SISTEMA EXPERTO BASADO EN REGLAS PARA UNA APLICACIÓN DE MONITORIZACIÓN DE PRODUCCIÓN INDUSTRIAL

UNIVERSIDAD POLITECNICA DE VALENCIA

Trabajo Final de Máster desarrollado dentro del Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Alexander Curiel Robles

Dirigido por:
Federico Barber Sanchís

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Valencia, Septiembre 2013

SISTEMA EXPERTO BASADO EN REGLAS PARA UNA APLICACIÓN DE MONITORIZACIÓN DE PRODUCCIÓN INDUSTRIAL

Trabajo Final de Máster desarrollado dentro del Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Valencia, Septiembre 2013

A quien siempre me
apoya y me hace ver
el lado bueno de todo.

Agradecimientos

Este trabajo de fin de máster no se habría podido realizar sin la ayuda, apoyo y ánimos de varias personas que han influido de forma decisiva en su desarrollo. Por ello, quiero dedicarles las siguientes líneas.

En primer lugar, agradezco a edinn Global por haberme dado la oportunidad de trabajar en este proyecto innovador en su ámbito, y por confiar en mí para llevarlo a cabo. Gracias a todos por vuestra paciencia y dedicación. No ha sido fácil. Nos queda mucho camino por recorrer, esto sólo acaba de empezar.

También quiero dar las gracias a Federico Barber por todos sus consejos, por orientarme siempre que ha sido necesario, por estar disponible cada vez que lo necesitaba. Se aprecia una atención así.

Por último, nombrar a mi familia, novia y amigos. Ellos han sido un apoyo fundamental durante todo este tiempo. Sin ellos, la realización de este trabajo no habría sido lo mismo. Gracias.

Resumen

Compañías de todo tipo se interesán cada vez más en aumentar su rendimiento, reducir sus costes y disminuir su impacto ambiental. Para ello, hay que ser conscientes de que todo proceso tiene pérdidas. Si no se controlan continuamente estos procesos para saber qué esta ocurriendo, se puede estar perdiendo capacidad productiva y, por lo tanto, dinero.

El software edinn® M2 es un sistema que monitoriza automáticamente en tiempo real a las personas y las máquinas de cualquier sector, e integra las funciones y estándares necesarios para la mejora total de la eficiencia.

Para poder tener una visión lo más real posible de lo que ocurre en una compañía, y poder tomar de esta forma las medidas adecuadas para mejorar, es muy importante la correcta configuración del sistema edinn® M2. Para ello, se da la formación inicial necesaria al cliente. Un elevado porcentaje del personal que recibe la formación no está familiarizado con todos los conceptos de producción que se manejan en el sistema. Además, ciertos cambios que se realizan en el proceso de producción (introducción de nuevo producto, o proceso, una modificación de la agrupación para un área de la planta, etc.) requieren, como es evidente, reajustar los parámetros del sistema. No realizar adecuadamente esta tarea produce efectos no deseados, ya que la monitorización no se ajusta a la realidad, y por tanto carece de sentido. La detección de estos problemas no es inmediata, por lo que se agrava la situación para el cliente, que muchas veces termina pidiendo soporte para la solución de una incidencia que él mismo ha provocado.

En este marco, parece lógico pensar en que lo ideal sería que el cliente tuviese a su disposición una herramienta capaz de analizar lo que está ocurriendo y avisar del problema y sus causas. Esto tiene dos grandes ventajas. Una es que el sistema puede detectar la incidencia mucho antes que el cliente. La otra, es que en la mayoría de los casos el cliente conocerá la causa del problema y podrá solucionarlo él mismo, o pedir soporte pero aportando una idea clara de lo que ocurre.

La solución aquí planteada es un sistema experto capaz de manejar las complejas relaciones que existen en el proceso de producción, así como la correcta configuración de los parámetros del sistema respecto a ellas.

Índice

Índice de figuras

Capítulo 1

Introducción

El comienzo es más de
la mitad de la totalidad.

Aristóteles

Resumen: En este capítulo se realiza una introducción al problema y se plantea una solución concreta definiendo los objetivos que se desean alacanzar.

1.1. Introducción

Para entender el contexto en el que se desarrolla este trabajo, es necesaria una breve introducción a la producción en la industria y a una aplicación concreta que da una solución a la búsqueda de mejorar el proceso productivo mediante el control de sus componentes.

Desde hace años, con el gran crecimiento de la competitividad en la industria, las compañías comenzaron a prestar más atención a su rendimiento. Para ser más competentes en el mercado es necesario obtener una mayor productividad, incrementar la eficiencia y la calidad y, por tanto, bajar los costes. De esta forma se obtiene una mayor rentabilidad, aumentando las ventas y generando más ingresos a la par que se mejora la marca de la compañía. Para la consecución de este fin, las empresas utilizan diferentes técnicas (tales como habilidades Kaizen) y pueden disponer de sistemas de gestión. La situación actual de la industria muestra que la productividad está ligada al crecimiento de las pequeñas y medianas empresas y a su capacidad de implementar planes eficaces de productividad, eficiencia, calidad e innovación. Esto se refleja en el creciente interés de pequeñas empresas por herramientas que les sirvan para aumentar su eficiencia.

Una de estás herramientas es edinn® M2, desarrollada por la empresa edinn Global[1]. Consiste en una herramienta modular que constituye un sistema MES (Manufacturing Execution System) y OES (Operational Execution System) completo ya que puede ser integrado con cualquier ERP (Enterprise Resource Planning) y planificador.

Las funciones de edinn® M2 son:

- Planificación de la producción
- Mejora de la eficiencia productiva y energética
- Gestión de stock y materiales
- Captura de datos de producción
- Planificación y gestión del mantenimiento
- Aseguramiento de la calidad
- Gestión de los tiempos y la actividad
- Control de acceso

Figura 1.1: Software edinn® M2

Por tanto, una de las principales funciones de este sistema es monitorizar en tiempo real máquinas y personas, dando a conocer al usuario en todo momento la información que necesita para saber dónde está teniendo pérdidas y el motivo. Con esto es posible diseñar y aplicar metodologías para la mejora de forma continua, y detectar problemas en procesos de producción tales como cuellos de botella, paradas de máquinas, o rendimientos por debajo de lo especificado. Este sistema automatizado es de gran valor en contraste con

[1]Más información sobre edinn Global y sus productos en `www.edinn.com`

métodos más clásicos en los que se van realizando anotaciones en informes de forma rutinaria, pues por una parte, no se pierde tiempo en realizarlos, y por otra, la precisión del sistema no se puede poner en duda en comparación con un humano.

1.2. Descripción del problema

Una vez descritas las principales características del sistema edinn® M2, conviene explicar el proceso de instalación y uso de esta herramienta para comprender el problema que se plantea.

Actualmente existen dos vías para la instalación de la herramienta. La primera de ellas es mediante un equipo de técnicos que ayudan al cliente a la configuración del sistema. Además, se proporcionan cursos adaptados a los diferentes perfiles de usuarios dentro del propio cliente, pues cada tipo de usuario tiene permiso para acceder a ciertas partes de la aplicación. Estos cursos son fundamentales para la correcta asignación de parámetros de configuración por parte del cliente, pues si se cometen errores la monitorización no se corresponderá del todo con la realidad y no tendrá mucho valor. Durante el curso, a los usuarios con un perfil que les permita configurar el sistema, se les da la formación necesaria respecto a ratios, parámetros, reglas y cálculos fundamentales que se utilizan en producción en la industria. De este modo se familiarizan con los términos si es que los desconocían y son capaces de proporcionar al sistema los datos que necesita para funcionar correctamente. Pero no sólo es esto útil para una configuración inicial, si no para posteriores modificaciones que se han de realizar, ya sea por exigencias de la producción o por reajustes necesarios. La otra vía de instalación es de forma remota, siendo el cliente quien se descarga la herramienta y, con la ayuda de un asistente de configuración básica, ajusta los parámetros del sistema.

El problema que surge, y que es el objeto de estudio de este trabajo, es la detección de que el sistema está configurado correctamente. Esto será así si los datos proporcionados por el usuario se corresponden con la realidad. Entender la gran variedad de problemas que pueden surgir requiere tener unos buenos conocimientos en producción industrial y del sistema edinn® M2. Aunque esto no es de especial interés en esta fase, y por ello se tratará en los siguientes capítulos, a continuación se exponen algunos ejemplos simplificados para ilustrar una parte del problema.

Por ejemplo, se debe asignar a cada proceso un tiempo de ciclo y una cantidad de productos que genera en ese tiempo. Además, el proceso puede estar en diferentes estados, tales como producción, espera, dependencia de línea o fallo. Por otra parte, el contador de cantidad de productos generados por el proceso se comprueba cada cierto periodo de tiempo, y existe un valor

umbral de tiempo mínimo en estado de producción para que realmente se contabilice como producción real. Todos estos parámetros deben ser introducidos por el usuario en base a las especificaciones de los procesos y las mediciones que sean necesarias. De esta forma se establecen los objetivos respecto a los cuales se obtienen los datos de monitorización.

Siguiendo con ejemplos, en este sistema existen condiciones de relaciones temporales entre parámetros que a su vez se ven condicionados por el estado en que se encuentre el proceso. Por ejemplo, si estamos buscando la causa de una bajada en la producción y el estado fuese dependencia de línea, sabríamos que dependemos de un proceso anterior que habrá que evaluar, y que a su vez dependerá de su estado y parámetros, y así sucesivamente. Una comprobación sencilla sería ver para un proceso si la cantidad producida en el tiempo de ciclo asignado está en el rango esperado. Si fuese más baja, en función del estado y relación con procesos adyacentes podríamos deducir si es un problema del proceso o lo hereda. También podría ser que los parámetros de tiempo de ciclo o cantidad que se produce en ese tiempo no se correspondan con la realidad. En ese caso, el motivo podría ser que no se midieron correctamente los tiempos o las especificaciones de la máquina no son las esperadas. O podría ser que no esté bien configurado el tiempo de volcado de datos desde el contador, o que no superamos el umbral de tiempo mínimo en estado de producción.

Con estos ejemplos se puede interpretar que existe una gran variedad de casos debido a la implicación de diferentes parámetros interrelacionados. En la mayoría de las ocasiones no es sencillo detectar el origen del problema, y es por ello que los usuarios solicitan soporte para solucionarlo. Muchas de estas incidencias se deben a la introducción de nuevos productos que implican cambios en los tiempos o unidades de medida y no han sido correctamente registrados en el sistema, o cambios en un área de procesos. El usuario puede advertir que algo no funciona como espera si observa las gráficas y datos que el sistema muestra, pero nadie se dedica a observar de continuo todos los datos, si no que se analizan cada cierto tiempo. En caso de un error de configuración de este tipo, este no será detectado hasta que se consulten los datos después de varios tiempos de ciclo, lo que provoca que los datos anteriores no sean fiables. Esto supone un problema para el usuario, pues pierde tiempo en solucionarlo, y para edinn Global ya que tiene que atender la incidencia y analizar los datos para encontrar el origen del fallo.

1.3. Objetivos

De igual forma que, al atender una llamada reportando una incidencia, el técnico realiza una serie de preguntas para orientar el origen del problema, se desea crear un sistema capaz de aglutinar los conocimientos de este experto,

y que a partir de los datos de la compañía alojados en los servidores sea capaz de detectar los posibles comportamientos no deseados e incidencias. De esta forma, se puede avisar al usuario del error mucho antes de que él pueda apreciarlo y reduciendo el impacto de una configuración que contenga fallos. Se debe destacar que la validación no se puede realizar en tiempo real hasta que no se haya cumplido el tiempo de ciclo asignado para poder comprobar el nivel de producción. Si se tiene en cuenta el proceso de instalación en el que sólo interviene el usuario, es de gran utilidad poder comprobar el correcto funcionamiento del sistema y validarlo sin ninguna intervención por parte de edinn Global.

