COMBAS: una herramienta para el análisis de procesos con información semántica*

E. González-López de Murillas, J. Fabra, P. Álvarez y J. Ezpeleta

Instituto de Investigación en Ingeniería de Aragón (I3A)
Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
María de Luna 3, E-50018 Zaragoza (España)
{edugonza,jfabra,alvaper,ezpeleta}@unizar.es

Resumen Desde la concepción de un sistema hasta su puesta en funcionamiento de una manera fiable hay un largo camino. Existen numerosas herramientas y técnicas que facilitan la tarea. En el contexto actual, la adición de información semántica en la especificación, análisis e implementación de sistema basados en tecnologías Web hace que las técnicas y tecnologías utilizadas deban ser adaptadas/extendidas para considerar estos aspectos.
En este trabajo se presenta COMBAS (COmprobador de Modelos BAsado en Semántica), un entorno que integra herramientas para la verificación de modelos (*model checking*) de procesos que incluyen información semántica. COMBAS permite, a partir de un modelo descrito mediante una clase de Redes de Petri de alto nivel anotadas con información semántica, las U-RDF-PN, la generación de su grafo de alcanzabilidad, la edición de fórmulas en lógica temporal para la expresión de las propiedades que se desean verificar en el modelo y la visualización de resultados, incluyendo la posibilidad de navegar por el grafo de estados alcanzables y visualizar sus anotaciones. Como caso de estudio se presenta la aplicación de estas herramientas al *First Provenance Challenge*, un caso de *workflow* tomado del mundo de la computación científica.

Palabras clave: Procesos con Anotaciones Semánticas, RDF, Redes de Petri, *Model Checking*.

1. Introducción

Para automatizar el ciclo de vida de los procesos de negocio es necesaria la descripción semántica de los diferentes elementos que los constituyen (tareas, estructuras y condiciones de control, datos, mensajes intercambiados entre procesos, etc.). Estos procesos descritos semánticamente reciben el nombre de *Procesos de Negocio Semánticos* (PNS). Hasta la fecha, los principales esfuerzos se han centrado en definir lenguajes de descripción semántica (RDF, WSMO u

* Este trabajo se ha financiado parcialmente gracias al proyecto de investigación del Ministerio de Ciencia e Innovación TIN2010-17905.

OWL, entre otros) y ontologías del dominio de aplicación, y en integrar estas soluciones en los lenguajes de procesos de negocio más utilizados (WS-BPEL, por ejemplo). Una vez disponemos de soluciones para modelar e incluso ejecutar PNS, el siguiente paso natural debería ser proponer soluciones que permitan analizar cómo se van a comportar estos procesos en ejecución, considerando como parte del análisis las descripciones semánticas de los mismos. En este sentido, se deberán plantear la utilización de nuevas versiones de las técnicas tradicionales de análisis con este nuevo tipo de procesos que, además, se están extendiendo ampliamente al campo de los *workflows* científicos, dada la utilidad del análisis aplicado a problemas cuya ejecución requiere un elevado coste en tiempo e infraestructura.

En [1] los autores presentaron un método de análisis basado en *Model Checking* para procesos de negocio semánticos. Este método permitía predecir el comportamiento de procesos con descripciones RDF utilizando lógica CTL (*Computation Tree Logic*) también enriquecida con información semántica. Las principales contribuciones del método eran: i) el análisis consideraba aspectos de control y datos (tanto internos del proceso, como los intercambiados con otros procesos) del proceso; y ii) no sólo se consideraban los datos conocidos en tiempo de diseño, sino también los que serán instanciados en tiempo de ejecución (datos simbólicos). Como parte del trabajo, finalmente, también se presentaba una implementación del método aplicado a un caso concreto y que justificaba la viabilidad del mismo.

En este trabajo se propone una generalización del prototipo anterior y una extensión y mejora. Esto se ha conseguido mediante la adición de una serie de herramientas y técnicas que facilitan las diferentes etapas necesarias para la verificación del modelo correspondiente. Como resultado presentamos COMBAS, un entorno que integra las herramientas necesarias para realizar *model checking* sobre PNS anotados mediante RDF y utilizando CTL. COMBAS abarca todas las fases del análisis, permitiendo la generación de los grafos de alcanzabilidad a partir del *workflow* de un PNS descrito mediante la herramienta Renew [2] y redes de Petri de alto nivel [1], el procesamiento de las anotaciones semánticas, la edición y visualización de fórmulas en CTL, la verificación del modelo y la posterior visualización gráfica de los resultados. COMBAS representa un enfoque totalmente novedoso frente a la ausencia y limitación de herramientas y *frameworks* para abordar los problemas de procesos con información semántica.

