

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MÀSTER UNIVERSITARIO EN INGENIERIA DEL SOFTWARE,
METODOS FORMALES Y SISTEMAS DE INFORMACIÓN



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



DEPARTAMENTO DE SISTEMAS
INFORMÁTICOS Y COMPUTACIÓN

APLICACIÓN DE UN SISTEMA DE GESTIÓN DE REGLAS DE NEGOCIO EN EL DESARROLLO E INTEGRACIÓN DE UN SISTEMA DE ALERTAS ACÚSTICAS

Trabajo Fin de Máster

Autor:

Rafael Vidal Pastor

Director:

Dr. D. Antonio Molina Marco

Valencia, a 14 de septiembre de 2015

ÍNDICE DE CONTENIDO

1. Introducción.	- 8 -
1.1 Nombre y descripción del proyecto.	- 8 -
1.2 Objetivos del proyecto.	- 9 -
1.2.1 Objetivos generales.	- 9 -
1.2.2 Objetivos específicos.	- 9 -
1.3 Contenido de la memoria.	- 10 -
2. Marco teórico.	- 11 -
2.1 Introducción a los Sistemas Expertos.	- 11 -
2.1.1 Antecedentes de los sistemas expertos.	- 11 -
2.1.2 Usos de los sistemas expertos.	- 13 -
2.1.3 Tipologías de Sistemas Expertos.	- 14 -
2.1.4 Ventajas de uso de los sistemas expertos.	- 15 -
2.1.5 Limitaciones de los sistemas expertos.	- 16 -
2.1.6 Arquitectura de los sistemas expertos.	- 17 -
2.1.7 Sistemas Expertos actuales.	- 20 -
2.2 Sistemas Basados en reglas.	- 22 -
2.2.1 La Base de Conocimiento en sistemas basados en reglas.	- 22 -
2.2.2 El motor de inferencia.	- 24 -
2.2.3 Control de Coherencia.	- 28 -
2.2.4 Explicación de Conclusiones.	- 29 -
2.3 Bussiness Rules Management Systems (BRMS).	- 30 -
2.3.1 ¿Qué es un BRMS?	- 30 -
2.3.2 ¿Qué son las reglas de negocio?	- 32 -
2.3.3 Tipos de reglas de negocio.	- 33 -
2.3.4 Ventajas e inconvenientes de los sistemas basados en reglas.	- 33 -
2.3.5 Cuándo usar un BRMS.	- 35 -
2.3.6 El motor de reglas de negocio.	- 35 -
2.3.7 <i>Forward Chaining, Backward Chaining</i> y Sistemas de razonamiento Híbrido.	- 36 -
2.3.8 BRMS que podemos encontrar.	- 38 -
2.4 El Framework Drools.	- 40 -
2.4.1 ¿Qué es Drools?	- 40 -
2.4.2 Historia del proyecto.	- 40 -
2.4.3 Herramientas de la suite.	- 40 -
2.4.4 Apuntes sobre Drools.	- 46 -
2.4.5 Implementación del algoritmo de RETE que hace Drools.	- 47 -
2.4.6 Las reglas de negocio en Drools.	- 51 -
3 Memoria Técnica.	- 60 -
3.1 Introducción.	- 60 -
3.2 Problemática a resolver.	- 60 -
3.3 Análisis de las opciones disponibles en el mercado.	- 60 -
3.4 Justificación de las tecnologías seleccionadas para el desarrollo del producto.	- 61 -
3.5 Análisis y diseño.	- 61 -
3.5.1 Análisis (especificaciones).	- 62 -
3.5.1.1 Introducción.	- 62 -

i.	Propósito.....	- 62 -
ii.	Alcance.....	- 62 -
iii.	Definiciones, acrónimos y abreviaturas.....	- 62 -
3.5.1.2	Descripción general.....	- 63 -
i.	Perspectiva del producto. Análisis funcional.....	- 63 -
ii.	Reporte del Modelo de Casos de Uso.....	- 65 -
iii.	Restricciones.....	- 66 -
iv.	Evolución previsible del sistema.....	- 66 -
3.5.1.3	Requisitos específicos.....	- 66 -
i.	Requisitos comunes de los interfaces.....	- 66 -
a.	Interfaces de usuario.....	- 66 -
b.	Interfaces de hardware.....	- 67 -
ii.	Requisitos funcionales.....	- 67 -
a.	Requisito funcional 1.....	- 67 -
b.	Requisito funcional 2.....	- 67 -
c.	Requisito funcional 3.....	- 67 -
d.	Requisito funcional 4.....	- 67 -
iii.	Requisitos no funcionales.....	- 67 -
a.	Fiabilidad.....	- 67 -
b.	Disponibilidad y Portabilidad.....	- 67 -
c.	Mantenibilidad.....	- 68 -
3.5.2	Diseño.....	- 68 -
3.5.2.1	Introducción.....	- 68 -
3.5.2.2	Modelado del problema.....	- 69 -
3.5.2.3	Detalles de implementación.....	- 71 -
3.5.2.4	Diagramas y especificación de funciones.....	- 76 -
i.	Diagrama de estados.....	- 76 -
ii.	Diagrama de secuencia.....	- 77 -
iii.	Diagrama de funciones. UML.....	- 78 -
iv.	Especificación por interfaz función.....	- 79 -
3.5.2.5	Prototipo de la aplicación web de configuración.....	- 82 -
3.5.3	Metodologías de desarrollo.....	- 86 -
3.6	Manual de puesta en funcionamiento.....	- 88 -
3.6.1	Instalación.....	- 88 -
3.6.2	Preparando el entorno.....	- 89 -
i.	Preparando los dispositivos.....	- 89 -
ii.	Preparando el entorno de red.....	- 92 -
iii.	Configurando el servicio web (LINUX).....	- 93 -
3.6.3	Puesta en marcha del programa en modo consola (WINDOWS).....	- 95 -
3.6.4	Uso mediante interfaz web.....	- 98 -
3.7	Resultados.....	- 102 -
3.7.1	Ejemplos de ejecución (salida consola).....	- 102 -
3.7.2	Videos.....	- 112 -
4	Conclusiones y posibles ampliaciones al proyecto.....	- 113 -
4.1	Conclusiones.....	- 113 -
4.2	Ampliaciones al proyecto.....	- 114 -
5	Presupuesto.....	- 116 -
6	Pliego de condiciones.....	- 117 -
6.1	Descripción del proyecto.....	- 117 -
6.2	Condiciones generales.....	- 117 -
6.2.1	Condiciones generales facultativas.....	- 117 -
i.	Promotor del proyecto.....	- 117 -

ii.	Obligaciones y derechos del proyectista encargado del proyecto.	- 117 -
iii.	Facultades del promotor del proyecto.	- 118 -
iv.	Comienzo, ritmo, plazo y condiciones generales de la ejecución del objetivo del proyecto.	- 118 -
v.	Recepción provisional del programa.	- 119 -
vi.	Periodo de prueba.....	- 119 -
6.2.2	Condiciones generales legales.	- 119 -
i.	Reconocimiento de marcas registradas.....	- 119 -
ii.	Derechos de autor.....	- 119 -
iii.	Causas de rescisión del proyecto.....	- 119 -
6.3	Condiciones Particulares.....	- 120 -
i.	Condiciones software.	- 120 -
ii.	Condiciones Hardware.....	- 120 -
iii.	Formación del usuario.....	- 120 -
	Bibliografía	- 121 -
	WEBGRAFÍA:.....	- 123 -
	Anexo 1: REPRESENTACIÓN DEL CONOCIMIENTO.....	-125-
	ANEXO 2:DOCUMENTACIÓN JAVADOC DE LA APLICACIÓN JAVA.....	-129-

Índice de Ilustraciones

Ilustración 1: Esquema de evolución de los sistemas expertos.....	- 12 -
Ilustración 2: Arquitectura de los Sistemas Expertos.....	- 17 -
Ilustración 3: Mecanismo de inferencia <i>Modus Ponens</i>	- 24 -
Ilustración 4: Mecanismo de inferencia <i>Modus Tollens</i>	- 25 -
Ilustración 5: Esquema estructura BRMS.....	- 31 -
Ilustración 6: Visión de alto nivel de un Sistema de Producción de Reglas.....	- 36 -
Ilustración 7: Esquema mecanismo de <i>Forward Chaining</i>	- 36 -
Ilustración 8: Mecanismo de Backward Chaining	- 37 -
Ilustración 9: Productos de la suite Drools	- 41 -
Ilustración 10: Interfaz Drools Worbench	- 42 -
Ilustración 11: Interfaz Drools Expert.....	- 43 -
Ilustración 12: Interfaz jBPM	- 44 -
Ilustración 13: OptaPlanner	- 45 -
Ilustración 14: Clasificación de nodos RETE.....	- 47 -
Ilustración 15: ObjectTypeNodes en red Rete.....	- 48 -
Ilustración 16: Alpha Node en una red Rete.....	- 48 -
Ilustración 17: Ejemplo matching parcial en red Rete	- 49 -
Ilustración 18: Reglas Drools.....	- 50 -
Ilustración 19: Ejemplo de compartición de nodo en red Rete.....	- 51 -
Ilustración 20: Relación entre Objetos en una regla	- 52 -
Ilustración 21: Ejemplo de mapeado DSL en Drools.....	- 54 -
Ilustración 22: Tablas de decisión.....	- 57 -
Ilustración 23: Plantilla RuleSet.....	- 57 -
Ilustración 24: Reporte del modelo de Casos de Uso.....	- 65 -
Ilustración 25: Esquema dispositivos en red.....	- 70 -
Ilustración 26: Dispositivos EWB-100.....	- 70 -
Ilustración 27: Diagrama de estados DTE.....	- 76 -
Ilustración 28: Diagrama de secuencia.....	- 77 -
Ilustración 29: Esquema de alto nivel de los mensajes intercambiados por el Hardware del sistema	- 77 -
Ilustración 30: Diagrama UML.....	- 78 -
Ilustración 31: Interfaz de configuración de grupos.....	- 82 -
Ilustración 32: Interfaz de puesta en marcha de la aplicación	- 83 -
Ilustración 33: Interfaz envío de mensajes de demostración	- 83 -

Ilustración 34: Consola envío de mensajes por teclado.....	- 84 -
Ilustración 35: Consola envío de mensajes desde terminal móvil	- 84 -
Ilustración 36: Visualización consola desde terminal móvil	- 85 -
Ilustración 37: Fichero txt para creación del audioclip	- 90 -
Ilustración 38: Consola de configuración de los dispositivos EWB100	- 91 -
Ilustración 39: Consola de carga de diccionario en dispositivos	- 92 -
Ilustración 40: Motorola EWB-100 Deployment Application.....	- 92 -
Ilustración 41: Consola de configuración de red y servicio Air Beam	- 93 -
Ilustración 42: Contenido del fichero de configuración	- 94 -
Ilustración 43: Fichero de configuración "config.cfg"	- 96 -
Ilustración 44: Orden de ejecución del programa	- 96 -
Ilustración 45: Consola mostrando la configuración de la aplicación.....	- 97 -
Ilustración 46: Esperando recepción de eventos.....	- 97 -
Ilustración 47: Configuración de grupos de difusión	- 98 -
Ilustración 48:Inicio procesamiento de eventos a través de interfaz web	- 99 -
Ilustración 49: PID de proceso ejecutado en servicios remoto.....	- 99 -
Ilustración 50: Estado interfaz después de parar el servicio de monitorización de eventos.....	- 99 -
Ilustración 51: Pestaña de envío directo de mensajes a dispositivo.....	- 100 -
Ilustración 52: Introducción de órdenes de voz por teclado	- 100 -
Ilustración 53: Terminal tablet industrial accediendo a aplicación.....	- 101 -
Ilustración 54: Contenido fichero de configuración " <i>config.cfg</i> ":	- 102 -
Ilustración 55: Orden de ejecución programa por consola	- 102 -
Ilustración 56: Configuración aplicada a programa en ejecución.....	- 103 -
Ilustración 57: Suspensión de hilos de ejecución en cola vacía.....	- 103 -
Ilustración 58: Contenido de directorio de eventos antiguos.....	- 104 -
Ilustración 59: Contenido fichero <i>taw1-1.xml</i>	- 104 -
Ilustración 60: Regla de negocio para evento en cola sur	- 105 -
Ilustración 61: Procesado de evento <i>taw1-1</i>	- 105 -
Ilustración 62: Acciones desencadenadas por evento <i>taw1-1</i>	- 106 -
Ilustración 63: Contenido fichero <i>taw2-1.xml</i>	- 106 -
Ilustración 64: Regla de negocio para posible hurto	- 107 -
Ilustración 65: Procesamiento fichero <i>taw2-1.xml</i>	- 107 -
Ilustración 66: Salida procesamiento de evento <i>taw3-1</i>	- 108 -
Ilustración 67: Regla de negocio para evento fuego en pasillo 5.....	- 109 -

Ilustración 68: Salida procesamiento de evento cámara ("*OccupancyRateTAW*")- 109

-

Ilustración 69: Envío señal de alerta de cola.....- 110 -

Ilustración 70: Recepción simultánea de de múltiples eventos- 111 -

1. Introducción.

1.1 Nombre y descripción del proyecto.

Nombre:

APLICACIÓN DE UN SISTEMA DE GESTIÓN DE REGLAS DE NEGOCIO EN EL DESARROLLO E INTEGRACIÓN DE UN SISTEMA DE ALERTAS ACÚSTICAS

Descripción:

El objetivo del TFM es el análisis, desarrollo e integración de un sistema de alertas acústicas (mensajes de voz), para el envío de mensajes mediante dispositivos de comunicación VOIP. Para la reconfiguración y gestión de la lógica de negocio se utilizará Drools: un sistema de gestión de reglas de negocio y motor de razonamiento que permite el acceso y gestión de cambio de políticas y reglas de negocio.

La aplicación final, desarrollada en un contexto laboral, tendrá como objetivo capturar los eventos que genera un sistema de cámaras capaz de detectar condiciones tales como una cola en la zona de cajas de un supermercado, la presencia de un objeto extraño en una zona, etc. Se tratará estos eventos para comprobar qué condiciones plantean y enviará una señal de emisión de un mensaje predefinido.

El desarrollo se centrará fundamentalmente en tres aspectos:

- Desarrollo del motor generador de alertas acústicas.
- Desarrollo de interfaz de configuración de grupos de dispositivos.
- Integración de la aplicación con un sistema de cámaras (desarrollo de una empresa externa).

El resultado final de este TFM pretende abordar tres elementos:

- Presentación de la tecnología, en particular, del sistema gestor de reglas de negocio (motor de la aplicación).
- Presentación de una aplicación desarrollada sobre la misma en el contexto de un proyecto laboral.
- Presentación de las posibilidades que brinda y desarrollos futuros.

Autor:

Rafael Vidal Pastor

Directores:

Dr. D. Antonio Molina Marco

1.2 Objetivos del proyecto.

1.2.1 Objetivos generales.

El objetivo general del proyecto consistirá en el desarrollo de un sistema de envío de alertas acústicas mediante dispositivos de comunicación VOIP. Este sistema será utilizado para realizar tareas de notificación de diferentes condiciones físicas o virtuales que se puedan dar en el contexto de cualquier proceso industrial, logístico o de negocio de cualquier sector.

1.2.2 Objetivos específicos.

Los objetivos específicos del proyecto se desglosan en los siguientes puntos fundamentalmente:

- Investigación y búsqueda de dispositivos hardware con características industriales que den soporte al envío de mensajes.
- Desarrollo de un motor generador de alertas acústicas.
- Desarrollo de una interfaz de configuración de los dispositivos.
- Desarrollo de una consola de trabajo que permita el envío de mensajes a un determinado dispositivo.
- Integración de la aplicación con un sistema de sensorización por cámaras que realiza un proceso de monitorización de un área física.

1.3 Contenido de la memoria.

En este documento se pretende incorporar todos los aspectos prácticos y teóricos del proyecto desarrollado, análisis y requerimientos del sistema, ingeniería de concepción etc.

Asimismo, incluiremos parte del código de aplicación clave para poder explicar cómo funciona el software desarrollado, el manual de instalación y funcionamiento del mismo, etc.

La memoria se estructura en 5 grandes secciones:

1- **Introducción**, sección en la que nos encontramos, se trata de una presentación del contenido y estructura del proyecto y la memoria.

2- **Marco teórico**, abordaremos algunos puntos clave de los antecedentes en sistemas de gestión de conocimiento, sistemas expertos, sistemas de gestión de reglas de negocio y de la plataforma *Drools* (base del desarrollo planteado).

3- **Memoria técnica**, en esta sección se recogen todos los aspectos de análisis, diseño e implementación de la solución software propuesta. También en este apartado presentaremos los resultados, tanto en forma de imágenes/fotografías de las diferentes soluciones como remitiendo a una serie de videos de simulaciones recogidos.

4- **Presupuesto**, en esta breve sección hacemos un presupuesto pormenorizado de cuales han sido los costes del proyecto.

5- **Pliego de condiciones**, en esta sección presentamos las condiciones del proyecto.

2. Marco teórico.

2.1 Introducción a los Sistemas Expertos.

Los sistemas expertos también conocidos como sistemas basados en el conocimiento, son sistemas informáticos capaces de tomar decisiones o resolver diferentes problemas aplicando razonamiento a partir de una serie de reglas definidas por el personal experto de un área determinada.

Los sistemas expertos surgen como una ramificación del área de la inteligencia artificial. Pero en ellos el poder de resolución de un problema parte del conocimiento adquirido por los expertos humanos sobre el propio dominio de aplicación a lo largo del tiempo. Por tanto, la idea básica de estos programas es capturar sobre una base de conocimiento, toda la experiencia adquirida por parte del personal experto en una determinada temática de una organización. De manera que el personal no experto en esa área pueda aprovechar dicho conocimiento. De este modo cuando un operador plantee un problema al sistema, éste deberá imitar las actividades y razonamientos que realizaría un experto humano para obtener la resolución del problema.

Los sistemas expertos se basan en conocimiento declarativo (hechos sobre objetos, situaciones, etc.) y conocimiento de control (seguimiento de una acción). A continuación detallaremos como este tipo de sistemas son capaces de mantener una base de conocimiento y extraer conclusiones a partir del mismo.

2.1.1 Antecedentes de los sistemas expertos.

Los primeros desarrollos de sistemas expertos surgen a mediados de la década de 1960. En este período, los investigadores del campo de la Inteligencia Artificial pretendían dotar a los ordenadores de las diferentes reglas de razonamiento que siguen los humanos, y de esta manera aprovechar su capacidad computacional para conseguir un rendimiento super-humano, el primer intento en esta dirección fue el General Problem Solver (GPS), solucionador de problemas de propósito general de Newell en 1958.

Esta tecnología definía los pasos necesarios para que desde un estado inicial se pudiera llegar a una meta deseada. Y para ello requería que se le otorgara de un conjunto de operaciones, precondiciones y post-condiciones que permitieran reducir las diferencias entre el estado inicial y la meta.

Posteriormente, viendo que las expectativas de obtener un solucionador de problemas de propósito general eran más bien escasas, los investigadores se centraron en obtener un sistema aplicable a un problema de un ámbito específico. El primer desarrollo en este sentido fue DENDRAL, que también fue el primer sistema experto en tener aplicaciones sobre problemas reales de química y biología, al margen de la investigación computacional.

Con el desarrollo de DENDRAL los científicos llegaron a las siguientes conclusiones:

- I. La complejidad de los problemas requiere un alto conocimiento especializado sobre el área del problema.
- II. Los sistemas de propósito general no permitían construir un sistema de alto rendimiento.

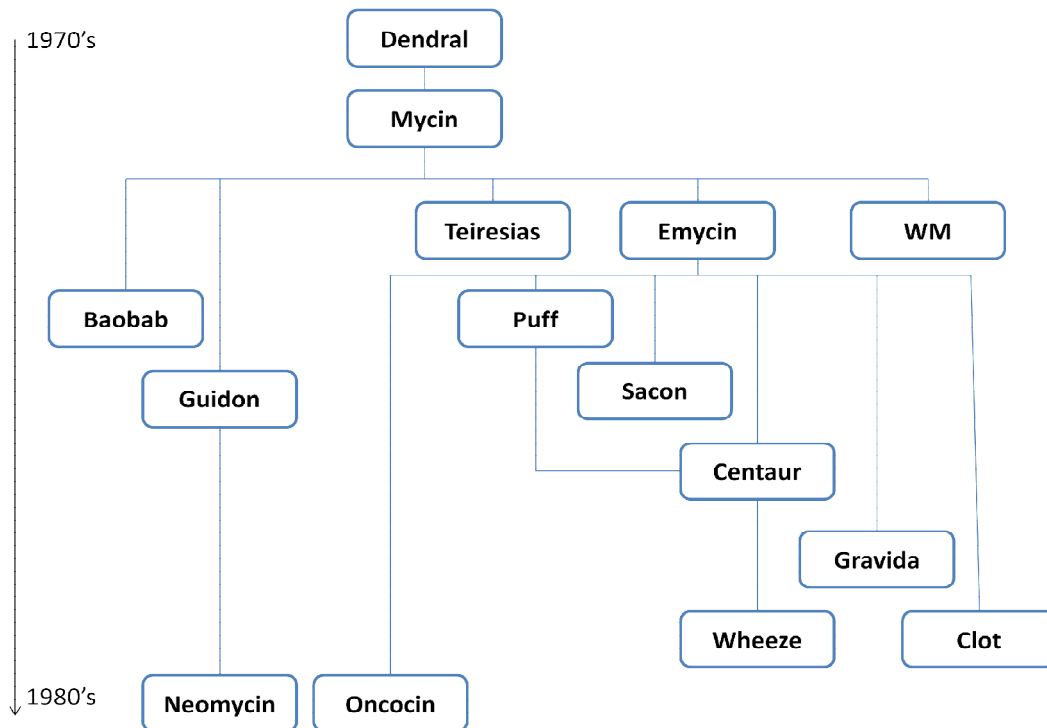
Posteriormente a este hito, fueron apareciendo diferentes sistemas expertos que fueron propiciando el avance en dicho campo. A continuación vemos algunos de los desarrollos que siguieron a DRENDAL:

MYCIN: inicialmente desarrollado por Ed. Feigenbaum y posteriormente ampliado por E. Shortliffe, era un software de asistencia médica especializado en enfermedades infecciosas de la sangre.

CADUCEUS: también de propósito médico pero acotado al campo de medicina interna, desarrollado a mediados de la década de 1980 por Harry Poole, en la Universidad de Pittsburgh.

XCON (*Expert Configurator*): sistema de producción basado en reglas escrito en OPS5 por John P. McDermott del CMU (1978). Tenía como objetivo asistir en la gestión de pedidos de los sistemas de computadores VAX de la DEC (*Digital Equipment Corporation*), seleccionando los componentes del sistema de acuerdo a los requerimientos del cliente.

Ilustración 1: Esquema de evolución de los sistemas expertos.



2.1.2 Usos de los sistemas expertos.

Los sistemas expertos poseen conocimiento altamente especializado de una determinada área, pero por lo general son carentes de todo conocimiento respecto al resto de áreas que quedan fuera de su ámbito de aplicación.

Es por ello que, por lo general, son usados para llevar a cabo tareas de monitorización y diagnóstico. Y son ampliamente utilizados en industrias tales como plantas nucleares y de energía, control de tráfico, búsqueda de yacimientos petrolíferos, etc.

Generalmente las aplicaciones se engloban dentro de las 9 áreas que detallamos a continuación:

- ✓ **DIAGNÓSTICO:** a partir de los síntomas son capaces de detectar fallos en sistemas. Para ello hacen uso de características del comportamiento anómalo de los sistemas. Por ejemplo, diagnóstico médico o fallos en circuitos o componentes mecánicos.
- ✓ **PREDICCIÓN:** infieren consecuencias que se pueden dar de manera probable a partir de determinadas condiciones detectadas. En ocasiones hacen uso de modelos de simulación, por ejemplo predecir daños ante desastres naturales.
- ✓ **INTERPRETACIÓN:** infieren la descripción de situaciones por medio de sensores. Además deben ser capaces de realizar tareas de filtrado de ruido sobre los datos o ausencia de estos.
- ✓ **MONITORIZACIÓN:** comparan observaciones del comportamiento de los sistemas con el comportamiento definido como adecuado. Por ejemplo sistemas de control aéreo o ferroviario.
- ✓ **CONTROL:** permite el gobierno automatizado de los sistemas. A partir de la interpretación del estado actual puede anticipar cuales son las acciones de control que se deberían tomar.
- ✓ **DEPURACIÓN:** sugieren correcciones ante fallos detectados.
- ✓ **DISEÑO:** a partir de una serie de limitaciones y restricciones son capaces de obtener la configuración óptima de objetos o sistemas. Para ello pueden hacer uso de análisis y simulación para verificar y probar diferentes combinaciones hasta encontrar la disposición óptima.
- ✓ **PLANIFICACIÓN:** diseñan un conjunto completo de acciones a tomar para optimizar un recurso o espacio. Por ejemplo planificación financiera.

- ✓ **INSTRUCCIÓN:** diagnóstico, revisión y aplicación de correctivos ante una actividad formativa.

Los sistemas expertos se recomiendan especialmente cuando se dan las siguientes situaciones:

- El conocimiento es difícil de adquirir o se basa en reglas que solo pueden ser adquiridas a través de la experiencia.
- La mejora del conocimiento es esencial.
- El problema está sujeto a reglas o códigos cambiantes.
- Sectores en los que los expertos humanos son caros o difíciles de encontrar.
- Cuando los usuarios poseen un conocimiento muy limitado sobre el tema.

2.1.3 Tipologías de Sistemas Expertos

Principalmente existen tres tipos de sistemas expertos, y cada uno de ellos aplica una serie de técnicas para obtener la resolución del problema que se le plantea:

:

- Sistemas basados en reglas previamente establecidas.
- Sistemas basados en casos o CBR (*Case Based Reasoning*).
- Sistemas basados en redes bayesianas y cadenas de Markov.

Los sistemas basados en reglas, generalmente apoyados por mecanismos de inferencia de lógica difusa, se basan en la aplicación de reglas heurísticas sobre sistemas deterministas en los que se puede establecer reglas de relación entre los diferentes objetos.

Este tipo de sistemas trabajan mediante la aplicación de reglas y comparación de resultados. Además en caso de que la aplicación de una regla genere una nueva condición contemplada en el conjunto de reglas, dará lugar a la aplicación de esta nueva regla. Pueden trabajar por inferencia o lógica dirigida, bien empezando con una evidencia inicial en una determinada situación y dirigiéndose hacia la obtención de una solución, o bien estableciendo una hipótesis sobre las posibles soluciones y volviendo

hacia atrás para encontrar una evidencia existente (o una deducción de una evidencia existente) que apoya una hipótesis en particular. Las reglas de tipo “*si..(condición).. entonces..(consecuencia)..*” son el principal tipo de conocimiento usado en Sistemas Expertos.

El razonamiento basado en casos es el proceso de solucionar nuevos problemas basándose en las soluciones de problemas anteriores, de esta manera se obtiene la solución a un problema a partir de la búsqueda de problemas que han aparecido con anterioridad y cuya solución se adapta al nuevo problema. El razonamiento basado en casos es una manera de razonar haciendo analogías.

Por último, los sistemas basados en redes bayesianas, utilizan la probabilidad y razonamiento (inferencia) de tipo probabilístico. Una red bayesiana consiste en un gráfico acíclico dirigido y constituye un modelo gráfico probabilístico (modelo estático) que representa un conjunto de variables aleatorias y sus dependencias condicionales.

2.1.4 Ventajas de uso de los sistemas expertos

El uso de sistemas expertos puede aportar numerosas ventajas, tanto a la hora de automatizar los procesos como en comparación con el rendimiento que podemos obtener de un operador humano. A continuación pasaremos a citar algunas de las ventajas que presentan este tipo de sistemas:

1. Personal con poca experiencia es capaz de resolver una problemática que normalmente requeriría de un experto de mayor nivel. Esto es especialmente importante en áreas donde no se da abundancia de personal experto en la materia.
2. El conocimiento de varios expertos humanos se puede combinar, de manera que los procedimientos de los sistemas expertos son más fiables.
3. Tiempo de respuesta mucho menor que el que tomaría a un experto o conjunto de expertos humanos. Esta característica resulta importante en contextos críticos donde el tiempo de respuesta es una variable crítica.
4. No desarrollan apreciaciones subjetivas o emocionales ni cometen errores de cálculo.
5. En ocasiones la complejidad de los sistemas y número de variables a tener en consideración es tal que un experto humano presentaría gran dificultad para efectuar un análisis de manera fiable. En estos contextos los sistemas expertos son mucho más fiables que los expertos humanos.

6. Se pueden utilizar para realizar tareas monótonas, aburridas o poco reconfortables que normalmente causarían fatiga y un pobre desempeño en el personal humano.
7. Ahorro económico, aunque el coste de despliegue puede ser elevado, a la larga la inversión se amortiza. Los sistemas expertos están disponibles de manera ininterrumpida ofreciendo su máximo desempeño. Además tienen una vida de servicio en principio ilimitada y se pueden replicar de manera sencilla.

En resumen las características de los sistemas expertos se pueden resumir en:

- I. Permanencia.
- II. Replicación.
- III. Rapidez.
- IV. Bajo costo.
- V. Fiabilidad.
- VI. Consolidación del conocimiento adquirido.

2.1.5 Limitaciones de los sistemas expertos

A pesar de las múltiples ventajas del uso de sistemas expertos, existen también una serie de limitaciones que resultaría interesante comentar:

- I. A diferencia de los seres humanos, carecen de sentido común, y a no ser que se especifique de manera explícita podrían dar por válida información errónea o incompleta. Por ejemplo, podrían asumir a partir de la sintomatología que presenta un hombre la posibilidad de que dichos síntomas correspondan a un embarazo.
- II. Dificultad para aprender de errores propios o ajenos. Por lo general en un sistema experto deben especificarse los cambios que deseamos vía la reprogramación del mismo.
- III. Carecen de una perspectiva global del problema que tratan de resolver, tienen dificultad para discernir que es relevante y que secundario.
- IV. Ausencia de flexibilidad, un humano es sumamente más flexible a la hora de aceptar datos. Los sistemas expertos por lo general son incapaces de manejar conocimiento poco estructurado.

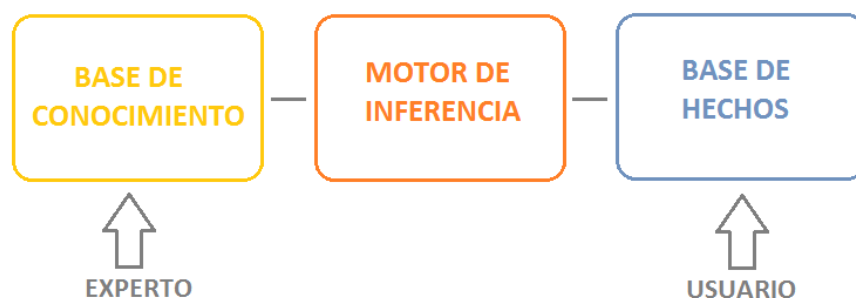
- V. Problemas ante la aparición de situaciones ambiguas. Ante la aparición de una situación no prevista el sistema no será capaz de tomar una decisión que vaya más allá de lo puramente racional, ya que carecen de cualquier mecanismo de improvisación.

2.1.6 Arquitectura de los sistemas expertos

La arquitectura básica de cualquier sistema experto está constituida por tres componentes principales:

1. La base de Conocimiento
2. Motor de Inferencia
3. Base de Hechos

Ilustración 2: Arquitectura de los Sistemas Expertos



Base de conocimiento (*Knowledge Base KB*)

Es el conjunto de conocimiento adquirido y codificado según una notación específica que incluye reglas, predicados, redes semánticas y objetos.

Las bases de conocimiento son la evolución lógica de los sistemas de bases de datos, en un intento de plasmar elementos de conocimiento (hechos y reglas), así como la forma en que estos deben ser empleados. Es decir, deben almacenar además de los hechos, las reglas que se servirán de dichos hechos para deducir información que no está almacenada de manera explícita. Para ello hacen uso de los KBMS o *Knowledge Base Management Systems*, equivalentes a los sistemas de gestión de bases de datos.

Generalmente los especialistas son los encargados de suministrar a los ingenieros de conocimiento una base de conocimientos estructurada y ordenada, así como las relaciones definidas y adecuadamente especificadas. De esta forma los expertos deben organizar y estructurar la base de conocimiento, generalmente como resultado de esto, el especialista aumenta su nivel de conocimiento técnico en el propio campo.

En este punto, cabe diferenciar entre el conocimiento y los datos. El conocimiento trata sobre las afirmaciones que tienen validez general y permanente, como por ejemplo las reglas, distribuciones de probabilidad, etc. Mientras que los datos son la información específica relacionada con un problema en particular y no forman parte permanente del sistema, es decir, son descartados después de haber sido usados.

De esta manera el conocimiento adquirido se almacena en la base de conocimiento, mientras que los datos se almacenan en la memoria de trabajo.

Motor de inferencia

La función de esta componente es sacar conclusiones aplicando el conocimiento sobre los datos, es decir, actúa como modelo del proceso de razonamiento humano. El motor de inferencia combina los hechos con las consultas realizadas, utilizando la base de conocimiento, selecciona los datos y pasos apropiados para representar los resultados. Las conclusiones las puede obtener mediante la aplicación de procedimientos deterministas o probabilísticos. Además hay que tener en consideración que en algunos contextos los hechos (datos) no se conocen de manera totalmente certera, en este caso el motor de inferencia también deberá encargarse de propagar esta información que puede resultar aleatoria o difusa. En este caso el tratamiento de información bajo criterios de incertidumbre resulta más complejo.

Base de hechos

Contiene los hechos (datos) sobre un problema que se ha descubierto durante la fase de análisis. Se trata de una memoria auxiliar que contiene los datos sobre una situación concreta en un instante determinado, a partir del cual se van a extraer conclusiones. Contiene los hechos iniciales que modelan el problema, y los resultados intermedios que se van obteniendo a lo largo del procedimiento de deducción.

A menos que el usuario requiera su almacenamiento, esta información es descartada tras la obtención de las conclusiones.

Interfaz de Usuario

También denominada Sistema de Consulta. Gobierna la comunicación entre usuario y sistema. Su objetivo es permitir un diálogo en lenguaje *cuasi-natural* entre el usuario y la máquina. Para ello comunica las consultas de usuario con el motor de inferencia y le remite los resultados de la consulta, así como todos los pasos seguidos hasta llegar a la conclusión. Además, sirve de entrada de información en los casos en que el motor de inferencia requiera más información.

Componente Humana

Los sistemas expertos son el resultado de la colaboración de diferentes expertos humanos especialistas en un tema de estudio y los ingenieros de conocimiento.

Los expertos humanos deben recopilar y suministrar a los ingenieros de conocimiento todo el conocimiento adquirido y éstos últimos deben trasladarlo al sistema experto en un lenguaje que este pueda entender.

Subsistema de Adquisición de Conocimiento

Controla el flujo de nuevo conocimiento que se inserta en la base de conocimiento. El sistema determina que nuevo conocimiento se requiere, o en caso de recibir nuevas entradas verificar si dicho conocimiento es efectivamente nuevo, verifica su

verosimilitud, discernir si aporta valor y en su caso añadirlo a la base de conocimiento manteniendo un control de coherencia respecto a la información existente en el sistema. Este nuevo conocimiento puede ser insertado por un experto, por el ingeniero de conocimiento, o provenir directamente de sistemas de bases de datos, sensorización, etc.

Control de coherencia

De aparición más reciente, este subsistema controla la consistencia de la base de conocimiento y evita el flujo de entrada de información inconsistente en el sistema. Para ello debe analizar la entrada de datos por parte del personal humano y deberá informar a los mismos de las inconsistencias detectadas.

Subsistema de ejecución de órdenes

Permite al sistema iniciar acciones basadas en las conclusiones establecidas por el motor de inferencia. Además, puede reportar al usuario las conclusiones en base a las cuales toma la determinación de actuar en un sentido u otro a través del subsistema de explicación.

Subsistema de explicación

Permite seguir los razonamientos (inferencias) efectuadas por el motor de inferencia. Este subsistema es altamente útil para el experto durante la construcción y verificación de la coherencia de la base de conocimiento. Además es de utilidad para dar explicación al usuario de como se ha realizado el proceso deductivo.

Subsistema de aprendizaje

Una de las principales características de un sistema experto es la capacidad que este tiene de aprender, es decir, obtener experiencia a partir de los datos disponibles. Datos que pueden ser obtenidos por expertos o no expertos y que se pueden utilizar en el subsistema de adquisición de conocimiento o subsistema de aprendizaje.

Este aprendizaje es de dos tipos fundamentalmente:

- Aprendizaje estructural, relacionado con la estructura del conocimiento (reglas, distribuciones de probabilidad, etc.).
- Aprendizaje paramétrico, estimación de los parámetros necesarios para construir la base de conocimiento (estimación de frecuencias o probabilidades).

2.1.7 Sistemas Expertos actuales

En esta sección se enumeran y describen brevemente distintos frameworks y tecnologías disponibles para la construcción de sistemas expertos:

PROLOG: Es un lenguaje de programación lógica de propósito general asociado con la inteligencia artificial y lingüística computacional. Es un lenguaje declarativo basado en reglas. Su nombre deriva del anagrama Programación Lógica.

La sintaxis del lenguaje es la siguiente:

- Declarar hechos sobre objetos y sus relaciones.
- Hacer preguntas sobre objetos y sus relaciones.
- Definir reglas sobre objetos y sus relaciones.

CLIPS: A mediados de los años ochenta, la NASA requería el apoyo de Sistemas Expertos para el desarrollo de proyectos. Para ello elaboraron una serie de prototipos, pero los resultados obtenidos no fueron lo suficientemente buenos para cumplir con los requerimientos internos de calidad.

Uno de estos prototipos fue denominado CLIPS (*C Language Integrated Production System*) cuya principal característica era su capacidad para funcionar con otros sistemas existentes. Posteriores mejoras y ampliaciones han convertido CLIPS en un punto de referencia para el desarrollo de otros Sistemas Expertos.

JESS: El motor de reglas JESS es un proyecto que tuvo su origen en CLIPS pero que fue escrito enteramente en Java. Se desarrolló durante la década de los noventa en los Sandia National Laboratories de EEUU. Además implementa la especificación de referencia JSR94.

Drools es la implementación y ampliación del algoritmo Rete diseñado por el Dr. Charles L. Forgy en la Universidad Carnegie Mellon. Básicamente, su algoritmo consiste en una red de nodos interconectados con diferentes características que evalúan las entradas mediante la propagación de los resultados del siguiente nodo cuando hay coincidencias siguiendo el algoritmo de RETE. Drools ofrece herramientas de integración con Java, capacidad de escalabilidad y una división clara entre los datos y la lógica de dominio. También implementa la especificación JSR94.

Jena: Jena es un framework desarrollado en tecnología Java que incluye un motor de inferencia basado en normas, una API ontológica y un motor de búsqueda.

Jeops: añade encadenamiento hacia adelante y normas de producción de primer orden con el fin de facilitar el desarrollo de Sistemas Expertos mediante programación declarativa.

OpenCyc: OpenCyc es la versión de código abierto de la tecnología CyC (<https://es.wikipedia.org/wiki/Cyc>), esta constituye la más completa base de conocimientos generales del mundo e incluye un motor de razonamiento de sentido común.

2.2 Sistemas Basados en reglas

Los sistemas basados en reglas resultan especialmente indicados cuando tratamos con situaciones de cierta complejidad pero que pueden ser expresadas de forma determinista. Ejemplos de estas situaciones podrían ser los sistemas de seguridad, sistemas de control de tráfico, etc. en los que todos los estados están acotados y resultan fácilmente identificables.