Por tanto, para alcanzar esta solución, será necesario:

- Estudiar de la viabilidad del proyecto
- Elegir un tipo de sistema experto
- Analizar qué metodología seguir para el desarrollo del sistema experto
- Adquirir los conocimientos de los expertos
- Adaptar los conocimientos al lenguaje del sistema experto
- Realizar prototipos sobre los que estudiar casos de prueba para evaluar su rendimiento
- Desarrollar una versión más consistente que pueda ser evaluada en un entorno real
- Implementar una solución para integrar la nueva herramienta en el sistema existente

No cabe duda de que integrar en la herramienta un sistema experto que cumpla estas especificaciones es de gran valor para ambas partes, edinn Global y cliente. Además, actualmente no se tiene conocimiento de ninguna otra herramienta dedicada a la monitorización de producción en la industria que cuente con un sistema de validación continuo capaz de avisar de posibles problemas e indicando sus soluciones. Entonces, con la implementación de este sistema propuesto, edinn Global se desmarca de su competencia directa con nuevas características innovadoras y que, desde luego, al cliente le resultan lo suficientemente interesantes como para decantarse por este producto.

1.4. Estructura de capítulos

A modo de orientación inicial, este trabajo está estructurado en los siguientes capítulos:

- El capítulo 2 se centra en el sistema experto. Se exponen conceptos generales sobre los sistemas expertos, el dominio concreto del sistema experto que se desea implementar, la justificación de esta solución frente a otras y, finalmente, la elección del motor de inferencia en función de las necesidades del proyecto.

- En el capítulo 3 se aborda la metodología que se aplica a la hora de diseñar sistemas expertos. Se explican las fases fundamentales de cualquier desarrollo de este tipo así como las definidas para el caso concreto de este sistema experto. Es de utilidad para conocer qué es lo que ocurre en cada una de las fases, entender el porqué de su necesidad y el flujo que existente entre ellas.

- El capítulo 4 trata el diseño y desarrollo del sistema experto propuesto. Se analizan los principales componentes definidos en el dominio del sistema para su modelado. Atendiendo a ello, se definen los prototipos desarrollados comentando las principales características de cada uno. Por último, se explica la integración del sistema experto dentro de la infraestructura ya existente de la aplicación edinn® M2.

- En el capítulo 5 se evalúan los resultados obtenidos por el sistema en sus diferentes fases de desarrollo, describiendo el método elegido para ello. También se analiza la integración del sistema experto en la aplicación edinn® M2, valorando su funcionamiento en una situación real atendiendo al rendimiento.

- El capítulo 6 pone fin a este trabajo enumerando una serie de conclusiones relacionadas con el desarrollo de esta aplicación, una valoración del sistema, su aplicabilidad y posibles desarrollos posteriores.

Capítulo 2

Sistema experto

Un experto es una persona que ha cometido todos los errores que se pueden cometer en un determinado campo.

Niels Bohr

Resumen: En este capítulo se expone el dominio del sistema experto, así como la justificación de este tipo de sistema frente a otras soluciones. También se tratan las características que se valoran del motor de inferencia basándose en las necesidades concretas de la aplicación y en su entorno. Finalmente se realiza una introducción al motor de inferencia elegido para el desarrollo del sistema con el fin de facilitar posteriormente la comprensión de las medidas tomadas en el diseño.

2.1. Dominio

Un sistema experto se ocupa de un subconjunto concreto de todo el conocimiento existente en el mundo. Este subconjunto se conoce como el dominio del sistema. El proceso de recopilación de la información necesaria para el desarrollo de un sistema experto basado en reglas se conoce como Ingeniería del Conocimiento. Entre toda esta información se encuentra la definición de los requisitos fundamentales. Es decir, conocer cuál es el problema que el sistema necesita resolver y de qué información dispone. Con la interpretación de estos datos se puede evaluar la importancia de cada elemento sobre el dominio y así identificar sus pilares, descartar la información no necesaria y con todo ello establecer sus fronteras.

En este trabajo se desea resolver un problema muy concreto que abarca conocimiento de producción industrial y conocimiento específico sobre el

funcionamiento y configuración de un sistema de monitorización de la producción. Tras un análisis del proceso de inferencia que sigue el experto, se pueden definir los elementos que intervienen en el sistema. Sobre ellos se realizarán las evaluaciones necesarias para llegar a un conjunto de conclusiones, tal y como el experto lo haría. Siguiendo el proceso descrito, se definieron los siguientes elementos del dominio siendo éstos identificados como las ideas principales con las que el experto trabaja a la hora de resolver un problema:

Procesos: Se entiende por proceso a todo desarrollo sistemático que conlleva una serie de pasos ordenados, los cuales se encuentran estrechamente relacionados entre sí y cuyo propósito es llegar a un resultado concreto. Un proceso puede estar formado por una máquina, una persona, o una combinación de ambos elementos. Los procesos suelen agruparse en áreas o líneas, y cada uno de ellos cuenta con unos prámetros de producción concretos tales como tiempos, cantidades, productos, ratios estimados. Estos últimos pueden ser de eficiencia, velocidad, disponibilidad o calidad. Están definidos en rangos de validez, indicando si se alcanzan los objetivos, no se llega a ellos o se sobrepasan.

Estados: Los estados representan las posibles situaciones en las que se puede encontrar un proceso. Algunos de estos estados pueden ser producción, dependecia de línea, parada, fallo o no planificado para producir. No todos los procesos poseen los mismos estados, pues cada proceso tiene sus peculiaridades, aunque sí que suelen compartir algunos estados elementales.

Productos: Un producto es el resultado que se obtiene a la salida de un proceso. Varios productos pueden asociarse un proceso, y varios procesos pueden pueden compartir productos. Los productos se asocian con procesos pero también con estados, ya que puede existir más de un estado de producción para el mismo proceso. Un producto lleva asociado un identificador, sus unidades de medida y fechas para el control de su creación entre otros parámetros.

Órdenes: Las órdenes indican los productos que se deben generar, en qué procesos, cuánta cantidad y fechas de inicio y fin. Estas órdenes están orientadas a la planificación de producción, y se encuentran relacionadas directamente con los procesos y los productos.

Parámetros de monitorización: Para la monitorización de una planta mediante el sistema edinn® M2 son necesarios definir algunos parámetros como tiempos de volcado de datos, de lectura de contadores, de reseteo de contadores, o control de los umbrales de producción y sus tasas. Estos parámetros afectan a lo que el sistema lee e interpreta, y está directamente relacionado con los procesos de producción y los estados en los que se encuentran éstos.

Entre estos elementos existen, por tanto, relaciones de dependencia. Estas relaciones son muy concretas e influencian al conjunto del sistema. De no establecerse correctamente u omitirse se obtendría un sistema que carecería de sentido. Por ejemplo, un proceso siempre se encuentra en un estado concreto, y este estado se define mediante la monitorización en función de sus parámetros configurados. Entonces, estos tres elementos forman una unidad indivisible a efectos de analizar el sistema, pues existe una conexión entre todos ellos y alterar un elemento influye en los otros. Este criterio de análisis de subconjuntos indivisibles dentro del dominio ha servido para definir unas fases de desarrollo mediante prototipos con diferente alcance dentro del problema, como se verá en los siguientes capítulos.

Una forma de representar estos grados de dependencia es realizar una lista ordenada que muestre la jerarquía de los elementos dentro del dominio según su influencia en los demás componentes, tanto de forma directa como indirecta. Si se enumeran en orden ascendente según el número de elementos del dominio sobre los que actúa, se obtiene la siguiente relación:

- Parámetros de monitorización
- Procesos
- Estados
- Productos
- Órdenes

Aplicando este análisis a la definición del problema en base al conocimiento transmitido por el experto se tiene que:

- Los parámetros de monitorización por sí mismos no tienen gran valor de análisis si no se relacionan con otros elementos, especialmente con los procesos y sus estados
- De la misma forma, los procesos han de combinarse al menos con los parámetros de monitorización, que implican un estado concreto del proceso, para alcanzar conclusiones
- Los estados son necesarios para los procesos, ya que a cada proceso le pertenece un conjunto de estados concreto, y además dependen de parámetros de monitorización (por ejemplo, tiempo necesario en producción para activar el estado de producción)
- Los productos se asocian a procesos y a estados de tipo producción, pero no tienen relación directa con los parámetros de monitorización

- Las órdenes se definen para productos y procesos, y tampoco tienen relación directa con los parámetros de monitorización

Con esta definición del dominio puede comprobarse que los casos de detección de problemas puede llegar a ser bastante amplio, con muchas relaciones y, en consecuencia, su programación no es trivial. Como avance, en el capítulo 3 se revelará la metodología seguida para el desarrollo del sistema experto, y en el capítulo 4 se estudiará su diseño sobre el motor de inferencia en base al dominio aquí presentado.

2.2. Justificación

La necesidad de agrupar el conocimiento del experto para dar solución al problema planteado es evidente. El dominio del problema resulta complejo, con gran cantidad de relaciones, parámetros y reglas. Si se conoce el funcionamiento y potencial de los sistemas expertos, estos datos parecen indicar que integrar uno de estos sistemas en la herramienta podría ser la solución más adecuada. Pero antes de tomar esa decisión es necesario analizar cuáles son las ventajas de los sistemas expertos frente a una solución más clásica, orientada de forma procedural, para poder realizar una correcta valoración.

En primer lugar, el experto podrá hacer una transferencia de conocimiento, en base a su propia experiencia, definiendo los pasos y reglas que sigue para hallar la solución a los diferentes problemas dentro de un dominio concreto. Este conocimiento se almacena en el sistema y no se pierde con el tiempo, e incluso se puede mejorar en base a nuevas experiencias del experto o a la colaboración de otros expertos. Cuando se realiza un análisis del conocimiento transferido, en determinadas situaciones la información puede ser poco precisa utilizando conceptos que no se pueden medir con exactitud como poco y mucho, bueno y malo, etc. Para poder interpretar estos conceptos es necesario hacer uso de la lógica difusa, mediante la cual podemos definir el grado de pertenencia de un elemento a un grupo. Muchos de los sistemas expertos disponen de funciones para modelar este tipo de conocimiento.

También es de importancia destacar que la solución del experto en ocasiones no es única, sino que puede estar formada por una combinación de posibles soluciones. El motivo de que esto ocurra puede ser que las reglas que se aplican a un problema o caso no proporcionan un resultado con el que se pueda definir con certeza un dato de salida, bien porque no se disponga de algún parámetro al aplicar las reglas correspondientes o porque el sistema no puede tener conocimiento real sobre algún elemento. Los sistemas expertos basados en reglas cuentan con esta capacidad, lo que los hace perfectos para este tipo de razonamientos frente a la utilización de lenguajes clásicos. Ofrecen, por tanto, una gran flexibilidad frente a datos fragmentados o poco

precisos, siendo capaces de encontrar las posibles combinaciones de soluciones en un problema sin necesidad de modificar el código. Esto se debe a la utilización de patrones, un punto fuerte a tener en cuenta frente a soluciones más clásicas.

Utilizar lenguaje declarativo como el de los sistemas basados en reglas permite centrarse directamente en el problema y en cómo aportar soluciones. Describe qué es lo que se debe hacer, pero omite un orden y en muchos casos el cómo hacerlo, a diferencia de una programación procedural. Otra ventaja es su gran flexibilidad frente a datos fragmentados o poco precisos. El programa será capaz de encontrar las posibles combinaciones de soluciones aunque falten por definir parámetros sin necesidad de modificar el código. Esto es debido a la utilización de patrones, un punto fuerte a tener en cuenta frente a una solución más clásica. Es también por ello que en general, con un buen desarrollo, un sistema experto presenta una buena escalabilidad. Es capaz de adaptarse ante el crecimiento del dominio sin perder calidad mediante la adición de nuevas reglas que satifagan las nuevas necesidades.

