SAKOnto: una ontología de conocimiento sobre arquitecturas de software

María Luciana Roldán, Silvio Gonnet, Horacio Leone
Instituto de Desarrollo y Diseño
(CONICET / UTN)
Avellaneda 3657, Santa Fe
{lroldan, sgonnet, hleone}@santafe-conicet.gov.ar

Resumen

El conocimiento sobre arquitecturas de software abarca aspectos relativos no solo a las descripciones arquitectónicas que conforman los modelos, sino además aspectos relativos al contexto del diseño de la arquitectura, al razonamiento aplicado y conocimiento general sobre el dominio, como son los patrones y tácticas de diseño de arquitecturas de software. Las formas de representar o documentar tal conocimiento son diversas, ya sea empleando soportes de archivos de texto o estructurándolo en de bases de datos relacionales. Se conoce, por otro lado, que las ontologías son útiles para compartir y diseminar conocimiento de un dominio en particular. Una ontología sobre el conocimiento del dominio del diseño de arquitecturas de software posibilitaría la representación de tal conocimiento y la utilización de razonadores para la inferencia de conocimiento, que as su vez, podría ser empleado para la verificación automática de conformidad arquitectónica, y el reuso de las descripciones arquitectónicas y de decisiones de diseño que se generaron en proyectos anteriores. En este trabajo se propone SAKOnto una ontología especificada en OWL 2, cuyos conceptos surgen a partir de conocimiento estructurado sobre diferentes dominios de arquitecturas de software y cuya población se obtiene a partir del conocimiento arquitectónico capturado en diversos en procesos de diseño arquitectónico.

1. Introducción

La importancia del conocimiento sobre arquitecturas de software (AK –Architectural Knowledge) y su gestión ha dado lugar a numerosos trabajos de investigación en relación a su captura y representación, así a como su uso para intercambiar/compartir dicho conocimiento, para validación/verificación de conformidad, descubrimiento de nuevo conocimiento y trazabilidad (*sharing*, *compliance*, *discovering*, *traceability*) [3]. Los primeros trabajos en representación de conocimiento sobre arquitecturas de software se enfocaban en cómo describir las arquitecturas en términos de su estructura y comportamiento. Más tarde, el foco fue puesto en la representación de conocimiento arquitectónico relativo a las decisiones arquitectónicas y su razonamiento. Mientras que los primeros se centraban en la representación del artefacto final, las últimas lo hacían en la representación de cómo éste había sido obtenido. Hoy en día podemos decir que el conocimiento arquitectónico abarca diferentes categorías de conocimiento: conocimiento general (por ejemplo, tipos de artefactos manejados, patrones de diseño y tácticas), conocimiento de contexto (requerimientos, restricciones), conocimiento de diseño (las diferentes descripciones arquitectónicas, vistas, modelos de una arquitectura de software obtenidas en diferentes proyectos), y conocimiento de razonamiento (las decisiones tomadas, los argumentos que las respaldaron, las alternativas que se evaluaron en los proyectos de diseño arquitectónico) [18]. Se han propuesto además diversas formas para representar tal conocimiento, empleándose enfoques basados en archivos o documentos textuales (como el uso de plantillas de decisiones arquitectónicas y wikis), anotaciones (que se vinculan a descripciones arquitectónicas), y estructuración del mismo en bases de datos vinculadas a las herramientas de captura.

Una de las modalidades de representación de conocimiento arquitectónico más utilizada es la documentación basada en archivos (file-based documentation). Generalmente se la organiza en plantillas para documentar decisiones arquitectónicas, y se la complementa con otras descripciones arquitectónicas basadas en vistas definidas en algún lenguaje de descripción de arquitecturas (ADL). La naturaleza "lineal" que tiene la documentación de AK basada en archivos dificulta estructurar el conocimiento de manera que sea útil para los diferentes usuarios de la documentación, ya que organiza el conocimiento de una manera estática, siendo difícil recuperar cierto tipo de conocimiento que esa estructura no soporta [5].

Entre los diferentes elementos que conforman el conocimiento arquitectónico, se establecen diversas relaciones. Un ejemplo de este tipo de relaciones puede

observarse en la vinculación entre requerimientos, decisiones de diseño y componentes de una arquitectura de software. Cuando el conocimiento se encuentra representado de una manera estática y lineal en documentos textuales, es difícil para los usuarios del mismo encontrar o recuperar la información deseada. Por ejemplo, para un arquitecto que debe evaluar una decisión arquitectónica, es importante conocer cuál es el razonamiento que la respalda, qué otras alternativas existen, qué otras decisiones y requerimientos relacionados debería considerar, o se han considerado si la decisión ya fue tomada hace tiempo en otro proyecto o contexto.

En trabajos previos [16] hemos propuesto un modelo que permite representar conocimiento arquitectónico perteneciente a las diferentes categorías mencionadas: conocimiento general, conocimiento de contexto, conocimiento de razonamiento, y conocimiento de diseño. Dicho modelo fue concebido bajo un enfoque operacional en donde las decisiones de diseño se materializan en la ejecución de operaciones arquitectónicas, las cuales permiten además establecer asociaciones de precedencia entre las múltiples versiones de modelo de la arquitectura de software que se van obteniendo a medida que el proceso de diseño tiene lugar. Basado en un esquema de versionamiento, todas las versiones de los productos generados como resultado de las operaciones ejecutadas por los arquitectos son mantenidas conservándose además los enlaces necesarios para mantener la trazabilidad entre argumentos, decisiones y resultados. El modelo fue implementado en la herramienta computacional TracED(aaS)[1] [10], la cual permitió la ejecución de casos de estudio en diferentes dominios de arquitecturas de software y, por lo tanto, ha posibilitado la generación de conocimiento arquitectónico a partir de la experiencia obtenida en esos proyectos de diseño. Los principales resultados del modelo propuesto estuvieron enfocados en la representación del conocimiento arquitectónico desde el punto de vista de los actores que son *productores* de dicho conocimiento. Fundamentalmente, se trabajó en facilitar herramientas y mecanismos de soporte a la captura del conocimiento de manera que se integren en forma natural con el proceso de diseño de arquitecturas de software. Sin embargo, es muy sabido que cuando las herramientas son intrusivas y no están integradas al proceso de diseño, el conocimiento aplicado o generado, las decisiones tomadas y el razonamiento que las acompaña, generalmente no se documenta, o se lo hace en forma incompleta, lo cual implica la pérdida de uno de los capitales más valiosos de la organización que se dedica al desarrollo de software.

