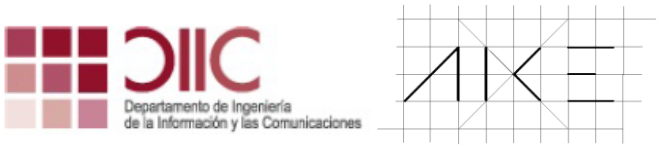
Trabajo fin de grado

Soporte a la decisión clínica en la farmacovigilancia

Directores: Manuel Campos – José Manuel Juárez

Autor: Tony Wang Chen – X1968208Y

2013-2014



FACULTAD DE INFORMÁTICA – UNIVERSIDAD DE MURCIA

AGRADECIMIENTOS

A mis directores Pepe y Manolo por estar encima de mí y animándome cuando no veía por dónde coger el proyecto. Si no fuera por ellos, no habría sido posible entregarlo.

A mi novia Rocío, por hacer esfuerzos por entender de qué va todo esto. Por estar ahí a todas horas, por ayudar en medida de lo posible y simplemente, por soportarme.

A mi gran amigo Rubén, compañero de jergas técnicas. Gracias por compartir los momentos de depresión conmigo. Por ayudarme con tu sabiduría y por soportar mis llantos.

A mi familia por hacer posible estos años de estudios. Por trabajar tan duro para que no me falte nada y por haberme tenido.

A la Facultad de Informática de la Universidad de Murcia, y a todos sus profesores por preparar de tan buena manera a todos los alumnos que salen. Gracias por hacer posible todo.

RESUMEN

Según la Organización Mundial de la Salud, la farmacovigilancia “trata de recoger, vigilar, investigar y evaluar la información sobre los efectos de los medicamentos, productos biológicos, plantas medicinales y medicinas tradicionales, con el objetivo de identificar información sobre nuevas reacciones adversas y prevenir los daños en los pacientes”. Tanto a nivel nacional como Europeo, existe una legislación que regula el uso de medicamentos, y que establece como aspecto clave la farmacovigilancia. En España, éste es uno de los objetivos principales de la Agencia Española de Medicamentos y Productos Sanitarios, dependientes de Ministerio de Sanidad, Servicios Sociales e Igualdad.

Los actores participantes en los procesos de recogida de información sobre efectos adversos son tanto los ciudadanos como los profesionales clínicos.

En un hospital, el número de fármacos administrados a los pacientes es muy alto, y, por tanto, son necesarios muchos recursos para hacer una revisión eficiente de los tratamientos. Por un lado, el volumen de información que los clínicos deben tener a su disposición es muy alto considerando el gran número de fármacos existentes, y la cantidad de información proporcionada por cada uno de ellos. Por otro lado, esta información tiene que ser además contrastada con los datos particulares de cada paciente antes de aplicar un tratamiento.

Un sistema de soporte a la decisión que asista a los clínicos y farmacéuticos aumentaría la eficiencia de la revisión de tratamientos, y, como consecuencia, la seguridad del paciente. Así, el objetivo de este Trabajo Fin de Grado es el **desarrollo de un sistema de soporte a la decisión clínica en farmacovigilancia**.

Para llevarlo a cabo, se han definido varios subobjetivos. En primer lugar, se modelará todo el conocimiento sobre los medicamentos, haciendo énfasis en los tipos de alertas existentes (por ejemplo, alergias o interacciones medicamento-medicamento) y sus mecanismos de disparo. Para ello, se recurrirá a las normativas nacionales sobre medicamentos y al catálogo de medicamentos aprobados por la Agencia Española de Medicamentos y Productos Sanitarios, y se estudiarán los estándares de representación.

En segundo lugar se desarrollará el sistema de soporte a la decisión clínica. Se estudiará el uso que se ha hecho de técnicas inteligentes para farmacovigilancia, así como del conocimiento requerido para su diseño. Inicialmente, este proyecto usará una aproximación basada en reglas, que han sido tradicionalmente uno de los sistemas más utilizados, incluso desde el inicio del desarrollo de sistemas basados en conocimiento como es *Mycin* de Edward Shortliffe. También, se estudiarán las plataformas disponibles para el desarrollo de soporte a la decisión clínica, prestando interés a las plataformas abiertas que den soporte a reglas.

En último lugar, se realizará una evaluación inicial en un dominio real. Se analizarán, entre otros, las limitaciones de la representación de conocimiento elegida para la información de los medicamentos y para el conocimiento dado por las guías clínicas de tratamiento. Otro parámetro de análisis será la capacidad de razonamiento del sistema, y su eficiencia en el dominio seleccionado.

EXTENDED ABSTRACT

Pharmacovigilance, according to the World Health Organization (WHO), strives for collecting, monitoring, investigating and evaluating the information regarding the effects of medicinal products, medical herbs and traditional medicines in order to identify potential adverse reactions and prevent harms in patients. There exist a legislation at both, European and national level, which regulates the use of medications, being the **pharmacovigilance** a cornerstone for it. This legislation framework is defined in Spain by the *Agencia Española de Medicamentos y Productos Sanitarios* who, in turn, directly depends on the *Ministerio de Sanidad, Servicios Sociales e Igualdad*.

The actors involved in the process of gathering information about the adverse effects are both, the citizens and the clinicians.

In a hospital, the number of drugs given to patients is very high, and therefore, many resources are required to ensure an efficient review of the medical treatments. On the one hand, the huge amount of medications available together with the endless data on them, produces a very large volume of information at the disposal of the clinicians. On the other hand, all this information must be evaluated against the specific circumstances of the patient before the treatment is given.

A decision support system that may help to decouple clinicians and pharmacists would improve the treatments' review efficiency and, in turn, the patient's safety. Therefore the main goal of this Final Degree Dissertation is the development of **a Clinical Decision Support System** (CDSS) for use in **pharmacovigilance**. To this end, the following sub-objectives are achieved:

1. The potential adverse effects in the field of microbiological surveillance will be studied.
2. Alternatives in the design and development of alerts in a Clinical Information System (CIS) will be covered.
3. Alerts that may be obtained in **pharmacovigilance** will be modelled.
4. **OpenCDS** platform will be studied.
5. A prototype will be implemented in the **OpenCDS** platform.

Before starting with it, a review of the state of the art will be carried out. Besides, because of the very nature of this dissertation, it is important to introduce and define those relevant concepts that will be used along the whole work such as clinical standards (HL7v2, OMG, vMR, …), clinical guidelines or the importance of the Information Technologies (IT) in the medicine (especially the pros and cons of the CDSS).

After that, a theoretical study of the potential tools and alternatives that could be used to tackle this problem will be accomplished. The alternatives are the following: 1) develop an application from scratch implemented directly in PL/SQL or in Java, 2) develop the application from scratch but using a ruled-based system such as *Drools*, *Jess* or *Clips* and 3) develop the application using the ***OpenCDS*** platform. The reason why the first alternative is excluded is twofold: firstly because of its lack of scalability and secondly because it provides no benefit over ruled-based systems (e.g. it does not decouple the logic interface from the application logic). Regarding the other two options, a thorough study on how they work, their main characteristics and a list of pros and cons is provided.

Drools will be the main tool under analysis for designing and implementing the rules. This is all the more important if the following is taken into account: the ***OpenCDS*** platform uses Drools as its core engine (also known as *interface engine*) and to provide the results.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE GRÁFICOS

1. INTRODUCCIÓN

Hoy en día existe un problema importante en las instituciones clínicas que es el uso ineficiente de los antibióticos en los tratamientos de los pacientes. Aunque en el día a día la gente no sea del todo consciente y haga un mal uso de ellos, desde el punto clínico y dentro de una institución médica como puede ser la UCI de un hospital, se debe evitar esto para evitar entre otros, los siguientes problemas [1]:

- Fracaso terapéutico.
- Desarrollo de resistencias bacterianas.
- Enmascaramiento de procesos infecciosos.
- Cronificación: la falta de erradicación de un número suficiente de bacterias dará lugar a la persistencia de algunas que mantienen su grado de patogenicidad sin ocasionar manifestaciones agudas.
- Recidiva[1]: las cepas supervivientes, sean resistentes o sensibles, inician una nueva proliferación que provocará una recaída o una reinfección.
- Efectos adversos debidos a la acción del medicamento (independientes de que sea o no eficaz). La toxicidad de algunos antibióticos es potencialmente grave y su aparición es inaceptable si el paciente no necesitaba el fármaco.

Para un manejo adecuado de los antibióticos se requieren conocimientos de 1) farmacología de los diversos antibióticos (es un catálogo muy grande), 2) indicaciones de primer orden y alternativas en las diversas enfermedades infecciosas y 3) efectos adversos y contraindicaciones. Como se ha indicado, el catálogo de antibióticos existentes así como los efectos adversos de estos, es muy amplio. También existen numerosas enfermedades infecciosas. Todo ello viene relacionado y recogido en las **guías clínicas**, las cuales orientan a un médico a la hora de tratar a un paciente. Esta carga de trabajo podría reducirse si se consiguiera informatizar el proceso mediante algún tipo de **soporte a la toma de decisiones clínicas**. De esta manera se consiguen ventajas como resultados más óptimos en el uso de antibióticos, lo cual provoca una reducción de costes aparte de evitar los problemas anteriormente mencionados.

Para abordar este problema con sistemas informáticos, se han de hacer uso de **estándares clínicos**, de manera que entre distintas instituciones pueda existir una comunicación eficaz, además de que permiten una comunicación humana con la máquina. Por tanto es conveniente comenzar explicando los conceptos clínicos relevantes en el mundo médico y farmacéutico (**guías clínicas** y **estándares clínicos**) así como en el de las tecnologías (**estándares clínicos en la informática** y **sistemas de soporte a la decisión clínica**). También se mencionarán las ventajas que conlleva tener sistemas informáticos en el campo de la salud así como las limitaciones que pudieran haber.

1.1. ESTÁNDARES CLÍNICOS

Según [2] un estándar propone reglas que permitan que la información sea procesada y compartida de manera uniforme y consistente. Cualquier persona está familiarizada con un estándar como por ejemplo, las reglas que sigue un determinado idioma. En el caso de estándares clínicos, se trasladan las mismas afecciones del estándar a este ámbito, resolviendo

[1] Según la RAE, reaparición de una enfermedad poco después de periodo de convalecencia.

cuestiones como por ejemplo, qué información de los pacientes se deben comunicar al médico y cómo o cuál es el formato que seguirá esta información (por ejemplo, si el género varón se representa con un código o simplemente una "M").

Los estándares en medicina son esenciales para permitir de una manera precisa y fiable trabajar en el dominio médico. Existen estándares de vocabulario médico (CIE, LOINC, CIAP,…), otros de comunicación de la información médica (como HL7v.2, vMR, OMG…) y otros de interoperabilidad entre sistemas (por ejemplo UN13606).

Gracias a esto se hace posible el intercambio electrónico de información de pacientes además de que se garantiza integridad, legibilidad de la información y evasión de errores entre los distintos sistemas que intervienen en los procesos de telediagnósis, teleconsulta, procesamiento, transmisión y almacenamiento de registros médicos [3]. No obstante existen una serie de barreras [2] a la hora de establecer los estándares. En resumen son los siguientes:

- Los estándares son creados por cooperativas, organizaciones, personas individuales, etc. De esta manera, pueden existir diferentes visiones sobre el mismo problema además de falta de recursos o experiencia.
- Los estándares de datos sólo pueden crear si son codificados. Hay estructuras de datos que son fáciles de codificar, pero la información médica no lo es, ya que es compleja, profunda y muy amplia.
- Una vez que el estándar es desarrollado, debe de ser adoptado por las diferentes instituciones de salud. Un estándar bueno puede que nunca llegue a colocarse en el mercado si no se dispone de una buena iniciativa y venta del producto.

1.2. LA INFORMÁTICA EN LA MEDICINA

El uso de las nuevas tecnologías en la salud, ha cobrado particular importancia este último siglo, y es principalmente, entre otras, por las siguientes razones [4]:

- Las tecnologías de la información están actualmente muy presentes en la sociedad, y está avanzando muy rápido y aumentando más la presencia en todos los campos.
- El volumen de conocimiento médico y de la salud está incrementando a un ritmo tan rápido que es imposible mantenerlo actualizado ni tener la esperanza de almacenarlo, organizarlo y recuperar la información existente sin el uso de nuevas tecnologías y técnicas de procesamiento.
- Se obtienen beneficios económicos con el uso de la informática como soporte en los servicios de salud.
- La calidad de la atención médica se ve incrementada.
- Se consigue mayor escalabilidad.

La **Informática Médica** se basa en cuatro pilares los cuales buscan el desarrollo de un nuevo paradigma para el manejo de la información, en lo relativo al campo de la salud que son [5]:

- Producir estructuras para representar datos y conocimiento.
- Desarrollar métodos para una correcta y ordenada adquisición y representación de los datos.
- Manejar el cambio entre los procesos y las personas involucradas para optimizar el uso de la información.
- Integrar la información de diferentes fuentes.

Por tanto se puede definir como **Informática Médica** [4][5] al estudio, intervención e implementación de estructuras y algoritmos destinados a mejorar la comunicación, el conocimiento y el manejo de la información médica con objetivo de recolectar datos, conocimientos para aplicarlos en la toma de decisiones cuando sea necesario. La posición que ocupa la **Informática Médica** en el campo de la salud es muy amplio, y está presente en sistemas de **estándares médicos**, sistemas orientados puramente a la informatización médica (sistemas de información clínico, sistemas para gestión farmacéutica o historia clínica electrónica), sistemas para ingreso de órdenes médicas, sistemas médicos expertos, sistemas de administración o **sistemas de alertas de apoyo médico** (sistemas de soporte a la decisión clínica) entre otros.

1.2.1. SISTEMAS DE SOPORTE A LA DECISIÓN CLÍNICA

Para mejorar la calidad del servicio médico en Estados Unidos, se hicieron esfuerzos para incrementar la práctica de medicina basada en evidencias a través de sistemas de soporte a la decisión clínica (CDSS) [6]. Los CDS permiten a los clínicos, pacientes y cuidadores que tengan conocimientos médicos e información específica de pacientes (historial clínico, datos del paciente…) tomar las decisiones adecuadas mejorar el cuidado del paciente. Esta información del paciente se asocia a una base de conocimiento clínica y las evaluaciones específicas o recomendaciones del mismo se le comunica a dicha base cuando sea idóneo durante el tratamiento del enfermo. Algunos diagnósticos usando CDS incluyen cuestionarios y plantillas para obtener la información del paciente, recordatorios, alertas y otros grupos de cuestiones que proporcionan sugerencias y soporte. Aunque los CDS pueden ser diseñados para ser usados por médicos, pacientes y cuidadores informales, la idea principal es centrarse en el uso que le dan los médicos para mejorar el proceso que siguen a la hora de tomar todas las decisiones clínicas.

El uso de CDSS ofrece muchos beneficios potenciales. Una importante, es que los diagnósticos con CDS pueden incrementar el uso de conocimiento médico basado en evidencias y reducir variaciones innecesarias en la práctica clínica. En el proceso de desarrollar e implementar sistemas CDS se pueden **establecer estándares en la estructura de la base de conocimiento** que vayan acorde a las guías clínicas ya existentes. Los CDSS también pueden asistir a los clínicos dándole soporte a la hora de manejar la información de manera que se mejoren las habilidades en la toma de decisiones, reduzcan el trabajo mental de los mismos y mejore el flujo de trabajo clínico. Si un CDSS está bien diseñado e implementado, tiene potencial para mejorar la calidad en el cuidado de la salud además de incrementar la eficiencia y reducir los costes médicos. Otros beneficios que proporcionan [7] podrían ser que evitaría algunos errores que podría cometer un humano y podría advertir de decisiones mal tomadas. Todo esto llevaría a tener un paciente más satisfecho, finalidad de todo personal sanitario.

Algunos de los sistemas expertos que hay o ha habido son *Mycin* (el cual diagnosticaba enfermedades infecciosas de la sangre), *CADUCEUS* o *Clinical Rules*. Estos sistemas, funcionan en cierto como un CDSS. Más adelante se hará hincapié en los más significativos.

1.2.2. LIMITACIONES

A pesar de que los CDSS son prometedores, [6] existen numerosas limitaciones y barreras para su desarrollo e implementación [8]. Estas diez barreras se pueden dividir en tres categorías grandes:

A. Mejorar la eficiencia de las intervenciones de los CDS.

B. Crear nuevas intervenciones CDS.
C. Expandir el conocimiento y las intervenciones CDS existentes.

Dentro de cada una de las categorías, las barreras más importantes que existen vienen enumeradas a continuación:

A. Mejorar la eficiencia de las intervenciones de los CDS:

1. **Necesidad de mejorar la interfaz que comunica a los humanos con los ordenadores**. Se necesita una interfaz fiable, que haga al médico que esté al cargo creer en las decisiones que está recomendándole el CDSS que tome. Por tanto estas decisiones han de estar fundamentadas sobre una base sólida (indicando la traza que ha seguido hasta llegar a la conclusión, por ejemplo). Además la interfaz debe de ser entendible por el médico para que sepa en todo momento qué hacer. También ha de ser posible que en caso de que haya algún error humano, el CDSS sea capaz de avisarlo.
2. **Resumir la información de los pacientes**. Es imposible crear un CDSS que procese todos los datos de un paciente. Hay mucha información que a nivel informático sería muy difícil e incluso imposible de modelar. Por tanto se han de resumir estos datos en aquellos que sean los más importantes y pertinentes para cada situación. La barrera de un CDSS sería que automáticamente y de manera inteligente fuera capaz de recolectar estos datos necesarios en lugar de coger todos. De esta manera se evitaría trabajo inútil lo que al final conlleva un ahorro en tiempo y costes.
3. **Priorizar y filtrar las recomendaciones al usuario**. Un sistema decisor debería poder automáticamente priorizar las recomendaciones en función de los datos del paciente, porcentaje de mortandad, preferencias del paciente, costes individuales y de organización, efectividad del tratamiento, tolerancia del paciente a poder recibir dicho tratamiento y muchísimos otros factores médicos y humanos. La primera barrera en este caso está en que el sistema debe barajar apropiadamente todas estas competencias y dar una solución que las tenga en cuenta. La segunda barrera es que el sistema ordene por prioridad las recomendaciones que vaya obteniendo, es decir, que sea capaz de establecer prioridades.
4. **Combinación de recomendaciones con pacientes que tengan comorbilidad**[2]. Las guías médicas actuales ignoran el hecho de que la mayoría de los pacientes ancianos tienen comorbilidad que deben ser tenidas en cuenta por su equipo médico. La barrera aquí está en crear mecanismos que identifiquen y eliminen redundancia, contraindicaciones, discordancias potenciales y guías clínicas que se excluyan entre ellas. Una de las principales razones por las que no hay tanto uso en las guías clínicas es precisamente porque no contempla el hecho de personas con varias enfermedades. Por ejemplo una persona diabética que a su vez tenga otra enfermedad, a la hora de tratar la diabetes una guía clínica puede sugerir un tratamiento que sea contraproducente con la segunda enfermedad. Para abordar este problema se necesitaría aproximaciones lógicas, combinatorias y semánticas, lo cual podría llegar a hacerse inviable.
5. **Uso de información libre de formato como entrada en el CDSS**. Se necesitan métodos que de una información de paciente en cualquier formato que tenga la institución se consiga una entrada que sea interpretable por el CDSS. Además esta

[2] Presencia de uno o más trastornos o enfermedades en un paciente, además de la enfermedad o trastorno primario. Fuente: http://es.wikipedia.org/wiki/Comorbilidad

información ha de ser acorde a la realidad y estar bien estructurada. Para ello lo lógico sería hacer uso de un estándar médico.

Crear nuevas intervenciones CDS:

6. **Priorizar los distintos contenidos en el desarrollo e implementación de CDSS**. Para el desarrollo e implementación de estos sistemas se necesita de personal cualificado (sobre todo en el ámbito médico) lo que conlleva un elevado precio y mucho tiempo invertido. Decidir qué tipo de sistema decisor implementar (por ejemplo, para enfermedades crónicas, infecciosas,…) primero está basado en muchos factores como el coste del sistema, la disponibilidad de datos precisos en el ámbito, la dificultad de implementación, la aceptación por parte de clínicos y pacientes, etc.
7. **Necesidad de una gran minería de datos para crear un nuevo CDSS**. Hoy en día existen muchísimas guías clínicas que están a la espera de ser implementadas en un CDSS pero que están basadas en conocimiento clínico que aun no ha sido sintetizado en su totalidad. Se necesita desarrollar y probar nuevos algoritmos y técnicas que permita a los investigadores trabajar con esta cantidad de datos clínicos y posteriormente integrarlas en los sistemas.

Expandir el conocimiento y las intervenciones CDSS existentes:

8. **Extender las buenas prácticas en el diseño, implementación y desarrollo de CDSS**. Algunas organizaciones de salud han tenido buenas experiencias ya con sistemas CDSS. Cuando se estudian dichas organizaciones hay factores comunes que han hecho que el uso de los sistemas haya sido satisfactorio, pero esta información no está textualmente disponible en ningún lado para que las organizaciones que estén a expensas de desarrollar algún CDSS puedan acceder e informarse. Se necesita por tanto, elaborar métodos para describir, evaluar, recolectar, catalogar, sintetizar y extender una guía de buenas prácticas en el diseño de CDSS. Establecer esto es esencial para la investigación y el desarrollo de propósitos, para refinar y acelerar el desarrollo de nuevas intervenciones y resaltar los pros y los contras para mejorar las propias bases de conocimientos de los sistemas ya desarrollados.
9. **Creación una arquitectura para compartir módulos y servicios CDSS**. El propósito de esto es crear una serie de estándares para externalizar los CDSS de tal manera que cualquier historial clínico de un paciente (EHR: *Electronic Health Record*) pueda utilizar el servicio además de que puedan coger el mismo y mejorarlo sin tener que reestructurarlo entero, sino simplemente adaptándolo.
10. **Crear repositorios CDSS accesibles a través de Internet**. La barrera está en crear uno o más CDS con módulos de conocimiento, repositorios de alta calidad y basado en evidencias. Estos servicios deberán de ser fácilmente descargados, mantenidos y con posibilidad de modificaciones locales además de estar certificados por alguna entidad.

Muchas de estas barreras están en proceso de ser eliminadas, otras son posibles eliminarlas de cara al futuro y por último hay otras que costaría mucho trabajo superarlas y de momento no es viable.

1.3. GUÍAS CLÍNICAS

Según [9], las **Guías de Práctica Clínica** (GPC) son un conjunto de recomendaciones basadas en una revisión sistemática de la evidencia en la evaluación de los riesgos y beneficios de las diferentes alternativas, con el objetivo de optimizar la atención sanitaria a los pacientes. El desarrollo de una GPC viene motivado por razones como la existencia de un problema de salud importante con impacto en la morbimortalidad, aparición de técnicas o tratamientos novedosos, posibilidad de conseguir un cambio para mejorar resultados en la atención, etc. Además, se comprende de las siguientes fases [9]:

- **Elaboración**: A su vez, esta etapa se subdivide en las siguientes fases: 1) Delimitación del alcance de los objetivos, 2) Creación del grupo elaborador de la GPC, 3) Formulación de preguntas clínicas, 4) Búsqueda, evaluación y síntesis de la literatura, 5) Formulación de recomendaciones, 6) Revisión externa, 7) Edición de la GPC.
- **Actualización**: Proceso que pretende mantener la vigencia y la calidad de una GPC. El plazo para que una GPC se quede obsoleta es de 3 a 5 años y necesita ser actualizada.
- **Adaptación**: En lugar de elaborar una GPC nueva, se promueve la modificación y adaptación de una GPC desarrollada en un contexto cultural y organizativo diferente. De esta manera se evita elaboración y actualización de una GPC, que son procesos complejos que requieren esfuerzo, tiempo y disponibilidad de recursos.
- **Evaluación**: En esta etapa se valora la calidad de una GPC con el objetivo de que las personas que la usan puedan confiar en las recomendaciones a la hora de utilizarla en su práctica clínica diaria.
- **Implementación**: Proceso que tiene como objetivo trasladar las recomendaciones de una GPC a la práctica clínica. En la implementación hay que prestar atención al contexto donde se desarrolla, a las barreras y facilitadores y a la valoración de estrategias de intervención que pueden resultar más efectivas y eficientes a la hora de culminar la implementación de la GPC de forma exitosa. La implementación implica utilizar estrategias de comunicación efectivas para promover el cambio.