En este tipo de sistema la base de conocimiento consistirá en el conjunto de reglas que definen el comportamiento del sistema, y el motor de inferencia aplicara lógica clásica para establecer conclusiones ante la entrada de los datos.

Dentro del campo de los sistemas expertos estos sistemas son los más sencillos.

2.2.1 La Base de Conocimiento en sistemas basados en reglas

En contextos deterministas las relaciones entre los diferentes conjuntos de objetos se pueden expresar en forma de reglas que modelen las relaciones entre objetos y que serán almacenadas en la base de conocimiento de manera permanente.

Una regla podría ser por ejemplo:

si nota ≥ 5 entonces calificación=aprobado

Como vemos, esta regla modela la relación entre dos objetos como son la nota y la calificación y se compone de dos partes diferenciadas:

- Premisa (entre "si" y "entonces") puede contener una o más afirmaciones *objeto-valor* conectadas con los operadores lógicos "y", "o", o "no".
- Conclusión: expresión lógica tras la palabra "entonces".

De esta forma una regla se puede definir de la siguiente manera:

"Una regla es una afirmación lógica que relaciona dos o más objetos e incluye dos partes, la premisa y la conclusión. Cada una de estas partes consiste en una expresión lógica con una o más afirmaciones objeto-valor conectadas mediante los operadores lógicos y, o, o no."

Como vemos, una regla se escribe normalmente como "Si premisa, entonces conclusión". Y tanto la premisa como la conclusión pueden contener múltiples afirmaciones objeto-valor. Una expresión lógica que contiene solo una afirmación objeto-valor se denomina expresión lógica simple, en caso contrario se la denomina expresión lógica compuesta.

Además algunos sistemas imponen ciertas restricciones sobre las reglas, como por ejemplo:

- No permitir en la premisa el operador lógico "o".
- Limitar las conclusiones a expresiones lógicas simples.

La razón por la que imponen estas restricciones es porque las reglas expresadas de esta forma son más fáciles de tratar y además no dan lugar a pérdida de generalidad, puesto que reglas más generales pueden ser reemplazadas por conjuntos de reglas de dicha forma. Esto es conocido como sustitución de reglas, y puede llevarse a cabo a partir de las reglas que introduce el experto humano para satisfacer las restricciones planteadas.

A continuación podemos ver una tabla con las sustituciones que se pueden aplicar en aras de simplificar la lógica de aplicación.

Tabla 1: Reglas de sustitución

Regla	Reglas Equivalentes
Si $A \text{ o } B$, entonces C	Si A , entonces C Si B , entonces C
Si $\overline{A \text{ o } B}$, entonces C	Si \overline{A} y \overline{B} , entonces C
Si $\overline{A \text{ y } B}$, entonces C	Si \overline{A} , entonces C Si \overline{B} , entonces C
Si $(A \text{ o } B) \text{ y } C$, entonces D	Si $A \text{ y } C$, entonces D Si $B \text{ y } C$, entonces D
Si $\overline{(A \text{ o } B) \text{ y } C}$, entonces D	Si $\overline{A} \text{ y } \overline{B} \text{ y } C$, entonces D
Si $\overline{A \text{ y } B} \text{ y } C$, entonces D	Si $\overline{A} \text{ y } C$, entonces D Si $\overline{B} \text{ y } C$, entonces D
Si A , entonces $B \text{ y } C$	Si A , entonces B Si A , entonces C
Si A , entonces $B \text{ o } C$	Si $A \text{ y } \overline{B}$, entonces C Si $A \text{ y } \overline{C}$, entonces B
Si A , entonces $\overline{B \text{ y } C}$	Si $A \text{ y } B$, entonces \overline{C} Si $A \text{ y } C$, entonces \overline{B}
Si A , entonces $\overline{B \text{ o } C}$	Si A , entonces \overline{B} Si A , entonces \overline{C}

Tabla 2: Tabla de verdad mostrando igualdad lógica de expresiones

A	B	\overline{A}	\overline{B}	$\overline{A \text{ o } B}$	$\overline{A} \text{ y } \overline{B}$
C	C	F	F	F	F
C	F	F	C	F	F
F	C	C	F	F	F
F	F	C	C	C	C

2.2.2 El motor de inferencia

Como comentábamos anteriormente, existen dos tipos de elementos: datos (hechos o evidencias) y el conocimiento (conjunto de reglas almacenado en la base de conocimiento). El motor de inferencia usa estos para obtener nuevas conclusiones o hechos. Si la premisa de una regla es cierta la conclusión que se extrae de la misma debe ser cierta también.

Además, las conclusiones pueden clasificarse en dos tipos:

- Conclusiones simples, resultado de una regla simple
- Conclusiones compuestas, resultado de la aplicación de más de una regla.

Para obtener estas conclusiones los sistemas expertos utilizan diferentes tipos de reglas y estrategias que pasamos a detallar a continuación:

❖ *Modus Ponens*

❖ *Modus Tollens*

❖ Resolución

y las estrategias de inferencia:

❖ Encadenamiento de reglas

❖ Encadenamiento de reglas orientado a un objetivo

❖ Compilación de reglas

i. *Modus Ponens* o *Modus Tollens*

El *Modus Ponens* es probablemente la regla de inferencia más utilizada para obtener conclusiones simples. Este método examina la premisa de una regla y de ser cierta la conclusión pasa a formar parte del conocimiento.

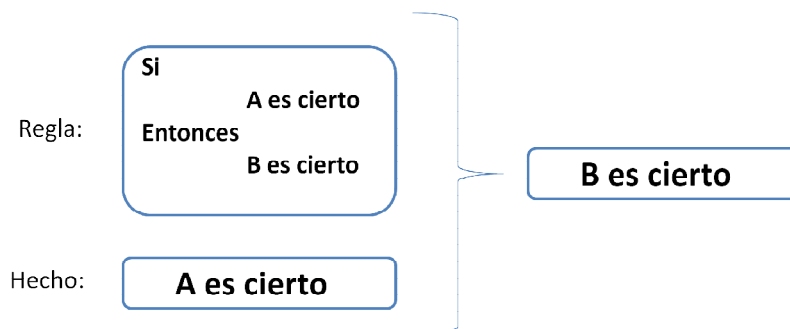
Esto se puede ilustrar con el siguiente ejemplo:

"Si *A* es cierto, entonces *B* es cierto"

y además conocemos que "*A* es cierto"

de modo que se puede concluir que "*B* es cierto" también.

Ilustración 3: Mecanismo de inferencia *Modus Ponens*



Esta regla tan sencilla, que parece trivial, es la base de muchos sistemas expertos.

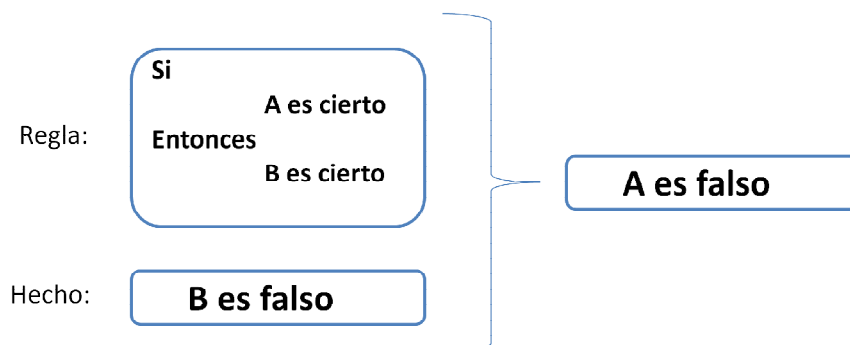
El **Modus Tollens** en cambio examina la conclusión y si es falsa, se concluye que la premisa resulta también falsa. Por ejemplo:

“Si **A** es cierto, entonces **B** es cierto”

pero se sabe que : “**B** es falso”

de manera que se puede concluir que : “**A** es falso”

Ilustración 4: Mecanismo de inferencia Modus Tollens



Como vemos, la regla *Modus Ponens* opera hacia adelante, es decir, desde la premisa de una regla a la conclusión de la misma, mientras que la regla *Modus Tollens* se mueve hacia atrás, desde la conclusión a la premisa.

Estas dos reglas se complementan en los sistemas expertos para expandir la capacidad de inferencia a partir de la base de conocimiento.

ii. Mecanismo de resolución

Las reglas de inferencia *Modus Ponens* y *Modus Tollens* pueden ser utilizadas para obtener conclusiones simples. Pero para poder obtener conclusiones compuestas que se basan en múltiples reglas, necesitamos hacer uso de un mecanismo de resolución. Este proceso involucra diferentes etapas de simplificación y combinación.

El mecanismo de resolución presenta las siguientes etapas:

1. Las Reglas son sustituidas por expresiones lógicas equivalentes.
2. Estas expresiones lógicas se combinan en otra expresión lógica.
3. Esta última expresión se utiliza para obtener la conclusión.

iii. Encadenamiento de reglas

Es uno de los mecanismos más utilizados para obtener conclusiones compuestas. Esta estrategia puede utilizarse cuando las premisas de ciertas reglas coinciden con las conclusiones de otras. Cuando se encadenan las reglas, las conclusiones sobre los

hechos dan lugar a nuevos hechos hasta que la iteración termina y no pueden encontrarse más conclusiones. El tiempo que toma este proceso depende por una parte de los hechos conocidos y de las reglas que se van activando.

El algoritmo que se sigue es el siguiente:

- **Datos de entrada:** Una base de conocimiento (objetos y reglas) y algunos hechos iniciales.
 - **Resultado:** El conjunto de hechos derivados lógicamente de ellos.
1. Asignar a los objetos sus valores conocidos tales como los dan los hechos conocidos o la evidencia.
 2. Ejecutar cada regla de la base de conocimiento y concluir nuevos hechos si es posible.
 3. Repetir la Etapa 2 hasta que no puedan ser obtenidos nuevos hechos.

iv. Encadenamiento de Reglas Orientado a un Objetivo

El algoritmo de encadenamiento de reglas orientado a un objetivo requiere de un usuario que previamente seleccione una variable o nodo objetivo, para permitir al algoritmo navegar a través de las reglas en busca de una conclusión para dicho objetivo.

En caso de que no se obtenga ninguna conclusión con la información existente, el algoritmo preguntará al usuario por nueva información de los elementos relevantes para obtener información sobre el objetivo.

Los algoritmos de encadenamiento se suelen conocer como encadenamiento hacia adelante, mientras que los de encadenamiento orientado a objetivo como encadenamiento hacia atrás, aunque ambos pueden hacer uso de las dos reglas de inferencia *Modus Ponens* y *Modus Tollens* que describíamos con anterioridad.

A continuación, presentamos el algoritmo de encadenamiento orientado a un objetivo:

- **Datos de entrada:** Una base de conocimiento (objetos y reglas), algunos hechos iniciales y un nodo o variable objetivo.
 - **Resultado:** El valor del nodo o variable objetivo.
1. Asigna a los objetos sus valores conocidos tal y como están dados en los hechos de partida. Marcar todos los objetos cuyo valor ha sido asignado.

Si el nodo objetivo está marcado, ir a la Etapa 7; en otro caso:

- a) Designar como objetivo inicial el objetivo en curso.
- b) Marcar el objetivo en curso.
- c) Sea $Objetivos\ Previos = \emptyset$, donde \emptyset es el conjunto vacío.
- d) Designar todas las reglas como activas (ejecutables).
- e) Ir a la Etapa 2.

2. Encontrar una regla activa que incluya el objetivo en curso y ninguno de los objetos en Objetivos Previos. Si se encuentra una regla, ir a la Etapa 3; en otro caso, ir a la Etapa 5.
3. Ejecutar la regla referente al objetivo en curso. Si concluye, asignar el valor concluido al objetivo en curso, e ir a la Etapa 6; en otro caso, ir a la Etapa 4.
4. Si todos los objetos de la regla están marcados, declarar la regla como inactiva e ir a la Etapa 2; en otro caso:
 - a) Añadir el objetivo en curso a Objetivos Previos.
 - b) Designar uno de los objetos no marcados en la regla como el objetivo en curso.
 - c) Marcar el objetivo en curso.
 - d) Ir a la Etapa 2.
5. Si el objetivo en curso es el mismo que el objetivo inicial, ir a la Etapa 7; en otro caso, preguntar al usuario por el valor del objetivo en curso. Si no se da un valor, ir a la Etapa 6; en otro caso asignar al objeto el valor dado e ir a la Etapa 6.
6. Si el objetivo en curso es el mismo que el objetivo inicial, ir a la Etapa 7; en otro caso, designar el objetivo previo como objetivo en curso, eliminarlo de Objetivos Previos e ir a la Etapa 2.
7. Devolver el valor del objetivo en curso si es conocido.

Como conclusión simplemente resumir en que las estrategias de encadenamiento de reglas se utilizan en problemas en los que algunos hechos son conocidos y se buscan algunas conclusiones. Mientras que las estrategias de encadenamiento de reglas orientadas a un objetivo se utilizan en problemas en que el objetivo es conocido, y se buscan los hechos que hacen que esta condición sea posible.

Un ejemplo para el primer caso sería, a partir de unos síntomas de un paciente deducir la enfermedad que padece; mientras que para el segundo tipo sería a partir de la enfermedad encontrar los síntomas que debería presentar.

v. Compilación de reglas

Otra forma de tratar con reglas encadenadas consiste en comenzar con un conjunto de datos y tratar de alcanzar algunos objetivos. Cuando tanto los datos como los objetivos se han determinado previamente, las reglas pueden ser compiladas, es decir, pueden escribirse los objetivos en función de los datos, de manera que se obtienen un conjunto de ecuaciones objetivo.

2.2.3 Control de Coherencia

Un mecanismo importante dentro de los sistemas expertos es el mantener un control sobre la coherencia de los datos que se presentan al sistema, ya que incluso los expertos humanos pueden dar información inconsistente ante situaciones complejas.

Por ejemplo, reglas no consistentes o combinaciones de hechos no factibles. Por ello resulta esencial controlar y asegurar la coherencia del conocimiento tanto durante la construcción de la base de conocimiento como durante el mantenimiento de la misma.

Si el conocimiento que posee el sistema está sujeto a inconsistencias las conclusiones que puede obtener podrían ser poco satisfactorias o absurdas.

El control de la coherencia pretende ayudar a que los usuarios no inserten en la base de conocimiento información inconsistente o contradictoria, y para ello puede demandar a los mismos una serie de restricciones a cumplir. De esta manera, se mantendrá tanto una coherencia de reglas como coherencia de hechos.

Coherencia de reglas

Definición de coherencia para un conjunto de reglas:

"Un conjunto de reglas se denomina coherente si existe, al menos, un conjunto de valores de todos los objetos que producen conclusiones no contradictorias."

De esta manera, un conjunto coherente de reglas no tiene por que devolver conclusiones no contradictorias para todos los posibles conjuntos de valores de los objetos. Es decir, es condición suficiente que exista un conjunto de valores que conduzcan a conclusiones no contradictorias.

Los conjuntos de valores que llevan a conclusiones inconsistentes aunque las reglas que los definan sean coherentes se denominan valores no factibles.

Definición factibilidad:

"Se dice que un valor a para el objeto A no es factible si las conclusiones obtenidas al hacer $A = a$ contradicen cualquier combinación de valores del resto de los objetos. Por ello, cualquier valor no factible debe ser eliminado de la lista de valores posibles de su correspondiente objeto para eliminar la posibilidad de que el motor de inferencia pueda obtener conclusiones inconsistentes."

De esta forma podemos comprobar que es suficiente con realizar la comprobación de coherencia de una regla cuando esta es introducida, y que todos los valores no factibles pueden ser eliminados de sus correspondientes listas cuando son detectados. Si la regla es consistente con el resto de reglas que forman parte de la base de conocimiento, la nueva regla se añadirá a la misma, y en otro caso se devolverá al experto humano para su análisis y en su caso corrección.

Coherencia de hechos

Además de las reglas, los datos o evidencias suministrados por los usuarios deben también ser consistentes entre sí y con respecto a las reglas de la base de conocimiento. Por ello, el sistema no debe aceptar hechos que contradigan las reglas.

La coherencia de hechos se logra mediante la siguiente estrategia:

Eliminar todos los valores no factibles (los que contradicen el conjunto de reglas y/o hechos) de los objetos una vez detectados. Al usuario se le preguntara por los valores de

los objetos y el sistema experto deberá aceptar únicamente objetos consistentes con las reglas y conocimiento previo:

- El motor de inferencia debe comprobar que los hechos conocidos no contradicen el conjunto de reglas.
- Suministrar al usuario una lista de objetos a los que no se ha asignado valores previamente.
- Para cada uno de los objetos, mostrar y aceptar sólo sus valores factibles.
- Actualizar continuamente la base de conocimiento y eliminar los valores no factibles. El conocimiento debe ser actualizado inmediatamente tras la incorporación de cada hecho.

2.2.4 Explicación de Conclusiones

Tal y como vimos en el apartado anterior, con los sistemas expertos no basta con que nos devuelvan una conclusión, sino que normalmente los usuarios esperan que el sistema otorgue algún tipo de explicación sobre por qué se debe tomar una decisión en un sentido u otro.

Durante el proceso que realiza el motor de inferencia, las reglas activas (aplicadas) forman parte de la base del mecanismo de explicación. En los sistemas expertos basados en reglas devolver una explicación sobre el razonamiento resulta muy sencillo, ya que el motor de inferencias obtiene conclusiones basándose en un conjunto de reglas y, por tanto, conoce qué regla hay asociada a cada conclusión. Por ello, el sistema puede listar los hechos concluidos junto con las reglas que ha utilizado para establecer tal conclusión.

2.3 Bussiness Rules Management Systems (BRMS).

Los **Sistemas de Gestión de Reglas de Negocio (BRMS)** se han convertido en una tecnología clave en la gestión de organizaciones complejas, ya que permite separar las reglas de negocio del código de las múltiples aplicaciones que deben utilizarlas.

Para ello se basan en la misma tecnología que los sistemas expertos, siendo un subtipo de los mismos, pero con unas características que los hacen especialmente indicados para dirigir los procesos de gestión.

2.3.1 ¿Qué es un BRMS?

Un BRMS (*Bussiness Rules Management System*) o Sistema de Gestión de Reglas de negocio, es un sistema software usado para modelar y definir, desplegar, ejecutar, así como monitorizar y mantener las reglas de negocio que se acumulan a nivel de organización. De esta manera la lógica o reglas de negocio se aíslan del código de la aplicación que las maneja, que puede ser desde bases de datos, programas o sistemas de explotación, etc.

Para ello deben proveer de herramientas que soporten el cambio que se aplica sobre los procesos de negocio. Las reglas deben poder ser administradas desde un entorno multiusuario. Sobre una interfaz amigable para los usuarios que permita que los analistas del negocio puedan ver y entender las reglas sin tener que depender del departamento de informática para realizar las modificaciones que consideren.

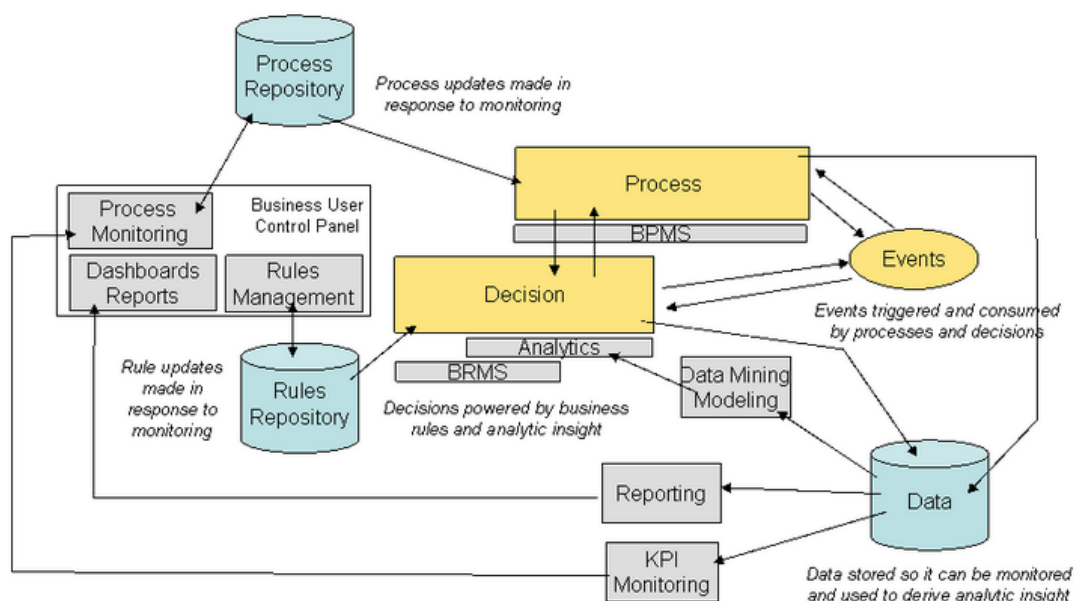
Gracias a los sistemas BRMS, los analistas del negocio pueden determinar y reescribir la lógica del negocio. Todo lo que se necesita es codificar las condiciones en una regla.

Normalmente, una regla de negocio se expresa con un: "Si xxx" -> "Entonces yyy", es decir, "Si" existen las condiciones "xxx", "Entonces" deben ejecutarse las acciones "yyy". Además se puede establecer relación entre las diferentes reglas.

Con los sistemas BRMS evitamos la codificación de las reglas de negocio dentro del código de la aplicación, dificultando su acceso y modificación e incrementando los costes de mantenimiento del software. Y permitimos realizar una gestión independiente.

El **BRMS** se ha convertido en la tecnología clave para las arquitecturas orientadas a servicios (**SOA**), la gestión de los procesos de negocio (**BPM**) y otras iniciativas tecnológicas innovadoras.

Ilustración 5: Esquema estructura BRMS



La tecnología *BRMS* permite a los usuarios determinar y escribir la lógica del negocio mediante el establecimiento de las diferentes reglas de negocio. De esta manera son los usuarios lo que directamente pueden definir, representar, ejecutar, monitorizar y actualizar las reglas de negocio. Estas reglas después serán utilizadas por las aplicaciones que realizan la toma de decisiones de manera automatizada.

Con esta descentralización de la lógica respecto a las aplicaciones se permite otorgar mayor control a los analistas de negocio obteniendo mayor independencia del personal de *TI*.

Un *BRMS* para que sea considerado como tal debe cumplir los siguientes requerimientos:

- ✓ Disponer de un repositorio externo para la edición, creación, organización e interpretación de las reglas de negocio y que sea independiente de la aplicación.
- ✓ Proveer de herramientas para que los expertos técnicos y los expertos de negocio puedan definir y administrar la lógica de negocio.
- ✓ Realizar diferentes procesos de comprobación, validación y simulación.
- ✓ Disponer de un entorno que permita la ejecución de las decisiones tomadas a partir del motor de reglas de negocio.

2.3.2 ¿Qué son las reglas de negocio?.

Las reglas del negocio describen las políticas, normas, operaciones, definiciones y restricciones presentes en una organización y que son de vital importancia para alcanzar los objetivos que está se propone.

Las reglas de negocio representan la lógica principal de funcionamiento del negocio e indican lo que una organización “*debe hacer*” para alcanzar sus objetivos, transformándose de este modo en un medio fundamental para la implementación de la **Estrategia Organizacional** y un componente clave en la toma de decisiones.

Una regla de negocio define o restringe algún aspecto del negocio, y solamente puede ser evaluada a verdadero o falso. Las reglas de negocio tienen como objetivo controlar o influir en el proceso de negocio, describiendo las restricciones, definiciones y operaciones que se llevan a cabo en la organización. Además, se pueden aplicar tanto a personas, procesos, sistemas computacionales y su objetivo es asistir en la consecución de los objetivos de la organización.

En esencia, las reglas de negocio constituyen abstracciones de las políticas y prácticas que llevan a cabo las organizaciones y suponen una formalización de las mismas en un lenguaje que tanto los administradores como el personal de IT puede comprender de una manera no ambigua.

Ejemplos de este tipo de reglas podrían ser por ejemplo: las condiciones que debe cumplir un cliente para que se le conceda un préstamo o hipoteca, o las condiciones que se dan para calificar un intento de acceso a un sistema como abusivo.

Las organizaciones funcionan siguiendo múltiples reglas de negocio, explícitas o implícitas, que están embebidas en los procesos, aplicaciones informáticas, documentos, etc. Además se pueden encontrar en la cabeza de algunas personas, expertos en la materia, o pertenecer al código fuente de las aplicaciones software que estos manejan.

Debido a que constituyen uno de los componentes más versátiles de cualquier organización, es indispensable para los expertos de negocio contar con las herramientas adecuadas para la formulación, verificación, validación y gestión de las reglas de negocio.

Desde algunos años atrás se puede observar una fuerte tendencia en el crecimiento de los sistemas de gestión centralizada a través de **Sistemas de Gestión de Reglas de Negocio** (*Business Rules Management System o BRMS*).

Para que una regla constituya una buena regla de negocio debe cumplir con los siguientes puntos:

1. Debe ser declarativa.
2. Atómica.
3. Construida de manera independiente.
4. Expresada en lenguaje natural.
5. Orientada al negocio.

2.3.3 Tipos de reglas de negocio

De acuerdo con el **Business Rules Group** (<http://www.businessrulesgroup.org/>) una regla de negocio puede entrar en una de las siguientes categorías:

- **Definiciones (de los términos de negocio).** El elemento más básico de una regla de negocio es el lenguaje usado para expresarla. La propia definición de un término constituye una regla de negocio que describe como es el objeto de negocio.
Los términos en las organizaciones tradicionalmente son documentados en los glosarios o como entidades del modelo conceptual.
- **Hechos (relacionan unos términos con otros).** La naturaleza de las actividades operativas de una organización puede ser descrita en forma de relaciones entre términos. Decir por ejemplo que un cliente puede hacer un pedido es una regla de negocio pero también un hecho. Los hechos son documentados como expresiones en lenguaje natural, relaciones, atributos, etc.
- **Restricciones.** Cualquier organización tiene una serie de restricciones que debe cumplir. Por ejemplo valores que no son válidos sobre los datos. Prevenir que una entrada no tenga un valor incorrecto puede ayudar a que acciones no deseadas tengan lugar.
- **Derivados.** Las reglas de negocio definen la forma en que el conocimiento se transforma en otro conocimiento, de la misma u otra naturaleza.

2.3.4 Ventajas e inconvenientes de los sistemas basados en reglas

La principal ventaja de los sistemas basados en reglas es que las reglas resultan mucho más sencillas de comprender que el código de aplicación. Esto es gracias al uso de la programación declarativa, que permite especificar que hay que hacer, abstrayendo al personal no técnico del cómo se hace. Los sistemas basados en reglas son capaces de resolver problemas complejos, proveyendo de una explicación de cómo se ha llegado a la solución, es decir permiten realizar trazabilidad de las decisiones tomadas.

De esta manera, las principales ventajas de estos sistemas son:

Aumento de la mantenibilidad de las aplicaciones: el mantener en repositorios separados la lógica respecto a los datos o la base de conocimiento aumenta la mantenibilidad del sistema, ya que de esta forma se mantiene de forma independiente toda la lógica de la aplicación respecto a la propia aplicación.

Mayor escalabilidad y velocidad: los algoritmos de *patern matching* que utilizan los sistemas expertos, como el algoritmo de Rete, proveen de un marco de ejecución altamente eficiente.

Centralización del conocimiento: mediante las reglas se puede crear un repositorio centralizado de conocimiento o base de conocimiento. De esta manera, dispondremos de un único punto donde se almacenan todos los hechos que son calificados como "verdad" dentro de la organización. Además, dichas reglas pueden ser reevaluadas en cualquier momento, para comprobar su validez.

Idealmente las reglas son expresadas en un lenguaje de alto nivel, de manera que pueden ser empleadas para generar la documentación del sistema.

Disponibilidad de herramientas de integración: a través de interfaces web o herramientas de desarrollo como *Eclipse*, que permiten editar, administrar, validar e incluso algunas ofrecen funciones para el control de versiones.

Proveen de mecanismos de explicación: trazabilidad (*log*) de las decisiones tomadas y la forma cómo se llegó a tal conclusión.

Modelado: las reglas son comprensibles a través de los diferentes modelos de objetos y permiten la creación de lenguajes específicos de dominio (*Domain Specific Languages o DSL*), que permiten expresar las reglas en un lenguaje muy próximo al natural. De manera que el personal menos técnico pueda comprender la naturaleza de las mismas, abstrayéndose de cuáles son los detalles de implementación.

De esta manera, los grandes beneficios del uso de BRMS incluyen:

- Unificación de la lógica de negocio.
- Menor coste debido a cambios en la lógica de negocio.
- Disminución de tiempo de desarrollo.
- Las reglas se pueden externalizar y compartir en múltiples aplicaciones.
- Los cambios se pueden aplicar de manera más rápida y con menor riesgo.
- Menor dependencia del personal técnico para hacer cambios en el sistema.
- Automatización y mejora en la eficiencia de los procesos de negocio.

En definitiva, las reglas de negocio representan un paso adelante en la aplicación de técnicas informáticas para la mejora de la eficiencia y productividad en el ámbito empresarial. Los sistemas de reglas de negocio aumenta la agilidad respecto al cambio (*business agility*) y la gestión de procesos.

Por último, destacar algunas de las desventajas de los BRMS:

- El personal técnico debe ser instruido y conocer en profundidad el dominio de problema.
- El sistema de reglas debe poder ser integrado en cualquier aplicación que utiliza la organización.
- Se alarga el ciclo de desarrollo debido a la necesidad de recopilación, verificación, migración de reglas y mantener un versionado de las mismas.
- Aunque idealmente se debería reducir la dependencia sobre el personal técnico esto en la práctica no suele darse. Esto se debe sobre todo a los constantes cambios que se aplican sobre la base de las propias reglas o sobre los modelos que se utilizan.

2.3.5 Cuándo usar un BRMS

Son múltiples los contextos en los que puede resultar de utilidad utilizar un BRMS para la gestión de la lógica y reglas de negocio, como por ejemplo:

- I. Sistemas en los que la lógica de las reglas de negocio varía de manera recurrente.
- II. Múltiples usuarios tienen necesidad de acceso a un repositorio común de reglas de negocio.
- III. Necesidad de infraestructura de administración de las diferentes reglas de negocio.
- IV. Múltiples reglas de negocio que se aplicaran para desarrollar las reglas técnicas que formarán parte de la aplicación.
- V. Los analistas de negocio no tienen gran formación técnica, de esta forma podemos expresar las reglas lógicas de negocio bajo sus propios términos.

2.3.6 El motor de reglas de negocio.

En los últimos años se viene observando una creciente tendencia a gestionar de forma sistemática y centralizada las reglas de negocio, de modo que resulte más sencillo consultarlas, entenderlas, utilizarlas, cambiarlas, etc. Para ello resulta recomendable utilizar un motor de reglas de negocio.

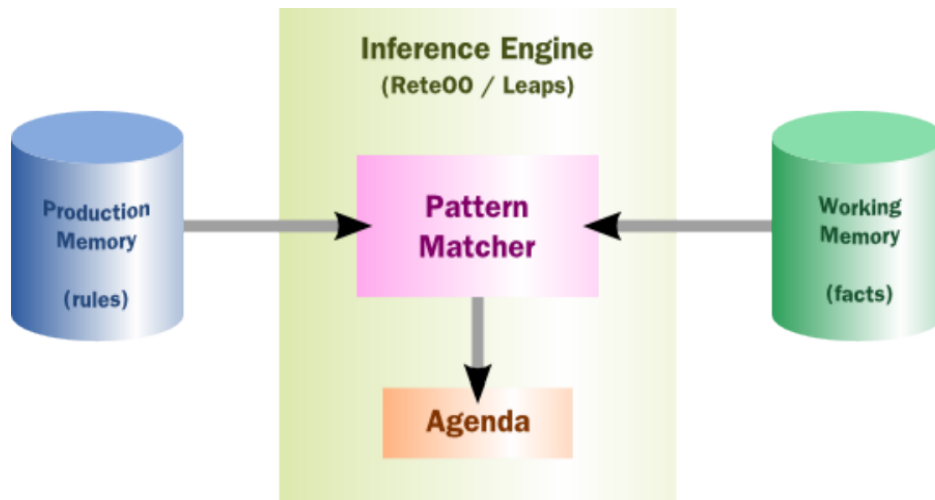
El motor de reglas de negocio es un sistema que se configura para dar servicio a las necesidades de negocio a través de la definición de objetos y reglas de negocio. Este software se rige por flujos que derivan responsabilidades a los distintos cargos de la empresa repartiendo así el trabajo equitativamente y cuantitativamente.

Los motores de reglas de negocio generalmente soportan reglas, hechos, objetivos, exclusión mutua, precondiciones así como otras funciones más avanzadas, y normalmente se distribuyen embebidos como un componente de un sistema completo de gestión empresarial.

El proceso de encontrar qué hechos encajan con las reglas de producción se llama *pattern matching*, y se lleva a cabo por el motor de inferencia. De esta manera, podemos ejecutar diferentes acciones en respuesta a los cambios sobre los datos que nos llegan. Estas acciones sobre los datos pueden transformar los mismos, y esta modificación hacer que encajen con otras reglas, desencadenando una nueva respuesta.

Las reglas se almacenan en la memoria de producción (o **Production Memory**) y los hechos que hacen *match* en el motor de inferencia son guardados en la memoria de trabajo (**Working Memory**). Los hechos insertados en la memoria de trabajo deben poder ser modificados o extraídos. Cuando un hecho hace *match* (encaja) en múltiples reglas, estas reglas se dice que entran en conflicto. La "**Agenda**" gestiona el orden de ejecución de estas reglas que entran en conflicto a través de una estrategia de resolución de conflictos.

Ilustración 6: Visión de alto nivel de un Sistema de Producción de Reglas



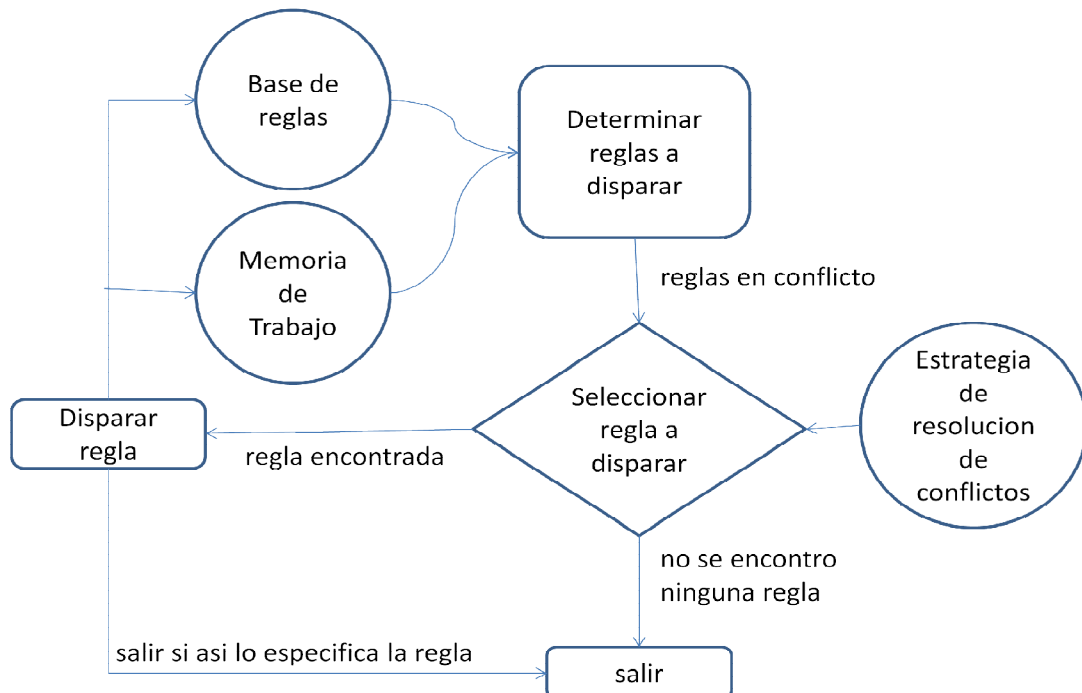
2.3.7 Forward Chaining, Backward Chaining y Sistemas de razonamiento Híbrido

Existen dos mecanismos básicos de inferencia: encadenamiento adelante (*forward chaining*), reactivo y orientado a los datos; y encadenamiento hacia atrás (*backward chaining*), enfocado a consultas pasivas.

A continuación pasamos a explicar cómo operan ambos enfoques:

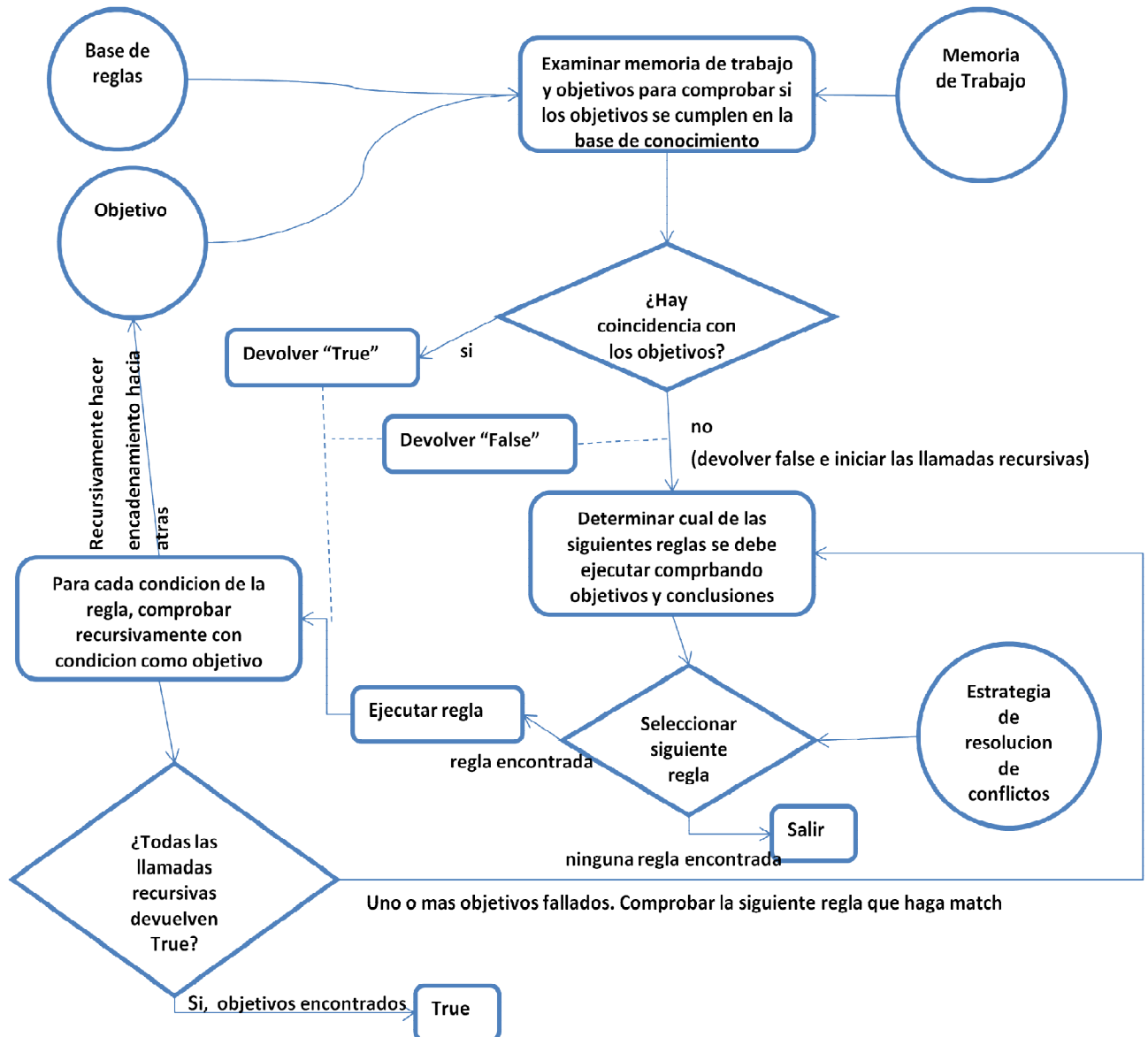
El encadenamiento hacia adelante es orientado a datos (*data-driven*), y de esta forma reactivo a los mismos. Los hechos son insertados en la memoria de trabajo, de manera que se evalúan con las reglas para comprobar la condición de *matching*. En caso de que los datos hagan encajen en alguna regla (o más de una), pasan a formar parte de la *Agenda* de ejecución. Los hechos de esta forma son propagados hasta establecer una conclusión.

