Instituto de Computación – Facultad de Ingeniería
Universidad de la República Oriental del Uruguay

Tesis de Maestría
en Ingeniería en Computación

Reuso de Reglas de Negocio: Una experiencia de reuso de ontologías en un dominio restringido.

Enrique Latorres

Supervisor: Juan Echagüe

Diciembre 2002

INDICE

1 Reconocimientos

Quiero dedicar este trabajo a mi querida esposa Leda Sánchez Bettucci. A mis hijos Andrés, Julián y Manuel, por su paciencia y apoyo. A mis padres, Enrique C. Latorres y Lais S. Martínez, que hicieron posible toda mi educación y a quienes debo quien soy.

Además un reconocimiento al apoyo brindado por mis suegros Mario H. Otero y Lina Bettucci. También quiero reconocer el apoyo brindado por mis compañeros de trabajo en el “Departamento de Informática” del “Despacho de Secretaría y Oficinas Dependientes” del “Ministerio de Transporte y Obras Públicas”.

Y por último, pero no por eso menos importante, un reconocimiento muy especial al apoyo y dirección de mi tutor, Juan Echagüe, cuyas directivas en los momentos justos fueron cruciales para que este trabajo se concluyera en tiempo y forma.

Enrique Latorres

2 Resúmen

Una *ontología* es una teoría lógica que da soporte a la conceptualización de un cierto dominio de conocimiento. En el área de Informática se conoce también con el nombre de *especificación formal*.

Dado un dominio de conocimiento determinado y dos implementaciones particulares dentro de ese dominio, si se analizan las reglas lógicas que definen las actividades de uno y otro a nivel conceptual es esperable encontrar una porción de conocimiento común y de reglas que son fundamentalmente similares.

Se considera *reutilización* toda especificación o parte de ella que se "repita" entre dos unidades ya sea por reutilización explícita o por re-especificación de un concepto común.

Este trabajo estudia la factibilidad de aplicar reconocedores de patrones a las ontologías/especificaciones (a través de la comparación de reglas lógicas por similaridad) con el objetivo de evaluar la reutilización de reglas dentro de un determinado dominio de conocimiento.

En particular, se busca definir conjunto de métricas a ser evaluadas en forma automática, con el fin de comparar varias ontologías sobre dominios similares, desarrollados con criterios similares.

El enfoque de este trabajo se valida a través de un caso de estudio: las reglas de juegos de cartas.

3 Introducción

Una de las maneras de enfrentar los desafíos propuestos por la llamada "crisis del software" es facilitar el reuso en el proceso de desarrollo. La idea de reusar el esfuerzo realizado en el pasado para disminuir los costos de los desarrollos futuros, ha estado en el tapete desde siempre. La puesta en práctica de esta idea es tan variada como actividades y paradigmas hay en el desarrollo de software. Se ha intentado reusar, con variada fortuna, subrutinas -en bibliotecas-, objetos, componentes, programas. Pero no solo el código se reusa, también el diseño (e.g. Los patrones o *patterns*) y también los requerimientos (e.g. La especificación de seguridad informática en el *Common Criteria*).

Las ideas de "qué es lo que debe reusarse" entre distintos proyectos de software es aún más un arte que una ciencia. Sin embargo algunos patrones parecen mantenerse:

a) Si el reuso es factible, la ganancia de rentabilidad crece con el nivel de abstracción del objeto usado.

b) La factibilidad del reuso, depende del “lenguaje” en el que se expresa lo que se espera reusar.
c) La reusabilidad aparece tanto en cosas de audiencia amplia (e.g. en cosas comunes a muchos sistemas) como en cosas de dominio restringido (e.g. los programas de un cierto dominio de negocios).
d) La reusabilidad depende de otras cosas que aún no sabemos prever.

En este trabajo nos interesamos en el reuso de un elemento particular en la construcción de software: las reglas de negocio. Se trata de elementos de alto nivel de abstracción (a nivel de especificación, no de diseño ni de código) y de dominio restringido.

Estas reglas de negocio no solo son un elemento interesante en la producción de software, sino que, en tanto representan conocimiento específico del negocio, son un elemento interesante en la gestión de conocimiento, otra área donde la reutilización -en este caso del conocimiento- es un elemento clave.

De la gestión del conocimiento tomaremos una idea clave de este trabajo: La idea de reglas de negocios, como muchos otros conocimientos, pueden representarse en ontologías.

El contenido central de este trabajo será el realizar experimentos de representación de reglas de negocios mediante ontologías, y ver cuanto de estas representaciones pueden reutilizarse cuando se trabaja en distintos problemas de un dominio reducido.

Las ontologías son un área dentro de la gestión del conocimiento que ha crecido mucho y augura una gran cantidad de aplicaciones al permitir el manejo automático e inteligente de información y conocimiento, de forma accesible y que pueda ser compartida por múltiples entidades, tanto a personas como sistemas.

Esta área de la gestión del conocimiento ha evolucionado mucho en los últimos años, y aparenta un futuro promisorio. También plantea nuevos paradigmas para el tratamiento de la información que van a tomar tiempo antes que sean incluidos como paradigmas habituales dentro de las Tecnologías de la Información.

Hoy estas tecnologías cuentan con límites prácticos, por lo que ciertas promesas todavía deben esperar pues no hay soluciones mágicas, aún. Pero identificar esos objetivos es importante para explorar esos límites y tratar de quebrar las barreras de la tecnología.

Este trabajo trata, también, de un pequeño esfuerzo orientado a traspasar alguno de esos límites tecnológicos en esta área.

Uno de los problemas fundamentales para el éxito de los sistemas de gestión del conocimiento, y en particular las ontologías es la necesidad de facilitar al máximo la integración y reutilización de información de diferentes orígenes en una nueva ontología o base de conocimiento **[Pin1999]**. El ideal es que la integración de una base de conocimiento con otra fuera lo más automática posible. Quizás la automatización total sea una utopía, muchas pautas dan para sospechar que al menos es un problema muy complejo. Pero buscar técnicas que puedan facilitar y automatizar lo máximo posible ese proceso, es fundamental para la usabilidad y el éxito de este tipo de paradigmas.

Una técnica necesaria para la integración de reglas en bases de conocimiento es disponer de métricas de similaridad o de reutilización conceptual entre reglas, para luego definir la estrategia de cómo deben ser ingresadas y tratadas en la base de conocimiento.

Considerando que el costo de elaborar bases de conocimiento útiles es alto **[Boi2001]**, entonces el éxito está dado en la medida que las ventajas sean superiores a los costos. Si el esfuerzo de otros se puede reutilizar sin una cantidad importante de esfuerzo extra, entonces el construir reutilizando realmente va a presentar una ventaja significativa. Y la construcción de nuevo conocimiento será eventualmente casi gratuito y ubicuo, lo que podría redundar en una nueva forma de evaluar el conocimiento y su valor en el futuro **[Ful1994]**.

El objetivo de este trabajo es experimentar con ontologías y algunos métodos de medir e identificar el reuso entre bases de conocimiento, para disponer de pautas sobre como elaborar metodologías o sistemas que permitan la integración y reutilización automática.

El planteo, en primera instancia, es modelar un conjunto de ontologías para experimentar y analizar el reuso de las definiciones y problemas en la integración de partes reutilizadas, aplicándolo en la modelación de algunos juegos de cartas tipo solitario, como un dominio acotado y más sencillo del que obtener la especificación a un conjunto de programas.

Un tema a definir es el criterio adecuado para diseñar estas ontologías de forma que sean lo más reusables posible. En principio, no se piensa que estas ontologías deban ser utilizadas para generar una aplicación que implemente los juegos sino para describir cada juego y sus reglas como un ejemplo para analizar y ver la factibilidad de reusar las especificaciones de un conjunto de dominios con gran similaridad. Con esta modelización tampoco tratamos de entender como ganar el juego, eso es un problema complejo de otra temática.

Sabemos que más que describir el “Cómo” de cada dominio analizado, se debe describir el “Qué”. Por lo tanto se delineará conceptualmente los elementos y las reglas que componen el juego sin pensar en una implementación particular e incluso tratar que el lenguaje de implementación de la ontología nos ate a modelos alejados de la realidad. Este es un tema de bastante discusión y desarrollo **[Mäd2000]**.

Incluso la elección de ciertos tipos abstractos de datos, como modelos de comportamiento en muchos ambientes, ya define la implementación interna del método de acceso a sus atributos u otros tipos que componga, como es el caso de algún contenedor (sets, bags, lists, etc.). En lo hechos si se representa algo con un conjunto de símbolos (distintos al objeto mismo), siempre se captura solo una parte de esa realidad, y esa especificación va a ir variando a medida que se incorpora mayor conocimiento sobre el dominio en cuestión.