En la Ilustración 1 se puede ver la relación existente entre las diferentes fases de la vida de una GPC.

Existen además, una serie de ventajas y desventajas en el uso de GPC que vienen en enumeradas a continuación [10].

Ventajas:

1. Los autores de las guías clínicas han dedicado más tiempo para revisar las decisiones que sugieren que la mayoría de los clínicos que están involucrados en la atención de los enfermos.
2. Las mejores guías han sido escritas por personas con experiencia en el problema clínico que tratan.
3. Un grupo de expertos que tenga en cuenta varios casos comunes es capaz de evaluar mejor las evidencias que un grupo de especialistas que trabaje de manera aislada.
4. Permiten un equilibrio entre la práctica y la teoría, entre diferentes especialidades médicas, entre médicos y administradores, entre administradores y político, etc.
5. Las guías son interesantes tanto para médicos como para personal de otras áreas importantes por lo que muchas personas se preocupan seriamente por desarrollar buenas guías.

6. El hecho de que los profesionales de la salud asuman criterios comunes favorece la implantación de diagnósticos adecuados.
7. Ofrecen posibilidad de adaptar guías con experiencias nacionales a situaciones locales.

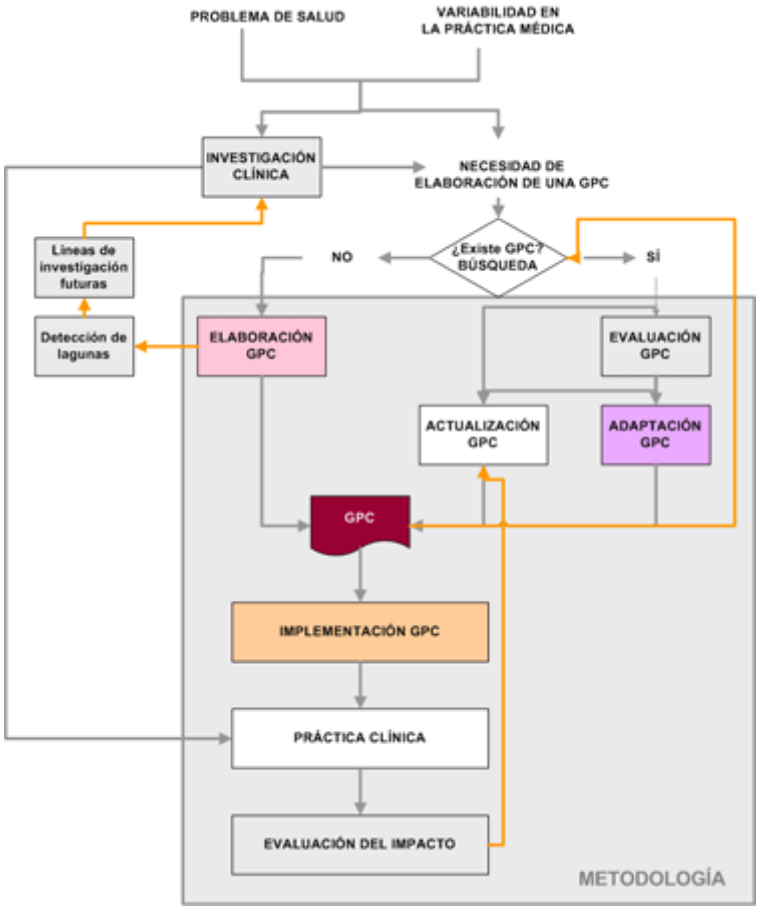


ILUSTRACIÓN 1: DIAGRAMA DE FASES DE UNA GUÍA CLÍNICA. OBTENIDA DE [9]

Desventajas:

1. Los propósitos de las guías tienden a ser vagos y los criterios para evaluar el progreso después de haberlas usados son imprecisos.
2. Los métodos para confeccionar guías son muy variados, no hay un marco estándar.
3. La atención que se le presta en la implementación y aplicación de las guías en escenarios prácticos no es la misma que se le da en el proceso de redacción.
4. No existe un compromiso por parte de los autores de las guías que evalúen el impacto de las guías en el comportamiento de profesionales, pacientes o costos de la atención médica.
5. Cuando se publica una guía, a menudo las mejores evidencias en las que se basan ya se han quedado obsoletas porque aparecen nuevos métodos de diagnóstico y tratamiento. Por esto se necesitan revisiones periódicas.
6. No existe un compromiso por parte de los médicos a la hora de usar las guías. Deben realizarse grandes esfuerzos para que haya una aplicación diaria del médico.
7. En ocasiones se emplean como referencia para evaluar malas prácticas profesionales, que fueron realizadas por personas que no tienen experiencia en la especialidad.

8. Carecen de características cruciales (claridad, especificidad, flexibilidad, validez,…) que afectan tanto en su aceptación por los médicos como en su impacto sobre la práctica clínica.

1.4. MOTIVACIÓN Y ENFOQUE DEL PROYECTO

Este proyecto viene motivado principalmente por la necesidad de un estándar clínico en los **sistemas de soporte a la decisión clínica**. El por qué de ello se menciona en las barreras de los CDSS. También, la idea es aprovechar al máximo los beneficios que aportan las nuevas tecnologías en la medicina. Implementando y desarrollando los CDSS bajo un estándar concreto, permitiría que también se aprovecharan los beneficios que aporta tener un estándar médico.

Hoy en día existen muchos estándares médicos pero no hay ninguno a la hora de diseñar un CDSS. Se prevé que de cara al futuro y debido al avance en las tecnologías y en el campo de la salud, esto sea algo necesario por lo que varias instituciones importantes ya están al tanto de ello y están desarrollando dicho estándar (plataforma ***OpenCDS***). Por tanto, la realización de este proyecto también viene motivada por lo novedoso que es, además de que deja las puertas abiertas para una futura investigación.

Otra motivación sería conseguir un sistema que sea capaz de funcionar de manera centralizada para tratar pacientes con cualquier tipo de enfermedad. Se quiere diseñar un sistema que sea independiente del tipo de conocimiento y que sea capaz de integrar nuevos módulos de conocimiento de manera sencilla. En resumen, se quiere un sistema que contenga de manera informatizada las diferentes GPC existentes y que sea capaz de integrar nuevas GPC.

Por último, el proyecto se centrará en un campo concreto: la farmacovigilancia. Debido a la cantidad de medicamentos, enfermedades y pacientes que hay, la necesidad de tener un CDSS de apoyo es inherente y es necesaria su implementación. Se espera que desarrollando un buen proyecto, el clínico encargado de la farmacovigilancia haga más uso del sistema informático.

Este proyecto se centra en alertas de apoyo al médico. En concreto, será en el ámbito de las enfermedades infecciosas. Se trata de diseñar un CDSS que ayude a tomar las decisiones a la hora de administrar antibióticos a pacientes con dichas enfermedades. Para ello se hará uso de:

1. Inteligencia artificial, más en concreto Sistemas Basados en Reglas (SBR).
   1.1. Las reglas se diseñarán en función a una **guía clínica.**
2. Estándares HL7v2, vMR, OMG y en concreto ***OpenCDS*** que es el encargado de unirlos como plataforma CDSS.

De esta manera se consigue un sistema de soporte a las decisiones clínicas estandarizado que puede usarse en cualquier institución. Cambiando las reglas del sistema, se podría orientar para una funcionalidad u otra. El prototipo final del proyecto sería un CDSS que muestre alertas de aviso en para que el médico pueda seguir correctamente una guía clínica, dándole soporte al clínico acerca de qué tratamiento debe de aplicarle al paciente.

2. OBJETIVOS

El principal objetivo de este trabajo final de grado será crear un sistema de soporte a la decisión clínica para la farmacovigilancia. Para ello se tendrán que cubrir una serie de subobjetivos que se nombran a continuación:

- Se estudiarán los tipos de efectos adversos en alertas de vigilancia microbiológica.
- Se estudiarán alternativas para el diseño y desarrollo de alertas en un sistema de información clínico.
- Se modelarán las alertas que se produzcan en farmacovigilancia.
- Se estudiará la plataforma ***OpenCDS***.
- Se implementará un prototipo en ***OpenCDS***.

Por otro lado, no formará parte de este proyecto las siguientes cuestiones:

- Evaluación clínica de la herramienta (será un prototipo que no estará validado por ningún experto en el campo).
- Modelado de todos los tipos de efectos adversos.
- Extensión de la plataforma ***OpenCDS***.
- Módulo de conexión con el SNS.

2.1. ESTADO DEL ARTE: SISTEMAS CON OBJETIVOS SIMILARES

En general los CDSS están transformando la forma en que se llevan a cabo las preinscripciones médicas en los diferentes ámbitos de la salud, y las instituciones sanitarias sienten la necesidad de adoptar incrementalmente herramientas de este tipo con el objetivo de mejorar los resultados clínicos [11]. Actualmente existen diferentes tipos de CDSS, entre los que se destacan aquellos que proveen soporte a las decisiones para el cuidado de pacientes, basados en guías clínicas.

Los CDSS han evolucionado rápidamente desde sus inicios hasta nuestros días, contando en la actualidad con diversas opciones que sirven de soporte a las decisiones en el ámbito sanitario. Estas son una valiosa herramienta de apoyo para el profesional de la salud y desde su surgimiento en la década de los 70 han ido evolucionando no solamente en lo que se refiere a las áreas médicas sino también respecto a los momentos en que son aplicados, la relación que éstos mantienen con el médico. Esta evolución mejora la calidad del soporte de quienes lo utilizan como herramienta de apoyo y también brinda la oportunidad de que sean utilizados en más de un área de salud. Además cada vez se permiten bases de conocimiento más amplias, con más claridad de respuestas así como la posibilidad de que los CDSS se integren como componentes a los sistemas de información de la salud.

Actualmente es poco habitual ver instituciones médicas que no dispongan de algún tipo de sistema de información que permita gestionar sus actos médicos y financieros. Cada vez son más las instituciones que ven en los HIS (*Health Information System*) el soporte necesario para la búsqueda de la calidad asistencial. Los CDSS no únicamente apoyan a la toma de decisiones para establecer un diagnóstico, protocolos clínicos para activar dicho diagnóstico, medicación y/o procedimientos a prescribirse a un paciente, sino que también se puede ver como un soporte a la prevención de futuras enfermedades. Al final todo esto conlleva a la posibilidad de construir

sistemas basados en conocimiento, así como la arquitectura de un modelo basado en ontologías que permite codificar guías clínicas y protocolos con el objetivo de generar recomendaciones específicas a un paciente y que pueda ser integrado a un HIS.

Para terminar, tras un una búsqueda con finalidad de pre estudio, se han encontrado una serie de sistemas que tienen un objetivo similar. Todos esos sistemas se tratan de CDSS que funcionan con una base de conocimiento en forma de reglas. Además, casi todos tienen su origen en el sistema experto *Mycin*. A continuación se enumeran y se describen las principales características de aquellos que han tenido más transcendencia en la historia.

Mycin

Desarrollado en la Universidad de Stanford, es un sistema experto que se basó en la experiencia previa adquirida en proyectos previos como *Dendral*[3] y *Mediphor* [12]. La función del sistema es diagnosticar enfermedades infecciosas en la sangre y determinar el tratamiento adecuado en un proceso compuesto en cuatro fases:

1. Decidir si la infección es significativa.
2. Determinar el/los organismo(s) implicado(s).
3. Seleccionar los fármacos que pueden ser apropiados.
4. Elegir el fármaco, o combinación de fármacos, más apropiada para el paciente.

El conjunto de reglas de *Mycin* (unas 500 en su última versión) estaban formadas por un antecedente (premisa) que consistía en una conjunción de una o más condiciones y un consecuente (acción). Las disyunciones se podían representar mediante múltiples reglas con la misma acción. La parte de acción podía suponer la ejecución de una o más acciones o la aserción de una conclusión.

Las decisiones que tomaba *Mycin* no sólo implican los datos relativos al paciente sino que también incluía información de laboratorio sobre los cultivos, organismos que han sido aislados y los fármacos que han sido administrados. Las reglas estaban organizadas de acuerdo con el tipo de contexto por el cual podían ser invocadas, por ejemplo grupos de reglas relacionadas con los fármacos administrados en la historia reciente del paciente, reglas aplicables para cualquier fármaco antimicrobiano aplicado, etc.

Tras realizar pruebas, el 65% de los tratamientos de *Mycin* fueron catalogados como aceptables por los evaluadores frente al 55.5% que obtuvo el personal sanitario. Con estos datos se demostró que el rendimiento de *Mycin* era ligeramente superior al de los miembros del departamento de enfermedades infecciosas del hospital de la Universidad de Stanford. Sin embargo, este estudio tenía ciertas limitaciones como el reducido número de casos o que los casos seleccionados del hospital sólo cubría un pequeño espectro de las posibles enfermedades.

A pesar de los buenos resultados, *Mycin* nunca fue utilizada de forma práctica en ningún hospital. Su base de conocimiento, a pesar de que era muy grande, sólo cubría una parte del dominio de las enfermedades infecciosas. Modelar el total de este conocimiento se podía hacer inviable para la época. Además su ejecución requería una potencia de cálculo que no estaba al alcance de la mayoría de los hospitales. Por último, otra razón de peso era por cuestiones éticas-morales como legales. En caso de que el sistema errara, ¿quién asumiría la culpa de esto? ¿Los

[3] Más información de *Drendral*: http://www.it.uc3m.es/jvillena/irc/practicas/estudios/DENDRAL.pdf

programadores, o el personal clínico a cargo? [13]. No obstante *Mycin* demostró la factibilidad de crear sistemas expertos basados en reglas causa-efecto y en la actualidad estos sistemas están cobrando más importancia.

INTERNIST/CADUCEUS

INTERNIST comenzó a desarrollarse a principios de los 70 por investigadores de la Universidad de Pittsburgh [14]. Su dominio se sitúa en el campo de la medicina interna. La idea inicial era desarrollar el sistema experto más elaborado que haya podido concebirse y tras la primera demostración en 1974 se utilizó para el análisis de cientos de problemas clínicos difíciles con notable éxito. Los propios autores del sistema, a su vez desarrollaron una versión más compleja aún llamada *CADUCEUS*.

El principal objetivo del proyecto *INTERNIST/CADUCEUS* era modelar la forma en que los clínicos razonan sobre el diagnóstico. Al abordar todo el campo de la medicina interna, no sólo está obligado a tratar gran número de enfermedades, sino también a considerar todas las combinaciones e interacciones posibles entre esas enfermedades. Este objetivo viene motivado por el excesivo número de las anteriores combinaciones y la imposibilidad de almacenarlo en una base de datos. Por ello, la única forma razonable de abordar el problema es un enfoque desde la IA, desarrollando modelos de las distintas enfermedades y haciendo que estos modelos interactúen.

INTERNIST funcionaba con una base de datos que incluía unas 500 enfermedades, cubriendo casi el 25% de la medicina interna. No obstante, aunque daba muy buenos resultados diagnosticando enfermedades, no agradaba a los médicos que hacían uso del sistema debido que perdía mucho tiempo considerando enfermedades que no venían al caso. Aunque el sistema terminaba por descartar las malas opciones y enfocarse en la correcta, tardaba bastante en llegar a esta solución.

CADUCEUS intentó eliminar estos problemas. Este usaba la misma base de datos que *INTERNIST* pero como se tuvo que reformular la base de conocimiento, al final causó que también la base de datos tuviera que reconstruirse desde 0 para incluir elementos que desde un principio no se contemplaban. De aquí que se haga hincapié en la importancia de tener una correcta representación inicial del conocimiento.

PUFF

Se construyó en 1979, está diseñado para la interpretación de medidas de pruebas respiratorias aplicadas a pacientes en un laboratorio de función pulmonar [14]. Las pruebas de función se utilizan para medir el volumen de los pulmones, la capacidad de éstos para llevar a oxigeno a la sangre y extraer el dióxido de carbono, etc. Un experto del ámbito puede interpretar esas medidas para diagnosticar la presencia y la gravedad de diversas enfermedades pulmonares. *PUFF* interpreta un conjunto de resultados de una prueba de función pulmonar y genera un documento escrito con la interpretación y el diagnóstico.

En resumen, diagnosticaba enfermedades obstructivas d las vías respiratorias utilizando el motor de inferencias de *Mycin* y una nueva base de conocimiento de unas 400 reglas. Las pruebas que se realizaron generaron informes correctos y sin anotaciones el 85% de los casos.

DXplain

Es un sistema de ayuda al diagnóstico que utiliza un conjunto de datos clínicos (signos, síntomas y resultados de pruebas de laboratorio) para producir una lista ordenada de probabilidad de diagnósticos que se pueden explicar, o estar asociados a esos datos [15]. Hace uso de 5000 diagnósticos asociados con 2000 síntomas diferentes. Para la decisión hace uso de la lógica Bayesiana.

En una prueba con 105 casos, se vio que en el 91% de los casos aparecía la dolencia entre los resultados, y en el 69% de los mismos, esta se encontraba entre las 20 primeras posiciones.

2.2. METODOLOGÍA Y HERRAMIENTAS

En esta sección se explicará de manera teórica las herramientas que se podrían haber usado para abordar el proyecto y se hará hincapié en aquella que finalmente se ha decidido usar. Se describirá las limitaciones, los estándares, se harán comparativas, etc. También se mencionarán sistemas que hayan ya existido que hicieran las mismas funciones, así como sus características.

2.2.1. HERRAMIENTAS

Para abordar este proyecto se barajaron las siguientes alternativas: 1) desarrollarlo desde 0 usando un lenguaje de programación como PL/SQL o Java, 2) desarrollarlo desde 0 pero en lugar de implementarlo puramente con programación, hacerlo integrando un sistema basado en reglas (SBR) como podría ser *Drools*, *Jess* o *Clips* y 3) utilizando el estándar ***OpenCDS***.

La primera opción se descartó desde un principio por razones obvias. En comparación con un SBR, el realizar una aplicación desde 0 con cualquier lenguaje de programación no aporta ninguna ventaja. Sin embargo con un SBR se podría conseguir lo siguiente:

- Representación de una forma más eficiente el conocimiento. Con reglas del tipo IF-ELSE se puede, en su mayoría, representar conocimiento de manera sencilla de cualquier ámbito. Además este conocimiento será más claro y entendible por parte de los expertos.
- Se independiza el conocimiento y los datos del resto de la aplicación. De esta manera se puede tener una interfaz común, y por debajo funcionar en dos ámbitos totalmente distintos. Con esto se consigue una modularidad y portabilidad. En cuanto a portabilidad, se refiere a que aplicaciones hechas en lenguajes distintos podrían llevar por debajo la misma base de conocimiento.
- Adaptabilidad: Con unos simples cambios en las reglas, se puede cambiar totalmente la funcionalidad de la aplicación. Además, la mayoría de SBR están implementados, y no habría que modelar reglas desde 0 sino adaptar las existentes.
- Robustez: Con un SBR se podría comprobar la traza que sigue el sistema hasta llegar a la solución. Esto, además de hacer el sistema más robusto, permitiría validar por parte del experto la solución inferida.

La principal desventaja de este diseño, sería la complejidad en algunos ámbitos a la hora de obtener las reglas que puedan representar ese conocimiento.

Respecto a la segunda opción, se realizará un análisis y se explicará en detalle, ya que aunque este proyecto se termina desarrollando con ***OpenCDS***, esta plataforma por debajo funciona con un sistema basado en reglas (*Drools*).

2.2.2. SISTEMA BASADO EN REGLAS

La aplicación se podría abordar perfectamente diseñando un sistema basado en reglas. Actualmente hay infinidad de herramientas para ello, como *Jess*, *Clips* o *Drools*. En este caso se analiza la herramienta *Drools* puesto que es la que utiliza ***OpenCDS***. Más adelante se explican las razones por las que esta opción finalmente se descarta también para el proyecto. Para explicar *Drools* es necesario conocer qué es un sistema basado en reglas, así como los componentes que tiene y los algoritmos que usa internamente para inferir soluciones.

2.2.2.1. ALGORITMO *RETE*

Creado por *Charles Forgy*, es un algoritmo diseñado para SBRs. El nombre proviene de **REdundancia TEmporal** y utiliza una red de nodos denominada **red RETE** [16]. El funcionamiento básico de *RETE* es el siguiente:

1. Realizar los cambios en la base de hechos.
2. Propagar los cambios por la red *RETE*.
3. Modificar el conjunto conflicto.

Surge porque en un SBR en cada ciclo equiparamos todas las reglas con todos los elementos de la base de hechos para formar el conjunto de conflicto y el SBR sería poco eficiente. *RETE* lo que introduce es un proceso de filtrados de reglas antes de generar el conjunto conflicto. Antes de explicar en más detalle *RETE,* se deben saber los aspectos más importantes de un SBR.

Un SBR se inspira en los sistemas de deducción en lógica proposicional o de primer orden. Se entienden las reglas como una sentencia del tipo:

```
IF condición THEN acción
```

Sus componentes son los siguientes:

- **Base de conocimiento**: Contiene las reglas que codifica todo el conocimiento (por norma general es estático y es el conocimiento que se ha ido adquiriendo a lo largo del tiempo). Una regla se disparará cuando se cumplan sus condiciones (antecedentes) y se ejecutará entonces una acción (consecuente).
- **Base de hechos**: Contiene propiamente los hechos ciertos a la hora de resolver el problema. Algunos de estos vendrán como datos de entrada y otros serán conclusiones inferidas (aplicándose las reglas de la base de conocimiento). Esta base es dinámica puesto que se irán añadiendo y eliminando hechos cuando se disparen las reglas.
- **Motor de inferencia**: Se encarga de seleccionar las reglas que se pueden disparar en un momento y las aplica siguiendo unas determinadas condiciones para obtener nueva información y actualizar las bases.
- **Conjunto conflicto**: Conjunto de reglas que pueden ser disparadas en un momento, con la base de hechos que haya.

Por lo general, el algoritmo de un SBR puede funcionar de dos maneras:

- **Encadenamiento hacia delante**: Se busca un conjunto de metas que se verifican a partir de un conjunto de hechos.
- **Encadenamiento hacia detrás**: Se determinan si se va cumpliendo la meta con el conjunto de hechos.

Los pasos que sigue un SBR, por norma general se pueden resumir en los siguientes:

1. Con una base de conocimiento y una base de de hechos inicial, y mientras no se tenga la meta en la base de hechos y se sigan teniendo reglas en el conjunto conflicto hacer:
    a. Meter en el conjunto conflicto aquellas reglas que pueden ser disparadas.
    b. Resolver el conflicto y seleccionar una regla.
    c. Aplicar la regla, sacarla del conjunto conflicto y actualizar los datos con el consecuente de la regla.
    d. Comprobar si está la solución en la base de hechos. Si es así, terminar.

Una vez explicado esto, se sabe que se puede situar el algoritmo *RETE* en el primer paso del algoritmo general de un SBR. Generar un buen conjunto conflicto hará que el SBR sea mucho más eficiente. Como se ha empezado diciendo, *RETE* tiene un filtro que permite que en cada ciclo no sea necesario equiparar todas las reglas con todos los elementos de la base de hechos y como normalmente los cambios que implica la ejecución de una regla son pocos, hace que el filtro sea eficiente. La comunicación a la red *RETE* de los cambios producidos en la base de hechos se hacen mediante la utilización de **testigos**, los cuales están formados por dos componentes: un signo (+ o -) para indicar si el elemento se ha borrado o añadido a la base de hechos y los elementos que son añadidos o borrados. Por ejemplo:

`−(A C)` que indica que se han eliminado los hechos `A` y `C` de la base.

La red *RETE* se genera al principio de la ejecución del *SBR* y sigue los siguientes pasos:

1. Se consideran los elementos que son constantes y los que, siendo variables, no aparecen en dos o más elementos de condición distintos en la misma regla.
2. Se consideran el resto de los elementos.