A parte de las ventajas hasta ahora mencionadas, debe resaltarse que la programación declarativa es en muchos casos la forma más natural de enfrentarse a problemas en los que se ven envueltos el control, el diagnóstico, la predicción o la clasificación entre otros. En resumen, es aplicable a todos aquellos problemas que no poseen una solución algorítmica clara, como el que se trata en este trabajo. Este tipo de sistemas cuentan con una ventaja sobre los expertos humanos: a diferencia de las personas, pueden trabajar 24 horas al día durante un periodo de tiempo indefinido. Si además el sistema experto es tan fiable como el experto humano, hasta el punto de ser capaz de sustituirlo, es algo muy valorado si existe la necesidad de la figura del experto en cualquier momento del día. Existen casos de éxito desde hace décadas, como por ejemplo el sistema experto MYCIN (R. Davis, 1977) en la década de los 70 dentro del campo de la medicina, que posteriormente dio lugar a EMYCIN (Melle, 1979), uno de los primeros *frameworks* para sistemas expertos. Actualmente existe una buena cantidad de aplicaciones que hacen uso de este tipo de sistemas en campos tales como el científico, comercio, ingeniería y finanzas entre otros, y trabajando siempre sobre unos dominios muy específicos.

Volviendo al problema planteado, puede apreciarse que todas estas características descritas son necesarias para la aplicación que se desea obtener. Por ejemplo, una fábrica puede trabajar durante la madrugada y si ocurre una incidencia de este tipo no cuenta con soporte hasta después de unas horas. Con un sistema experto tendría atención inmediata e incluso antes de que detecten que existe un problema. Respecto a la forma de enfrentarse al problema, este tipo de programación encaja perfectamente con lo que se propone. Los casos posibles que se pueden dar cuentan con soluciones que

pueden llegar a ser una combinación de soluciones, debido a que no se cuenta con todos los datos de la realidad, si no con los introducidos por el usuario. Por ejemplo, si un proceso no esta produciendo lo que debe en un periodo de tiempo definido, puede darse la situación en la que el sistema no sepa si es debido a un problema en la configuración del umbral de producción, un fallo del sensor o, si la velocidad de la máquina es la adecuada, la cantidad a producir por unidad de tiempo se estimó mal. Sin embargo el sistema puede devolver las posibles causas del fallo, con lo que el usuario puede desplazarse al lugar y comprobar cuál de esos casos es el que ocurre. Otra necesidad muy importante en este sistema, y que como se ha citado anteriormente este tipo de sistemas cumple, es su capacidad de crecimiento. La herramienta edinn® M2 siempre se encuentra en estado de crecimiento para satisfacer nuevas necesidades de los clientes, lo que implicará en ocasiones un aumento del dominio del sistema al que ha de adaptarse sin que esto implique una pérdida de tiempo o calidad.

Para que el sistema experto sea una realidad, es necesario transmitir el conocimiento del experto al sistema definiendo un mapa de inferencia, lo que conlleva un gran esfuerzo. Generalmente en el desarrollo de sistemas expertos este paso es el más costoso y, sin duda, de mayor importancia debido a que condiciona que el posterior desarrollo sea un éxito. La metodología a seguir durante la implementación de la aplicación también afectará de forma decisiva. Se debe analizar el diseño que se desea realizar sobre un motor de inferencia para adaptarlo a las necesidades de la forma más eficiente y establecer unas fases de testeo de prototipos para la evaluación del proyecto.

Si después de este análisis se sigue pensando que incluir un sistema experto es la solución, debemos analizar que se cumplen los siguientes puntos, como indica George Rudolph en su artículo (Rudolph, 2008) en la web del proyecto de Jess, para garantizar que un sistema experto es la mejor opción:

- El número de ramificaciones condicionales o de toma de decisiones en el algoritmo es significante
- Las decisiones son complejas, compuestas de al menos tres condiciones para cada regla
- El algoritmo debe ser flexible, las reglas pueden cambiar debido a la naturaleza de la aplicación
- El código va a ser mantenido a lo largo del tiempo, no es un proyecto puntual sin mantenimiento de la aplicación
- El rendimiento no es un factor crítico, no se necesita optimizar la velocidad o el uso de la memoria

- Analizar el retorno de la inversión y la capacidad para costear los gastos durante el ciclo de vida del proyecto

En esta proyecto se cumplen estos puntos ya que:

- Existe un gran número de ramificaciones y un elevado número de casos a contemplar
- Se ha mostrado que las decisiones a tomar en este dominio son complejas, en función de subconjuntos relacionados
- Se necesita una aplicación flexible por la naturaleza de la herramienta, que implementa nuevas características dependiendo de las necesidades
- El proyecto se va a mantener a lo largo del tiempo aumentando su dominio gradualmente
- El rendimiento de velocidad y memoria no es crítico, pero sí necesario de cuantificar para garantizar un hardware lo suficientemente potente como para que la integración de esta herramienta no afecte al funcionamiento de edinn® M2
- El retorno de inversión de momento no es un problema, en parte porque la inversión hasta ahora no ha implicado un gasto muy significativo

Como conclusión, incluir un sistema experto en la herramienta parece lo acertado por las características de su lenguaje, su flexibilidad, disponibilidad, escalabilidad y el tipo de problema que resuelve. Sin embargo, se requiere una gran inversión inicial de tiempo y esfuerzo debido a la necesidad de adquirir los conocimientos del experto y, a partir de ellos, diseñar modelos de inferencia válidos para la aplicación.

2.3. Necesidades del motor de ineferencia

La pieza fundamental en un sistema experto basado en reglas es el motor de inferencia o de reglas. Éste se encarga de ejecutar, mediante la aplicación de un algoritmo, las reglas en el momento en que sean necesarias a partir de los hechos que conozca para hallar una solución, sin necesidad de ser guiado por el programador.

A la hora de elegir el motor de reglas se evaluaron las siguientes características en función de las necesidades:

Entorno de ejecución: Es interesante que el motor sea multiplataforma, porque de esta manera no nos vemos limitados a un sistema operativo en concreto y, puesto que se desea que sea una tarea que realice el servidor, de esta forma no presenta ningún problema para funcionar en diferentes tipos de servidores.

Características del lenguaje: Se espera utilizar un lenguaje intuitivo y robusto, capaz de expresar el problema de forma clara, y que permita agrupar conceptos para facilitar su escalabilidad.

Integración con la herramienta edinn® M2: Si se dispone de una solución integrada o API que permita las conexiones entre el motor, los servidores y la herramienta edinn® M2, y se puede embeber en otra aplicación, se valora muy positivamente ya que ahorrá tiempo de desarrollo.

Rendimiento: Se necesita un motor rápido, pero sin necesidad de ser esta característica una necesidad crítica por el momento. También es interesante el consumo de memoria ya que influye directamente en el rendimiento del servidor donde se aloje.

Documentación: Que la herramienta disponga de una buena documentación, ejemplos, versiones de prueba, o exista una comunidad de usuarios, es muy valorada ya que esto probablemente reduzca la curva de aprendizaje.

Tipo de licencia: Un factor importante a la hora de desarrollar una aplicación comercial es la licencia. Se valora el software libre, pero no parece que exista una solución tan robusta como las de pago.

Teniendo en cuenta las necesidades descritas y buscando un equilibrio entre la funcionalidad del lenguaje, rendimiento[1],tiempo de desarrollo y coste se decide utilizar el motor de inferencia Jess frente a otros como OPSJ o JRULES que carecen de varias de sus características más útiles. La valoración de Jess puede encontrarse en la sección 2.4.3.

2.4. Jess

A continuación, se realiza una introducción a la herramienta de lenguaje basado en reglas elegida, Jess, y se exponen las principales características por las que se decidió utilizar para implementar una solución.

2.4.1. Introducción

Jess es un motor de reglas y un lenguaje de scripting desarrollado a finales de los 90 por Sandia National Laboratories en Livermore, California. Está escrito en Java, por lo que es una herramienta ideal para integrar en software basado en Java. Jess se inspiró en el conocido CLIPS, un motor de

[1]Pueden consultarse los resultados de Jess frente a otros motores en `http://www.mail-archive.com/jess-users@sandia.gov/msg03278.html`

reglas escrito en C, para su desarrollo. Aunque existen similitudes entre estos lenguajes de reglas, en realidad sus implementaciones son muy distintas. A diferencia de CLIPS, Jess es dinámico y basado en Java, lo que permite acceder automáticamente a todas las APIs de Java, tales como accesos a bases de datos, gráficos, etc.

Entre Jess y CLIPS, a pesar de contar con una sintaxis muy similar, existen diferencias notables. Por ejemplo, Jess cuenta con constructores como `defclass`, `definstance`, o `defmodule` con semántica muy distinta a la de CLIPS, y ofrece nuevas características como `defquery`. A su vez, Jess no implementa todo lo que CLIPS ofrece, como por ejemplo COOL (CLIPS Object Oriented Language).

Jess se ha utilizado en el desarrollo de un amplio rango de software comercial, incluyendo entre otros:

- Sistemas expertos de evaluación de seguros e hipotecas
- Detectores de intrusión en redes y auditores de seguridad
- Asistentes de diseño para ingenieros
- Aplicaciones de predicción de la bolsa
- Webs de e-commerce
- Juegos

2.4.2. Características principales

Similar a otros motores de inferencia, Jess está formado por los elementos básicos de un sistema experto:

- Base de hechos, alojada en una memoria global para datos que modela el entorno del problema
- Base de reglas o conocimiento, que contiene todas las reglas que modelan el problema
- Motor de inferencia, que controla la ejecución de las reglas mediante un comparador de patrones

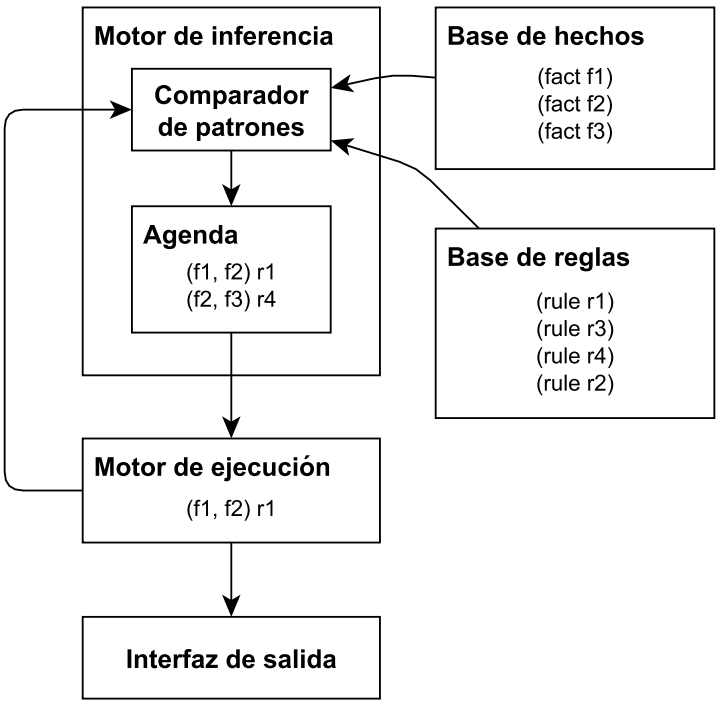

Figura 2.1: Elementos principales de Jess

El motor de inferencia controla todo el proceso de aplicación de reglas a la base de hechos para la obtención de salidas en el sistema. Es, por tanto, la pieza central de un motor de reglas. Generalmente un motor de inferencia funciona mediante ciclos de la siguiente forma:

- Se comparan todas las reglas con la base de hechos mediante el comparador de patrones para decidir qué reglas se deben activar en ese ciclo. La unión de estas reglas activadas junto con las activadas en ciclos anteriores forman el conjunto conflicto.
- Se ordena el conjunto conflicto para formar la agenda, que es la lista de reglas cuya parte derecha se ejecutará. El proceso de ordenación del conjunto conflicto se llama resolución del conflicto. La forma en que este proceso se lleva a cabo depende de muchos factores, y sólo alguno de ellos están bajo el control del programador.
- Para finalizar el ciclo se ejecuta la primera regla de la agenda, lo que probablemente modificará la base de hechos, y se vuelve a repetir todo el proceso.