El reto es entonces poder realizar una conceptualización de los objetos del dominio, de forma que sea lo más independiente posible del lenguaje de modelización y que soporte el mantenimiento de esas especificaciones a medida que se obtiene un mayor conocimiento sobre el dominio y es registrado en esta base de conocimiento.

Además se debe cuestionar si las descripciones son las más adecuadas para el uso que se le va a dar a la ontología. En general los lenguajes de especificación de ontologías son descriptivos, y no tienen elementos intrínsecos para describir aspectos dinámicos del dominio de conocimiento que se está conceptualizando. Para estos casos los mismos aspectos dinámicos del dominio, eventos o cambios de estado, deben ser representados con las facilidades de la herramienta que se utiliza para su representación. Por lo tanto la representación habitual de Orientación a Objetos, con métodos que representan la acción de los objetos, no es la más adecuada para este caso. De todas maneras para poder usar estas herramientas como forma de especificar programas, se debe agregar a los lenguajes existentes sentencias que permitan la manipulación del conocimiento (modificación y manipulación de datos y ranuras) y no sólo de las restricciones que deben cumplir.

Muchas representaciones de tipos abstractos de datos se hacen sobre la base de declaraciones constructivas, es decir que definen como se construye el objeto que se quiere representar. Pero para la definición de Ontologías, se prefiere un definición descriptiva. Si se dispone de un estado sobre el cual hay información de las clases y ranuras de las entidades representadas, y además se cuenta con un conjunto de instancias de esas clases representadas en la base de conocimiento, no siempre se puede inferir sobre cómo fueron construidas esas estructuras, y determinar si pudieron haber sido construidas con las reglas descriptas puede ser muy complejo o computacionalmente costoso.

En este caso supondremos que se representan las clases y ranuras necesarias para modelizar un estado cualquiera de cada uno de los juegos a analizar, y esperamos poder responder si es aplicable o no tal o cual jugada o movimiento de cartas, independientemente que una elección errónea de las instancias de las clases, permita definir un estado del juego que sea imposible en la realidad.

Asegurar que las instancias sean creadas mostrando un estado posible del juego, será responsabilidad, para este caso, del creador de la base de conocimiento, incluyendo un estado inicial del juego. Algunas otras jugadas constructivas, como ser repartir cartas en forma aleatoria entre montones de cartas, no van a ser modelados con esta herramienta ya que no dispone de esa funcionalidad, además ese tipo de jugadas son muy elementales de programar y no tienen gran interés de por sí, pero se puede identificar para una situación del juego en la que esa jugada sea aplicable.

3.1 Ontologías

Las ontologías son teorías lógicas que dan soporte a una conceptualización la cual provee una definición consensuada de un cierto dominio de conocimiento, y que pueden ser comunicadas a personas o aplicaciones, en particular para su tratamiento o deducción automatizada **[Gru1993] [Gua1995]**.

Hay muchas áreas en las que se está investigando y trabajando con herramientas aplicables en la gestión de conocimiento.

Uno de los inconvenientes que tiene este tema es que el estudioso del área se integra en una cadena de documentos que van definiendo los conceptos de una herramienta o una teoría en otra. Es un área que se está desarrollando y tiene muchos hilos relativamante trenzados, sobre los que se continúa haciendo trabajo de investigación.

El área de integración de información Web tiene ya estándares de codificación para el uso compartido de esa información facilitando la creación de *information-brokers*, filtros de información, agentes de búsqueda, etc. Muchos sistemas extienden ciertos estándares de codificación de información en Web como XML/XML-Schema y RDF/RDF-Schema, un ejemplo de eso es OIL (Ontology Inference Layer) descrito en **[Fens2000]**.

Una gran cantidad de trabajo se ha realizado para definir métodos de interoperabilidad de sistemas y/o bases de datos. En Particular **[Omel2001]** describe un conjunto de estándares y procedimientos para la integración automática de información de catálogos de ventas basados en XML, integrables por un sistema.

Hay experiencias de creación de aplicaciones a partir de Ontologías como la de **[Izum2001]** las que han mostrado el potencial, en particular por los tiempos de desarrollo y la gran reusabilidad definida intrínsecamente en el proceso. Aunque este es aún muy complicado para usuarios no vinculados a la gestión de conocimiento y depende aún de bloques precodificados para resolver muchas de las funcionalidades. Además solo se ha aplicado a desarrollos experimentales sin clientes verdaderos.

Asimismo, existe gran cantidad de herramientas para la codificación de ontologías, las cuales no todas tienen filosofías cien por ciento comunes entre ellas, pero si casi todas plantean algún mecanismo de traducción entre dos o más sistemas de codificación, con o sin perdida de información o conocimiento. Algunas de estas permiten la codificación en aplicaciones de

PC, y otras se encuentran centralizadas en sistemas con interfase web en las universidades que los crearon. Hay también aplicaciones comerciales para gestión del conocimiento con ontologías.

Una pregunta a plantearse aquí es qué tan bien se hacen esas traducciones entre sistemas cuando la filosofía de algunos es tan dispar. Indirectamente lo podemos evaluar con el caso de [Izum2001] entre muchos otros. El desarrollo hecho por el grupo de trabajo de [Izum2001]fue mínimo reutilizando gran cantidad de herramientas de otras universidades y equipos de investigación con resultados aparentemente satisfactorios.

Este reuso se ve mucho entre los diferentes grupos que utilizan conceptos, herramientas y estándares, extendiéndolos y reutilizándolos en sus propios proyectos. Se debe trabajar así ya que construir una buena base de conocimiento orientada a ontologías implica un esfuerzo considerable. Todas las herramientas tienen algunas facilidades en este sentido, ya sea para el desarrollo colaborativo de ontologías, como para compartir bases de conocimiento entre sistemas, importar y exportar en diferentes formatos y estándares.

Existen estándares de Intercambio de información entre aplicaciones como ser OKBC (Open Knowledge Base Conectivity) y el lenguaje de especificación ONTOLINGUA, y algunos basados en XML/XML-schema y extensiones, entre otros.

Las ontologías tienen diferentes enfoques en cuanto a su concepción, al menos dos tipos fundamentales, los orientados a marcos (*frames-slots*) que conceptualmente constituyen un enfoque parecido a la orientación a objetos en la definición del conocimiento, y los orientados a definiciones lógicas, basados en Description Logics, Lógica Proposicional, Lógica de Predicados y otros. Estas técnicas no son blanco o negro, entonces hay varias implementaciones que disponen de diferentes niveles mixtos, e incluso razonadores que permiten un cierto nivel de traducción de unos a otros.

Uno de los problemas fundamentales hoy es la posibilidad de integrar en una nueva base de conocimiento, dos ontologías desarrolladas en forma más o menos independiente, que aún se puede agravar al caso de desarrollo en múltiples idiomas, o dominios de negocio o conocimiento diferentes. Esto se refleja en problemas cuando se incluye conocimiento que está parcialmente registrado en ambas bases de conocimiento y se genera ambigüedad en las declaraciones.

Para ver un modelo sencillo de cómo es la realización de una ontología, en particular con la herramienta Protege-2000, se recomienda la siguiente lectura **[Nat2001]**.

En el Anexo se encuentra en “Ejemplos del Ambiente Protégé-2000” un conjunto de vistas sobre la operación de la herramienta y el uso de consultas y restricciones.

3.2 Aplicaciones para el desarrollo del software

Las Ontologías como mecanismo de especificación formal de nivel conceptual pueden ayudar al proceso de desarrollo de software, en la medida que se pueda traducir desde las especificaciones a código ejecutable de una forma más o menos automática. No es este el tema que nos trata, hay otros trabajos en esa línea un ejemplo es [Izum2001].

La ventaja del manejo de ontologías como especificación formal para sistemas se halla en las técnicas desarrolladas para integración y reuso de conocimiento que pueden aportar a la disminución de costos de mantenimiento y desarrollo de sistemas customizados para ambientes determinados de un cierto dominio de negocio.

Si el desarrollo de sistemas a medida fuera posible, con costos de mano de obra controlados, el consultor y desarrollador independiente dispondría de oportunidades para brindar soluciones en tecnologías de la información de gran calidad a bajo costo y en poco tiempo, permitiéndole competir con los grandes paquetes y los sistemas *world class* al poder proponer soluciones adaptadas a las necesidades del cliente, y disminuyendo el impacto en el cliente de la aplicación de un nuevo sistema.