Un ejemplo de red *RETE* con la siguiente regla:

`R1: (A B) ∧ (C ?x D ?x)` → …

se representaría como en la Ilustración 2. Se ve que no se hace referencia a las variables ya que se pueden equiparar con cualquier valor que se encuentre en su posición. En este particular caso sólo se ha de comprobar que el segundo componente y el cuarto componente de la segunda parte de la regla coincidan. En caso de que alguna entrada no atraviese la red completa hasta un nodo terminal, quedaría registrada su posición por si en el siguiente ciclo cambiara. La red *RETE* se puede considerar como una función que traslada los cambios de la base de hechos a cambios en el conjunto conflicto. Además se pueden encontrar los siguientes tipos de nodo:

- **Nodo de una entrada** (nodo raíz, nodos de componentes variables o constantes, nodos terminales y nodos de comprobación de variables).
- **Nodos de dos entradas** (se sitúan siempre después de los nodos de una entrada, a excepción de los nodos terminales y pueden ser nodos de unión de elementos condición, nodos de unión de elementos condición con variables iguales de una misma regla y nodos de unión de los elementos condición negados con los restantes elementos condición de una regla)

En última instancia se tendrá un árbol con todas las reglas de nuestro sistema en forma de red *RETE* y cada vez que se introduzcan o eliminen nuevos hechos se modificará el conjunto conflicto (cuando todos los hechos llegan a algún nodo terminal, que son las reglas). No se va a

entrar más en detalle, así que a continuación y para terminar se detallan las características más importantes:

- Como en cada ciclo los cambios en la base de hechos suelen ser pocos aunque los contenidos sean muchos, es muy ineficiente equiparar en cada ciclo toda la base de hechos. En una red *RETE* se almacenan las equiparaciones que se producen en cada ciclo con lo que estas pueden ser reutilizadas en el ciclo siguiente y de esta forma el coste computacional depende de la velocidad de cambio de la base de hechos (que suele ser baja) y no se su tamaño (que suele ser grande.
- Aprovecha la **similitud estructural**. Las condiciones de las reglas tienen muchos patrones similares aunque no idénticos. La red *RETE* agrupa los patrones comunes con el fin de evitar repetir las equiparaciones idénticas y elimina las operaciones redundantes construyendo un único nodo por cada operación distinta y uniendo dicho nodo con aquellos que necesitan dicho resultado.

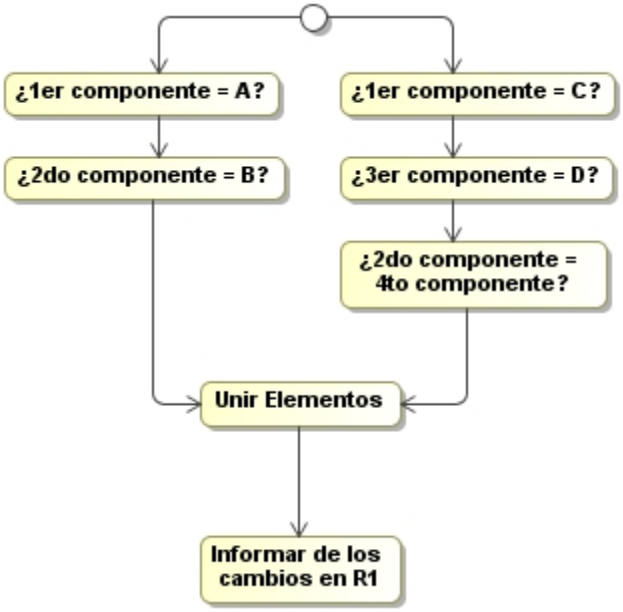


ILUSTRACIÓN 2: EJEMPLO DE RED *RETE*. OBTENIDA DE [16]

2.2.2.2. *DROOLS*

Una vez se sabe que la plataforma ***OpenCDS*** funciona por debajo con *Drools* como sistema de inferencia de reglas, se va a hacer un estudio del mismo para ver la eficiencia, escalabilidad, etc. Este estudio se realizará con las versiones 5 y 6, puesto que de una a otra cambian bastantes aspectos importantes como por ejemplo, el algoritmo usado en el motor de inferencias. Además, se tiene que tener en cuenta que la plataforma ***OpenCDS*** usa la versión 5 de *Drools* y se espera que para el futuro se adapte a la versión 6, que en teoría debería de proporcionar mejoras al mismo.

2.2.2.2.1. ALGORITMO *RETEOO* VS ALGORITMO *PHREAK*

Como se ha mencionado anteriormente, el aspecto que más diferencia a ambas versiones es su algoritmo del motor de inferencias. Mientras que *Drools* 5 usa un algoritmo *ReteOO*, la nueva versión tiene un algoritmo llamado *PHREAK* [17]. Antes de meternos a la explicación del algoritmo *PHREAK*, se recuerda cómo funcionaba el motor de inferencias de las versiones

anteriores a la 6 de *Drools*. En concreto, se refiere en este caso al anteriormente mencionado algoritmo *ReteOO* que usa como base el conocido *RETE* pero aplicando una serie de mejoras que se documentan a continuación:

- ***Node Sharing:*** Los nodos compartidos se aplican tanto para el *alpha network* como el *beta network*[4]. En el caso del *beta network* es siempre compartido desde la raíz.
- ***Alpha indexing*:** Los nodos *alpha* que tienen muchos nodos hijo utilizan un mecanismo de búsqueda *hash* para evitar tener que probar cada uno de los resultados.
- ***Beta indexing:*** Los nodos *join*, *not* y *exist* indexan su memoria usando un *hash*. Esto nos reduce los intentos de unión para las comprobaciones de igualdades. Recientemente se ha añadido la indexación de rango a los nodos *not* y *exists*.
- ***Tree based graphs:*** Las coincidencias de los *join* no contienen referencias a las de su nodo padre o nodos hijos. Por tanto cuando se elimine se tendrán que volver a calcular todas las coincidencias de los *join* de nuevo, lo que implica recrear todos los objetos coincidentes del *join* para que se sea capaz de encontrar las partes de la red en la que se deban de eliminar las tuplas. Todo este método explicado es conocido como *symmetrical propagation*. Un grafo basado en árbol nos proporciona referencias al nodo padre y a los nodos hijos por lo que cuando se elimine, lo único que se deba de hacer es seguir las referencias que se tienen. De esta manera, se tiene lo conocido como *asymmetrical propagation*. El resultado es que de esta manera se hará más rápido y el impacto al recolector de basura será menor, además de ser más robusto porque los cambios en los valores no causan pérdidas de memoria cuando ocurren sin que el motor de inferencias sea notificado.
- ***Modify-in-place:*** El algoritmo *RETE* tradicional implementa el modificar como un borrar e insertar de nuevo el objeto. Esto causa que el recolector de basura limpie todas las tuplas *join* y por tanto muchos se recrean de nuevo como parte de la inserción. Con *modify-in-place* se modificaría directamente por lo que se evitaría este problema.
- ***Property reactive:*** También llamado *"new trigger condition"* permite mas reactividad en las actualizaciones de grano fino. Un patrón puede reaccionar a cambios de propiedades específicas e ignorar otras. Esto evita problemas de recursividad y mejora el rendimiento.
- ***Sub-networks:*** Los nodos *not*, *exists* y *accumulate* pueden tener cada uno de ellos anidados elementos condicionales en forma de *sub-networks*.
- ***Backward Chaining:*** Puede funcionar con árboles de derivación de encadenamiento hacia atrás al igual que *Prolog*. La aplicación usa una pila, por lo que no tiene problemas de recursión para grafos grandes.
- ***Lazy Truth Maintenance***
- ***Heap based agenda:*** La agenda usa una cola *binary heap*[5] para ordenar las reglas que se dispararían por relevancia.
- ***Dynamic Rules:*** Las reglas pueden añadirse y eliminarse en tiempo de ejecución mientras que el motor siga rellenándose con los datos.

Una vez se ha explicadas las características de *Drools* 5 se puede pasar a definir la última versión que, como bien se ha mencionado antes, cambia el algoritmo para introducir *PHREAK*.

[4] Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17–37, 1982.
[5] Chris L. Kuszmaul. "binary heap". Dictionary of Algorithms and Data Structures, Paul E. Black, ed., U.S. National Institute of Standards and Technology. 16 November 2009.

Este algoritmo trata de abordar algunos de los principales problemas de *RETE*. Por tanto, no es ningún algoritmo que se haya escrito desde cero, sino que incorpora todo lo existente en *ReteOO* así como las mejoras que tuviera. Aun siendo una evolución del algoritmo *RETE* este no se clasifica ya como una aplicación de la misma, sino que se introduce una nueva clasificación puesto que los cambios que lleva así lo merece.

Los algoritmos *RETE* tienen como características principales (y en donde se buscan las optimizaciones) que son algoritmos *greedy*[6] (produce todas las coincidencias parciales en las reglas) y que está orientado a los datos, por lo que prácticamente todo el trabajo se realiza durante la inserción, la actualización y el borrado de los mismos. Como *RETE* es un algoritmo *greedy*, esto puede ocasionar que en grandes sistemas se produzcan fallos y se desperdicie mucho trabajo (trabajo sobre todo en esfuerzos para encontrar reglas que se puedan disparar y que al final no son disparadas).

Todo lo contrario ocurre con el algoritmo *PHREAK* que se caracteriza por ser del tipo perezoso (las coincidencias parciales se retrasan) y orientado a mejorar el algoritmo. *PHREAK* está inspirado por otra serie de algoritmos (que no limitados por) como por ejemplo *LEAP*, *RETE/UL* y *Collection-Oriented Match*. *PHREAK* además, tiene todas las mejoras que se han comentado anteriormente de *ReteOO*, añadiendo a ellas el siguiente conjunto de mejoras que se detallan a continuación:

- Tres capas de memoria contextual. Una para los nodos, otra para los segmentos y otra para las reglas.
- Reglas, segmentos y nodos basados en vinculaciones.
- Evaluación perezosa de las reglas.
- Evaluación aislada de las reglas.
- Orientado a establecer propagaciones.
- Evaluaciones usando una pila, con opción de pausar y reanudar.

Cuando el motor en *PHREAK* arranca, todas las reglas empiezan estando desvinculadas por lo que no puede ocurrir ninguna evaluación de las mismas. Las operaciones de inserción, actualizado y borrado son encoladas antes de entrar al *beta network*. Se usa una heurística simple para seleccionar la siguiente regla para evaluar, que está basada en la regla que más probabilidad tiene de ser disparada; esto retrasa la evaluación y el disparado de las otras reglas. Las reglas sólo podrán ser vinculadas una vez y será cuando se cumplan todos sus antecedentes, a pesar de que todavía el sistema no haya empezado a trabajar. En su lugar, se crea un objetivo que representa a la regla y se coloca en una cola de prioridad que es ordenada por relevancia. Cada una de estas colas se asocian con una *AgendaGroup* y sólo la *AgendaGroup* activa inspeccionará su cola, sacando de ella aquella el objetivo de la regla con mayor prioridad y sometiéndolo para su evaluación. Por lo tanto, el trabajo realizado cambia desde la fase de inserción, actualizado y borrado hasta la fase *fireAllRules*. Sólo la regla para la cual el objetivo fue creado es evaluada, mientras que las demás reglas potenciales que llevan los mismos hechos se retrasan. Mientras se están evaluando las reglas individualmente *node sharing* se sigue consiguiendo a través del proceso de segmentación que se explica más adelante.

Cada vez que en *RETE* se hace un *join* se crea una tupla (o coincidencia parcial) que se propaga a los nodos hijo. Por esta razón es caracterizado por ser un algoritmo orientado a tuplas. Por

[6] Introduction to Algorithms (Cormen, Leiserson, and Rivest) 1990, Capítulo 17 "Greedy Algorithms" p. 329.

cada nodo hijo al que la tupla llegue, intentará llegar al otro lado del nodo y así sucesivamente hasta el final (tal y como se ha explicado en la red *RETE*). Esto lo que genera es un "efecto" de recursión descendente, hiperpaginando la red ya que desde el punto de entrada hasta el *beta network* intenta propagarse hacia debajo, arriba, derecha e izquierda hacia todos los nodos hoja.

Sin embargo la propagación de *PHREAK* es orientado a colecciones en vez de a tuplas. Para cada regla que se está evaluando, se visita el primer nodo y se procesan las operaciones de inserción, borrado y actualizado que estuvieran en la cola. Los resultados se añaden a una colección y se propagan al nodo hijo. En este nodo hijo se vuelven a procesar las operaciones de inserción, borrado y actualizado de la cola y los resultados los introducen en la misma colección para ser propagado y así sucesivamente hasta que se llegue a un nodo terminal. Con eso se consigue se sólo se haga una pasada (efecto *pipeline*) y se aísla a la regla que se está evaluando. Se consigue un efecto de proceso por lotes que proporciona ventajas de rendimiento en algunas de las reglas (al igual que las sub-redes con las acumulaciones). También de cara al futuro esto permitirá sacarle partido a las máquinas multi-núcleo de varias maneras.

Las "conexiones" y "desconexiones" usan una máscara de bits por capas, basadas en la segmentación de la red. Cuando se está construyendo la red para una regla los segmentos son creados para los nodos que comparten el mismo conjunto de reglas. Una regla está formada por un *path* de segmentos aunque si no es una regla compartida será un elemento único. Una máscara de bit *offset* es asignada a cada nodo del segmento y otra a cada segmento en el *path* de la regla. Cuando al menos hay una entrada a la hora de la propagación de los datos, el bit del nodo se pone en activo. Cuando cada nodo del segmento tiene su bit activo entonces el bit del segmento se pone también en activo, de la misma manera que si algún bit de algún nodo se desactiva, este también se desactivará. Si cada segmento del camino que sigue la regla tiene su bit activo, entonces se puede decir que la regla se ha "conectado" y se crea el objetivo que se trata de programar la evaluación de dicha regla. La misma técnica es usada para detectar nodos, segmentos y reglas sucios. Esto permite que una regla que ya fue "conectada" que vuelva a ser programada para la evaluación si se considera sucia desde la última evaluación. Con la anterior técnica se asegura que ninguna regla se evalúe cuando hay coincidencias parciales en el antecedente, mientras que en *RETE* clásico esto se produce incluso si el último *join* que se realiza es un resultado vacío. Además, como la evaluación de las reglas se realiza siempre desde el nodo raíz, el bit sucio comentado anteriormente permite que los nodos y segmentos que no estén sucios puedan ser saltados.

RETE tiene una única unidad de memoria, la memoria del nodo. En *PHREAK* existen 3 niveles (Ilustración 3) de memoria lo que permite mucha más comprensión contextual durante la evaluación de una regla.

Todas las evaluaciones de las reglas son incrementales y no desperdician el trabajo recalculando coincidencias que ya se han produciendo. El algoritmo de evaluación es basado en pila en lugar del método recursivo, por lo que puede ser parado y reanudado en cualquier momento usando un *StackEntry* para representar al nodo actual que se está evaluando. Cada vez que la evaluación de una regla llegue a una sub-red se crea un *StackEntry* para el segmento del *path* exterior y el segmento de la sub-red. El segmento de la sub-red es evaluado primero y cuando la colección (la que se explicó antes que se iba transfiriendo entre nodos) alcance el final del *path* de la su-red se une a una lista de clasificación para que se alimente el nodo exterior. Dicho *StackEntry* entonces se reanuda y puede procesar los resultados de la sub-red. Esto además tiene un

beneficio añadido que es que todo el trabajo se procesa en lotes antes de propagarlo al nodo hijo, lo que es mucho más eficiente para los nodos acumulativos.

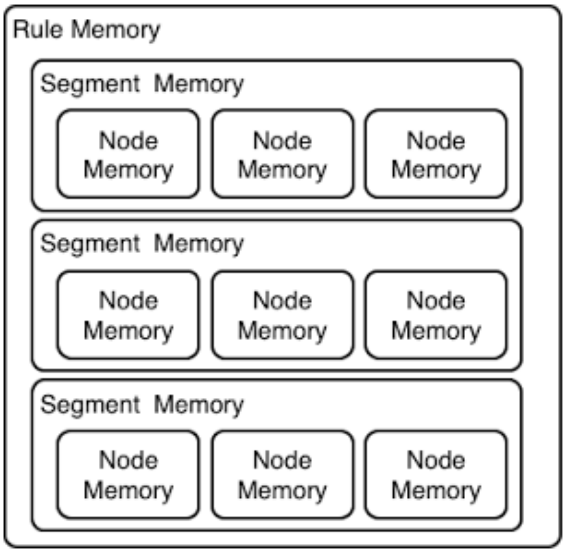


ILUSTRACIÓN 3: CAPAS DE MEMORIA EN *PHREAK*. OBTENIDA DE [17]

Este mismo sistema basado en pila se puede usar para un encadenamiento hacia detrás eficiente. Cuando la evaluación de una regla alcanza un nodo *query* se para la evaluación actual, poniéndolo en la pila. Entonces se evalúa la *query* lo que produce un resultado que se guarda en memoria para que al reanudar el *StackEntry* pueda obtenerlo y propagarlo al nodo hijo. Si esta misma *query* llama a otras *queries* el proceso se repetiría pausando la *query* actual y empezando una nueva evaluación de la *query* actual.

Como último punto en el rendimiento, una única regla por norma general no se evaluará más rápido con *PHREAK* que con *RETE*. Para una única regla y los mismos datos ambos crearán las mismas coincidencias y producirán el mismo número de instancias de reglas por lo que tardarán lo mismo. *PHREAK* también se puede considerar más indulgente que *RETE* para las bases de conocimiento que tengan reglas escritas muy pobremente. Cuando la complejidad de las reglas se incrementa, se degrada también el rendimiento. *RETE* produce coincidencias parciales para reglas que no tienen datos en todos sus *joins* y *PHREAK* evita esto. Por tanto no es tanto que *PHREAK* sea más rápido que *RETE* sino que no se ralentiza tanto a medida que el sistema crece.

2.2.3. PLATAFORMA *OPENCDS*

OpenCDS es un proyecto de código abierto desarrollado por varias instituciones clínicas que trata de generar un estándar en el soporte a la decisión clínica [18]. Con esto se proporcionan varías herramientas y recursos a los que cualquier institución puede adaptarse de tal manera que exista una comunicación eficaz a la hora de, por ejemplo, emitir y recibir datos de algún determinado historial clínico. Este proyecto fue fundado por el doctor *Kensaku Kawamoto*, facultativo del Departamento de Informática Biomédica y copresidente del grupo *HL7 CDS*. Además en la comunidad de ***OpenCDS*** colaboran entidades como *Apelon*, *CDS Consortium*, *HLN Consulting*, *HP Advanced Federal Healthcare Innovation Lab*, *Intermountain HealthCare*, *IsoDynamic*, *ej-technologies*, *Keona Health*, *KS Consulting*, *New York City Department of Health and Mental Hygiene*, *Open Health Tools*, *Tolven*, *University of Utah*, *Utah Department of Health*, *Veterans Health Administration* y *Wolters Kluwers Health* [19].

La arquitectura de ***OpenCDS*** se establece siguiendo esta serie de cuestiones [20]:

- ¿Por qué usar servicios CDS?
  - De esta manera se encapsula conocimiento en componentes altamente reusables.
  - Permite múltiples métodos de representación del conocimiento.
- ¿Por qué usar un estándar?
  - Permite interoperabilidad y escalabilidad.
- ¿Por qué código abierto?
  - Para fomentar la adaptación y la participación

El diagrama de la Ilustración 4 representa el funcionamiento en general de ***OpenCDs***. En él, el **Servicio de Soporte a la Decisión** está implementado con ***OpenCDS*** y es independiente de cada una de las aplicaciones CDS del cliente. Gracias a la estandarización este servicio es capaz de ser utilizada con cada una de las organizaciones de salud y los sistemas informáticos de salud. El funcionamiento es sencillo, desde un hospital cualquiera se envía desde su propia aplicación los datos en este estándar. El **Servicio de Soporte a la Decisión** al tener el mismo estándar, será capaz de interpretar los datos y enviarle los resultados pedidos del paciente. Estos resultados también podrán ser enviados a otras instituciones distintas que implementen el mismo estándar. De esta manera se consigue que todos los hospitales del mundo puedan llegar a **comunicarse de manera eficaz**, cosa que hoy en día es más que útil, pues serviría entre otras cosas para poder trasladar a un paciente a otro hospital sin la necesidad de, a la hora de trasladar su historial clínico, tener que adaptarlo al sistema de este nuevo hospital.

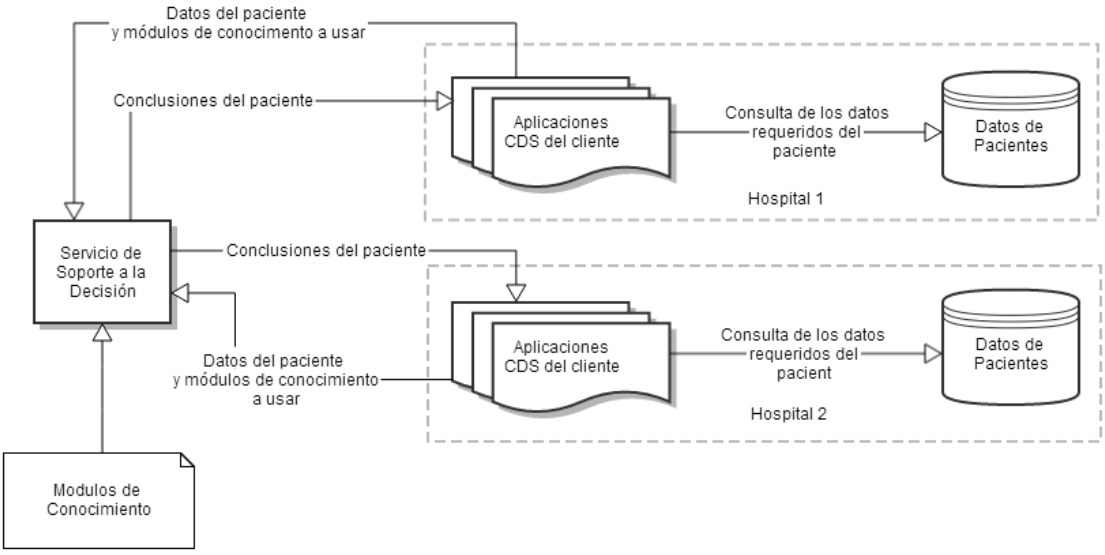


ILUSTRACIÓN 4: DIAGRAMA DE LA ARQUITECTURA DE *OPENCDS*

Se han comentado algunos de los beneficios que se tendrían a la hora de usar esta plataforma, pero no obstante aún no está implementado ningún tipo de estándar común para todas las instituciones. A lo largo de este proyecto se va a hacer una pequeña implementación usando la plataforma y se irán comentando más ventajas y desventajas que se han ido encontrando así como las dificultades a la hora de desarrollar. Antes de esto, se describe cómo funciona por "debajo" ***OpenCDS***, esto es, los componentes y demás estándares de los que hace uso [21]. Esa descripción será de manera general y de "alto nivel" (para detalles de implementación consultar

la sección del proyecto en ***OpenCDS*** y la creación de un pequeño proyecto de ejemplo). Por tanto a grandes rasgos la composición sería la siguiente:

La parte de la interfaz se implementa siguiendo los estándares DSS que se adoptan en HL7[7] y OMG[8]. De esta manera se especifica un estándar en el soporte para la decisión clínica como un servicio software. Este estándar lleva tiempo y esfuerzo implementándose y es conocido como *HL7-OMG Healthcare Services Specification Project*. El proyecto de estandarización está liderado por el doctor *Kensaku Kawamoto*; la misma persona que fundó el proyecto ***OpenCDS***. Principalmente esta interfaz que implementa se subdivide a su vez en tres interfaces que son las siguientes:

- ***Evaluation*:** Usada para evaluar los datos del paciente usando módulos de conocimiento para generar conclusiones específicas para cada uno.
- ***Metadata Discovery*:** Usada para identificar metadatos del servicio y sus módulos de conocimiento.
- ***Query*:** Usadas para consultar módulos de conocimiento de interés.

Como modelo de datos, usa vMR. Este, es un estándar que está contenido en HL7 que está optimizado especialmente para el soporte en la decisión clínica. Se trata de un modelo de datos para representar datos que son analizados y/o producidos por un CDS [22]. El objetivo del proyecto es definir tales modelos de datos que sean escalables e interoperables entre CDS distintos. También implementado por el doctor *Kawamoto* vMR dispone de las siguientes características:

- Diseñado de manera que es fácil de entender y usar.
- Como está basado en un estándar, comparte las características semánticas de otros estándares de HL7 como puede ser CCD (*Continuity of Care Document*).
- Es extensible y soporta plantillas de estructuras de datos.
- Está adaptado a motores de inferencia y otro software diseñados para analizar eficientemente cantidades grandes de datos.