El motor de inferencia de Jess utiliza lógica de primer orden y soporta tanto *forward chaining* como *backward chaining*. Esto puede resultar interesante en función de la aplicación.

La base de reglas o conocimiento almacena todas las reglas que el sistema conoce. Mediante un compilador de reglas, éstas se almacenan de una forma óptima para facilitar el trabajo al motor de inferencia. El compilador de reglas de Jess construye una estructura indexada compleja conocida como red *Rete* (Forgy, 1982), que acelera el procesamiento de las reglas.

La base de hechos contiene toda la información con la que el sistema está trabajando en un determinado momento. Puede contener tanto las premisas como las conclusiones de las reglas. Las reglas operan con esta información. Esta base de hechos puede contener diferentes tipos de estructuras para almacenar los datos. En el caso concreto de Jess también puede almacenar objetos Java.

El comparador de patrones decide las reglas aplicables en función del contenido de la base de hechos. De esta forma el motor de inferencia puede decidir las reglas a ejecutar y el momento adecuado. La comparación de patrones suele ser la fase más costosa en este tipo de sistemas.

La agenda representa una lista ordenada de reglas potenciales a ejecutarse, con las que el motor de inferencia comparará las reglas que debe lanzar, y de esta forma se decidirá la prioridad de ejecución. La agenda es la responsable de implementar una estrategia para la resolución de conflictos entre reglas y decidir la prioridad de cada una de ellas. Es posible añadir prioridad a las reglas en el momento en que se declaran.

El motor de ejecución es la parte de un motor de reglas que ejecuta las reglas. Procesa por tanto la acción que implica una regla. En el caso de Jess, la acción puede ejecutar código compilado ya que se define un lenguaje de programación completo.

Jess se puede utilizar en un entorno multihilo y es generalmente rápido debido al algoritmo *Rete* que utiliza para la coincidencia de patrones en las reglas, pero a cambio sacrifica algo más de memoria. Puede utilizarse en línea de comandos, mediante un GUI, servlets o applets.

Jess ofrece la posibilidad de programar de diferentes formas, utilizando el lenguaje de reglas propio de Jess o en Java mediante el acceso a sus clases. Esto es de gran utilidad, ya que en la misma aplicación se pueden utilizar ambos tipos de programación, decidiendo el nivel de implicación de cada uno en función de las necesidades. También permite embeber Jess en un aplicación Java y extender el lenguaje de Jess para desarrollar nuevas funciones, lo que potencia su utilidad y rápido desarrollo.

Uno de los puntos más importantes a la hora de desarrollar una aplicación con Jess es elegir la arquitectura de entre las numerosas posibilidades. Una de las formas de organizar estas posibilidades es mediante una lista ordenada en función del nivel de implicación de Java de forma ascendente:

- Puro lenguaje Jess, sin código Java
- Puro lenguaje Jess, pero el programa accede a las APIs de Java
- La mayoría de código es Jess, pero con parte de código en Java para personalizar nuevos comando de Jess
- Mitad de lenguaje Jess y mitad de Java con un uso sustancial de este para nuevos comandos y uso de las APIs
- Código Java en su mayoría, que carga código Jess en su ejecución
- Todo es código Java, que controla Jess completamente desde su API de Java

Puede encontrarse información más detallada sobre el funcionamiento y las características de este motor de reglas en el libro su creador (Friedman-Hill, 2003) y en la web del proyecto (Friedman-Hill, 2009).

2.4.3. Valoración

La posibilidad de embeber Jess directamente en una aplicación Java es una gran ventaja para nuestro fin, pues podemos incluir las clases necesarias en nuestro proyecto, que necesita una comunicación con la base de datos para obtener los parámetros correspondientes de los clientes de forma dinámica, y una comunicación con la herramienta edinn® M2. Esto evita la implementación de una capa de comunicación, por lo que acelera el desarrollo e incrementa su usabilidad.

La capacidad de poder utilizar el lenguaje de Jess en conjunto con Java en el grado de fusión que se desee, es muy útil ya que, por ejemplo, podemos crear conjuntos de reglas de forma rápida y sencilla, añadir nuevos comandos personalizados según las necesidades de la aplicación, o hacer uso del multihilo para procesar datos en paralelo simplemente instanciando un nuevo objeto de la clase que implementa el motor de inferencia.

Un punto más a su favor es la extensa documentación que se ofrece para el desarrollo de aplicaciones, con diferentes manuales y libros, y un foro activo donde los usuarios pueden compartir dudas y sus desarrollos. El paquete de Jess consta de un conjunto de ejemplos variados que sirven de complemento para los manuales y facilitan el aprendizaje notablemente.

Por último, decir que si bien Jess no impone ninguna restricción de su uso para investigación y docencia, sí que es necesario una licencia especial para aplicaciones comerciales.

En resumen, Jess un motor de reglas multiplataforma, que puede ser embebido en aplicaciones Java, que permite programar en un lenguaje clásico

de reglas y en Java, y que cuenta con un gran número de funciones que facilitan desde la creación de plantillas a conexiones remotas para adquirir datos, o la posibilidad de utilizar lógica difusa. Se ajusta perfectamente a las necesidades de la aplicación que se desea desarrollar ya que cumple con creces lo demandado, y ofrece una capacidad de personalización y escalabilidad mucho mayor que otros motores.

Capítulo 3

Metodología

We understand human mental processes
only slightly better than a fish
understands swimming.

John McCarthy

Resumen: Este capítulo se centra en explicar cuáles son las principales fases en las metodologías de desarrollo en sistemas expertos, y en definir la metodología adoptada para el sistema experto de este trabajo.

3.1. Metodologías de desarrollo

El desarrollo de sistemas expertos es un proceso costoso y complejo que requiere una metodología para poder alcanzar los objetivos fijados. Las diferentes metodologías comparten tres fases principales:

- Adquisición del conocimiento
- Análisis y modelado del conocimiento
- Verificación del conocimiento

La adquisición del conocimiento suele ser considerada como el cuello de botella en la construcción de sistemas expertos, pues una de las tareas más complejas en esta primera etapa es capturar el conocimiento que es relevante para el sistema. Los expertos, aunque sean colaborativos, pueden aportar un conocimiento al que le falta precisión o le sobran datos.

Realizar un buen análisis del conocimiento adquirido es fundamental para modelarlo adecuadamente en el lenguaje sobre el que se trabajará. En

consecuencia, un buen análisis puede facilitar bastante la tarea de programación. En esta etapa se deben definir las estructuras necesarias para definir el dominio del problema y diseñar la base de conocimiento.

Por último, la fase de verificación del conocimiento consiste en el testeo de las inferencias del sistema. Éstas se evaluan contra las que aporta el experto, y si no son las deseadas se sigue refinando la aplicación.

La forma en que estas fases se desarrollan, las iteraciones y transiciones sobre cada una de ellas, dan lugar a diferentes interpretaciones de cómo se puede abordar el desarrollo de un sistema experto. Esto se manifiesta en las diferentes metodologías que han surgido a lo largo de los años. Por ejemplo, la metodología KADS (B. Wielinga, 1991) surgió en Europa con el fin de establecer un modelo de desarrollo de este tipo de sistemas. KADS era una oposición a otras metodologías conocidas como cíclicas o de rápido prototipado, que proponían un análisis parcial del conocimiento para rápida creación de prototipos. Por el contrario KADS apuesta por el análisis del conocimiento al completo antes de cualquier desarrollo para, de esta forma, estimar mejor la duración de cada fase y desarrollar teniendo un conocimiento completo de las posibles estructuras. Actualmente existe una gran variedad de metodologías para el desarrollo de sistemas expertos, que consisten en fusiones de otras metodologías y nuevos métodos para iterar entre las fases de desarrollo. Como ejemplo, grupos con experiencia en el desarrollo de sistemas expertos proponen una metodología que se basa en iterar entre estas fases describiendo una espiral (Yasser Abdelhamid, 2005) para garantizar una validación continua y un diseño de prototipos en diferentes fases del proyecto. Otro ejemplo son los desarrollos condicionados por los testeos iterativos conocidos como *test-driven development*, influenciados por la metodología conocida como *eXtreme Programming* (Beck, 2000) en ingeniería del software.

3.2. Metodología empleada

Los pilares de la metodología empleada para el desarrollo de este sistema experto, basándose en las necesidades concretas de este proyecto, son:

- Adquisición del conocimiento
- Estructuración del conocimiento
- Diseño de las reglas
- Construcción de prototipos
- Testeo iterativo
- Desarrollo iterativo

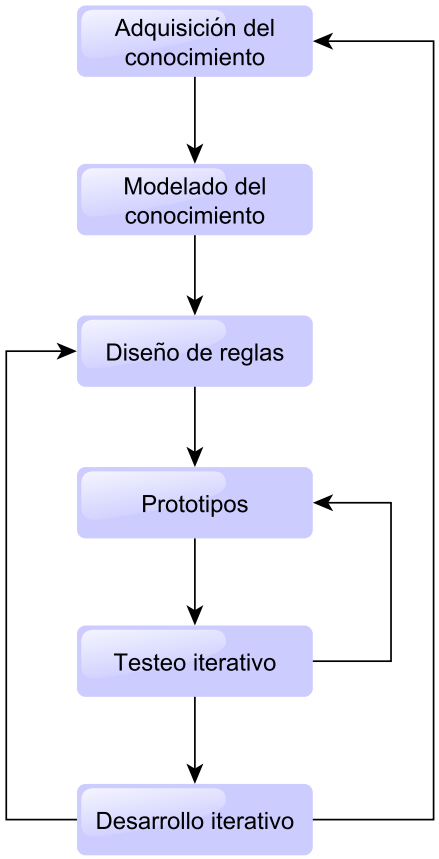

Figura 3.1: Metodología aplicada en el desarrollo del sistema experto

En la figura 3.1 puede apreciarse el flujo entre las diferentes etapas que se describirán a continuación.

3.2.1. Adquisición del conocimiento

La primera fase, adquisición del conocimiento, consiste en conseguir el conocimiento del experto. Para que esta fase sea exitosa se debe contar con la dedicación del experto y que éste sea capaz de expresar su conocimiento de una forma clara y concisa. Esta parte es sin duda la más costosa.

En este trabajo se ha contado con la colaboración de dos expertos de edinn Global. Ambos conocen el funcionamiento del sistema edinn® M2, ya que han participado en su desarrollo, y tienen conocimientos de producción industrial. Asisten al cliente cuando tiene problemas, y están presentes en instalaciones del sistema para dar formación y ayudar en la configuración del sistema.

3.2.2. Estructuración del conocimiento

Una vez adquirido el conocimiento, se puede empezar a pensar en cómo transmitir ese conocimiento al sistema. Por ello es necesario analizar los datos y estructurarlos de forma que se simplifique en la medida de lo posible la implementación de las reglas.

Esto tiene una similitud con el análisis que se realiza en lenguajes orientados a objetos. Inicialmente, se han de detectar los conceptos más importantes del conocimiento adquirido y, a continuación, se han de identificar las variables o parámetros que poseen cada uno de estos conceptos. Esto se corresponde con la definición del dominio del sistema experto, tal como se expuso en la sección 2.1.

Las estructuras en las que almacenar los datos dependen de la herramienta sobre la que se desarrolle. En el caso de Jess, podemos utilizar módulos para almacenar diferentes conjuntos de reglas, y disponemos de un conjunto de estructuras definidas para almacenar datos, plantillas de patrones y funciones.