Ya existen numerosas herramientas de *model checking* que pueden clasificarse en función del lenguaje de modelado y de propiedades que utilicen. De entre todas ellas, vamos a centrarnos en las principales y que aceptan CTL como lenguaje de expresión de propiedades[1]. Algunas herramientas permiten definir propiedades en CTL y LTL. BANDERA [3] utiliza la técnica de análisis de código para verificar propiedades sobre modelos definidos en Java, y CADENCE SMV [3] usa una técnica de model checking plano sobre modelos expresados en los lenguajes Cadence SMV, SMV o Verilog. Otro ejemplo es NuSMV, que permite, además

[1] En `http://anna.fi.muni.cz/yahoda/` se puede encontrar una extensa comparativa de estas herramientas.

de CTL y LTL, definir propiedades en lenguaje PSL para hacer model checking plano sobre modelos SMV. APMC [4] usa una técnica aproximada y probabilística sobre *Reactive Modules* y las propiedades se describen en los lenguajes PCTL y PLTL. PRISM también soporta ambos lenguajes, además de CSL, y usa una técnica probabilística sobre modelos en gran variedad de lenguajes como PEPA, PRISM language o Plain MC [4]. Entre los *model checker* probabilísticos también encontramos MCMR, que soporta propiedades en lenguajes CSL, CSRL, PCTL o PRCTL y hace la verificación usando técnicas de tiempo real y probabilísticas sobre modelos modelados en Plain MC.

Otras herramientas aceptan propiedades en el lenguaje π-calculus, como TAPAs que también soporta CTL sobre modelos CCSP, y ARC que también acepta propiedades en CTL* haciendo model checking plano sobre modelos AltaRica. GEAR es otra herramienta que también acepta propiedades en π-calculus, además de en AFMC y CTL [5]. Similar a esta última encontramos CWB-NC que, usando model checking plano y temporizado sobre modelos CCS, CSP, LOTOS y TCCS, soporta propiedades en lenguajes AFMC, CTL y GCTL. Por último, existe una aplicación, MCMAS, que hace model checking plano y epistémico sobre modelos ISPL verificando propiedades en lenguajes CTL y CTLK [6].

Sin embargo, no existe ninguna herramienta de *model checking* que soporte la inclusión de información semántica RDF en el modelo de origen, su procesamiento y su posterior análisis. Esto justifica la utilidad de COMBAS, representando una alternativa muy válida para abordar este tipo de problemas.

Este artículo está organizado de la siguiente manera. La Sección 2 presenta el diseño de COMBAS y algunos detalles de implementación. Su aplicabilidad se demuestra en la Sección 3 mediante un caso de estudio. Finalmente, se presentan las conclusiones.

2. COMBAS: COmprobador de Modelos BAsado en Semántica

La herramienta desarrollada, llamada COMBAS (*COmprobador de Modelos BAsado en Semántica*), posibilita la verificación formal de un sistema que se le pasa como entrada mediante la utilización de bases de datos semánticas, un algoritmo de *Model Checking* que soporta RDF y predicados descritos en CTL. Como resultado, COMBAS determina si el modelo verifica o no las propiedades.

La herramienta cubre cinco funcionalidades: la generación de grafos de alcanzabilidad, revisión de grafos, edición de fórmulas CTL, *CTL Model Checking* y la revisión de resultados.

En la Figura 1 se muestra la arquitectura del sistema. El proceso es como sigue:

1. El ingeniero diseña el *workflow* con la herramienta gráfica Renew, guardándolo como PNML. Debe generar también los ficheros XML con la descripción de grafos y patrones que etiquetan el modelo.

2. El generador de grafos de alcanzabilidad se alimenta de los ficheros anteriores, produciendo dos salidas. Una es el grafo de alcanzabilidad almacenado en el *RDF Triple Store*. La otra salida está formada por un fichero XML que contiene la relación entre estados y su marcado, una colección de ficheros XML con los RDFGs (*RDF Graph Pattern*) correspondientes a los marcados y un fichero *dot* con la representación gráfica del grafo de alcanzabilidad, visible con la herramienta *Graphviz*.
3. El visor Web utilizará los ficheros XML y gráficos generados mediante el generador de grafos para visualizar el resultado.
4. Se debe crear la fórmula CTL a verificar con el *model checker*. Esta fórmula puede generarse usando el editor Web, que la exportará como un fichero XML.
5. El *Model checker* utiliza la fórmula CTL descrita en XML y generada con el editor, así como el grafo de alcanzabilidad almacenado en el *Triple Store*, y genera como salida una colección de ficheros XML relacionados que representan los estados del grafo de alcanzabilidad que verifican las partes correspondientes de la fórmula CTL. Dichos ficheros permiten la visualización y análisis en la interfaz Web.

2.1. Generación del grafo de alcanzabilidad

La generación del grafo de alcanzabilidad se realiza a partir del modelo de entrada: una red de Petri de la clase de las U-RDF-PN [1] con un marcado inicial, correspondiente al estado inicial del sistema. Las anotaciones semánticas pueden aparecer sobre tres tipos de elementos diferentes: 1) en los arcos, donde pueden aparecer patrones RDF; 2) en las transiciones, para definir las guardas asociadas así como las postcondiciones, y 3) en los lugares, donde los *tokens* también serán grafos RDF.

El grafo de alcanzabilidad generado también se almacena como tripletas RDF, según la ontología que se muestra en la Figura 2.