No se usa el estándar en su completitud sino que usa una forma simplificada. Además ***OpenCDS*** puede también comunicarse con sistemas externos usando un formato XML para representar vMR[9].

Como lógica de la plataforma, usa la herramienta *Drools* que se explica en el apartado anterior. *Drools* es un motor de inferencias desarrollado por *JBoss* y que actualmente se usa por todo el mundo. Puede ser usado de dos maneras diferentes:

- Como proyecto web para crear bases de conocimiento, pruebas, versiones y empaquetado usando una compilación de *Drools* conocida como *Guvnor*.
- Creando reglas usando programación de bajo nivel y probándolo bajo *Eclipse*, ya que *Drools* tiene integración con el mismo.

Hasta aquí los componentes principales de la arquitectura de ***OpenCDS*** y a los que se le hará más hincapié en el proyecto. Además, también usa los siguientes componentes opcionales y/o no de tan importancia a la hora de programar usando el estándar ***OpenCDS***:

[7] Más información acerca de HL7: http://hssp-dss.wikispaces.com/hl7_specification
[8] Más información acerca de OMG: http://hssp-dss.wikispaces.com/omg_specification
[9] Más información de vMR: http://wiki.hl7.org/index.php?title=Virtual_Medical_Record_(vMR)

- ***JBoss jBPM*:** Gestor de procesos de negocio de código abierto. Se usa para permitir diagramas de flujo para representar las decisiones clínicas.
- ***Apelon DTS*:** Las siglas corresponden a *Distributed Terminology System* y es un framework de código abierto que da soporte completo a terminologías. Por tanto incluye la posibilidad de mantener contenido de vocabulario médico y relaciones y mapeos entre vocabularios.
- ***Drools DSL*:** *Drools* permite la posibilidad de crear un DSL para manejar de manera intuitiva la base de conocimiento. Con esta herramienta se ocultan todos los detalles técnicos que hay a la hora de crear reglas y muestra una interfaz clínica sencilla de tal manera que se permite la creación de reglas gráficamente seleccionando los componentes que describan los aspectos relevantes, los cuales se mostrarán en un lenguaje clínico común.

Para finalizar este apartado, al final se decide hacer el proyecto en esta herramienta por las siguientes razones:

- Por lo novedoso que es. Por ahora no hay mucha gente que lo haya utilizado por lo que es un gran punto a favor de cara a la innovación.
- Se siguen estándares médicos existentes, por lo que si se consiguiera que todas las instituciones siguieran un mismo estándar se optimizaría muchísimo el servicio médico (si se realiza una aplicación con solamente con reglas, al final no sería compatible en todos los lugares).
- Suple perfectamente las necesidades que plantea este trabajo fin de grado (hacer un soporte a la decisión clínica en farmacovigilancia).
- De cara al futuro, se piensa que va a ser una herramienta fundamental en la investigación, por lo que este trabajo podría ser una buena introducción a la misma.
- El lenguaje de programación sea *Java*, el cuál es de los lenguajes de programación de alto nivel que más se usan y en el cuál es más fácil desarrollar.
- Únicamente se usan componentes de código abierto.

Las limitaciones que tiene la plataforma, son las mismas limitaciones que cualquier CDS. No obstante, con ***OpenCDS*** se intenta principalmente poner fin a la limitación de que no existe una comunicación entre distintos sistemas, puesto que no hay una estandarización.

3. DISEÑO Y RESOLUCIÓN

En esta sección se explica de manera más técnica el proyecto una vez se elige ***OpenCDS*** como herramienta de desarrollo. Se presenta además el caso clínico que se abordará y se hará un estudio de rendimiento de la plataforma elegida para demostrar la eficiencia de la misma.

3.1. CASO CLÍNICO: VAP

[TODO]

3.2. LA APLICACIÓN EN *OPENCDS*

El objetivo principal de este proyecto es desarrollar una aplicación usando la plataforma ***OpenCDS***. En concreto es un CDSS que se centra en el ámbito de las infecciones, para un control en el uso de antibióticos. En cada uno de los apartados siguientes se explicarán aquellas cuestiones más relevantes del proyecto.

3.2.1. ETAPAS DE DESARROLLO

Para abordar este proyecto, se dividirá en tres fases distintas y con un propósito muy claro cada una de ellas. En orden, serían las siguientes:

1. Entender la plataforma ***OpenCDS***. Estudiar los estándares de los que se compone y el funcionamiento del mismo. Realizar un proyecto pequeño en el que intervenga algún paciente sencillo y una base de conocimiento sencilla. Comprobar que funcione correctamente.
2. Modelar la guía clínica en forma de reglas (no toda, sino la parte del proyecto que corresponde a enfermedades infecciosas en la sangre y la orina). Se hará con la herramienta *Drools*. Estudiar cuáles son los conceptos de los estándares son necesarios para esta aplicación (datos de paciente, tipos de síntomas, etc…). Generar por tanto, la base de conocimiento.
3. Incorporar la base de conocimiento obtenida en el paso anterior, a la aplicación que se programó en el paso primero. Crear pacientes con datos reales y hacer las pruebas para comprobar que los resultados sean los esperados.

3.2.2. PAQUETES DE LA PLATAFORMA

Antes de comenzar con el desarrollo del proyecto, es conveniente explicar cada uno de los paquetes de los que se compone la plataforma. Estos paquetes están disponibles una vez que se haya instalado la misma siguiendo las instrucciones del Anexo I. Cada uno de ellos tiene un propósito distinto y son los siguientes:

- `dss-java-stub`:
  - Implementación *Java* de OMG para servicios Web SOAP (protocolo mediante el cual dos objetos en diferentes procesos pueden comunicarse mediante intercambio de datos XML).
  - Código auxiliar auto generado usando *Apache CXF*.
- `opencds-apelon`:

  - Contiene la clase `ApelonDtsUtiliy` que proporciona los métodos convenientes para interactuar con *Apelon DTS*.
  - **Este paquete no es necesario** para usar ***OpenCDS***. No obstante con esta arquitectura se permite cualquier número de servidor de terminologías.
- `opencds-common`:
  - Incluye los componentes comunes que se usan a lo largo de ***OpenCDS***. Por tanto son las utilidades generales, elementos estructurales del núcleo (los datos de HL7) y clases de ayuda a la manipulación de XML.
  - La clase `AbsoluteTimeDifference` es particularmente útil a la hora de realizar inferencias temporales de manera intuitiva.
- Componentes DSS (`opencds-decision-support-service`, `opencds-dss-drools-adapter` y `opencds-dss-evaluation`):
  - En estos sub-proyectos se hallará una instancia de un DSS usando ***OpenCDS*** vMR y los módulos de conocimiento. Por defecto el conocimiento se crea usando *Drools Guvnor* pero el código de la plataforma está diseñado para permitir usar otros motores de conocimiento (por tanto este proyecto final de carrera se centra en gran parte en estos componentes).
  - La plataforma lo único que implementa es la interfaz de evaluación. Las consultas y el manejo de metadatos se deben de implementar también para adaptarlo a nuestras necesidades.
  - La clase principal es un adaptador de *Drools* a la plataforma. Esta clase recoge una petición, convierte un vMR externo en un vMR interno (conversión de los datos a unos que interprete la plataforma) y ejecuta una base de conocimiento con esos datos. Los resultados finales se envían al usuario que realizó la petición.
    1. La primera vez que se usa una base de conocimiento tardará más porque habrá que cargarla. Si se vuelve a usar será mucho más rápido ya que estará previamente cargada en memoria.
    2. Actualmente las bases de conocimiento se crean con *Drools Guvnor* y se guardan en un fichero de formato `.drl` (fichero de *Drools*). En el caso de este proyecto las reglas se diseñarán directamente en el fichero. De cara al futuro la plataforma intentará crear bases de conocimiento dinámicas.
- `opencds-knowledge-repository`:
  - Este modulo contiene los repositorios para los módulos de conocimiento (en este caso las reglas), el mapeo de los conceptos y terminologías de ***OpenCDS*** y otra información asociada que se necesite en tiempo de ejecución.
  - Contiene los métodos necesarios para guardar y recuperar la información actualizable de la base de datos.
- `opencds-terminology-service`:
  - Esta parte tampoco es necesaria. Este sub-proyecto se intentará convertir en un servicio Web en el futuro.
  - Contiene de manera abstracta un gestor de terminologías de manera que otros sub-proyectos de ***OpenCDS*** puedan interactuar con él para obtener capacidad de inferir terminologías a través de una interfaz común.
- `opencds-simple-terminology-service`:
  - Es una implementación concreta del sub-proyecto abstracto anterior. Proporciona una implementación basada en ficheros XML (por ejemplo,

especifica los mapeos que existen entre el código de las terminologías estándar y los conceptos en ***OpenCDS***).
    - Se intenta obtener una base de datos que maneje el contenido terminológico contenido en los ficheros XML.
- Los módulos vMR (`opencds-vmr-v1_0-internal`, `opencds-vmr-1_0-mappings` y `opencds-vmr-1_0-schema`):
    - `internal` como su nombre bien indica, corresponde a la representación interna vMR. Se hace de tal manera que los datos sean compatibles con *Drools Guvnor* ya que será el núcleo de la plataforma. Conceptualmente es equivalente al vMR externo (como es lógico). Se emplea una notación llamada *OpenCDSConcept* lo que permite una separación entre el mapeo de terminologías y la lógica de la aplicación (de esta manera se puede actualizar las clases que realizan el mapeo sin tener que modificar los módulos de conocimiento).
    - `schema` contiene el esqueleto XML para representar el estándar vMR y corresponde al vMR externo.
    - `mappers` como bien se puede imaginar, contiene las clases para poder transformar de vMR externo a vMR interno. Por tanto convierte de XML a formato *Java Beans* y de este a XML.

3.2.3. FICHEROS DE CONFIGURACIÓN DE LA PLATAFORMA

Antes de empezar a programar, se necesitan una serie de ficheros con una determinada estructura, los cuales serán como la configuración del sistema. Estos ficheros se pueden descargar si eres usuario registrado de la web de la plataforma. El contenido del comprimido descargable es un directorio que a su vez, incluye subdirectorios, cada uno con un uso concreto. Al descomprimir, se tiene la siguiente jerarquía de directorios y archivos de la Ilustración 5.

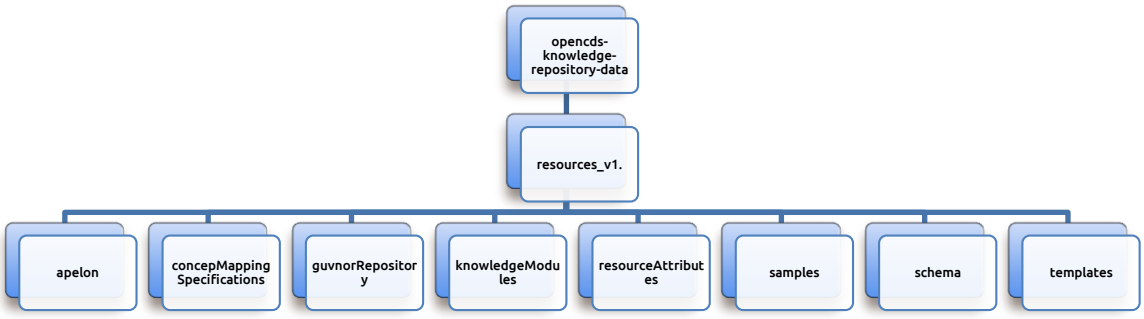


ILUSTRACIÓN 5: JERARQUÍA DE DIRECTORIOS DE FICHEROS DE CONFIGURACIÓN EN *OPENCDS*

De todo esto, las únicas carpetas que son de interés para el proyecto son las de:

- `knowledgeModules`
- `resourceAttributes`
- `samples`
- `schema`

`schema`

Se pone en primer lugar porque es el directorio que incluye todos los ficheros que indican el formato que seguirán todos los ficheros de configuración que se usan para la plataforma. Para ello, usa el lenguaje de esquematización **XML Schema** (ficheros de formado `.xsd`). Todos los ficheros que se usarán en este proyecto con ***OpenCDS***, a excepción de los módulos de conocimiento, están en formato **XML.**

`knowledgeModules`

Incluye las bases de conocimiento de la aplicación. Aunque la plataforma acepte varios formatos para representar las reglas, el que se usará en este proyecto es el formato `.drl` correspondiente a los ficheros de reglas de *Drools*. Estos ficheros tendrán un formato de nombrado que sigue el estándar de HL7. El formato es `entidad^nombre^version.drl` (por ejemplo `umu_fac_info^tfg_tony_wang^1.1.drl`).

`resourceAttributes`

En esta carpeta se encuentra los ficheros principales para configurar correctamente la plataforma. Entre ellos está el fichero que indica la lista de módulos de conocimiento que se usarán, los conceptos médicos de la plataforma, los motores de inferencia disponibles… La estructura de estos ficheros viene explicada en `schema`.

`samples`

Contiene una serie de ficheros de entrada (pacientes) y salida (diagnósticos de pacientes) en formato **XML**. Al igual que antes, la estructura de estos ficheros está incluida dentro del directorio `schema`.

3.2.4. FUNCIONAMIENTO GENERAL DE LA PLATAFORMA

Aunque desde una visión general la plataforma parece muy sencilla, por debajo de la misma funciona de una manera compleja, sobre todo por el hecho de que lo realiza todo bajo estándares clínicos. La complejidad reside en la conversión de los datos clínicos reales en datos informáticos para su posterior análisis con algún motor de reglas y por último, conversión del diagnóstico inferido en datos médicos reales de nuevo. Otra parte compleja para este proyecto, es modelar la guía clínica en forma de reglas, de tal forma que estas reglas reflejen la realidad de la guía además de que respete el estándar de datos que tiene la plataforma.

En una ejecución simple de la plataforma, en la que un hospital cualquiera usa el CDSS intervendrían dos entidades (Ilustración 4) que serían el cliente (el propio hospital) y el CDSS (un sistema común centralizado).

Cliente:

1. Se obtiene de su base de datos el historial clínico del paciente en cuestión que se enviará a analizar al CDSS.
2. Como la aplicación se habrá desarrollado bajo el mismo estándar de representación de los datos clínicos que ***OpenCDS***, simplemente se generará el archivo del historial clínico de este paciente con este estándar.
3. Se elige la base de conocimiento que se usará entre los disponibles del CDSS.
4. Se envía al CDSS el historial del paciente indicando la base de conocimiento que se ha elegido en el paso 3.

5. Se queda esperando la respuesta por parte del CDSS.
6. Una vez recibida la respuesta, se interpreta en la aplicación del cliente.

CDSS:

1. El sistema estará esperando nuevas peticiones por parte de los distintos hospitales que estén conectados a él.
2. Al recibir una petición por parte de algún cliente, será capaz de interpretar la información del paciente recibido, independientemente de quién lo haya enviado puesto que está estructurado bajo el mismo estándar.
3. Dicho lo anterior, la plataforma ***OpenCDS*** dispone de los métodos necesarios para crear clases internas de *Java* y hechos de *Drools*. De esta manera es capaz de mapear toda la información recibida por parte del cliente en formato XML, convertirlos en conceptos ***OpenCDS*** y usarlos como hechos en el SBR.
4. Coge la base de conocimiento que le indica el cliente, y crea un SBR con esa base de datos usando *Drools*. Las bases de conocimiento también se han diseñado previamente respetando el estándar de ***OpenCDS***.
5. Mete los datos del paciente recibidos en SBR que se creó en el paso anterior.
6. Ejecuta el motor de inferencias del SBR y obtiene una solución.
7. Vuelve a deshacer el proceso de mapeo, ahora de hechos de *Drools* y clases internas de *Java* genera un archivo XML.
8. Envía el archivo XML al cliente que solicitó la consulta.

En las Ilustraciones 6 y 7 se puede ver los pasos citados anteriormente en sendos diagramas.

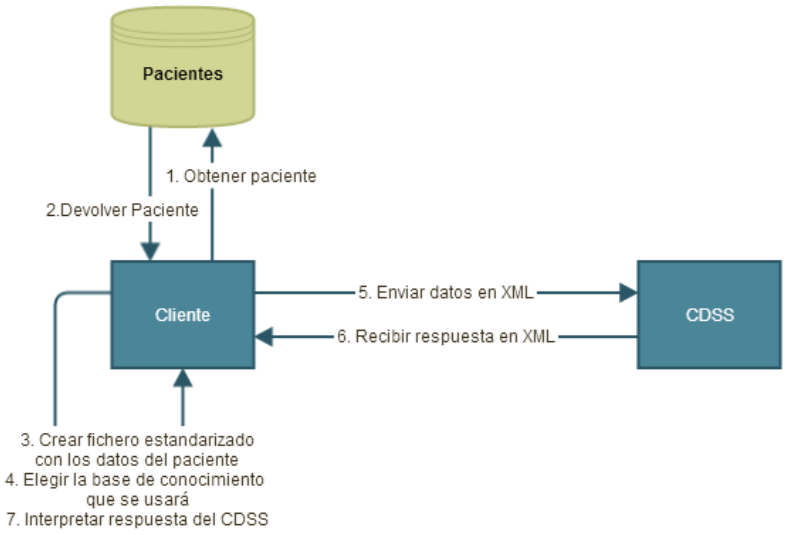


ILUSTRACIÓN 6: DIAGRAMA DE EJECUCIÓN DEL CLIENTE *OPENCDS*

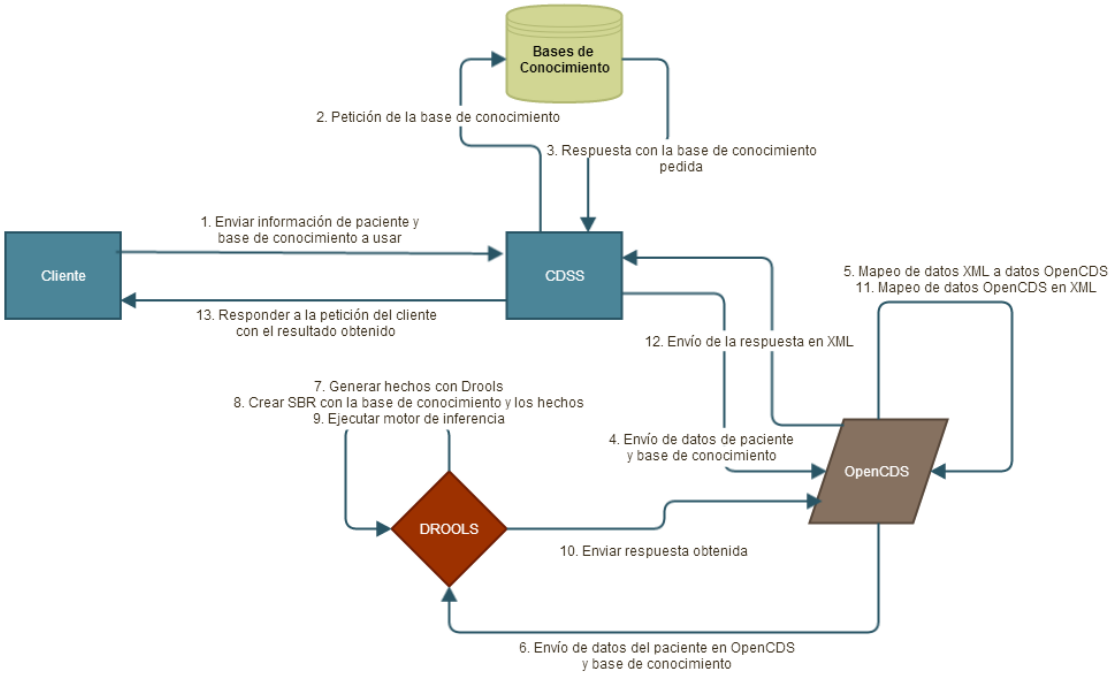


ILUSTRACIÓN 7: DIAGRAMA DE EJECUCIÓN DEL CDSS *OPENCDS*

Una vez estudiado esto, quedaría implementar la parte de las bases de conocimiento, y el mapeo desde una base de datos de una institución al archivo XML siguiendo el estándar ***OpenCDS***. Esta parte es la que se verá en los siguientes apartados.

3.2.5. IMPLEMENTACIÓN Y DESARROLLO DEL PROYECTO

[TODO] Hablar de cómo se ha adaptado la plataforma a la guía clínica que tengo. Cómo he adaptado los pacientes y demás.

3.2.5.1. ESPECIFICACIÓN DE LOS DATOS DE LOS PACIENTES

[TODO]

3.2.5.2. ESPECIFICACIÓN DE LA BASE DE CONOCIMIENTO: MODELADO DE LAS REGLAS

[TODO]

3.2.6. INSTRUCCIONES DE USO

[TODO]

3.2.7. INTERFAZ DE USUARIO

[TODO]

3.3. ESTUDIO DE RENDIMIENTO DE *OPENCDS*

En este apartado se realizará un estudio del rendimiento de la plataforma. Más en concreto se centrará en la herramienta *Drools* que es la que usa para toda la parte de la lógica de la

plataforma. Este apartado será una pequeña parte con el resumen de lo más significativo del estudio, mientras que el estudio completo está disponible en el Anexo I.

Las pruebas se han realizado bajo un ordenador de las siguientes características:

- **Procesador:** Intel® Core™ i7-3610QM CPU @ 2.30Ghz
- **Memoria RAM:** 8,00 GB
- **Sistema Operativo:** Windows 7 Professional N (SP1) de 64 bits

En primera instancia, se explica aquí los detalles de las pruebas realizadas. Se ha implementado un hecho que contiene un único atributo numérico con valor inicial de 0. Por otro lado, se ha desarrollado un software que genera reglas para nuestra aplicación de pruebas que siguen la siguiente estructura:

```
Archivo Rules.drl
      Regla 1: SI hecho.numero=0 ENTONCES hecho.numero=1
      Regla 2: SI hecho.numero=1 ENTONCES hecho.numero=2
      Regla 3: SI hecho.numero=2 ENTONCES hecho.numero=3
      Regla 4: SI hecho.numero=3 ENTONCES hecho.numero=4
…
```

Dependiendo del número de reglas que se quiera generar y empezando con una regla, cada regla siguiente lo único que hará será modificar el valor del atributo numérico para incrementarlo en una unidad. De esta manera, se consigue que por cada hecho nuevo que se cree, se disparen todas las reglas de la base de conocimiento (a partir de ahora BC). Por tanto, si se tienen dos hechos en la base de hechos (a partir de ahora BH) y tres reglas en la BC, se dispararán un total de 6 reglas y la aplicación finalizará.

Si se analiza una aplicación de 2000 reglas y cambiando el número de hechos. Con esto ya se tendría suficiente para hacer el estudio de escalabilidad, puesto que bases más grandes no serían realistas y en rara ocasión tendría lugar en la vida real.

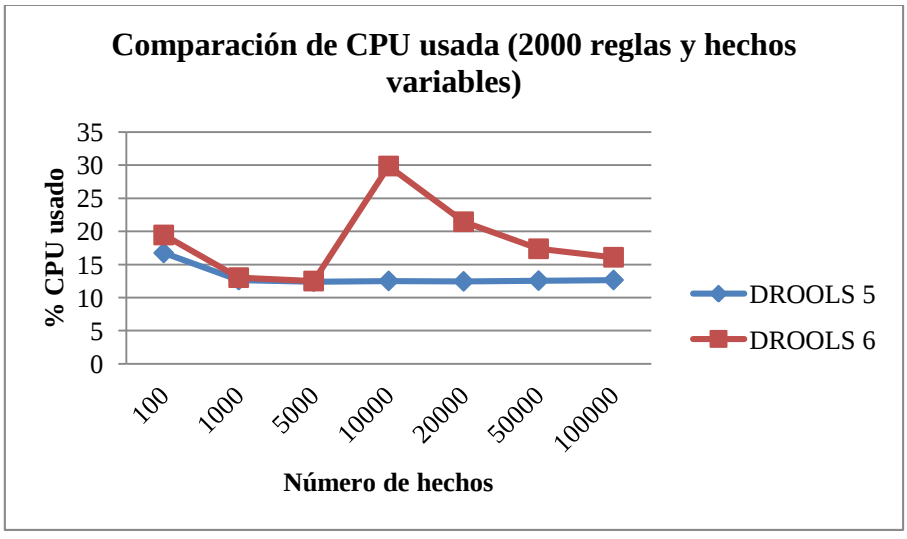


GRÁFICO 1: COMPARATIVA DE CPU USADA CON 2000 REGLAS Y HECHOS VARIABLES

Si se observa el Gráfico 1, en media, *Drools* 5 usa un 13.13% de CPU mientras que *Drools 6* usa un 18.52%. Por lo tanto se puede concluir que *Drools* **6 usa un 41% más de CPU** que *Drools 5*.