Las limitaciones en el bajo costo del desarrollo con este modelo están dadas por las dificultades de reusar y de integrar especificaciones formales, con relativo poco esfuerzo. Al estar trabajando en el reuso de elementos al nivel más abstracto que la tecnología actual permite, se puede asumir que el nivel de rentabilidad del desarrollo será el mejor posible.

Este experimento plantea buscar soluciones o caminos de estudio a esta problemática.

3.3 Descripción de la investigación y los experimentos

Se tratará de explorar el reuso de reglas como mecanismos de especificación formal de programas, útiles para un concepto de reuso de nivel más abstracto y genérico, de nivel conceptual, que las tecnologías actuales, así también como para una eventual generación automática de ejecutables.

Para ello se plantea el uso de una herramienta de captura de conocimiento, donde se registrarán las reglas para un conjunto de programas. En nuestro caso, planteamos analizar programas de un cierto dominio: Juegos de Cartas (Solitarios) de computadoras. Las reglas que se registrarán son las que se cumplen para las jugadas de cada juego según el estado del mismo, entendiendo como estado a una situación arbitraria del juego en cuanto a cartas o montones dispuestos sobre el tablero.

Además se tomó un juego relativamente diferente para comparar los resultados con el resto.

El planteo de este experimento es codificar las reglas en la herramienta, para cada uno de los juegos seleccionados, llevando adelante criterios de diseño

homogéneos entre los modelos. Y luego comparar las ontologías y hacer mediciones de reuso. Las ontologías se hicieron efectivamente reutilizando reglas ya elaboradas para casos anteriores, o partiendo de ontologías elaboradas anteriormente en este experimento y quitando, agregando y modificando declaraciones. Aún así se pueden haber incluido nuevas reglas que sean comparables a otras ya usadas y que no sea trivial su identificación o el origen de su reuso.

También se identificarán ciertas reglas que tienen significado a nivel conceptual, reglas básicas de cada juego y se analizará su impacto dentro del conjunto de juegos, como planteo para el análisis de identificación automática de conceptos reusables.

3.4 Aportes de este trabajo

Este trabajo presenta un conjunto de métricas elaboradas en forma automática a partir de las ontologías procesadas. Estas métricas se elaboraron a efectos de demostrar la factibilidad y buscar soluciones a problemas relativos al desarrollo, reuso e integración de ontologías como base para el desarrollo de aplicaciones a partir de sus especificaciones.

Se muestra que la disponibilidad de reglas de un mismo dominio para reutilizar, diseñadas en forma coherente dentro de un repositorio puede dar una base para desarrollar con bajo esfuerzo nuevas ontologías, y por ende aplicaciones, de un cierto dominio de negocio.

Este experimento muestra los resultados del desarrollo de siete ontologías y la cantidad de declaraciones reusadas entre todas ellas. Muestra los resultados de esfuerzo necesarios para el desarrollo de las ontologías sobre la base de ese reuso, y plantea una primer aproximación para elegir conjuntos de declaraciones en forma automática que puedan tener una semántica especial a ser reutilizada en futuros desarrollos. Demás se hace una evaluación del resultado que se hubiera obtenido de haberse desarrollado las ontologías partiendo de un repositorio con las reglas y declaraciones de todas las otras ontologías del experimento.

3.5 Organización del Trabajo

El trabajo se divide en una explicación de cómo se definió el experimento a realizar, con su hipótesis de trabajo y mecanismos para llevarlo adelante. Luego se sigue con una explicación de los experimentos y sus resultados puntuales, terminando con un análisis de los resultados y una presentación de las conclusiones.

4 Diseño experimental

4.1 Hipótesis de los experimentos

Dado un dominio de negocio, el cual puede ser tanto un giro de negocio (fabricación, distribución y venta de los productos de un determinado nicho de negocio), como una actividad de negocio (gestión contable y financiera de la empresa), incluyendo todo el know-how implícito o explícito; Definimos como

implementación del dominio de negocio, al conjunto de especificaciones de reglas y procesos de negocio de una organización en particular.

Asumiendo que las Ontologías son una herramienta adecuada para la especificación de programas y procesos de negocio, y que si tenemos implementaciones del dominio de negocio de un conjunto grande de organizaciones, entonces:

a) Existe una forma de comparar las especificaciones de las reglas y procesos de negocio, y es posible obtener métricas automáticas de reutilización conceptual.
b) Todas las implementaciones del dominio de negocio tienen un alto grado de similaridad entre sí, a excepción de un conjunto relativamente pequeño de reglas o de detalles de las reglas, que hacen a la idiosincrasia de cada organización. Por lo tanto debe ser factible un alto grado de reuso en las especificaciones.
c) Si se dispusiera de un repositorio con todas las implementaciones del dominio de negocio de un conjunto grande de organizaciones y se debiera desarrollar la implementación del dominio de negocio de otra organización diferente a las anteriores, esta debería tener un conjunto muy grande de especificaciones que coinciden con algunas de las que se encuentran en el repositorio, y esa cantidad debiera ser mayor que si se compara la implementación en estudio con cada una de las otras implementaciones del repositorio. O sea que la disponibilidad de un repositorio importante debe garantizar un alto nivel de reuso en las declaraciones ya disponibles.
d) Además debe haber un conjunto de especificaciones, o partes de especificaciones que corresponden a conceptos importantes de la organización y el negocio, las cuales subyacen dentro de las especificaciones de la implementación (conocimiento tácito), las cuales no fueron explicitadas o no se les asignó semántica, pero que sin embargo pueden ser identificadas en forma más o menos automática para poder clasificarlas y asignarles una semántica a efectos que sean plausibles de reuso.

Si se dispone de un par de ontologías que definen o describen dos conceptualizaciones sobre un mismo dominio de conocimiento, potencialmente un dominio de negocio y sus reglas, estas ontologías deben estar constituidas por un conjunto importante de conceptos comunes (clases y restricciones) que pueden ser reutilizados entre ellas y entre otras ontologías del mismo dominio.

La integración entre dos ontologías en una tercera, debería ser trivial al menos para las reglas que son compartidas. El resto de las reglas pueden ser ingresadas caso a caso, verificando la coherencia con el resto de las reglas de la ontología que se esta construyendo. Planteado de esta manera la integración de dos ontologías se parece al desarrollo de la segunda partiendo de la primera y reutilizando o rediseñando las reglas no compartidas, obviando las reglas que se descartarían.

Se asume que subconjuntos de declaraciones (subárboles) de una base de conocimiento se repiten en otros del mismo dominio y que varios de estos deben representar conceptos y elementos importantes que se reusan entre ambas bases y que por lo tanto se les puede asignar una semántica. Se investigará la posibilidad de identificar las reglas o subconjunto de declaraciones que puedan tener un interés especial y una semántica para ser reutilizadas, y puedan ser útiles para el desarrollador, fomentando y promoviendo el reuso.

4.2 Experimentos realizados

Los experimentos consisten en modelizar las reglas de varios juegos de cartas, en particular solitarios elegidos por el autor, y un juego de cartas elegido por el supervisor para comparar.

Los modelos elegidos fueron los solitarios Carpet, Klondike, FreeCell, LaBelleLucie y MonteCarlo según la implementación de 123 Free Solitaire 2002 –version 5.9 – July 18,2002. TreeCardGames.com. Luego, Yukon con la implementación de Absol Free Solitaire Versión 8.0, Softgame Company 2002. Y por ultimo el supervisor sugiere el Robamontón, juego popular de cartas, al parecer no implementado en computadora.

4.3 Factores considerados en la elección de los experimentos

El dominio de los juegos de cartas fue sugerido principalmente por haber antecedentes en la creación de programas para familias de juegos, en ser un dominio acotado y manejable en función de los tiempos disponibles para este trabajo. Pero estos antecedentes (bibliotecas, motores de juegos, etc.) no fueron utilizados como base para los experimentos, ya que en su mayor parte no eran aplicables por tener las especificaciones en formatos no extraíbles. Se optó por redefinir los experimentos a partir de las observaciones y descripciones de los juegos.

Se acotó el conjunto de juegos a aquellos que se limitan a un solo mazo de cartas, aunque luego de la experiencia se reconoció que los juegos que utilicen varios mazos de cartas no afectarían en forma significativa los resultados generales.

4.4 Metodología aplicada

Para cada juego se consideró la modelización de las reglas que rigen sus jugadas. Se definieron las reglas que determinan si un cierto movimiento es válido para todos los escenarios posibles de instancias del juego, esto es todas las posibles situaciones de conjuntos de cartas.