3.2.3. Diseño de reglas

Una vez que se tiene definida una estructura de datos, se puede comenzar a diseñar las reglas. Para que el sistema sea escalable y resulte más sencillo de comprender, Jess ofrece la posibilidad de definir módulos con reglas. Por tanto, se pueden agrupar conjuntos de reglas en módulos cuando éstas son sólo relevantes en una parte específica de la ejecución. En este trabajo se hace uso de los módulos para agrupar los conceptos principales definidos en el dominio, pues cada uno de ellos consta de reglas propias como se verá en el capítulo 4.

3.2.4. Construcción de prototipos

Un prototipo consiste en una versión simplificada de la versión final. En este trabajo se han establecida diferentes fases de desarollo en función de su complejidad y el número de conceptos implicados. Concretamente, se establecen tres prototipos que fijan tres objetivos que se desean alcanzar de forman progresiva. La estructura de estos proptotipos y sus objetivos se comentan en el capítulo 4. Además de estos prototipos, en el desarrollo se crearán varios subprototipos entre cada una de estas fases para evaluar el desarrollo de manera continua.

3.2.5. Testeo iterativo

Esta metodología enfatiza el testeo de prototipos en diferentes fases de desarrollo y complejidad. Esta elección se debe a la creencia de que, si se es

riguroso en cada un de la fases de desarrollo que se definan, el sistema final será más robusto y más modular, con la ventaja de la depuración continua que reducirá los problemas una vez alcanzada la fase final.

Para llevar a cabo esta estrategia se definen unos casos de test que varían en cada etapa de desarrollo. Estos casos de test se construyen junto al experto adecuándolos a cada fase. Utilizando un framework proporcionado por Jess para este fin, resulta muy cómodo y práctico ejecutar casos de prueba cada vez que es necesario y comprobar si los resultados son los esperados. En función del número de casos en los que el sistema infiera la misma solución que aporta el experto, se puede estimar cómo de cerca se encuentra el prototipo de alcanzar sus objetivos para una fase concreta.

3.2.6. Desarrollo iterativo

Una vez comenzada la fase de implementación de las reglas, se puede dar el caso en el que no se pueda continuar porque falta información. Esto puede ser porque no se analizó correctamente el conocimiento, o porque el experto omitió involuntariamente alguna información acerca de sus conocimientos y esto no se había detectado hasta ese momento. Este procedimiento iterativo de desarrollo encaja perfectamente con el desarrollo de sistemas expertos basados en reglas. Añadir una nueva regla, en la mayoría de ocasiones, no es costoso.

Capítulo 4

Diseño y desarrollo del sistema

Hay que aprender las reglas del juego. Y luego tienes que jugar mejor que nadie.
Albert Einstein

Resumen: En este capítulo se describe el diseño del sistema experto adaptado al motor de inferencia Jess. También se muestran los prototipos desarrollados para su posterior evaluación. Por otra parte, se explicará la integración de este sistema con la herramienta edinn® M2.

4.1. Diseño

Como se ha comentado en los capítulos anteriores, gracias a la flexibilidad de Jess, éste puede utilizarse como lenguaje de scripting de Java, como sólo código Java, o simplemente como el lenguaje de reglas definido por Jess. Estas técnicas pueden combinarse en el grado que se desee, aumentando el potencial del lenguaje. Por ejemplo, podemos extender las clases Java de Jess, añadir nuevas funcionalidades o utilizar objetos de Java de los que Jess carece y luego invocarlos desde Jess. En este trabajo se utiliza el lenguaje de Jess extendido con Java por la necesidad de añadir ciertas funciones, y la creación de nuevas clases de Java en Jess para su integración con edinn® M2.

A partir del análisis del dominio visto en la sección 2.1, se obtienen unos conceptos que deben ser organizados y estructurados de forma intuitiva para facilitar el desarrollo. Para ello se debe realizar un análisis exhaustivo sobre el dominio del sistema. A nivel de lenguaje podremos definir los hechos en patrones mediante plantillas con `deftemplate` y se agruparán los diferentes tipos de reglas en módulos con `defmodule`. También será necesario definir funciones según las necesidadas de las reglas mediante `deffunction`.

4.1.1. Hechos

Inicialmente deben definirse las estructuras para los hechos mediante la función `deftemplate`. Esta función es similar a la declaración de una clase en Java, pero con las variables del objeto de la clase siendo slots. Un slot es un espacio reservado para una información específica. Partiendo del dominio, se pueden evaluar las distintas categorías como candidatos. Además, necesitaremos una estructura adicional para representar las recomendaciones que aporta el sistema. Las estructuras a definir son:

`monitor`— Contiene los parámetros de configuración de la monitorización del sistema

`process`— Identifica a un proceso en concreto y sus parámetros

`status`— Representa el conjunto de estados disponibles para un proceso y sus parámetros

`result`— Identifica los productos o resultados para los procesos en función del estado

`order`— Representa las órdenes de producción para los procesos

`problem`— Es la salida del programa, que realizará una recomendación para solucionar la incidencia si existiese

Tras esto, se estudian los parámetros representativos y necesarios que componen cada una de estas estructuras para definirlas con el comando `deftemplate`. A continuación se muestra algún ejemplo de ello para facilitar su comprensión.

Para la categoría de configuración del monitor, los parámetros están definidos por diferentes tiempos de muestreo y volcado de datos, umbrales de tiempos, señales de reset, señal de control, y contadores en referencia a un proceso concreto. A continuación se muestra la definición de `monitor`. Para que las operaciones numéricas sean seguras se define el valor por defecto en aquellos slots que es necesario.

```
(deftemplate monitor
    (slot id-process)
    (slot signal)
    (slot th-low (default 0))
    (slot th-high (default 0))
    (slot chk-status (default 0))
    (slot dump (default 0))
    (slot reset-at (default 0))
    (slot counter-p (default 0))
    (slot counter-t (default 0)))
```

La categoría de proceso está formada por el identificador del proceso, su estado actual, orden actual, ratios de eficiencia, disponibilidad, velocidad, calidad, tiempos de ciclo y producción. De forma similar a la anterior categoría, definimos un proceso tal que así:

```
(deftemplate process
    (slot id-process)
    (slot status)
    (slot order)
    (slot oee (default 0))
    (slot availability (default 0))
    (slot speed (default 0))
    (slot quality (default 0))
    (slot cycle-t (default 0))
    (slot cycle-q (default 0)))
```

Las demás categorías se implementan de formas similar a los ejemplos anteriores, pero empleando los parámetros necesarios para cada caso. Con esta tarea completada, se tendría la base para rellenar la base de hehchos del sistema.

4.1.2. Reglas

El siguiente paso es la organización de las reglas y su diseño. Esta aplicación, tras obtener la información desde la base de datos, necesita seguir los pasos descritos a continuación para su correcto funcionamiento:

- Inicialización de la aplicación
- Analizar problemas en los parámetros de la monitorización
- Analizar problemas entre parámetros de la monitorización y los procesos
- Analizar problemas entre los productos o resultados y los parámetros de proceso según sus estados, y relacionarlos con los parámetros de monitorización
- Analizar problemas en las órdenes planificadas en función de los productos o resultados generados por los procesos

Para definir reglas se emplea el comando `defrule` de Jess. Un ejemplo sencillo para una regla que comprueba un valor umbral definido por el usuario en el monitor, en función de los parámetros del proceso, en un estado de tipo producción, es el siguiente:

```
(defrule th-low-too-high
  (process
    (id-process ?id cycle-t
     ?cyclet ?cycle-q ?cycleq))
  (monitor
    (id-process ?id counter-p
     ?counterp&:(< ?counterp ?cycleq)))
  (monitor (id-process ?id th-low
   ?thl&:(< ?thl ?cycle-t)))
  (process
   (id-process ?id cycle-t ?cyclet
    ?cycle-q ?cycleq))
  (status (id-process ?id type production))
=>
  (assert (problem th-low-too-high)))
```

A la hora de diseñar reglas se pueden encontrar partes en común que se repiten. Para ello, conviene utilizar el comando `deffunction` de Jess que permite declarar funciones. Por ejemplo, una tarea que se repite en diferentes reglas en este tipo de sistema es la comprobación de si el nivel de producción de un proceso es bueno. Para ello se necesitan un conjunto de parámetros comunes, que se pasarán a la función `is-good-production` y devolverá un resultado. Este cálculo es algo extenso y complejo, por lo que no tendría sentido reescribirlo cada vez que se utiliza.

```
(deffunction is-good-production (
  ?oee ?cyclet ?cycleq
  ?ratioprod ?counterp ?countert)
  (if (< ?oee ?ratioprod) then
    ...
    (return (> ?ratioprod (/ ?cycleq ?cyclet)))
  else (if (< (/ ?cycleq ?cyclet) ?ratioprod) then
    ...
    (return (< ?rdisp ?ratioprod)))
  else (if (<  (/ ?counterp ?countert)) then
    ...
    (return (< ?ratioprod ?rcycle)))
  else (if (>  (/ ?counterp ?countert)) then
    ...
    (return (< ?rcycle ?oee)))
```

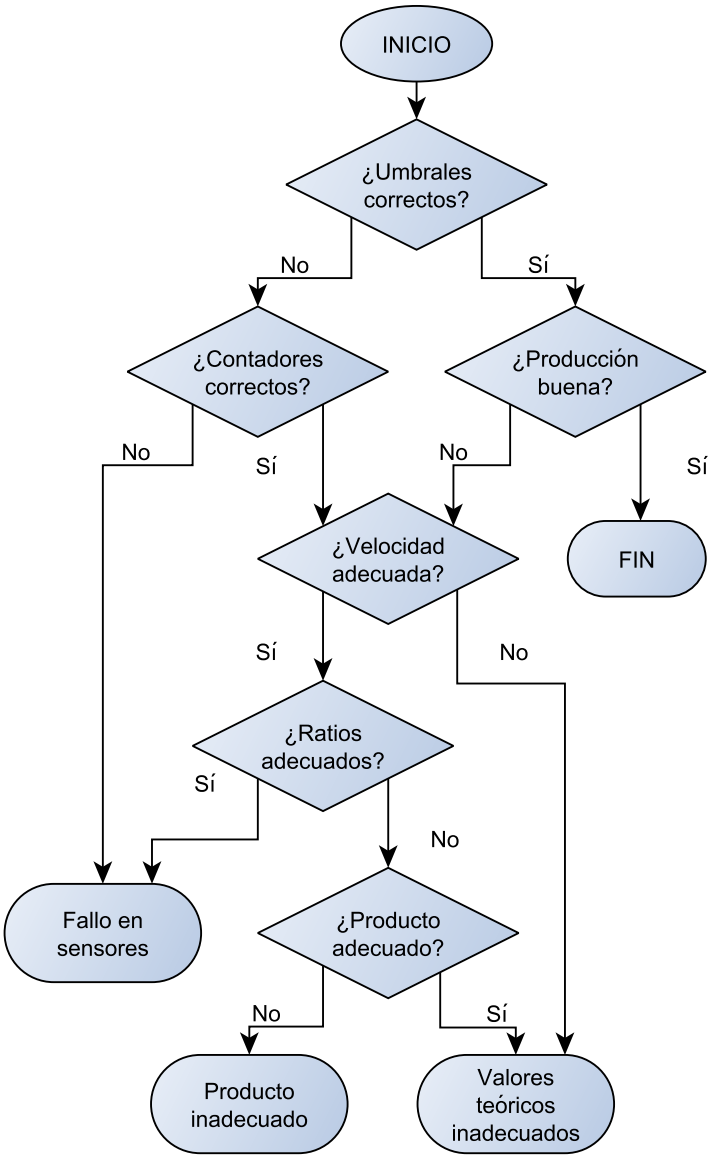

Figura 4.1: Ejemplo de una secuencia de inferencia

Con el análisis de toda esta información se deduce que las reglas pueden agruparse en función de las relación que tienen con los otros elementos del sistema. Estas agrupaciones dividen el problema general en problemas más sencillos en los que no se tendrá en cuenta relaciones de niveles superiores. Con esta idea se definen diferentes módulos que agrupen conjuntos de reglas que afecten a un elemento en concreto, siguiendo la lógica aportada por el experto. Lo que se consigue con este planteamiento es que se asemeje a la estructura del esquema que el experto aplica y que además permite

construir la aplicación de forma incremental, aumentado gradualmente la complejidad al incluir más reglas en cada uno de los módulos. Por tanto se decide desarrollar el sistema paso a paso mediante prototipos, en función de las relaciones entre elementos, tal y como se describe en la sección 4.2. La figura 4.1 representa un ejemplo de un proceso de inferencia extraído del experto, al que se le ha aplicado una capa de abstracción para que resulte más sencillo de visualizar.