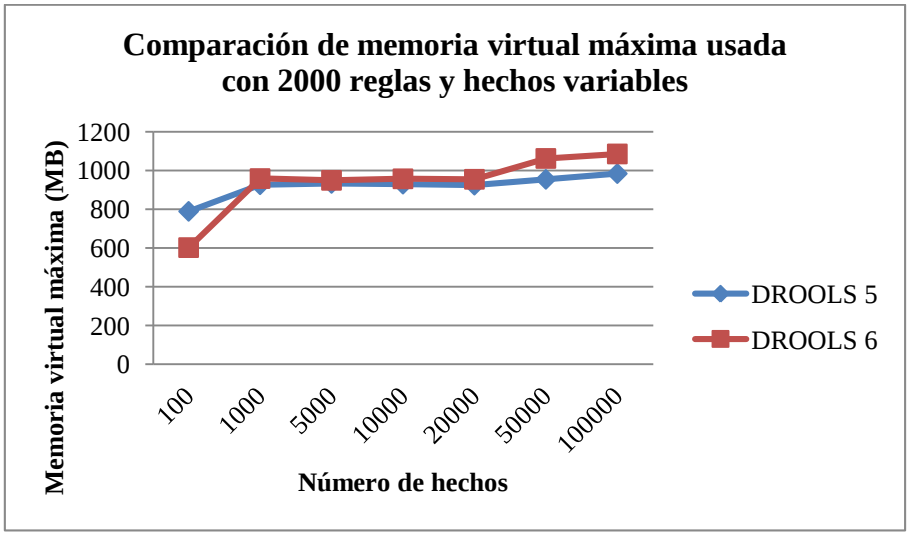


GRÁFICO 2: COMPARATIVA DE MEMORIA VIRTUAL MÁXIMA USADA CON 2000 REGLAS Y HECHOS VARIABLES

Drools 5 usa de media de memoria virtual máxima de 920.14 MB (Gráfico 2) y *Drools* 6 un poco más, 938.41 MB. La diferencia no es mucha y es más importante la memoria física que utilice que se representa en el siguiente gráfico.

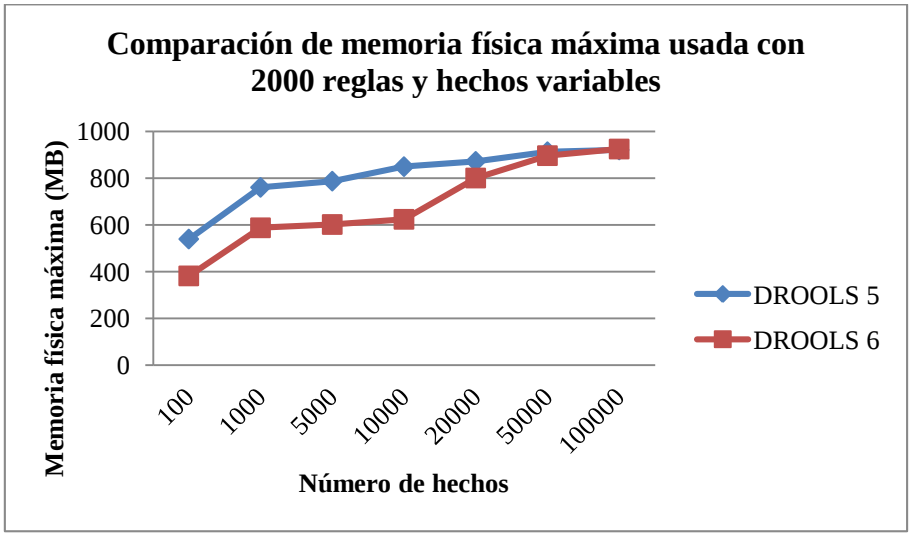


GRÁFICO 3: COMPARATIVA DE MEMORIA FÍSICA MÁXIMA USADA CON 2000 REGLAS Y HECHOS VARIABLES

Se puede ver directamente que *Drools* 6 está por debajo en la mayoría de casos hasta que ambos convergen. Cabe destacar que rara vez una aplicación podría llevar tantos hechos. De media, *Drools* 5 usa de máximo 805.88 MB mientras que *Drools* 6 tiene de media 687.93 MB (Gráfico 3). Por tanto la última versión de *Drools* **ahorra aproximadamente un 15% de memoria** física en sus ejecuciones.

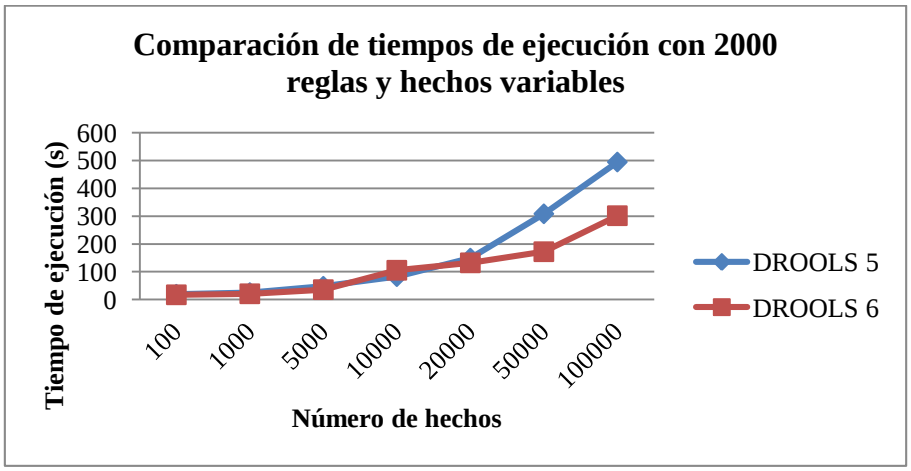


GRÁFICO 4: COMPARATIVA EN TIEMPOS DE EJECUCIÓN CON 2000 REGLAS Y HECHOS VARIABLES

Lo más significativo aquí es que en cuestión de tiempos (Gráfico 4), *Drools* 6 es mucho más escalable. A partir de un tamaño grande, el tiempo de ejecución es mucho menor en esta última versión, aspecto que en la mayoría de casos es la que más importancia tiene. En media, *Drools* 5 se ejecuta en 161.19s mientras que *Drools* 6 tarda de media 112.4s **siendo un 43% más rápido** que su predecesor.

Como conclusión final se puede decir que la última versión de *Drools* usa más CPU para funcionar pero a cambio consigue que la aplicación finalice en bastante menos tiempo cuando las BH y BC superan un umbral. En cuanto al tema de memoria, usada y la memoria física también se mejora pero no es tanta la mejoría. Por lo general, es un cambio bastante significativo el usar un algoritmo distinto al clásico *RETE* que se lleva usando desde el principio.

Una vez hecho esto, existen dos conceptos en *Drools* que son de interés analizar también. Estos son los *Cross Product* y las *Queries*. Los *Cross Product* funcionan como un *join* de SQL [23]. Esto se trata simplemente se realizar un producto cruzado cuando se tienen creados varios objetos y están como antecedentes en una regla. Si se hace buen uso de los *Cross Product* se ahorrará bastante trabajo así como memoria, puesto que se evitaría generar datos inútiles que lo único que hacen es que el motor tarde más en terminar de inferir y ocupar espacio.

Para que se entienda mejor se presenta un ejemplo:

- Sean dos tipos de clases: `Contenedor` y `Contenido`.
- La clase `Contenido` pertenece a un `Contenedor`. Por lo tanto tiene un atributo `Contenedor` que indica cuál es su `Contenedor`.
- Se tienen cuatro objetos de cada tipo: `contenedorA`, `contenedorB`, `contenedorC`, `contenidoA`, `contenidoB` y `contenidoD` (cada una asociada a su contenedor correspondiente).

A la hora de definir las reglas, se deberá de tener en cuenta qué datos son los que interesan ser comparados y guardados. Por ejemplo la regla:

```
rule “Mostrar contenido” when
      $contenedor: Contenedor()
      $contenido: Contenido()
then
```

```
      System.out.println( “contenedor:” + $contenedor.getNombre() + “
contenido:” + $contenido.getNombre() );
end
```

Al no especificar ninguna restricción, se hará un *cross product* de todos los objetos, y devolverá 9 parejas de datos, de los cuales la mayoría no interesan pues no coincide el `Contenido` con su `Contenedor`. Lo que devuelve en este caso es:

```
Contenedor: contenedorA Contenido: contenidoA
Contenedor: contenedorA Contenido: contenidoB
Contenedor: contenedorA Contenido: contenidoC
Contenedor: contenedorB Contenido: contenidoA
Contenedor: contenedorB Contenido: contenidoB
Contenedor: contenedorB Contenido: contenidoC
Contenedor: contenedorC Contenido: contenidoA
Contenedor: contenedorC Contenido: contenidoB
Contenedor: contenedorC Contenido: contenidoC
```

De este resultado sólo interesan las parejas que coinciden, por lo que se tiene que restringir el *cross product* para que sea más eficiente en cuanto a tiempo y memoria. Si no se especificasen correctamente, para reglas que utilicen muchos objetos, el *cross product* podría ser demasiado grande, causando que el motor de inferencias sea extremadamente lento cuando no hay necesidad de ello. La regla correcta sería:

```
rule “Mostrar contenido” when
      $contenedor: Contenedor()
      $contenido: Contenido( contenedor == $contenedor)
then
      System.out.println( “contenedor:” + $contenedor.getNombre() + “
contenido:” + $contenido.getNombre() );
end
```

De esta manera se asegura que los contenidos que se muestren sean sólo aquellos que se asocien de manera correcta a su contenedor. El resultado en este caso sería:

```
Contenedor: contenedorA Contenido: contenidoA
Contenedor: contenedorB Contenido: contenidoB
Contenedor: contenedorC Contenido: contenidoC
```

En este ejemplo tan sencillo ya se está utilizando 2/3 menos de memoria, lo cual de cara a una aplicación grande con muchas reglas y hechos podría traducirse a algo más grande.

En cuanto a las *queries* se trata de un tipo de estructura integrado en *Drools* que se puede especificar en el fichero `.drl` [24]. Es una manera más sencilla de buscar en la memoria de trabajo hechos que coincidan con unas condiciones dadas. Por lo tanto, sólo contiene la parte derecha de una regla o en este caso la parte que se especifica en el `when` obviando poner el `then` y la parte que le sigue. En una *query* también se tiene una serie de parámetros que son opcionales; por ejemplo si no se pone el tipo del objeto, se asume que es el tipo *object* y el motor intentará asociarle el valor que considere que es. Además los nombres de las *queries* son globales para toda la *KieBase* por lo que no se pueden añadir *queries* del mismo nombre a diferentes paquetes para una misma *RuleBase*.

Si se quiere hacer una consulta, únicamente se escribirá `kSession.getQueryResults("nombre")` donde `nombre` es el nombre de la *query*. Esto devolverá la lista de resultados de la *query* lo que permite recuperar los objetos que coinciden con la misma. Por ejemplo, si se tiene una BC de personas con un atributo edad, y se quieren recuperar todas las personas con una edad mayor de 30, se escribiría la siguiente *query*:

```
query "personas mayores de 30"
      persona: Persona ( edad > 30 )
end
```

Aplicando el método anteriormente explicado, se tiene una lista de personas que cumpla esa *query*. También se puede parametrizar la *query* para que cumplan ciertas condiciones. Por ejemplo si se quiere obtener la lista de personas de una edad superior a `x` que vivan en `y` se escribirá:

```
query "mayores de x que viven en y" (int x, String y)
      persona: Persona ( edad > x, localidad == y )
end
```

Esta lista que se obtiene al consultar la *query* se puede recorrer usando un iterador *for*. Cada elemento de la lista es un `QueryResultsRow` y se puede acceder a los datos de la misma que estará en forma de columnas (cada fila es un resultado y cada columna de la misma los datos). El acceso a las columnas se puede hacer por índice o por nombre.

El código *Java* equivalente a realizar una consulta sobre las personas mayores de 30 años sería el siguiente:

```
QueryResults resultados = kSession.getQueryResults("personas mayores
de 30");
System.out.println("Tenemos " + resultados.size() + " personas mayores
de 30 años");
System.out.println("Estas son las personas mayores de 30:");
for (QueryResultsRow fila : resultados){
      Persona persona = (Persona)fila.get("persona");
      System.out.println("persona.getNombre());
}
```

Además para hacer el código más compacto, se añade una sintaxis posicional. Por defecto, cuando se declara un tipo de objeto con varios atributos, la posición de cada uno será en función al orden de declaración. No obstante se puede modificar usando la anotación `@position`. En un ejemplo se puede ver más claro:

```
declare Queso
      nombre: String @position(1)
      tienda: String @position(2)
      precio: int @position(0)
end
```

Si no se hubiera usado la anotación `@position` entonces nombre sería la 0, tienda la 1 y precio la 2. De esta manera la *query* `isContainedIn` que sirve para comprobar si un objeto

está contenido en una localización pasará de tener el patrón `Location(x, y;)` en vez de `Location(objeto == x, localización == y)`.

De esta manera ahora se puede llamar desde una *query* a otra *query* que combinado con algunos argumentos opcionales nos permite una realizar *queries* como una derivación del mecanismo de encadenamiento hacia atrás. Un ejemplo con una *query* que se llama a si misma sería el siguiente:

```
query isContainedIn(String x, String y)
      Location(x, y;)
      or
      (Location (z, y;) and isContainedIn(x, z;))
end
```

De esta manera se devolverían aquellas `Location` que cumplen que tienen un objeto `x` y una localización `y` o bien que tienen una localización `y`, pero el objeto que tiene, es una localización de otro `Location` que tiene como objeto la `x` pasada como argumento.

Una vez explicado esto, también se hace un estudio de ambas versiones de *Drools* a la hora de usar los *facts*. Para este ello se va a crear estructuras de datos nuevas acerca de individuos y se hacen *queries* para consultar los datos de los mismos y calcular el tiempo que tarda *Drools* 5 y *Drools* 6 en resolver la consulta.

Se va a hacer uso de una clase `Persona` que tiene como atributos un entero que será la `edad` y un atributo que será una cadena que represente el `nombre`. En el fichero de *Drools* se tendrá una *query* que permitirá buscar en la BH aquellas personas que cumplan una condición de edad y de nombre. En la BH inicial se introducirán una serie de personas, y sólo 1/3 de ellas cumplirán esas condiciones (de esta manera también se puede ver qué versión de *Drools* es más eficiente a la hora de comparar y descartar hechos).

Se medirá la RAM máxima que usa en la ejecución (tanto física como virtual) y el tiempo que tarda en devolver la consulta. Estas mediciones se harán con las mismas herramientas que se usó para evaluar el tiempo de inferencia dado un determinado número de reglas y hechos que se analizó antes. Los datos que se obtienen son los de los Gráficos 5, 6 y 7.

En ellos se ve como curiosidad que *Drools* 6 reserva mucha menos memoria virtual cuando son pocos hechos, pero al final a medida que el sistema tiene más hechos crece mucho más rápido, y puede ser debido a que el sistema tiene un máximo de memoria virtual permitido y en esos casos ambos se sitúan al límite. De media *Drools* 5 tiene un pico de memoria virtual de 710.36 MB mientras que *Drools* 6 tiene 510.55 MB por lo que **ocupa un 39% menos**.

En cuanto a memoria física, parte que se considera más importante porque es la parte limitada de un ordenador se ve que *Drools* 6 está bastante más optimizado, aunque al igual que con la memoria virtual, llega un punto en el que ambos convergen por no disponer de más memoria disponible para la aplicación. De media *Drools* 5 ocupa un pico de 503.77 MB y *Drools* 6 tan sólo 259.54 MB por lo que **ocupa un 94% menos**, que es casi la mitad.

Al igual que con el motor de inferencias, donde más ventaja saca la última versión de *Drools* es en cuanto a tiempo. Se ve claramente que es mucho más eficiente en cuanto a tiempo las *queries* en *Drools* 6. El tiempo promedio de ejecución en estas pruebas en *Drools* 5 es de 6.67s y el de *Drools* 6 es de 1.36s por lo que es un **390%** más rápido.

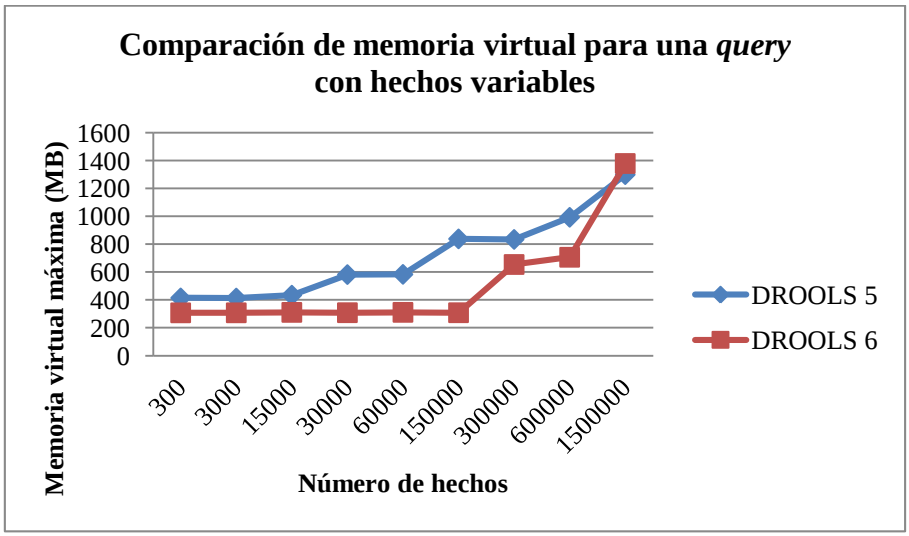


GRÁFICO 5: COMPARATIVA DE MEMORIA VIRTUAL MÁXIMA USADA PARA UNA *QUERY* CON HECHOS VARIABLES

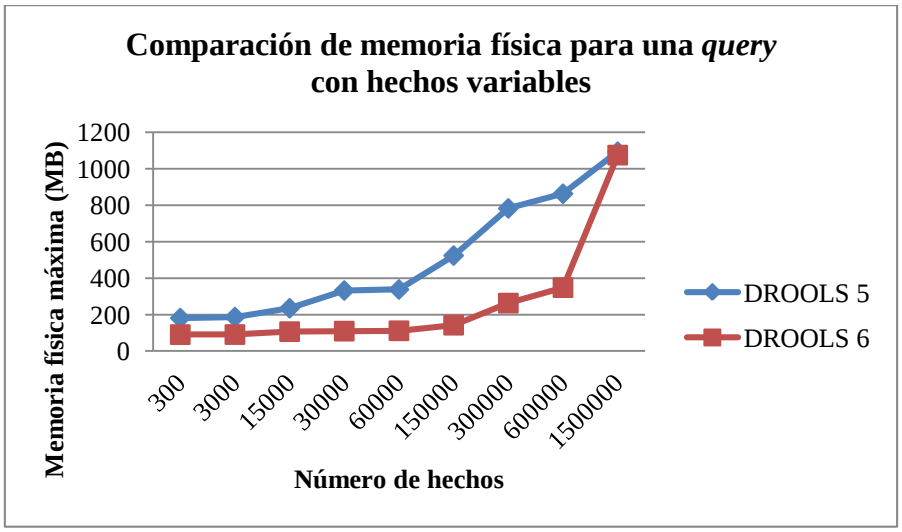


GRÁFICO 6: COMPARATIVA DE MEMORIA FÍSICA MÁXIMA USADA PARA UNA *QUERY* CON HECHOS VARIABLES

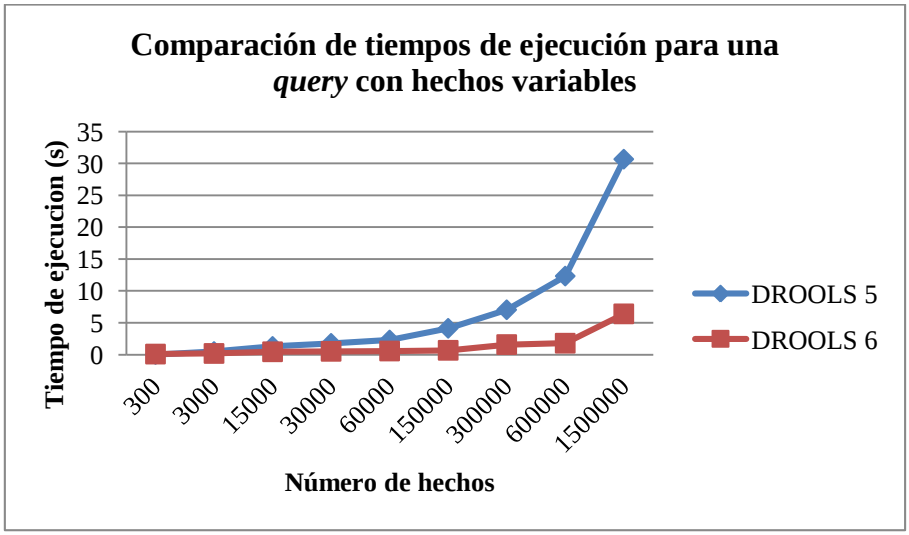


GRÁFICO 7: COMPARATIVA DE TIEMPOS DE EJECUCIÓN PARA UNA *QUERY* CON HECHOS VARIABLES

4. CONCLUSIONES Y VÍAS FUTURAS

[TODO] Comentar las conclusiones del framework así como las dificultades que se han encontrado.

Añadir entre otras cosas, que el framework al estar en continua mejora, cada vez sacan nuevas versiones lo que causa que tengamos que adaptarnos y volver a entenderlo.

Otra contra es que al ser novedoso apenas hay información en Internet.

El código apenas viene comentado y cuesta entenderlo.

La curva de aprendizaje es muy grande, y al principio suele desanimar

Hay que tener nociones de estándares médicos

Técnicamente hablando es muy difícil para una persona que no sepa de informática (usa muchas tecnologías y estándares)

Las actualizaciones ahora mismo son muy lentas (por ejemplo usa DROOLS 5)

No hay manuales de uso

Las limitaciones de OpenCDS (Comparándolo con las limitaciones de los CDS).

[TODO] Apartado incompleto, cuesta mucho entender la plataforma. Hay que generar XML y no se hace de manera automática. Los ficheros de reglas también que generarlos, si no es con guvnor, manualmente y también cuesta. Sobre todo el entender los estándares que usa por debajo.

La dificultad de esto es adaptar todo lo existente a OpenCDS (siguiendo vMR) y luego a DROOLS, modelar las reglas correctamente de la guía clínica…

Aunque este pensado para ser un servicio web, en el proyecto no se ha hecho como tal (vías futuras), se ha emulado el funcionamiento. Trasladarlo a web no debe de ser muy complicado. (Vias futuras) hacer que mediante Drools Guvnor (viene ya con la plataforma) un clínico pueda modelar de manera “fácil” sus propias reglas, así como modificar las ya existentes.

Se podría haber hecho directamente como una APP con reglas, pero entonces no aprovecharíamos las ventajas de un CDS y de usar el estándar.