En continuidad con ese trabajo, la presente propuesta se enfoca en lograr ventajas para aquellos actores que son *consumidores* o usuarios del conocimiento arquitectónico, es decir, en facilitar la recuperación del conocimiento que ha sido documentado o representado previamente. El punto de partida del enfoque propuesto es el conocimiento arquitectónico que se encuentra representado de manera estructurada según el modelo de representación de conocimiento de diseño de arquitecturas de software que se ha propuesto en trabajos previos. Somos conscientes que la cantidad de objetos de conocimiento arquitectónico y de razonamiento generados en proyectos de mediana y alta complejidad, torna la recuperación de conocimiento arquitectónico en una actividad inmanejable empleando mecanismos consultas tradicionales sobre bases de datos de tipo relacionales. Sin embargo, el hecho que el conocimiento se encuentre organizado de esta manera y no en forma textual y dispersa en diversos documentos de texto, constituye una ventaja que podemos aprovechar. Considerando esta ventaja y buscando superar las limitaciones del modelo previo para el "consumo" del conocimiento arquitectónico, proponemos la construcción de una ontología de conocimiento arquitectónico, a la que denominamos SAKOnto. Una ontología se refiere a un modelo de dominio formal que describe los conceptos y relaciones de ese dominio [17]. Las ontologías posibilitan la clasificación jerárquica de conceptos de dominio interrelacionados y pueden ser representadas empleando un esquema RDF[2] o el lenguaje OWL[3]. El uso de estos lenguajes permite que las ontologías sean legibles por los humanos e interpretable por las máquinas, permitiendo realizar consultas para obtener conocimiento, así como también inferir nuevo conocimiento. El objetivo del presente trabajo es la definición formal de dicha ontología, que posibilite la recuperación de conocimiento arquitectónico de una manera eficiente.

El resto del trabajo se estructura de la siguiente manera. En la sección 2 se analizan una serie de trabajos de otros autores relacionados con ontologías para arquitecturas de software. En la sección 3 se presenta el enfoque propuesto, dentro del que se enmarca la ontología SAKOnto. Luego, en la sección 4 se describe el proceso de construcción de la ontología. En la sección 5, se bosquejan algunos lineamentos en relación a la posibilidad de integración de SAKOnto con otras ontologías existentes. Finalmente en la sección 6 se presentan las conclusiones.

[1] http://traced-doc.appspot.com

[2] https://www.w3.org/RDF/

[3] https://www.w3.org/TR/owl-features/

2. Trabajos relacionados

Con el surgimiento del concepto de decisión arquitectónica, aparecieron las primeras ideas en el empleo de ontologías aplicadas al diseño de arquitecturas de software. Kruchten y colab. [12] propusieron una ontología para describir decisiones arquitectónicas y relaciones entre ellas, incluyendo aspectos de razonamiento. Para explotar el conocimiento de la ontología propusieron además una herramienta para preservar los grafos de decisiones de diseño y todas sus interdependencias con el fin de brindar soporte a la evolución y mantenimiento de los sistemas. En la misma dirección la propuesta de Akermann y Tyree [1] apuntaba a registrar decisiones arquitectónicas, artefactos de software (software assets), hojas de ruta (roadmaps) y intereses arquitectónicos (architectural concerns). Tal propuesta fue ilustrada con una versión simplificada del estándar de IBM para Descripciones Arquitectónicas.

Propuestas más recientes [17] emplean OWL y SWRL[4] para definir modelos ontológicos que son específicos para el diseño de arquitecturas de software. Mediante el lenguaje OWL se representan las estructuras y restricciones en las relaciones entre los elementos de la arquitectura. Además con el lenguaje de reglas SWRL se capturan algunas interacciones dinámicas dentro de los modelos arquitectónicos, como restricciones adicionales. De esta manera, se define una meta ontología (básicamente para un estilo arquitectónico) que puede ser usada para crear especificaciones de arquitecturas de software particulares, a través de la extensión de la meta-ontología. Esta representación permite describir formalmente las arquitecturas de software, y mediante la aplicación de razonadores se realizan algunas verificaciones sobre el diseño. A diferencia de SAKOnto, esta propuesta sólo considera la representación de modelos arquitectónicos basados en algún estilo posibilitando fundamentalmente la representación de conocimiento arquitectónico general, como podrían ser los estilos arquitectónicos predefinidos. En esta propuesta no se representa información vinculada con el proceso de diseño, ni del razonamiento involucrado en el mismo.

Otros trabajos que aplican el uso de ontologías a la representación del conocimiento del dominio de las arquitecturas de software [4, 5, 7] se enfocan en la búsqueda y recuperación de conocimiento en documentos textuales (file-based documentation). Figueiredo y colab. [7] proponen un framework para posibilitar la búsqueda de información de arquitecturas de software en artefactos de documentación generados en ambientes de comunidades virtuales, como por ejemplo, emails, minutas de reunión, y Wikis. El enfoque consiste en definir una ontología de arquitecturas de software, junto con ontologías del dominio de aplicación, que modelen el conocimiento del dominio de desarrollo del sistema. Estas ontologías son usadas para indexar los artefactos, así como para visualizar los resultados de las búsquedas, ayudar a los usuarios a explorar, descubrir y analizar información, por medio de un mecanismo de búsqueda semántico. A diferencia de nuestra propuesta, se basa en artefactos donde el conocimiento arquitectónico prácticamente no está estructurado.

En forma similar, López y colab. (2012) [13] presentan un enfoque conducido por ontologías, para recuperar razonamiento arquitectónico a partir de documentos en texto plano, y sintetizarlo en un repositorio de conocimiento centralizado. El enfoque propuesto, denominado TREx tiene también dos ontologías: una ontología para representar la arquitectura de software del sistema, y otra ontología para describir el razonamiento del proyecto.

Otros autores [5, 6] que proponen métodos y herramientas para reemplazar los métodos de documentación basada en archivos que apuntan a superar los problemas de creación y recuperación de conocimiento documentado durante el desarrollo de software. Para ello proponen cambiar las prácticas actuales basadas en archivos hacia el uso de documentos de software indexados con una ontología liviana que sea efectiva y fácil de usar. Con este enfoque, implementan un prototipo empleando una ontología genérica para brindar soporte al indexado y recuperación del conocimiento desde documentos de software. De esta manera, emplean una ontología de software en un wiki semántico optimizado para la documentación arquitectónica, que busca superar las dificultades que tiene la documentación basa en archivos. En un trabajo posterior, siguiendo el enfoque de Gruninger y Fox (1995) [8] en relación al empleo de preguntas de competencia para definir el alcance de una ontología, estos autores también han propuesto un método para la construcción de ontologías para documentación de arquitecturas de software que hace uso de preguntas típicas para la elicitación y construcción de la ontología.

Vázquez y colab. [20] utilizan ontologías para mapear una arquitectura de software con su implementación. Se basan en un enfoque automatizado basado en técnicas de alineación de ontologías que apunta a solucionar el problema de la escasa correspondencia que suele haber entre la documentación arquitectónica y la implementación en código de una arquitectura de software. Para ello utiliza dos ontologías, una para representar la arquitectura y otra para su implementación. Las correspondencias entre estas dos ontologías proveen los mapeos entre elementos de la arquitectura y su implementación. La ontología para representar la arquitectura de software considera básicamente conocimiento de diseño de un modelo arquitectónico,

[4] https://www.w3.org/Submission/SWRL/

limitándose a elementos relativos a su vista de componentes y conectores, junto con sus responsabilidades, el cual es el conocimiento que de mayor interés al momento de establecer correspondencias entre la arquitectura y su implementación.

3. Enfoque propuesto

En la Figura 1 se presenta una vista preliminar del enfoque al cual se integra SAKOnto. La ontología que se desarrolla en este trabajo es la que permitirá la representación del conocimiento arquitectónico y facilitará su recuperación desde el punto de vista de los consumidores de dicho conocimiento. Una ontología se refiere a un modelo de dominio formal que describe los conceptos y relaciones de ese dominio [19]. Las ontologías posibilitan la clasificación jerárquica de conceptos de dominio interrelacionados y pueden ser representadas por ejemplo usando un esquema RDF o el lenguaje OWL. El uso de estos lenguajes permite que las ontologías sean legibles por los humanos e interpretable por las máquinas, permitiendo la consulta de conocimiento, así como también la inferencia de nuevo conocimiento. El alcance de este trabajo es la construcción de la ontología de conocimiento arquitectónico, lo que implica la definición completa de la jerarquía clases y sus propiedades, las relaciones posibles entre individuos de éstas, mediante propiedades de objetos, y los axiomas necesarios para expresar ciertas restricciones entre los conceptos.