Por otra parte, se decide utilizar *forward-chaining* para la inferencia, ya que no se parte de una hipótesis que se quiera contrastar contra los datos. Al contrario, se intentan hallar todos los posibles problemas a partir del conocimiento de los datos que estén disponibles, tal y como actuaría un experto.

4.2. Desarrollo

Para el desarrollo del sistema experto se establecieron tres prototipos objetivo que implementan de forma gradual el conocimiento transmitido por el experto. Para la definición de los dominios de estos prototipos se estudiaron las relaciones presentadas en la sección 2.1. Se pretende con ello medir de forma continua los avances del sistema y la detección de errores en fases determinadas. Debe tenerse en cuenta que estos prototipos al obviar alguna relación con elementos dependientes de un nivel superior necesitarán implementar nuevas reglas en función de éstos. En otras palabras, un prototipo no asegura la finalización de reglas para los conceptos del dominio que alberga, pues posteriormente pueden darse dependencias bidireccionales. Aunque esto pueda parecer confuso, si se tiene una buena estructuración del problema no es complicado. Para esta aplicación es más sencillo comenzar a modelar reglas sobre un par de elementos definidos por completo que para todo el conjunto de elementos si estos no se ven implicados en el proceso de inferencia. Esta estrategia modular será beneficiosa para el desarrollo, pues no es costoso añadir nuevas reglas a un módulo ya testeado o modificarlas con nuevas condiciones en función de las necesidades.

Los tres prototipos que se establecieron son:

Primer prototipo: Un prototipo en el que se analizarán las relaciones entre los parámetros de configuración, los procesos y sus estados

Segundo prototipo: Un prototipo que asocia productos a los procesos con cierto tipo de producción

Tercer prototipo: Un prototipo similar al sistema experto final que se desea, que establece relaciones entre las órdenes de producción, productos y procesos, estableciendo nuevas reglas y relaciones entre estos elementos

El desarrollo mediante módulos en Jess nos permite agrupar conceptos similares, con reglas propias, y que facilite la comprensión del programa y su escalabilidad.

4.2.1. Primer prototipo

En este primer prototipo implementado se contemplan las relaciones fundamentales entre los elementos básicos del dominio del sistema: procesos, estados y parámetros de configuración del monitor. Con ello se obtiene una primera versión básica del sistema final que se desea alcanzar y que servirá para evaluar su funcionalidad. Su fase de desarrollo requiere un tiempo de significativo. Esto es debido a que, a pesar de la reducción del dominio en el que se aplica, define las bases sobre las que se ampliará el sistema. Desarrollar correctamente esta parte del sistema facilitará las posteriores ampliaciones del dominio.

Este protipo está compuesto por:

- Un módulo para los estados y sus parámetros
- Un módulo para los procesos y sus parámetros
- Un módulo principal para los parámetros de configuración del monitor

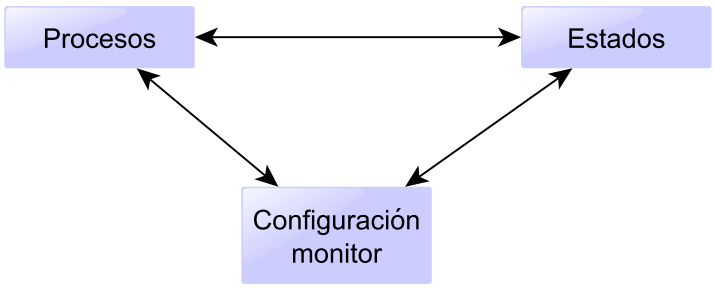

Figura 4.2: Relaciones entre elementos en el primer prototipo

Se definen los siguientes patrones:

- Patrón de estados
- Patrón de procesos
- Patrón de parámetros de monitorización

El prototipo implementa las siguientes reglas:

- Reglas propias de estados
- Reglas propias de procesos
- Reglas propias de configuración de la monitorización
- Reglas entre estados y monitorización
- Reglas entre procesos y monitorización
- Reglas entre estados y procesos

Se consiguen detectar fallos a nivel de:

- Estados de procesos
- Ciclos de tiempo y producción de procesos
- Ratios de eficiencia, velocidad, calidad y disponibilidad de procesos
- Valores umbrales de funcionamiento en función del estado y su producción
- Control de contadores

Entre otras funciones, este prototipo permite:

- Reconocer estados inadecuados de un proceso en función de la producción
- Reconocer una configuración errónea de parámetros propios de un de proceso
- Reconocer una configuración errónea de monitorización de un proceso en función de su producción
- Evaluar los ratios de un proceso y decidir si el nivel de producción real es el adecuado
- Detectar comportamientos extraños debidos a fallos en los contadores de un proceso u otro agente externo

4.2.2. Segundo prototipo

El segundo prototipo desarrollado añade los productos al dominio del sistema. La introducción de este elemento implica redefinir ciertas reglas del prototipo anterior para adaptarlas a la relación entre los productos, los estados y los procesos. Se recuerda que un proceso produce unos resultados o productos en un estado concreto.

Este protipo está compuesto por:

- Un módulo para los productos y sus parámetros
- Un módulo para los estados y sus parámetros
- Un módulo para los procesos y sus parámetros
- Un módulo principal para los parámetros de configuración del monitor

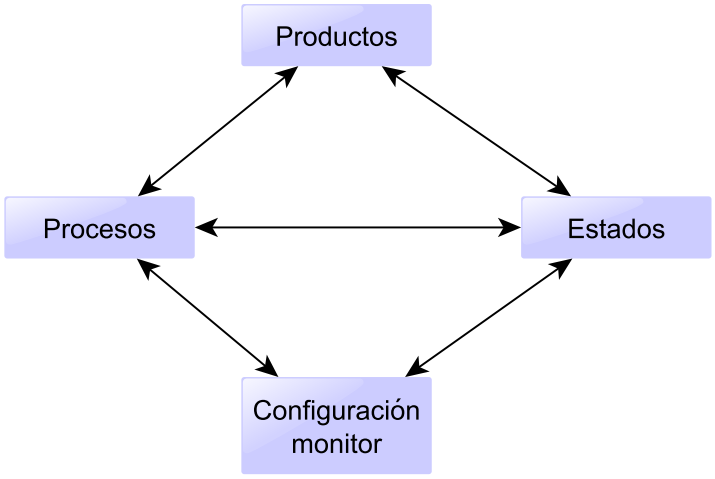

Figura 4.3: Relaciones entre elementos en el segundo prototipo

Se definen los siguientes patrones:

- Patrón de productos
- Patrón de estados
- Patrón de procesos
- Patrón de parámetros de monitorización

El prototipo implementa las siguientes reglas:

- Reglas propias de productos
- Reglas propias de estados
- Reglas propias de procesos
- Reglas propias de configuración de la monitorización
- Reglas entre estados y monitorización
- Reglas entre procesos y monitorización

- Reglas entre estados y procesos
- Reglas entre estados y productos
- Reglas entre procesos y productos

Se consiguen detectar fallos a nivel de:

- Parámetros de un producto
- Estados de procesos
- Ciclos de tiempo y producción de procesos
- Ratios de eficiencia, velocidad, calidad y disponibilidad de procesos
- Valores umbrales de funcionamiento en función del estado y su producción
- Control de contadores

Entre otras funciones, este prototipo permite:

- Reconocer productos y asociarlos a un estado
- Identificar productos en proceso con estados no deseados
- Detectar productos con parámetros que no se correspondan con lo definido en un proceso
- Reconocer estados inadecuados de un proceso en función de la producción
- Reconocer una configuración errónea de parámetros propios de un de proceso
- Reconocer una configuración errónea de monitorización de un proceso en función de su producción
- Evaluar los ratios de un proceso y decidir si el nivel de producción real es el adecuado
- Detectar comportamientos extraños debidos a fallos en los contadores de un proceso u otro agente externo

4.2.3. Tercer prototipo

El tercer prototipo agrupa todos los elementos del dominio del sistema definido. El nuevo elemento introducido son las órdenes. Éstas se asocian con productos y procesos, e inderectamente con los estados. Como ocurría en el prototipo anterior, será necesario ampliar las reglas entre los elementos ya definidos para ajustarlas a las órdenes.

Este protipo está compuesto por:

- Un módulo para las órdenes y sus parámetros
- Un módulo para los productos y sus parámetros
- Un módulo para los estados y sus parámetros
- Un módulo para los procesos y sus parámetros
- Un módulo principal para los parámetros de configuración del monitor

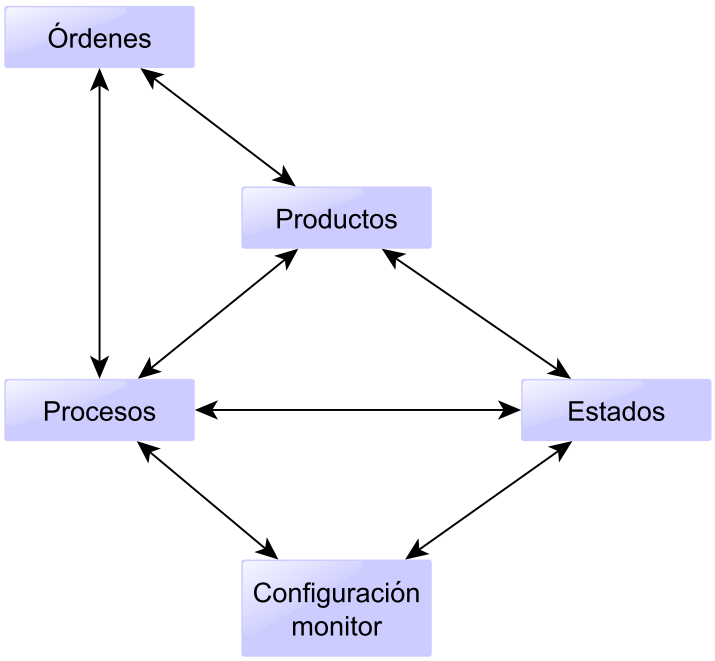

Figura 4.4: Relaciones entre elementos en el tercer prototipo

Se definen los siguientes patrones:

- Patrón de órdenes
- Patrón de productos

- Patrón de estados
- Patrón de procesos
- Patrón de parámetros de monitorización

El prototipo implementa las siguientes reglas:

- Reglas propias de órdenes
- Reglas propias de productos
- Reglas propias de estados
- Reglas propias de procesos
- Reglas propias de configuración de la monitorización
- Reglas entre estados y monitorización
- Reglas entre procesos y monitorización
- Reglas entre estados y procesos
- Reglas entre estados y productos
- Reglas entre procesos y productos
- Reglas entre órdenes y productos
- Reglas entre órdenes y procesos

Se consiguen detectar fallos a nivel de:

- Parámetros de una orden
- Parámetros de una orden
- Parámetros de un producto
- Estados de procesos
- Ciclos de tiempo y producción de procesos
- Ratios de eficiencia, velocidad, calidad y disponibilidad de procesos
- Valores umbrales de funcionamiento en función del estado y su producción
- Control de contadores

Entre otras funciones, este prototipo permite:

- Controlar que las órdenes contengan productos adecuados a los procesos
- Controlar los tiempos definidos para las órdenes
- Reconocer productos y asociarlos a un estado
- Identificar productos en proceso con estados no deseados
- Detectar productos con parámetros que no se correspondan con lo definido en un proceso
- Reconocer estados inadecuados de un proceso en función de la producción
- Reconocer una configuración errónea de parámetros propios de un de proceso
- Reconocer una configuración errónea de monitorización de un proceso en función de su producción
- Evaluar los ratios de un proceso y decidir si el nivel de producción real es el adecuado
- Detectar comportamientos extraños debidos a fallos en los contadores de un proceso u otro agente externo

4.3. Integración

Uno de los retos del desarrollo de este sistema experto era su integración con la herramienta edinn® M2. Ésta se instala en los equipos del cliente y básicamente existen dos modalidades, la versión de terminal de planta y la versión de informes. La versión de terminal de planta está orienatada a los operarios y supervisores de áreas. La versión de informes está orientada a personal del departamento de producción. Este último tipo de usuario es el que analiza la información proporcionada por el sistema para mejorar los procesos, tiempos, ratios, etc. Además, configuran el sistema de monitorización ya que son los que cuentan con los conocimientos necesarios sobre el área de producción. Por tanto, este será el perfil de usuario al que le resulte de más interés el sistema experto, pues le ayudará a detectar fallos de configuración cuando se realice cualquier modificación y le ahorrará tiempo de análisis de un problema de este tipo.