Hablar de las barreras que se han sorteado con este proyecto

BIBLIOGRAFÍA Y REFERENCIAS

[1] Maguiña-Vargas, C., Ugarte-Gil, C.A., Montiel, M., (2006, 01). *Uso adecuado y racional de los antibióticos*. Recuperado 06, 2014, de http://www.scielo.org.pe/pdf/amp/v23n1/a04v23n1

[2] (2004, 11). California Healthcare Foundation. *Clinical Data Standards Explained*. Recuperado 06, 2014, de http://iha.org/pdfs_documents/calinx/FactSheetClinicalDataStandardsExplained.pdf

[3] (2014, 06). Wikipedia. *Estándares informáticos de uso en telemedicina*. Recuperado 06, 2014, de http://es.wikipedia.org/wiki/Telemedicina#Est.C3.A1ndares_inform.C3.A1ticos_de_uso_en_telemedicina

[4] Goñi, R. (n.d.). *Informática Médica*. Recuperado 06, 2014, de http://www.fm.unt.edu.ar/ds/Dependencias/InformaticaMedica1/manuales/informatica_medica.pdf

[5] González, C. (2004, 05). *La Informática Médica y los Sistemas de Información*. Recuperado 06, 2014, de http://www.medicinadefamiliares.cl/Trabajos/infosiscgs.pdf

[6] June, M.S., Maya, M.D., Maya, J.D., NORC at the University of Chicago, (2010, 03). *Challenges and Barriers to Clinical Decision Support (CDS) Design and Implementation Experienced in the Agency for Healthcare Research and Quality CDS Demonstrations*. Recuperado 06, 2014, de http://healthit.ahrq.gov/sites/default/files/docs/page/CDS_challenges_and_barriers.pdf

[7] (n.d.). U.S. Department of Health and Human Services. *Clinical Decision Support (CDS)*. Recuperado 06, 2014, de http://www.healthit.gov/policy-researchers-implementers/clinical-decision-support-cds

[8] Sittig, D.F., Wright, A., Osheroff, J.A., Middleton, B., Teich, J.M., Ash, J.S., Campbell, E., Bates, D.W., (2008, 04). *Grand Challenges in Clinical Decision Support v10*. J Biomed Inform, pp. 387-392.

[9] (2014, 03). GuiaSalud. *Guías de Práctica Clínica*. Recuperado 06, 2014, de http://portal.guiasalud.es/web/guest/guias-practica-clinica

[10] Espinosa, A.D., Del Sol, L.G., Espinosa, A.A., Garriga, J.L., Viera, B., (2010, 09). *Guías de práctica clínica. Ventajas y desventajas. Una propuesta de indicadores.* Medisur Online, vol. 7, pp. 44-47.

[11] Silva, M.E., Falappa, M., Simari, G., (n.d.). *Sistemas de Soporte a las Decisiones Clínicas*. Facultad de Ciencias de la Administración, Universidad Nacional del Entre Ríos. Recuperado 06, 2014, de http://sedici.unlp.edu.ar/bitstream/handle/10915/19976/3801-Sistemas_de_Soporte_a_las_Decisiones_Cl_nicas_WICC_Silva_Layes__Falappa__Simari.pdf

[12] Palma, J.T., (2010). *Sistemas Basados en Reglas: Mycin*. Desarrollo de Sistemas Inteligentes, DIIC, UMU.

[13] Lopez, M. (2013, 04). *Mycin: un sistema expert asombroso que no se usa*. Recuperado 06, 2014, de http://www.unocero.com/2013/04/22/mycin-un-sistema-experto-asombroso-que-no-se-usa/

[14] Harmon, P., King, D., (1998). *Sistemas expertos. Aplicaciones de la inteligencia artificial en la actividad empresarial* (1ª ed., pp. 186-199). Madrid.

[15] Villena, J., (n.d.). *Dxplain.* Inteligencia en Redes de Comunicaciones, Universidad Carlos III de Madrid.

[16] Palma, J.T., (2010). *Sistemas Basados en Reglas: Técnicas de Equiparación.* Desarrollo de Sistemas Inteligentes, DIIC, UMU.

[17] Proctor, M. (2013, 11). *R.I.P. RETE time to get PHREAKY*. Drools & jBPM. Recuperado 02, 2014, de http://blog.athico.com/2013/11/rip-rete-time-to-get-phreaky.html

[18] (2014, 03). OpenCDS. Open *Clinical Decision Support, Tools and Resources!*. Recuperado 05, 2014, de http://www.opencds.org/FeaturedCollaborators.aspx

[19] (2013, 08). OpenCDS. *Featured Collaborators*. Recuperado 05, 2014, de http://www.opencds.org/FeaturedCollaborators.aspx

[20] (n.d.). OpenCDS. *Architecture*. Recuperado 05, 2014, de http://www.opencds.org/TheSolution/Architecture.aspx

[21] (n.d.). OpenCDS. *Key Components*. Recuperado 05, 2014, de http://www.opencds.org/TheSolution/KeyComponents.aspx

[22] (n.d.). Health Level Seven International. *Product Brief*. Recuperado 06, 2014, de http://www.hl7.org/implement/standards/product_brief.cfm?product_id=271

[23] The JBoss Drools Team, (2013). *Drools Documentation* (Version 6.0.1.Final, pp. 144-145).

[24] The JBoss Drools Team, (2013). *Drools Documentation* (Version 6.0.1.Final, pp. 283-286).

[25]

[m] Shields, D. (2013, 01). OpenCDS. *OpenCDS Developer Workstation Configuration Instructions for OpenCDS 1.1 Release*. Recuperado 03, 2014, de http://develop.opencds.org/OpenCDSProjectDocs/latest/OpenCDSDeveloperWorkstationConfigurationInstructionsfor11Release.doc

[XX] (2014, 01). *Knowledge API (Drools, jBPM). Javadoc*. Recuperado 06, 2014, de http://docs.jboss.org/drools/release/5.6.0.Final/knowledge-api-javadoc/index.html

[XX] The JBoss Drools Team, (2014). *Drools Introduction and General User Guide* (Version 5.6.0.Final).

[XX] Kawamoto, K., (2011). *Virtual Medical Record (vMR) for Clinical Decision Support – Domain Analysis Model*.

ANEXO I: ESTUDIO COMPLETO DEL RENDIMIENTO DE *OPENCDS*

En esta sección se va a hacer una serie de comparativas de los algoritmos explicados en el apartado 2, en la parte correspondiente a reglas, usando un número determinado de reglas y hechos y midiendo el uso de memoria y CPU así como el tiempo tardado[10] hasta que infieren una solución. Realmente se está comparando íntegramente las versiones de *Drools* donde cada una de ellas usa un algoritmo diferente, ya que no se puede aislar y comparar únicamente la parte en la que se hace uso del algoritmo.

Estas pruebas se han realizado bajo un ordenador de las siguientes características:

- **Procesador:** Intel® Core™ i7-3610QM CPU @ 2.30Ghz
- **Memoria RAM:** 8,00 GB
- **Sistema Operativo:** Windows 7 Professional N (SP1) de 64 bits

En primera instancia, se explica aquí los detalles de las pruebas realizadas. Se ha implementado un hecho que contiene un único atributo numérico con valor inicial de 0. Por otro lado, se ha desarrollado un software que genera reglas para nuestra aplicación de pruebas que siguen la siguiente estructura:

```
Archivo Rules.drl
      Regla 1: SI hecho.numero=0 ENTONCES hecho.numero=1
      Regla 2: SI hecho.numero=1 ENTONCES hecho.numero=2
      Regla 3: SI hecho.numero=2 ENTONCES hecho.numero=3
      Regla 4: SI hecho.numero=3 ENTONCES hecho.numero=4
…
```

Dependiendo del número de reglas que se quiera generar y empezando con una regla, cada regla siguiente lo único que hará será modificar el valor del atributo numérico para incrementarlo en una unidad. De esta manera, se consigue que por cada hecho nuevo que se cree, se disparen todas las reglas de la base de conocimiento (a partir de ahora BC). Por tanto, si se tienen dos hechos en la base de hechos (a partir de ahora BH) y tres reglas en la BC, se dispararán un total de 6 reglas y la aplicación finalizará.

Con estos primeros resultados, en el Gráfico 8 se puede ver que como los tiempos de cómputo son tan pequeños no se puede medir bien el uso de CPU que lleva cada ejecución por lo que podría provocar valores incoherentes (se deja a 0.00 cuando el tiempo es inferior a 1 segundo).

En los Gráficos 9 y 10 se puede ver que por mucho que se aumente el número de hechos al tener una regla nada más, la memoria que usa la aplicación no crece tanto (los hechos no necesitan tanta memoria) y el tiempo de cómputo hasta inferir la solución no es tan alto.

La cantidad de memoria que se necesita en *Drools* 6 es menor que para la versión 5. Cuando se tienen pocos hechos, parece que *Drools* 5 tarda menos en terminar, pero la diferencia es ínfima. A medida que aumentan los hechos se ve que la eficiencia de *Drools* 6 aumenta bastante

[10] Para la medición de la CPU se usa el monitor de sistema de Windows. Para le medición de la RAM se usa el método totalMemory() de *Java* y por último para la medición del tiempo, se coloca una bandera antes de introducir los hechos en la BH y se comprueba al final de la ejecución cuánto tardó.

respecto a su antecesor. En cuanto a la CPU, *Drools 6* necesita más cómputo y debe de ser debido a la optimización en el algoritmo que usa. En *Drools* 6 también se ve que llegado a un punto, como la memoria está llena el recolector de basura se encarga de vaciarla.

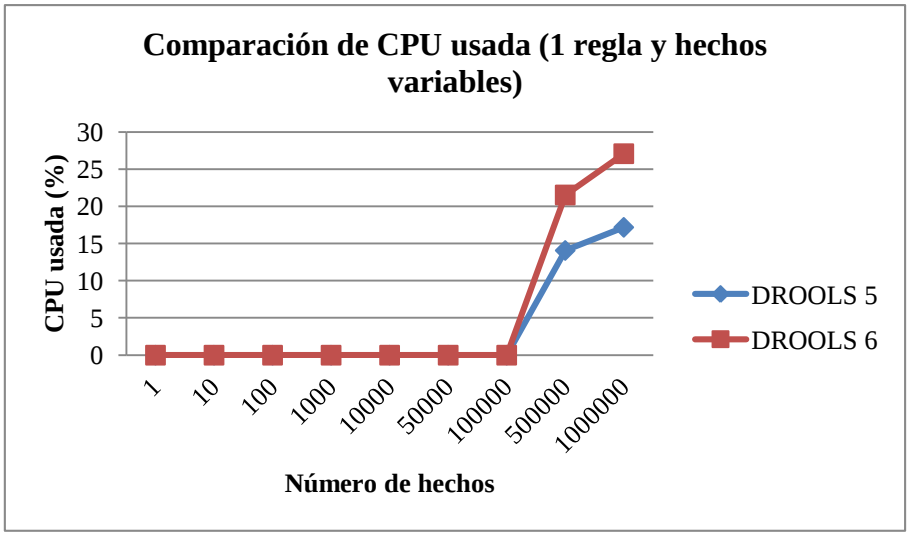


GRÁFICO 8: COMPARATIVA DE CPU USADA CON 1 REGLA Y HECHOS VARIABLES

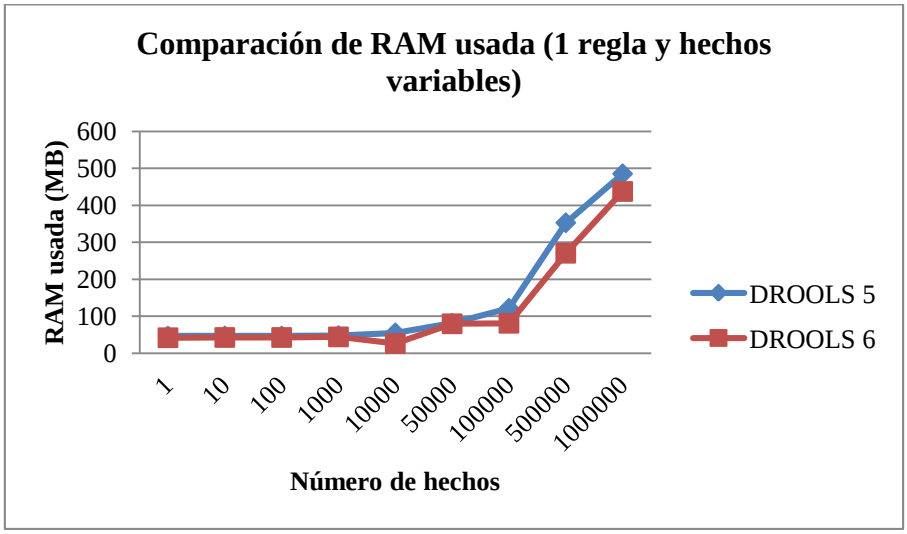


GRÁFICO 9: COMPARATIVA DE MEMORIA USADA CON 1 REGLA Y HECHOS VARIABLES

En cuanto a tiempos de ejecución (Gráfico 11) no se puede apreciar mucho la diferencia puesto que ambos, en estos ejemplos finalizan muy rápido. No obstante parece ser que a partir de tener bastantes más hechos *Drools* 6 empieza a funcionar de manera más rápida. Ahora se realizan las mismas pruebas pero en vez de aumentar los hechos, esta vez se aumentan las reglas y se obtienen los resultados de los Gráficos 12, 13, 14 y 15.

En ellos se puede ver que el número de reglas que permite la aplicación es más limitada (llega un momento en el que se produce error). En estas ejecuciones, el tiempo que tarda en cargar todas las reglas en la BC es bastante grande (el tiempo que sale en la tabla es el tiempo de ejecución). En los datos se observa que las reglas ocupan mucho más en memoria aunque en un punto el recolector de basura trabaje y limpie la memoria. Igualmente en una aplicación real tener más de 2000 reglas no es algo que pueda ser habitual. Entre *Drools* 5 y 6, se cumple también lo mismo que para los hechos. La última versión está más optimizada en cuanto a

memoria pero en tiempo tarda más debido a la optimización que sufre. Aun así, las diferencias de tiempo siguen siendo despreciables.

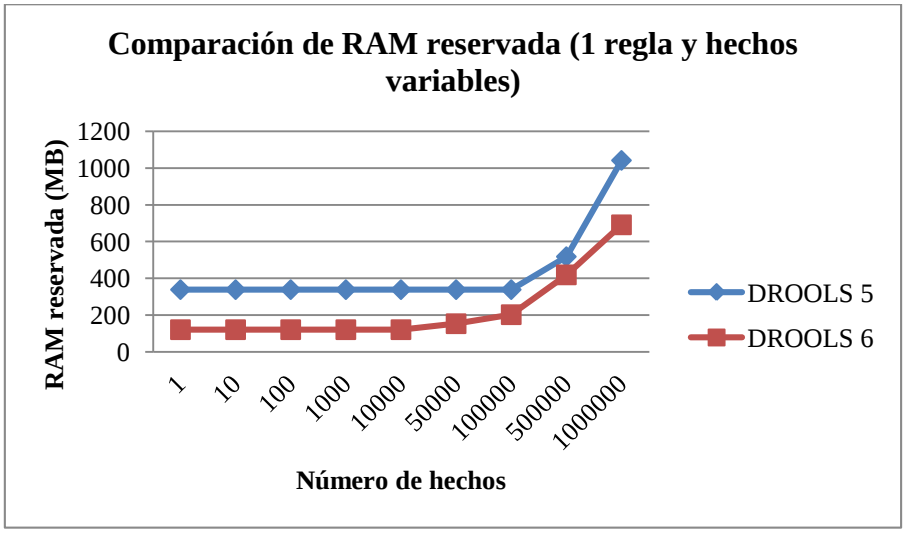


GRÁFICO 10: COMPARATIVA DE MEMORIA RESERVADA CON 1 REGLA Y HECHOS VARIABLES

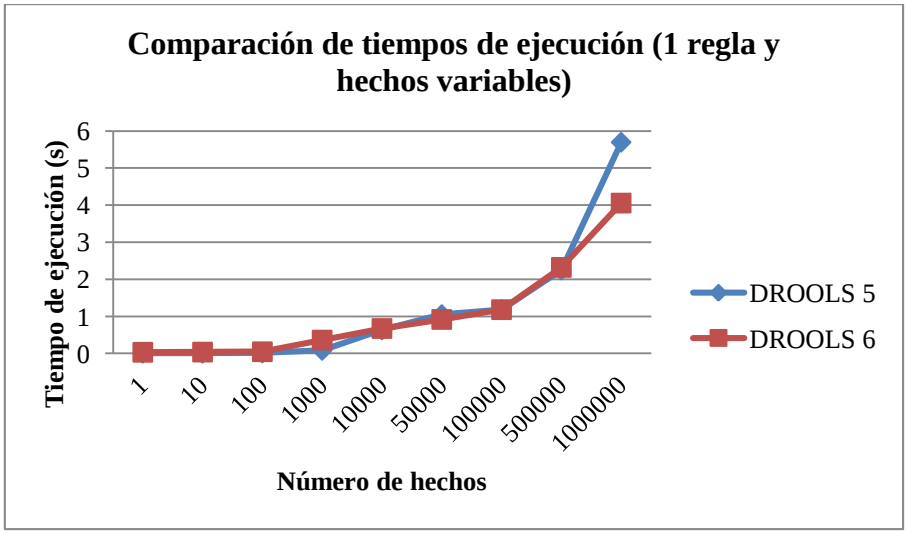


GRÁFICO 11: COMPARATIVA DE TIEMPOS DE EJECUCIÓN CON 1 REGLA Y HECHOS VARIABLES

Se ve que por normal general *Drools* 6 se sitúa por debajo aunque tampoco hay una gran diferencia. En el gráfico se ve que se admite un número máximo de reglas, y que a partir de ese máximo la memoria que ocupa ya es infinita puesto que la aplicación no sería capaz de terminar.

Como caso curioso se puede observar que *Drools* 5 tarda menos tiempo en procesar (Gráfico 15) un número de reglas mayor cuando sólo se tiene un hecho en la base. A partir de ahora al ser BC y BH más grandes y se harán combinaciones de ambos, por lo que los datos de medición serán más precisos y diferirán más por lo que se puede obtener una comparativa más precisa de la escalabilidad.

Lo siguiente es una aplicación con 100 reglas y cambiando el número de hechos y luego una de 500 reglas cambiando también el número de hechos. Estos resultados se representan en los Gráficos comprendidos entre el 16 y el 23.