En varias metodologías de desarrollo de ontologías se sugiere la elaboración de un conjunto de preguntas para la creación de nuevas ontologías, que se denotan como las "*Competency questions*". Estas preguntas, que podríamos traducir como "preguntas de capacidad", son las que el sistema experto de gestión de conocimiento será capaz de responder **[Uschol96]**. Para el caso de

este experimento las preguntas fueron si el juego estaba en un estado que permitía tal o cual movimiento de cartas.

Por lo tanto se podría definir un cierto escenario de cartas en los distintos mazos y montones del juego, y consultar al sistema si tal o cual jugada es válida de ser realizada en esa situación para ese juego en particular.

Se definieron conceptualizaciones de los elementos de cada juego, en función de su rol y/o su estructura, y se realizaron ciertos compromisos en la representación por las limitaciones del lenguaje y la herramienta. Estas definiciones y compromisos se reutilizaron a lo largo de todos los modelos desarrollados.

Un ejemplo es la carencia de modelizaciones como contenedores abstractos, eso exigió poner un índice a los objetos contenidos en una relación, cuando debían estar en orden. Esto exige tener un *slot* dentro de ciertos objetos que no es intrínseco a la conceptualización de la realidad, para poder representar conjuntos ordenados de cartas.

Ciertas tareas constructivas, como ser repartir cartas en forma aleatoria entre varios mazos, al iniciar el juego o al cumplirse una ronda, no fueron consideradas, principalmente porque la herramienta (Protégé) no soporta modelización de reglas que cambian las instancias de conocimiento dentro de la base, además que esas operaciones no son las más interesantes del juego y en general no aportan a la lógica y reglas fundamentales del juego, por lo tanto no afectan mayormente los resultados. Son simplemente mecanismos de dar aleatoriedad a la distribución de cartas en uno o varios montones sin necesidad de precondiciones complejas.

Para cada conjunto de reglas de cada juego, la ontología fue elaborada reutilizando reglas y definiciones de otros juegos modelados anteriormente por lo que se puede asegurar un cierto nivel de reuso. Incluso, a medida que se disponía de varios juegos, cada vez que se quería modelar uno nuevo se analizaba factores de similitud aparente para basarse en uno de ellos como punto de partida. También se podía copiar partes de otros no usados directamente como base de desarrollo.

Se hicieron métricas de tiempos de desarrollo, aunque están afectadas por las dificultades técnicas que se tuvo con la herramienta, debidas al desconocimiento y falta de experiencia con la misma. Como comentario, es de notar que la última ontología fue modelada en algo menos de tres horas, ya con experiencia y con modelos previamente elaborados de los que partir. Los tiempo incluyen las pruebas para verificar que las reglas y consultas eran capaces de responder sobre situaciones particulares del juego.

Ya que la herramienta de modelización codifica sus bases de conocimiento en un lenguaje derivado de KIF **[KIF1998] [Gin1991]**, este a su vez derivado al menos en el formato de LISP**[Moy1991]**, define que las declaraciones o predicados tengan el formato: (operador operando1 [operando2 ...]) que implica la ejecución de las operaciones en formato prefijo. Así por ejemplo la

suma de dos números se codificaría (+ 1 2) cuyo resultado sería 3. De la misma manera se pueden anidar predicados: (and(> valor1 0) (= valor2 valor3)).

Esto hace que el conjunto de predicados anidados se pueda analizar y modelar como un árbol sintáctico **[Aho1986]**. En particular toda la ontología puede ser modelada de esta manera, incluyendo tanto declaraciones de reglas, consultas y restricciones, la representación de instancias particulares de elementos de los modelos, declaraciones de las clases y ranuras (slots) sobre los que se basan.

La herramienta Protégé tiene archivos separados para las definiciones de reglas, restricciones e instancias, de las declaraciones de las clases y ranuras. Como las declaraciones de las clases o marcos y ranuras están implícitas en las referencias de los predicados de las reglas y en las declaraciones de las instancias que declaran los valoras para las ranuras, se consideró que agregar los archivos de declaración de clases no añadía información extra a las reglas del juego, pues en particular si hay una declaración de las clases y no era inicializada de alguna manera en alguna de las instancias, es simplemente porque la clase o ranura no es utilizada o se usan sus valores por defecto. Toda la información de la base de conocimiento es codificada de forma que puede ser incorporada en el árbol sintáctico de la ontología.

Sobre la base de las hipótesis definidas se realizaron una serie de experimentos. Las bases de conocimiento generadas fueron procesadas por un programa (Protégé Analyzer) desarrollado en el experimento para este propósito, que construye el árbol sintáctico y compara las estructuras del árbol para dos reglas dadas. Se compararon todas las reglas y se obtuvieron informes de reutilización en función de subárboles similares entre las diferentes ontologías. Es de notar que si bien hubo reuso en las reglas, por la forma como fueron construidas las ontologías partiendo de otras previas, la comparación de subárboles trata de ser más amplia pues compara similitud entre árboles, que incluirá tanto al reuso explícito de estructuras y reglas como a la similaridad o reuso implícito de otras estructuras que fueron agregadas en el proceso de desarrollo.

Esta información se ingresó a una base de datos (Protégé Database) también desarrollada para este experimento, donde se analizó la cantidad de veces que cada subárbol de cada ontología es reutilizado en otras ontologías, junto con la información de tamaño y una declaración sintáctica del subárbol considerado. Sobre la base de esta información se desarrollaron las conclusiones de este experimento.

Para la comparación de los posibles resultados de un desarrollo basado en un repositorio o base de bases de conocimiento se desarrolló una herramienta (Protégé Merge) capaz de hacer la unión de dos o más ontologías, de forma que el resultado, más allá de que tenga significado o aplicación semántica, contenga todos los subárboles de ambas estructuras.

Se realizó dos experimentos, para el mejor y peor caso de índices de reutilización, combinando todas las ontologías menos una, y se analizó el resultado contra la ontología no incluída en la base para comprobar el nivel de reuso consolidado.

4.5 Descripción del ambiente Protégé

Protégé-2000 Versión 1.7 **[Protégé] [Noy2000]** es un ambiente visual de diseño y registro de ontologías, orientado a *frames* y *slots* desarrollado en Java y que puede correr en forma independiente en un PC. Dispone de un conjunto importante de plugins con orígenes diversos. En particular el que es más aplicable es el PAL plugin **[PALplug]**. Consiste en un agregado que permite el ingreso de restricciones y consultas declaradas en un subconjunto de KIF que este módulo soporta.

Protégé permite ingresar las definiciones estructurales de los conceptos con los que se va a trabajar como clases y ranuras. A su vez, a efectos de facilitar el ingreso de datos, cada clase se asocia a un Form, y cada ranura se asocia a un SlotWidget (objeto de edición del slot según el tipo de dato que sea), de forma que la interfase al usuario pueda ser diseñada y modificada para facilitar el ingreso de instancias al usuario.

La interfase permite mucha flexibilidad y facilidad para el ingreso de datos uno a uno, pero no hay mecanismos prácticos para el ingreso masivo de información (e.g. un estado completo de un juego de cartas, con sus 52 cartas, y los diferentes montones que lo componen, y los vínculos de orden y pertenencia de las cartas con esos montones). En general para el ingreso masivo se optó por ingresar la información con otros medios directamente dentro de los archivos de cada proyecto.

El PAL plugin, agrega funcionalidad a las reglas y restricciones básicas de Protégé mediante la incorporación de un editor de reglas, verificador sintáctico, la ejecución y verificación de las reglas y consultas.

Cada regla o consulta se compone de un nombre (:PAL-NAME), una descripción (:PAL-DESCRIPTION), un rango que es la declaración de las clases y eventualmente las ranuras que estarán involucradas (:PAL-RANGE), y una declaración de la regla o consulta en KIF (:PAL-STATEMENT).

Las reglas se pueden asociar a clases o ranuras, como restricciones, lo que hace opcional la declaración de esas clases en la ranura de rango de la regla. De todas maneras para completitud siempre se declaró el rango de las clases para las instancias involucradas en Consultas y Restricciones.