Una vez desarrollada SAKOnto con todas las clases, propiedades y relaciones que se requieren para modelar diferentes dominios de arquitecturas de software, dicha ontología necesita ser poblada con conocimiento arquitectónico importado desde los diferentes elementos de conocimiento diseño que se capturaron en diversos proyectos de diseño con la herramienta TracED(aaS) (actividad *AK Capturing* en la Figura 1). También es importante poblar la ontología con individuos que permitan representar conocimiento arquitectónico general, como son patrones de diseño, tipos de componentes, vistas, etc. (actividad *SAKOnto Population*, en la Figura 1). Para automatizar la población de la ontología a partir de conocimiento recabado durante la definición de dominios y proyectos de diseño arquitectónicos ejecutados con la asistencia de TracED(aaS), se prevé desarrollar un componente de conversión e importación de individuos en la ontología (*AK Loader*, en Figura 1).

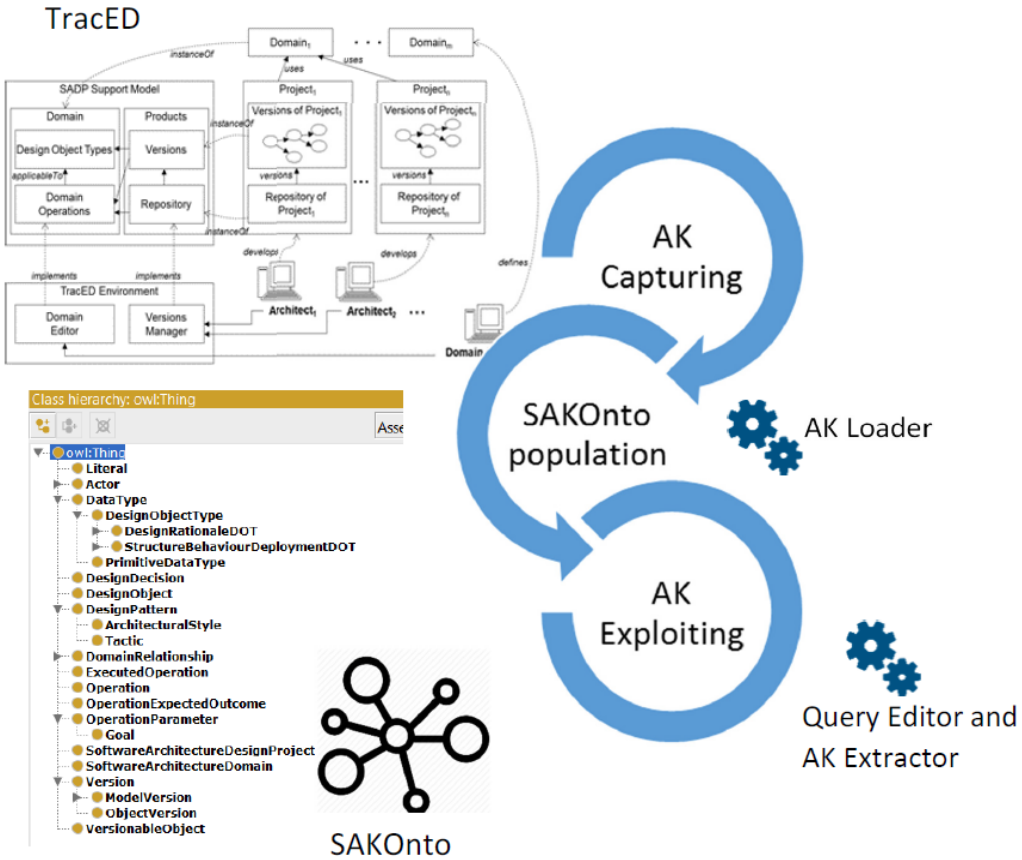

Figura 1. Vista preliminar del enfoque

Por otro lado, el enfoque incluye el desarrollo un componente *Query editor / AK Extractor* cuya función es obtener y explotar el conocimiento contenido en la ontología, el cual estará basado en el uso de razonadores y consultas en SPARQL[5] sobre tripletas RDF.

4. Construcción de SAKOnto

Para desarrollar SAKOnto en primer lugar necesitamos determinar el domino y alcance de la ontología. Para ello se busca responder a varias preguntas básicas del tipo: i) ¿Cual es el dominio de aplicación de la ontología? ii) ¿Cuál es la utilidad de la ontología? iii) ¿Para qué tipos de preguntas la información en la ontología debería proveer respuestas? iv) ¿Quién usará y mantendrá la ontología? Las respuestas a estas preguntas pueden cambiar durante el proceso de diseño de la misma, pero en principio éstas ayudarán a limitar el alcance del modelo.

Una manera de determinar el alcance de la ontología, es bosquejando una lista de preguntas que la base de conocimiento basada en dicha ontología debería ser capaz de responder, las cuales se denominan preguntas de competencia [8]. Además, estas preguntas servirán posteriormente como una manera de validar o controlar la calidad de la ontología definida, ya que permitirán verificar si la ontología contiene suficiente información para responder a este tipo de preguntas, y ver si las respuestas requieren un nivel particular de detalle o representación en un área en particular. Las preguntas de competencia son preliminares no necesitan ser exhaustivas.

Siguiendo esta metodología, las preguntas de competencia se agruparon en dos grupos, de acuerdo a su alcance:

1. proceso de diseño de arquitecturas de software
2. arquitecturas de software

A continuación de describen las agrupaciones mencionadas.

4.1. Primer grupo de preguntas de competencia

El primer grupo (CQG#1) de preguntas de competencia surgió considerando los objetivos y alcance de la ontología en relación a la representación de procesos de diseño de arquitecturas de software, teniendo en cuenta el modelo en que se basa TracED(aaS) a partir del cual se alimentará SAKOnto, el cual fue detallado en [15, 16]. Este a herramienta se sustenta en un esquema de administración de versiones que permite capturar y representar la evolución de los distintos productos del diseño de arquitectura de software en forma conjunta con las operaciones aplicadas a los mismos. El modelo considera el proceso de diseño como una secuencia de actividades que opera sobre los productos del proceso de diseño, denominados objetos de diseño. Los objetos de diseño representan modelos del artefacto arquitectónico que está siendo diseñado, requerimientos a cumplir y restricciones impuestas al modelo. Naturalmente, estos objetos evolucionan durante un proceso de diseño, dando lugar a múltiples versiones de los mismos. Un conjunto de estas versiones constituyen una versión del modelo de la arquitectura, la cual describe el estado del proceso de diseño en un determinado instante. En este esquema, cada versión de modelo es generada mediante la aplicación de una secuencia de operaciones a una versión de modelo predecesora. La secuencia de operaciones puede incluir la eliminación, creación, y modificación de versiones que forman la versión de modelo predecesora. El modelo además de proveer un conjunto de operaciones básicas (*agregar*, *eliminar*, *modificar*), posibilita también la definición de operaciones más complejas. La herramienta permite definir un dominio con un conjunto de operaciones complejas (por ejemplo: *refinar*, *aplicarMVC*, *aplicarClienteServidor*, etc.), junto a los objetivos o intenciones que se persiguen con su ejecución, de manera de capturar mayor información durante dicha ejecución. El esquema de representación de versiones divide la representación de los objetos de diseño en dos niveles: i) nivel repositorio, donde se representa cada objeto de diseño por medio de un *objeto versionable*, y se mantienen las asociaciones con otros *objetos versionables* (que representan a otros objetos de diseño); ii) nivel versiones, donde se representan los distintos estados alcanzados por cada objeto de diseño durante un proyecto. Los objetos que pertenecen a este nivel se denominan *versión de objeto*. Estas versiones de objetos pertenecen a las diferentes *versiones de modelo* que se obtengan a lo largo del proceso de diseño. El enfoque empleado para la administración de versiones, modela el proceso de diseño a partir de una versión de modelo inicial. Las sucesivas versiones de modelo se van obteniendo a medida que se aplican las correspondientes secuencias de operaciones que se traducen en la creación, eliminación, o modificación de algunos de los elementos del modelo origen.