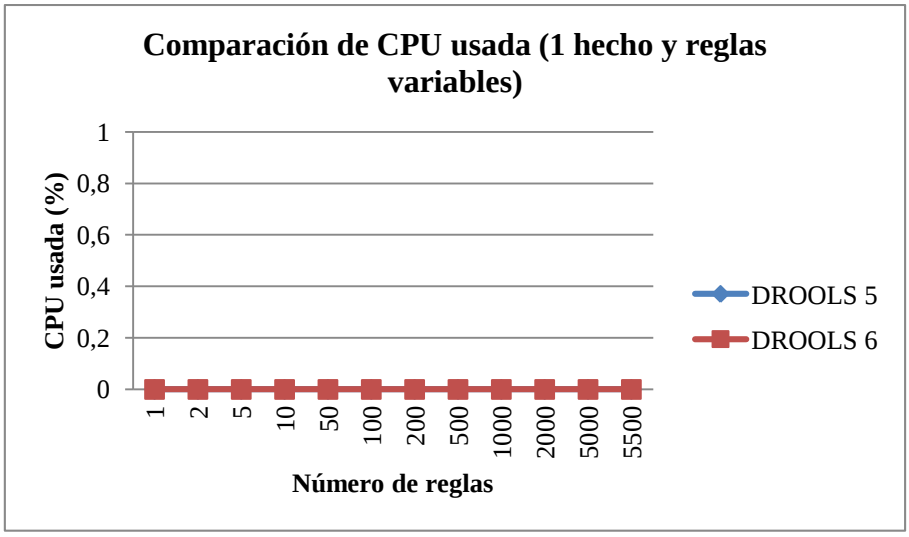


GRÁFICO 12: COMPARATIVA DE CPU USADA CON 1 HECHO Y REGLAS VARIABLES

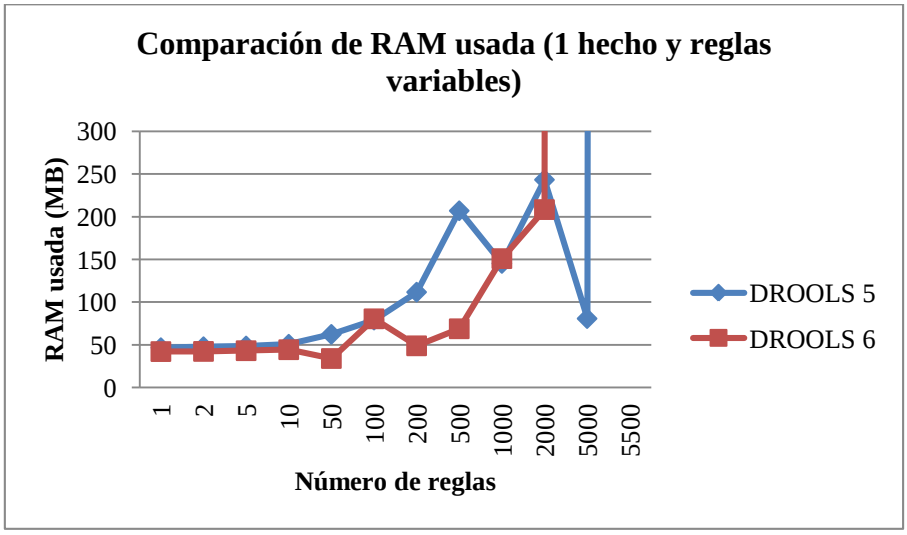


GRÁFICO 13: COMPARATIVA DE MEMORIA USADA CON 1 HECHO Y REGLAS VARIABLES

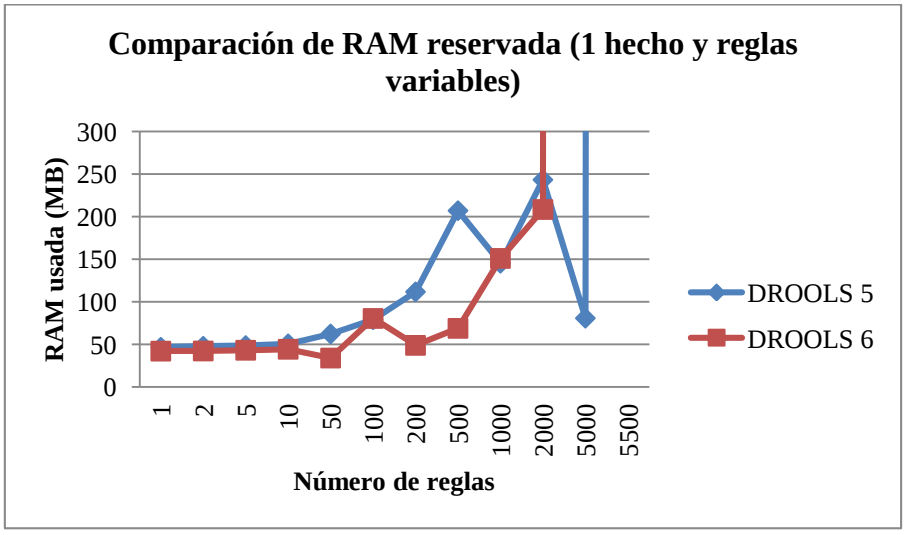


GRÁFICO 14: COMPARATIVA DE MEMORIA RESERVADA CON 1 REGLA Y HECHOS VARIABLES

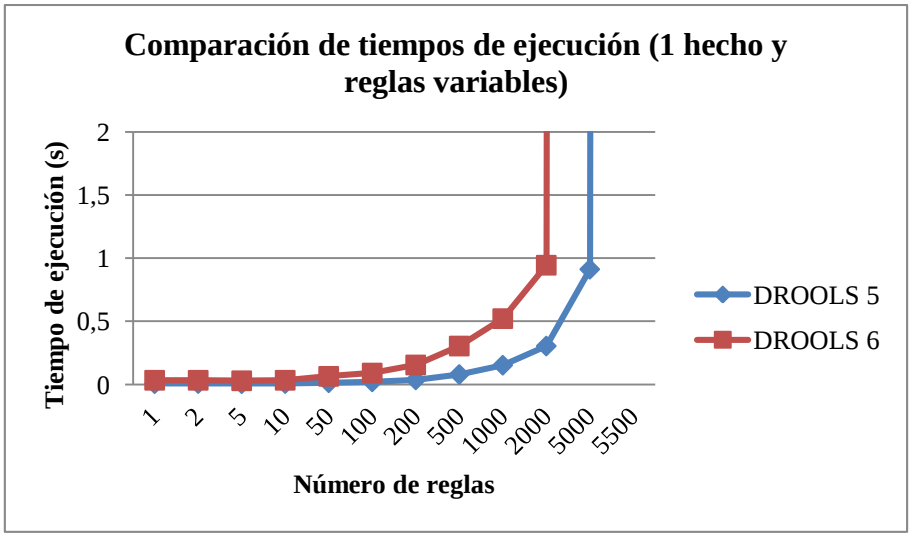


GRÁFICO 15: COMPARATIVA DE TIEMPOS DE EJECUCIÓN CON 1 HECHO Y REGLAS VARIABLES

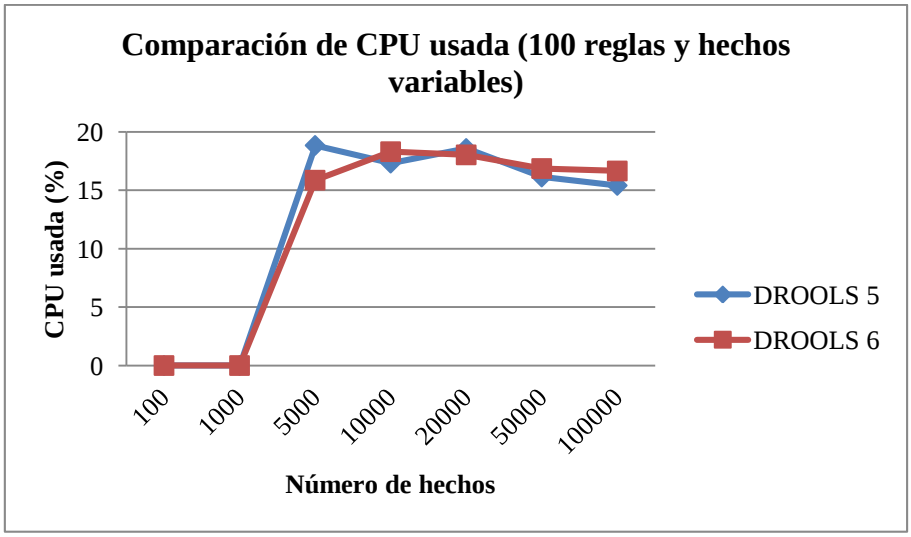


GRÁFICO 16: COMPARATIVA DE CPU USADA CON 100 REGLAS Y HECHOS VARIABLES

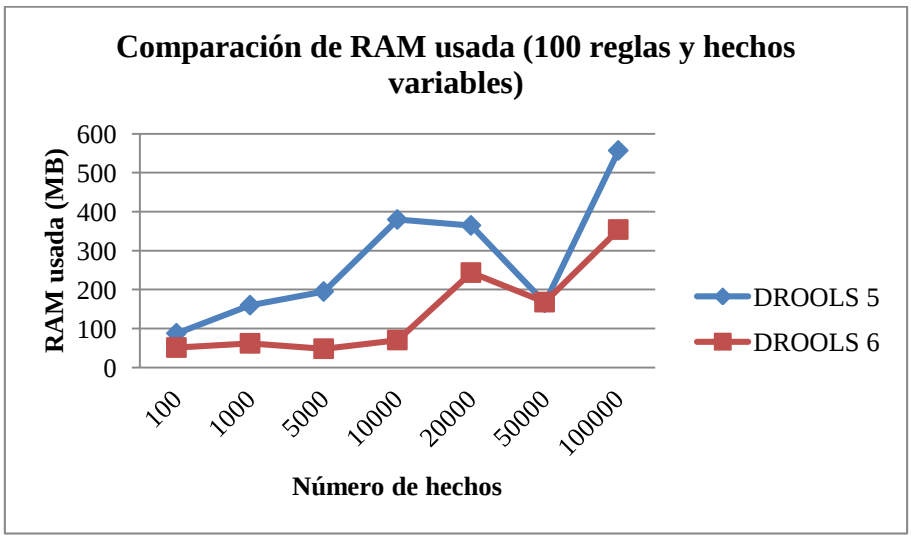


GRÁFICO 17: COMPARATIVA DE MEMORIA USADA CON 100 REGLAS Y HECHOS VARIABLES

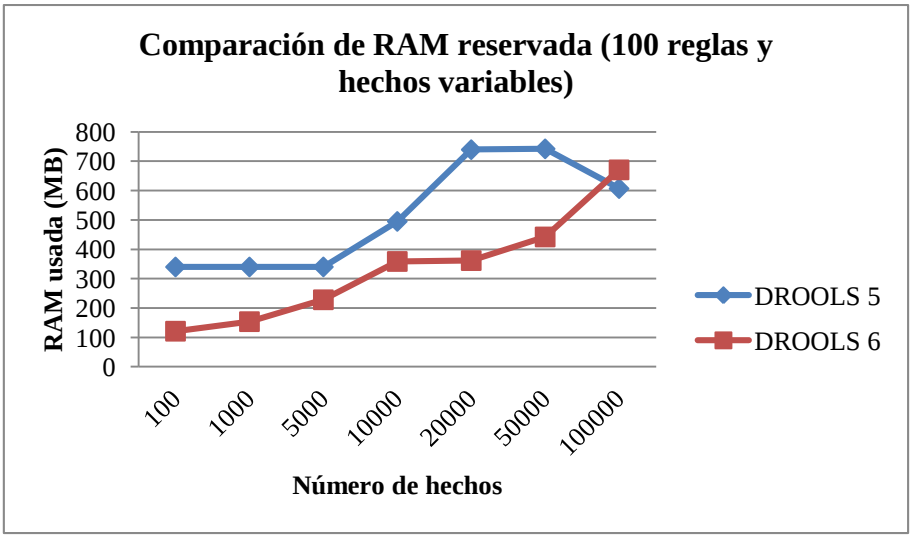


GRÁFICO 18: COMPARATIVA DE MEMORIA RESERVADA CON 100 REGLAS Y HECHOS VARIABLES

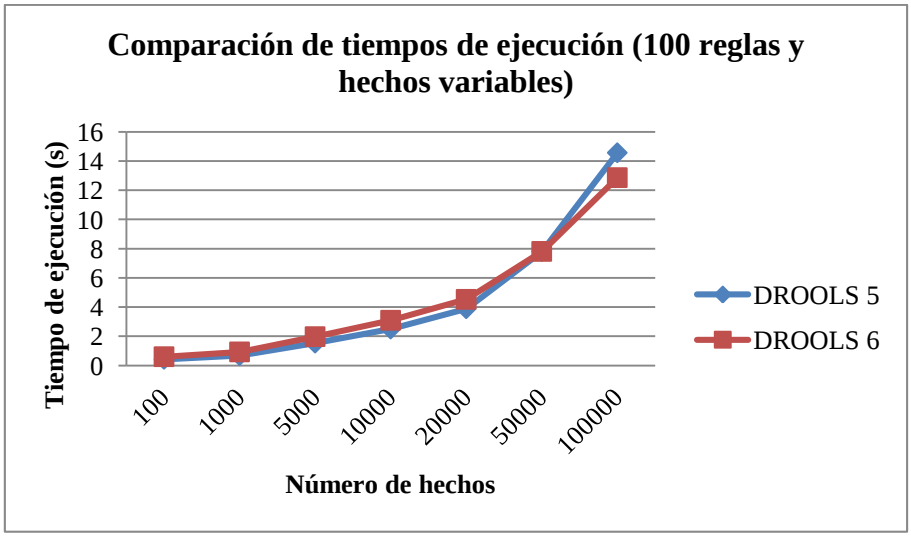


GRÁFICO 19: COMPARATIVA DE TIEMPOS DE EJECUCIÓN CON 100 REGLAS Y HECHOS VARIABLES

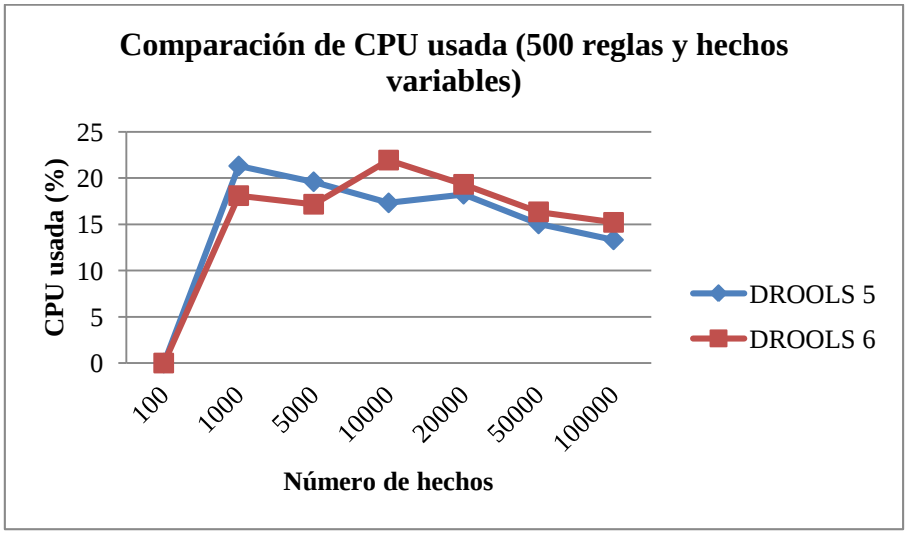


GRÁFICO 20: COMPARATIVA DE CPU USADA CON 500 REGLAS Y HECHOS VARIABLES

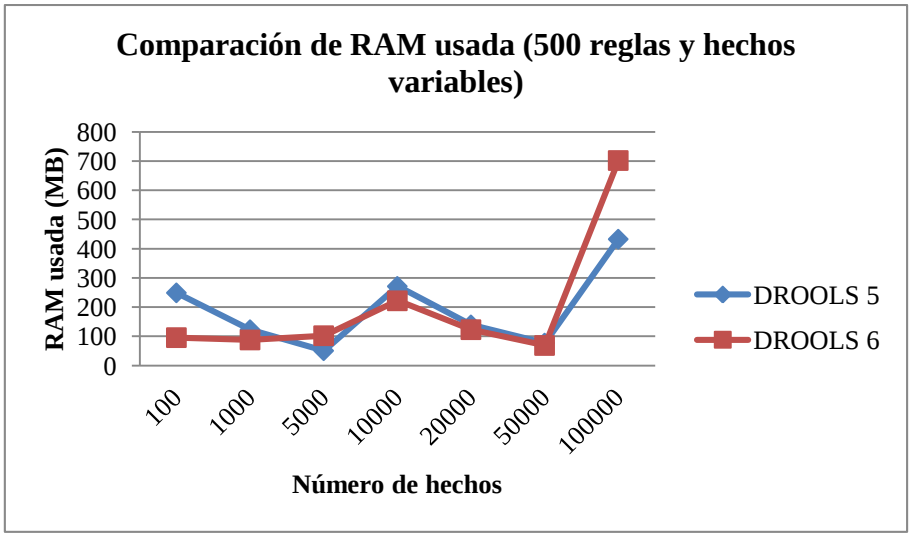


GRÁFICO 21: COMPARATIVA DE MEMORIA USADA CON 500 REGLAS Y HECHOS VARIABLES

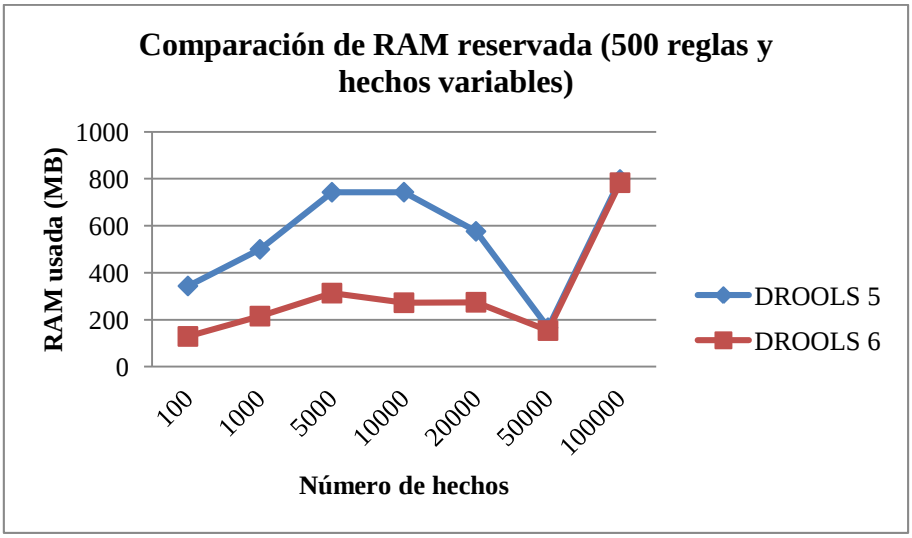


GRÁFICO 22: COMPARATIVA DE MEMORIA RESERVADA CON 500 REGLAS Y HECHOS VARIABLES

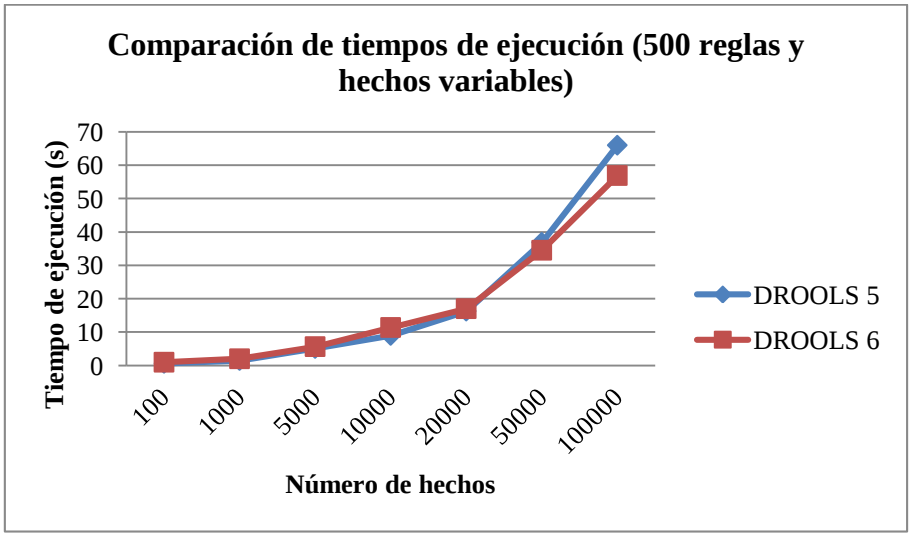


GRÁFICO 23: COMPARATIVA DE TIEMPOS DE EJECUCIÓN CON 500 REGLAS Y HECHOS VARIABLES

En ellos se puede ver claramente que a medida que la base de hechos aumenta, *Drools* 6 es más eficiente en memoria y en tiempo respecto a *Drools* 5. La cantidad de CPU que se utiliza es similar en ambos casos. Por lo tanto cuantos más datos se estén introduciendo, la mejora es mayor (por lo que se demuestra que *Drools 6* es mucho más escalable).

Por último se analiza una aplicación de 2000 reglas y cambiando el número de hechos. Para obtener estos resultados se ha optado por usar un software que sea más preciso a la hora de calcular los datos[11]. Con esto ya se tendría suficiente para hacer el estudio de escalabilidad, puesto que bases más grandes no serían realistas y en rara ocasión tendría lugar en la vida real.

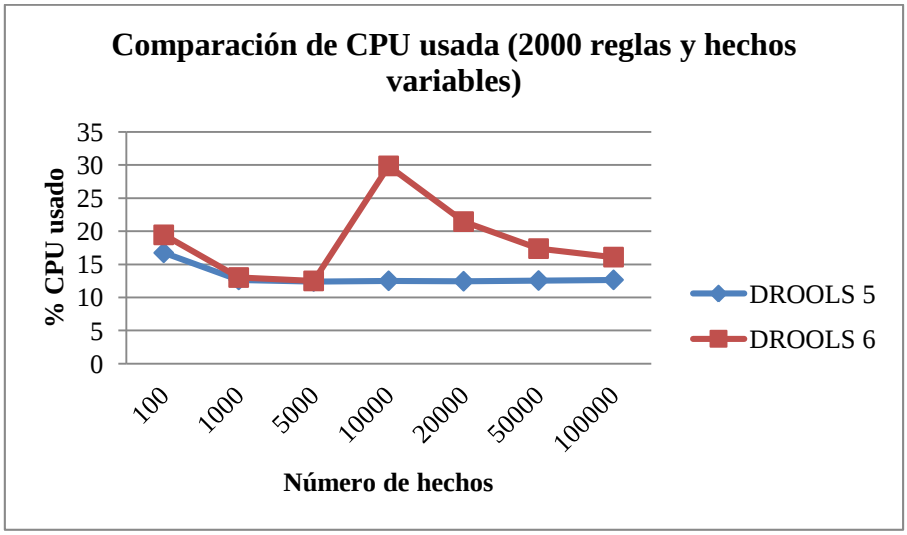


GRÁFICO 24: COMPARATIVA DE CPU USADA CON 2000 REGLAS Y HECHOS VARIABLES

Siguiendo los resultados del Gráfico 24, en media *Drools* 5 usa un 13.13% de CPU mientras que *Drools 6* usa un 18.52%. Por lo tanto se puede concluir que *Drools* **6 usa un 41% más de CPU** que *Drools 5*.

Con la información que se ve en el Gráfico 25, *Drools* 5 usa de media de memoria virtual máxima de 920.14 MB y *Drools* 6 un poco más, 938.41 MB. La diferencia no es mucha y es más importante la memoria física.

En ese caso (Gráfico 26) se puede ver directamente que *Drools* 6 está por debajo en la mayoría de casos hasta que ambos convergen. Cabe destacar que rara vez una aplicación podría llevar tantos hechos. De media, *Drools* 5 usa de máximo 805.88 MB mientras que *Drools* 6 tiene de media 687.93 MB. Por tanto la última versión de *Drools* **ahorra aproximadamente un 15% de memoria** física en sus ejecuciones.

Lo más significativo aquí es que en cuestión de tiempos (Gráfico 27), *Drools* 6 es mucho más escalable. A partir de un tamaño grande, el tiempo de ejecución es mucho menor en esta última versión, aspecto que en la mayoría de casos es la que más importancia tiene. En media, *Drools* 5 se ejecuta en 161.19s mientras que *Drools* 6 tarda de media 112.4s **siendo un 43% más rápido** que su predecesor.

[11] Para esta medición haremos uso de un software externo, y tomaremos los datos desde la carga de las reglas hasta que termina el motor de inferencia. El software usado es el *process explorer* (http://technet.microsoft.com/es-es/sysinternals/bb896653.aspx) y se mide el pico de memoria física y virtual, el tiempo de ejecución desde que se cargan las reglas hasta que infiere la solución y el uso medio de CPU.

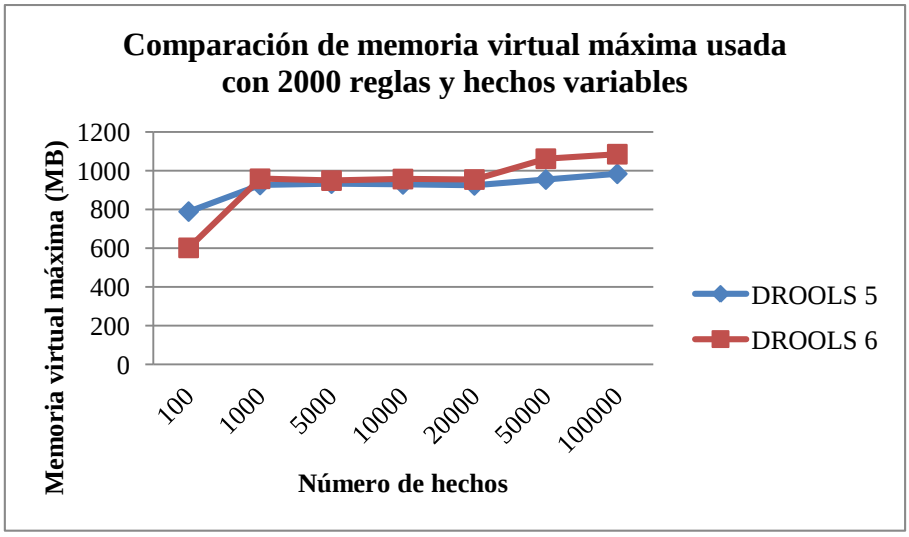


GRÁFICO 25: COMPARATIVA DE MEMORIA VIRTUAL MÁXIMA USADA CON 2000 REGLAS Y HECHOS VARIABLES

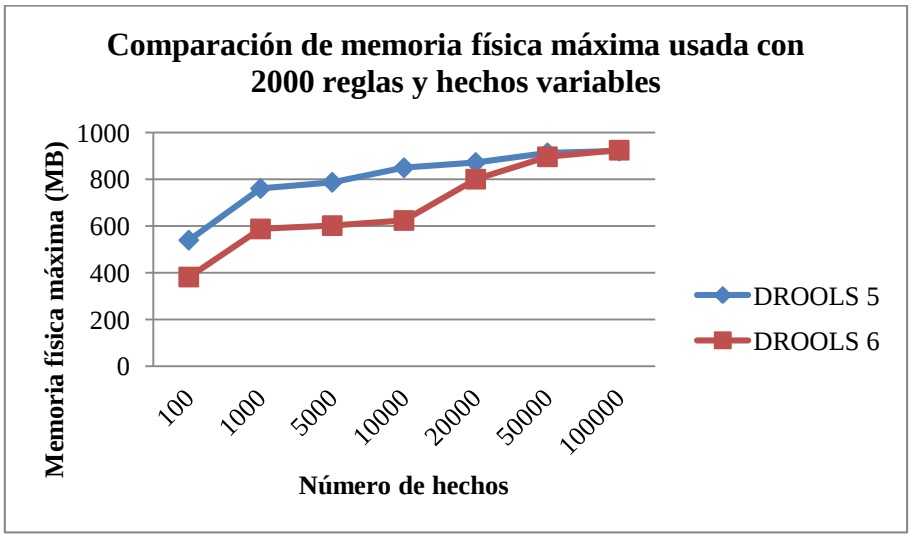


GRÁFICO 26: COMPARATIVA DE MEMORIA FÍSICA MÁXIMA USADA CON 2000 REGLAS Y HECHOS VARIABLES

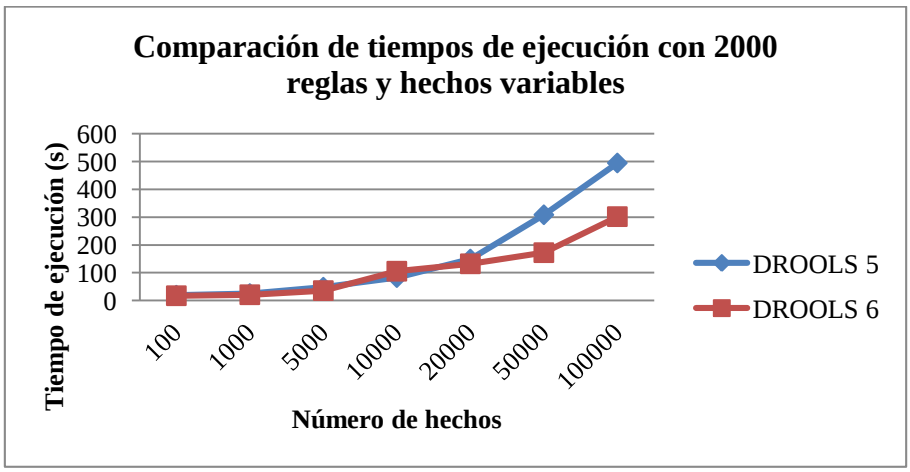


GRÁFICO 27: COMPARATIVA EN TIEMPOS DE EJECUCIÓN CON 2000 REGLAS Y HECHOS VARIABLES

Como conclusión final se puede decir que la última versión de *Drools* usa más CPU para funcionar pero a cambio consigue que la aplicación finalice en bastante menos tiempo cuando las BH y BC superan un umbral. En cuanto al tema de memoria, usada y la memoria física también se mejora pero no es tanta la mejoría. Por lo general, es un cambio bastante significativo el usar un algoritmo distinto al clásico *RETE* que se lleva usando desde el principio.

Una vez hecho esto, existen dos conceptos en *Drools* que son de interés analizar también. Estos son los *Cross Product* y las *Queries*. Los *Cross Product* funcionan como un *join* de SQL [23]. Esto se trata simplemente se realizar un producto cruzado cuando se tienen creados varios objetos y están como antecedentes en una regla. Si se hace buen uso de los *Cross Product* se ahorrará bastante trabajo así como memoria, puesto que se evitaría generar datos inútiles que lo único que hacen es que el motor tarde más en terminar de inferir y ocupar espacio.