El siguiente es un ejemplo de una regla, en particular una restricción que verifica que todas las cartas si pertenecen a un montón deben estár ordenadas y numeradas, necesario para poder hacer referencia a la carta de una posición dada en el montón:

```
([klondike_57210] of :PAL-CONSTRAINT
  (:PAL-RANGE "
    (defrange ?card1 :FRAME Card)
```

```
    (defrange ?card2 :FRAME Card)
    (defrange ?deck :FRAME Deck)")
  (:PAL-NAME "X_cards_of_the_deck_order")
  (:PAL-STATEMENT "
    (forall ?card1
      (=>
        (own-slot-not-null deck_of_card ?card1)
        (or
          (=
            (coerce-to-integer
              (card_order_in_deck ?card1))1)
          (exists ?card2
            (and
              (/= ?card1 ?card2)
              (deck_of_card ?card1
                (deck_of_card ?card2))
              (=
                (coerce-to-integer
                  (card_order_in_deck ?card1))
                (+
                  (coerce-to-integer
                    (card_order_in_deck ?card2))1)))))))")
  (:PAL-DESCRIPTION "Toda carta, si pertenece a un deck, o es la primer carta del deck,
o existe otra carta diferente de la primera pero que coinciden en el deck y que
nuestra primer carta le sigue en orden a esta."))
```

El razonador puede verificar y evaluar las reglas, consultas o restricciones, indicando si son correctas, dando advertencias y/o presentando las instancias que verifican las consultas o que transgreden las reglas.

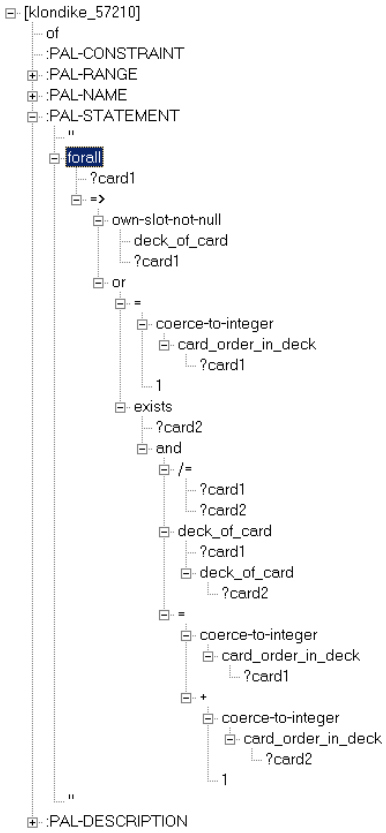
Desde cualquier ventana de instancias se puede hacer un drill-down a las otras instancias asociadas en las diferentes relaciones, lo que hace muy práctica la navegación entre los conceptos. Además es capaz de mantener relaciones y sus relaciones inversas en forma automática.

Como contrapartida, no tiene muy desarrolladas las verificaciones sintácticas, ni tampoco indicadores útiles para que el desarrollador pueda encontrar el origen de los problemas. Además tiene problemas para manejar reglas y consultas grandes. Muchos de estos problemas probablemente se subsanen en versiones posteriores.

Por mayor información se sugiere revisar el Anexo: Ejemplos del Ambiente Protégé-2000.

4.6 Elementos de medición

Para cada base de conocimiento correspondiente a cada juego se realizó el árbol sintáctico que modela toda la base. Se realizó un programa especial (Protégé Analyzer) que identifica los tokens del lenguaje y los organiza en un árbol, para que puedan ser tratados electrónicamente. Como ejemplo la regla anterior la representamos según la vista de árbol como se puede apreciar en la aplicación.



Una de las hipótesis es que subárboles de una base de conocimiento se repiten en otros del mismo dominio y que deben representar conceptos y elementos importantes que se reusan entre ambas bases.

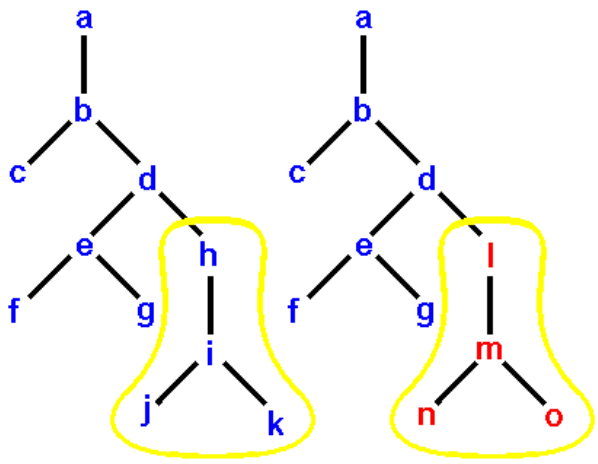
Para ello se consideró que se debía comparar los árboles sintácticos de las bases de conocimiento identificando todos los subárboles que se repetían. El algoritmo de comparación que se eligió es el que se describe a continuación:

> *Para cada nodo **a** del árbol **A** y Para cada nodo **b** del árbol **B**, Se compara el subárbol con raíz en **a** con el subárbol con raíz en **b**. Si el token de **a** coincide con el token de **b** incremento la cuenta de tokens para el subárbol y se continúa comparando todos los hijos de **a** con todos los de **b** en todas las combinaciones posibles, para cada hijo de **a** se responde con la comparación con el hijo de **b** que dio mayor valor. Cuando todo el subárbol fue analizado registro la cuenta y el subárbol formado con los nodos que coincidieron.*

Por lo tanto lo que devuelve es la cuenta de tokens máxima que coinciden entre dos subárboles considerando solo las ramas que tienen nombres iguales o que definen ambas una instancia de una clase.

Para el siguiente ejemplo donde tengo dos subárboles de once nodos cada uno y suponiendo que los token de h y l son diferentes, el resto del subárbol no es comparado, aún cuando m e i, y los otros nodos continuando por la rama pudieran tener igual token al correspondiente del otro subárbol.

En este caso la cuenta de reuso daría siete tokens. Además para este algoritmo no importa el orden de los subárboles hijo pues va a comparar todos con todos.



Podría suceder que si un subárbol A tiene dos subárboles similares y en cambio el subárbol B tiene dos subárboles diferentes, uno igual a los dos de A, que la comparación de los subárboles se cuente solo la comparación al subárbol de B que dé la mayor similaridad, pero lo que estamos considerando es el nivel de reutilización y no tanto la igualdad de la estructura de los árboles. En este caso si dos ramas del subárbol A coinciden con una rama del arbol B, se van a contar ambas ramas.

Se considera la comparación de todos los nodos hijos del nodo *a* del subárbol A con todos los nodos hijos de *b* del subárbol B porque se notó, primero que el orden y tipo de los tokens en un predicado (nodo) está determinado por el operador, y luego que cuando los operandos son del mismo tipo, el orden de estos no afecta el resultado.

Como el operador determina los tipos de los datos se optó por no comparar por tipos de instancias o los nombres que se usan para determinarlos o asociarlos a rangos. Entonces como todas las referencias a instancias (variables ligadas) son de la forma *?nombre*, se hizo que el programa de comparación diera como verdadero la comparación entre dos declaraciones de instancia y continuara con la comparación, independientemente del nombre de la instancia.

A medida que el programa Protégé Analyzer va comparando todos los nodos de cada árbol, va creando una base de datos con la cantidad de tokens reutilizados (tamaño) asociado al subárbol analizando siempre que al menos haya un nodo similar.

Luego la información se ingresa en otro programa creado para este experimento, el Protégé Database Analyzer que compila la información y lleva cuenta de cuantas veces cada subárbol es encontrado dentro de la lista entregada por el programa anterior.

Los resultados son una lista de subárboles, con su tamaño y la cantidad de veces que se encuentran para cada combinación de dos bases de conocimiento. Esto nos permite analizar tanto los datos estadísticos de la cantidad de los subárboles reutilizados como de la estructura de las ramas encontradas, más allá de la reutilización explícita que se haya realizado en el proceso de creación de las bases de conocimiento.

5 Resultados experimentales

Las bases de conocimiento desarrolladas tienen las siguientes dimensiones en cantidad de tokens según el programa Protégé Analyzer.