Ilustración 7: Esquema mecanismo de Forward Chaining



El encadenamiento hacia atrás, por contra, es orientado a objetivos (*goal-driven*), esto significa que empezamos con una conclusión que el motor de inferencia trata de satisfacer. Si no puede, entonces busca conclusiones que si pueda satisfacer. Estas condiciones se conocen como subobjetivos (*subgoals*), que ayudarán a satisfacer de manera parcial una parte conocida del objetivo global. De esta manera, el proceso continúa hasta que se comprueba la condición inicial o no quedan más subobjetivos. Esta es la forma como opera *Prolog*, por ejemplo.

Ilustración 8: Mecanismo de Backward Chaining



Históricamente uno tendría que elegir entre el enfoque de encadenamiento adelante de sistemas (como OPS5) o encadenamiento hacia atrás (Prolog). Pero los sistemas modernos proveen de ambos tipos de razonamiento de manera simultánea.

Además, existen muchas otras técnicas que aumentan los modos de razonar sobre los datos: lógica borrosa (*fuzzy logic*), sistemas de creencia, factores de certeza,

razonamiento temporal, correlación, etc. Los sistemas modernos combinan estos diferentes enfoques y capacidades y constituyen lo que denominamos sistemas de razonamiento híbrido o HRS.

2.3.8 BRMS que podemos encontrar

Los sistemas BRMS que vamos a enumerar a continuación provienen principalmente de la comunidad *open-source*, pero no obstante existen gran número de desarrollos comerciales como **WebSphere ILOG** de IBM.

Pero nos vamos a centrar en los que podemos encontrar para la plataforma Java, que permite operar con múltiples entornos, como Windows, Linux, etc.

Drools es un motor de reglas orientado a objetos para Java, que ofrece una implementación del algoritmo Rete de Forgy, a medida para este lenguaje de programación.

La adaptación de Rete a una interfaz orientada a objetos permite una mayor expresión natural de reglas de negocio en lo que respecta a los objetos de negocio. Más importante aún, Drools establece la lógica de programación declarativa y es lo suficientemente flexible como para que coincida con la semántica de dominio del problema.

Jess es un motor de reglas también para la plataforma Java. Consiste en un superconjunto del lenguaje de programación *CLIPS*, desarrollado por Ernest Friedman de los Sandia National Labs. Desarrollado a finales de 1995. Utiliza programación basada en reglas para la automatización, es decir, requiere de la formalización de una serie de reglas que pueden modificar la colección de hechos, o ejecutar cualquier código Java.

Jess se puede utilizar para construir servlets Java, EJB, applets y aplicaciones completas que utilizan el conocimiento en forma de reglas declarativas para sacar conclusiones y hacer inferencias. El motor de reglas de Jess utiliza también el algoritmo de Rete.

Es *OpenSource* con licencia para estudiantes, pero como producto comercial es de pago.

JLisa es un potente framework para construir reglas de negocio. Es accesible con Java y compatible con JSR94 V, la API Java™ Rule Engine.

OpenRules es un sistema de propósito general de gestión de reglas y toma de decisiones, disponible como producto OpenSource. Permite a los expertos y desarrolladores crear, probar, ejecutar y mantener aplicaciones de soporte a la toma de decisiones.

Entre sus características destaca:

- Integración con Java o .NET.

- Permite hacer análisis predictivo.
- Las reglas se especifican a través de interfaz web.
- Resuelve diferentes tipos de restricciones y optimización.

Prova (*Prolog+Java*) es un sistema basado en reglas (*rule-based system*) para Java. Este lenguaje abre un nuevo camino en la combinación de la programación declarativa y expresiva. Combina la sintaxis natural y la tipificación de Java con las reglas de Prolog.

JEOPS plantea un gran avance en los motores basados en reglas de encadenamiento. Utiliza un motor de reglas para alimentar el proceso de negocio mediante la especificación de diferentes reglas.

2.4 El Framework Drools

Dentro de los sistemas de gestión de reglas de negocio más populares encontramos **Drools**, disponible para el lenguaje de programación Java.

Drools es un BRMS de **JBoss** que implementa la *Java Rule Engine API (JSR 94)* y utiliza una implementación mejorada del algoritmo de *Rete* para la ejecución de reglas bajo programación orientada a objetos POO.

2.4.1 ¿Qué es Drools?

Drools o **JBoss Rules** es un sistema experto basado en reglas, que actúa como sistema de gestión de reglas de negocio para la administración, almacenamiento, modificación e implementación de las mismas (*Bussines Rule Management System*).

Este framework utiliza un motor de reglas basado en inferencia con encadenamiento hacia adelante (*forward chaining*) y encadenamiento hacia atrás (*backward chaining*), más conocido como sistema de reglas de producción y utiliza una implementación avanzada del algoritmo Rete (**ReteOO**).

Drools se apoya en el estándar JSR-94 para su motor de reglas de negocio y framework de construcción, mantenimiento, y refuerzo de políticas de empresa en una organización, aplicación o servicio. Además, es software libre distribuido según los términos de la licencia Apache.

2.4.2 Historia del proyecto

Drools inicia su historia en el año 2001, año en que Bob McWhirter inició y registró el proyecto "*The Drools Project*" en *SourceForge*. La primera versión de Drools, 1.0 no se liberó, ya que era muy vulnerable a ataques por fuerza bruta. De inmediato se apostó por su segunda versión, Drools 2.0 basado en el algoritmo de RETE. En este punto el proyecto fue movido a *Codehaus*, convirtiéndose la versión 2.0 en una versión final de un motor de reglas de negocio de código abierto sobre la plataforma JAVA.

En el año 2005 el proyecto pasó a formar parte de JBoss y renombrado a JBoss Rules, y en el 2006 y gracias al apoyo económico de la empresa RedHat se reescribió para dotarle de una versión mejorada del algoritmo de Rete y herramientas de Interfaz gráfica. No obstante y debido a discusiones sobre el nombre con que referirse al proyecto, en 2007 pasó a tener el nombre final de **Drools (JBoss Rules)**, con el que se conoce actualmente.

2.4.3 Herramientas de la suite

Drools nació como un motor de reglas de negocio. Pero sus desarrolladores rápidamente se dieron cuenta que esta aproximación no era suficiente si querían convertir su producto en un producto unificado para el modelado comportamental de los sistemas. De manera que aparte de la gestión de las reglas de negocio debían otorgar también importancia a otros aspectos.

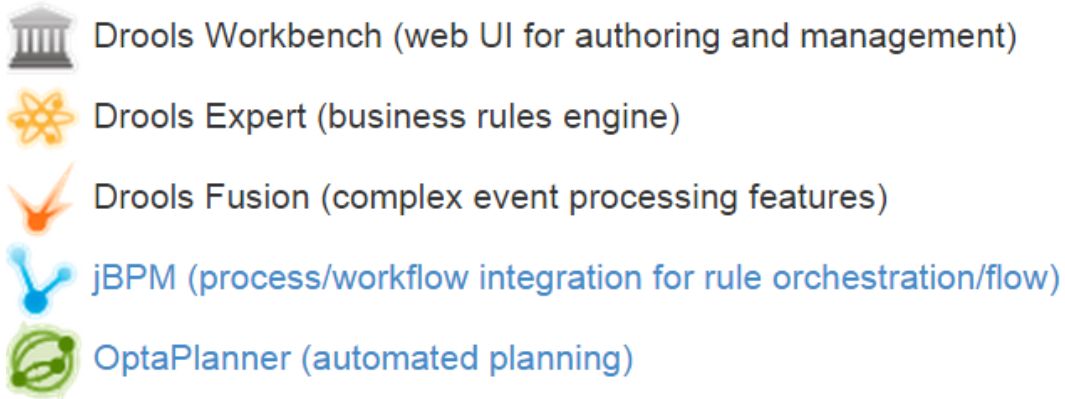
Finalmente la suite Drools ofrece capacidad para desarrollar las siguientes tareas:

- ✓ Administración de reglas de negocio (BRMS).
- ✓ Procesamiento de Eventos Complejos (CEP).

- ✓ Gestión de Procesos de Negocio.

La suite Drools consta de los siguientes productos, que a continuación pasamos a describir:

Ilustración 9: Productos de la suite Drools



Drools Workbench (anteriormente Guvnor)

Workbench consiste en una interfaz web de usuario para el manejo y control de auditoría de las diferentes reglas de negocio. Este módulo se compone de toda clase de editores, pantallas y servicios para facilitar la administración de las diferentes reglas y activos. Se usa de manera conjunta con *Drools Expert* o *Drools Fusion*.

Workbench consiste en un repositorio centralizado de las reglas de negocio y permite gestionarlas desde una interfaz web, además de crear reglas de negocio podemos realizar las siguientes tareas:

- Crear reglas guiadas.
- Crear tablas de decisión de una forma visual.
- Crear, editar y almacenar modelos de negocio, bien creados en lenguaje DSL o realizando una carga (*upload*) de POJOS (*Plain Old Java Objects*), para que puedan usarse como modelo de las reglas de negocio.
- Crear los formularios asociados a tareas del proceso de negocio.
- Llevar un histórico de cambio de versiones de todos los recursos que almacena.

Estos son algunos de los activos que se pueden crear a través de esta interfaz:

Package (Paquete): Permite clasificar, administrar y configurar los elementos que forman parte de un conjunto de reglas de negocio.

Category (Categoría): Son el equivalente a un directorio para la clasificación de las reglas que forman parte de un paquete.

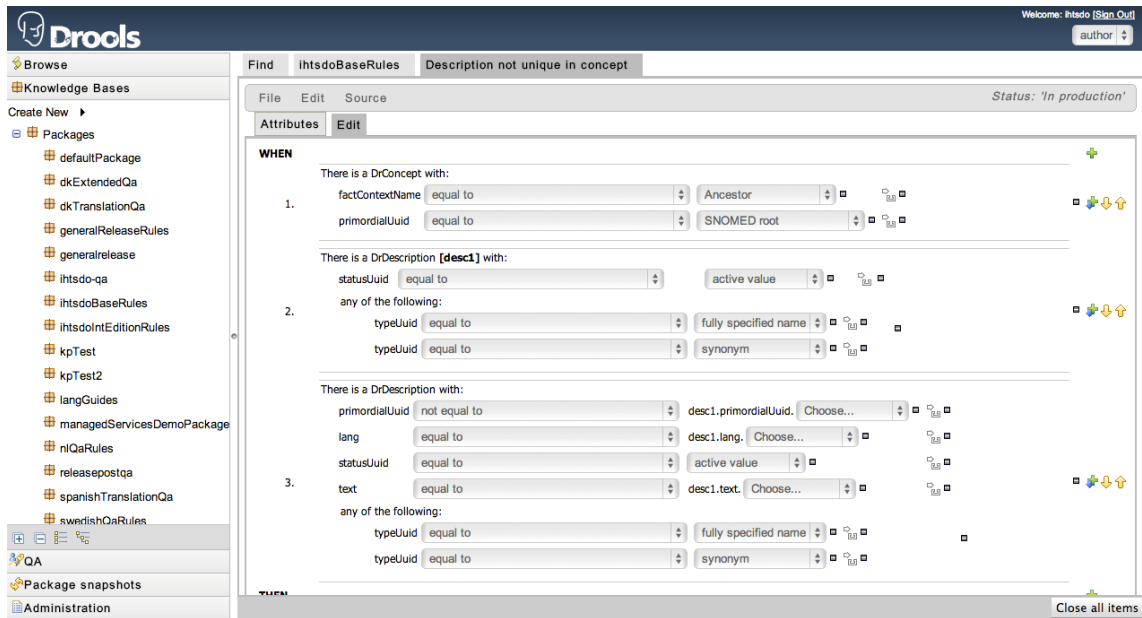
Fact Model (Modelo de hechos): Es el conjunto de datos de base que permitirán ejecutar y aplicar las reglas de negocio, es decir, el conjunto de acciones y clases que permitirán modelar las reglas de negocio.

Rule (Reglas): Las reglas son las sentencias que contienen la lógica de negocio (la perspectiva del usuario empresarial).

Y además cuenta con diferentes opciones de creación y modelado de reglas:

- ✓ Business Rule (editor guiado).
- ✓ DSL Business Rule (editor textual).
- ✓ DRL Rule (editor de reglas técnicas).
- ✓ Decisión Tabla (hoja de cálculo) y Decisión Table (tabla formato web).

Ilustración 10: Interfaz Drools Worbench

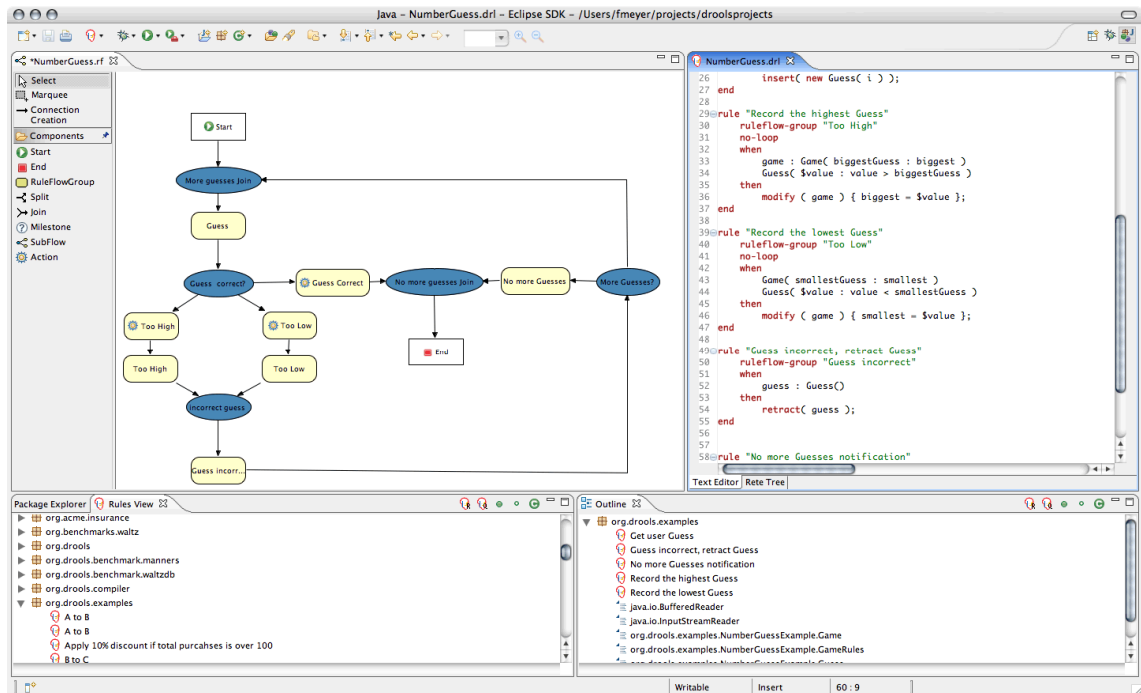


	#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
+	1		131000	200000	30	20000	Asset	true	0	2
+	2		10000	100000	25	2000	Job			4
+	3		12000	200000		500				
+	4			20000		30000			1	2
+	5		13000		20	0	Asset		5	4
+	6		4000	13000		55000				
+	7		1000	25000	20					6
+	8			20000		27000	Asset		3	5
+	9		5000			34000	Job	false	8	4
+	10		1000	14000	20	12000	Asset		7	5
+	11		90000	100000	25	430000				7
+	12		0	23000	30	5000	Job		4	1
+	13		100001	130000	20	3000		true	10	6

Drools Expert

Drools expert es el núcleo del proyecto Drools. Aporta un motor de reglas de negocio, que bien a través de una interfaz web o mediante el uso de una API pública que se puede utilizar desde proyectos Eclipse y permite el manejo de reglas, realizar la inserción de hechos en la memoria de trabajo, etc.

Ilustración 11: Interfaz Drools Expert



Drools Fusion

Drools Fusion es el módulo encargado de añadir capacidades de procesamiento de eventos complejos en Drools.

Los escenarios en que resulta de interés este enfoque son los siguientes:

- Existe un número grandísimo de eventos pero solo algunos resultan de interés.
- Los eventos son inmutables desde el momento en que se registran.
- Existen fuertes relaciones temporales entre los eventos.
- Los eventos a nivel individual no son tan importantes como los patrones que aparecen sobre las relaciones entre los mismos.
- Normalmente el sistema necesita agregar los diferentes eventos.

Para ello este módulo debe realizar entre otras las siguientes tareas:

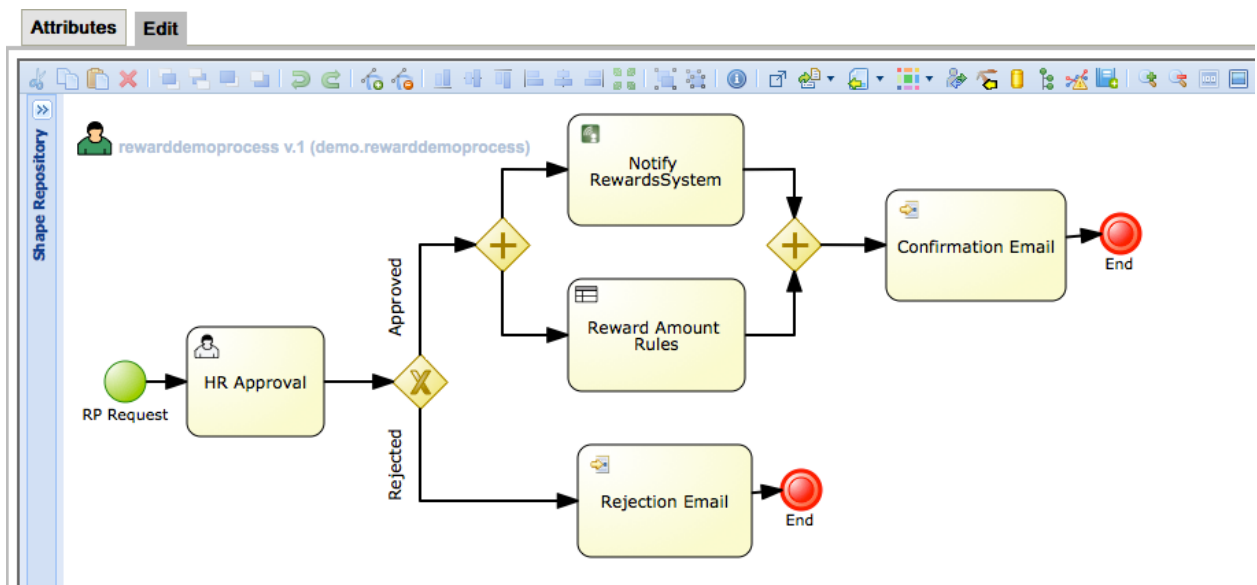
- Soporte para eventos con su propia semántica como "*ciudadanos de primera clase*".
- Permitir detección, agregación, correlación y composición de eventos.
- Ofrecer soporte para reglas reactivas.
- Ofrecer soporte para adaptadores de entrada de eventos (*pipeline*).
- Ofrecer un *reloj* unificado en la sesión, para establecer sincronización de eventos.
- Ofrecer soporte a las diferentes relaciones temporales, de manera que se puedan modelar las relaciones temporales entre los eventos.

jBPM

jBPM es un motor de flujo de trabajo de código abierto escrito en Java que puede ejecutar cualquier proceso de negocio descrito en BPMN 2.0 (o en versiones anteriores su propio lenguaje de definición jPDL). Este software también es distribuido bajo licencia Apache. Y proporciona varias herramientas tanto para desarrolladores en Eclipse, como para usuarios a través de una interfaz web, que permite crear, implementar, ejecutar y gestionar los procesos de negocio a lo largo de su ciclo de vida.

Esta herramienta toma las descripciones gráficas de los procesos como entradas. Estos procesos se componen de tareas conectados por flujos de secuencia y representan los flujos de ejecución. Este diagrama gráfico (diagrama de flujo), se utiliza como la base de comunicación entre los desarrolladores y los usuarios no técnicos.

Ilustración 12: Interfaz jBPM



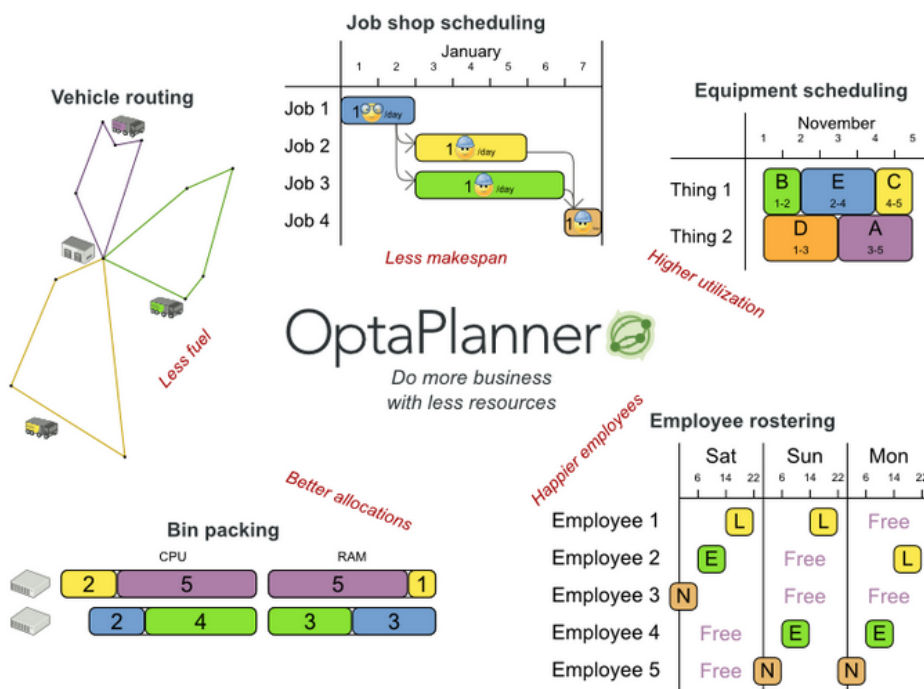
OptaPlanner

OptaPlanner es un sistema de resolución de restricciones, se trata de un motor ligero que se puede integrar con otras aplicaciones y 100% desarrollado en Java. Este sistema actúa para dar soporte a diferentes tareas de planificación.

Sirve para optimizar recursos y la planificación de las tareas, como por ejemplo:

- Establecer rutas de vehículos.
- Establecer horarios de los empleados, calendario de ligas deportivas, horarios escolares, etc.
- Gestión almacenamiento en almacén, optimización de líneas de montaje.
- Optimización de uso de material en procesos industriales.
- Optimización del portafolio de inversiones financieras.

Ilustración 13: OptaPlanner



2.4.4 Apuntes sobre Drools

La mayoría de las empresas a lo largo del tiempo van adquiriendo información que modela la lógica de negocio y que se almacena de manera dispersa en diferentes sitios, como pueden ser hojas de cálculo, bases de datos, diferentes aplicaciones, así como en la mente de los expertos.

Los sistemas de gestión de reglas de negocio como **Drools** surgen ante la necesidad de centralizar y gestionar toda esta lógica de negocio. Para ello, esta lógica deberá ser codificada en forma de reglas de negocio, que servirán para la toma de decisiones dentro de un contexto.

Dichas reglas se ejecutarán dentro de un motor de reglas, de manera que no solamente tendremos una representación lógica de las mismas, sino que además seremos capaces de ejecutarlas en tiempo real. Asimismo, al no encontrarse encapsuladas dentro de una aplicación, conseguiremos que el personal de carácter menos técnico de la organización pueda trabajar con ellas de manera más simple.

En Drools una regla de negocio es una sentencia del tipo cuando/entonces (*when/then*). Cuando se cumple una condición se desencadenará una acción que puede ser de cualquier tipo, de manera que se utilizan las reglas de negocio para la toma de decisiones sobre unos criterios o condiciones establecidos. De esta manera, el escenario ideal para utilizar Drools es cuando queremos separar la lógica de negocio de las aplicaciones, quedando la lógica centralizada y pudiendo ser gestionada por los expertos del negocio directamente.

Además, un motor de reglas de negocio de estas características es muy eficiente y ayuda a simplificar los procesos cuando la lógica de negocio es altamente cambiante a lo largo del tiempo. Ya que el realizar cambios directos sobre la lógica de negocio resulta un proceso muy ágil.

Un punto muy interesante del proyecto Drools es hacer notar que cuenta con un *plugin* de Eclipse y una serie de librerías que nos facilitan el trabajo a la hora de escribir reglas de negocio y desarrollar aplicaciones utilizando el motor de reglas de negocio que aporta. De esta manera, el BRM puede ser utilizado incorporándolo directamente en el código de nuestra aplicación.

Sin embargo, también puede ser utilizado a través de una interfaz web como servicio de toma de decisiones o implantándolo bajo una arquitectura SOA, de manera que este servicio actuaría como el "*cerebro*" de nuestro sistema, comunicándose con otros servicios o procesos a través de un ESS o Enterprise Service Bus.

Como último punto de esta sección cabe resaltar tres de las principales características de Drools:

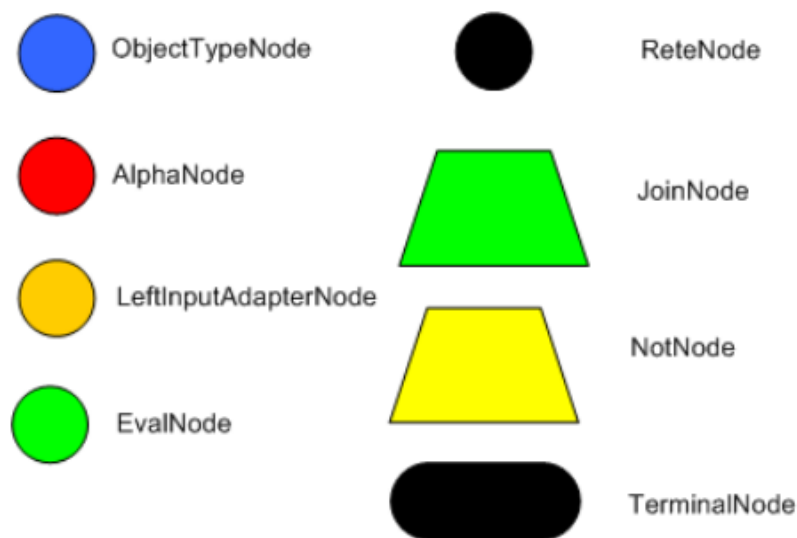
- Motor de reglas: basado en el algoritmo Rete que permite que las reglas se ejecuten de manera muy eficiente. JBoss llama a este módulo "*Drools Expert*".
- Lenguaje DRL: permite la creación de reglas de manera sencilla. También se permite la creación de reglas en lenguaje específico usando DSL (*Domain Specific Language*). Gestionadas mediante hojas de cálculo.
- BRMS ("*Workbench*" anteriormente "*Guvnor*"): permite gestionar las reglas de manera centralizada con una interfaz web.

2.4.5 Implementación del algoritmo de RETE que hace Drools

El algoritmo de Rete fue inventado por el Doctor Charles Forgy en su tesis doctoral de 1978-79. El término *rete* proviene del latín y significa red. El algoritmo de Rete se puede dividir en dos partes: **1- compilación de reglas** y **2- tiempo de ejecución**.

El algoritmo de compilación describe cómo las reglas se procesan en la **Memoria de Producción** para generar una red de discriminación eficiente. Dicho de manera informal, la red de discriminación se usa para filtrar los datos y propagarlos a través de la red. Los nodos en la parte superior de la red tendrán muchas coincidencias y a medida que navegamos por la red, estas coincidencias disminuirán. En la parte más inferior de la red se encuentran los nodos terminales. En una versión simplificada del Dr. Forgy de 1982 describe 4 nodos básicos: **root** (raíz), **1-input**, **2-input** y **terminal**.

Ilustración 14: Clasificación de nodos RETE

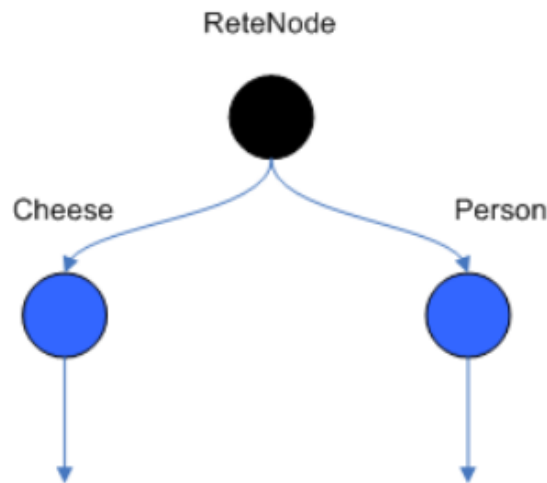


El nodo **root** o raíz es por donde todos los objetos entran en la red. Desde ahí, inmediatamente van al **ObjectTypeNameode**, cuyo propósito es asegurarse de que el motor de reglas no hace más trabajo del necesario.

Por ejemplo, si tenemos dos objetos: *Cuenta* y *Pedido*, si el motor de reglas trata de evaluar cada uno de los nodos con cada uno de los objetos se malgastarán un gran número de ciclos. Para hacer el proceso más eficiente, el motor debería pasar únicamente los objetos a los nodos que son de dicho tipo de objeto. La manera más sencilla de hacer esto es creando un **ObjectTypeNameode** y que todos los nodos 1-input y 2-input descendan de él. De esta manera si la aplicación inserta una nueva *Cuenta* no se propagara a los nodos pertenecientes al objeto *Pedido*.

En Drools cuando un objeto se inserta se recibe una lista de **ObjectTypeNodes** válidos mirando a través del **HashMap** del objeto **Class**. Si esta lista no existe, se escaneará todos los **ObjectTypeNodes** buscando *matches* que encajen en la lista. Esto permite a Drools hacer *matching* con respecto a cualquier tipo **Class** que encaja mediante la comprobación de "*instanceof*".

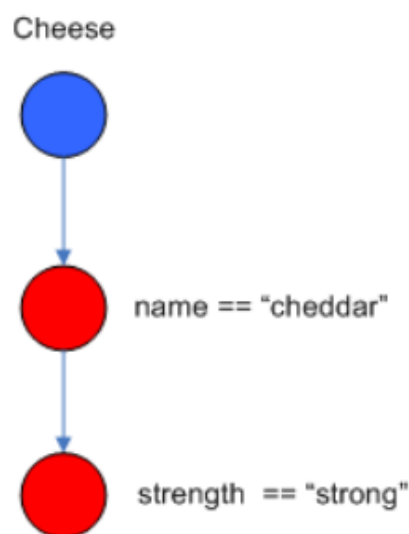
Ilustración 15: ObjectTypeNodes en red Rete



Los *ObjectTypeNodes* pueden propagar a los *AlphaNodes*, *LeftInputAdapterNodes* y los *BetaNodes*. Los *AlphaNodes* se usan para evaluar condiciones literales. A pesar de que el artículo de 1982 solo cubría condiciones de igualdad, la mayoría de implementaciones de Rete soportan otros tipos de operaciones. En el ejemplo, *Cuenta, nombre=="Mr. Trout"* es una condición literal. Cuando una regla tiene condiciones literales múltiples para un único tipo de objeto que aparecen de manera conjunta, esto significa que si una aplicación inserta un objeto Cuenta primero deberá satisfacer la primera condición literal antes de que se pueda enviar al siguiente *AlphaNode*. En el artículo del Dr. Forgy hace mención a las condiciones de intra-elementos (*IntraElement*).

El siguiente diagrama muestra las combinaciones de *AlphaNode* para Queso (*nombre(name)=="cheddar", intensidad(strength)=="fuerte"*):

Ilustración 16: Alpha Node en una red Rete

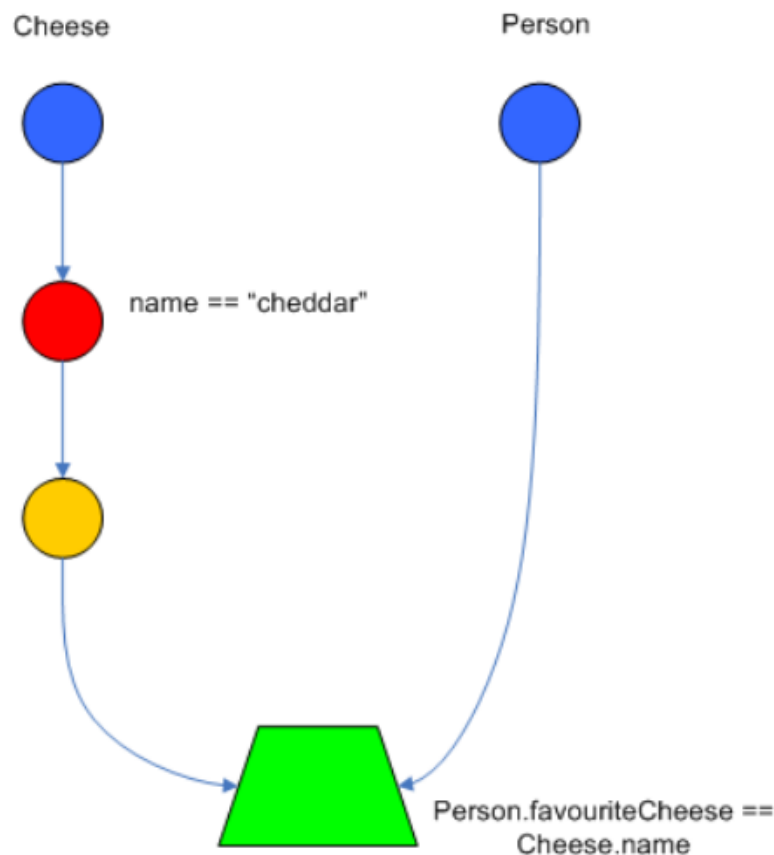


Drools utiliza Rete optimizando la propagación desde el *ObjectTypeNode* al *AlphaNode* utilizando técnicas de **hashing**. Cada vez que un *AlphaNode* se añade al *ObjectTypeNode* se añade el valor literal como clave al *HashMap* con los *AlphaNode* como valor. Cuando aparece una nueva referencia en el nodo *ObjectType*, en lugar de propagarse a cada *AlphaNode*, se envía al *AlphaNode* referenciado en el *HashMap* para el tipo de objeto, evitando comprobaciones innecesarias. Hay 2 nodos de dos entradas (two-input nodes), *JoinNode* y *NotNode*, y ambos son *BetaNodes*. Los *BetaNodes* se usan para comparar dos objetos y sus campos uno respecto a otro. Por convención nos referimos a las dos entradas como derecha e izquierda. La entrada izquierda de un *BetaNode* es normalmente una lista de objetos; en Drools es una **tupla**. La entrada derecha es un único **objeto**. Dos Nodos se pueden usar para implementar la comprobación de existencia "exists". Los *BetaNodes* tienen también memoria.

La entrada izquierda se llama *Beta Memoria (Beta Memory)* y mantiene todas las tuplas que van llegando. La entrada derecha se llama *AlphaMemory* y mantiene todos los objetos que van llegando. Además, Drools mejora el algoritmo de Rete mejorando la indexación de los *BetaNodes*.

Por ejemplo, si sabemos que un *BetaNode* está realizando una comprobación en un campo *String*, a medida que se inserta cada objeto podemos hacer una búsqueda hash sobre el valor de dicho campo. Esto significa que cuando un hecho entra por el lado contrario, en lugar de iterar sobre todos los hechos para encontrar uniones válidas, podemos retornar valores de candidatos potencialmente validos. En cualquier punto, una unión válida se puede encontrar en la *tupla* y se unirá al Objeto; estableceremos que tenemos un match parcial y lo propagaremos al siguiente nodo.

Ilustración 17: Ejemplo matching parcial en red Rete



Para permitir al primer objeto, en el ejemplo *Cheese* entrar en la red, usamos un *LeftInputNodeAdapter*. Este toma un objeto como entrada y propaga una unión *ObjectTuple*. Los nodos terminales (*Terminal Nodes*) se usan para indicar que una regla ha hecho *match* en todas sus condiciones, en este punto podemos decir que la regla encaja en las condiciones.

Una regla con un conector condicional disyuntivo (OR), genera diferentes subreglas para cada una de las ramas que son lógicamente posibles, de esta manera una regla puede tener múltiples nodos terminales.

Drools también realiza **compartición de nodo**. Muchas reglas repiten los mismos patrones, y la compartición de nodos permite colapsar todos estos patrones de manera que no deben ser reevaluados para cada una de las instancias.