La generación del grafo de alcanzabilidad se ha implementado mediante la adaptación a la naturaleza semántica de las anotaciones del algoritmo clásico, tal y como se esquematiza a continuación:

1. Transformación del fuente PNML con la U-RDF-PN en RDF, de acuerdo a la ontología en la Figura 3, y almacenamiento en el *triple store*;
2. Introducción del marcado inicial m_0 en la pila de 'Marcados por procesar' y el conjunto de 'Estados' alcanzables (inicialmente vacío);
3. Mientras que ('Marcados por procesar' != vacía) {
   a) m_i = desapilado de la cima de 'No revisados';
   b) Obtención de las transiciones sensibilizadas por el marcado m_i (con sus *bindings* correspondientes) ;
   c) Generación de los marcados resultantes del disparo de dichas transiciones;
   d) Para cada m_j generado en el paso anterior {

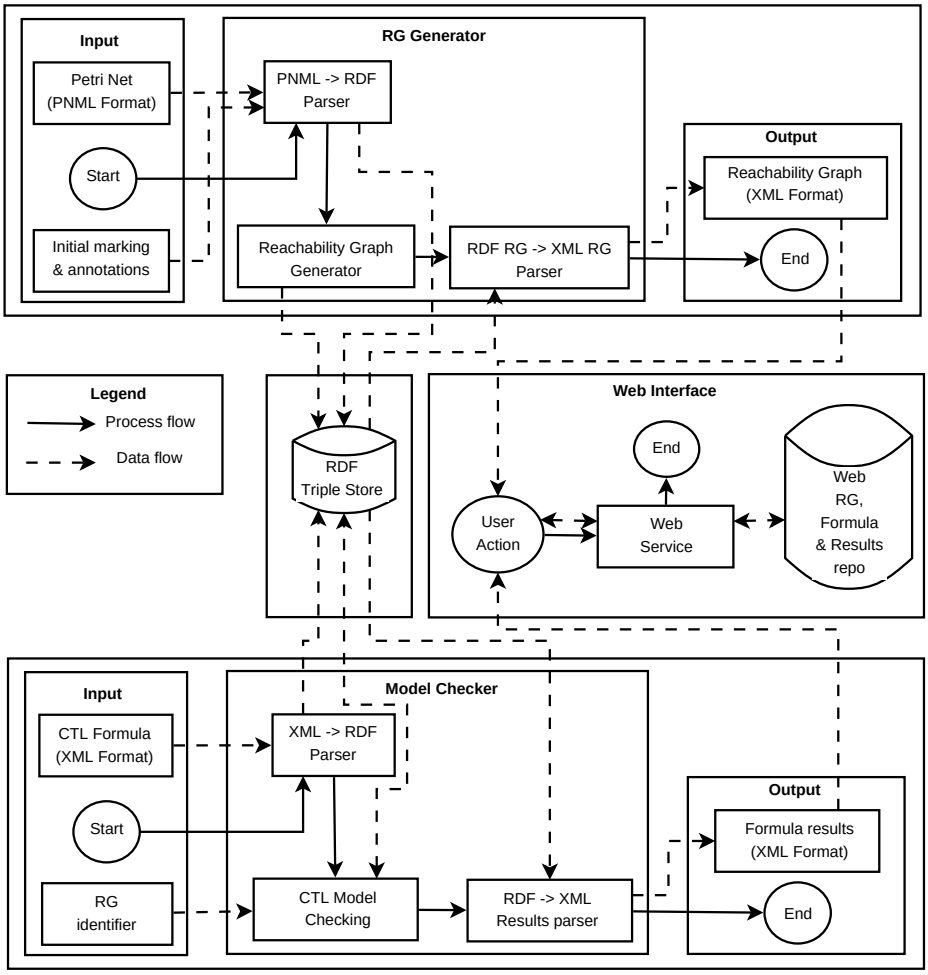

Figura 1. Arquitectura de COMBAS

1) Si no existe en 'Estados' uno equivalente a m_j, añadir m_j a 'Estados' y apilarlo en 'Marcados por procesar';
2) Añadir transición (m_i, m_j) al grafo; }

4. } Si no se generó ningún marcado nuevo en el paso anterior, enlazar el estado m_i consigo mismo, para convertirlo en un estado de *deadlock* o final (requisito para que el resultado sea una *estructura de Kripke*, necesaria como entrada al *model checker*).

Una vez obtenida la representación RDF del grafo de alcanzabilidad (*Reachability Graph*, RG), se genera el XML correspondiente.