Base	**Tamaño**
Klondike	2941
LaBelleLucie	2038
MonteCarlo	1798
Robamonton	1373
Carpet	1412
FreeCell	3152
Yukon	2209

Para cada par de bases de conocimiento se hizo la comparación con Protégé Analyzer, 21 en total, y se obtuvo, entre otras, una gran regla que coincidía con la comparación de toda la base de conocimiento desde la raíz con la otra. Esta regla fundamental (Arbol Máximo de Comparación) es la que se tomó para considerarla como principal métrica de reutilización. Los datos obtenidos fueron los siguientes:

Cantidad de Token Comunes

		LaBelle Lucie	**Monte Carlo**	**Roba-monton**	**Carpet**	**FreeCell**	**Yukon**
		2038	**1798**	**1373**	**1412**	**3152**	**2209**
Klondike	**2941**	1857	1220	1054	1086	2538	1196
LaBelleLucie	**2038**		1179	1025	1056	1748	1030
MonteCarlo	**1798**			943	934	1257	955
Robamonton	**1373**				1060	1033	1028
Carpet	**1412**					1047	1035
FreeCell	**3152**						1138

Esta métrica es lo bastante buena como para tener una primer aproximación al nivel de reutilización conceptual de las bases de conocimiento. El algoritmo da peor que una comparación más minuciosa que permitiera además comparar subárboles salteándose algún nodo que no coincide en la comparación pero que algunos niveles más profundo vuelve a mantener su igualdad con el otro subárbol al que se compara o con una interpretación semántica de algunas de sus declaraciones. Por lo tanto el nivel de reuso real total puede ser mayor o igual al que se obtiene con este algoritmo. Razones de tiempo no permitieron mejorar el algoritmo para considerar también esta opción.

Los porcentajes de reutilización obtenidos entonces son los siguientes:

Porcentage de Tokens reusados entre dos Juegos cualesquiera.

		LaBelle Lucie	Monte Carlo	Roba-monton	Carpet	FreeCell	Yukon
		2038	1798	1373	1412	3152	2209
Klondike	**2941**	91%	68%	77%	77%	86%	54%
LaBelleLucie	**2038**		66%	75%	75%	86%	51%
MonteCarlo	**1798**			69%	66%	70%	53%
Robamonton	**1373**				77%	75%	75%
Carpet	**1412**					74%	73%
FreeCell	**3152**						52%

Este cuadro muestra el mayor nivel de reutilización entre dos juegos, o sea el nivel mayor entre el tamaño de un juego dividido por el tamaño del mayor subárbol común de ambos juegos.

Es de notar el rango de reutilización de los juegos, donde por ejemplo “La Belle Lucie” reutiliza el 91% de las sentencias con respecto a las del juego “Klondike”, mientras que con “Yukon” es de un 51%.

Esto es más curioso aún si se ve ambos juegos “Yukon” y “Klondike” que desde el punto de vista del jugador tienen una similitud muy importante como para esperar que entre ambos hubiera una nivel de reutilización muy grande, sin embargo el “Yukon” carece de ciertas estructuras del juego y un conjunto importante de restricciones que sí tiene “Klondike”, por lo que la similitud aparente para quien no haya analizado en profundidad los juegos puede no coincidir con la realidad.

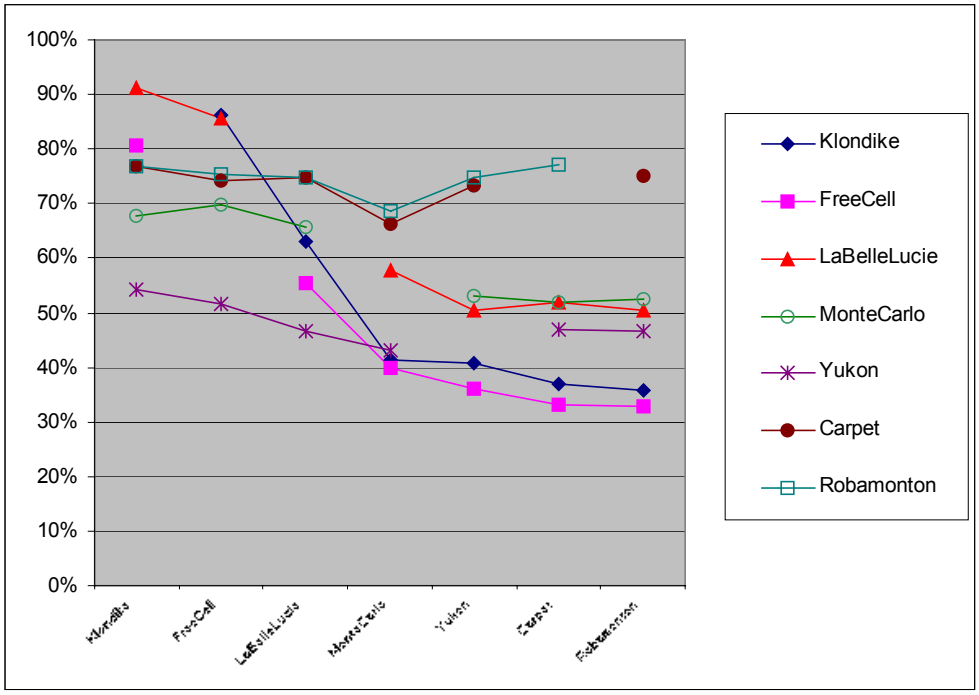
Vemos que el porcentaje de reutilización de un juego en otro no es conmutativo, por lo que se construyó el siguiente cuadro:

Porcentaje de Tokens de la base de la columna que se encuentran en la base de la fila

	Klondike	FreeCell	LaBelle Lucie	Monte Carlo	Yukon	Carpet	Roba-monton	MAX	MIN
Klondike		86%	63%	41%	41%	37%	36%	**86%**	**36%**
FreeCell	81%		55%	40%	36%	33%	33%	**81%**	**33%**
LaBelleLucie	91%	86%		58%	51%	52%	50%	**91%**	**50%**
MonteCarlo	68%	70%	66%		53%	52%	52%	**70%**	**52%**
Yukon	54%	52%	47%	43%		47%	47%	**54%**	**43%**
Carpet	77%	74%	75%	66%	73%		75%	**77%**	**66%**
Robamonton	77%	75%	75%	69%	75%	77%		**77%**	**69%**
MAX	**91%**	**86%**	**75%**	**69%**	**75%**	**77%**	**75%**		
MIN	**54%**	**52%**	**47%**	**40%**	**36%**	**33%**	**33%**		

El cuadro en un principio no se ordenó de esta manera, se notó que había una relación entre los juegos de similaridad que de alguna manera permitía segmentarlos por una aparente reutilización entre ellos. Al graficar el nivel de reutilización entre los juegos se fueron intercambiando filas y columnas para

llevar las líneas de gráficos a una curva más o menos bien definida. El resultado fue el siguiente:



El cuadro anterior muestra para cada juego (línea de puntos) el porcentaje de reuso de las declaraciones de los juegos contra los juegos definidos abajo.

Sobre esta base vemos que el Robamontón y Carpet son juegos ampliamente distintos a los otros, y al mismo tiempo relativamente sencillos en sus reglas como para que tengan un nivel de reutilización con los otros de alrededor del 70%. Es de notar que el promedio de reutilización es 59,7% similar a lo identificado en **[Coh1999]**. Por lo tanto correlaciones de reutilización podrían ser aplicables para la comparación de bases de conocimiento y una caracterización de ellas para determinar su aplicabilidad de reuso.

Ahora, si consideramos que cualquiera de los juegos se pudo haber desarrollado a partir de una base de bases de conocimiento (repositorio), entonces el nivel de reutilización debería ser mayor que el mayor nivel de reutilización de cada uno de los juegos.

Se hicieron dos bases con todas las ontologías (menos una) a efectos de comparar los resultados con la que quede afuera. Todos menos “La Belle Lucie” quedó de 4555 tokens y Todos menos “Yukon” tiene 4077. Nótese que con la fusión de las bases de conocimiento se obtuvo una nueva base de conocimiento que si bien puede no tener sustento conceptual, sí tiene todas las reglas aportadas por todas las bases de conocimiento involucradas, y que en general las bases fusionadas crecieron mucho menos que la concatenación simple de las bases de conocimiento.

Se consideraron estas bases, “La Belle Lucie” y “Yukon”, pues son el mejor y el peor caso de los máximos de reutilización, para ver el impacto que tendría la reutilización a partir de un repositorio de reglas que fuera la unión de muchas reglas similares, potencialmente de un dominio determinado de interés. El análisis de “Yukon” y de “La Belle Lucie” contra la suma de los otros juegos da aproximadamente igual que el mejor caso, por lo que no hay mejora significativa. Habrá que estudiar las causas más adelante, o ver si el criterio de similaridad de subárboles extendido que consiste en profundizar un poco por las ramas no iguales para ver si la similaridad de los árboles se vuelve a sincronizar, cambia la forma en que se puede considerar los coeficientes de reutilización.

Analizando los subárboles de reglas que se encuentran reutilizados encontramos que en general se destaca un subárbol de gran tamaño, el árbol máximo de comparación, que coincide con la comparación de toda una base de conocimiento con otra. Y sobre esta única gran comparación es con la que hemos basado los estudios presentados antes en este documento.