Por ejemplo, las siguientes dos reglas comparten el primer patrón, pero no el segundo:

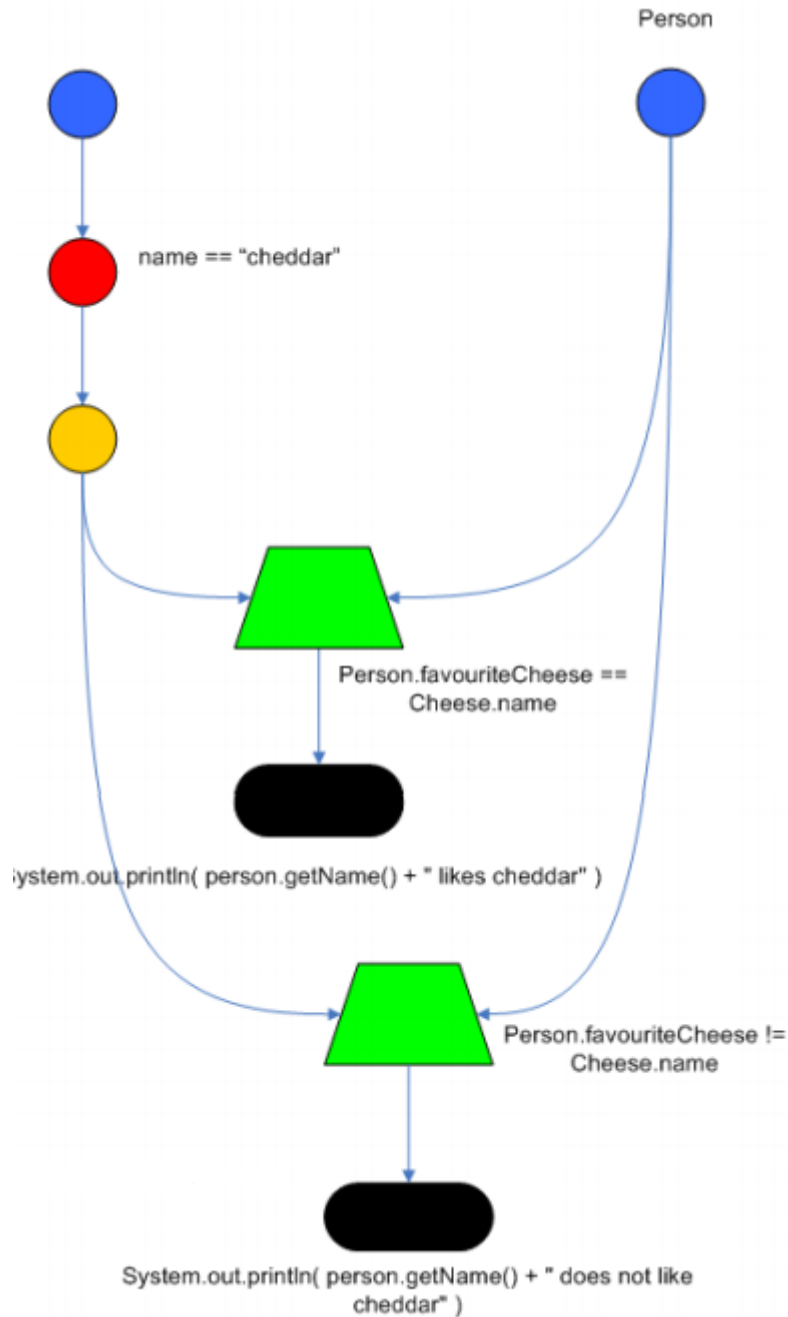
Ilustración 18: Reglas Drools

```
rule
when
    Cheese( $cheddar : name == "cheddar" )
    $person : Person( favouriteCheese == $cheddar )
then
    System.out.println( $person.getName() + " likes cheddar" );
end
```

```
rule
when
    Cheese( $cheddar : name == "cheddar" )
    $person : Person( favouriteCheese != $cheddar )
then
    System.out.println( $person.getName() + " does not like cheddar" );
end
```

Como se puede comprobar más abajo, la red Rete compilada muestra que el *AlphaNode* se comparte, pero no es así con los *BetaNodes*. Cada *BetaNode* tiene su propio terminal. Y si el segundo patrón hubiera sido el mismo, también habría sido compartido.

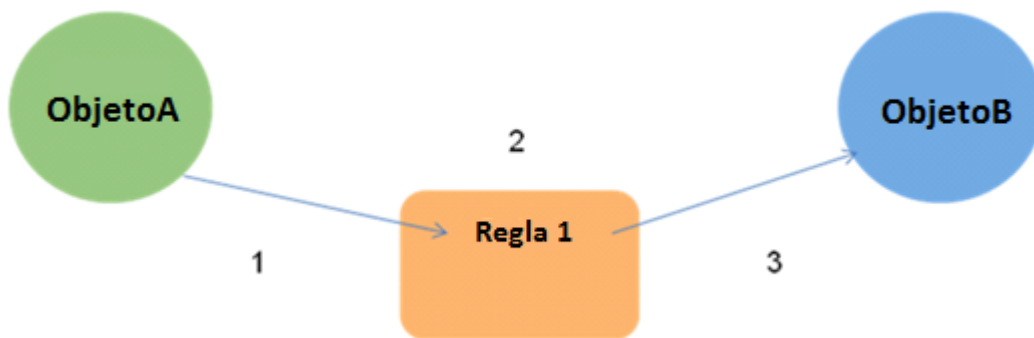
Ilustración 19: Ejemplo de compartición de nodo en red Rete



2.4.6 Las reglas de negocio en Drools

Como comentábamos anteriormente, las reglas en Drools modelan la relación entre diferentes objetos. Además estas reglas pueden ser especificadas en diferentes formatos de diferente nivel de abstracción, de manera que puedan ser administradas tanto por el personal técnico como por parte del personal no técnico.

Ilustración 20: Relación entre Objetos en una regla



En esta sección vamos a repasar la forma en que se pueden especificar las reglas en Drools.

- Fichero de reglas (drl)

Una de las formas de especificar las reglas de Drools es mediante el lenguaje de reglas propio (DRL), que permite especificar las condiciones, acciones y funciones expresadas en diferentes lenguajes como Java o MVEL. Estas reglas son guardadas en un fichero con extensión drl.

Estos ficheros son ficheros en texto plano con extensión (.drl), y especifican las reglas siguiendo la siguiente estructura:

```
rule "nombre"  
  atributos  
  when  
    LHS  
  then  
    RHS  
end
```

Donde LHS significa **Left-Hand Side** o lado izquierdo y RHS **Rigth-Hand Side** o lado derecho de la regla. En LHS se especifica la parte condicional de la regla, siguiendo una sintaxis propia del lenguaje de especificación de reglas de Drools, que es un dialecto de Java. Mientras que en el RHS, después de la sentencia "*then*" se especifica el código que se va a ejecutar.

Una regla debe ser especificada mediante un nombre único dentro del paquete que la contiene. Si se incluye más de una regla con el mismo nombre dentro del fichero DRL se producirá un error, y si existe dentro del mismo paquete se producirá una sustitución de la misma. Además, si el nombre de la regla contiene espacios hay que encerrar el nombre entre comillas dobles (siempre se recomienda que se haga de esta manera, con espacios o sin ellos).

Una regla finaliza cuando aparece la palabra reservada "*end*".

A continuación, vamos a ver algunos ejemplos de reglas de negocio:

```
package com.ejemplo1.rrhh;
import com.ejemplo1.rrhh.Empleado;
import java.math.BigDecimal;
rule "Programador"
when
    empleado : Empleado(promedioConocimientos >= 8, promedioConocimientos <= 10)
then
    System.out.println("Programador: " + empleado.getNombre());
    empleado.setCargo("Programador");
    empleado.setSalario(BigDecimal.valueOf(1000));
end

rule "Lider de Proyecto"
when
    empleado : Empleado(promedioConocimientos >= 4, promedioConocimientos <= 7)
then
    System.out.println("Lider de Proyecto: " + empleado.getNombre());
    empleado.setCargo("Lider de Proyecto");
    empleado.setSalario(BigDecimal.valueOf(2000));
end

rule "Gerente"
when
    empleado : Empleado(promedioConocimientos >= 0, promedioConocimientos <= 3)
then
    System.out.println("Gerente: " + empleado.getNombre());
    empleado.setCargo("Gerente");
    empleado.setSalario(BigDecimal.valueOf(3000));
end
```

Las reglas han de ubicarse dentro de un paquete, y se deberán importar las diferentes clases de Java necesarias para utilizar el archivo de reglas que las contiene.

Mediante la palabra reservada **“rule”** definimos el nombre de la regla. Con la palabra reservada **“when”** se indicarán las diferentes condiciones que representarán a la regla, es decir, las diferentes condiciones que deberán cumplir los hechos para que dicha regla les sea aplicable.

Como podemos comprobar, lo que estamos haciendo es tomar objetos de la clase *Empleado* y verificar si el promedio de conocimientos está dentro de un rango dado. En el lenguaje DRL las condiciones se separan mediante una coma (,) que sería equivalente a realizar una conjunción lógica o AND, aunque también se podría aplicar una negación o la disyunción lógica (OR).

Después de las condiciones, mediante la palabra reservada **“then”** especificamos las acciones que se ejecutaran como consecuencia del cumplimiento de las condiciones y por tanto aplicación de la regla.

Como podemos ver en el ejemplo, simplemente realizamos una asignación de cargo y salario e imprimimos por pantalla la regla aplicada y el empleado sobre el que se aplica.

La lógica del ejemplo es muy sencilla, pero en la práctica se puede llamar a cualquier método de cualquier objeto insertado en la memoria de trabajo, de manera que se podrían implementar diferentes operaciones más o menos complejas que podrían ser invocadas directamente desde las reglas.

- Domain Specific Languages (DSL)

Lenguaje específico de dominio o DSL, es una forma de crear reglas usando directamente la terminología del dominio de problema.

La especificación mediante DSL permite separar la capa de modelado del objeto de dominio respecto a la capa de la gestión de autoría de reglas y su uso a más alto nivel. El modelo de objeto tiene carácter puramente técnico, y especifica la implementación del objeto en el lenguaje del motor de reglas, de manera que mediante la especificación DSL se ocultan los detalles técnicos y de implementación y se centra en el nivel lógico de la regla.

De esta manera, el problema y el comportamiento del sistema se describe en el lenguaje del propio usuario, y éste podrá realizar modificaciones sobre los diferentes aspectos de la lógica de negocio en tiempo de ejecución. Sin necesidad de que realice un recompilado de la aplicación o un nuevo despliegue de la misma. De esta manera, se pretende disminuir la participación del personal técnico en la definición de los diferentes aspectos de negocio.

Las definiciones DSL consisten en transformación de sentencias DSL a construcciones DRL. Dado un DSL, las reglas se especifican en reglas DSL (o DSLR), que se transformarían en ficheros DLR. DSL y DSLR son formatos de fichero de texto plano, y también se pueden crear o editar mediante cualquier editor de texto. No obstante, existen editores dedicados de DSL y DSLR tanto en el IDE de Drools como en el aplicativo web.

Además, las sentencias DSL pueden ser utilizadas como plantillas, encapsulando las diferentes condiciones de las reglas, de manera que mediante composición pueden establecer condiciones lógicas de más alto nivel.

Especificar las reglas en este formato no tiene ningún impacto sobre las prestaciones del motor de reglas, ya que éstas son compiladas mediante un *parser* específico y de esta forma transformadas para que el motor de reglas las entienda directamente.

Ilustración 21: Ejemplo de mapeado DSL en Drools

```
[when]Something is {colour}=Something(colour=="{colour}")
```

El mecanismo DSL de Drools permite personalizar las expresiones condicionales y las acciones.

En el ejemplo anterior, la palabra reservada "[when]" inicia la expresión de sustitución. Después se especifica la expresión usada en la regla, normalmente una expresión en lenguaje natural aunque no tiene por qué ser así. La parte a la izquierda del "=" es el **mapping** de la expresión en el lenguaje de reglas. El formato de esta cadena depende de si formará parte del RHS o del LHS. Si es para el LHS, deberá seguir la sintaxis del término LHS; en cambio, si es para el RHS puede constituir una simple sentencia Java.

Cuando el *parser* DSL encuentra una línea en el fichero con una expresión de definición DSL, realiza una manipulación de la cadena de 3 pasos: Primero extrae los valores de la cadena que contienen nombres entre corchetes (en el ejemplo {colour}). Después, los valores obtenidos de esta captura son interpolados en los lugares en que dicho nombre aparece otra vez entre corchetes en el lado izquierdo de la expresión. Y

por último se hace una sustitución con el valor encontrado a lo largo de todo el fichero de definición DSL.

Es importante hacer notar que el compilador realiza una transformación del fichero línea a línea. En el ejemplo anterior, todo el texto después de "*Something is* " hasta el final de la línea se captura como un reemplazo de valor por "{*colour*}" y es usado para interpolar la cadena objetivo. Esto puede no ser siempre lo que queremos, por ejemplo, cuando intentamos unir diferentes expresiones DSL para generar un patrón DLR será necesario transformar a DSLR en diferentes operaciones de manera independiente. La mejor manera para conseguir esto es asegurándose de que las capturas están rodeadas por el texto de interés, palabras o caracteres.

Como resultado del proceso de las operaciones de *matching* hechas por el *parser* se extraen diferentes subcadenas de la línea. En el ejemplo anterior, las comillas se usan como caracteres distintivos. Los caracteres que rodean la captura no son incluidos durante la interpolación, únicamente el contenido entre ellos.

Pero vamos a ver cómo podemos crear un DLS mediante un ejemplo:

Cada línea de un fichero .dsl tiene el siguiente formato:

```
[<scope>][<Type>]<language expression>=<rule mapping>
```

donde scope puede tener los siguientes valores:

condition: se remapeará a una condición

consequence: se remapeará como el consecuente de una regla

*: se puede usar como condición o consecuencia

keyword: el mapeo se aplicará a la regla por completo. Esto se utiliza normalmente cuando el dsl se escribe en un idioma que no es el inglés o cuando se quiere personalizar la estructura de la regla básica "*rule-when-end*".

Mediante el DSL seremos capaces de convertir el lenguaje técnico DRL en uno de más alto nivel y amigable para los usuarios. Para ello, crearemos un fichero con extensión .dsl y mapearemos las construcciones sintácticas entre el DSL y DRL de manera que el motor de inferencia sea capaz de interpretar las mismas.

Ejemplo de definición DSL

```
[condition][Customer]There is a customer=$customer:Customer()
```

```
[condition][Customer]-The expense of the customer is greater than  
{expense}=eval($customer.getExpense() > {expense})
```

```
[condition][Customer]-The status of the customer is  
registered=eval($customer.isRegistered())
```

```
[condition][Customer]-The status of the customer is not  
registered=eval($customer.isRegistered()==false)
```

```
[consequence][[]Calculate a {percentage} percent  
discount=$customer.setDiscount(getDiscount($customer.getExpense(),  
{percentage}));
```

Después crearemos un fichero DSLR que nos permitirá utilizar las sentencias creadas directamente como si de líneas de código se tratara.

```
package com.plugtree.dsrl
import com.plugtree.dslExample.entities.*
expander dslExample.dsl
rule"expense greater than 1000"
when
    There is a customer
        -The expense of the customer is greater than 1000
        -The status of the customer isnot registered
then
    Calculate a 10 percent discount
end

rule"Registered customer and expense greater than 1000"
when
    There is a customer
        -The expense of the customer is greater than 1000
        -The status of the customer is registered
then
    Calculate a 15 percent discount
end

function Double getDiscount(Double expense, int discount){
    return (expense * discount) / 100;
}
```

Como se puede comprobar, la estructura de un fichero *.dsrl* es idéntica a la de un fichero *.dlr*, pero las condiciones y consecuencias aparecen especificadas de manera diferente.

Una sentencia importante del ejemplo que mostramos es la línea que contiene la palabra reservada “*expander*”, que informa a Drools cómo transformar las sentencias en reglas válidas. Drools leerá el fichero y realizará una traducción línea a línea a partir de los mapeos definidos.

-Tablas de decisión

Las tablas de decisión se codifican dentro de hojas de cálculo de una manera compacta pero precisa. De esta manera, se puede representar la lógica condicional en un formato de alto nivel de abstracción. Drools soporta tanto el formato de Excel (**XLS**) como **CSV**, de manera que es compatible con diferentes programas de gestión de hojas de cálculo como Microsoft Excel, OpenOffice Cal, etc. Además, en versiones más recientes también se pueden especificar directamente sobre la aplicación web que sirve de BRMS.

Las tablas de decisión son un concepto muy antiguo en términos de sistemas software, pero que han demostrado resultar de gran utilidad. El uso de tablas de decisión es idóneo cuando existe la posibilidad de codificar las reglas en un formato tabular, de esta forma la información que se inserta en las celdas puede ser utilizada como entrada condicional a evaluar. De esta manera, cada fila del fichero contiene codificada las condiciones, y se utilizará conjuntamente con la plantilla especificada en la parte

superior del propio fichero para generar cada una de las reglas, permitiendo realizar una abstracción del modelo de objetos subyacente. El uso de hojas de cálculo y los programas que le dan soporte permiten facilitar el proceso de importación y manipulación de datos, así como la realización de diferentes cálculos.

Además, por lo general las empresas ya utilizan hojas de cálculo para el manejo de los activos empresariales, realización de cálculos, etc. de manera que ya están familiarizados con su uso.

No obstante, el uso de tablas de decisión sólo es recomendable cuando podemos establecer una plantilla sobre las reglas que vamos a aplicar y el número de reglas que vamos a implementar es elevado.

Ilustración 22: Tablas de decisión

	B	C	D	E	F	G
16	Type of New Claim	Is case catastrophic	Allocation code		Insurance Class	Date of accident is after
17	Catastrophic Claim	Y				
18	New Claim with previous Accident som		Z			
19	Each row results in a rule					
20	Improving Claim					
21	Dependency Claim					
22	Interstate Claim					
23	Interstate Claim					
24	Interstate Claim					
25	Interstate Claim					

Finalmente, para clarificar algunos conceptos, simplemente hacer notar que en una tabla de decisión como la de la imagen cada fila constituirá una regla, mientras que las columnas podrán constituir tanto una condición como una acción para dicha regla.

A continuación, vamos a describir como se podría especificar de manera adecuada un **RuleSet** dentro de una hoja de cálculo mediante un ejemplo, de modo que veremos cómo se constituye la plantilla y se especifican las diferentes condiciones.

Ilustración 23: Plantilla RuleSet

RuleSet com.drools.ejemplo3.rrhh				
Import java.math.BigDecimal				
RuleTable AscensosPorConocimientos				
	CONDITION	CONDITION	ACTION	ACTION
	empleado: Empleado			
	promedioConocimientos >= \$param	promedioConocimientos <= \$param	empleado.setCargo("\$param"); System.out.println("\$param" + " " + empleado.getNombre());	empleado.setSalario(Big cimal.valueOf(\$param));
Reglas de Ascenso	Promedio Conocimiento Mínimo	Promedio Conocimiento Máximo	Ascender a Cargo:	Salario:
Gerente	0	3	Gerente	3000
Lider de Proyecto	4	7	Lider de Proyecto	2000
Programador	8	10	Programador	1000

La palabra reservada **RuleSet** indica que la plantilla contiene un conjunto de reglas, cuyo nombre se especifica en la celda contigua. Esta directiva no es obligatoria, pero mediante ella conseguiremos que las clases del paquete asociado sean accesibles.

Posteriormente, mediante un **import** declaramos las clases Java que vamos a utilizar en la plantilla. En caso de haber más de una se deben separar mediante comas.

A continuación, a través de la directiva **RuleTable**, indicaremos el comienzo de la tabla de reglas. El nombre de la misma aparecerá en esta misma celda separada por un espacio. Incluir la directiva **RuleTable** es obligatorio, pero dar un nombre a la misma es opcional y debe aparecer justo encima de la primera columna de condición. Todas las columnas a la izquierda de la columna donde aparece esta palabra van a ser ignoradas, es decir, no serán interpretadas.

Cabe reseñar que una misma hoja de cálculo puede contener diferentes **RuleTable**'s, para ello únicamente se requerirá de una separación mediante una fila en blanco.

Después de estos encabezados vendrá la definición de la regla propiamente dicha. Mediante la directiva **CONDITION** indicaremos que se trata de una columna de condiciones y mediante la directiva **ACTION** indicaremos que se trata de una columna de acciones o consecuencias.

En la fila siguiente especificamos el objeto sobre el que se aplicará la búsqueda de patrones que constituirán los hechos, en el ejemplo la clase **Empleado** y consultaremos el atributo **promedioConocimientos**. Mediante la palabra **\$param** definimos las variables reemplazadas por los datos. Para la primera columna, por ejemplo, **\$param** tomará los valores 0,4, y 8 en la respectiva regla.

También podemos ver qué acciones se realizarán al encontrar un **match** entre **hecho** y **regla**. Por ejemplo, en el caso de la columna más a la derecha, es la asignación de un salario al empleado, según sus promedios de conocimiento.

La fila que sigue a estas directivas es un comentario que también será ignorado, pero puede ser de utilidad para clarificar las condiciones o acciones codificadas, de manera que no tengamos que reinterpretar el fichero cada vez que accedemos al mismo.

A continuación, cada fila representará una regla, con una condición específica de acuerdo al contenido de la plantilla, de manera que al llegar un hecho nuevo se comprobará en cuál de las reglas encaja mejor, tal y como se haría con un fichero DRL.

Por último, simplemente hacer notar que existe toda una serie de palabras reservadas para modificar el comportamiento lógico del fichero: directiva **"no loop"** para que al no hacer match el hecho no reitere sobre el fichero, **"retract"** para extraer el hecho que hace match, etc.). Toda esta información se puede encontrar fácilmente en las guías de usuario de Drools.

- Plantilla de reglas (Rule Templates)

Similar a las tablas de decisión, pero no requieren de una hoja de cálculo. Para ello, debe introducirse la información en forma de tabla, generando una plantilla que se puede utilizar para crear múltiples reglas. De esta manera se flexibiliza tanto la entrada de reglas desde una hoja de cálculo o desde una base de datos existente. En este sentido, únicamente deberíamos generar la plantilla, en lugar de tener que generar cada una de las reglas.

Con las plantillas de reglas la información sobre la propia regla es separada de la especificación de la misma, de manera que aparte de lo que se puede realizar mediante tablas decisión, también pueden realizar las siguientes tareas:

- Almacenar la información sobre las reglas en una base de datos (o cualquier otro formato).
- A través de diferentes condiciones generar reglas basándonos en los valores contenidos en la fuente de dato
- Usar la información en cualquier parte de la regla (operador condicional, nombre de clase, nombre de la propiedad, etc.).
- Generar diferentes plantillas usando la misma información.

-XML

Opcionalmente Drools también soporta un lenguaje nativo para la especificación de reglas, alternativo al DRL. Las reglas pueden ser especificadas y administradas a través de un fichero XML. Este fichero, como el resto, se transforma en una representación de uso interno mediante un mecanismo de transformación.

3 Memoria Técnica.

3.1 Introducción.

En este documento vamos a presentar la memoria técnica del proyecto, que cubrirá desde la descripción del problema al que nos enfrentamos, a la fase de análisis, diseño e implementación del producto software que vamos a presentar como solución.

Definiremos los requisitos tanto hardware como software y daremos todas las directrices necesarias para que se pueda poner el software en funcionamiento mediante un manual de despliegue y uso del mismo.

3.2 Problemática a resolver.

Durante la gestión de diferentes procesos de negocios industriales hemos detectado que se generan una serie de eventos que deben ser reportados al personal de manera inmediata. Esta necesidad de inmediatez en la comunicación hace que resulte imposible que soluciones de tipo comunicación vía correo electrónico u otros instrumentos resulte útil, ya que el personal raramente se encuentra frente al equipo en el que recibe las alertas o comunicaciones.

Por ello, plantamos la posibilidad de que dichas notificaciones se realicen de manera privada sobre un dispositivo de voz (Walkie-Talkie VOIP EWB-100). De esta modo, los operarios o personal en general podrán recibir información en tiempo real de diferentes condiciones físicas o virtuales que se puedan dar y que deban ser comunicadas.

Como ejemplos podemos indicar:

- ✓ Notificación de una cola en área de cajas de un supermercado.
- ✓ Notificación de ausencia de existencias de un artículo.
- ✓ Notificación de una persona accediendo a una área no autorizada.

Aunque las aplicaciones de nuestro sistema pueden ser numerosas y las citaremos más adelante en la sección de "**Conclusiones y posibles ampliaciones al proyecto**", lo que nos interesa es monitorizar cierto recurso y realizar una notificación cuando detectamos un cambio sobre el mismo. De esta manera nos podemos abstraer de cuál sea la herramienta de monitorización o sensor y centrarnos en la notificación de dicho cambio.

3.3 Análisis de las opciones disponibles en el mercado.

Tras un período de análisis y búsqueda de dispositivos semejantes en el mercado, no hemos encontrado ninguna solución que se adapte a los requerimientos del sistema que tratamos de implementar.

Es por ello que dentro de la gama de productos de comunicación sobre IP (**voice over IP "VOIP" VoWLAN**) del proveedor con el que habitualmente trabajamos hemos escogido el dispositivo EWB-100. Por un lado porque es un dispositivo que presenta

prestaciones de carácter industrial y por otro su coste unitario no es excesivo en comparación con otros productos de gama más alta. Además, cumple con un requerimiento imprescindible como es la posibilidad de mandar órdenes de ejecución de diferentes comandos de manera remota (pilotos parpadeando, reproducción de un vocabulario previamente pregrabado, etc.)

3.4 Justificación de las tecnologías seleccionadas para el desarrollo del producto.

Durante las fases previas de análisis del producto detectamos de manera prematura que la aplicación tenía unos requisitos especiales que la hacían idónea para ser candidata a requerir hacer uso de un sistema de gestión de reglas de negocio.

Por un lado, la aplicación ha de ser lo suficientemente flexible para que pueda ser reconfigurada de una manera rápida, ágil y precisa. Utilizando un sistema de gestión de reglas de negocio como Drools cubrimos este objetivo desde dos vertientes diferentes, por una lado podemos modificar las condiciones que desencadenan las acciones, simplemente editando el fichero que contiene las reglas de negocio especificadas. Característica especialmente útil cuando la lógica de la aplicación puede cambiar de manera recurrente a lo largo del tiempo. Además, también podemos modificar el comportamiento que se desencadenará a partir de la evaluación positiva de dichas condiciones de la misma forma. Y por ejemplo, modificar el contenido de las órdenes acústicas a reproducir.

Esto permite que la aplicación sea altamente reconfigurable, sin necesidad de realizar un recompilado o redistribución del *software* cuando queremos aplicar cualquier cambio. Y permitiendo que cualquier cambio pueda aplicarse de manera centralizada simplemente modificando la base de reglas.

Además, el uso del motor de inferencia de Drools nos permite realizar un procesamiento altamente eficiente de los eventos registrados. Para ello Drools hace uso de una versión mejorada del algoritmo de **RETE**, optimizada para programación orientada a Objetos. Esta es la principal característica que nos ha hecho decantarnos por el uso de este sistema de gestión de reglas de negocio para realizar la implementación de la solución software propuesta.

3.5 Análisis y diseño.

En este apartado se presentan y describen las especificaciones de requisitos y análisis del proyecto. Definiremos el panorama de requerimientos de alto y bajo nivel que son necesarios para realizar el diseño e implementación del software requerido.

Mostraremos los requisitos funcionales, no funcionales y de implementación. Asimismo aportaremos diferentes diagramas de casos de uso y diagramas secuenciales.

3.5.1 Análisis (especificaciones).

3.5.1.1 Introducción.

i. Propósito.

El propósito del software diseñado es otorgar a los operarios de cierta industria información en tiempo real de diferentes condiciones físicas que se dan en una determinada área de su espacio de trabajo, por ejemplo alta ocupación de la misma, presencia de un objeto que dificulta el paso, etc.

ii. Alcance.

El producto desarrollado se plantea como un sistema capaz de enviar alertas vocales a una serie de operarios encargados de un área determinada a partir de cierta información que es capaz de capturar del entorno.

Para ello, nuestro desarrollo se integrará con un sistema externo desarrollado por una tercera parte, y que consiste en la sensorización a través de una serie de cámaras que hacen uso de técnicas de visión artificial. De esta manera, coordinaremos la forma de comunicación entre ambas aplicaciones y de manera consensuada acordaremos los protocolos de intercambio de información de manera que monitorizaremos un directorio con los eventos que detecta las cámaras, procesaremos los mismos y generaremos las alertas acústicas que se reproducirán en los dispositivos de comunicación.

iii. Definiciones, acrónimos y abreviaturas.

TTS: (*Text To Speech*) conversor de texto a voz

VOIP: (*voice over IP*) voz sobre protocolo de internet. Hace posible que la señal de voz viaje a través de internet empleando un protocolo IP (protocolo de internet).

VoWLAN: Voice over WLAN.

XML: siglas en inglés de eXtensible Markup Language ("*lenguaje de marcas extensible*"). Es un lenguaje de marcas desarrollado por el World Wide Web Consortium (*W3C*) utilizado para almacenar datos de forma legible.

XML-RCP: XML-RPC es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.

MQTT: protocolo ligero para intercambio de mensajes entre sensores y dispositivos móviles, optimizado para redes sujetas a una alta latencia.

PTT: en inglés "*push to talk*", es un método para hablar en líneas *half-dúplex* de comunicación, apretando un botón para transmitir y liberándolo para recibir. Este tipo de comunicación permite llamadas de tipo uno-a-uno o bien uno-a-varios (llamadas de grupos).

3.5.1.2 Descripción general.

i. Perspectiva del producto. Análisis funcional.

La aplicación planteada consiste en la integración de un motor de reglas de negocio con un módulo de desarrollo propio que permite realizar la notificación de diferentes situaciones que se pueden dar en el contexto de los procesos logísticos propios de una superficie comercial o almacén dedicado a la venta de alimentos y otros bienes no perecederos.

Mediante nuestra aplicación podremos reportar las siguientes situaciones:

- Cola en una zona de cajas (pudiendo establecer cuál es la zona de cajas: acceso norte, acceso sur, este, oeste, caja 1, caja 2, etc.).
- Falta de existencias de artículos.
- Objeto bloqueando un pasillo o zona.
- Artículo caído o fuera de zona.
- Aviso de posibilidad de robo.
- Fuego en una zona determinada.
- etc.

Nuestro desarrollo se centrará en la búsqueda de estas condiciones en los diferentes eventos que registra una cámara que opera mediante algoritmos de visión artificial. Y realizará una correlación de los mismos, de forma que podamos determinar si existe alguna acción asociada a tal evento. De esta manera, realizaremos una categorización de los mismos y adicionalmente podremos establecer una serie de prioridades a la hora de desencadenar estas acciones de notificación.

Para el proyecto a realizar resultará indispensable:

- ✓ Coordinación entre la empresa que desarrolla el elemento de sensorización (cámaras) y nuestro sistema de notificación mediante mensajes vocales.
- ✓ Establecer el formato de los mensajes a intercambiar entre las aplicaciones.
- ✓ Coordinar el despliegue y puesta en marcha de la aplicación.

Además, para el correcto despliegue y manejo de la aplicación deberemos desarrollar un aplicativo web que permita realizar diferentes tareas de configuración de los grupos de usuarios que definimos:

- Personal de gerencia.
- Personal de cajas.
- Personal de seguridad.

Así como la puesta en marcha y paro de la aplicación, de manera que los usuarios puedan parar los procesos de notificación cuando consideren que éstos resultan excesivamente invasivos.

Adicionalmente utilizaremos esta interfaz con la aplicación para desarrollar la funcionalidad de enviar mensajes de demostración a un dispositivo que previamente se tendrá que seleccionar. Con esto pretendemos cubrir dos aspectos fundamentalmente:

Por un lado, realizar una demostración de la reproducción de las diferentes notificaciones vocales, y por otro, y en vista a aumentar la funcionalidad del sistema, crear una consola de aplicación que permita el envío directo de mensajes a un dispositivo. De este modo podremos enviar mensajes de forma anónima y privada de una forma automatizada.

ii. Reporte del Modelo de Casos de Uso.

A continuación mostramos el reporte de casos de uso desde el punto de vista del operador. Como se puede comprobar la interacción de este con la aplicación será mínima, ya que la única interacciones con el sistema de manera manual consisten en: configuración de los grupos de difusión, puesta en marcha de la aplicación, paro de la aplicación y purga de eventos antiguos en el directorio monitorizado.

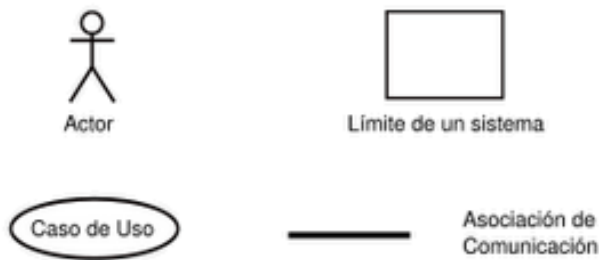
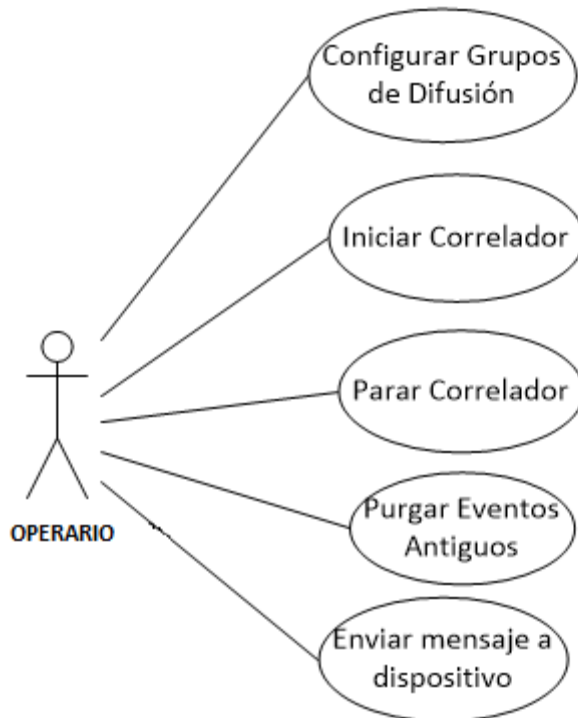


Ilustración 24: Reporte del modelo de Casos de Uso



iii. Restricciones.

- El sistema podrá ser desplegado sobre diferentes entornos (Linux, Windows, etc.), preferentemente sobre plataformas libres para evitar incurrir en sobrecostos debido a licenciamiento de software.
- El sistema tendrá que ser capaz de enviar los mensajes de recepción de los diferentes eventos en tiempo real, de manera que exista el menor desfase posible entre la aparición de las condiciones que generan el evento y la reproducción de su alerta correspondiente.
- La gestión de la lógica de negocio será fácilmente reconfigurable, tanto de las condiciones que desencadenan los eventos como las alertas que se reproducirán ante la aparición de cada uno de los eventos.
- El sistema permitirá configurar diferentes grupos de dispositivos de manera que múltiples dispositivos puedan recibir simultáneamente la orden de reproducción de una alerta sonora.

iv. Evolución previsible del sistema.

El sistema se desarrollará manteniendo un enfoque abierto, de manera que mantendremos el ciclo de vida del producto abierto. De este modo los diferentes departamentos que participan en el mismo podrán aportar sus ideas, tanto en lo que respecta a la funcionalidad del mismo como a otras posibles aplicaciones de un sistema con las características que presentamos.

Todas estas posibles ampliaciones y alguna más las explicaremos en el apartado correspondiente de "**Conclusiones y posibles ampliaciones al proyecto**".

3.5.1.3 Requisitos específicos

i. Requisitos comunes de los interfaces.

- Definición de interfaz de los módulos compatible con los tipos de datos que coordinamos con una tercera empresa para la generación de eventos.
- Posibilidad de envío de eventos de manera manual mediante un protocolo especificado (FTP, SCP, etc.).

a. Interfaces de usuario.

- Usuario básico: no requiere interfaz alguna al coger un dispositivo de comunicación de la cuna de carga, éste ya estará listo tanto para la comunicación típica de *broadcast* entre dispositivos como para la recepción de alertas desde el sistema desarrollado.

- Usuario avanzado: dispone de una interfaz web desde la que poder administrar los grupos de dispositivos, poner en marcha y parar el software desarrollado y enviar de manera manual alertas directamente desde el teclado o a través de una utilidad de composición de mensajes.

b. Interfaces de hardware.

- Dispositivos EWB100: dispositivos *walkie-talkie* VoIP VoWLAN.
- Dispositivo cliente (tablet, PC, smartphone, etc.): para realizar la puesta en marcha de la aplicación, gestionar los grupos de dispositivos y envío de mensajes de manera manual, seleccionando de una base de vocabulario previamente grabada en los dispositivos.

ii. Requisitos funcionales

a. Requisito funcional 1

Disponer de un servidor **Linux**, preferiblemente **Debian** en su versión más actualizada.

b. Requisito funcional 2

Instalar en el servidor descrito en el apartado "a" el servicio Apache que alojará la aplicación de configuración de grupos.

c. Requisito funcional 3

Tener disponible en el servidor una instalación de Java v.1.7 de manera que sirva de entorno de ejecución de la aplicación principal de análisis de eventos y reproducción de alertas acústicas.

d. Requisito funcional 4

Disponer de un directorio compartido con cualquiera de los sistemas (FTP, SAMBA), sobre el que la aplicación externa (sensor cámara) depositará los eventos que detecta a través de ficheros XML.

iii. Requisitos no funcionales

a. Fiabilidad.

Es un factor importante del producto. El sistema ha de ser fiable, y para ello, realizaremos diferentes pruebas e intentaremos ajustar todos los parámetros de manera que no solamente se reproduzcan los mensajes asociados a los eventos, sino que además lo haga con el menor retraso posible y sin que dé lugar a la pérdida de ningún mensaje.

b. Disponibilidad y Portabilidad.

Aunque inicialmente será concebido para ser ejecutado bajo entorno Linux, el sistema se mantendrá lo más abierto posible de modo que resulte posible llevarlo a otros entornos de ejecución.

De manera preliminar realizaremos el despliegue del sistema en un servidor Debian, por lo que de forma inmediata será compatible con otras distribuciones de Linux.

c. *Mantenibilidad.*

La aplicación estará estructurada de manera que se pueda ampliar y modificar de una forma sencilla, tanto en lo que es el núcleo de la aplicación como en toda la lógica de negocio que utiliza. Al tratarse de un desarrollo software no se necesitará gran mantenimiento más que realizar una “*limpieza*” de los ficheros de logs y otros ficheros que queden almacenados en el servidor de manera temporal.

Además, daremos soporte a la configuración y mantenimiento del hardware involucrado.

3.5.2 Diseño.

3.5.2.1 *Introducción.*

Uno de los puntos clave de la tecnología que vamos a utilizar son los dispositivos *walkie-talkie* VoIP (VoWLAN). Estos dispositivos pueden ser usados como *walkie-talkies* tradicionales, pero en lugar de funcionar usando frecuencias de radio utilizando la infraestructura WiFi de un almacén, tiendas, oficinas, etc. De esta manera, damos un valor añadido a las instalaciones WiFi y aseguramos la posibilidad de comunicación en entornos cerrados de donde no entra ni sale la señal de radio, pero en cambio sí que está dotado de conectividad WiFi. Por ejemplo, en el interior de las cámaras frigoríficas en determinadas industrias.

Resulta imprescindible que al dispositivo seleccionado se le puedan enviar comandos de manera que podamos invocar remotamente la orden de reproducción de una determinada alerta y además que ésta se reproduzca con un menor retardo desde el envío de la orden.

Después de estudiar diversas opciones, finalmente hemos escogido un dispositivo que no solamente tiene la capacidad de emitir señales acústicas, sino que además utiliza la tecnología TTS o *Text To Speech* para poder invocar órdenes de ejecución de forma remota. De este modo, el dispositivo recibe una orden de reproducción de una determinada palabra o cadena de palabras y siempre que éstas se encuentren en su vocabulario interno reproducirá las mismas de manera secuencial. Cabe hacer notar que el dispositivo es totalmente reconfigurable y se le puede asignar diferentes códigos visuales en los pilotos LED, dispone de botones reconfigurables y posibilidad de regrabar el vocabulario interno.

De este modo, tenemos una serie de dispositivos a los que debemos dotar de un vocabulario específico y mediante composición iremos generando diferentes mensajes de alerta. Por ejemplo es fácil imaginar el vocabulario específico que requerirá un proceso de negocio como puedan ser las actividades de una tienda de alimentos (cola, caja, acceso, artículo, producto, robo, en), con otras palabras auxiliares (alerta, 1, 2, 3, 4, caído, robo, sin, existencias etc.).

De esta manera podemos ir componiendo las alertas:

alerta+alerta+cola+en+caja+1

En este punto, los dispositivos ya son capaces de reproducir los mensajes a partir de una orden que les llega de manera remota, en este caso codificados en una petición http.