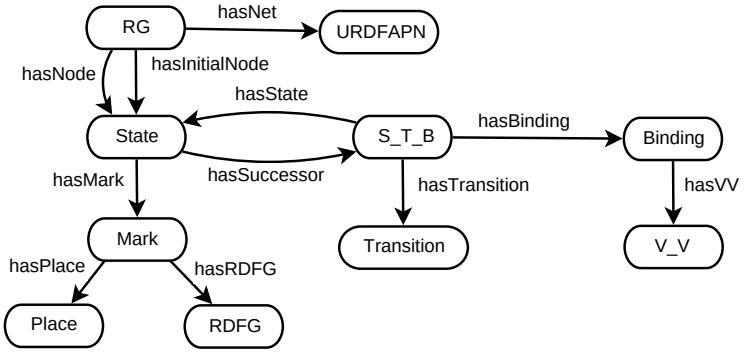

Figura 2. Diagrama de la ontología para definir el grafo de alcanzabilidad

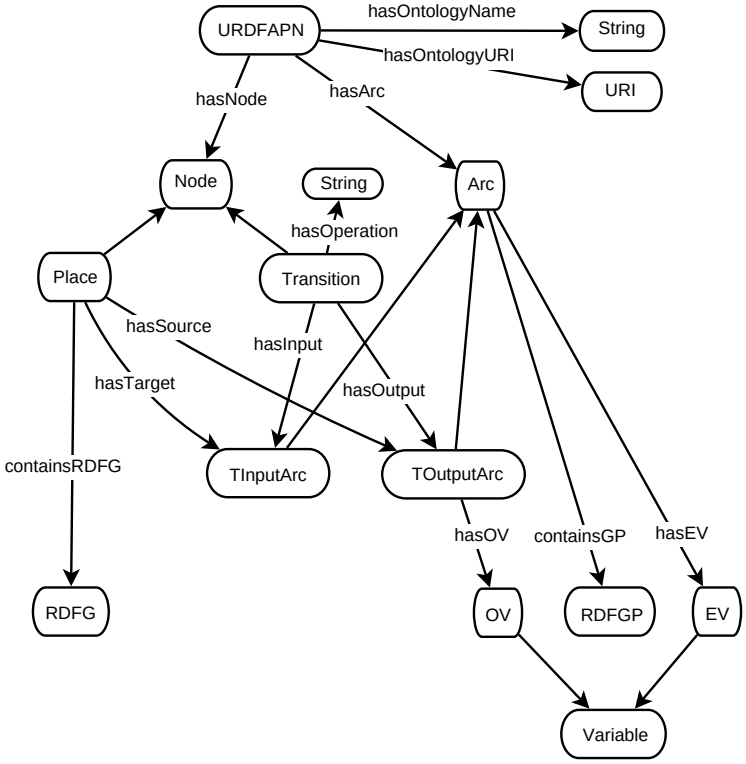

Figura 3. Diagrama de la ontología para describir U-RDF-PN

2.2. Servicio de edición y visualización

Las funciones básicas de la interfaz Web de usuario son la visualización de los grafos de alcanzabilidad generados por el *RG-Generator*, la construcción y edición de las fórmulas CTL que usará el *Model Checker* y la revisión de los

resultados de la verificación de propiedades. Aquí, la idea principal fue integrar todas estas funciones en un mismo interfaz, de forma que interactúen entre sí. Pasemos, por tanto, a describir con algo más de detalle cada una de ellas.

Visualización de grafos de alcanzabilidad. Como se ha explicado anteriormente, la salida del generador está formada por un fichero principal que describe la estructura del grafo de alcanzabilidad y una colección de ficheros que contienen los RDFG del marcado de cada estado. Todos estos ficheros están descritos en XML. El grafo generado puede contener un número de estados muy grande, del orden de cientos en un caso simple, pero puede crecer exponencialmente conforme aumenta la complejidad. Así, la revisión de la corrección del grafo puede ser una ardua tarea. De ahí surgió la idea de implementar un visor, que represente el grafo mediante un diagrama y permita consultar el marcado de cada estado.

Con la aplicación desarrollada es posible seleccionar un grafo de alcanzabilidad determinado, visualizar su estructura en forma de diagrama y consultar el marcado de cada estado.

Edición de fórmulas. Contar con una herramienta de edición de fórmulas CTL eficaz beneficia el proceso de verificación, minimizando errores en escritura, mejorando la experiencia visual y colaborando, en definitiva, a reducir tanto la curva de aprendizaje por parte del usuario como los tiempos de creación y edición de fórmulas. Por todo esto, es importante diseñar una forma intuitiva de interactuar con el usuario. Como idea inicial pareció acertado desarrollar un constructor/editor interactivo de fórmulas mediante una lista de componentes que funcionan como *nodos* de un árbol. A su vez, se consideró interesante añadir mecanismos que garantizan que la fórmula construida es válida y no contiene errores sintácticos. Además, se permite la visualización de la fórmula de forma gráfica en todo momento.

Revisión de los resultados del *Model Checker*. Tan importante como la edición de fórmulas es la revisión de los resultados la evaluación respecto a un modelo determinado. Saber si el modelo satisface la fórmula o no es inmediato, puesto que el *Model Checker* lo especifica al finalizar su ejecución. Sin embargo, el análisis de los resultados de la verificación puede ser tedioso si no se cuenta con una herramienta que facilite el proceso. Este caso es especialmente importante si el modelo no satisface la fórmula, ya que entonces es necesario analizar en qué estados del modelo se evalúa a falsa. Por esta razón, se decidió implementar una herramienta integrada en el visor de grafos que permite inspeccionar el resultado del análisis en un estado concreto y, a su vez, el marcado de dicho estado.

De esta forma, es posible seleccionar un grafo y observar rápida e intuitivamente el resultado de la ejecución de una fórmula, así como los estados del grafo, consultando qué nodos de la fórmula satisface cada estado.

2.3. Representación de la fórmula CTL mediante RDF

La fórmula CTL es uno de los parámetros de entrada del *model checker*, por lo que es necesario definir un formato de representación válido para alimentar la herramienta. La fórmula se describe mediante RDF de acuerdo con la ontología mostrada en la Figura 4 y, como se ha dicho, se almacena como un fichero en XML.