Pero el ProtégéAnalizer también registra todos los subárboles que son reutilizados a lo largo de toda la base de conocimiento. Por lo que decidimos estudiar esos subárboles para sacar algunas conclusiones.

Se realizó la fusión de todas las bases en una y esta se comparó consigo misma para obtener todos los subárboles. Con el resultado se analizó la distribución de los subárboles reutilizados en función de su tamaño, la cantidad de subárboles de ese tamaño reutilizados, y la cantidad de subárboles distintos para ese tamaño.

La fusión (merge) de todas las bases de conocimiento dio como resultado una base de 4797 tokens, aproximadamente igual a la suma del tamaño de la mayor base con la menor.

Se analizaron los subárboles identificados en la comparación. Luego del subárbol máximo de comparación, para la fusión de todas las bases, en orden decreciente de tamaño, aparecen subárboles de dimensiones importantes pero que son solo una fracción de la base de conocimiento. Estos subárboles de 150 tokens o menos son en general instancias completas de restricciones o consultas que se han reutilizado, por ejemplo reglas generales que describen el comportamiento de un mazo o un montón de cartas que se reutiliza en muchos de los juegos. Luego comienzan a aparecer las restricciones o consultas solas, mezcladas con declaraciones o partes de las declaraciones fuera del contexto de donde fueron declaradas, en la base de conocimiento.

Al nivel de tamaño 58 aparecen las declaraciones de la instancia del objeto más grande del juego, el mazo de cartas, que incluye sus 52 cartas y algunos atributos. Hasta el nivel 31 de tamaño solo aparecen reglas y consultas o pedazos de estas, mezclados con alguna declaración de instancias de juego (montones de cartas con sus listas de cartas correspondientes). En el nivel 30 aparecen las primeras declaraciones de rango mezcladas también con

pedazos de restricciones. Así van apareciendo pedazos de restricciones, mezclados con declaraciones de rangos, o sus partes.

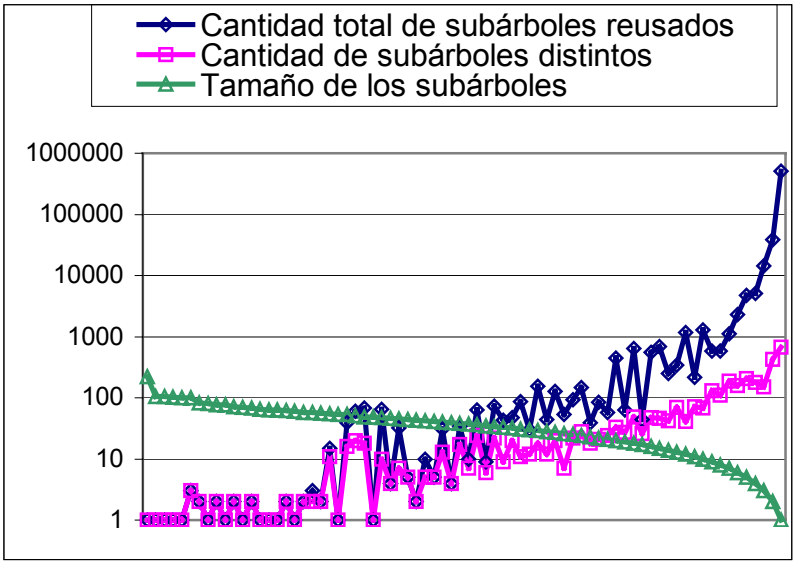
Un problema que se notó es que con el algoritmo utilizado aparecen en la lista tanto el mayor subárbol encontrado como cualquiera de los subárboles menores dentro del primero lo que causa confusión al aumentar considerablemente la cantidad de subárboles a considerar. En próxima etapa se modificarán los algoritmos para saltearse las ramas de subárboles que claramente ya están incluidos en subárboles previamente evaluados.

Otro problema es como identificar subárboles que tengan un significado conceptual de forma que sean plausibles de reutilización en forma automática.

Para pequeños proyectos esto parece ser fácil de implementar, al menos en forma manual, pero para grandes desarrollos la reutilización debería hacerse a nivel de componentes conceptuales, con ayuda del sistema para encontrar los conceptos aplicables. Tratar de identificar componentes conceptuales a partir de los subárboles reutilizados parece ser bastante difícil de automatizar pues hay que dar una interpretación semántica a cada una de las construcciones de subárboles, al menos a las más reutilizadas.

En primera instancia, una fórmula que dio bastante buen resultado para encontrar rápidamente subárboles útiles, y con algún significado conceptual, es buscar por el mayor de un coeficiente que es el producto de la cantidad de reusos registrados del subárbol por el tamaño del mismo, y filtrando por tipos de objetos (declaraciones de rango, de restricciones, de instancias, de consultas, totales o parciales).

Analizamos luego la distribución de subárboles reutilizados en función de su tamaño y consideramos para cada tamaño de subárbol, la cantidad total de subárboles reutilizados y la cantidad de árboles diferentes reutilizados.



Vemos que entre los tamaños 42 y 56 hay un pico tanto en variabilidad de los subárboles reutilizados como en la cantidad de reusos. Estudiando los subárboles vemos que algunos de los más interesantes conceptualmente se encuentran allí, por ejemplo subárboles que determinan si es posible mover una carta a un “suit stack” o sea un montón ordenado de cartas de un mismo mazo, etc. Vemos también que no se previó que hay consultas muy similares a restricciones que cambian el operador raíz de “exists “ a “findall” y que se ven como reglas diferentes.

Esto da un criterio para separar ese grupo de árboles de declaraciones cuya variabilidad y cantidad de reusos es mayor a la de otros en el entorno, para ser analizados en forma manual como propuestas para identificar conceptos reusables que no fueron especificados de forma explícita. Se puede mejorar la búsqueda si se hace alguna interpretación de los subárboles identificados eliminando los que contienen declaraciones que muestran que ya se les ha asignado alguna semántica al ser una declaración de una regla o consulta.

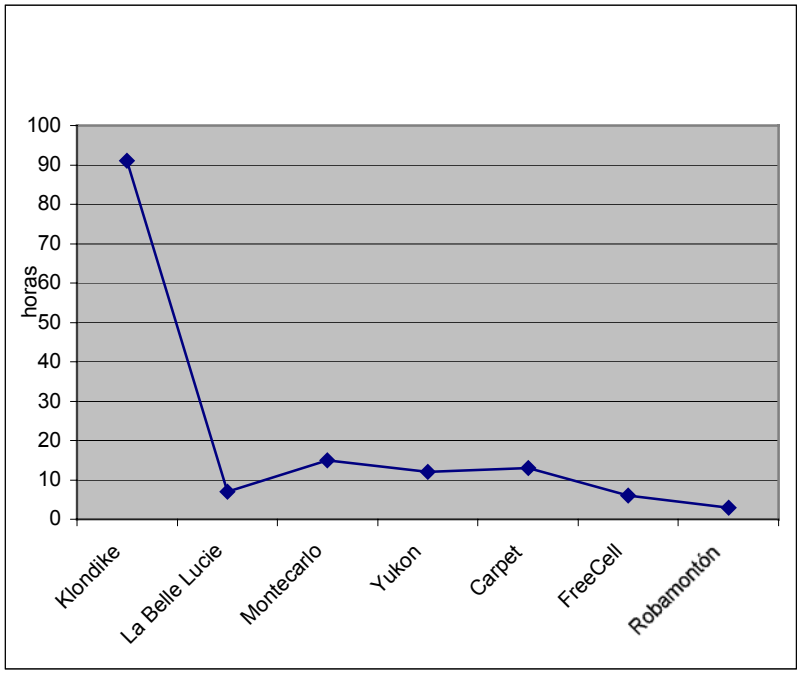
Esto implica que una primer aproximación para separar algunos de los conceptos implícitos reusados dentro de un repositorio de reglas, podría basarse en analizar algunos aspectos de tamaño, cantidad de reusos y cantidad de variaciones, que se diferencian de otras declaraciones del entorno.