Las preguntas de competencia dentro de este grupo ayudaron a identificar todos esos conceptos para ser incorporados en la ontología. Se presentan algunas de ellas en la Tabla 1.

4.2. Segundo grupo de preguntas de competencia

El segundo grupo de preguntas de competencia (CQG#2) amplía el alcance de la ontología para permitir

[5] https://www.w3.org/TR/rdf-sparql-query/

la representación de conocimiento de diferentes dominios de arquitecturas de software. Para la elicitación de las preguntas de competencia nos basamos en el conocimiento que tenemos del modelo implementado por TracED(aaS). La herramienta permite la definición de diferentes dominios de diseño de arquitecturas de software, de manera que en un proyecto en particular se puedan manejar los conceptos específicos del dominio del sistema que se pretende construir (arquitecturas móviles, arquitecturas en la cloud, etc.). El dominio se especifica en términos de los *tipos de objetos de diseño* que se desean modelar y las *operaciones* que son aplicables a los mismos. Cada tipo de objeto de diseño posee un conjunto de *propiedades*. En otras contribuciones hemos definido diferentes dominios, y entre ellos un dominio genérico para el proceso de diseño de arquitecturas de software que toma conceptos comunes de los lenguajes de descripción de arquitecturas de software, el método de diseño ADD, de las diferentes vistas con que puede describirse una arquitectura de software [2], y conceptos relativos a razonamiento arquitectónico. En la Figura 2 se muestra una definición de ese dominio genérico en el Editor de dominios de TracED, en la cual se observan los tipos de objetos de diseño definidos, sus propiedades y operaciones.

Las preguntas de competencia que se elicitaron dentro de este grupo, están relacionadas con los tipos de objetos de diseño que se van a manejar en el dominio, los posibles patrones de diseño a aplicar y otras operaciones de diseño aplicables según los tipos de elementos que se manejarán en el dominio. También abarcan cuestiones sobre qué conocimiento del contexto del diseño de la arquitectura de software nos interesa, es decir, requerimientos a cumplir, restricciones, suposiciones, etc. En este grupo de preguntas de competencia se incluyen también las cuestiones relativas al conocimiento generado durante el proceso de diseño, es decir, las decisiones tomadas, elementos arquitectónicos resultantes, razonamiento detrás de las decisiones tomadas, etc. Algunas de las preguntas de competencia del grupo CQG#2 y términos derivados se presentan en la Tabla 2.

Tabla 1. Primer grupo de preguntas de competencia

Preguntas de Competencia	**Términos identificados**
¿Qué modelos intermedios o diferentes versiones de la arquitectura de software se generaron durante la ejecución del proyecto de diseño?	Modelo, Proyecto, Versión
¿Qué decisiones arquitectónicas u operaciones de diseño se tomaron durante el proceso de diseño, la cuales dieron lugar al producto final?	Decisión arquitectónica, Operación, Producto
¿Qué productos de diseño se obtienen a partir de la toma de una decisión/ejecución de operación?	Producto de diseño, Decisión, Operación
¿Cómo evoluciona durante el proceso de diseño un determinado objetos de diseño (elemento arquitectónico) que aparece en una versión del modelo arquitectónico? ¿Qué versiones de estos objetos se alcanzaron? ¿A partir de la ejecución de qué operación surgió?	Objeto de diseño, Versión de objeto
¿Cuales fueron los elementos arquitectónicos afectados por la ejecución de una operación o secuencia de operaciones?	Secuencia de operaciones, Operación, Elemento arquitectónico
¿Cuál es el objetivo perseguido por una decisión o secuencia de operaciones?	Operación, Objetivo perseguido
¿Qué decisiones se tomaron (secuencias de operaciones se tomaron) para satisfacer un objetivo?	Decisiones, Secuencia de operaciones, Satisfacer, Objetivo

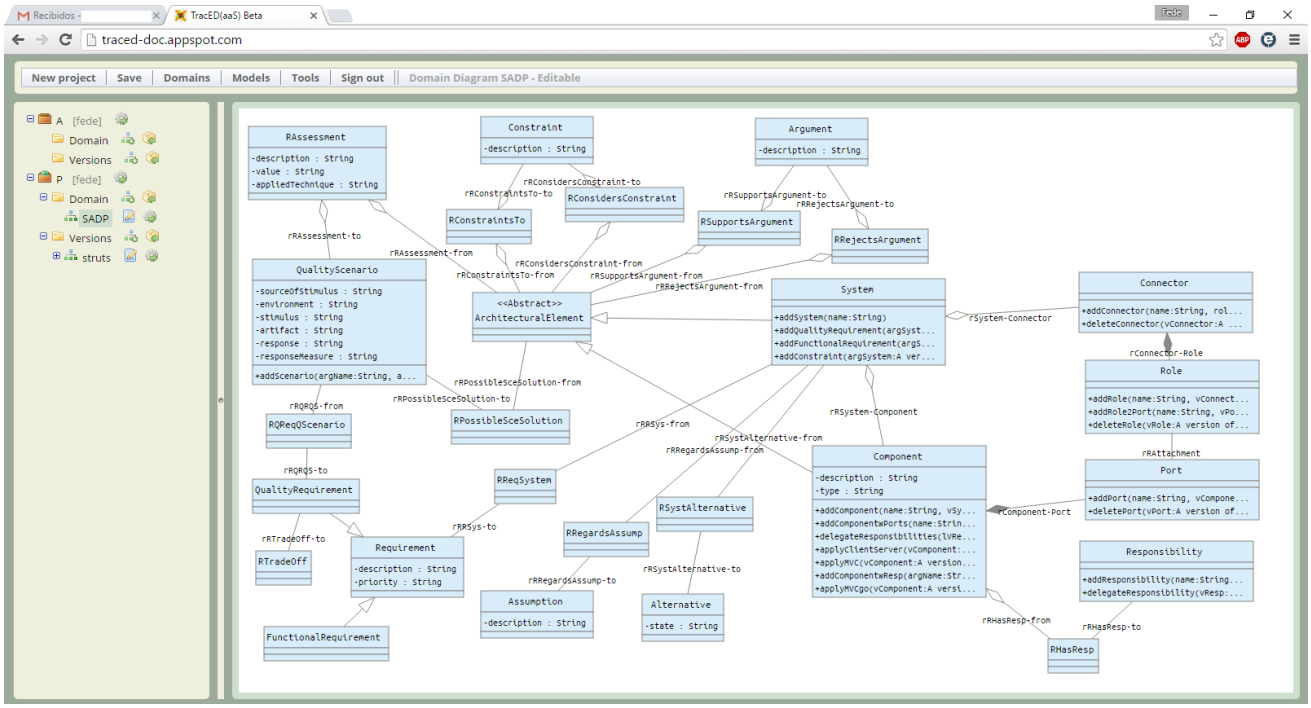

Figura 2. Vista parcial de un dominio especificado en TracED(aaS) de donde se tomaron conceptos a incluir en SAKOnto