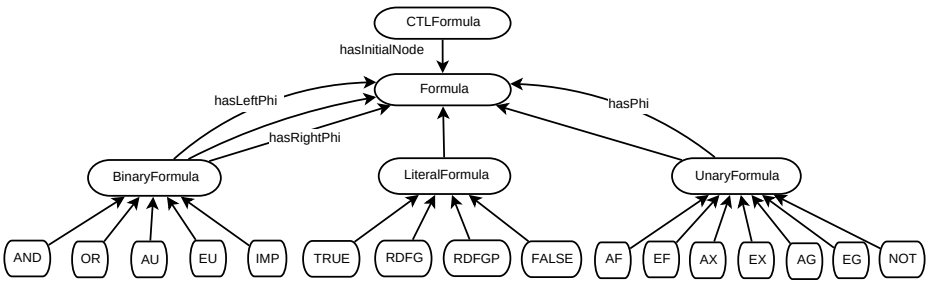

Figura 4. Diagrama de la ontología diseñada para definir fórmulas CTL

3. El *First Provenance Challenge*

Para evaluar el sistema se consideró interesante utilizar un modelo de *workflow* científico. Un buen ejemplo es el caso del *First Provenance Challenge* [7]. Se trata de un problema de referencia en *workflows* científicos por su complejidad y escalabilidad, y se puede definir y modelar con redes de Petri. La posibilidad de escalar la complejidad del problema posibilita que el grafo de alcanzabilidad generado crezca exponencialmente, de manera que el espacio de estados en los que se verifican las propiedades también aumenta su complejidad.

El *First Provenance Challenge* propone un ejemplo de *workflow* científico para crear *atlas cerebrales* de la población, obteniendo los datos del archivo de datos anatómicos en alta resolución del *fMRI Data Center* [8]. El *workflow* se esquematiza en la Figura 5, donde se pueden diferenciar cinco fases. Los procedimientos utilizados en cada etapa hacen uso del juego de herramientas AIR (*Automated Image Registration*) [9] para crear una imagen cerebral media a partir de una colección de datos anatómicos en alta resolución, y el juego de herramientas FSL [10] para crear imágenes 2D de las capas en cada dimensión del cerebro. Además de los datos representados en el diagrama, existen una serie de parámetros (constantes) que se proporcionan a los procesos, y que especifican opciones de configuración [7].

Los parámetros de entrada del *workflow* son un conjunto de imágenes cerebrales (*Anatomy Images*) y una imagen cerebral de referencia (*Reference Image*). Todas estas imágenes son escaneos 3D de un cerebro en varias resoluciones, de

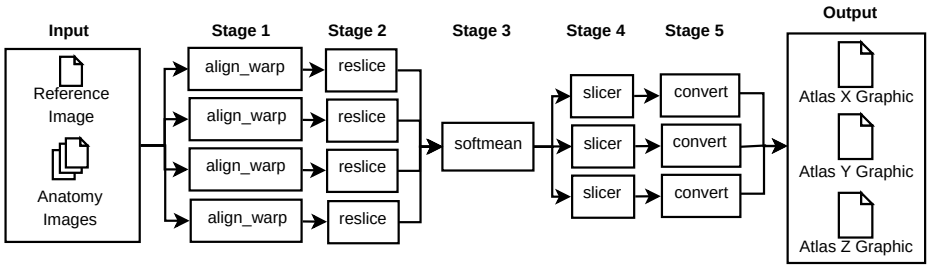

Figura 5. Diagrama del *workflow* correspondiente al *First Provenance Challenge*

modo que aparecen diferencias evidentes. Para cada imagen, además de la propia imagen existe información adicional. Los datos de imágenes fueron publicados en [11].

En la primera etapa, la herramienta *align_warp* compara la imagen de referencia con cada imagen cerebral para determinar cómo debe ser tratada (posición y forma de la imagen ajustada) para encajar con la imagen cerebral de referencia. La salida de cada procedimiento en esta fase es un conjunto de parámetros *warp* que definen la transformación espacial que se realizará. La transformación de la imagen se realiza en la fase *reslice* a partir de cada conjunto de parámetros *warp*, generando una nueva versión de la imagen original como salida. A continuación, todas las imágenes transformadas son promediadas en una sola usando un proceso de *softmean* en la tercera etapa. Para cada dimensión (*x, y, z*), la imagen promedio se trocea para obtener un mapa 2D del plano desde el centro de la imagen en 3D. La salida es un *atlas* de imágenes generado mediante *slicer*, una herramienta que forma parte del juego de utilidades FSL [10]. Finalmente, cada conjunto de datos es transformado en una imagen utilizando *convert*, incluido en la utilidad GNU *ImageMagick*.