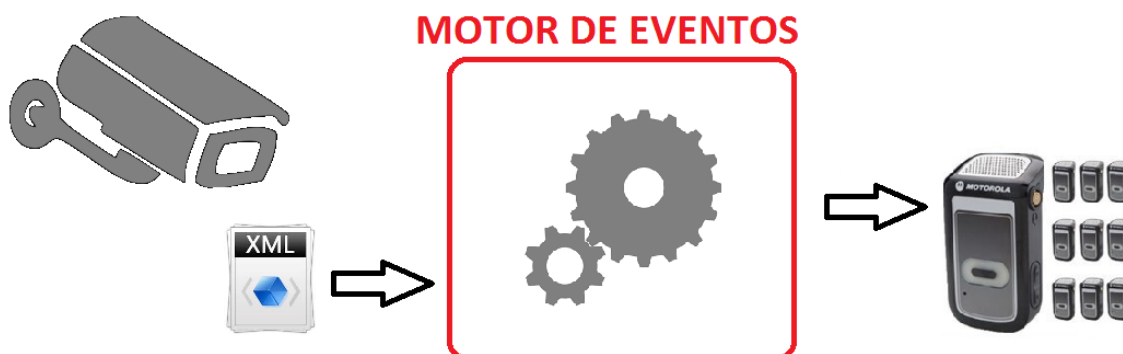
Podemos comprobar cómo se reproducen estos mensajes desde cualquier navegador conectados a una red en la que se encuentre tanto el equipo como los comunicadores.

El siguiente paso consistirá en proveer a nuestra infraestructura de un marco que le permita enviar estas alertas de manera dinámica. Para ello deberemos desarrollar un programa que sea capaz de enviar estas peticiones de forma automática, y crear un marco que nos permita realizar un análisis de cuál es el evento registrado y que alerta asociada deberá reproducirse.

3.5.2.2 Modelado del problema.

El problema que se plantea es sencillo de modelar. Por un lado tenemos una cámara que realiza capturas de imagen y mediante un algoritmo que queda fuera de nuestro alcance es capaz de deducir diferentes condiciones. Para simplificar vamos a considerar que la cámara está focalizada en detectar la presencia de una cola sobre una línea de cajas.

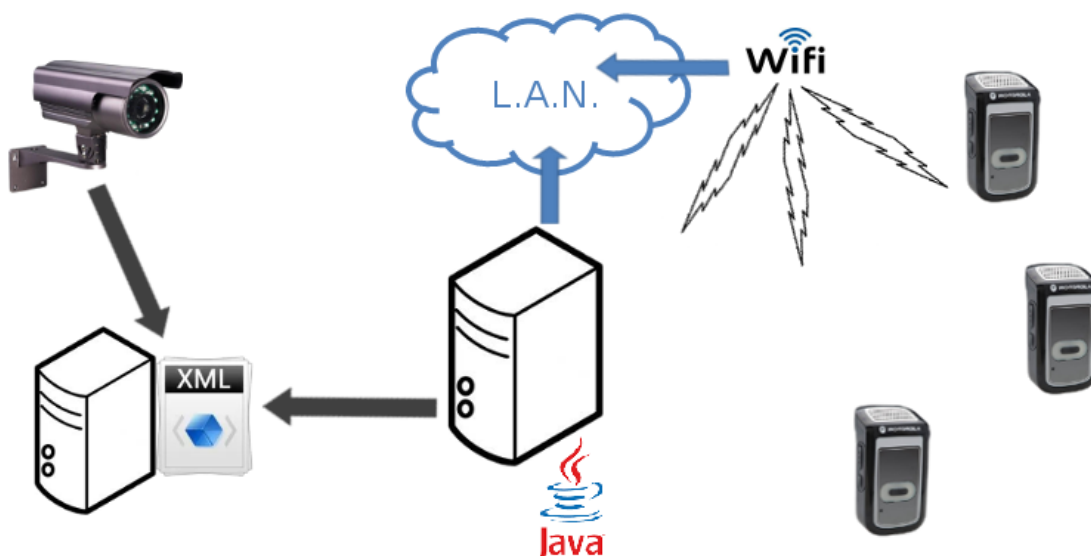
Cuando esta situación (evento) es identificada, genera un fichero xml, con un formato determinado que la codifica. Este fichero es almacenado en un directorio en el servidor que cumple las veces de analítica de las imágenes, que sabemos es Windows. Nuestra aplicación deberá monitorizar dicho directorio a la espera de recepción de dichos eventos y procesarlos. Es decir, deberá comprobar que condiciones representan y enviar un mensaje determinado a un dispositivo o grupo de dispositivos asignados a los operarios, que se ocupa de atender dicha incidencia (siguiendo el ejemplo de la línea de cajas, al personal de caja).



Para aumentar la versatilidad del sistema consideraremos la posibilidad de crear diferentes grupos de recepción, de manera que, siguiendo con el ejemplo del supermercado, se pueda tener un grupo asignado a una línea de cajas ubicadas en una salida, y otros dos grupos en otras salidas. Pero también un grupo seguridad que puede recibir alertas de tipo hurto, incendio, etc.

Y por último, el grupo gerencia, que puede recibir otro tipo de alertas, desde camión esperando para realizar una descarga de productos hasta recibir la alerta de que el parking esta completo.

Ilustración 25: Esquema dispositivos en red



Además, todo el proceso de puesta en marcha y paro de la generación de alertas deberá ser sencilla para los operarios y personal administrativo y podrá realizarse desde cualquier punto de las instalaciones desde sus terminales móviles.

La cámara queda fuera de nuestro alcance, pero los dispositivos VOIP que utilizaremos serán unos dispositivos tipo *walkie-talkie* con funcionalidad *push-to-talk* (PTT) EWB100 pertenecientes a la gama de Motorola, actualmente distribuidos por Zebra Technologies. Estos dispositivos pueden hacer las veces de comunicadores normales, funcionando sobre la infraestructura WiFi, pero también permiten mantener una tabla de memoria interna, donde pueden almacenar un vocabulario que podemos modificar. Y mediante una petición HTTP dar la orden de reproducción de un mensaje al mismo, tal y como describíamos en el punto anterior.

Ilustración 26: Dispositivos EWB-100



3.5.2.3 Detalles de implementación.

Para solucionar el problema planteado teníamos que cumplir una serie de objetivos. Por un lado teníamos que dar con una herramienta que nos permitiera reprogramar la lógica de la aplicación de una manera ágil y sencilla. No solo debíamos poder ampliar el número de mensajes que se reproducen de una manera rápida, sino que además resultaba vital que pudiéramos modificar tanto las ordenes vocales reproducidas, como las condiciones que dan lugar a la necesidad de reproducir dicho mensaje.

Todas estas condiciones que modelan el estado de un sistema las agrupamos en entidades que llamamos eventos. Estos eventos deben ser capturados por diferentes sistemas de sensorización y analizados, de manera que constituyan los disparadores (*trigger*) de diferentes acciones. En el caso de nuestra aplicación todas las acciones que se desencadenan son la reproducción de mensajes de voz. Pero resulta fácil imaginar contextos en los que podría ser importante desencadenar otras acciones, que pueden ir desde el envío de SMS, hasta iniciar acciones sobre un **hardware** específico. Como por ejemplo el bloqueo de una puerta, cierre de compuertas hidráulicas o puesta en marcha de un sistema de apagado de fuego mediante aspersión.

En este punto ya teníamos definidos algunos de los puntos clave de lo que constituiría nuestra aplicación. Por un lado teníamos que capturar toda una serie de eventos provenientes de un mecanismo de sensorización externo, y por otro, tratar dichos eventos generando diferentes acciones. Estos eventos debían ser codificados de manera adecuada, mediante un formato de intercambio de mensajes que debía ser acordado con la empresa externa desarrolladora del elemento de sensorización.

Para ello, y teniendo en cuenta las consideraciones de que la lógica de negocio podía variar a lo largo del desarrollo procedimos a buscar una solución que nos permitiera modificar las condiciones y contenido del mensaje a reproducir de una manera rápida. Y que además requiriera la menor interacción posible, es decir minimizar la necesidad de tener que recompilar el código y realizar un nuevo despliegue de la aplicación.

Después de indagar sobre diferentes sistemas con los que podríamos llevar a cabo tales tareas decidimos que la opción más eficiente es utilizar un sistema de gestión de reglas de negocio (**BRMS**) para realizar la correlación de los eventos registrados. Y de esta manera utilizar las funcionalidades que estos aportan para comprobar el **match** entre evento capturado y regla definida.

El primer paso a la hora de iniciar el desarrollo fue la coordinación de los mensajes que iban a intercambiar la aplicación de sensorización, es decir, de la cámara y nuestra aplicación. Al tratarse de un desarrollo piloto acordamos que los eventos vendrían constituidos por ficheros xml en los que se especificaría la siguiente información:

Analytic, es decir "*tipo de evento*"

y *Source* que codifica el HW origen del evento

A continuación mostramos el contenido de uno de estos ficheros para que podamos comprobar de manera más exhaustiva su estructura.

```
1 |<Alert>
2 |   <Analytic>TAW1</Analytic>
3 |   <Source>3e6dc29a-86d1-4764-bf87-16b0f9567e30</Source>
4 | </Alert>
```


Estos ficheros serían volcados a un directorio de un servidor remoto (**Windows**). De manera que tenemos que montar dicho directorio como un recurso compartido en nuestro servidor **Linux** y proceder a monitorizar dicho directorio.

La monitorización del directorio consistirá en comprobar de manera periódica la aparición de nuevos ficheros que constituirán los eventos capturados. De esta manera nuestra aplicación realiza una monitorización activa del recurso y al comprobar la existencia de un nuevo fichero insertamos el evento registrado en el motor de reglas a través de una sesión **KIE**. Esto lo hacemos en el código de la siguiente manera:

```
⊖ /**metodo que busca todos los eventos (ficheros .xml en la carpeta eventos, los parsea a objeto
   * @param session KieSession en que se insertara el evento */
⊖ private static final void fetchEvents(KieSession session) {
    Alert event = null;
    ArrayList<Alert> events = null;
    try {
        for(File xmlFile : getXmlFilesFromDirectory(XML_DIRECTORY))
        {
            System.out.println("Recogiendo evento de: "+xmlFile);
            event = (Alert)unmarshall(xmlFile);
            System.out.println("Soy el evento: "+event.toString());
            System.out.println("    mi origen es: "+event.getSource());
            System.out.println("    soy de tipo: "+event.getAnalytic());
            session.insert(event);
            //System.out.println("Mover: "+XML_DIRECTORY+"\\"+xmlFile.getName()+" a: " +OLD
            //moveFile(XML_DIRECTORY+"\\"+xmlFile.getName(),OLD_DATA_DIRECTORY+"\\"+xmlFile
            moveFile(XML_DIRECTORY+xmlFile.getName(),OLD_DATA_DIRECTORY+xmlFile.getName());

        }

    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Después de su captura, el evento desaparece del directorio monitorizado y provisionalmente lo depositamos en un directorio local especificado en el fichero de configuración.

Cuando un evento es insertado en el motor de reglas (como el objeto en que es encapsulado, en nuestro caso *Alert*) inmediatamente pasa a su evaluación mediante el motor de Drools. Este proceso es independiente a nuestro desarrollo y como apuntábamos en el marco teórico de esta memoria, se realiza mediante la aplicación de una versión del algoritmo de Rete optimizada para el tratamiento de objetos en programación orientada a objetos.

Como consecuencia de la evolución positiva de las reglas se realiza las acciones que esta tenga asociadas, estas reglas se encuentran almacenadas en un fichero con extensión (.drl) y al que hemos dado nombre de **BussinesRules.drl**.

A continuación mostramos el contenido de estas reglas:

```
package com.sample

import com.damal.Alert;
import com.damal.Correlador;
import com.damal.Dispositivo;

//regla para procesar una cola en el acceso norte
rule "Evento con origen (3e6dc29a-86d1-4764-bf87-16b0f9567e30) de tipo cola en caja"
  salience 8 //prioridad de evento
  when
    m : Alert( Source == "3e6dc29a-86d1-4764-bf87-16b0f9567e30" && Analytic=="OccupancyRateTAW", miOr
  then
    System.out.println("-----");
    System.out.println( "Soy un evento con origen: " + miOrigen );
    System.out.println( " y tipo: " + miTipo );
    //buscar dispositivos a notificar
    for(int i=0;i<Correlador.GRUPO_4.size();i++){
      //no puedo utilizar indexOf de clase ArrayList =>comparacion String/Dispositivo
      //de momento estoy obligado a iterar
      //TODO: buscar solucion mas eficiente --> HashMap??
      //System.out.println("Indice primer for: "+i);
      System.out.println("Dispositivos en grupo: "+Correlador.GRUPO_4.get(i));
      for (int a=0;a<Correlador.devices.size();a++){
        //TODO: delete debug
        //System.out.println("Indice segundo for: " +a);
        //System.out.println("Dispositivo en device: "+Correlador.devices.get(a).getId());
        if(Correlador.GRUPO_4.get(i).compareTo(Correlador.devices.get(a).getId())==0){
          Correlador.devices.get(a).insertEvent("alerta+silence+alerta+cola+en+acceso+norte");
          break;
        }
      }
    }
    System.out.println( "Evento procesado por tener un origen en (3e6dc29a-86d1-4764-bf87-16b0f9567e30) de tipo cola en caja" );
    System.out.println("-----");
end
```

En la clausula "**when**" podemos observar cuales son las condiciones de activación de esta regla, en este caso la recepción de un evento, en cuyos campos **Source** contenga la cadena "3e6dc..." y sea de tipo (**Analytic**) "**OccupancyRateTAW**".

Además mediante el **salience** podemos especificar la prioridad del evento.

Posteriormente a partir del "**then**" se especifican las acciones a desencadenar, que a "*grosso modo*" representan la inserción del evento en la cola de los dispositivos que se encuentren dentro del grupo de difusión GRUPO_4, que según el fichero de configuración "*config.cfg*" se corresponde con la 6 línea cajas_acceso_este: 192.168.0.239 y 192.168.0.240.

```
C:\Users\Rafael\Google Drive\event_final\event_correlator\config.cfg - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
config.cfg
1 event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\events\\
2 old_event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\old_events\\
3 encargado 192.168.0.237
4 seguridad 192.168.0.238
5 cajas_acceso_norte 192.168.0.239 192.168.0.240
6 cajas_acceso_sur 192.168.0.239 192.168.0.240
7 cajas_acceso_este 192.168.0.239 192.168.0.240
```

A partir de este momento el mensaje se encuentra en la cola de los diferentes dispositivos y será reproducido cuando el dispositivo no se encuentre ocupado. Esto será cuando el hilo de ejecución del dispositivo ejecute el método *run*.

```

/** insercion de orden de reproduccion en cola de dispositivo
 * @param msg mensaje codificado a enviar, palabras separada por signo +
 */
public synchronized void insertEvent(String msg){
    queueMsg.offer(msg);
    System.out.println("Insercion en cola : " +msg+ " tamaño cola: " +queueMsg.size());
    notifyAll();//listo para consumir
}

/** hilo activo de ejecucion de dispositivo, si eventos en cola, enva orden de reproduccion*/
public synchronized void run(){
    while (true){
        System.out.println(" Ejecuto run de hilo dispositivo: " +this.getIdent());
        try {
            //Thread.sleep(1000);

            //Verifico si hay datos encolados
            if (queueMsg.size()>0){
                sleep(5000);
                System.out.println("Desencolo de cola en dispositivo: "+this.getIdent());
                //System.out.println("Contenido: "+queueMsg.poll());
                String url="http://" +this.getIdent()+"/audio?play="+queueMsg.poll();
                System.out.println("Reproduciendo: "+url);
                Reproductor r=new Reproductor();
                r.reproduce(url);
                //tiempo de reposo hasta reproduccion nuevo mensaje, calculado como tiempo maximo de mensaje anterior
                sleep(15000);
                //System.out.println("Despierto");
            }
            else{
                System.out.println("Cola vacia me mantengo a la espera en dispositivo: "+this.getIdent());
                wait();
            }
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}

```

Como comentábamos en la sección anterior los dispositivos **EWB100** contienen un servidor web mínimo que permite la creación de conexiones **HTTP** al mismo. Cada conexión deberá contener codificada los argumentos y órdenes que se ejecutarán de manera remota en el dispositivo

Estas órdenes constituyen una serie de rutinas simples que es capaz de llevar a cabo el dispositivo, devolviendo en su caso una respuesta informativa. A pesar de que en principio el servidor puede aceptar múltiples conexiones en paralelo, la propia naturaleza de las operaciones (reproducción de mensaje, control de los led, ejecución de comandos CLI, etc.), dificultará el procesado de dichas operaciones de manera paralela. Es por ello que necesitamos encolar las diferentes alertas que van a ser reproducidas.

En este punto simplemente hacer constar que para reproducir un audio de manera remota a partir del contenido de los clips de audio precargados en los dispositivos deberemos crear una petición http con el siguiente formato:

http://192.168.0.104/audio?play=Manager+to+register+5

Tal y como haríamos en un navegador web, pero de manera asíncrona, de forma que no esperemos respuesta por parte del servidor, ya que esta respuesta bloquearía los diferentes hilos de ejecución de la aplicación.

En las secciones de código mostradas anteriormente podemos visualizar cómo este mensaje es generado. Por un lado en las reglas ya especificamos las órdenes vocales a reproducir:

```
if(Correlador.GRUPO_3.get(i).compareTo(Correlador.devices.get(a).getIdent())==0){  
Correlador.devices.get(a).insertEvent("alerta+silence+alerta+cola+en+acceso+sur");
```

y luego el método run() del dispositivo antepone la orden de ejecución a esta cadena que constituye los argumentos:

```
String url="http://" + this.getIdent() + "/audio?play="+queueMsg.poll();  
System.out.println("Reproduciendo: "+url);  
Reproductor r=new Reproductor();  
r.reproduce(url);
```

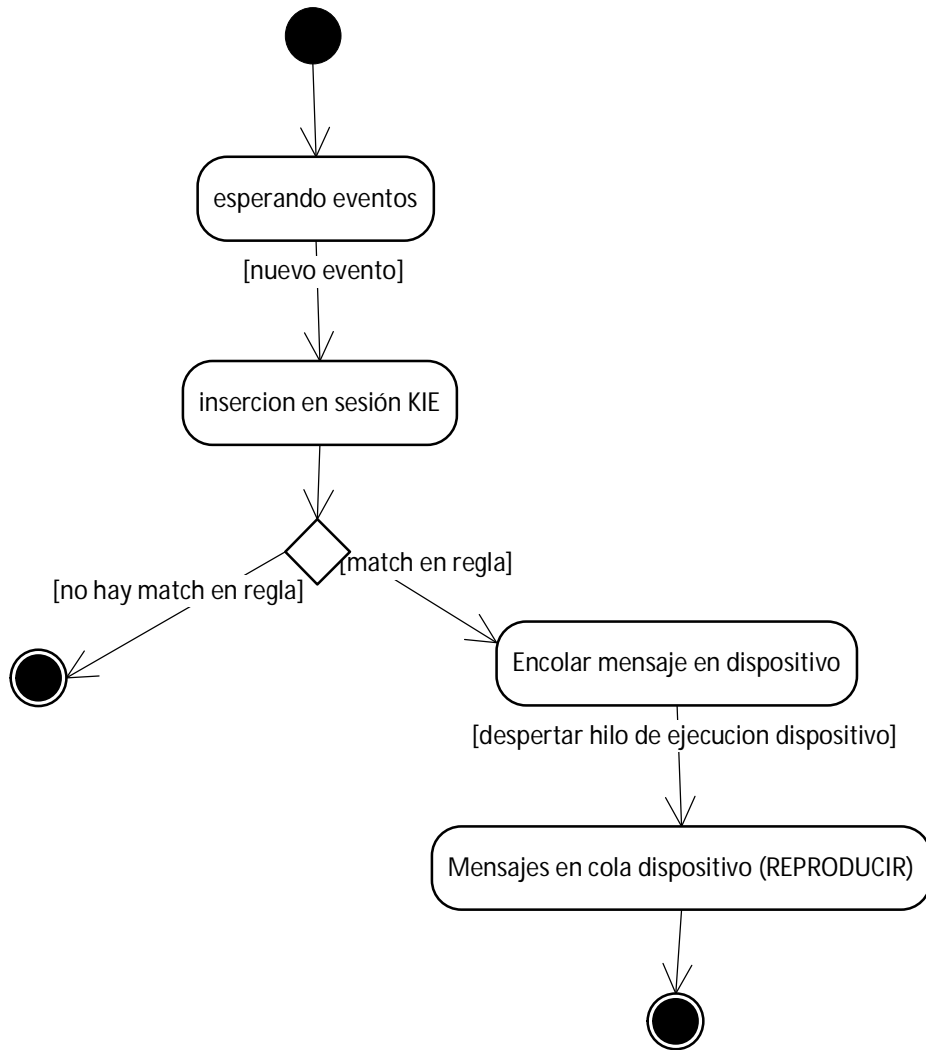
Seguidamente es ya el motor de reproducción (clase **Reproductor**) el que genera la petición HTTP a los dispositivos, emitiéndose las órdenes vocales especificadas.

3.5.2.4 Diagramas y especificación de funciones.

i. Diagrama de estados.

En la siguiente imagen podemos comprobar cuáles son los estados por los que va transitando el sistema de manera global, puesto que cada dispositivo manejará un hilo de ejecución independiente:

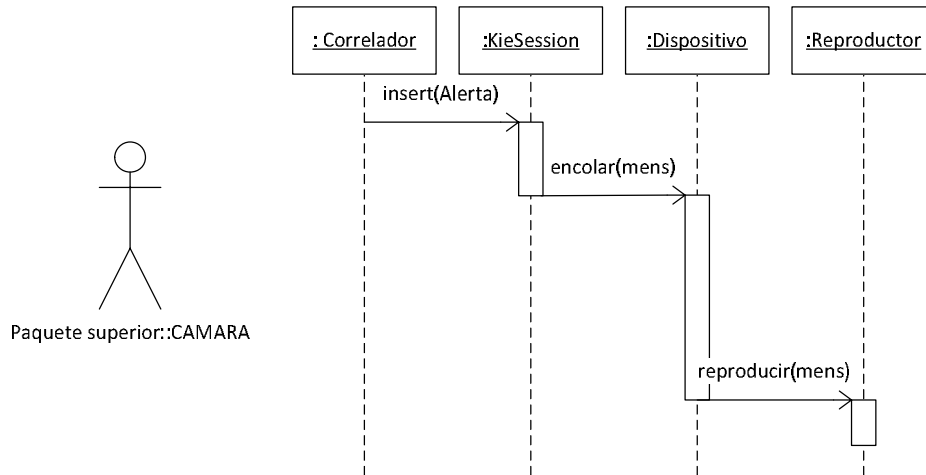
Ilustración 27: Diagrama de estados DTE



ii. Diagrama de secuencia.

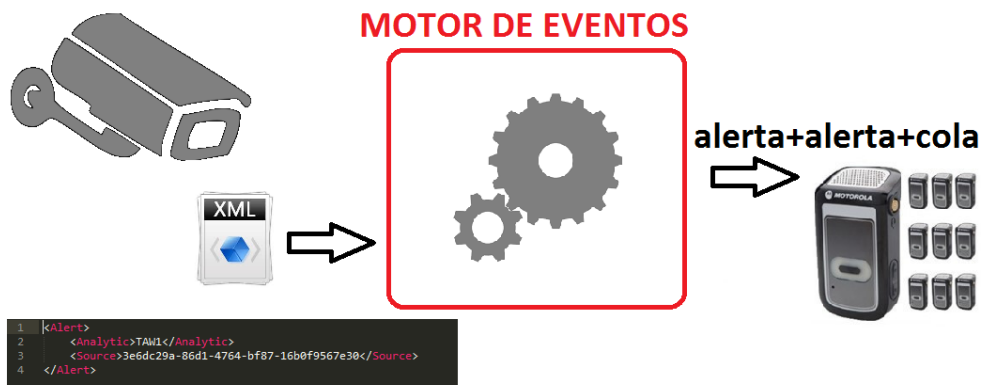
En la siguiente imagen podemos ver el intercambio de mensajes que se da entre las diferentes entidades que componen nuestro sistema:

Ilustración 28: Diagrama de secuencia



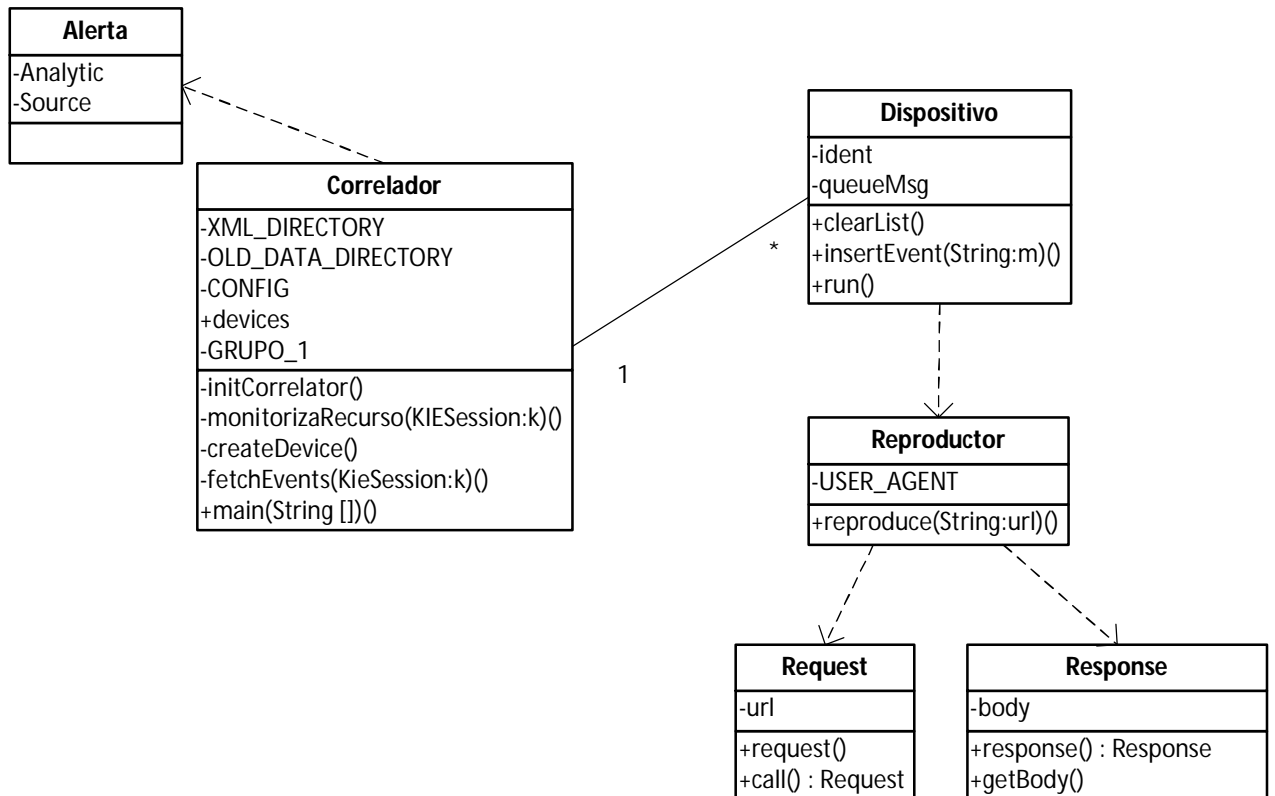
Dónde la clase Reproductor realiza el envío de la orden de reproducción a los dispositivos, de acuerdo al siguiente esquema global:

Ilustración 29: Esquema de alto nivel de los mensajes intercambiados por el Hardware del sistema



iii. Diagrama de funciones. UML

Ilustración 30: Diagrama UML



Como vemos en la imagen la única relación fuerte entre clases que existe se da entre las clases **Correlador** y **Dispositivo**, ya que la clase Correlador mantiene una relación de asociación de uno a muchos con la clase Dispositivo. Mientras que el resto de relaciones son de tipo débil representando una relación de dependencia que muestra la relación existente entre un cliente y el proveedor de un servicio usado por este cliente.

En este caso tanto las clases **Request**, **Response**, como **Reprodutor** son usadas como un servicio. La clase Dispositivo instancia a la clase Reprodutor cuando quiere hacer uso de ella, pero en ningún momento mantiene la relación después de que esta le provea del servicio solicitado, en este caso el envío de orden de reproducción de mensaje.

Así mismo la clase **Alerta** se utiliza meramente como medio de encapsulación del objeto alerta antes de ser insertado en el motor de correlación de eventos.

iv. Especificación por interfaz función.

En esta sección vamos a mostrar un breve resumen mediante la especificación por interfaz-función de las funciones desarrolladas. También daremos algunos detalles pormenorizados sobre la implementación de las clases. No entraremos en gran detalle acerca de la implementación de las mismas, únicamente presentaremos una breve descripción en vista a poder comprender que uso tienen dentro del sistema.

Paquete: com.damal

Clase : Alerta

Encapsula el tipo de objetos Alerta, que son insertado en la sesión KieSession especificada.

Contiene los métodos típicos de representación de objetos en **Java**, constructor, getters y setters. Y está codificada mediante anotaciones de manera que los ficheros **XML** sean convertidos en objetos directamente de este tipo gracias a la clase *javax.xml.bind* y mediante una notación específica.

En la versión inicial del desarrollo presentaba un método "toVoice" que se encargaba de enviar la orden de voz directamente al dispositivo. Actualmente utilizamos una clase auxiliar, "Reproductor" para gestionar una cola de alertas que esperan poder ser atendidas.

Clase : Correlador

Presenta la mayor parte de funcionalidad del sistema se encarga de la recuperación de ficheros (eventos), los inserta en una sesión **KIE** y mueve el fichero origen a un directorio destino.

Función: initCorrelator

Entrada:-

Salida: KieSession session

Acción: Inicia una sesión **KIE**, (motor de correlación de eventos de **Drools**).

Crea la sesión, carga la base de conocimiento "BussinesRules.drl" e inicia un hilo de ejecución que mantiene la sesión activa y en espera de la llegada de nuevos eventos a procesar.

Función: monitorizaRecurso

Entrada: KieSession session

Salida: -

Acción: mantiene un hilo de ejecución que periódicamente inserta los eventos recogidos en la sesión activa.

Función: main

Entrada: String[]

Salida: -

Acción: carga la configuración del sistema especificada en el fichero "config.cfg", muestra por pantalla dicha configuración y crea e inicia el hilo de ejecución de los diferentes dispositivos detectados.

Función: main

Entrada: String[] args

Salida: -

Acción: carga la configuración del sistema especificada en el fichero "config.cfg", muestra por pantalla dicha configuración y invoca los métodos encargados de la creación de los objetos Dispositivo e inicia el hilo de ejecución de los mismos.

Función: createDevice

Entrada: -

Salida: -

Acción: creación de los objetos Dispositivo.

Función: getDevicesIP

Entrada: -

Salida: -

Acción: a partir de la lectura del fichero de configuración obtiene las **IPs** de los dispositivos sin duplicados.

Función: fetchEvents

Entrada: KieSession session

Salida: -

Acción: recoge los eventos del directorio monitorizado y los inserta en la sesión especificada, moviendo el fichero origen al directorio destino a partir de unas variables privadas de clase.

Función: getXmlFilesFromDirectory

Entrada: String xmlDirectory

Salida: File[]

Acción: escanea el directorio a monitorizar y devuelve los ficheros encontrados como objeto **File**.

Función: unmarshall

Entrada: File fich

Salida: Alert

Acción: convierte los ficheros xml en objetos **Alert** gracias a la clase "javax.xml".

Función: moveFile.

Entrada: String source, String destination.

Salida: -

Acción: mueve el fichero establecido en la ruta origen al destino especificado.

Función: getConfig

Entrada: -

Salida: -

Acción: carga la configuración del sistema en las diferentes variables encapsuladas en el objeto Correlador (directorio origen eventos, destino evento procesados y configuración de grupos).

Clase : Dispositivo

Mantiene el hilo de ejecución de los diferentes dispositivos y una cola de alertas almacenadas sobre los mismos.

Función: clearList

Entrada:-

Salida: -

Acción: vacía la cola de alertas (mensajes) almacenados en el objeto Dispositivo.

Función: insertEvent

Entrada: String msg

Salida: -

Acción: encola la alerta en la cola del dispositivo, la alerta es una codificación como string del mensaje. Por ejemplo: "alerta+cola+en+acceso+norte". Método "synchronized", despierta el hilo de ejecución del dispositivo cuando detecta inserción de evento.

Función: run

Entrada: String msg

Salida: -

Acción: método "synchronized", desencola las alertas de la cola de dispositivo y reproduce su contenido sin posibilitar interrupción de mensaje durante 15 segundos. Cuando la cola se encuentra vacía suspende el hilo de ejecución.

Clase : Reproductor

De esta clase tampoco vamos a entrar en detalles, simplemente hacer notar que los objetos representados por esta clase son utilizados como servicio por la clase dispositivo generando la petición http al servidor web interno de cada dispositivo.

3.5.2.5 Prototipo de la aplicación web de configuración

Como apuntábamos anteriormente, uno de los puntos claves de la aplicación es disponer de una utilidad que permita poner en marcha el servicio que realizará la correlación de los eventos en nuestro servidor remoto. Para ello, deberá iniciar la ejecución del programa que hemos desarrollado en **Java** y también permitirá matar este proceso cuando lo queramos parar. Además de esto, debemos tomar en consideración que mientras el proceso se encuentra parado el directorio en el que recibimos los eventos seguirá recibidos, de manera que deberemos purgar el contenido de dicho directorio antes de reiniciar la aplicación. De esta manera eliminaremos los ficheros antiguos de manera que no generen alertas.

Después de estudiar diferentes opciones para desarrollar estas tareas, establecimos que la forma más transparente y sencilla para el usuario sería acceder a una consola de administración que no sólo le permitiera poner en marcha y parar el procesamiento de eventos, sino además también configurar los grupos de recepción y envío de mensajes de demostración.

Ilustración 31: Interfaz de configuración de grupos

Configuración de grupos Iniciar/Parar Ejemplos de comandos de voz Envío directo TTS Envío directo TTS v2

Configuración básica de Grupos de Recepción

Asigne la IP de los dispositivos a los diferentes grupos.

IP's Dispositivos GRUPO 1: (Gerencia)
192.168.0.240 192.168.0.238 192.168.0.239

IP's Dispositivos GRUPO 2: (Seguridad):
192.168.0.240 192.168.0.238 192.168.0.239

IP's Dispositivos GRUPO 3: (Cajas 1)
192.168.0.240 192.168.0.238 192.168.0.239

IP's Dispositivos GRUPO 4: (Cajas 2)
192.168.0.239

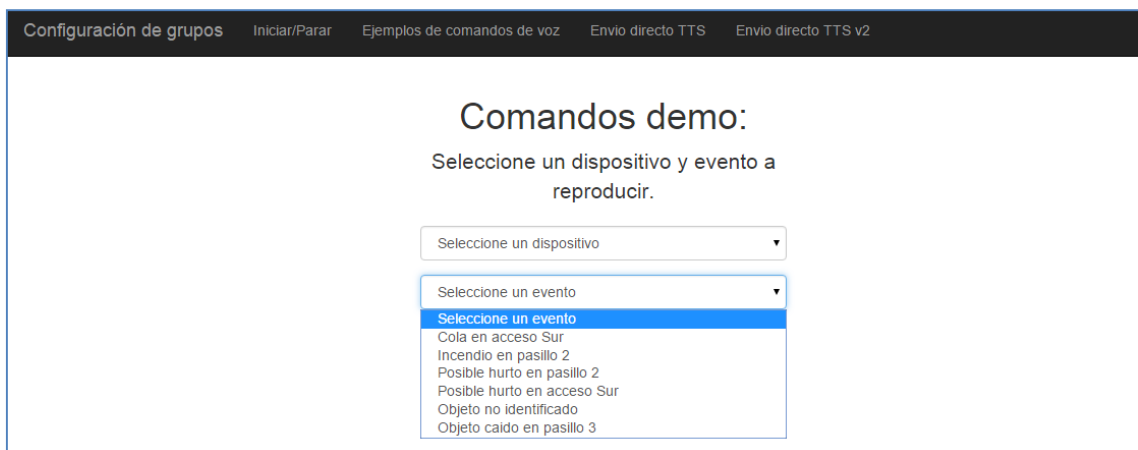
IP's Dispositivos GRUPO 5: (Cajas 3)
192.168.0.239

Grabar cambios

Ilustración 32: Interfaz de puesta en marcha de la aplicación



Ilustración 33: Interfaz envío de mensajes de demostración



Pero una vez con estas funcionalidades básicas cubiertas, decidimos dotar de un valor añadido a la aplicación creando la funcionalidad de poder enviar textos personalizados a los diferentes dispositivos. De esta manera, el operario puede seleccionar un dispositivo e introducir el mensaje que desea enviar.

Para ello, en primera instancia creamos una pestaña que nos permitiera introducir vía teclado las diferentes palabras pertenecientes al vocabulario cargado que se puede utilizar:

Ilustración 34: Consola envío de mensajes por teclado

Configuración de grupos Iniciar/Parar Ejemplos de comandos de voz Envío directo TTS Envío directo TTS v2

Texto a voz:

Seleccione un dispositivo y introduzca el texto a reproducir.

Vocabulario reconocido:

alerta acceso caído caja cola hurto identificado no sin
norte sur este oeste pasillo posible objeto
bloqueada volcado articulo existencias fuego puerta

1 2 3 4 5 6

192.168.0.238 ▼

Introduzca el texto que desea enviar al dispositivo

Alerta cola en caja sur

Enviar

Pero posteriormente y viendo que los operarios operaban principalmente con un terminal móvil, decidimos que esta forma de trabajar era poco operativa y resultaría más sencillo que simplemente seleccionaran las palabras pulsando sobre ellas en la pantalla táctil de su terminal.

Ilustración 35: Consola envío de mensajes desde terminal móvil

Configuración de grupos Iniciar/Parar Ejemplos de comandos de voz Envío directo TTS Envío directo TTS v2

Texto a voz:

Seleccione un dispositivo y introduzca el texto a reproducir.

Vocabulario reconocido:

alerta acceso caído caja cola hurto identificado no
sin norte sur este oeste pasillo posible objeto
bloqueada volcado articulo existencias fuego puerta

1 2 3 4 5 6

Seleccione un dispositivo ▼

Introduzca el texto que desea enviar al dispositivo

Enviar Resetear

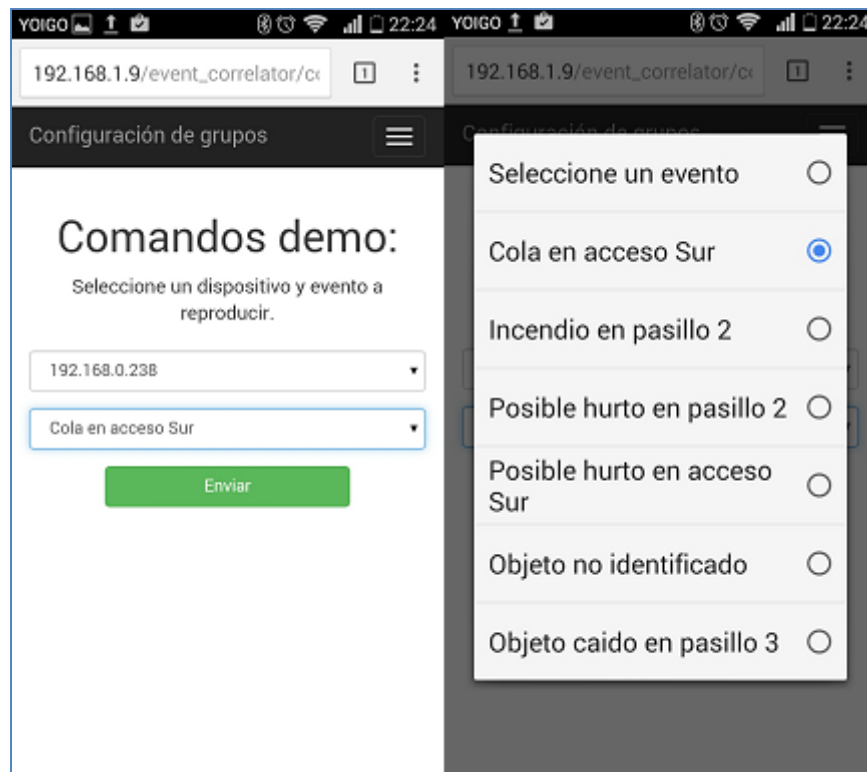
Cabe reseñar que en este punto de desarrollo decidimos conservar ambas formas de interactuar, ya que al tratarse de un desarrollo piloto de esta manera podremos preguntar a los operarios sobre como es su forma de interactuar con la aplicación.

Como desarrollo piloto la consola de configuración no se encuentra limitada en acceso, es decir, cualquier operario puede acceder a la misma y eliminar su IP dentro de los grupos. Con esto conseguimos que si un operario concreto quiere dejar de estar suscrito a la recepción de mensajes el mismo puede acceder mediante la interfaz web y eliminar su dispositivo de los grupos de recepción a los que pertenezca.

De cara al futuro resultaría interesante dotar a esta interfaz de un mecanismo de autenticación, de manera que podamos dar de alta diferentes usuarios con diferentes permisos para interactuar con la misma. Asimismo también podemos dotar a la aplicación de funcionalidades añadidas, como pueda ser permitir la adición y eliminación de diferentes grupos de recepción de mensajes o permitir el envío de mensajes de manera directa ya no sobre un dispositivo concreto, sino sobre todo los dispositivos que se agruparan bajo un grupo de recepción.

Además la consola desde interfaz web ha sido desarrollada para ofrecer un diseño web adaptable al tamaño de la pantalla de diferentes dispositivos (*Responsive Design*).

Ilustración 36: Visualización consola desde terminal móvil



3.5.3 Metodologías de desarrollo

Al tratarse de un **desarrollo piloto** estamos utilizando un enfoque muy abierto. El trabajo presentado constituye la segunda iteración de un proyecto que sigue en fase de desarrollo. Lo que se presenta en esta memoria no es un producto final, sino un desarrollo piloto que pretende capturar la opinión de los operarios y personal no técnico con el fin de evaluar las posibles utilidades de un sistema con las características que presentamos.

El ámbito de aplicación ha sido el sector alimentario (tienda de alimentación) porque se trata del principal cliente de la empresa donde se realizó este proyecto. Pero a lo largo del proceso de desarrollo hemos ido ofreciendo la solución a diferentes clientes que nos han ofrecido mediante su *feedback* nuevas posibles áreas de aplicación.

De esta manera, hemos desarrollado un **producto mínimo viable** (MVP) que nos permita hacer un despliegue piloto en diferentes clientes y a partir de sus requerimientos dirigir el desarrollo de una manera personalizada. De este modo, pretendemos que nuestros clientes guíen el desarrollo. Nosotros nos hemos centrado en proporcionar una imagen de conjunto de lo que se podría hacer a través de la notificación por voz extrayendo conocimiento de los eventos que podamos capturar.

De esta forma, a través de nuestros clientes podremos seleccionar aquellas características que realmente aportan valor a nuestro producto, siguiendo las directrices del *Lean Developing* mediante un proceso de desarrollo ágil de manera iterativa e incremental.

Esta metodología de desarrollo es una adaptación de los principios y las prácticas de la forma de producción industrial Lean, enfocada hacia el área del desarrollo software. Inicialmente este sistema de producción fue originado en el Sistema de Producción de Toyota y ahora, gracias al apoyo recibido por parte de la comunidad Agil está adquiriendo gran representatividad.

De las metodologías de desarrollo Lean hemos tomado las siguientes características fundamentalmente:

Eliminar los desperdicios: Todo lo que no añade valor al cliente se considera un desperdicio.

Ampliar el aprendizaje: Integración del cliente en el ambiente de desarrollo para concentrar la comunicación en las soluciones futuras y no en las soluciones posibles.

Decidir lo más tarde posible: Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación. La velocidad asegura el cumplimiento de las necesidades actuales del cliente y no lo que éste requería para ayer.

Reaccionar tan rápido como sea posible: La velocidad asegura el cumplimiento de las necesidades actuales del cliente y no lo que éste requería para ayer.

El cliente debe tener una experiencia general del sistema.

Visión de conjunto: *"Piensa en grande, actúa en pequeño, equivócate rápido; aprende con rapidez."*

De este modo, hemos tenido que coordinar el despliegue con un elemento de sensorización que en principio no presentaba las funcionalidades necesarias para la generación de los eventos que necesitábamos procesar. En este sentido, nos

encontramos en disposición de colaborar con cualquier otro fabricante *hardware* para poder integrar una solución conjunta, de una manera rápida y ágil.

Como apuntábamos en la sección "**Análisis de las opciones disponibles en el mercado**" no hemos encontrado producto alguno que pueda llevar a cabo la funcionalidad de nuestro desarrollo.

En este punto nos encontramos sobre la segunda iteración del producto. La primera iteración consistió en una demo muy básica con un único dispositivo y un único emisor de eventos, de manera que no existía posibilidad de colisión entre mensajes. Pero rápidamente detectamos la necesidad de dotar al producto de un sistema de gestión de colas en el dispositivo. De manera que las diferentes peticiones sobre los dispositivos fueran almacenadas y atendidas sin colisión, de esta forma repercutíamos mínimamente sobre el requisito de tiempo real, pero evitamos la pérdida de mensajes en entornos donde se generan gran número de mensajes que pueden colisionar, al no ser los dispositivos capaces de atender más de una petición simultánea.

En la sección "**Conclusiones y posibles ampliaciones al proyecto**" se plantean algunas de las posibilidades que hemos identificado para integrar nuestro desarrollo en diferentes sistemas y aplicaciones.

3.6 Manual de puesta en funcionamiento.

3.6.1 Instalación.

El proceso de instalación es muy sencillo, únicamente tendremos que coger todos los ficheros que se distribuyen en el Cd en el directorio "*interfaz_web*" y volcarlos en el servidor web que hemos preparado previamente.

Para el escenario descrito vamos a hacer una instalación sobre entorno Linux, pero los pasos necesarios para poner en funcionamiento la aplicación en otro entorno serían muy similares.

Cabe hacer notar que estamos utilizando un servidor propio, realmente un equipo portátil con una instalación de Linux Debian, con el servidor web Apache y PHP instalado, y con la última versión disponible de Java.

Por ejemplo, para el despliegue que hemos hecho nosotros la ruta donde el servidor aloja los documentos *html/php* por defecto es: `"/var/www"`. Cuando todos los ficheros se encuentran en el directorio ya tendremos tanto la interfaz de configuración como el programa Java, que incluye la monitorización del directorio de recepción de eventos, el procesamiento de los mismos y generación de alertas acústicas.

El volcado de estos ficheros lo podemos hacer mediante **FTP** por ejemplo si tenemos acceso al servidor habilitado mediante este protocolo. También lo podríamos hacer por ejemplo utilizando la utilidad **WinSCP** desde entorno Windows o mediante el comando *scp* si nos conectamos mediante *ssh* desde otro equipo Linux.

Adicionalmente se deberá crear en el servidor anfitrión un directorio donde se recibirán los eventos, en nuestro caso esto lo hacemos en: `/home/correlador-cam/eventos`, pero podría tratarse de cualquier otro directorio.

Además, para el montaje definitivo lo que hacemos es montar un directorio compartido con el servidor de la cámara, esto lo hacemos tal y como se detalla en el siguiente apartado: "**Preparando el entorno**", apartado 3 **Configurando el servicio web (LINUX)**.

3.6.2 Preparando el entorno.

i. Preparando los dispositivos

El primer paso consiste en conseguir los ficheros de audio que vocalizan las diferentes palabras que compondrán la base de vocabulario contenida en los dispositivos. Para ello, podríamos buscar directamente estos ficheros en formato *wav* de manera preferente, para evitarnos tener que realizar la conversión entre formatos o acudir a una de las múltiples utilidades que te permiten transformar texto escrito a habla.

Nosotros nos decantamos por esta segunda opción, pero cualquiera de las dos es igualmente válida si obtenemos como resultado final un fichero *wav* con las siguientes características que codifique el audio:

8KHz sampling rate

16 bit words

Mono

El uso de otros formatos de audio puede producir resultados impredecibles.

Aunque el dispositivo soporta ficheros codificados en múltiples formatos (*g729*, *g711u*, *g711a*, etc.), normalmente los clips de audio se codifican en formato *G729* que consume 1Kb/sec, mientras que el formato *G711* consume 8Kb/sec. Cabe reseñar que los ficheros de audio dentro del dispositivo se organizan en diferentes tablas. Cada una de estas tablas almacena clips de audio de un mismo formato y contiene la referencia al lugar donde están almacenados los clips en la memoria del dispositivo. De esta manera existen tablas para clips en formato *g729*, *G711*, etc.

Para más información a este respecto leer el documento *EWB100_Usage_and_Deployment_Guide_v1.13.pdf*, que ofrece detalles de cómo se organizan los diferentes sistemas de memoria del dispositivo (flash, RAM, etc.) y su contenido. Este documento se puede encontrar en la carpeta "*Documentacion dispositivos EWB100*" en el *cd* que se distribuye con esta memoria.

Pero es importante quedarnos con la idea de que el nombre que contendrá el fichero *wav* deberá ser coherente con el vocablo que reproducirá, y preferiblemente sin usar acentos u otros caracteres especiales. Es decir, si queremos vocalizar la palabra "*alerta*", el nombre del fichero deberá ser "*alerta.wav*".

Una vez tenemos todos los ficheros de audio que queremos que compongan el diccionario de nuestros dispositivos los ubicamos dentro de un mismo directorio. Dentro de este directorio también tendremos que crear un fichero de texto en el que aparecerá el nombre de todos los clips de audio que vamos a cargar.

El contenido de este fichero consistirá en una palabra-fichero por línea, permitiéndose líneas en blanco. Si bien, resulta vital que la primera línea contenga la

cadena "--729", que representa la codificación escogida en que se traducirán los ficheros de audio.

Ilustración 37: Fichero txt para creación del audioclip