Tabla 2. Segundo grupo de preguntas de competencia

Preguntas de Competencia	**Términos identificados**
¿Qué patrones de diseño / estilos arquitectónicos / tácticas se conocen y han sido catalogados? ¿En qué contexto se aplican? ¿Qué atributos de calidad/requerimientos no funcionales favorecen/desfavorecen?	Patrón de diseño, Estilo arquitectónico, Táctica, Atributo de calidad, A favor de, En contra de, Requerimiento no funciona
¿Qué requerimientos de calidad se conocen y pueden ser necesario satisfacer en una arquitectura de software?	Requerimiento de calidad, Satisface
¿Qué restricciones se impusieron sobre/limitan a la arquitectura de software a construir? ¿Existen suposiciones que deben asumirse?	Suposición, Asume, Restricción, Limita a
¿Qué requerimientos de calidad para la arquitectura de software son contrarios o tienen un trade-off entre ellos?	Trade-off, Contrario-a
¿Qué requerimientos funcionales se especifican sobre una arquitectura de software?	Requerimiento funcional
¿De qué componentes / módulos /nodo / interfaces consta el modelo buscado?	Componente, Modulo, Nodo, Interfaz
Dado un componente de la arquitectura ¿A través de qué puertos puede comunicarse con otros? ¿Cuáles son sus responsabilidades en la arquitectura de software?	Componente, Puerto, Responsabilidad
¿Qué argumentos respaldan una decisión dada? ¿Cuáles son los argumentos que respaldan la necesidad de que esté presente un cierto elemento arquitectónico?	Elemento arquitectónico, Argumento
¿Qué alternativas se exploraron en la definición de la arquitectura de software de un sistema? ¿Cómo se relacionan las alternativas exploradas entre sí? ¿Son complementarias? ¿Son opuestas?	Alternativa, Sistema, Opuesta, Complementaria

¿Cómo se evalúa la satisfacción de cierto requerimiento de calidad? ¿Es posible verificar escenarios de calidad que fueron definidos para el requerimiento?	Requerimiento de Calidad, Evaluación, Satisface, Escenario de calidad
¿Qué operaciones de diseño arquitectónico pueden utilizarse en el dominio?	Operaciones

4.3. Especificación de la jerarquía de clases

A partir de las preguntas de competencia que se definieron, se identificaron y enumeraron los términos que son importantes para la ontología. Inicialmente, se generó una lista integral de términos sin tener en cuenta si existe algún solapamiento entre los conceptos que representan, las relaciones entre los términos, o cualquier propiedad que los conceptos puedan tener. Estos términos son los que se listaron en la segunda columna de las Tablas 1 y 2.

Luego, a partir de estos términos se fueron definiendo las clases y propiedades de SAKOnto. Dado que SAKOnto se concibió considerando el modelo de representación de procesos de diseño de arquitecturas de software (materializado en la base de datos de la herramienta TracED(aaS)), muchos conceptos surgieron de preguntas que consideraban la forma de representación de productos del diseño que posee este enfoque. Otros conceptos surgieron en forma directa a partir del modelo subyacente a la herramienta TracED.

Los siguientes pasos en la construcción de SAKOnto fue desarrollar la jerarquía de clases [19] y definir las propiedades de los conceptos (*slots*), los cuales están estrechamente relacionados. Se empleó un proceso de desarrollo usando una combinación de los enfoques top-down y bottom-up, comenzando por definir los conceptos más sobresalientes, luego generalizarlos y/o especializarlos apropiadamente.

Dado que las clases por sí solas no proveen suficiente información para responder las preguntas de competencia se describió la estructura interna de los conceptos. Para lo cual, a partir de los términos identificados se analizó si se trata de propiedades intrínsecas, extrínsecas, partes (si el objeto que se está describiendo es estructurado) y relaciones con otros conceptos.

SAKOnto fue definida empleando la herramienta Protégé. Protégé es un editor de ontologías gratuito y de código abierto, y un sistema de gestión de conocimiento. Provee una interfaz gráfica para la definición de ontologías. Además incluye clasificadores deductivos para validar que los modelos sean consistentes y para inferir nueva información en base al análisis de una ontología. El modelo creado se representó en lenguaje OWL[6] (Ontology Web Language), el cual es un lenguaje de etiquetado semántico para publicar y compartir ontologías en la Web. Se trata de una recomendación del W3C, y puede emplearse para representar ontologías de forma explícita, es decir, permite definir el significado de términos en vocabularios y las relaciones entre aquellos términos. En realidad, OWL es una extensión del lenguaje RDF (Resource Description Framework) y emplea las tripletas de RDF, aunque es un lenguaje con más poder expresivo que éste. El uso de este lenguaje tiene la ventaja de que está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos, lo cual lo hace adecuado para los objetivos de recuperación y reuso de conocimiento que tiene el enfoque del que forma parte la ontología SAKOnto.

De esta manera, basándose en OWL, SAKOnto se formalizó en término de clases, individuos, y propiedades de estos individuos y clases, así como relaciones que existen entre ellas. En particular, OWL permite expresar los siguientes diferentes tipos de propiedades: (i) propiedades de objeto, las cuales permiten definir relaciones entre individuos; (ii) propiedades de tipo de dato, las cuales definen relaciones entre individuos y literales (por ejemplo, strings, enteros, etc.); y propiedades de anotaciones (annotations), que pueden ser usadas para describir metadatos de individuos, clases y propiedades, tales como el idioma en que se encuentra, definiciones y comentarios de estos conceptos. Además, sobre las propiedades de objeto definidos se especificó el dominio y rango, así como también tipo, valores admitidos y cardinalidad, sobre las propiedades de dato.

Para completar la definición de la ontología, y dada la suposición de mundo abierto que se tiene en Protegé, se agregaron los axiomas de *disjoint*, *closure*, y *covering* necesarios para que sobre la ontología pueda ejecutarse un razonador (lo cual se llevará a cabo en una etapa posterior), y que éste funcione correctamente. Dado que OWL asume que las clases se solapan entre ellas a menos que explícitamente se diga lo contrario, se incorporaron axiomas *disjoint* indicando qué clases son disjuntas. En la Figura 3 se presenta una vista parcial de la especificación de SAKOnto en Protegé, donde se incluyen los conceptos que fueron mencionados.