Para que se entienda mejor se presenta un ejemplo:

- Sean dos tipos de clases: `Contenedor` y `Contenido`.
- La clase `Contenido` pertenece a un `Contenedor`. Por lo tanto tiene un atributo `Contenedor` que indica cuál es su `Contenedor`.
- Se tienen cuatro objetos de cada tipo: `contenedorA`, `contenedorB`, `contenedorC`, `contenidoA`, `contenidoB` y `contenidoD` (cada una asociada a su contenedor correspondiente).

A la hora de definir las reglas, se deberá de tener en cuenta qué datos son los que interesan ser comparados y guardados. Por ejemplo la regla:

```
rule “Mostrar contenido” when
      $contenedor: Contenedor()
      $contenido: Contenido()
then
      System.out.println( “contenedor:” + $contenedor.getNombre() + “
contenido:” + $contenido.getNombre() );
end
```

Al no especificar ninguna restricción, se hará un *cross product* de todos los objetos, y devolverá 9 parejas de datos, de los cuales la mayoría no interesan pues no coincide el `Contenido` con su `Contenedor`. Lo que devuelve en este caso es:

```
Contenedor: contenedorA Contenido: contenidoA
Contenedor: contenedorA Contenido: contenidoB
Contenedor: contenedorA Contenido: contenidoC
Contenedor: contenedorB Contenido: contenidoA
Contenedor: contenedorB Contenido: contenidoB
Contenedor: contenedorB Contenido: contenidoC
Contenedor: contenedorC Contenido: contenidoA
Contenedor: contenedorC Contenido: contenidoB
Contenedor: contenedorC Contenido: contenidoC
```

De este resultado sólo interesan las parejas que coinciden, por lo que se tiene que restringir el *cross product* para que sea más eficiente en cuanto a tiempo y memoria. Si no se especificasen correctamente, para reglas que utilicen muchos objetos, el *cross product* podría ser demasiado

grande, causando que el motor de inferencias sea extremadamente lento cuando no hay necesidad de ello. La regla correcta sería:

```
rule “Mostrar contenido” when
      $contenedor: Contenedor()
      $contenido: Contenido( contenedor == $contenedor)
then
      System.out.println( “contenedor:” + $contenedor.getNombre() + “
contenido:” + $contenido.getNombre() );
end
```

De esta manera se asegura que los contenidos que se muestren sean sólo aquellos que se asocien de manera correcta a su contenedor. El resultado en este caso sería:

```
Contenedor: contenedorA Contenido: contenidoA
Contenedor: contenedorB Contenido: contenidoB
Contenedor: contenedorC Contenido: contenidoC
```

En este ejemplo tan sencillo ya se está utilizando 2/3 menos de memoria, lo cual de cara a una aplicación grande con muchas reglas y hechos podría traducirse a algo más grande.

En cuanto a las *queries* se trata de un tipo de estructura integrado en *Drools* que se puede especificar en el fichero `.drl` [24]. Es una manera más sencilla de buscar en la memoria de trabajo hechos que coincidan con unas condiciones dadas. Por lo tanto, sólo contiene la parte derecha de una regla o en este caso la parte que se especifica en el `when` obviando poner el `then` y la parte que le sigue. En una *query* también se tiene una serie de parámetros que son opcionales; por ejemplo si no se pone el tipo del objeto, se asume que es el tipo *object* y el motor intentará asociarle el valor que considere que es. Además los nombres de las *queries* son globales para toda la *KieBase* por lo que no se pueden añadir *queries* del mismo nombre a diferentes paquetes para una misma *RuleBase*.

Si se quiere hacer una consulta, únicamente se escribirá `kSession.getQueryResults(“nombre”)` donde `nombre` es el nombre de la *query*. Esto devolverá la lista de resultados de la *query* lo que permite recuperar los objetos que coinciden con la misma. Por ejemplo, si se tiene una BC de personas con un atributo edad, y se quieren recuperar todas las personas con una edad mayor de 30, se escribiría la siguiente *query*:

```
query “personas mayores de 30”
      persona: Persona ( edad > 30 )
end
```

Aplicando el método anteriormente explicado, se tiene una lista de personas que cumpla esa *query*. También se puede parametrizar la *query* para que cumplan ciertas condiciones. Por ejemplo si se quiere obtener la lista de personas de una edad superior a `x` que vivan en `y` se escribirá:

```
query “mayores de x que viven en y” (int x, String y)
      persona: Persona ( edad > x, localidad == y )
end
```

Esta lista que se obtiene al consultar la *query* se puede recorrer usando un iterador *for*. Cada elemento de la lista es un `QueryResultsRow` y se puede acceder a los datos de la misma que estará en forma de columnas (cada fila es un resultado y cada columna de la misma los datos). El acceso a las columnas se puede hacer por índice o por nombre.

El código *Java* equivalente a realizar una consulta sobre las personas mayores de 30 años sería el siguiente:

```
QueryResults resultados = kSession.getQueryResults("personas mayores
de 30");
System.out.println("Tenemos " + resultados.size() + " personas mayores
de 30 años");
System.out.println("Estas son las personas mayores de 30:");
for (QueryResultsRow fila : resultados){
      Persona persona = (Persona)fila.get("persona");
      System.out.println("persona.getNombre());
}
```

Además para hacer el código más compacto, se añade una sintaxis posicional. Por defecto, cuando se declara un tipo de objeto con varios atributos, la posición de cada uno será en función al orden de declaración. No obstante se puede modificar usando la anotación `@position`. En un ejemplo se puede ver más claro:

```
declare Queso
      nombre: String @position(1)
      tienda: String @position(2)
      precio: int @position(0)
end
```

Si no se hubiera usado la anotación `@position` entonces nombre sería la 0, tienda la 1 y precio la 2. De esta manera la *query* `isContainedIn` que sirve para comprobar si un objeto está contenido en una localización pasará de tener el patrón `Location(x, y;)` en vez de `Location(objeto == x, localización == y)`.

De esta manera ahora se puede llamar desde una *query* a otra *query* que combinado con algunos argumentos opcionales nos permite una realizar *queries* como una derivación del mecanismo de encadenamiento hacia atrás. Un ejemplo con una *query* que se llama a si misma sería el siguiente:

```
query isContainedIn(String x, String y)
      Location(x, y;)
      or
      (Location (z, y;) and isContainedIn(x, z;))
end
```

Así se devolverían aquellas `Location` que cumplen que tienen un objeto `x` y una localización `y` o bien que tienen una localización `y`, pero el objeto que tiene, es una localización de otro `Location` que tiene como objeto la `x` pasada como argumento.

Una vez explicado esto, también se hace un estudio de ambas versiones de *Drools* a la hora de usar los *facts*. Para este ello se va a crear estructuras de datos nuevas acerca de individuos y se

harán *queries* para consultar los datos de los mismos y calcular el tiempo que tarda *Drools* 5 y *Drools* 6 en resolver la consulta.

Se va a hacer uso de una clase `Persona` que tiene como atributos un entero que será la `edad` y un atributo que será una cadena que represente el `nombre`. En el fichero de *Drools* se tendrá una *query* que permitirá buscar en la BH aquellas personas que cumplan una condición de edad y de nombre. En la BH inicial se introducirán una serie de personas, y sólo 1/3 de ellas cumplirán esas condiciones (de esta manera también se puede ver qué versión de *Drools* es más eficiente a la hora de comparar y descartar hechos).

Se medirá la RAM máxima que usa en la ejecución (tanto física como virtual) y el tiempo que tarda en devolver la consulta. Estas mediciones se harán con las mismas herramientas que se usó para evaluar el tiempo de inferencia dado un determinado número de reglas y hechos que se analizó antes. Los datos que se obtienen son los del Gráfico 28.

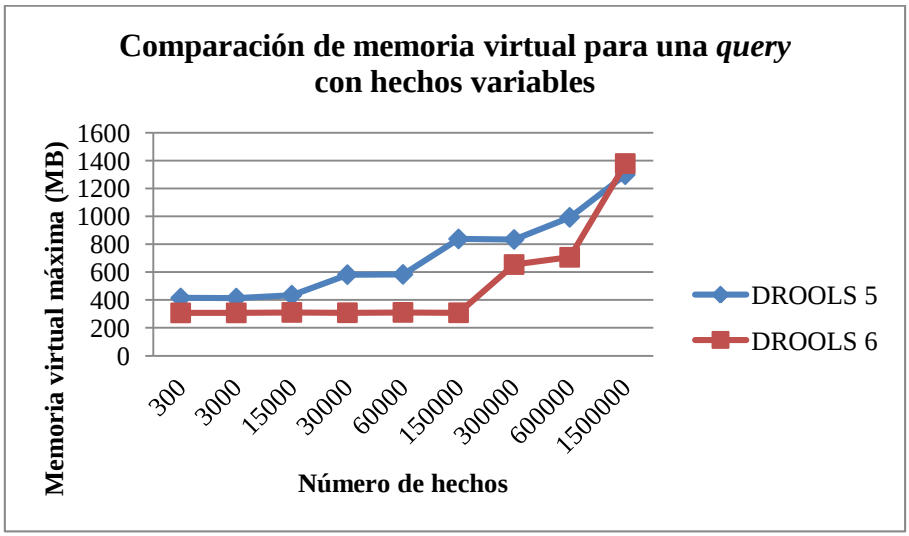


GRÁFICO 28: COMPARATIVA DE MEMORIA VIRTUAL MÁXIMA USADA PARA UNA *QUERY* CON HECHOS VARIABLES

En él se ve como curiosidad que *Drools* 6 reserva mucha menos memoria virtual cuando son pocos hechos, pero al final a medida que el sistema tiene más hechos crece mucho más rápido, y puede ser debido a que el sistema tiene un máximo de memoria virtual permitido y en esos casos ambos se sitúan al límite. De media *Drools* 5 tiene un pico de memoria virtual de 710.36 MB mientras que *Drools* 6 tiene 510.55 MB por lo que **ocupa un 39% menos**.

En cuanto a memoria física, parte que se considera más importante porque es la parte limitada de un ordenador se ve que *Drools* 6 está bastante más optimizado, aunque al igual que con la memoria virtual, llega un punto en el que ambos convergen por no disponer de más memoria disponible para la aplicación. De media *Drools* 5 ocupa un pico de 503.77 MB y *Drools* 6 tan sólo 259.54 MB por lo que **ocupa un 94% menos**, que es casi la mitad (Gráfico 29).

Al igual que con el motor de inferencias, donde más ventaja saca la última versión de *Drools* es en cuanto a tiempo. Se ve claramente que es mucho más eficiente en cuanto a tiempo las *queries* en *Drools* 6. El tiempo promedio de ejecución en estas pruebas en *Drools* 5 es de 6.67s y el de *Drools* 6 es de 1.36s por lo que es un **390%** más rápido (Gráfico 30).

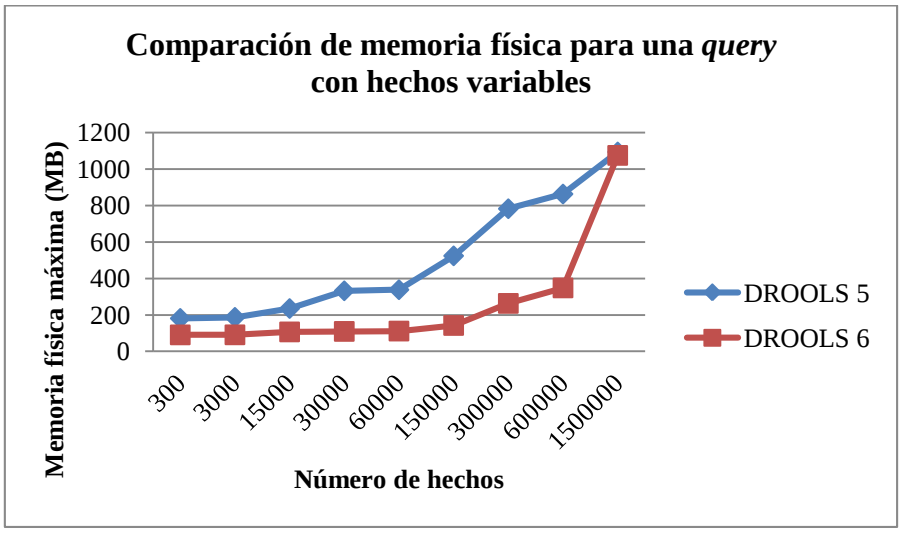


GRÁFICO 29: COMPARATIVA DE MEMORIA FÍSICA MÁXIMA USADA PARA UNA *QUERY* CON HECHOS VARIABLES

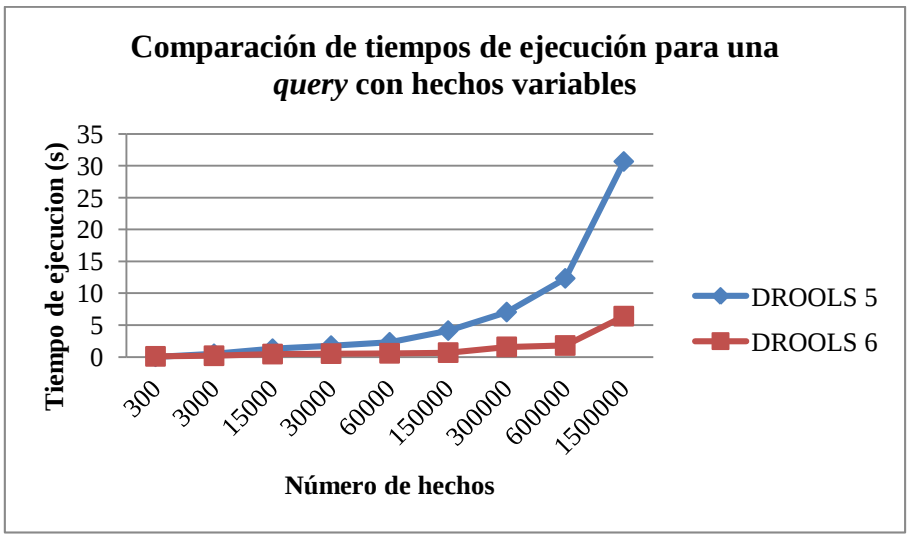


GRÁFICO 30: COMPARATIVA DE TIEMPOS DE EJECUCIÓN PARA UNA *QUERY* CON HECHOS VARIABLES

ANEXO II: INSTALACIÓN Y CONFIGURACIÓN DE *OPENCDS*

En este apartado se va a explicar cómo se instala ***OpenCDS*** [m] para poder desarrollar tu propio sistema de soporte a la decisión médica usando el estándar. Este manual de instalación se realiza sobre Windows 7 de 64 bits. Para el correcto funcionamiento es recomendable desactivar la función de Control de Cuentas de Usuario[12]. Si ya dispone de todo el software necesario para trabajar con ***OpenCDS*** puede pasar directamente al paso 8.

1. Instalación de *Java*. La versión que se usará será JDK 6.
    a. Se accede al Panel de control → Sistema → Configuración avanzada del sistema → Variables de entorno
    b. Se crea una nueva variable del sistema llamada `JAVA_HOME` y con el valor de la variable, la ruta de instalación de JDK 6.
    c. Una vez creada esa variable, se añade a la variable que existe ya `Path` al principio de todo lo que llevase anteriormente, introduciendo `%JAVA_HOME%\bin;.`
2. Instalación de *Maven*.
    a. Descargar la versión 3.0.3 de *Maven* de http://archive.apache.org/dist/maven/binaries/ (Es la versión que se usa para este proyecto aunque existen versiones más nuevas).
    b. Extraer el contenido de apache en: `C:\Program Files\Apache Software Foundation`, lo cual se tendría el directorio `C:\Program Files\Apache Software Foundation\apache-maven-3.0.3`.
    c. De la misma forma que en el apartado 1, se añade una variable `MAVEN_HOME` con el valor `C:\Program Files\Apache Software Foundation\apache-maven-3.0.3` y se añade a la variable `Path`, delante de todo `%MAVEN_HOME%\bin;.`
3. Instalación de *Apache CXF*.
    a. Descargar los binarios de la versión 2.7.11 de http://cxf.apache.org/download.html.
    b. Descomprimir en `C:\Program Files\Apache Software Foundation`, lo cual se tendría el directorio `C:\Program Files\Apache Software Foundation\apache-cxf-2.7.11`.
4. Instalación de *Tortoise SVN*.
    a. Instalar la última versión de http://tortoisesvn.net/downloads.html
    b. Usar la configuración por defecto en la instalación, y una vez terminada reiniciar el ordenador.
5. Instalación de *Tomcat*.
    a. Instalar la versión 6 o 7 de http://tomcat.apache.org/ (está probado y funciona con ambas).
    b. En la instalación, dejar las configuraciones por defecto y poner unas credenciales seguras cuando lo solicite.

[12] Manual para desactivar UAC: http://windows.microsoft.com/es-es/windows/turn-user-account-control-on-off#1TC=windows-7

c. Una vez instalado, y de igual manera que el paso 1 y 2, se añaden a las variables de entorno una nueva variable `CATALINA_HOME` con la ruta `C:\Program Files\Apache Software Foundation\Tomcat X.0`.
d. Una vez hecho esto, también a la variable `Path` se la añade al principio `%CATALINA_HOME%\bin;` al principio del todo.

6. Instalación de la última versión estable de *Eclipse*.
   a. Se usará *Eclipse Helios 3.6.2* descargado de http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2
   b. Se descomprime el contenido en `C:\Program Files\` siempre que no haya ya una versión descomprimida en dicho directorio. Se tendría un directorio ahora llamado `C:\Program Files\eclipse`.
   c. Se crea un acceso directo en el escritorio y en las propiedades del icono, en el campo destino, se cambia por `"C:\Program Files\eclipse\eclipse.exe" -vm "C:\Program Files\Java \jdk1.6.0_45\bin" -vmargs -Xmx900m` poniendo las correspondientes rutas que se hayan elegido de instalación.
   d. Cuando se inicie *Eclipse* por primera vez, se pone (en caso de no tener ya una versión de ***OpenCDS***) `C:\OpenCDS` como ruta de trabajo.
7. Instalación de los *plugins* de *Eclipse*.
   a. Instalación de *M2Eclipse*.
      i. Con *Eclipse* abierto, se pincha en Help → Install New Software.
      ii. Se pulsa sobre *add* y en el campo de *name* se escribe `m2eclipse` y en el campo de *location* se pone `http://download.eclipse.org/technology/m2e/releases/1.3` (una versión posterior de *Maven* no funciona bien con *Eclipse Helios*).
      iii. Se selecciona todo para instalar, y se aceptan los términos. A continuación se instalará y cuando lo pida, se reinicia *Eclipse*.
   b. Asegurarse que se esté usando JDK 6. Para ello se pulsa en Windows → Preferences → Java → Installed JREs → Execution Enviroments. Se selecciona JAVASE-1.6 y se marca la versión 6 de JDK.
   c. Instalación de *Subclipse*.
      i. Se vuelve a pinchar en Help → Install New Software.
      ii. En este caso, en *name* se pone `Subclipse` y en *location* se pone `http://subclipse.tigris.org/update_1.8.x/`.
      iii. Se selecciona el paquete *Subclipse*, se aceptan los términos y se instala. En caso de salga algún mensaje de advertencia, se acepta.
   d. Configuración de *Eclipse* para *Subversion*.
      i. Se selecciona Window → Preferences → Team → SVN → Menu Icons y se selecciona *TortoiseSVN*.
   e. Configuración de *Tomcat* en *Eclipse*.
      i. Se selecciona File → New → Other → Server → Server.
      ii. Se elige la versión de *Tomcat* que se instaló.
      iii. Cuando pida el directorio de instalación, se pone el directorio donde se instaló en el paso 5.
   f. Configuración de *Eclipse* para *Maven*.

i. Se selecciona Window → Preferences → Maven → Installations → Add y se añade la ruta donde se escogió descomprimir *Maven* en el paso 2.

ii. Se crea un archivo `settings.xml` con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings
xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS
/1.0.0          http://maven.apache.org/xsd/settings-
1.0.0.xsd">
<pluginGroups>
      </pluginGroups>
      <proxies>
      </proxies>
      <servers>
      </servers>
      <mirrors>
      </mirrors>
      <profiles>
      </profiles>
</settings>
```

Poniendo la ruta de dicho archivo al que esté especificado en Window → Preferentes → Maven → User Settings.

iii. Se reinicia *Eclipse*.

iv. Actualizar el repositorio central de *Maven*. Para ello se selecciona Windows → Show View → Other → Maven → Maven Repositories. Se hace click en Global Repositories → Click derecho en "central" y Update Index.

g. Instalar el espacio de trabajo de *JBoss Drools* para *Eclipse*.

i. Se selecciona Help → Install New Software.

ii. En este caso se pone en *name* `Drools` y en *location* http://download.jboss.org/jbosstools/updates/stable/helios/.

iii. Se instalan los paquetes *JBoss Drools Core*, *JBoss Drools Guvnor* y *JBoss Drools Task* de *All JBoss Tools 3.2.x*.

iv. Se descarga *Drools Runtine version 5.5.0.Final* de http://download.jboss.org/drools/release/5.5.0.Final/drools-distribution-5.5.0.Final.zip.

v. Se descomprime el contenido de la carpeta *binaries*, por ejemplo en C:\Data\Drools\Drools5.0.0.Final

vi. Se configura *Drools* en *Eclipse*. Para ello se selecciona Windows → Preferences → Drools → Installed Drools Runtime → Add y se selecciona la ruta donde anteriormente se descomprimieron los binarios. Se le pone un nombre, por ejemplo, `Drools 5` y una vez hecho esto, se selecciona esa versión de *Drools* como la que se va a usar.

8. Configuración de ***OpenCDS*** en *Eclipse.*
    a. Llegados a este paso, se han configurado ya todos los componentes necesarios para poder trabajar con ***OpenCDS***. Todos los subproyectos están contenidos en un proyecto padre llamado *opencds-parent* y se explica cómo se instala en los siguientes pasos. Importar proyectos a *Eclipse*.
        i. Se selecciona Window → Show View → Other → SVN → SVN Repositories.
        ii. Se hace click derecho sobre la nueva pestaña que se abre, se selecciona *new* y luego *new repository*. El link que se introduce es `http://develop.opencds.org/svn_v1.0/trunk/opencds-parent`.
    b. Ahora se deben establecer los proyectos *Maven*. Para ello se cierra *Eclipse* y se abre la carpeta que anteriormente se seleccionó como *workspace*. Se hace click derecho y se selecciona `SVN Checkout`. Se abrirá una ventana con el enlace que anteriormente se puso en *Eclipse*. Se acepta para que descargue la última versión de ***OpenCDS***.
    c. Se abre un terminal de Windows, y dentro de la carpeta donde se ha descargado *opencds-parent* se teclean los siguientes comandos:
    ```
    > mvn clean
    > mvn install
    > mvn eclipse:eclipse
    ```
    d. Una vez termine de realizar la consola de realizar las operaciones, se vuelve a abrir *Eclipse* y se le da a File → Import → Maven → Existing Maven Projects → Next. En Root Directory → Browser se escoge la localización del proyecto descargado. Esto listará todos los proyectos, se le da a *finish* y termina.
    e. Ya se podrán ver los proyectos descargados en nuestro espacio de trabajo y estará listo para trabajar con ello. Se recomienda usarlo como copia local y no hacer *update* del repositorio, pues borrará todas las modificaciones que se hagan. Lo más sencillo sería desconectarlo del repositorio de *subversion*.

ANEXO III: GLOSARIO DE TÉRMINOS

- **BC:** Base de Conocimiento
- **BH:** Base de Hechos
- **CDS:** Clinical Decision Support (Soporte a las Decisiones Clínicas)
- **CDSS:** Clinical Decision Support System (Sistemas de Soporte a la Decisión Clínica)
- **DSL:** Domain Specific Language
- **DTS:** Distributed Terminology System
- **EHR:** Electronic Health Record
- **GPC:** Guías de Práctica Clínica
- **HIS:** Health Information System
- **HL7:** Health Level 7
- **OMG:** Object Management Group
- **SBR:** Sistema Basado en Reglas
- **SNS:** Servicio Nacional de Salud
- **SOAP:** Simple Object Access Protocol
- **VAP:** Ventilator-associated pneumonia
- **vMR:** Virtual Medical Record
- **XML:** eXtensible Markup Language