También se midieron las horas/hombre para desarrollar cada uno de los modelos. Los resultados son los siguientes:

Modelo	**horas**
Klondike	91
La Belle Lucie	7
Montecarlo	15
Yukon	12
Carpet	13
FreeCell	6
Robamontón	3

Las ontologías fueron desarrolladas en ese orden. La Klondike tuvo el costo de ser la primera, con la que se estaba aprendiendo, además otras dificultades como un equipo de computación sin la capacidad de memoria y procesador necesarios y la falta de experiencia afectaron los tiempos. Se puede ajustar un poco esas 91 horas considerando que hubo 14 horas de planificación, 6 de diseño, 36 de codificación y 22 horas de retrabajo debido a que se consideró que el diseño era incorrecto. El resto de los modelos no tuvieron tantos inconvenientes y las horas registradas son mayormente de codificación. Destaca el tiempo bajo que se necesitó para el modelo LaBelleLucie, debido a tener pocas reglas y que se reusó bastante de klondike (91%). Los modelos Carpet, FreeCell y Robamontón se desarrollaron juntos y reutilizan principalmente el concepto de celda (Cell) para poner una carta en forma temporal. Carpet tuvo el costo de desarrollar el concepto de celda y sus reglas, que luego con pocas modificaciones se reutilizó en FreeCell y Robamontón. El tiempo bajo de Robamontón se debe

al reuso de las ontologías anteriores, a la mayor experiencia con la herramienta, y a que se había desarrollado una herramienta para facilitar el manejo de reglas complejas.



No están considerados otros tiempos necesarios para el desarrollo de las aplicaciones con que se analizaron las bases de conocimiento, y otras tareas de depuración.

6 Conclusiones

Este experimento de reuso de ontologías como especificaciones formales muestra que se puede comparar especificaciones y obtener métricas de reuso en forma automática.

Es posible obtener obtener un nivel de reuso de hasta el 91%, para dos aplicaciones diferentes dentro de un mismo dominio.

En la medida que se disponga de un conjunto importante de reglas y declaraciones para reutilizar, los esfuerzos de desarrollo pueden disminuir en forma importante. En particular un sistema que permita sugerir reglas a ser aplicadas en el proceso de desarrollo puede resolver los problemas presentados por [Coh1999], donde declara la dificultad de mantener alineadas varias ontologías si los desarrolladores no disponen fácilmente de acceso a los conceptos que necesitan para ser reutilizados.

La reutilización de un conjunto de reglas consolidadas de todos los modelos elaborados, devuelve un coeficiente de reutilización que es mayormente idéntico al nivel de reutilización obtenido con el mejor modelo dentro de la base. Esto puede estar afectado por la forma como se construyeron las ontologías, o la imprecisión de las métricas analizadas, pues era de esperar

un coeficiente de mejora en el experimento al identificar nuevas reglas que eran la suma de varias ontologías.

Es posible una cierta caracterización para preseleccionar un conjunto de los subárboles reutilizados que tengan una semántica interesante dentro del dominio de conocimiento al que se aplican, y que puedan extraerse y asignarles la semántica para mayor facilidad del desarrollador. Aún no tenemos elementos para automatizar el proceso, pero un conjunto de métricas de los subárboles comparados nos permite preseleccionar algunos de los que más probablemente puedan ser aplicados. Luego un tratamiento caso por caso los puede identificar y clasificar.

Se ve la necesidad de extender estas métricas con comparación extendida, información semántica y sintáctica para analizar los subárboles y afinar las métricas.

El universo de reglas analizados es chico, con pocas reglas y de baja complejidad. Hay que probar las hipótesis con reglas de negocio reales o con algún otro dominio para comparar los resultados.

Este tema necesita más estudio, mejorando las métricas y con otros experimentos necesarios para validar estas conclusiones. Se espera realizar estas comprobaciones en futuros estudios dentro de esta misma área de gestión de conocimiento.

7 Glosario

Arbol Máximo de Comparación: es el árbol de tokens resultante de extraer la estructura común de dos árboles sintácticos, a partir de las dos ontologías correspondientes, aplicando en concepto de reutilización conceptual.

Common Criteria: Criterios estándar de seguridad informática.

Dominio de Negocio: Un área de conocimiento asociada a un giro o una actividad de negocios.

Frame: Marco o Clase

Gestión del Conocimiento: Conjunto de técnicas y herramientas para la administración, distribución y uso de conocimiento.

Information-broker: Agente de gestión de información, unidad de software que permite la gestión, recolección, identificación, clasificación y búsqueda de información de múltiples fuentes en una base de conocimiento consolidada.

Marco: Estructura para la representación de conocimiento basada en esquemas de clases que describe una concepción común a un conjunto de objetos que deben ser representados. Frame.

Ontología: es una teoría lógica que da soporte a una conceptualización la cual provee una definición consensuada de un cierto dominio de conocimiento, y que pueden ser comunicadas a personas o aplicaciones, en particular para su tratamiento o deducción automatizada.

Ranura: Conceptualización de los atributos de objetos a ser representados mediante marcos. Slot.

Regla de Negocio: Regla o restricción que determinan la forma adecuada de llevar adelante una o más actividades de negocios.

Reuso: Reutilización (anglicismo).

Reutilización Conceptual: Por cuanto se busca analizar la máxima reutilización práctica (a nivel conceptual), se considera reutilización toda especificación que se repita entre dos unidades ya sea por reutilización explícita o por re-especificación de un concepto común.

Reutilización Productiva: Es la reutilización de líneas de código o especificaciones, de forma que no tengan un costo en horas hombre asociadas a su incorporación a un sistema.

Slot: Ranura.

Referencias

[Pin1999] Some Issues on Ontology Integration. H. Sofia Pinto, A Gómez-Pérez, J.P.martins. Proceedings of the IJCAII99's Workshop on Ontologies and problem solving methods: Lessons Learned and Future Trends.

[Boi2001] Ontologies and the Knowledge Acquisition Bottleneck. Mihai Boicu, Gheorghe Tecuci, Bodgan Stanescu, Gabriel C. Balan, Elena Popovici. Learning Agents Laboratory. Department of Computer Sciences. George Mason University. VA, USA.

[Ful1994] Why Post Industrial Society Never Came. Steve Fuller. ACADEME November-December 1994. pp 22-29.

[Mäd2000] Representation Language-Neutral Modeling of Ontologies. A. Mädche, H. P. Schnurr, S. Staab & R. Studer, Modellierung 2000.

[Gru1993] Tom R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Padua Workshop on Formal Ontology, March 1993.

[Gua1995] Ontologies and Knowledge Bases: Towards a Terminological Clarification. Nicola Guarino, National Research Council, LADSEB-CNR, Pavoda Italy. Peirdaniele Giaretta. Institute of History of Philosophy, University of Padova, Italy. 1995.

[Fens2000] OIL in a Nutshell; D. Fensel; I Horrocks; F. Van Harmelen; S. Decker; M. Erdmann; and M. Klein.

[Omel2001] A Two-Layered Integration Approach for Product Information in B2B E-Commerce. Borys Omelayenko and Dieter Fensel.

[Izum2001] Building Business Applications By Integrating Heterogeneous Repositories Based on Ontologies, Noriaki Izumi and Takashira Yamaguchi.

[Nat2001] Ontology Development 101: A Guide to Creating Your First Ontology, Natalya F. Noy, Deborah L. McGuinness. Stanford University, CA, USA.
http://protégé.stanford.edu/publications/ontology_development/ontology101.html

[Uschol96] Building Ontologies: Towards a Unified Methodology. Mike Uschold. AIAI-TR-197. September 1996. University of Edimburg.

[KIF1998] M.R.Genesereth. Knowledge Interchange Format. Draft Proposed American National Standard (dpans). ncits.t2/98-004. http://logic.stanford.edu/kif/dpans.html. http://www-ksl.stanford.edu/knowledge-sharing/kif/

[Gin1991] Knowledge Interchange Format. The KIF of Death. M.Ginsberg. AI Magazine, 5(63), 1991.

[Moy1991] John A. Moyne. LISP: A first language for computing. Van Nostrand Reinhold. NY. USA, 1991.

[Aho1986] Compilers: Principles, Techniques and Tools. Chapter 5. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. Bell Telephone Laboratories. Addison Wesley Publishing Company, MA, USA, 1986.

[Protégé] Protégé web site: http://protégé.stanford.edu.

[Noy2000] N. F. Noy, R. W. Fergerson, & M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France, . 2000.

[PALplug] PAL plugin. PAL Constraints and Queries Tabs. http://protege.stanford.edu/plugins/paltabs/PAL_tabs.html. The Protégé Axiom Language and Toolset ("PAL") http://protege.stanford.edu/plugins/paltabs/pal-documentation/index.html.

[Coh1999] Does Prior Knowledge Facilitate the Development of Knowledge-based Systems?. Paul Cohen, Vinay Chaudri, Adam Pease, Robert Schrag. Department of Computer Sciance, University of Massachusetts, 1999.