[6] https://www.w3.org/TR/owl2-quick-reference/

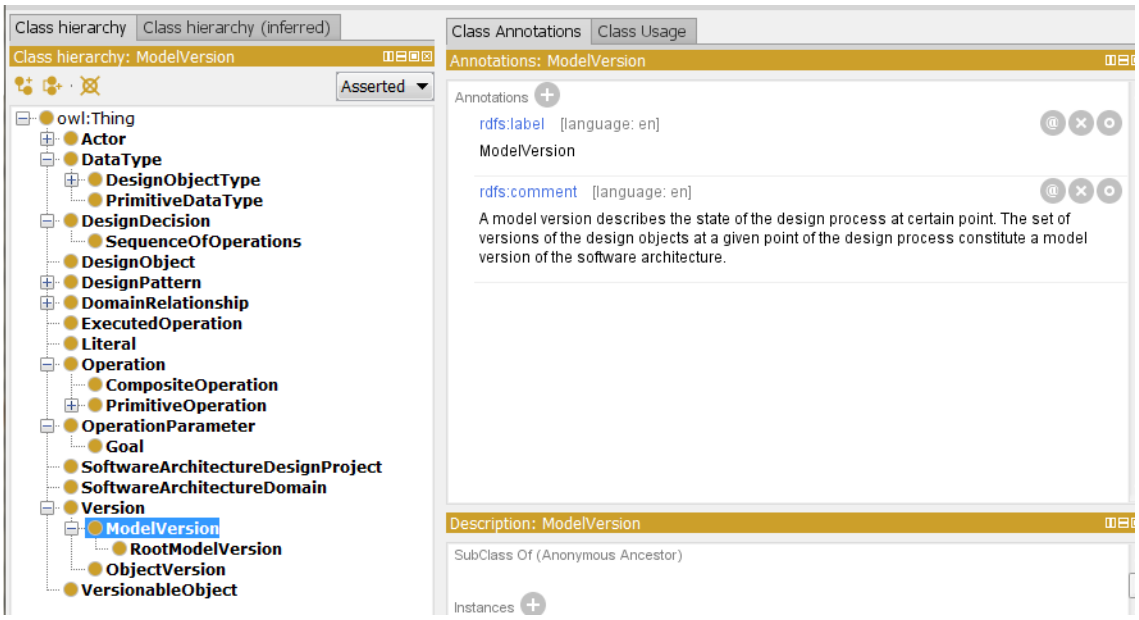

Figura 3. Primer conjunto de conceptos definidos como clases en SAKOnto

4.4. Extensión de SAKOnto para un dominio en particular

La característica de extensibilidad SAKOnto permite incorporar en ella conocimiento de cualquier dominio de arquitecturas de software (por ejemplo, arquitecturas orientadas a servicios, arquitecturas móviles, arquitecturas en la nube). En la Figura 2, se presentó una ventana de la herramienta TracED(aaS) donde se visualiza el dominio definido para un caso de estudio que se llevó a cabo con el soporte de dicha herramienta [15]. Este dominio, es un dominio genérico para el proceso de diseño de arquitecturas de software, que incluye conceptos de estructura/comportamiento/despliegue y de razonamiento arquitectónico, y fue utilizado para ilustrar la incorporación de conocimiento de dominios de arquitecturas de software en SAKOnto.

Tal extensión se realiza a partir de la especialización de la clase *DesignObjectType*, es decir definiendo subclases de dicha clase en la jerarquía. Dado que en SAKOnto se decidió considerar tanto elementos descriptivos de la arquitectura de software, es decir, relativos a la estructura/comportamiento, así como también elementos que permiten describir el razonamiento que respalde las decisiones tomadas, se especializó *DesignObjectType* creando las subclases *StructureBehaviourDeploymentDesignObjectType* y *ArchitecturalRationaleDesignObjectType*. En consecuencia para incorporar en la ontología conceptos de un dominio particular, se especializaron dichas clases según correspondía al concepto. Por ejemplo, la clase *StructureBehaviourDeploymentDesignObjectType* se especializó en las clases que representan a conceptos específicos del dominio relativos a la estructura y comportamiento o despliegue de la arquitectura, tales como *Component*, *Interface*, *Responsibility*, *Role*, *Port*, *Connector*, *Module*, entre otros. Vale indicar que tales conceptos fueron identificados con las preguntas de competencias del grupo *CQ#2*.

Por otro lado, la clase *ArchitecturalRationaleDesignObjectType* se especializó con los conceptos *Argument*, *Assumption*, *Constraint*, *Requirement*, *QualityRequirement*, y *Alternative*, entre otros. Las relaciones de dominio posibles entre los tipos de objetos de diseño del dominio se especificaron como propiedades de objeto. Para ello se creó la propiedad de objeto *associationAmongArchitectureElements*, y las posibles relaciones de dominio se definieron como subpropiedades de ésta. Por ejemplo, entre la clase *System* y *QualityRequirement* se definió la propiedad *hasToBeSatisfied* que permite indicar que un requerimiento de calidad debería ser cumplido por la arquitectura desarrollada. La propiedad de objeto *hasResponsibility* permite expresar que cierto componente (*Component*) tiene una determinada responsabilidad (*Responsability*). Para expresar la relación entre una restricción y un conjunto de elementos arquitectónicos se definieron las propiedades de objeto *constrains* o *considersConstraint*, sea que se desee expresar que una restricción ha sido "impuesta" sobre la arquitectura o que ha sido "considerada" o tenida en cuenta al momento de realizar cierta decisión

arquitectónica. Dado que la herramienta TracED, a partir de la cual se poblará la ontología, emplea relaciones de dominio reificadas (relaciones convertidas en objetos para posibilitar su versionamiento) para hacer posible el versionamiento de relaciones entre estos elementos arquitectónicos, se consideró además la posibilidad de definirlas como clases en SAKOnto. De esta manera, se cuenta con una forma alternativa para la representación de conocimiento sobre asociaciones entre objetos. Las mismas relaciones que fueron definidas como propiedades de objeto, se definieron como subclase del concepto *DomainRelationship*. Por ejemplo, los conceptos *HasResponsibility* (que establece la relación entre un componente y una responsabilidad), *Attachment* (que establece la relación entre un puerto de un componente y un rol de un conector), *Trade-off* (entre dos requerimientos de calidad), entre otros. En la Figura 4 se muestran los conceptos del dominio considerado en la jerarquía de clases definidoa en Protegé.

Además, en la ontología se definieron las propiedades de datos y de objeto necesarias para caracterizar individuos y establecer relaciones entre ellos. En la Figura 5, se presenta una pieza del archivo de especificación en donde observa la definición de las clases *SoftwareArchitectureDesignProject* y *SoftwareArchitectureDomain*, y la propiedad de objeto *domain* que permite expresar que un cierto proyecto de diseño manejará conceptos de un cierto dominio.