```
tts_build.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
--729
acceso.wav
alerta.wav
caido.wav
caja.wav
cola.wav
este.wav
hurto.wav
identificado.wav
no.wav
norte.wav
objeto.wav
oeste.wav
pasillo.wav
posible.wav
sur.wav
norte.wav
en.wav
1.wav
2.wav
3.wav
4.wav
5.wav
6.wav
volcado.wav
bloqueada.wav
articulo.wav
sin.wav
existencias.wav
fuego.wav
puerta.wav
```

Después de esto se requiere hacer uso de 3 programas:

audiomsgcreate

crccalc

cat

El primero de estos programas (*audiomsgcreate*) construye el fichero de clips de audio. El segundo (*crccalc*) genera la cabecera requerida para la programación flash del dispositivo, y el tercero (*cat*) combina los dos ficheros creados en los pasos anteriores para construir un fichero descargable sobre el dispositivo. Todos estos programas se ejecutan por línea de comandos de Windows.

La secuencia de pasos a seguir es la siguiente:

```
audiomsgcreate -l tts_build.txt -binary
```

```
crccalc g729AudioTable.bin 1.1.1030 06630080 111 Parrot "07/03/15" "10:20:00"
```

Cuyo significado es el siguiente:

1.1.1030 (versión)

"07/03/15" (fecha de creación)

"10:20:00" (hora de creación)

y son valores reemplazables por el usuario, sin embargo son de obligado uso y deberán especificarse en el formato que se especifica.

```
cat header.dat g729AudioTable.bin > audioclipimage.bin
```

Este último fichero ya puede ser cargado en los dispositivos mediante USB o *Airbeam*. Este último es el sistema de Motorola mediante el cual se puede distribuir aplicaciones y actualizaciones en dispositivos móviles sin necesidad de cables. Y que se puede ver como aparece configurado en la sección "**Preparando el entorno de red**".

Por último, simplemente hacer notar que los nombres de los ficheros se pueden personalizar para que adquieran significado para la persona que realice el despliegue.

Ilustración 38: Consola de configuración de los dispositivos EWB100

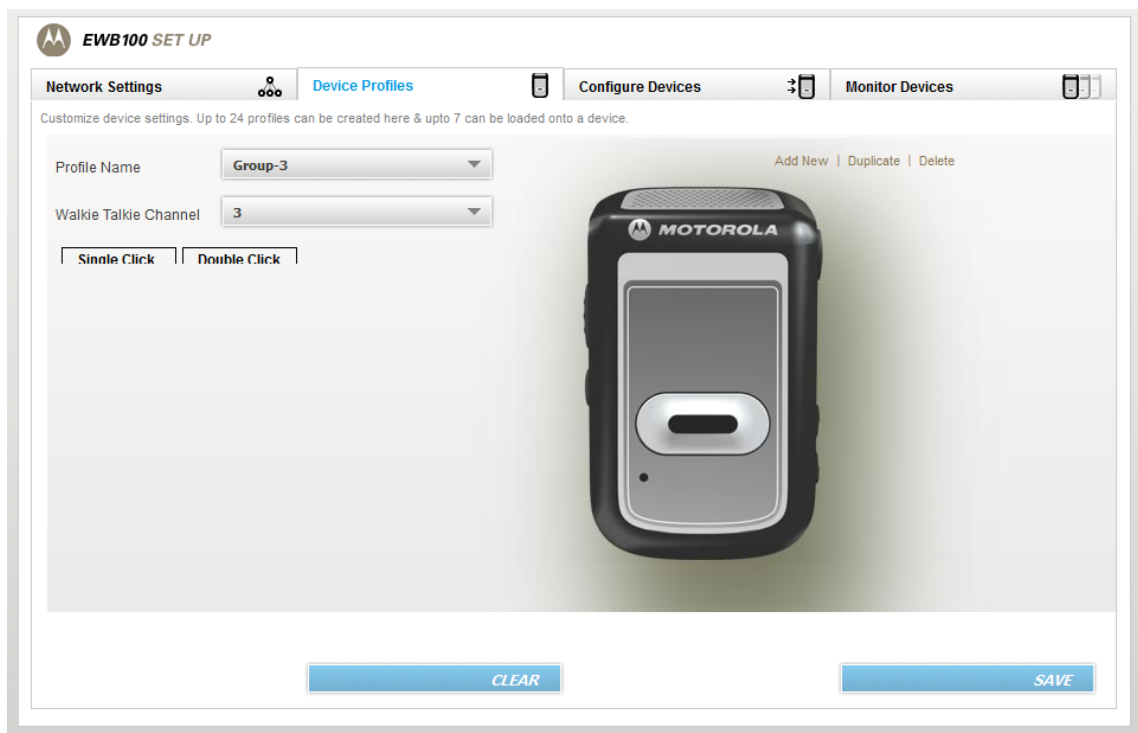
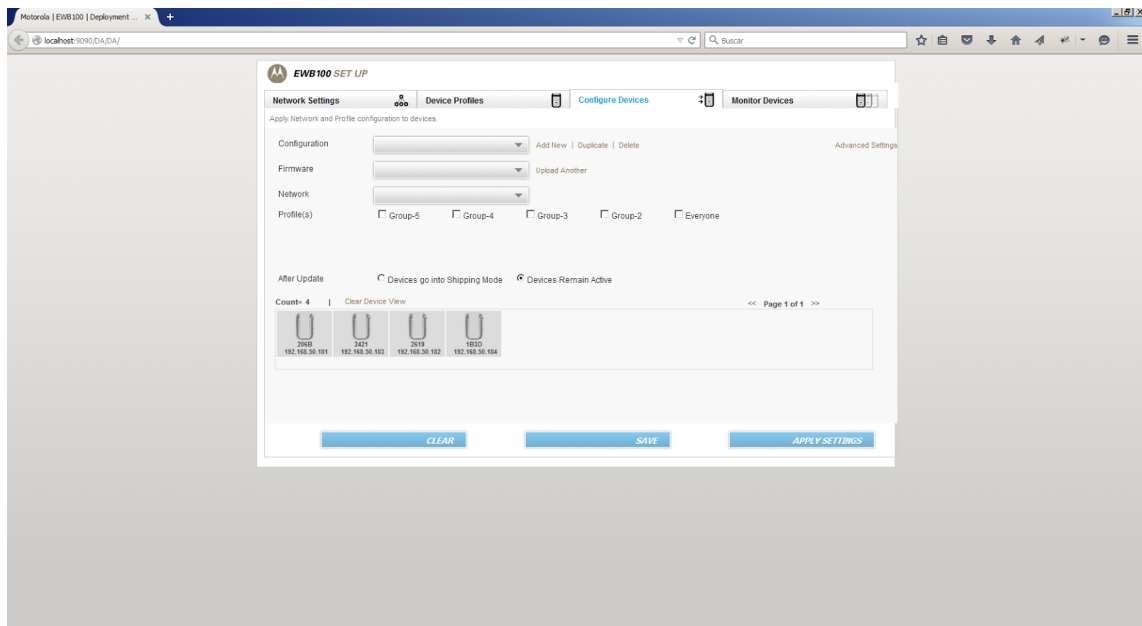


Ilustración 39: Consola de carga de diccionario en dispositivos



ii. Preparando el entorno de red

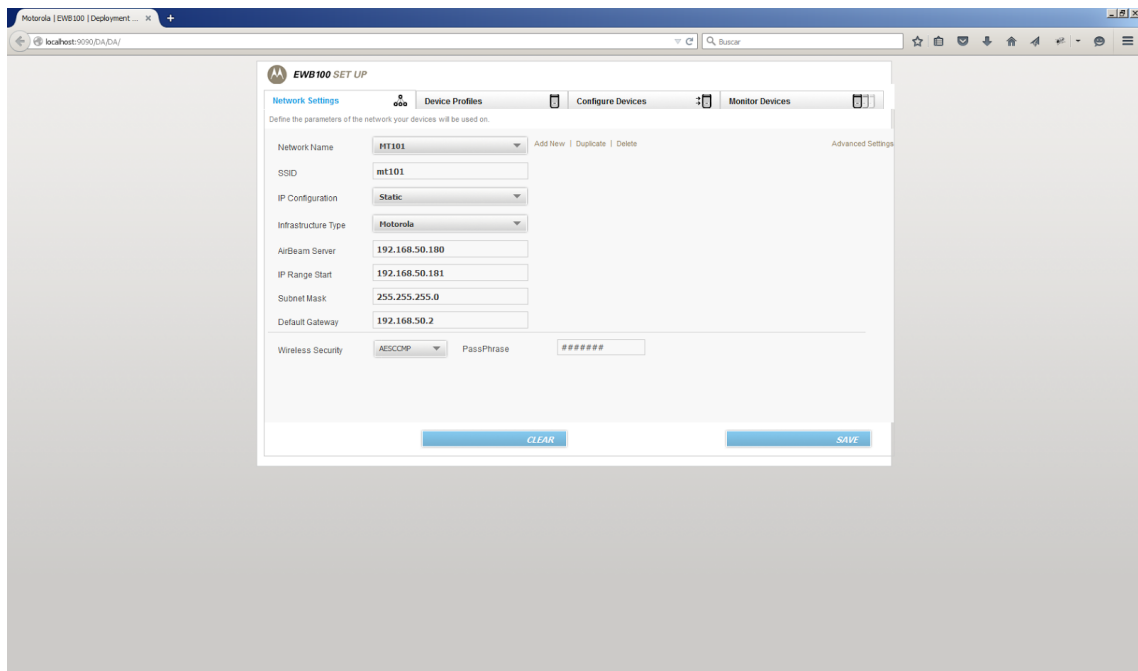
Los dispositivos de voz funcionan sobre infraestructura Wi-Fi, de manera que deberemos disponer de la misma para poder poner en funcionamiento el sistema de alertas por voz. Tanto el servicio **Java** como los dispositivos deberán formar parte de la misma LAN, de manera que tengan visibilidad entre ellos y se puedan comunicar.

Para configurar los dispositivos y dotarles de una dirección IP dentro de la infraestructura inalámbrica podemos hacer uso de la aplicación de despliegue (**Deployment Application DA**), que se distribuye con el paquete o bien mediante línea de comandos conectándonos directamente por **Telnet** a los dispositivos con tal de configurarlos bajo una IP estática.

Ilustración 40: Motorola EWB-100 Deployment Application



Ilustración 41: Consola de configuración de red y servicio Air Beam



iii. Configurando el servicio web (LINUX).

El primer paso para tener el entorno listo consiste en crear el directorio donde recibiremos los ficheros xml que constituyen los eventos, como estamos en un entorno Linux esto lo podemos hacer de manera cómoda mediante el mandato *mkdir*. Además como en nuestro caso no ubicaremos este directorio en un lugar al que tenga acceso el usuario **www-data** de Apache, tendremos que editar el fichero */etc/sudoers*, hay múltiples formas de hacer esto. Pero al tratarse de un fichero crítico del sistema de ficheros Linux lo editaremos mediante *visudo*, y simplemente añadiremos al final del mismo la siguiente línea, para que este usuario pueda ejecutar cualquier acción sin que le sea preguntada la contraseña.

```
www-data ALL=(ALL) NOPASSWD:ALL
```

En el escenario planteado para el montaje final, nuestra aplicación debía establecer conexión con un directorio remoto del que leería y procesaría los ficheros xml. Para ello tenemos diferentes opciones, pero finalmente decidimos montar el directorio mediante cifs para que sea visto desde el servidor como un directorio local. Para ello podemos ejecutar la siguiente sentencia en el servidor (siempre con permisos administrativos):

```
mount -t cifs //192.168.1.13/eventos -o username=ravipas,password=xxxx
```

Dónde 192.168.1.13 representa la dirección **IP** del servidor remoto, y */eventos* el directorio remoto donde se depositan los distintos ficheros xml que codifican los eventos.

A partir de este momento ya podemos editar el fichero de configuración *config.cfg*, para especificar el directorio que deseamos monitorizar, el directorio donde se volcarán los ficheros procesados y las IP's de los dispositivos asociados a los diferentes grupos de difusión.

Si queremos que no se pierda la conectividad entre las aplicaciones también podemos editar el fichero *fstab*, de esta manera si se produce el reinicio de cualquiera de los servidores el directorio especificado será inmediatamente montado y por tanto la aplicación configurada.

```
nano /etc/fstab
```

y añadir al final del mismo la siguiente línea:

```
//192.168.1.13/eventos /home/correlador-cam/eventos cifs  
noserverino,rw,icharset=utf8,password=xxxxx,username=ravipas,uid=0 0 0
```

Además, podemos programar el sistema para que ponga la aplicación en marcha en el arranque, esto nos servirá por si por cualquier razón el servidor es reiniciado de una forma ajena a nuestro control. Para ello simplemente tenemos que editar el fichero */etc/rc.local* y añadir el siguiente contenido:

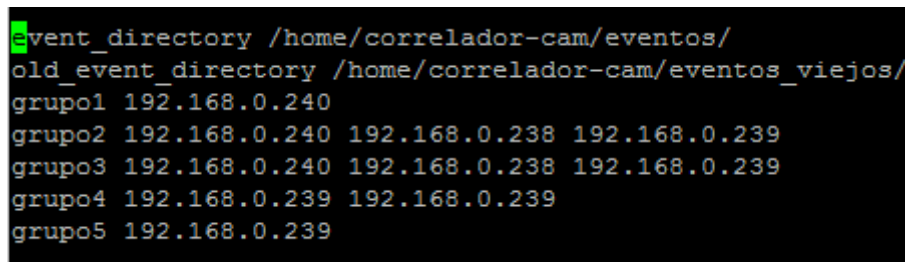
```
cd /var/www/html/event_correlator
```

```
sudo java -jar /var/www/html/event_correlator/correlador-camDamalv1.1.jar
```

```
exit 0
```

Así mismo debemos asegurarnos de que el fichero de configuración "*config.cfg*" apunte de manera adecuada a este directorio y de que exista el directorio en el que se volcarán los eventos ya procesados.

Ilustración 42: Contenido del fichero de configuración



```
event_directory /home/correlador-cam/eventos/  
old_event_directory /home/correlador-cam/eventos_viejos/  
grupo1 192.168.0.240  
grupo2 192.168.0.240 192.168.0.238 192.168.0.239  
grupo3 192.168.0.240 192.168.0.238 192.168.0.239  
grupo4 192.168.0.239 192.168.0.239  
grupo5 192.168.0.239
```

De esta manera ya podremos acceder desde la consola web para iniciar y parar el motor de correlación de eventos. Para ello simplemente debemos escribir la ruta en cualquier navegador web: http://ip_servidor/event_correlator/index.php

Con el escenario planteado los ficheros *xml* son generados y almacenados en un servidor remoto, pero interpretados y tratados en un recurso local (servidor Linux), pero existen otras posibilidades.

A continuación presentamos otro escenario en el que mostramos los pasos necesarios para interactuar con la aplicación en un equipo Windows.

3.6.3 Puesta en marcha del programa en modo consola (WINDOWS)

Uno de los requisitos del sistema era la posibilidad de que este pudiera operar en múltiples entornos, es decir, establecer una aplicación independiente de plataforma. Con la solución proporcionada podemos poner en marcha el sistema en múltiples entornos, Windows, Linux, etc.

A continuación, describiremos el procedimiento para poner en marcha el sistema y recibir los eventos mediante, por ejemplo, el protocolo **FTP** sobre un equipo que tiene instalado un entorno Windows, concretamente Windows 7.

Para ello, tenemos que cumplir con dos requisitos. Por un lado, ubicar la aplicación Java desarrollada en un dispositivo con conectividad dentro de la LAN en que se ubican los dispositivos de comunicación y por otro, que este equipo disponga de un entorno de ejecución (máquina virtual) Java instalado. De esta manera cuando aparezcan los ficheros xml que representan los eventos, el sistema los procesará y emitirá las correspondientes alertas acústicas.

Para configurar la aplicación editaremos el fichero *config.cfg* que debe estar ubicado en el mismo directorio donde se encuentra la aplicación.

La primera línea de este fichero deberá contener el recurso a monitorizar (directorio), en el caso de que este directorio este localizado en un equipo **Windows** deberemos usar los caracteres “/”, para componer la ruta absoluta del directorio, ya que de otro modo el carácter “/” se interpreta de manera errónea, sin embargo en entornos *Linux* al estar las rutas compuestas de la siguiente manera: “/home/rvidal/”, esto no será necesario.

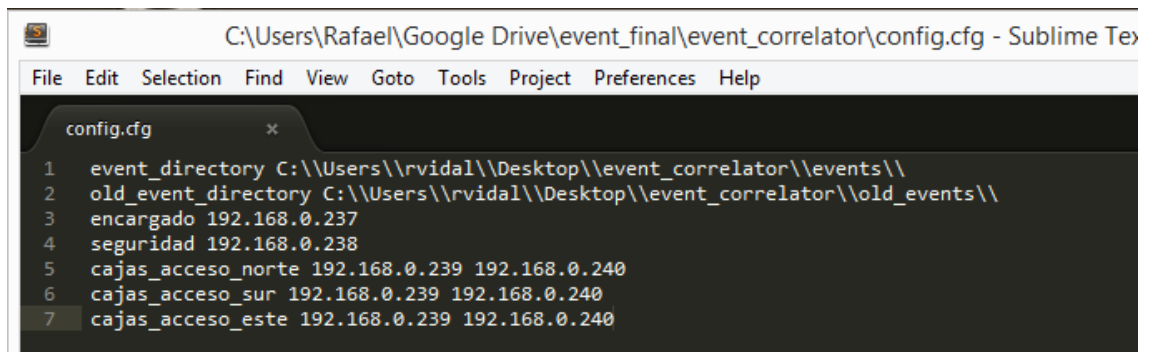
La segunda línea de este fichero deberá contener la ruta a otro directorio donde volcaremos los ficheros **xml** que representan los eventos de manera que no vuelvan a ser analizados y generen nuevas alertas acústicas.

Las siguientes líneas representan los grupos de difusión de las alertas, y podemos tener hasta 5.

Esta línea está compuesta por el nombre del grupo, que es simplemente orientativo, la única condición que debe cumplir es que no puede contener espacios seguido por las direcciones **IP** con las que fueron configurados los dispositivos.

Cuando la dirección **IP** de un dispositivo aparece en la línea de un grupo automáticamente este dispositivo estará suscrito a las alertas que se generen para dicho grupo. De esta forma si queremos eliminar un dispositivo de un grupo simplemente tenemos que editar este fichero para eliminar el dispositivo del grupo de difusión.

Ilustración 43: Fichero de configuración "config.cfg"

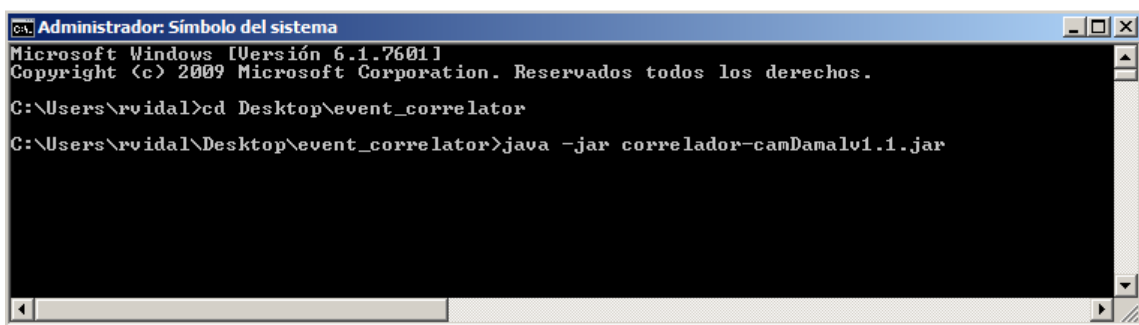


```
C:\Users\Rafael\Google Drive\event_final\event_correlator\config.cfg - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
config.cfg
1 event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\events\\
2 old_event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\old_events\\
3 encargado 192.168.0.237
4 seguridad 192.168.0.238
5 cajas_acceso_norte 192.168.0.239 192.168.0.240
6 cajas_acceso_sur 192.168.0.239 192.168.0.240
7 cajas_acceso_este 192.168.0.239 192.168.0.240
```

A continuación tendremos que ejecutar el programa **Java**, para ello simplemente tenemos que ejecutar el siguiente mandato:

```
java -jar correlador-camDamalv1.1.jar
```

Ilustración 44: Orden de ejecución del programa



```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\rvidal>cd Desktop\event_correlator
C:\Users\rvidal\Desktop\event_correlator>java -jar correlador-camDamalv1.1.jar
```

De inmediato el sistema se pondrá en marcha, el programa nos mostrará diferente información, los dispositivos detectados en el fichero de configuración, la ruta al directorio monitorizado, etc.

Y el sistema quedara a la espera de la llegada de eventos:

Ilustración 45: Consola mostrando la configuración de la aplicación

```
Administrador: Símbolo del sistema - java -jar correlador-camDama1v1.1.jar
C:\Users\rvidal\Desktop\event_correlator>java -jar correlador-camDama1v1.1.jar
PROCESADOR DE EVENTOS
-----
cargando configuración global ...
leyendo fichero: config.cfg
Creado dispositivo:Thread[Thread-0,5,main]
Creado dispositivo:Thread[Thread-1,5,main]
Creado dispositivo:Thread[Thread-2,5,main]
Creado dispositivo:Thread[Thread-3,5,main]
Dispositivos detectados:
Dispositivo 1: 192.168.0.237
Dispositivo 2: 192.168.0.239
Dispositivo 3: 192.168.0.238
Dispositivo 4: 192.168.0.240
Los eventos serán leídos del directorio: C:\Users\rvidal\Desktop\event_correlator\events\
Los eventos antiguos serán volcados al directorio: C:\Users\rvidal\Desktop\event_correlator\old_events\
IP's asociadas a grupo 1:
192.168.0.237
IP's asociadas a grupo 2:
192.168.0.238
IP's asociadas a grupo 3:
192.168.0.239
192.168.0.240
IP's asociadas a grupo 4:
192.168.0.239
192.168.0.240
IP's asociadas a grupo 5:
192.168.0.239
192.168.0.240
-----
Iniciando correlador de eventos...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
-
```

Ilustración 46: Esperando recepción de eventos

```
esperando eventos...
cola vacia :0
cola vacia :0
Ejecuto run de hilo dispositivo: 192.168.0.239
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.239
cola vacia :0
Ejecuto run de hilo dispositivo: 192.168.0.238
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.238
Ejecuto run de hilo dispositivo: 192.168.0.240
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.240
```

3.6.4 Uso mediante interfaz web.

El primer paso después del despliegue consistirá en la puesta en marcha del sistema. Por motivos de seguridad tanto la ruta donde llegarán los ficheros *xml* que representan los eventos como el directorio donde se vuelcan los eventos ya procesados no se pueden modificar directamente desde la interfaz web, de manera que quedarán tal y como especificamos en el punto anterior.

A partir de este punto podremos especificar los dispositivos asociados a los diferentes desde este mismo fichero o bien recurrir a la interfaz web, simplemente insertando la dirección IP del servidor y la ruta a la aplicación (*http://192.168.1.9/event_correlator/*).

En esta guía vamos a proceder de esta manera:

Primero introducimos la dirección IP con la que han sido configurados los dispositivos en los diferentes *textbox*, de manera que al aparecer una dirección en esta casilla automáticamente pertenecerá a dicho grupo de difusión. Esto se hace en la página principal de la aplicación o "*index.php*".

A continuación, cuando ya hemos terminado de editar los grupos podemos guardar los cambios haciendo click sobre el botón "*Guardar Cambios*".

Ilustración 47: Configuración de grupos de difusión

Configuración básica de Grupos de Recepción

Asigne la IP de los dispositivos a los diferentes grupos.

IP's Dispositivos GRUPO 1: (Gerencia)

192.168.0.239

IP's Dispositivos GRUPO 2: (Seguridad):

IP's Dispositivos GRUPO 3: (Cajas 1)

IP's Dispositivos GRUPO 4: (Cajas 2)

IP's Dispositivos GRUPO 5: (Cajas 3)

Grabar cambios

A continuación hacemos click sobre la opción "*Iniciar/Parar*" del menú de navegación. De esta manera accedemos a la página web que nos permite lanzar el proceso remoto que monitorizará el directorio especificado en busca de eventos que hagan *match* con las diferentes reglas, y se ejecutará la orden de reproducción de las diferentes alertas en los dispositivos.

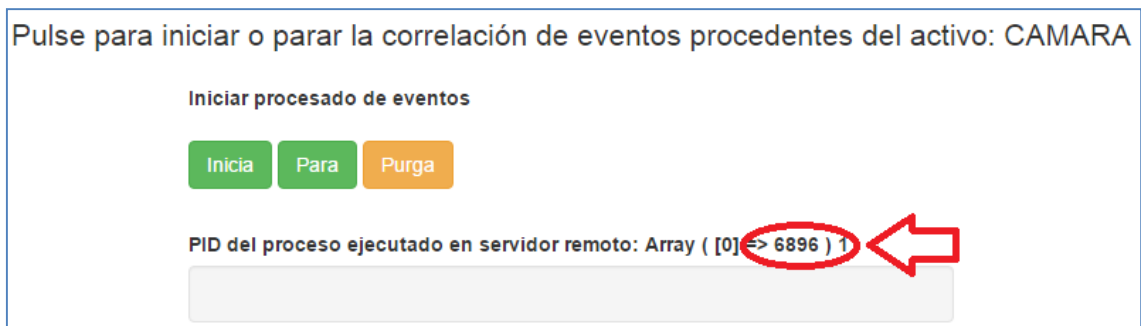
Un primer paso recomendable es realizar el borrado de los eventos antiguos que puedan existir en el directorio, de esta manera no enviamos alertas correspondientes a eventos antiguos que hayan podido llegar (3).

Ilustración 48: Inicio procesamiento de eventos a través de interfaz web



Como vemos al iniciar presionando sobre el botón "**Iniciar**", aparece el PID del proceso remoto que se está ejecutando:

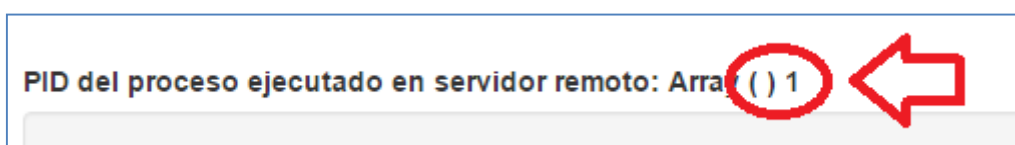
Ilustración 49: PID de proceso ejecutado en servidores remoto



Esto es una confirmación de que el proceso se está ejecutando correctamente.

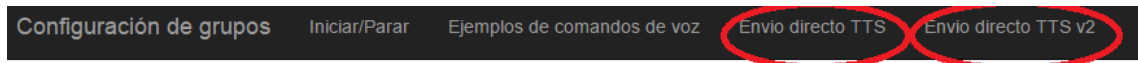
Si lo paramos, esta cadena desaparece. Cabe reseñar en este punto que la opción parar mata todos los procesos Java en el sistema, esta es una aproximación que como implantación piloto hemos dado por buena, ya que el servidor únicamente ejecuta esta tarea.

Ilustración 50: Estado interfaz después de parar el servicio de monitorización de eventos



Además, mediante la opción correspondiente del menú de navegación podemos enviar mensajes directamente a los dispositivos. Para esta funcionalidad no es necesario que el motor de correlación de eventos se encuentre en funcionamiento, ya que los mensajes se envían directamente al dispositivo.

Ilustración 51: Pestaña de envío directo de mensajes a dispositivo



Para ello, únicamente deberemos seleccionar el dispositivo, y bien teclear las palabras que queremos que se reproduzcan o bien seleccionar directamente la palabra que aparecerá en el cuadro de texto.

Ilustración 52: Introducción de órdenes de voz por teclado

Texto a voz:

Seleccione un dispositivo y introduzca el texto a reproducir.

Vocabulario reconocido:

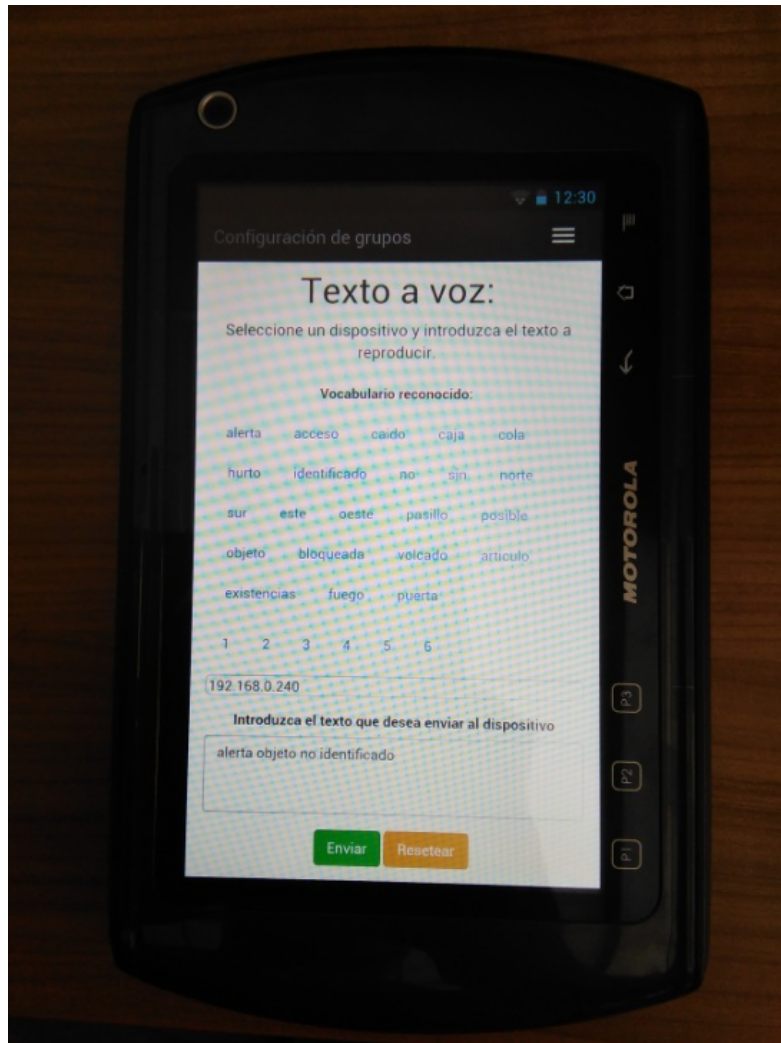
alerta acceso caído caja cola hurto identificado no sin en
norte sur este oeste pasillo posible objeto
bloqueada volcado articulo existencias fuego puerta
1 2 3 4 5 6

192.168.0.239 ▼

Introduzca el texto que desea enviar al dispositivo

alerta hurto

Ilustración 53: Terminal tablet industrial accediendo a aplicación

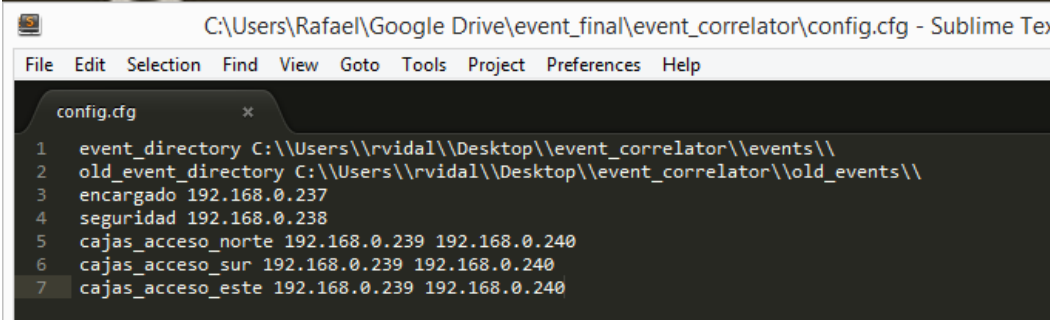


3.7 Resultados.

3.7.1 Ejemplos de ejecución (salida consola).

Comprobamos los patrones de configuración de la aplicación, para poder seguir los pasos de ejecución a partir de la llegada de diferentes eventos.

Ilustración 54: Contenido fichero de configuración "config.cfg":

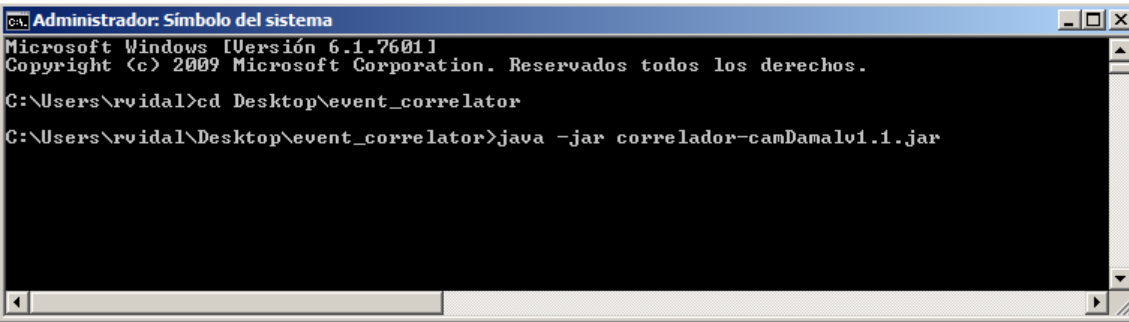


