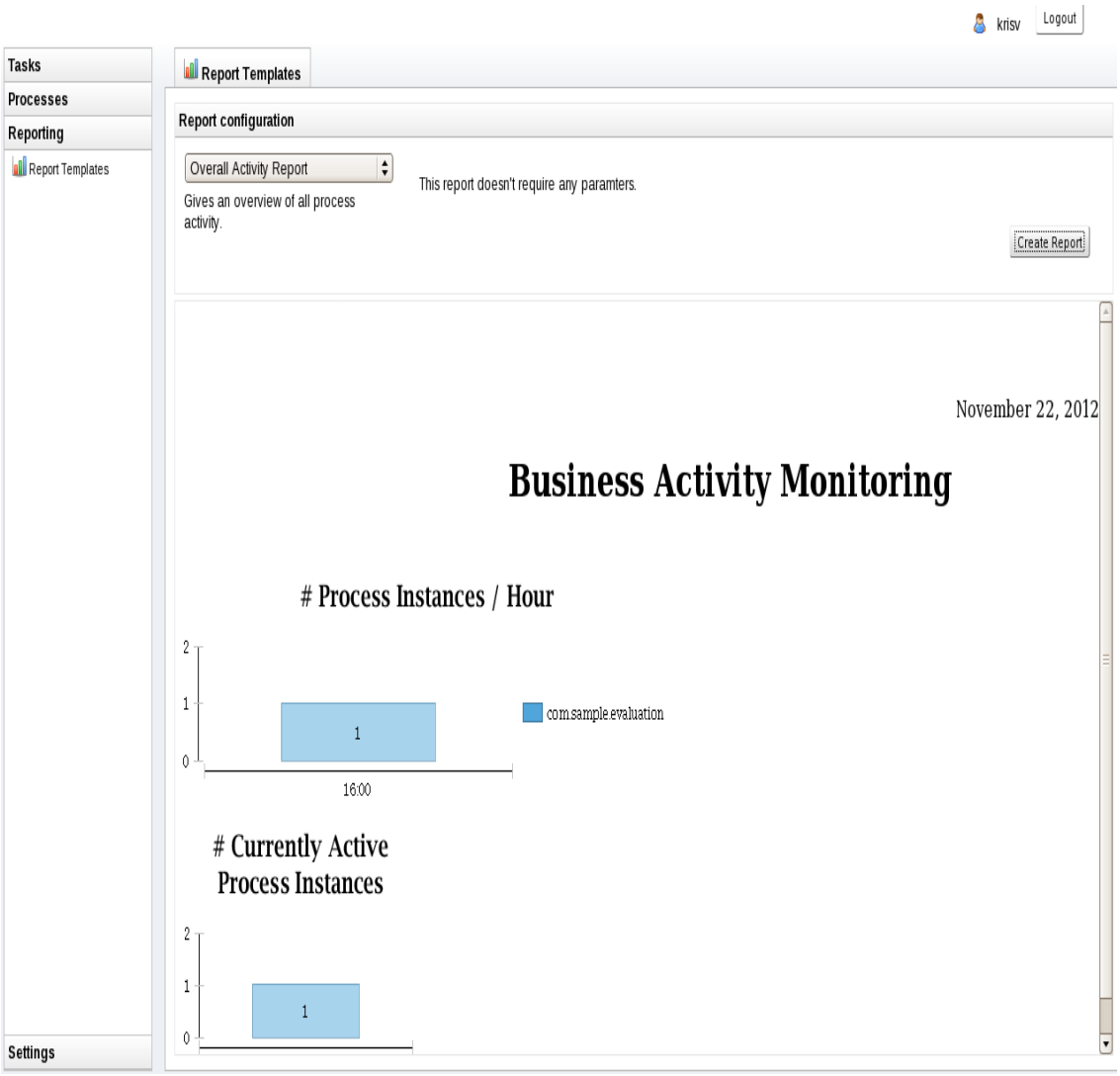

Figura 49: Interfaz de gestión con la monitorización de la actividad en el servidor