La U-RDF-PN que modela el *workflow* se muestra en la Figura 6. A partir de este modelo se genera el grafo de alcanzabilidad, cuya versión simplificada (debido a su gran tamaño) se muestra en la Figura 7. Como se ha comentado, la generación del grafo de alcanzabilidad es una tarea muy costosa debido, fundamentalmente, a la ejecución de dos operaciones: la que determina los posibles *bindings* para la sensibilización de una transición y, la más costosa, la que determina si un potencial estado nuevo es equivalente a uno ya calculado anteriormente. Ambas operaciones hacen un uso intensivo de invocaciones a operaciones del *RDF-store*. El coste real obtenido para la generación del RG utilizando uno o dos juegos distintos de imágenes se muestra en la Tabla 1, donde se aprecia claramente el aumento del coste computacional al escalar el problema. Las ejecuciones se realizaron sobre un procesador Intel Core 2 Duo E7400 de 64 bits (2 x 2.8GHz), con 4GB de RAM, sistema de E/S SATA y corriendo un SO Ubuntu 11.04 con un kernel optimizado 2.6.38.8 y la JVM JRE6u24. El *RDF-store* utilizado fue la versión 6.1.1 de Virtuoso Open Source.

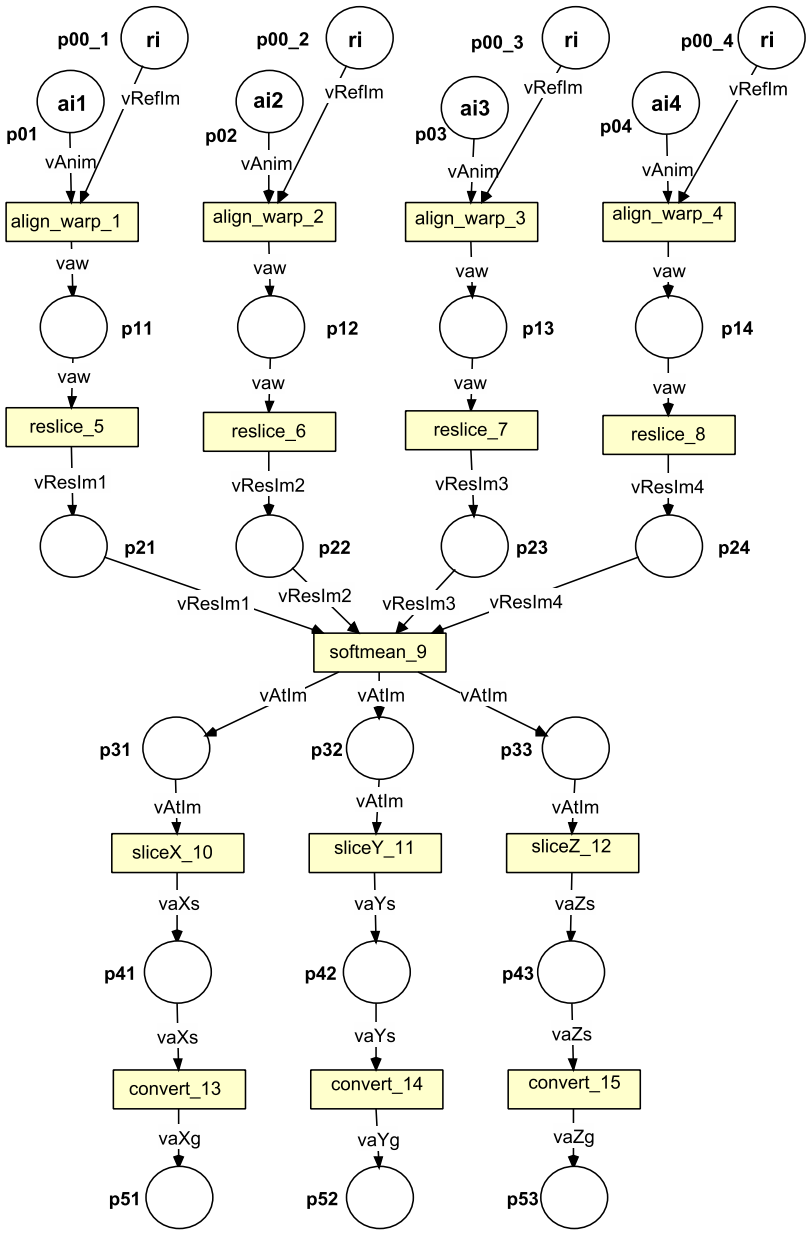

Figura 6. U-RDF-PN correspondiente al *First Provenance Challenge*

3.1. Ejecución y resultados de *Model Checking*

Una vez generado el grafo de alcanzabilidad, se pasa a la fase de *model checking*. Para ello es necesario establecer cuáles son las fórmulas que representan