```
C:\Users\Rafael\Google Drive\event_final\event_correlator\config.cfg - Sublime Te
File Edit Selection Find View Goto Tools Project Preferences Help
config.cfg
1 event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\events\\
2 old_event_directory C:\\Users\\rvidal\\Desktop\\event_correlator\\old_events\\
3 encargado 192.168.0.237
4 seguridad 192.168.0.238
5 cajas_acceso_norte 192.168.0.239 192.168.0.240
6 cajas_acceso_sur 192.168.0.239 192.168.0.240
7 cajas_acceso_este 192.168.0.239 192.168.0.240
```

A continuación tendremos que ejecutar el programa **Java**, para ello simplemente tenemos que acudir al directorio donde está ubicado y ejecutarlo de la siguiente manera:

```
java -jar correlador-camDamalv1.1.jar
```

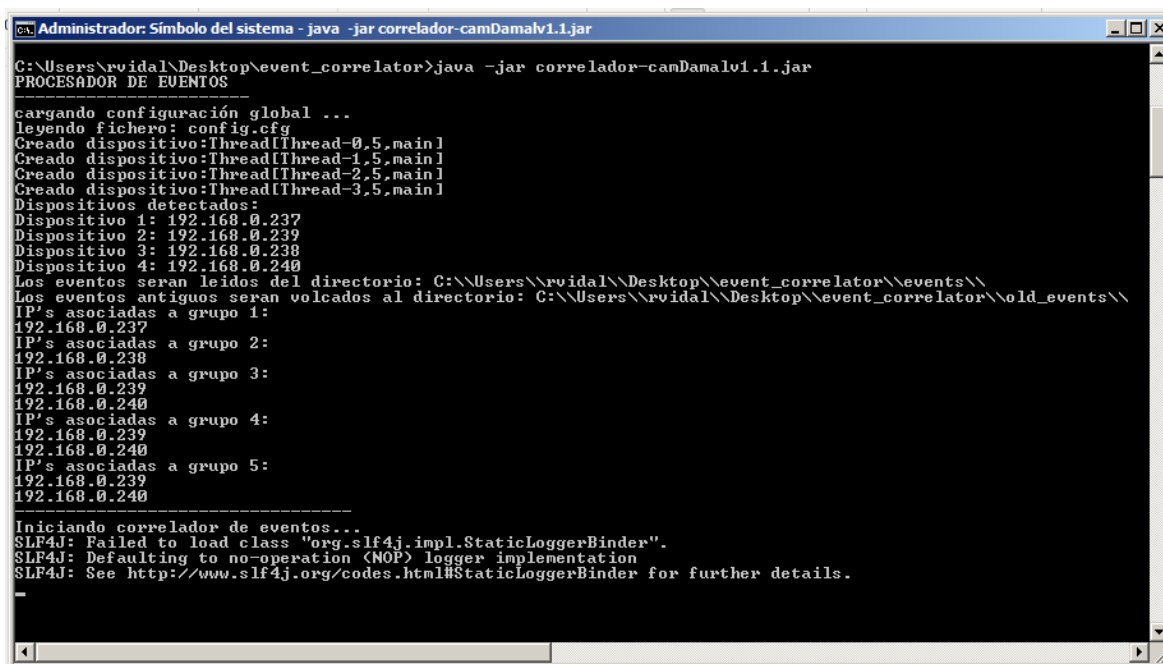
Ilustración 55: Orden de ejecución programa por consola



```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\rvidal>cd Desktop\event_correlator
C:\Users\rvidal\Desktop\event_correlator>java -jar correlador-camDamalv1.1.jar
```

De inmediato el sistema se pondrá en marcha, el programa nos mostrara diferente información, los dispositivos detectados en el fichero de configuración, la ruta al directorio monitorizado, etc.

Ilustración 56: Configuración aplicada a programa en ejecución

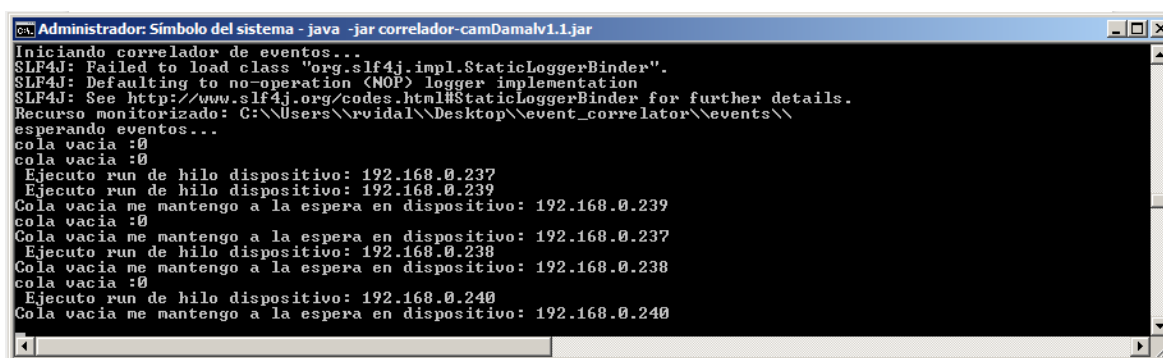


```
Administrador: Símbolo del sistema - java -jar correlador-camDamalv1.1.jar
C:\Users\rvidal\Desktop\event_correlator>java -jar correlador-camDamalv1.1.jar
PROCESADOR DE EUEMOTOS
-----
cargando configuración global ...
leyendo fichero: config.cfg
Creado dispositivo:Thread[Thread-0,5,main]
Creado dispositivo:Thread[Thread-1,5,main]
Creado dispositivo:Thread[Thread-2,5,main]
Creado dispositivo:Thread[Thread-3,5,main]
Dispositivos detectados:
Dispositivo 1: 192.168.0.237
Dispositivo 2: 192.168.0.239
Dispositivo 3: 192.168.0.238
Dispositivo 4: 192.168.0.240
Los eventos seran leidos del directorio: C:\Users\rvidal\Desktop\event_correlator\events\
Los eventos antiguos seran volcados al directorio: C:\Users\rvidal\Desktop\event_correlator\old_events\
IP's asociadas a grupo 1:
192.168.0.237
IP's asociadas a grupo 2:
192.168.0.238
IP's asociadas a grupo 3:
192.168.0.239
192.168.0.240
IP's asociadas a grupo 4:
192.168.0.239
192.168.0.240
IP's asociadas a grupo 5:
192.168.0.239
192.168.0.240
-----
Iniciando correlador de eventos...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

Adicionalmente dotamos al sistema de una gestión de colas sobre los dispositivos, de esta forma evitamos que exista colisión entre las diferentes alertas que se enviarán a los mismos.

Como vemos en la imagen el estado de las colas se muestra al iniciar la aplicación y de manera predeterminada se encuentran vacías.

Ilustración 57: Suspensión de hilos de ejecución en cola vacía

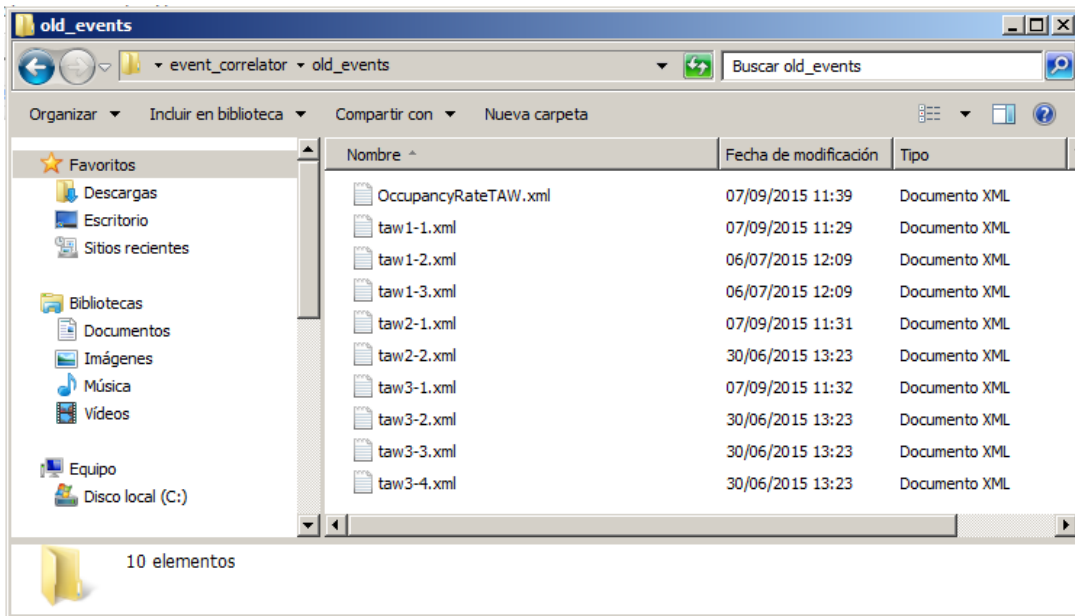


```
Administrador: Símbolo del sistema - java -jar correlador-camDamalv1.1.jar
Iniciando correlador de eventos...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Recurso monitorizado: C:\Users\rvidal\Desktop\event_correlator\events\
esperando eventos...
cola vacia :0
cola vacia :0
Ejecuto run de hilo dispositivo: 192.168.0.237
Ejecuto run de hilo dispositivo: 192.168.0.239
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.239
cola vacia :0
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.237
Ejecuto run de hilo dispositivo: 192.168.0.238
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.238
cola vacia :0
Ejecuto run de hilo dispositivo: 192.168.0.240
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.240
```

Como vemos nuestra aplicación maneja para cada dispositivo una cola de mensajes que únicamente es atendida mediante un hilo de ejecución asociado a cada dispositivo cuando se detecta la condición de que contiene al menos un mensaje en cola.

A continuación y con el objetivo de comprobar la funcionalidad del sistema podemos acudir al directorio donde se vuelcan los eventos procesados, copiar uno de ellos y volcarlo en el directorio monitorizado.

Ilustración 58: Contenido de directorio de eventos antiguos



De manera inmediata, este fichero será procesado por el correlador de eventos, generando las alertas acústicas, haciendo desaparecer el evento del directorio y mostrando por la consola las diferentes acciones iniciadas (encolado de mensaje, orden remota ejecutada en dispositivo, etc.)

En la siguiente imagen podemos comprobar cómo al recibirse un evento sobre los dispositivos **192.168.0.240** y **192.168.0.239** los hilos de ejecución de estos dispositivos han retomado la actividad para emitir la orden de ejecución.

Esta orden llega a estos dispositivos por que el evento tenía codificado el siguiente contenido:

Ilustración 59: Contenido fichero taw1-1.xml

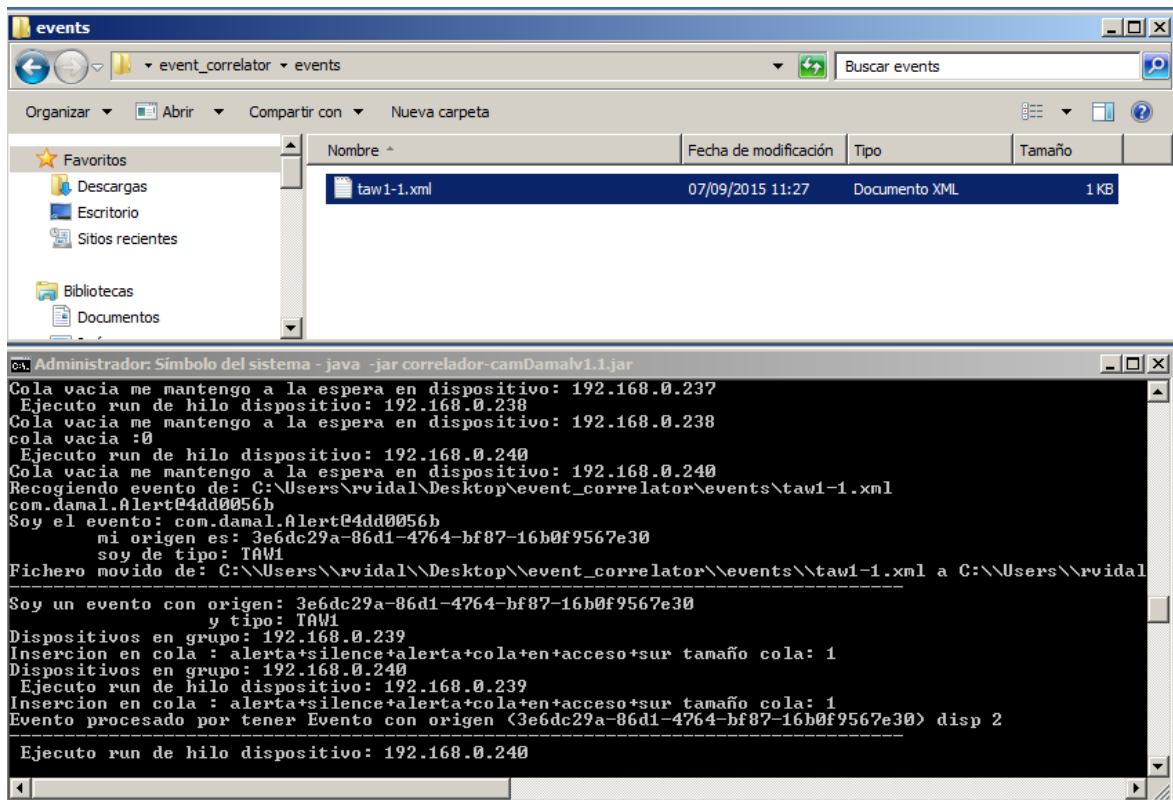
Node	Content
Alert	
Analytic	TAW1
Source	3e6dc29a-86d1-4764-bf87-16b0f9567e30

Que ha desencadenado la ejecución de la siguiente regla:

Ilustración 60: Regla de negocio para evento en cola sur

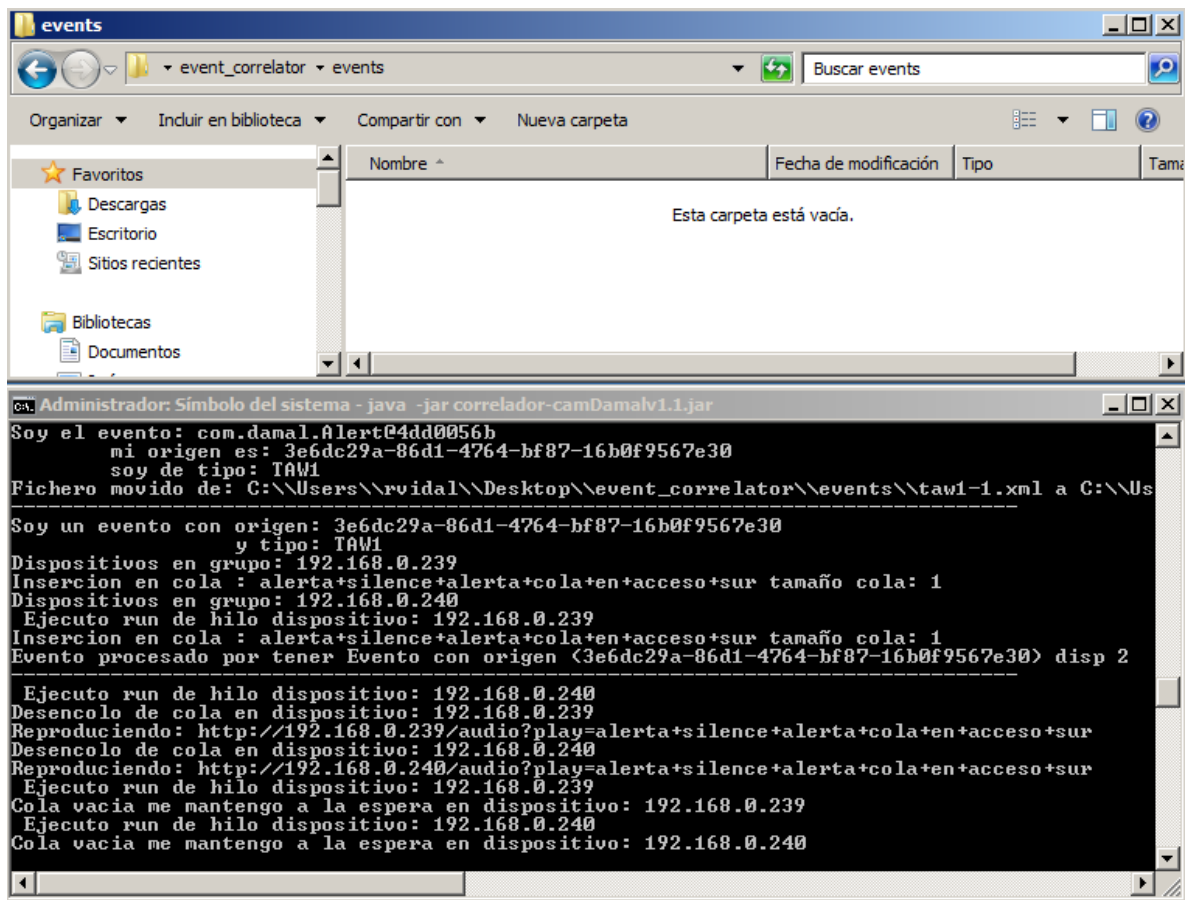
```
//notif cla acceso sur
rule "Evento con origen (3e6dc29a-86d1-4764-bf87-16b0f9567e30) de tipo cola acceso Sur"
  salience 8
  when
    m : Alert( Source == "3e6dc29a-86d1-4764-bf87-16b0f9567e30" && Analytic=="TAW1", miOrigen : Source, miTipo : analy
  then
    System.out.println("-----");
    System.out.println( "Soy un evento con origen: " + miOrigen );
    System.out.println( " y tipo: " + miTipo );
    //buscar dispositivos a notificar
    for(int i=0;i<Correlador.GRUPO_3.size();i++){
      System.out.println("Dispositivos en grupo: "+Correlador.GRUPO_3.get(i));
      for (int a=0;a<Correlador.devices.size();a++){
        if(Correlador.GRUPO_3.get(i).compareTo(Correlador.devices.get(a).getIdent())==0){
          Correlador.devices.get(a).insertEvent("alerta+silence+alerta+cola+en+acceso+sur");
          break;
        }
      }
    }
    System.out.println( "Evento procesado por tener Evento con origen (3e6dc29a-86d1-4764-bf87-16b0f9567e30) disp 2"
    System.out.println("-----");
end
```

Ilustración 61: Procesado de evento *taw1-1*



En la siguiente imagen podemos visualizar como a partir de este instante se han generado las órdenes de emisión de alerta y el fichero a desaparecido del directorio monitorizado.

Ilustración 62: Acciones desencadenadas por evento *taw1-1*



A continuación vamos a muestra algunas de las salidas del programa cuando recibe diferentes eventos:

En la siguiente imagen vemos como ha sido procesado el fichero *taw-2.1.xml* cuyo contenido es:

Ilustración 63: Contenido fichero *taw2-1.xml*

Node	Content
Alert	
Analytic	TAW2
Source	3e6dc29a-86d1-4764-bf87-16b0f9567e30

y cómo podemos comprobar encaja con la regla de negocio definida en el fichero *BussinesRules.drl*

Ilustración 64: Regla de negocio para posible hurto

```
rule "Evento con origen (3e6dc29a-86d1-4764-bf87-16b0f9567e30) de posible hurto"
salience 8
when
  m : Alert( Source == "3e6dc29a-86d1-4764-bf87-16b0f9567e30" && Analytic=="TAW2", miOrigen : S
then
  System.out.println("-----");
  System.out.println( "Soy un evento con origen: " + miOrigen );
  System.out.println( "                y tipo: " + miTipo );
  for(int i=0;i<Correlador.GRUPO_2.size();i++){
    System.out.println("Indice primer for: "+i);
    System.out.println("Dispositivo en grupo: "+Correlador.GRUPO_2.get(i));
    for (int a=0;a<Correlador.devices.size();a++){
      System.out.println("Indice segundo for: " +a);
      System.out.println("Dispositivo en device: "+Correlador.devices.get(a).getId());
      if(Correlador.GRUPO_2.get(i).compareTo(Correlador.devices.get(a).getId())==0){
        Correlador.devices.get(a).insertEvent("alerta+silence+alerta+posible+hurto+en+pasill
        break;
      }
    }
  }
  System.out.println( "Evento procesado por tener un origen en (3e6dc29a-86d1-4764-bf87-16b0f9
  System.out.println("-----");
end
```

Ilustración 65: Procesamiento fichero *taw2-1.xml*

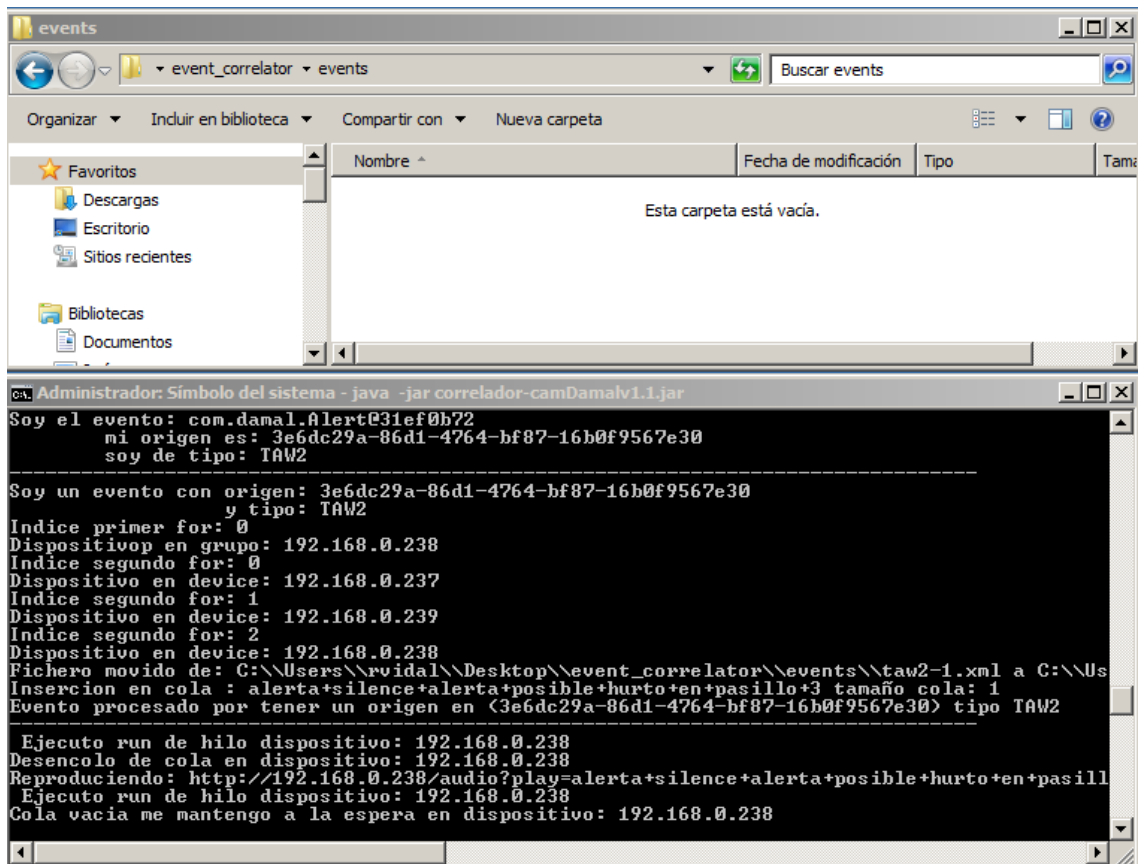


Ilustración 66: Salida procesamiento de evento *taw3-1*

Node	Content
Alert	
Analytic	TAW3
Source	3e6dc29a-86d1-4764-bf87-16b0f9567e30

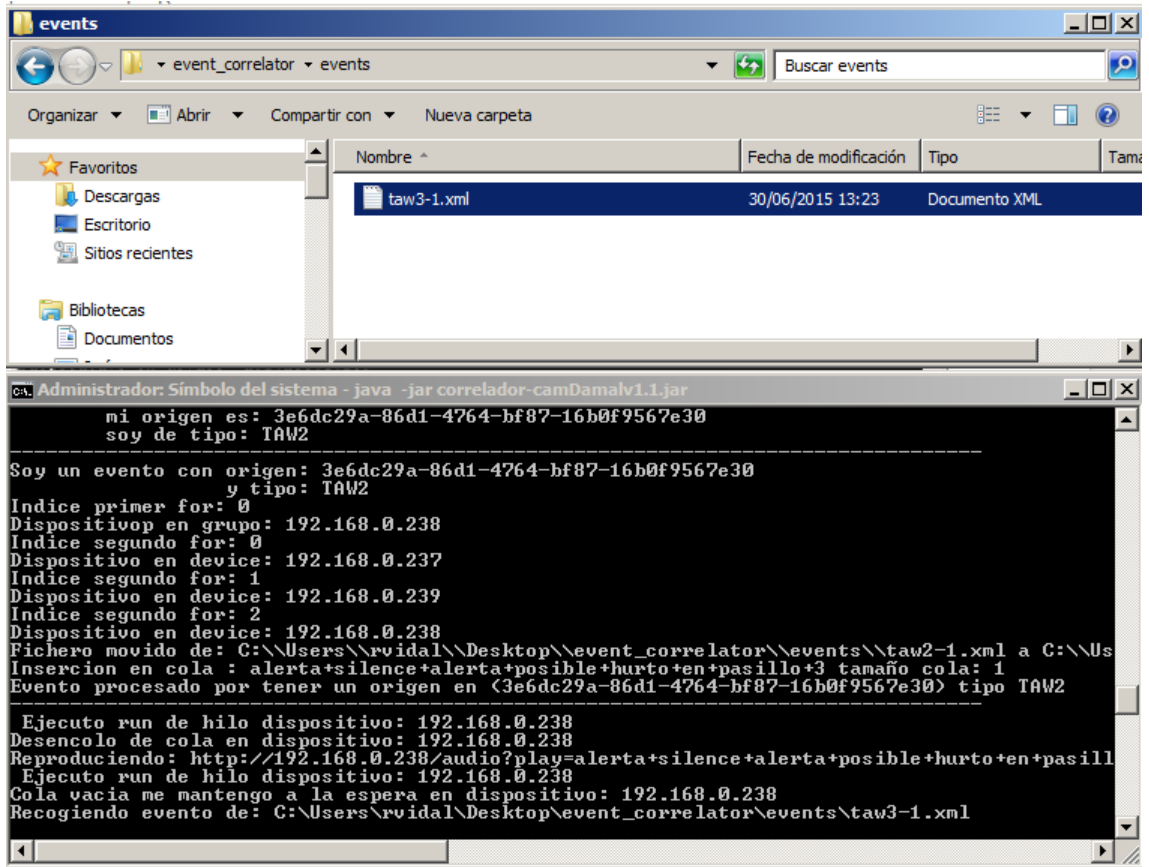


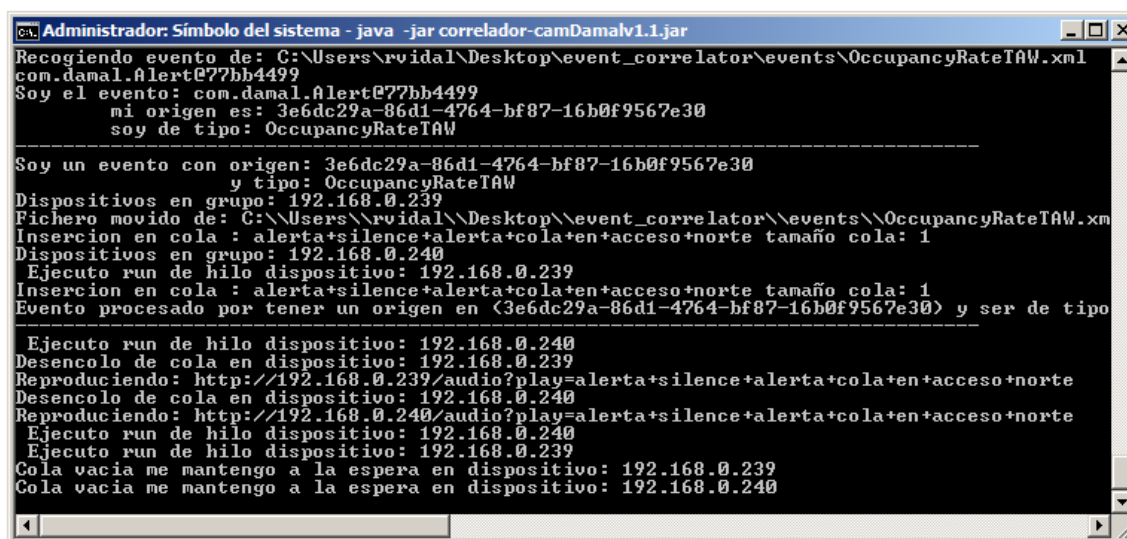
Ilustración 67: Regla de negocio para evento fuego en pasillo 5

```
rule "Evento con origen (3e6dc29a-86d1-4764-bf87-16b0f9567e30) de tipo TAW3"
  salience 8
  when
    m : Alert( Source == "3e6dc29a-86d1-4764-bf87-16b0f9567e30" && Analytic=="TAW3", miOrigen : Sourc
  then
    System.out.println("-----");
    System.out.println( "Soy un evento con origen: " + miOrigen );
    System.out.println( "                               y tipo: " + miTipo );
    //buscar dispositivos a notificar
    for(int i=0;i<Correlador.GRUPO_1.size();i++){
      System.out.println("Indice primer for: "+i);
      System.out.println("Dispositivo en grupo: "+Correlador.GRUPO_2.get(i));
      for (int a=0;a<Correlador.devices.size();a++){
        System.out.println("Indice segundo for: " +a);
        System.out.println("Dispositivo en device: "+Correlador.devices.get(a).getIdent());
        if(Correlador.GRUPO_1.get(i).compareTo(Correlador.devices.get(a).getIdent())==0){
          Correlador.devices.get(a).insertEvent("alerta+alerta+fuego+en+pasillo+5");
          break;
        }
      }
    }
    System.out.println( "Evento procesado por tener un origen en (3e6dc29a-86d1-4764-bf87-16b0f9567e
    System.out.println("-----");
  end
```

Ilustración 68: Salida procesamiento de evento cámara ("OccupancyRateTAW")

The image shows two windows. The top window is a Windows Explorer window titled 'events' showing a folder named 'event_correlator' containing a file 'OccupancyRateTAW.xml' with a modification date of 07/09/2015 11:37. The bottom window is a command prompt titled 'Administrador: Símbolo del sistema - java -jar correlador-camDamalv1.1.jar'. The output shows the start of an event correlator, listing IP addresses for group 5 (192.168.0.239 and 192.168.0.240), and then a series of log messages indicating the execution of threads for each device, including messages like 'cola vacia :0' and 'Ejecuto run de hilo dispositivo: 192.168.0.239'. The final line of output is 'Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\OccupancyRateTAW.xml'.

Ilustración 69: Envío señal de alerta de cola en caja



```
Administrador: Símbolo del sistema - java -jar correlador-camDamalv1.1.jar
Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\OccupancyRateTAW.xml
com.damal.Alert@77bb4499
Soy el evento: com.damal.Alert@77bb4499
mi origen es: 3e6dc29a-86d1-4764-bf87-16b0f9567e30
soy de tipo: OccupancyRateTAW
-----
Soy un evento con origen: 3e6dc29a-86d1-4764-bf87-16b0f9567e30
y tipo: OccupancyRateTAW
Dispositivos en grupo: 192.168.0.239
Fichero movido de: C:\Users\rvidal\Desktop\event_correlator\events\OccupancyRateTAW.xml
Insercion en cola : alerta+silence+alerta+cola+en+acceso+norte tamaño cola: 1
Dispositivos en grupo: 192.168.0.240
Ejecuto run de hilo dispositivo: 192.168.0.239
Insercion en cola : alerta+silence+alerta+cola+en+acceso+norte tamaño cola: 1
Evento procesado por tener un origen en <3e6dc29a-86d1-4764-bf87-16b0f9567e30> y ser de tipo
-----
Ejecuto run de hilo dispositivo: 192.168.0.240
Desencolo de cola en dispositivo: 192.168.0.239
Reproduciendo: http://192.168.0.239/audio?play=alerta+silence+alerta+cola+en+acceso+norte
Desencolo de cola en dispositivo: 192.168.0.240
Reproduciendo: http://192.168.0.240/audio?play=alerta+silence+alerta+cola+en+acceso+norte
Ejecuto run de hilo dispositivo: 192.168.0.240
Ejecuto run de hilo dispositivo: 192.168.0.239
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.239
Cola vacia me mantengo a la espera en dispositivo: 192.168.0.240
```

Una de las características más interesantes de nuestra aplicación es la capacidad de procesar múltiples eventos y gestiona las colas e hilos de ejecución de los dispositivos en tiempo real. De manera que si aparecen diferentes eventos que hacen *match* en múltiples reglas las alertas a emitir serán almacenadas en el motor de procesamiento y reproducidas de manera secuencial.

En la siguiente imagen podemos comprobar el funcionamiento de la aplicación cuando tiene que atender múltiples eventos de manera simultánea.

Ilustración 70: Recepción simultánea de de múltiples eventos

The image shows a Windows File Explorer window titled 'events' with the address bar set to 'event_correlator > events'. The window contains a table of files:

Nombre	Fecha de modificación	Tipo
taw3-1.xml	07/09/2015 11:32	Documento XML
taw3-2.xml	30/06/2015 13:23	Documento XML
taw3-3.xml	30/06/2015 13:23	Documento XML
taw3-4.xml	30/06/2015 13:23	Documento XML

Below the table, the selected file 'taw3-2.xml' is shown with details: 'Documento XML', 'Fecha de modificación: 30/06/2015 13:23', 'Fecha de creación: 07/09/2015 11:42', and 'Tamaño: 101 bytes'.

The command prompt window below shows the execution of the Java application 'correlador-camDamalv1.1.jar'. The output includes the following text:

```
soy de tipo: TAW1
Fichero movido de: C:\Users\rvidal\Desktop\event_correlator\events\taw1-3.xml a C:\
Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\taw2-1.xml
com.damal.Alert@4d184d44
Soy el evento: com.damal.Alert@4d184d44
mi origen es: 3e6dc29a-86d1-4764-bf87-16b0f9567e30
soy de tipo: TAW2
Fichero movido de: C:\Users\rvidal\Desktop\event_correlator\events\taw2-1.xml a C:\
Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\taw2-2.xml
Desencolo de cola en dispositivo: 192.168.0.239
Reproduciendo: http://192.168.0.239/audio?play=alerta+silence+alerta+cola+en+acceso+sur
Desencolo de cola en dispositivo: 192.168.0.240
Reproduciendo: http://192.168.0.240/audio?play=alerta+silence+alerta+cola+en+acceso+sur
com.damal.Alert@87f3e2f
Soy el evento: com.damal.Alert@87f3e2f
mi origen es: 3e6dc29a-86d1-4764-bf87-16b0f9567e30
soy de tipo: TAW2
Fichero movido de: C:\Users\rvidal\Desktop\event_correlator\events\taw2-2.xml a C:\
Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\taw3-1.xml
com.damal.Alert@3d8dbd3b
Soy el evento: com.damal.Alert@3d8dbd3b
mi origen es: 3e6dc29a-86d1-4764-bf87-16b0f9567e30
soy de tipo: TAW3
Fichero movido de: C:\Users\rvidal\Desktop\event_correlator\events\taw3-1.xml a C:\
Recogiendo evento de: C:\Users\rvidal\Desktop\event_correlator\events\taw3-2.xml
```

3.7.2 Videos.

Para visualizar los videos con el sistema en funcionamiento, por favor acuda a la carpeta “*Videos*” disponible en el CD en el que se hace entrega la memoria.