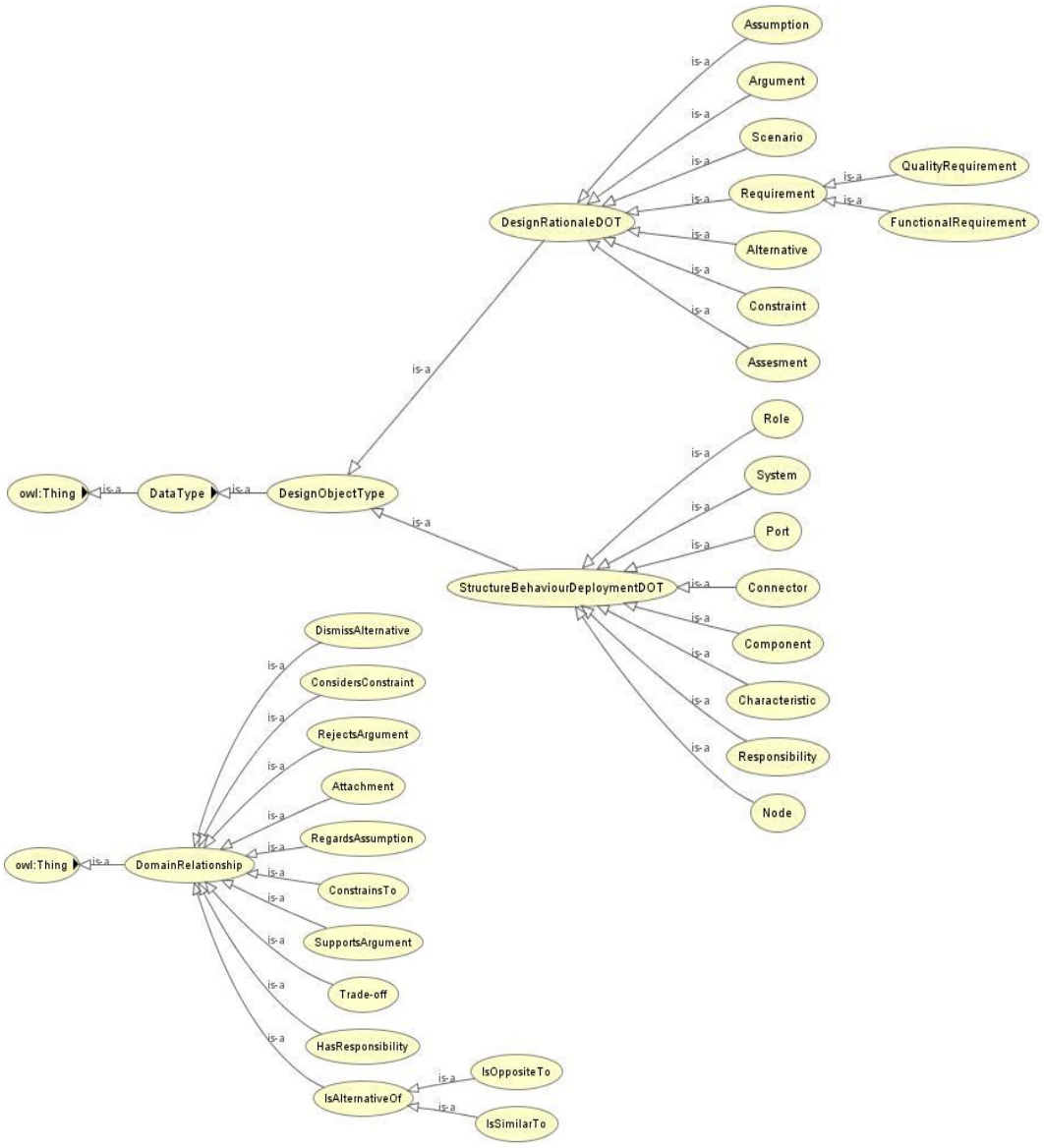

Figura 4. Conceptos de un dominio en SAKOnto

```
<!-- http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDesignProject -->

<owl:Class rdf:about="http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDesignProject">
    <rdfs:comment xml:lang="en">Process to design/develop the software architecture of a system.</rdfs:comment>
</owl:Class>

<!-- http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDomain -->

<owl:Class rdf:about="http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDomain">
    <rdfs:comment xml:lang="en">A domain is defined for a particular design project to be carried out. It comprises the definition
    of the design object types, rationale types, and operations able of being applied during the execution of the design process.
    </rdfs:comment>
</owl:Class>

<!-- http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#domain -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#domain">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#AsymmetricProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDesignProject"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/luciana/ontologies/2016/7/SAKOnto#SoftwareArchitectureDomain"/>
</owl:ObjectProperty>
```

Figura 5. Especificación de clases *SoftwareArchitectureDesignProject* y *SoftwareArchitectureDomain* y la propiedad de objeto *domain* entre éstas.

4.5. Población de la ontología

Dado que aún no se cuenta con el desarrollo de módulo AK Loader que permitirá importar conocimiento arquitectónico generado por expertos de diferentes dominios y mediante la captura de la ejecución de diferentes proyectos de diseño con la herramienta TracED (aaS), se realizó la importación manual de una porción del conocimiento que se tiene representado en su base de datos.

Para ello se tomó un caso de estudio sencillo [13], y, con el apoyo de un servicio de la herramienta que permite obtener la historia completa de un proceso de diseño, junto con revisión manual, se llevó a cabo la importación del mismo en la ontología SAKOnto, generándose así una población de individuos suficiente para llevar a cabo una validación sencilla para probar los conceptos propuestos. La ontología poblada se cargó en un servidor Apache Jena Fuseki[7], y se procedió a realizar algunas consultas SPARQL simples, de manera de responder a algunas de las preguntas de competencia que se enunciaron al principio. En la Figura 6, se observa una consulta para conocer las versiones de modelo generadas durante la ejecución de un proyecto (*Project_01*).

Dado que Fuseki presenta la respuesta como una lista de tripletas, en la figura se muestra el resultado en forma de gráfica para ver los individuos resultantes y las relaciones entre ellos. La consulta se puede ampliar para obtener las secuencias de operaciones ejecutadas para generar cada una de las diferentes versiones, las operaciones individuales involucradas, argumentos con los que dichas operaciones se ejecutaron, los resultados de esas operaciones. Obviamente, la calidad del conocimiento que se pueda recuperar dependerá de la riqueza con que se capturó el mismo durante el proceso de diseño.

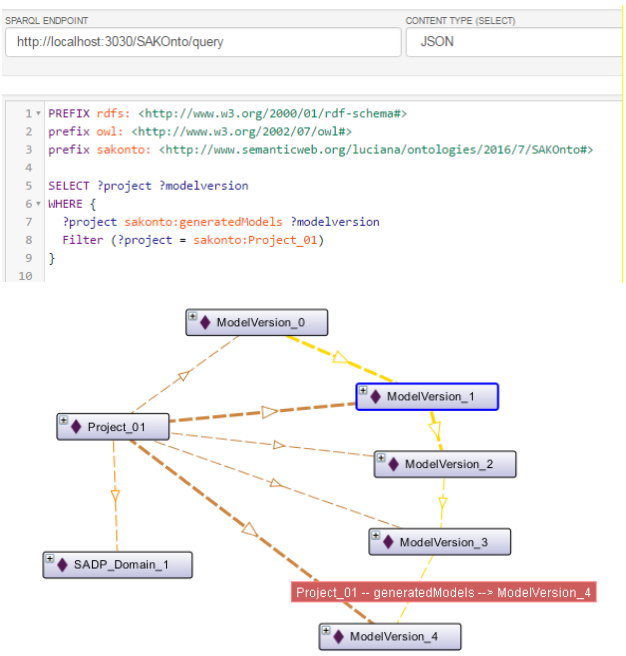

Figura 6. ¿Qué versiones de modelo se obtuvieron durante la ejecución del proyecto?