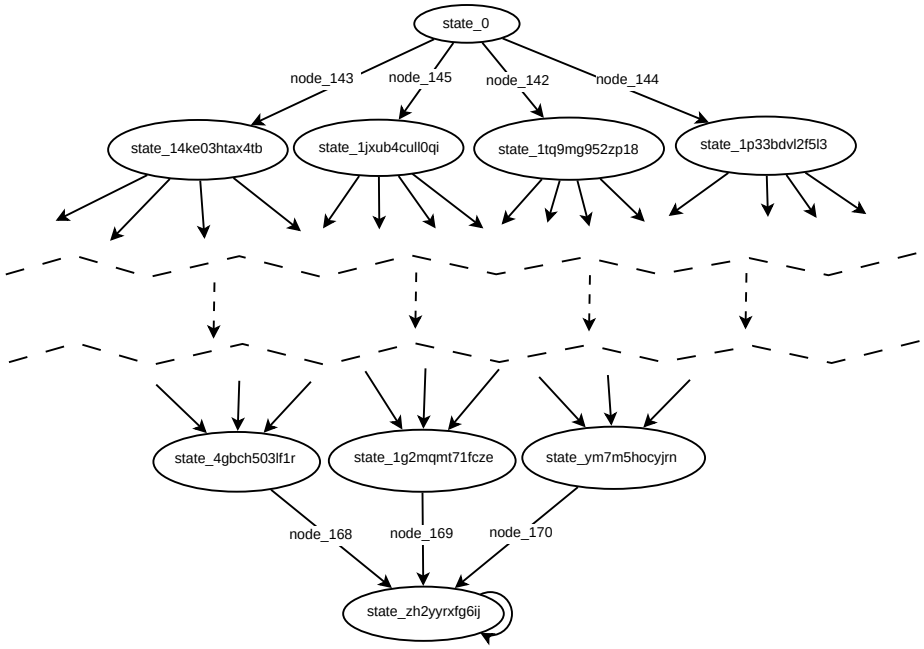

Figura 7. Diagrama resumido del grafo generado para el *First Provenance Challenge*

Caso	Tiempo de generación	Estados generados
1 paquete de imágenes	2 min 16 seg	108
2 paquetes de imágenes	44h y 23 min	11664

Tabla 1. Resultados de la ejecución del generador de RG para el ejemplo

las propiedades que queremos probar. Nuestro modelo corresponde a un sistema que, a partir de un conjunto de cinco imágenes cerebrales, genera un promedio de cuatro de ellas y extrae las vistas correspondientes a las dimensiones X, Y y Z del cerebro. De este modo, al final, obtenemos tres imágenes en 2D codificadas en formato GIF. Además, para garantizar el procesado por paquetes, incluimos un identificador de grupo de imágenes. Así pues, podemos construir una fórmula que responda a la pregunta:

Dada una imagen de referencia RI y las imágenes iniciales AI_1, AI_2, AI_3 y AI_4, correspondientes al paquete P, ¿obtendré siempre una imagen de la vista X, una imagen de la vista Y y una imagen de la vista Z para el paquete de imágenes P (Package01)?

En la pregunta anterior es fácil identificar algunos operadores lógicos ordinarios y otros temporales. Así, podríamos traducir dicha pregunta al lenguaje CTL de lógica temporal en los siguientes términos intuitivos: el operador

CTL $AF(p)$ indica que la propiedad p se verificará eventualmente en el futuro (AF se puede leer como *Always in the Future*, en toda ejecución), a partir del estado inicial. Por lo tanto, estamos preguntando si desde estado inicial del sistema se alcanza un estado del sistema en el que existen las imágenes convertidas *Package01_SliceX_Converted*, *Package01_SliceY_Converted* y *Package01_SliceZ_Converted*, correspondientes al paquete de imágenes *Package01*:

```
AF( AND (RDFG_X, AND(RDFG_Y,RDFG_Z) ) )
RDFGP_X: t:Package01  t:hasConverted  t:Package01_SliceX_Converted
RDFGP_Y: t:Package01  t:hasConverted  t:Package01_SliceY_Converted
RDFGP_Z: t:Package01  t:hasConverted  t:Package01_SliceZ_Converted
```

Tras ejecutar el *Model Checker* para verificar dicha fórmula contra el RG generado anteriormente, obtenemos la siguiente salida:

```
INFO [main] (CombasApp.java:239) - Model satisfies the formula!
INFO [main] (CombasApp.java:244) - Output files generated.
INFO [main] (CombasApp.java:248) - Checked in 13471 millis
INFO [main] (CombasApp.java:249) - Formula: formula_zi0lxko6oc16
INFO [main] (CombasApp.java:250) - Model: netId1lda11bfzilll_RG
```

Lo cual confirma que el modelo satisface la propiedad. Si se aplica al modelo que incluye dos paquetes de imágenes, la propiedad también se verifica. El tiempo requerido para resolver la pregunta anterior se muestra en la Tabla 2.

Caso	**Tiempo de verificación**	**Resultado**
1 paquete de imágenes	13 seg	modelo válido
2 paquetes de imágenes	32 min	modelo válido

Tabla 2. Resultados de la ejecución del *model checker* para el ejemplo

También es posible verificar propiedades de seguridad, en el sentido de asegurar que algo malo no va a ocurrir, mediante la *no verificación* de fórmulas. Para ello, podríamos comprobar si existe algún camino tal que se obtenga alguna imagen convertida final que no contenga información de las imágenes origen. Pasemos por tanto a estudiar la construcción de dicha fórmula. Análogamente al operador AF (en toda ejecución), existe el operador $EF(p)$ (*there Exists an execution*, hay al menos una ejecución) para especificar si existe una ejecución posible que cumpla p en el futuro.