4 Conclusiones y posibles ampliaciones al proyecto.

4.1 Conclusiones

Como hemos visto a lo largo de este documento podemos utilizar los sistemas de gestión de reglas de negocio para facilitar las tareas de reprogramación de aplicaciones. Mediante la gestión de las reglas podemos modificar la lógica de negocio de la aplicación, de manera que podemos sin tener que recompilar o realizar un nuevo despliegue modificar el comportamiento de la aplicación ante el registro de nuevos eventos o bien modificar el comportamiento que se desencadena ante la recepción de los ya existentes.

La plataforma **Drools** presenta diferentes opciones para el despliegue de un **BRMS**. Podríamos haber utilizado la interfaz web para el despliegue y gestión de todas las reglas de negocio, pero también podemos aprovechar únicamente el motor de reglas para realizar una integración con una aplicación propia. Éste es el enfoque que hemos seguido para la elaboración de este proyecto.

No obstante, esto es así porque de momento no consideramos que la lógica de negocio sea modificada por parte de personal no técnico. Lo que pretendíamos al utilizar el motor de reglas era por un lado, obtener un mecanismo de rápida y eficiente evaluación de condiciones, lo que conseguimos aprovechando la implementación del algoritmo de **Rete** que hace **Drools**, y por otro, lado permitir la reprogramación del comportamiento de la aplicación sin tener que recompilar el código, lo cual conseguimos mediante los cambios que realizamos sobre las acciones que desencadena la aplicación de una regla.

Además, en el caso que hemos presentado utilizamos como mensajes de intercambio ficheros **xml**, pero podemos imaginar escenarios más complejos que requieran de mecanismo más avanzados de intercambio de mensajes. En este sentido sobre la base del proyecto hemos está probando diferentes implementaciones de intercambio de ficheros y mensajes, como **XML-RCP** o **MQTT** (*Message Queue Telemetry Transport*). De esta forma, podemos llegar a implementar un **ESB** o **Enterprise Service Bus** que permita realizar el proceso de mediación entre la aplicación con otros desarrollos dentro de las aplicaciones empresariales o desarrollos externos, mediante **SOAP, RNI, REST, JNI**.

Esto permite aumentar la capacidad de nuestra aplicación y aumenta la flexibilidad y escalabilidad del sistema. Y dispondremos de mayor número de mecanismos para capturar e interpretar los eventos que llegan al sistema para realizar una correlación de los mismos, es decir evaluarlos y determinar que comportamiento desencadenarán.

Con el desarrollo planteado ya disponemos de un sistema de lo más versátil, que permite realizar tareas de notificación de manera no solo acústica sino además vocal, de forma que se puede notificar a los operarios de manera inmediata diferentes eventos que van teniendo lugar.

Como hemos visto, el proyecto se englobaba dentro de un sistema que permitía realizar la notificación de condiciones sobre un contexto en tiempo real. Para ello hemos tenido que integrar el sistema evaluador de reglas con un módulo capaz de enviar

peticiones de reproducción de diferentes palabras sobre unos determinados dispositivos hardware como son los *walkie talkies* EWB100 de Motorola.

En principio hemos realizado un despliegue piloto de este sistema en el contexto de la notificación de eventos que se pueden dar sobre una tienda de alimentación, que son capturados por una cámara inteligente que funciona mediante visión artificial.

Sin embargo el sistema podría ser adaptado y utilizado para otras finalidades de manera muy ágil. Para ello deberíamos readaptar los objetos que dan soporte a los eventos y las reglas que podríamos requerir que desencadenarían un mensaje concreto.

A continuación presentamos algunos de los contextos en los que podría ser de utilidad un sistema con cómo el que hemos desarrollado, así como la posible evolución del mismo e integración sobre diferentes soportes y aplicaciones.

4.2 Ampliaciones al proyecto

El utilizar un sistema de gestión de reglas de negocio y motor de razonamiento como **Drools** nos permite realizar una gestión eficaz de los eventos que van apareciendo. El análisis de los mismos se realiza con gran rapidez y además podemos crear nuevas reglas y modificar las existentes sin tener la necesidad de recompilar todo el proyecto cada vez que aplicamos un cambio. No obstante, en aras de simplificar más la gestión de los eventos consideramos que un punto importante de cara al futuro del proyecto será el utilizar los ficheros de configuración *Excel*. De manera que resulte más sencillo para el personal no técnico realizar actualizaciones sobre las reglas.

A la vez que desarrollamos el sistema de notificación de eventos en tiempo real buscábamos nuevas aplicaciones sobre las que desplegar el sistema. Para ello tenemos que conseguir que el sistema desarrollado sea compatible con diferentes aplicaciones. Como vemos, de momento los eventos que capturamos son de un tipo muy específico. Nuestra aplicación identifica la cámara que detectaba las condiciones físicas, y el tipo de incidencia que se da (*Source* y *Analytic*):

```
<Alert>
  <Analytic>TAW1</Analytic>
  <Source>3e6dc29a-86d1-4764-bf87-16b0f9567e30</Source>
</Alert>
```

Sin embargo, resultaría sencillo identificar otras condiciones que resulte interesante notificar a los operarios. Siguiendo con el ejemplo de una tienda de productos de alimentación podemos imaginar otra información a reportar a los operarios, la recepción de correos electrónicos que puedan ser críticos cuando no se encuentran trabajando sobre un terminal, avisar de un camión esperando en la zona de descarga, etc.

Pero desde el punto vista del desarrollo de producto, resulta especialmente interesante la integración de nuestro sistema sobre sistema de seguridad. Por ejemplo, podemos imaginar que el sistema que hemos desarrollado podría notificar al personal de seguridad la presencia de un objeto sospechoso o de una persona en un espacio donde no está permitido el acceso de personas a partir de una hora determinada. Dicha integración sería interesante por ejemplo en un hotel, zona portuaria, etc. De esta manera un sistema de cámaras u otros sensores podría actuar como los ojos, y nuestro

sistema sería el cerebro capaz de procesar las condiciones y voz que notifica al operario. De esta manera nuestra aplicación constituiría una ampliación a los sentidos del usuario.

Otra aplicación también de interés es el aprovechar la capacidad de detectar condiciones sobre dispositivos para su reporte al personal técnico de una industria. Aprovechando el auge de los dispositivos que se comunicarán entre sí (*IoT, internet of things*), podemos imaginar, por ejemplo, en una industria de sistemas de mecanizado o estampado en la que existe diferente maquinaria que realiza diferentes funciones. Estas máquinas-herramienta pueden sufrir diferentes incidencias y resultaría interesante disponer de un sistema centralizado para el reporte de las mismas. De esta manera podremos discriminar los eventos y notificarlos de una forma dirigida según las reglas que establezcamos. Estas incidencias podrían ser, por ejemplo una impresora que se ha quedado sin papel o una herramienta de mecanizado con un cabezal obstruido, etc.

En este sentido, de cara a desarrollos futuros necesitamos dejar de depender de un sistema externo de sensorización como son las cámaras descritas en apartados anteriores y buscar una solución propia a la problemática de detección de las condiciones que disparan los eventos. Esto lo podríamos conseguir, por ejemplo, con cualquier cámara que opere mediante IP y mediante la biblioteca **OpenCV**, ya que el problema de detección de personas es recurrente y existe gran número de algoritmos que incorporan esta funcionalidad. De esta forma podremos extender las funcionalidades del sistema.

Así como las cámaras que operan mediante visión artificial también resultaría de interés encontrar otros sensores que sean capaces de generar señales que nuestra aplicación pueda capturar, procesar y notificar, por ejemplo sensores de presencia, detectores de humo, de inundaciones, etc. Es en sentido hacia donde el proyecto está evolucionando.

5 Presupuesto.

1. Coste de los materiales.

Descripción	Unidades	Parcial	Total
Equipo servidor.	1	300 €	300 €
Dispositivos Motorola EWB100	4	200 €	800€
Cuna de carga dispositivos Motorola EWB100	1	240 €	240€

2. Coste de la mano de obra.

Descripción	Horas	Parcial	Total
Especialista analista diseñador de sistemas informáticos.	70	14 €	980 €
Programador informático.	90	10 €	900 €
Informático depurador de programas.	30	10 €	300 €

3. Coste total del proyecto.

Coste materiales + coste mano de obra = 1340 + 2180 = **3520 €**

6 Pliego de condiciones.

6.1 Descripción del proyecto.

En este proyecto se pretende desarrollar un sistema de notificación de eventos mediante alertas acústicas. Para ello se utilizará un hardware específico (*walkie-talkie* EWB100), que presenta la utilidad de reproducir mensajes codificados de manera textual (sistemas *TTS* o *Text To Speech*).

El desarrollo planteado permitirá la configuración y modificación de los mensajes a reproducir y los eventos que los desencadenan (condiciones) de manera dinámica. Así como permitir la integración del mismo bajo cualquier sistema de monitorización mediante sensores.

Según las especificaciones de diseño, el programa debe reunir las siguientes cualidades:

- Multiplataforma (entornos Windows, Linux, MAC OS-X, etc.).
- Reproducción de las alertas en tiempo real (en el momento en que se dan las condiciones que disparan la creación del evento).
- Disponer de un entorno visual amistoso y agradable.
- Facilitar la tarea de puesta en marcha del sistema y reconfiguración del mismo.
- Disponer de una entrada de datos sencilla y flexible.

6.2 Condiciones generales.

6.2.1 Condiciones generales facultativas.

i. Promotor del proyecto.

El promotor del presente proyecto tiene las facultades que se le asignan en las leyes bajo la reglamentación vigente durante la realización del mismo.

Asimismo, dispondrá de la facultad para cambiar las especificaciones del sistema, siempre y cuando dicha modificación no suponga un perjuicio claro para el proyectista al alterar parte del trabajo ya realizado.

Cualquier modificación se recomienda que se efectúe previa consulta con el proyectista y cualquier cambio se aplicará preferentemente de manera consensuada.

Asimismo, se le faculta para decidir sobre los plazos de entrega, en caso de una demora excesiva y no razonada en la ejecución del proyecto.

ii. Obligaciones y derechos del proyectista encargado del proyecto.

El proyectista tiene las siguientes obligaciones:

- Cumplir con la legislación vigente.
- Respetar las leyes de derechos de autor.
- Realizar el proyecto según las indicaciones del promotor.
- Consultar con el promotor del proyecto cualquier modificación de las especificaciones iniciales, así como proponer soluciones alternativas que puedan resolver los problemas que se planteen.

- Informar periódicamente al promotor del estado en que se encuentra el proyecto.

Asimismo, tiene los siguientes derechos:

- Ser informado por parte del promotor de los derechos legales sobre el proyecto.
- Recibir solución a los problemas técnicos no previstos que aparezcan durante la ejecución del proyecto y no imputables a una mala ejecución del mismo.
- Disponer con la suficiente antelación de una lista con las especificaciones del programa.
- Disponer de los diferentes ficheros que constituirán los eventos y la información sobre los mismos que le permitan realizar una interpretación de los mismos para la adaptación de la aplicación.
- En caso de ausencia del promotor del proyecto, ante cualquier imprevisto u objeción que apareciese, el proyectista responsable tiene plenas potestades para tomar una decisión, que deberá ser asumida por el promotor.

iii. Facultades del promotor del proyecto.

El promotor del presente proyecto tiene las facultades que se le asignan en las leyes bajo la reglamentación vigente durante la realización del proyecto.

Asimismo dispondrá de la facultad para cambiar las especificaciones del sistema, siempre y cuando dicha modificación no suponga un perjuicio claro para el proyectista, al alterar parte del trabajo ya realizado.

Cualquier modificación se recomienda que se efectúe previa consulta con el proyectista y cualquier cambio se aplicara preferentemente de manera consensuada.

Asimismo, se le faculta para decidir sobre los plazos de entrega en caso de una demora excesiva y no razonada en la ejecución del proyecto.

.

iv. Comienzo, ritmo, plazo y condiciones generales de la ejecución del objetivo del proyecto.

El comienzo del proyecto será indicado por el promotor del proyecto.

El ritmo de trabajo será el disponible por parte del proyectista dentro de sus funciones dentro de la empresa, trabajo que realizará mientras atiende otras funciones de desarrollo y oficina técnica, siempre de forma razonada y justificada ante el promotor.

De igual manera, el plazo de entrega y condiciones generales se establecerá por mutuo acuerdo entre el proyectista y el promotor, y el ciclo de vida del producto desarrollado se adaptara a su viabilidad económico-comercial y de despliegue.

.

v. Recepción provisional del programa.

El promotor del proyecto recibirá una copia del programa y manual de uso con la suficiente antelación para poder revisar el mismo, poner a prueba y confirmar su correcto funcionamiento.

vi. Periodo de prueba.

Se establece como periodo de prueba el periodo de tiempo entre la entrega preliminar y la fecha en la que se cierra el plazo para presentar alegaciones al proyecto.

Durante este periodo el proyectista tendrá la obligación de subsanar cualquier error que se observe por parte del proyecto.

6.2.2 Condiciones generales legales.

i. Reconocimiento de marcas registradas.

El autor del proyecto y del siguiente documento reconoce los derechos de autor de la bibliografía que ha sido consultada y para que conste la aporta en el apartado correspondiente.

Asimismo también reconoce los derechos de autor sobre las imágenes , fotografías, catálogos de productos, logotipos de terceras firmas etc. utilizadas para la elaboración de la memoria del proyecto.

ii. Derechos de autor.

Los derechos de autor del siguiente proyecto son los estipulados por las leyes y la reglamentación vigente en el momento de comienzo del proyecto, salvo posibles correcciones legales como resultado de los recursos legales que se interpongan contra las citadas leyes y reglamentaciones.

iii. Causas de rescisión del proyecto.

El promotor del proyecto puede rescindir el contrato con el proyectista cuando se den las siguientes causas:

- Por un retraso excesivo en la ejecución del proyecto.
- Por un abandono del proyecto sin causa justificada.
- Por causas administrativas.
- Por mutuo acuerdo entre las partes, siempre y cuando ninguna de ellas se considere gravemente perjudicada.

6.3 Condiciones Particulares.

Al tratarse de un proyecto software, en el mismo se derivan una serie de condiciones que limitan de alguna forma la utilización del mismo. Estas condiciones las detallamos a continuación.

i. Condiciones software.

Para poder desplegar y ejecutar el software se deberá proceder tal y como se indica en el manual de usuario.

A pesar de ser concebido como un proyecto multiplataforma, para un despliegue preliminar se recomienda emular los procedimientos de despliegue y ejecución que se siguen en dicho manual.

ii. Condiciones Hardware.

Los requisitos hardware son los siguientes:

Servidor de la aplicación

Cualquier equipo doméstico, PC, sobremesa, preferentemente y de cara a asegurar la fiabilidad del sistema se recomienda hacer uso de un servidor. Con una instalación de servidor web (Apache) y con JAVA v. 1.7.

- CPU: en general cualquier modelo capaz de ejecutar una maquina virtual Java es suficiente.

- RAM: todo el código se ha probado en un ordenador con 1 GB de memoria RAM, por lo que consideramos que con esta cantidad es suficiente.

- Disco duro con el tamaño suficiente para realizar una instalación de una distribución de Linux más el servidor Apache y con espacio en disco para guardar de manera provisional los eventos que se generan.

Equipo cliente

Cualquier dispositivo con capacidad para conectarse mediante http al equipo servidor.

Hardware adicional

- Antenas Wi-Fi (red inalámbrica).

- *Walkie-Talkie's* MOTOROLA EWB100.

iii. Formación del usuario.

El usuario deberá tener los conocimientos adecuados para interactuar con una aplicación web y seguir las pautas y procedimientos desarrollados en el manual que se distribuye conjuntamente con la aplicación.

Bibliografía

- Badaró, S.; Ibañez, L.J. y Agüero, M.J. (2013): *Sistemas Expertos: Fundamentos, Metodologías y Aplicaciones*. México, Universidad de Palermo. Nº13 , pp. 349-364. Extraído de:
http://www.palermo.edu/ingenieria/pdf2014/13/CyT_13_24.pdf
Recuperado el 02, 11 y 12 de agosto de 2015.
- Carlos Soto, M (2011): *Teoría de Sistemas Expertos*. Perú, Sistema experto de diagnostico médico del Síndrome Guillem de Bare, Universidad Nacional de Mayor de San Marcos. Extraído de:
http://sisbib.unmsm.edu.pe/bibvirtualdata/tesis/basic/carlos_sm/cap1.pdf
Recuperado el 12,13 y 16 de junio y 23,24 y 25 de Agosto de 2015.
- Castillo, E.; Gutiérrez, J.M.; Hadi, A.S. (2012): *Sistemas Expertos y Modelos de Redes Probabilísticas*. Universidad de Cantabria. Editorial SEPA. Extraído de:
<http://personales.unican.es/gutierjm/papers/BookCGH.pdf> Recuperado el 12 y 14 de junio, el 26 y 27 de julio y 02 de agosto de 2015.
- Friedman-Hill, E. (2003): *Jess in Action: Java Rule-Based Systems*. England, Manning Publications.
- Forgy, L.C. (1982): *Rete: A fast algorithm for the many pattern/many object pattern match problem*. Artificial Intelligence, Vol. Nº19, Issue Nº11, pp. 17-37.
- Giarratano J., Riley G. D. (2015): *Expert Systems: Principles and Programming*. England, Fourth Edition.
- Jackson P. (1997): *Introduction to Expert Systems*. England, Allison Westley.
- León Quintanar, T. (2007): *Sistemas Expertos y sus aplicaciones*. Tesis Monografía. México, Universidad Autonoma del estado de Hidalgo , Pachuca. Extraído de :
<http://www.uaeh.edu.mx/docencia/Tesis/icbi/licenciatura/documentos/Sis>

[temas%20expertos%20y%20sus%20aplicaciones.pdf](#) Recuperado el 03,04 y 06 de agosto de 2015.

- López Toledo, D. (2012): *Creación de Sistemas Expertos*. México, Universidad Politécnica de Tapachula. Catarina. Extraído de: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/moreno_a_jl/capitulo3.pdf Recuperado el 13 y 14 de mayo y el 23 y 24 de julio de 2015.
- Pignani, J.M. (2013): *Sistemas Expertos Sistemas Expertos (Expert System)*. México, Universidad Tecnológica Nacional, Facultad Regional Rosario. Extraído de: http://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_ano/orientador_a1/monografias/pignani-sistemasexpertos.pdf Recuperado el 25 y 26 de mayo, 12, 14 y 24 de junio y 03 y 08 de julio de 2015.

WEBGRAFÍA:

- <https://sites.google.com/a/ci2s.com.ar/wiki/ci/expert-systems/a-brief-introduction-about-expert-system-rules-in-drools> Consultado el 04,05,06,08 y 14 de mayo de 2015.
- <https://sites.google.com/a/ci2s.com.ar/wiki/ci> Consultado el 10 y 12 de abril, el 12,14 y 15 de mayo y el 16 de julio de 2015.
- <http://java.sys-con.com/node/45082> Consultado el 12,13,14 y 15 de mayo y el 15 y 16 de junio de 2015.
- <http://tratandodeentenderlo.blogspot.com.es/2010/04/jboss-drools.html> Consultado el 26 y 28 de junio de 2015.
- https://en.wikipedia.org/wiki/Business_rules Consultado el 22 y 23 de junio y 02,03 de julio de 2015.
- <http://www.ibm.com/developerworks/java/library/j-drools/> Consultado el 23,25 y 14,15 y 16 de junio de 2015.
- <http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html> Consultado el 10,12 y 14 de junio de 2015.
- <http://www.drools.org/> Consultado el 03,04 y 18 de junio de 2015.

ANEXO 1:

REPRESENTACIÓN DEL CONOCIMIENTO

(Fuente: <http://www.plugtree.com/theoretical-background-for-drools-%E2%80%93-expert-systems-%E2%80%93-part-2/>)

Existen múltiples formas de representar el conocimiento, pero una de las más comunes es la representación mediante reglas de producción.

Introducción

El estudio del conocimiento también se conoce como epistemología. Hay dos tipos especiales de conocimiento, el conocido como conocimiento "*a priori*" y el conocimiento "*a posteriori*".

El primero viene antes y es independiente de los sentidos, por ejemplo: *todo lo que tiene un principio tiene un final*. El conocimiento "*a priori*" se considera como una verdad universal y no se puede negar sin una contradicción. Las sentencias lógicas y leyes matemáticas son ejemplos de conocimiento "*a priori*".

El otro tipo de conocimiento es el conocimiento "*a posteriori*", la veracidad o falsedad de este conocimiento se puede verificar a través de los sentidos, como por ejemplo la sentencia "*está lloviendo*". Sin embargo, como su veracidad depende de los sentidos, no siempre resulta confiable. Este conocimiento puede ser refutado a través de la adquisición de nuevo conocimiento sin necesidad de encontrar contradicciones.

Clasificación del conocimiento

Conocimiento Procedural: consiste en saber cómo desarrollar una tarea o procedimiento, por ejemplo, levantar una pared de ladrillos.

Conocimiento Declarativo: consiste en conocer la veracidad o no de algo, por ejemplo, los ladrillos no se unen mediante pegamento.

Conocimiento Tácito: a veces llamado conocimiento inconsciente porque no se puede expresar haciendo uso del lenguaje (por ejemplo, cómo mover una parte del cuerpo).

Jerarquía del Conocimiento

La adquisición de conocimiento forma parte de una jerarquía, tal y como se representa en la siguiente pirámide:



Ruido: cosas que no resultan de interés

Datos: cosas con interés potencial

Información: datos procesados y que resultan de interés

Conocimiento: información altamente especializada

Meta-Conocimiento: es conocimiento acerca del conocimiento y "*expertise*".

Como se puede comprobar en la definición de meta-conocimiento aparece el término "*expertise*".

La "*expertise*" es el conocimiento altamente especializado que poseen los expertos y que no se encuentra en libros o artículos científicos. La "*expertise*" es un conocimiento implícito del experto y debe ser extraído preguntando al mismo, de manera que pueda ser codificado e insertado en un sistema experto.

Producciones

Pero además, existe otro tipo de técnicas de representación del conocimiento: reglas, redes semánticas, grafos conceptuales, lenguajes de representación de conocimiento, etc.

Normalmente una forma de definir producciones es la notación Backus-Naur Form o Backus Normal Form (BNF). Esta notación representa un metalenguaje para definir la sintaxis del lenguaje.

En un ejemplo:

Imaginemos que somos los propietarios de un restaurante italiano y hemos contratado recientemente un experto cocinero. Cuando éste empieza a hablar sobre sus trabajos

anteriores, entonces nos damos cuenta que podríamos haber planteado una serie de reglas sobre porque esta persona es experto en su campo. Estas reglas son las siguientes:

regla 1

Si mientras la pasta hierve yo no presto atención a la olla con el agua, la pasta se echará a perder.

PRODUCCIÓN:

No estoy prestando atención a la olla -> la pasta se ha echado a perder

regla 2

Si estoy prestando atención a la olla y ésta contiene buena pasta, buena agua y buena sal, la pasta se cocinará bien.

PRODUCCIÓN:

A = Estoy prestando atención

B = El cuenco contiene buena pasta

C = El cuenco contiene buen agua

D = El cuenco contiene buena sal

E = La pasta se cocinará bien

$A \wedge B \wedge C \wedge D \rightarrow E$

regla 3

Si la pasta está hecha 100% con Semolina Durum y es importada de Italia es una buena pasta.

PRODUCCIÓN:

A = La pasta está hecha 100% con Semolina Durum

B = La pasta es importada de Italia

C = es una buena pasta

regla 4

Si la sal se ha añadido cuando el agua estaba hirviendo, echa la pasta en la olla.

PRODUCCIÓN:

La sal se ha añadido cuando el agua está hirviendo -> echa la pasta en la olla

regla 5

Si la pasta está "*al dente*" la pasta se ha cocinado.

PRODUCCIÓN:

La pasta está "*al dente*" -> la pasta se ha cocinado

regla 6

Si la pasta se ha cocinado echa la salsa a la pasta.

PRODUCCIÓN:

La pasta está cocinada -> echa la salsa a la pasta

Conclusiones

- La forma más habitual de representar el conocimiento en los sistemas expertos es mediante las reglas de producción.
- La base de conocimiento en un sistema experto consiste en el conjunto de experiencias y conocimiento que aporta un experto humano en el área.
- Los hechos pueden ser tanto DATOS como INFORMACIÓN

ANEXO 2:

DOCUMENTACIÓN JAVADOC DE LA APLICACIÓN JAVA PRESENTADA

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)[PREV PACKAGE](#) [NEXT PACKAGE](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

Package com.damal

Class Summary

Class	Description
Alert	
Correlador	
Dispositivo	
Evento	
Reproductor	
Request	
Response	

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)[PREV PACKAGE](#) [NEXT PACKAGE](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.damal

Class Alert

java.lang.Object
com.damal.Alert

```
public class Alert
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
private java.lang.String	Analytic tipo de datos recogido
private java.lang.String	Source origen de evento

Constructor Summary

Constructors

Constructor and Description
Alert()

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	getAnalytic()
java.lang.String	getSource()
void	setAnalytic (java.lang.String tipo)
void	setSource (java.lang.String fuente)

```
void toVoice(java.lang.String val)  
Metodo que envia al motor de reproduccion una alerta codificada  
como string
```

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

Analytic

```
private java.lang.String Analytic
```

tipo de datos recogido

Source

```
private java.lang.String Source
```

origen de evento

Constructor Detail

Alert

```
public Alert()
```

Method Detail

getAnalytic

```
public java.lang.String getAnalytic()
```

setAnalytic

```
public void setAnalytic(java.lang.String tipo)
```

getSource

```
public java.lang.String getSource()
```

setSource

```
public void setSource(java.lang.String fuente)
```

toVoice

```
public void toVoice(java.lang.String val)
```

Método que envía al motor de reproducción una alerta codificada como string

Parameters:

val - string que contiene las palabras a reproducir separadas por el signo +

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

com.damal

Class Correlador

java.lang.Object
com.damal.Correlador

```
public class Correlador
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type

```
private static java.lang.String

static java.util.ArrayList<Dispositivo>

private static java.util.ArrayList<java.lang.String>

static java.util.ArrayList<java.lang.String>

static java.util.ArrayList<java.lang.String>

static java.util.ArrayList<java.lang.String>

static java.util.ArrayList<java.lang.String>

static java.util.ArrayList<java.lang.String>

private static java.lang.String

private static java.lang.String
```

Field and Description

CONFIG
fichero de configuración siempre en el mismo directorio que aplicación

devices
ArrayList contenedor de objeto Dispositivo construidos a partir del campo IP

disp
ArrayList IP's de los dispositivos que aparecen bajo el fichero de configuracion del sistema

GRUPO_1
ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_1

GRUPO_2
ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_2

GRUPO_3
ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_3

GRUPO_4
ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_4

GRUPO_5
ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_5

OLD_DATA_DIRECTORY
fichero donde se ubicaran los eventos viejos

XML_DIRECTORY
directorio sobre el que se leeran los eventos (fichero xml-contenedor)

Constructor Summary

Constructors**Constructor and Description****Correlador()****Method Summary****All Methods****Static Methods****Concrete Methods**

Modifier and Type	Method and Description
private static void	createDevice() Metodo que crea los objetos Dispositivo a partir de las IP de los mismos
private static void	fetchEvents (org.kie.api.runtime.KieSession session) metodo que busca todos los eventos (ficheros .xml en la carpeta eventos, los parsea a objeto Evento e inserta en la sesion activa)
private static void	getConfig() Parsing del fichero de configuracion "config.cfg" para lectura de los diferentes detalles de configuración, directorio origen de eventos, directorio destino de eventos procesados y IP's asociadas a los diferentes grupos
private static void	getDevicesIP() Metodo que obtiene todas las diferentes IP's de dispositivos a partir de fichero de configuración
private static java.io.File[]	getXmlFilesFromDirectory (java.lang.String xmlDirectory) Metodo que monitoriza un recurso (carpeta) e inserta los eventos capturados en una sesion Kie
private static org.kie.api.runtime.KieSession	initCorrelator() Metodo de inicializacion del correlador bajo hilo de ejecucion que se mantiene a la espera de recibir hechos en sesion activa
static void	main (java.lang.String[] args) Metodo main, muestra por pantalla mensajes de carga de ficheros y configuración y hace asociacion de dispositivo a grupo de difusion
private static void	monitorizaRecurso (org.kie.api.runtime.KieSession kie) Metodo que monitoriza un recurso (carpeta) e inserta los eventos capturados en una sesion Kie
private static void	moveFile (java.lang.String source, java.lang.String destination) Metodo que elimina un fichero xml, moviendolo al destino
private static Alert	unmarshall (java.io.File fich) Metodo que elimina un fichero xml, moviendolo al destino

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail**XML_DIRECTORY**

```
private static java.lang.String XML_DIRECTORY
```

directorio sobre el que se leeran los eventos (fichero xml-contenedor)

OLD_DATA_DIRECTORY

```
private static java.lang.String OLD_DATA_DIRECTORY
```

fichero donde se ubicaran los eventos viejos

CONFIG

```
private static java.lang.String CONFIG
```

fichero de configuración siempre en el mismo directorio que aplicación

GRUPO_1

```
public static java.util.ArrayList<java.lang.String> GRUPO_1
```

ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_1

GRUPO_2

```
public static java.util.ArrayList<java.lang.String> GRUPO_2
```

ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_2

GRUPO_3

```
public static java.util.ArrayList<java.lang.String> GRUPO_3
```

ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_3

GRUPO_4

```
public static java.util.ArrayList<java.lang.String> GRUPO_4
```

ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_4

GRUPO_5

```
public static java.util.ArrayList<java.lang.String> GRUPO_5
```

ArrayList contenedor de las configuraciones/grupos de usuarios GRUPO_5

disp

```
private static java.util.ArrayList<java.lang.String> disp
```

ArrayList IP's de los dispositivos que aparecen bajo el fichero de configuracion del sistema

devices

```
public static java.util.ArrayList<Dispositivo> devices
```

ArrayList contenedor de objeto Dispositivo construidos a partir del campo IP

Constructor Detail

Correlador

```
public Correlador()
```

Method Detail

initCorrelator

```
private static org.kie.api.runtime.KieSession initCorrelator()
```

Metodo de inicializacion del correlador bajo hilo de ejecucion que se mantiene a la espera de recibir hechos en sesion activa

monitorizaRecurso

```
private static void monitorizaRecurso(org.kie.api.runtime.KieSession kie)
```

Metodo que monitoriza un recurso (carpeta) e inserta los eventos capturados en una sesion Kie

Parameters:

kie - sesion Kie

main

```
public static final void main(java.lang.String[] args)
```

Metodo main, muestra por pantalla mensajes de carga de ficheros y configuración y hace asociacion de dispositivo a grupo de difusión

Parameters:

args - main args

createDevice

```
private static void createDevice()
```

Metodo que crea los objetos Dispositivo a partir de las IP de los mismos

getDevicesIP

```
private static void getDevicesIP()
```

Metodo que obtiene todas las diferentes IP's de dispositivos a partir de fichero de configuración

fetchEvents

```
private static final void fetchEvents(org.kie.api.runtime.KieSession session)
```

metodo que busca todos los eventos (ficheros .xml en la carpeta eventos, los parsea a objeto Evento e inserta en la sesion activa)

Parameters:

session - KieSession en que se insertara el evento

getXmlFilesFromDirectory

```
private static java.io.File[] getXmlFilesFromDirectory(java.lang.String xmlDirectory)
```

Metodo que monitoriza un recurso (carpeta) e inserta los eventos capturados en una sesion Kie

Parameters:

xmlDirectory - directorio contenedor de los ficheros xml que representan los eventos

Returns:

ficheros capturados en directorio

unmarshall

```
private static Alert unmarshall(java.io.File fich)
```

Metodo que elimina un fichero xml, moviendolo al destino

Parameters:

fich - evento codificado bajo fichero xml

Returns:

objeto Alerta generado a partir del xml

moveFile

```
private static void moveFile(java.lang.String source,  
                             java.lang.String destination)
```

Metodo que elimina un fichero xml, moviendolo al destino

Parameters:

source - directorio origen del evento (fichero xml)

destination - directorio destino del evento

getConfig

```
private static void getConfig()
```

Parsing del fichero de configuracion "config.cfg" para lectura de los diferentes detalles de configuración, directorio origen de eventos, directorio destino de eventos procesados y IP's asociadas a los diferentes grupos

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS **NEXT CLASS** FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.damal

Class Dispositivo

```
java.lang.Object
  java.lang.Thread
    com.damal.Dispositivo
```

All Implemented Interfaces:

java.lang.Runnable

```
public class Dispositivo
  extends java.lang.Thread
```

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler

Field Summary

Fields

Modifier and Type	Field and Description
private java.lang.String	ident
(package private) java.util.LinkedList<java.lang.String>	queueMsg cola de mensajes a enviar a dispositivo

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

Constructors

Constructor and Description

Dispositivo(java.lang.String id)

Constructor de objeto Dispositivo

Method Summary**All Methods** Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	clearList() vacía la cola de eventos en dispositivo
java.lang.String	getIdent()
void	insertEvent (java.lang.String msg) inserción de orden de reproducción en cola de dispositivo
void	run() hilo activo de ejecución de dispositivo, si eventos en cola, envía orden de reproducción
void	setIdent (java.lang.String ident)

Methods inherited from class java.lang.Thread

activeCount, checkAccess, clone, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail**ident**

private java.lang.String ident

queueMsg

```
java.util.LinkedList<java.lang.String> queueMsg
```

cola de mensajes a enviar a dispositivo

Constructor Detail

Dispositivo

```
public Dispositivo(java.lang.String id)
```

Constructor de objeto Dispositivo

Parameters:

identificador - de dispositivo

Method Detail

getIdent

```
public java.lang.String getIdent()
```

setIdent

```
public void setIdent(java.lang.String ident)
```

clearList

```
public void clearList()
```

vacía la cola de eventos en dispositivo

insertEvent

```
public void insertEvent(java.lang.String msg)
```

inserción de orden de reproducción en cola de dispositivo

Parameters:

msg - mensaje codificado a enviar, palabras separadas por signo +

run

```
public void run()
```

hilo activo de ejecucion de dispositivo, si eventos en cola, enva orden de reproduccion

Specified by:

run in interface `java.lang.Runnable`

Overrides:

run in class `java.lang.Thread`

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

com.damal

Class Evento

java.lang.Object
com.damal.Evento

```
public class Evento
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
(package private) int	nivel
(package private) java.lang.String	origen
(package private) java.lang.String	tipo

Constructor Summary

Constructors

Constructor and Description

[Evento\(\)](#)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
int	getNivel()
java.lang.String	getOrigen()
java.lang.String	getTipo()
void	setNivel(int nivel)
void	setOrigen(java.lang.String origen)

```
void      setTipo(java.lang.String eventType)
void      toVoice(java.lang.String val)
          Metodo que envia al motor de reproduccion una alerta codificada
          como string
```

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

origen

java.lang.String origen

tipo

java.lang.String tipo

nivel

int nivel

Constructor Detail

Evento

```
public Evento()
```

Method Detail

getOrigen

```
public java.lang.String getOrigen()
```

setOrigen


```
public void setOrigen(java.lang.String origen)
```

getTipo

```
public java.lang.String getTipo()
```

setTipo

```
public void setTipo(java.lang.String eventType)
```

getNivel

```
public int getNivel()
```

setNivel

```
public void setNivel(int nivel)
```

toVoice

```
public void toVoice(java.lang.String val)
```

Metodo que envia al motor de reproduccion una alerta codificada como string

Parameters:

val - string que contiene las palabras a reproducir separadas por el signo +

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

com.damal

Class Reprodutor

java.lang.Object
com.damal.Reprodutor

```
public class Reprodutor
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
private static java.lang.String	USER_AGENT user agent

Constructor Summary

Constructors

Constructor and Description
Reprodutor()

Method Summary

All Methods Static Methods Concrete Methods

Modifier and Type	Method and Description
static void	reproduce (java.lang.String url) metodo que realiza el envio de peticion http a dispositivo remoto

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

USER_AGENT

```
private static final java.lang.String USER_AGENT
```

user agent

See Also:

[Constant Field Values](#)

Constructor Detail

Reproductor

```
public Reproductor()
```

Method Detail

reproduce

```
public static void reproduce(java.lang.String url)
```

metodo que realiza el envio de peticion http a dispositivo remoto

Parameters:

url - url completa que constituira la peticion http

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

com.damal

Class Request

java.lang.Object
com.damal.Request

All Implemented Interfaces:

java.util.concurrent.Callable<Response>

class **Request**
extends java.lang.Object
implements java.util.concurrent.Callable<Response>

Field Summary

Fields

Modifier and Type	Field and Description
private java.net.URL	url

Constructor Summary

Constructors

Constructor and Description
Request (java.net.URL url)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
Response	call ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

url

```
private java.net.URL url
```

Constructor Detail

Request

```
public Request(java.net.URL url)
```

Method Detail

call

```
public Response call()  
    throws java.lang.Exception
```

Specified by:

call in interface [java.util.concurrent.Callable<Response>](#)

Throws:

[java.lang.Exception](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.damal

Class Response

java.lang.Object
com.damal.Response

class **Response**
extends java.lang.Object

Field Summary

Fields

Modifier and Type	Field and Description
private org.omg.CORBA.portable.InputStream	body

Constructor Summary

Constructors

Constructor and Description

Response(org.omg.CORBA.portable.InputStream body)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
org.omg.CORBA.portable.InputStream	getBody ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

body

```
private org.omg.CORBA.portable.InputStream body
```

Constructor Detail**Response**

```
public Response(org.omg.CORBA.portable.InputStream body)
```

Method Detail**getBody**

```
public org.omg.CORBA.portable.InputStream getBody()
```

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV](#) [NEXT](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

Hierarchy For Package com.damal

Class Hierarchy

- java.lang.Object
 - com.damal.**Alert**
 - com.damal.**Correlador**
 - com.damal.**Evento**
 - com.damal.**Reproductor**
 - com.damal.**Request** (implements java.util.concurrent.Callable<V>)
 - com.damal.**Response**
 - java.lang.Thread (implements java.lang.Runnable)
 - com.damal.**Dispositivo**

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV](#) [NEXT](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)[PREV](#) [NEXT](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain six categories:

- Interfaces (*italic*)
- Classes
- Enums
- Exceptions
- Errors
- Annotation Types

Class/Interface

Each class, interface, nested class and nested interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description
- Nested Class Summary
- Field Summary
- Constructor Summary
- Method Summary
- Field Detail
- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Annotation Type

Each annotation type has its own separate page with the following sections:

- Annotation Type declaration
- Annotation Type description
- Required Element Summary

- [Optional Element Summary](#)
- [Element Detail](#)

Enum

Each enum has its own separate page with the following sections:

- [Enum declaration](#)
- [Enum description](#)
- [Enum Constant Summary](#)
- [Enum Constant Detail](#)

Use

Each documented package, class and interface has its own Use page. This page describes what packages, classes, methods, constructors and fields use any part of the given class or package. Given a class or interface A, its Use page includes subclasses of A, fields declared as A, methods that return A, and methods and constructors with parameters of type A. You can access this page by first going to the package, class or interface, then clicking on the "Use" link in the navigation bar.

Tree (Class Hierarchy)

There is a [Class Hierarchy](#) page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

Deprecated API

The [Deprecated API](#) page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Index

The [Index](#) contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Prev/Next

These links take you to the next or previous class, interface, package, or related page.

Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

All Classes

The [All Classes](#) link shows all classes and interfaces except non-static nested types.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

Constant Field Values

The [Constant Field Values](#) page lists the static final fields and their values.

This help file applies to API documentation generated using the standard doclet.

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV](#) [NEXT](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)