5. Integración con otras ontologías de arquitecturas de software

Una ventaja asociada al uso de ontologías es que se puede partir de una ontología ya desarrollada, tomándola como base, y refinar y extender los recursos existentes para el dominio y tarea particular para el cual la vamos a usar. Como parte del proceso de contrucción de SAKOnto, consideramos además la posibilidad de

[7] https://jena.apache.org/documentation/serving_data/

reutilizar ontologías existentes. Para ello llevamos a cabo una investigación exhaustiva de la literatura en relación a arquitecturas de software y ontologías de conocimiento arquitectónico. Se encontramos numerosas propuestas en la literatura [3, 6, 7, 9, 13, 14, 17], pero pocas brindaban acceso a las ontologías propuestas en algún formato que permita incorporarlas o importarlas a la ontología propuesta bajo nuestro enfoque. De todas maneras, este relevamiento fue provechoso para recabar ideas de diferentes propuestas de modelado de la ontología de conocimiento arquitectónico. Podemos destacar, entre las contribuciones analizadas, la de Guessi y colab. [9], quienes propusieron OntolAD, la cual es una ontología de descripciones arquitectónicas basada en los conceptos del estándar ISO/IEC 42010. En cuanto a conocimiento arquitectónico de razonamiento, existe la ontología NDR [13] que permite describir razonamiento arquitectónico usando grafos de interdependencia de metas "blandas" (soft-goals), los cuales describen requerimientos no funcionales como metas, que pueden ser descompuestas en otras submetas. Si bien el enfoque para modelar razonamiento es diferente del empleado en el modelo de representación de conocimiento en el que se basa SAKOnto, puede ser útil incorporar los conceptos ontológicos que define para incorporar cuestiones como razonamiento, argumentación, y evaluación de decisiones de diseño adoptadas o descartadas.

6. Conclusiones

En este trabajo presentamos la construcción de la ontología SAKOnto. Esta ontología incluye conceptos que posibilita la representación de conocimiento arquitectónico de diversas categorías, desde conocimiento general, como el que reside en los numerosos catálogos de patrones arquitectónicos existentes, conocimiento de contexto y de diseño arquitectónico. Esta ontología puede poblarse con conocimiento que haya sido capturado mediante una herramienta para la representación de procesos de diseño de arquitecturas de software, donde el conocimiento se conserva de manera estructurada. La ontología propuesta es un componente clave para la obtención y explotación de conocimiento arquitectónico. Como trabajo a futuro, continuaremos trabajando en las posibles aplicaciones de extracción de conocimiento arquitectónico. Una de ellas es el reuso de conocimiento de razonamiento arquitectónico de proyectos de diseño existentes, de manera que sea posible reconocer problemas de diseño que ya han sido abordados en el pasado, y que pueden servir como soluciones iniciales para emplearlas como punto de partida de un nuevo diseño, en un nuevo contexto. Otra aplicación que exploraremos es el uso del conocimiento como soporte a procesos de auditoría o validación/chequeos de consistencia de modelos arquitectónicos, de manera de poder determinar la conformidad con estándares o emplearlo en herramientas de diseño. Finalmente otro posibilidad a explorar es, a partir del conocimiento que SAKOnto posibilita mantener e inferir, el desarrollo de herramientas de entrenamiento para arquitectos jóvenes o con poca experiencia, ya que se puede conocer cómo se obtuvo una solución, porqué ciertas decisiones arquitectónicas fueron tomadas, y cuál fue el impacto de las mismas.

7. Agradecimientos

Este trabajo ha sido financiado en forma conjunta por CONICET, la ANPCyT, y la UTN. Se agradece el apoyo brindado por estas instituciones.

8. Referencias

[1] A. Akerman, J. Tyree, Using ontology to support development of software architectures, IBM Syst. J. 45 (4) (2006) 813–825.

[2] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice: Third Edition, Addison-Wesley Professional, 2012.

[3] Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M. A., 10 years of software architecture knowledge management: Practice and future, Journal of Systems and Software, Vol. 116, 2016, Pages 191-205.

[4] Choobdaran, N., Sharafi, S. M., Khayyambashi, M. R., An Ontology-Based Approach For Software Architectural Knowledge Management, Journal of mathematics and computer science, 11 , 93-104, 2014.

[5] De Graaf, K. A., P. Liang, A. Tang, W. R. Van Hage, and H. Van Vliet. 2014. An exploratory study on ontology engineering for software architecture documentation. Comput. Ind. 65, 7 (September 2014), 1053-1064.

[6] De Graaf K. A., Tang, A., Liang, P., Van Vliet, H., Ontology-based Software Architecture Documentation. In 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), pp. 121–130, 2012.

[7] Figueiredo, A. M., Dos Reis, J. C., Rodrigues, M.A., Improving Access to Software Architecture Knowledge An Ontology-based Search Approach, International Journal Multimedia and Image Processing (IJMIP), 2 (1/2), 124-149.

[8] Gruninger M. and Fox M. S.: Methodology for the Design and Evaluation of Ontologies. IJCAI Workshop on Basic Ontological in Knowledge Sharing. Montreal, Canada. 1995.

[9] Guessi, M., D. Moreira, G. Abdalla, F. Oquendo, E. Nakagawa. OntolAD: a formal ontology for architectural descriptions. Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, New York, USA, 1417-1424, 2015.

[10] Hernández, F., Roldán, M.L., Vegetti, M., Gonnet, S., Leone, H. TracED(aaS): Captura y Trazabilidad de Artefactos del Proceso de Diseño, 2° CoNaIISI, San Luis, (996-1007) 2014.

[11] ISO, May 2011. Systems and Software Engineering - Architecture Description. ISO/IEC/IEEE ¬-42010, pp. 1-46.

[12] Kruchten, P., P. Lago, H. van Vliet, Building up and exploiting architectural knowledge, in: QoSA'05: Second International Conference on Quality of Software Architectures, Springer, Berlin/Heidelberg, 2006, pp. 43–58.

[13] López C., V. Codocedo, H. Astudillo, L. M. Cysneiros. Bridging the Gap between Software Architecture Rationale Formalisms and Actual Architecture Documents: An Ontology-driven Approach. Science of Computer Programming. 77(1), 2012.

[14] Marwat, M., J. Sadaqat, Shah, S. Z. Ali Shah, Software Architectural Design Ontology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 7(12), 2013.

[15] Roldán, M.L., S. Gonnet and H. Leone (2010) TracED: a tool for capturing and tracing engineering design processes, Advances in Engineering Software, 41, 1087–1109.

[16] Roldán, M.L., S. Gonnet and H. Leone (2016) Operation-based approach for documenting software architecture knowledge, Expert Systems, Available online.

[17] Sun H., H. Wang, T. Hu. Design Software Architecture Models using Ontology. Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, 2011.

[18] Tang, A., P. Avgeriou, A. Jansen, R. Capilla and M. Ali (2010) A comparative study of architecture knowledge management tools, Journal of Systems and Software, 83, 352–370.

[19] Uschold, M., Gruninger M.: Ontologies: Principles, Methods and Applications. Knowledge Engineering Review. 1996.

[20] Vázquez, H. C., Díaz Pace, J. A., Marcos, C. Uso de Ontologías para Mapear una Arquitectura de Software con su Implementación. Proceedings of the 1st Argentine Symposium on Ontologies and their Applications, CEUR Workshop Proceedings, Vol-1449, 2015, págs. 101-110.