Así podemos construir la fórmula:

```
EF(AND(NOT(RDFGP_atlas),OR(RDFGP_x,OR(RDFGP_y,RDFGP_z))))
```

donde

```
== PROPIEDADES ==
RDFGP_x :
  ?package t:hasConverted ?converted .
  ?converted rdf:type "XviewGIF"
RDFG_y :
  ?package t:hasConverted ?converted .
  ?converted rdf:type "YviewGIF"
RDFG_z:
  ?package t:hasConverted ?converted .
  ?converted rdf:type "ZviewGIF"
RDFGP_atlas:
  ?package t:hasConverted ?converted .
  ?converted t:hasSlice ?slice .
  ?slice t:hasAtlas ?atlas .
  ?atlas t:hasRefImg ?refimg .
  ?atlas t:hasAnatomyImg1 ?img1 .
  ?atlas t:hasAnatomyImg2 ?img2 .
  ?atlas t:hasAnatomyImg3 ?img3 .
  ?atlas t:hasAnatomyImg4 ?img4 .
```

Esta fórmula establece si existe algún camino con un estado futuro que cumpla lo siguiente: *no existe información de origen y, además, se ha obtenido la imagen final de la vista X, Y o Z.* Si el resultado es cierto, nuestro sistema no funcionará de forma correcta. De modo que esperamos que el resultado sea *falso*. Tras ejecutar el *model checker* para verificar dicha fórmula, obtenemos la siguiente salida:

```
ERROR [main] (CombasApp.java:241) - Model doesn't satisfy the formula.
INFO [main] (CombasApp.java:244) - Output files generated.
INFO [main] (CombasApp.java:248) - Checked in 23300 millis
INFO [main] (CombasApp.java:249) - Formula: formula_1pydnao80fxve
INFO [main] (CombasApp.java:250) - Model: netId1lda11bfzilll_RG
```

Lo cual confirma que el modelo no satisface la propiedad, lo que quiere decir que nuestro modelo es válido. El tiempo requerido para verificar esta fórmula en los dos grafos considerados se muestra en la Tabla 3.

Caso	**Tiempo de verificación**	**Resultado**
1 paquete de imágenes	23 seg	modelo válido
2 paquetes de imágenes	58 min	modelo válido

Tabla 3. Resultados de la ejecución del *model checker* para el ejemplo con una fórmula alternativa

4. Conclusiones

En este trabajo se ha presentado COMBAS, un entorno completo para el procesamiento, *model checking* de PNS y verificación/visualización de los resultados mediante un entorno visual y adaptado a la utilización de RDF y CTL. La carencia de herramientas de *model checking* que soporten la inclusión de información semántica RDF en el modelo de origen y que no integren las herramientas necesarias para su posterior procesamiento y análisis provocan que COMBAS represente un enfoque novedoso para abordar problemas de PNS, así como de *workflows* científicos, tal y como se ha demostrado mediante su aplicación al *First Provenance Challenge*. A corto plazo se está estudiando la generación de los grafos de alcanzabilidad en entornos de computación *grid*, la extensión del *model checker* para soportar análisis simbólico y la integración de otras tecnologías semánticas en el entorno.

Bibliografía

1. Ibáñez, M., Álvarez, P., Ezpeleta, J.: Predicción simbólica del comportamiento en ejecución de los procesos de negocio semánticos. In: V Jornadas Científico-Técnicas en Servicios Web y SOA – JSWEB'09. (2009) 183–196
2. Kummer, O., Wienberg, F.: Renew - the Reference Net Workshop. In: Tool Demonstrations, 21st International Conference on Application and Theory of Petri Nets, Computer Science Department, Aarhus University, Aarhus, Denmark (2000) 87–89
3. Garlan, D., Khersonsky, S., Kim, J.S.: Model checking publish-subscribe systems. In: Proceedings of the 10th international conference on Model checking software. SPIN'03, Berlin, Heidelberg, Springer-Verlag (2003) 166–180
4. Duflot, M., Fribourg, L., Hérault, T., Lassaigne, R., Magniette, F., Messika, S., S.Peyronnet, Picaronny, C.: Probabilistic model checking of the csma/cd protocol using prism and apmc. Electr. Notes Theor. Comput. Sci. **128** (2005) 195–214
5. Grumberg, O., Veith, H., eds.: 25 Years of Model Checking - History, Achievements, Perspectives. In Grumberg, O., Veith, H., eds.: 25 Years of Model Checking. Volume 5000 of Lecture Notes in Computer Science., Springer (2008)
6. Lomuscio, A., Qu, H., Raimondi, F.: Mcmas: A model checker for the verification of multi-agent systems. In: CAV. (2009) 682–688
7. Moreau, L. et al: Special issue: The first provenance challenge. Concurr. Comput.: Pract. Exper. **20** (2008) 409–418
8. fMRI Data Center: Web del archivo de recursos anatómicos en alta resolución del *fMRI Data Center*. http://www.fmridc.org/ (2010)
9. AIR (*Automated Image Registration*): juego de herramientas AIR de tratamiento de imágenes 3D anatómicas. http://air.bmap.ucla.edu/AIR5/index.html (2010)
10. FSL: conjunto de herramientas FSL de tratamiento de imágenes anatómicas en 3D. http://www.fmrib.ox.ac.uk/fsl/ (2010)
11. Head, D., Snyder, A., Girton, L., Morris, J., Buckner, R.: Frontal-hippocampal double dissociation between normal aging and alzheimer's disease. Cereb Cortex **15** (2005) 732–9