La herramienta edinn® M2 almacena los datos de los clientes en servidores seguros donde se realiza el análisis de los datos. Este análisis se inicia

bajo demanda de un servicio que invoca tareas en el servidor y se comunica con el cliente. Para añadir el sistema experto desarrollado a la aplicación será necesario crear la comunicación entre este servicio y el sistema experto. Como se comentó en el capítulo 2, Jess nos facilita la tarea de integración debido a que puede invocarse desde Java. Los datos con los que se alimenta el sistema experto provienen de la base de datos del cliente. Sin embargo, no accede directamente a ellos, sino que los recibe por el canal de comunicación establecido con el servicio. Esta decisión se debe a peculiaridades del tratamiento de los datos y reutilización de clases que no aportan interés a este trabajo. En la figura 4.5 se representa un esquema general de las comunicaciones descritas entre el cliente, la aplicación edinn® M2 y el sistema experto desarrollado.

A lo largo de este trabajo no se ha nombrado la interfaz de usuario. Esto se debe a que se utilizará la propia interfaz de edinn® M2 ya que sólo se necesita mostrar una serie de avisos y unos links que dirijan al usuario al problema. No es un sistema experto en el que el usuario va introduciendo datos que el sistema demanda. Es un sistema de diagnóstico del funcionamiento correcto de la producción real frente a los parámetros de producción y monitorización del sistema que el usuario ha definido.

El sistema experto será un módulo opcional de edinn® M2, y podrá ser ejecutado de forma continua o bajo demanda. Si se desea de forma continua, una tarea periódica en el servidor invocará al sistema experto que demandará el conjunto de datos que sean necesarios. Si fuese bajo demanda, la tarea se lanzará una vez en función del tiempo de ciclo configurado en el sistema. Existe una limitación a la hora de pedir al sistema experto un análisis de los datos: no se puede evaluar el correcto funcionamiento y configuración hasta que no pase al menos un ciclo de producción. Esto quiere decir que, si solicitamos un análisis del sistema en un momento concreto, deberemos esperar a que pase un ciclo de producción para que existan datos de producción, y por tanto se puedan contrastar contra los ratios esperados, parámetros y demás variables. Este caso se nombra debido a que en algunos clientes existen procesos con ciclos de producción muy grandes, ya que el producto puede necesitar de varias horas para producirse. El otro extremo son clientes con gran volumen de producción. Por ejemplo, una proceso puede llegar a producir una pieza cada tres segundos. Si se lanza el sistema experto cada tres segundos es posible que el rendimiento del servidor se vea afectado. La solución es establecer un periodo para la ejecución del sistema experto que abarque a aquellos procesos que se deseen comprobar.

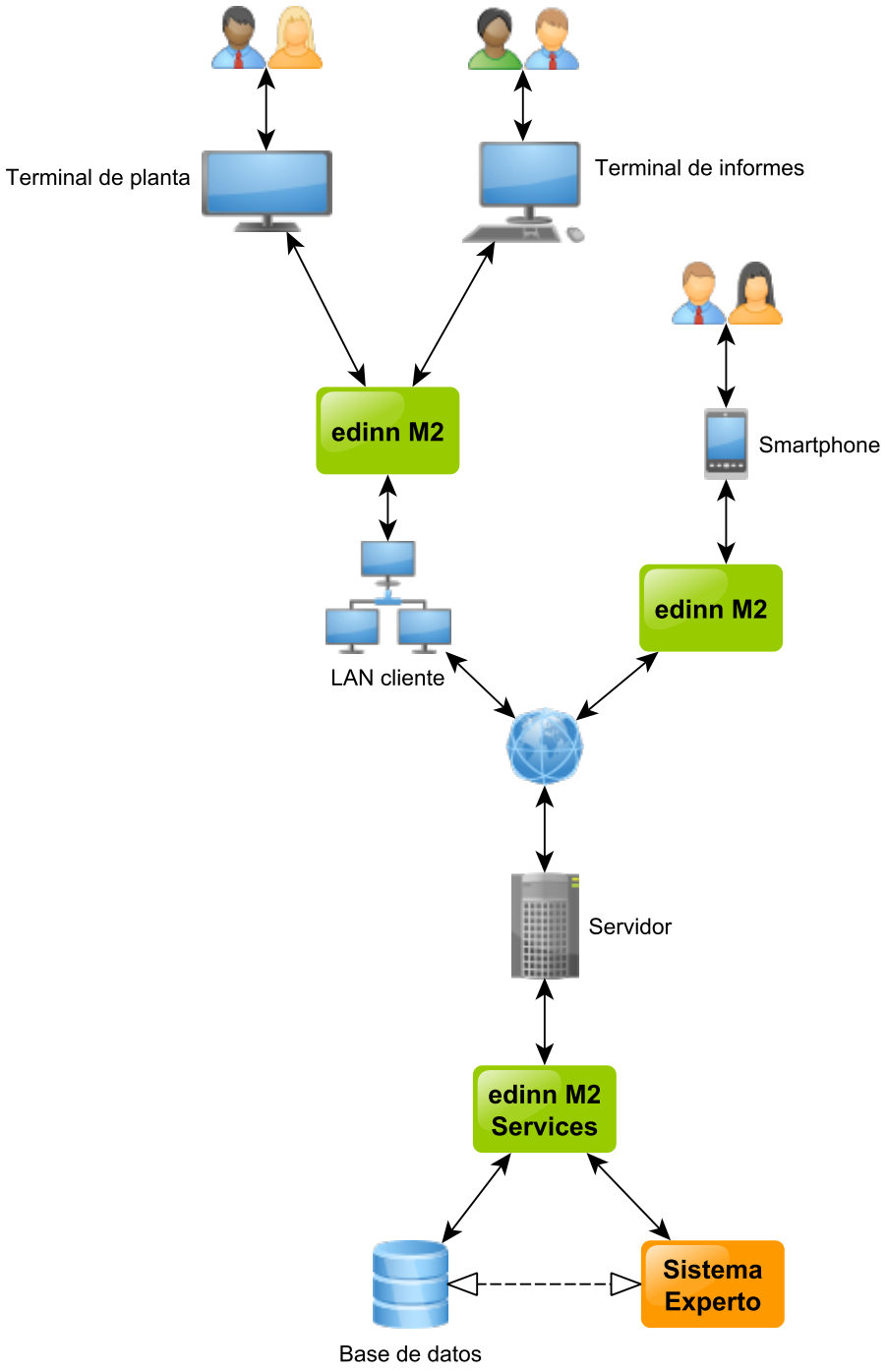

Figura 4.5: Integración del sistema experto en la herramienta

Capítulo 5

Evaluación

La simplicidad es requisito
previo para la fiabilidad.

Edsger Dijkstra

Resumen: En este capítulo se describe la metodología empleada para la evaluación del sistema experto indicando cada una de sus fases y objetivos. Posteriormente se comentan los resultados obtenidos tras este análisis y se valora el sistema en función de ellos.

5.1. Metodología

Para evaluar correctamente un sistema experto se requiere aplicar una metodología concreta que difiere de la típica evaluación de software, ya que la forma en que se aborda el problema y las expectativas del sistema son muy diferentes.

El objetivo de las evaluaciones es asegurar que el sistema experto ofrezca una respuesta correcta y de la forma adecuada a un problema de su dominio. Para ello es necesario:

- Asegurar la calidad del producto desarrollado
- Asegurar su utilización en dominios críticos
- Asegurar su aceptabilidad en la rutina diaria

El proceso de evaluación del sistema puede entenderse como una fase general de validación, donde se valoran aspectos del sistema como su utilidad, robustez, velocidad, eficiencia, posibilidades de ampliación o manejo entre

otros. La evaluación está formada principalmente por dos fases: la verificación y la validación.

La verificación es una comprobación de que el sistema es consistente, no tiene errores y cumple con las especificaciones iniciales. Durante el desarrollo de los prototipos será necesario comprobar por tanto la estructuración del conocimiento para garantizar que el sistema se modela tal como se especificó y está libre de errores. Este análisis puede hacerse en función o no del dominio dependiendo de la fase que se desee verificar. También es necesario comprobar que las reglas definidas son íntegras, exactas y evitan redundancias.

La fase de validación es más compleja ya que depende del criterio con el que se evalúe el sistema y su intención de uso. Se comprueba que la salida del sistema sea correcta y se cumplen las necesidades y requisitos del usuario. Demostrar que un sistema experto es correcto es una tarea crítica, pues un sistema con errores puede provocar fallos que resulten costosos y no cumplirá con las expectativas. Para una correcta validación se debe definir el nivel o calidad de las respuestas que el sistema genere frente a las dadas por un experto. En base a ello se puede definir un rango de tolerancia del nivel de respuestas del sistema, así como la fiabilidad esperada que implicará el nivel de confianza en el sistema en base a su credibilidad (Robert M. O'Keefe, 1993).

La validación del sistema experto requiere los siguientes elementos para obtener una respuesta cuantitativa y cualitativa de los resultados:

- Casos de prueba
- Expertos
- Sistema experto a evaluar
- Interpretación de los casos de prueba
- Validación de las interpretaciones de los casos de prueba mediante un criterio de evaluación concreto

Los casos de prueba para la validación del sistema deben ser de un número y representatividad suficente para asegurar el correcto funcionamiento del sistema frente a problemas reales. Es más importante cubrir las casuísticas que disponer de un gran número de casos. Los casos de prueba se han de presentar al sistema experto a evaluar y al conjunto de expertos para que, independientemente, sean interpretados. Para poder analizar los resultados es necesario haber establecido un determinado criterio. Este criterio no sólo debe tener en cuenta si el resultado es bueno, sino que debe poder medir cómo de bueno es, su nivel de detalle, su utilidad, velocidad de respuesta, etc. Una vez establecido el criterio, se contrastan las respuestas de los expertos con

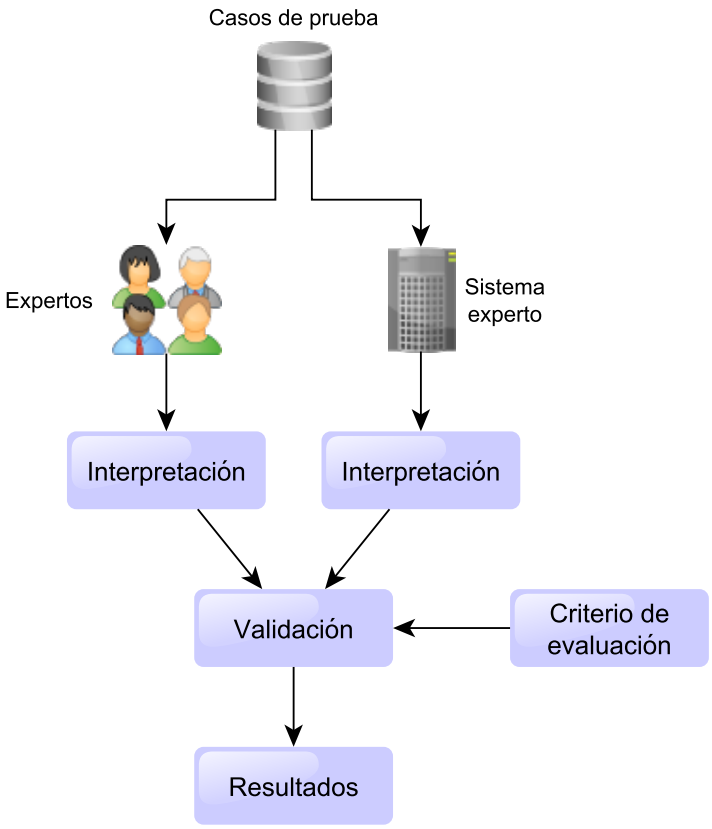

Figura 5.1: Metodología de validación del sistema experto

las del sistema experto. En base al criterio elegido se validan los resultados generando unos índices de acuerdo en base a diferentes técnicas cuantitativas de validación, que representan cómo de bueno es el resultado proporcionado por el sistema experto. En la figura 5.1 puede verse una representación de la metodología descrita.

Por último, se deberá evaluar para este sistema concreto la integración y rendimiento en conjunto con la herramienta edinn® M2. Este dato es fundamental, pues al tratarse de una tarea que realiza el servidor para cada cliente puede ser muy exigente y estresar el funcionamiento del sistema, provocando efectos no deseados sobre otras partes del sistema en el que se encuentra embebido.

5.2. Resultados

Siguiendo esta metodología para la evaluación del sistema experto, tras finalizar la fase de implementación de cada prototipo propuesto se procede a la verificación y validación.

Ya que cada prototipo cuenta con un dominio específico, tal como se vió en el capítulo 4, las exigencias del criterio varían acorde a esto. Además de lo acertada que sea la respuesta ofrecida por el sistema, se valora el detalle de ésta, así como su utilidad de cara al perfil de usuario que dispondrá de esta herramienta. Estos datos se cuantificaron y sirvieron de baremo para dar el visto bueno a los prototipos. Si no se alcanzaban las especificaciones definidas se revisaba de nuevo la estructuración del conocimiento y las reglas descritas. Tras la detección de fallos o mejoras se implementaba el nuevo modelo y se sometía una vez más a la evaluación.

Los casos de prueba empleados para la evaluación del sistema fueron definidos por los expertos que aportaron el conocimiento con el que se modeló el sistema experto. Estos casos recogen una gran variedad de casos específicos para cada prototipo. Generalmente, los casos de prueba más cercanos a los límites del dominio son los que dan una visión del alcance real del sistema evaluado, y ayudan a fortalecer estos casos extremo. En alguna ocasión, aunque el conocimiento estuviera satisfactoriamente implementado, era necesario dar prioridad a ciertas reglas sobre otras para obtener la respuesta esperada.

Los tres prototipos propuestos finales aportaron respuestas satisfactorias a diferentes niveles de experto. Una de las carencias que quizás más se repitió y costó más de definir correctamente es el detalle de la respuesta. Una respuesta poco específica, aunque sea correcta, puede resultar de muy poca utilidad y frustrará al usuario. La tolerancia de este tipo de respuestas se fue reduciendo gradualmente según se avanzaba en la implementación hacia el modelo final completo.

A nivel de integración en la infraestructura y su rendimiento preocupa la cantidad de memoria necesaria para la ejecución del sistema experto. Aunque esto es un dato que varía según el conjunto de datos del cliente, en general el consumo de memoria es notable. Esto se justifica con el uso que hace Jess del algoritmo *Rete* para la coincidencia de patrones como se nombró en el capítulo 2. Se sacrifica memoria a favor de una mayor velocidad de ejecución. Lanzar esta tarea en clientes con gran volumen de datos, alta producción y ciclos de producción cortos, puede llegar a provocar falta de memoria si Jess consume más de lo esperado, pues la comparte con servicios de edinn® M2. Aunque esta problemática no es del todo grave ya que depende del hardware, sí que requiere de un estudio independiente en el que se puedan medir las necesidades de ambos sistemas trabajando simultaneamente para diferentes

clientes en el mismo servidor.

La velocidad de respuesta del sistema es bastante buena, incluso con clientes grandes. Aquí el algoritmo *Rete* parece mostrar sus beneficios en contraposición a los problemas de memoria descritos antes. La velocidad total del sistema integrado depende, además del sistema experto, de la base de datos que contiene la información del cliente. En general, la velocidad en conjunto es buena en todos los casos, pero una vez más depende del volumen de datos con los que trabaja el cliente.

Los resultados recogidos del sistema a lo largo de este tiempo de desarrollo en numerosas pruebas a nivel interno hace ver que que se trata de un sistema fiable, robusto y con unas respuestas adecuadas. Su comportamiento es satisfactorio aunque se desean aplicar algunas mejoras para dar respuestas más concisas que sean de utilidad al usuario final. Actualmente este sistema sigue bajo desarrollo y añadiendo mejoras. Las pruebas que se realizan con datos reales de clientes sirven para detectar características a modificar en las siguientes versiones.

Capítulo 6

Conclusiones y trabajo futuro

Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer.

Alan Turing

Resumen: Para finalizar este trabajo, en este capítulo se encuentran las principales conclusiones relativas al diseño, desarrollo, evaluación y resultados obtenidos. En base a ello, se dan unas ideas del trabajo futuro que se desea realizar sobre la aplicación y las expectativas.

6.1. Conclusiones

En líneas generales, este trabajo ha servido para desarrollar una aplicación de tipo sistema experto para una empresa real con un problema concreto, y con fines comerciales. Este sistema experto integrado en la herramienta con la que trabaja edinn Global proporciona un sistema único frente a la competencia directa, pues no se conoce una aplicación similar. La repercusión de ello se nota en el interés que genera en el cliente tener un sistema capaz de avisar de los fallos cuando éstos aún son prácticamente imperceptibles. Esto ahorra mucho tiempo de trabajo frente a encontrarse con un problema en un estado avanzado que probablemente requerirá la modificación de una gran cantidad de datos que ya han sido generados. Si se necesitase soporte por parte de edinn Global para solucionar una incidencia, el usuario ya tendría cierto conocimiento acerca del problema como sus causas y posibles soluciones. Con un sistema experto de calidad, esto facilita enormemente la tarea de edinn Global en la resolución de incidencias, pues ya tienen un primer diagnóstico con el que evaluar la situación. Se debe destacar también que

el usuario, tras la experiencia con este sistema, será mejor usuario ya que aprenderá de sus errores.

Como se ha visto, el sistema experto desarrollado se encuentra integrado en una aplicación ya existente. Esto ha requerido una inversión de tiempo importante para garantizar las expectativas, en función del tiempo y los recursos. Por tanto, se ha visto la necesidad de medir el rendimiento de un sistema en este tipo de proyectos. Esta fase de integración ha sido de vital importancia para obtener un sistema final que fuese viable y funcionase correctamente en lo clientes. Para ello se evaluaron las diferentes herramientas para el desarrollo de sistemas expertos en base a las necesidades concretas de este proyecto.

La repercusión de esta herramienta en la empresa es muy positiva. A día de hoy se tiene asentado un sistema sobre el que seguir desarrollando para mejorar la herramienta final. Esto es algo que edinn Global buscaba desde hace un tiempo, cuando por primera vez se plantearon que disponer de un sistema de este tipo sería beneficioso para los usuarios y para ellos mismos. Resulta evidente tras este trabajo que estos sistemas tienen una aplicación real en diversos ámbitos y, sin duda, resultan útiles. Su desarrollo no es una tarea sencilla. La comunicación con el experto, su colaboración y dedicación es un factor crítico para el éxito del proyecto. El posterior modelado del conocimiento debe ser muy exigente para garantizar la robustez del sistema, por lo que en la fase de desarrollo se realizan gran cantidad de evaluaciones de forma continua.

Por último, la impresión personal tras esta experiencia es que, aunque la utilidad de sistemas expertos para aplicaciones complejas es innegable y cuando se proponen como solución son valorados, actualmente no se emplean en la medida que cabría esperar. La forma en que se trabaja para desarrollar un sistema de este tipo es muy distinta a la tradicional. Es un enfoque diferente de transmisión de conocimiento desde el experto a la máquina que requiere de una metodología concreta para tener éxito en un plazo de tiempo razonable. Probablemente la causa de que los sistemas expertos no se encuentren extendidos a más ámbitos es que no se cuente con el tiempo suficiente y con la colaboración de las personas adecuadas, pues sin duda son proyectos costosos y que pueden alargarse bastante en el tiempo.

6.2. Trabajo futuro

Este proyecto ha asentado las bases de algo que se deseaba realizar hace tiempo. Los prototipos desarrollados han servido para evaluar su utilidad y detectar las debilidades del sistema. Éstos no dejan de ser versiones reducidas de un dominio muy extenso y complejo como es el de la producción industrial. Esto no quiere decir que el sistema desarrollado hasta ahora no sea aplicable,

si no que se desea ser más ambicioso.

Algunas de las mejoras principales que se desean implementar son:

- Capacidad de análisis por áreas (agrupación de procesos definida por el usuario)
- Añadir al dominio los usarios implicados en cada proceso
- Detección de fallos en integración con sistemas ERP
- Potenciar el sistema a nivel de planta
- Mejorar el módulo de planificación
- Potenciar la utilización de lógica difusa

Además de estas mejoras que requieren un tiempo de desarrollo considerable, se quiere incidir en la sencillez, claridad y utilidad de las respuestas que ofrece el sistema. Creemos que es un valor fundamental para medir la calidad del sistema y que realmente es lo que provocará en el usuario una sensación de confianza y fiabilidad en la herramienta.

Bibliografía

B. Wielinga, J. B. Kads: A model approach to knowledge engineering. *Esprit*, vol. 1991(5), 1991.

Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

Forgy, C. L. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, vol. 1982(19), 1982.

Friedman-Hill, E. *Jess in Action: Java Rule-based Systems*. Manning Publications Co., 2003.

Friedman-Hill, E. Online jess documentation. 2009. Disponible en `http://www.jessrules.com/jess/docs/index.shtml` (último acceso, Septiembre, 2013).

Melle, W. V. A domain-independent production rule system for consultation programs. *International Joint Conference on Artificial Intelligence*, 1979.

R. Davis, E. H. S., B. G. Buchanan. Production systems as a representation for a knowledge-based consultation program. *Artificial Intelligence*, vol. 1977(8), 1977.

Robert M. O'Keefe, D. E. O. Expert system verification and validation: a survey and tutorial. *Kluwer Academic Publishers*, vol. 1993(7), 1993.

Rudolph, G. Some guidelines for deciding whether to use a rules engine. 2008. Disponible en `http://www.jessrules.com/jess/guidelines.shtml` (último acceso, Septiembre, 2013).

Yasser Abdelhamid, A. R., Hesham Hassan. A proposed methodology for expert system engineering. 2005.