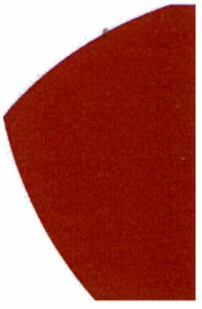
UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Aplicación de Tecnologías de Web Semántica en un Sistema de Gestión de Incidentes
Autores: Tomás Campodónico, Máximo Zarza
Director: Dr. Alejandro Fernández
Codirector: Dra. Alicia Díaz
Asesor profesional: -
Carrera: Licenciatura en Sistemas – Plan 2007



Resumen

En los últimos años se ha hablado mucho acerca de la Web Semántica y su uso para publicar datos, de manera tal que puedan ser entendidos por una computadora. Si bien éste es el propósito principal para el cuál fue propuesta, nosotros creemos que su aplicabilidad no se limita sólo a esa función, sino que también podría ser usada en otros ámbitos de no menor interés. En particular, en este documento trataremos de demostrar las ventajas y desventajas que presenta la utilización de las tecnologías de la Web Semántica en un ambiente cerrado, donde la publicación de los datos no es el objetivo principal. Para ello, utilizaremos la Web Semántica para desarrollar un Sistema de Gestión de Incidentes (Issue Tracker System) ya que estamos convencidos de que se puede sacar provecho a estas tecnologías. Luego del desarrollo, analizaremos cómo se comporta dicho sistema ante los problemas que suelen ocurrir en el desarrollo de software, comparando los resultados con los obtenidos de un sistema semejante desarrollado con las tecnologías clásicas. Llegaremos así a la conclusión de que es totalmente viable utilizar estas tecnologías con el sólo objeto de aprovechar su modelo semántico, pero sin necesidad de publicar información para terceros. No solo es viable, sino que, dependiendo del contexto, puede resultar altamente beneficioso para los desarrolladores y usuarios.

Palabras Claves

Web Semántica – Linked Open Data – Issue Tracker – Integración de datos – Cambio organizacional - Callimachus

Conclusiones

- El prototipo desarrollado evidencia las ventajas de la utilización de la Web Semántica en contraste con las tecnologías convencionales en ambientes corporativos. Estas ventajas inciden tanto sobre los usuarios como sobre los desarrolladores.

- Se encontraron ontologías que permitieron crear un rico modelo de datos sin la necesidad de crear las propias.

Trabajos Realizados

- Se desarrolló un prototipo de sistema web para la gestión de incidentes respecto de cambios organizacionales, utilizando tecnologías de la Web Semántica. El mismo es flexible a la incorporación de nueva y variada información.

- Se diseño un modelo semántico, a partir de ontologías encontradas, para desarrollar un sistema de gestión de incidentes.

Trabajos Futuros

- Contemplar la utilización de vocabulario SEON diseñar el modelo.

- Realizar una implementación de la aplicación utilizando un modelo más completo. De esta manera se lograrla definitivamente un aplicativo, para la gestión de incidentes semántico y genérico, utilizable.

- El prototipo desarrollado no utiliza inferencia. Agregar esta capacidad incrementaria el potencial de la aplicación.

Fecha de la presentación: MAYO 2015

Aplicación de Tecnologías de Web Semántica en un Sistema de Gestión de Incidentes

Tomás Campodónico y Máximo Zarza

11 de mayo de 2015

Resumen

En los últimos años se ha hablado mucho acerca de la Web Semántica y su uso para publicar datos, de manera tal que puedan ser entendidos por una computadora. Si bien éste es el propósito principal para el cuál fue propuesta, nosotros creemos que su aplicabilidad no se limita sólo a esa función, sino que también podría ser usada en otros ámbitos de no menor interés. En particular, en este documento trataremos de demostrar las ventajas y desventajas que presenta la utilización de las tecnologías de la Web Semántica en un ambiente cerrado, donde la publicación de los datos no es el objetivo principal. Para ello, utilizaremos la Web Semántica para desarrollar un Sistema de Gestión de Incidentes (Issue Tracker System) ya que estamos convencidos de que se puede sacar provecho a estas tecnologías. Luego del desarrollo, analizaremos cómo se comporta dicho sistema ante los problemas que suelen ocurrir en el desarrollo de software, comparando los resultados con los obtenidos de un sistema semejante desarrollado con las tecnologías clásicas. Llegaremos así a la conclusión de que es totalmente viable utilizar estas tecnologías con el sólo objeto de aprovechar su modelo semántico, pero sin necesidad de publicar información para terceros. No solo es viable, sino que, dependiendo del contexto, puede resultar altamente beneficioso para los desarrolladores y usuarios.

Índice general

Capítulo 1

Introducción

1.1. Sistemas de Gestión de Incidentes

Los incidentes no son ajenos a ninguna actividad u organización, ocurren en todos lados: en los procesos dentro del desarrollo de software, dentro de la estructura misma de una empresa, en la administración de un hospital, etc. En cada ámbito de nuestras vidas, se debe lidiar con incidentes que ocurren de manera continua e interrumpen el estado normal de las cosas.

El primer objetivo de la gestión de incidentes es restablecer el orden y el correcto funcionamiento del servicio afectado, minimizando el impacto negativo en la organización de forma tal que la calidad y la disponibilidad del servicio se mantengan inalterados. Para facilitar la gestión de estas anomalías, se utilizan los llamados Sistemas de Gestión de Incidentes (**SGI**).

Un Sistema de Seguimiento de Incidentes es un programa informático que administra y mantiene un listado de incidentes que afectan a un determinado objeto u organización. Su principal objetivo es el de asistir a los usuarios para mejorar el tratamiento y resolución de dichos incidentes [17]. Los sistemas existentes al día de la fecha, tales como **Jira**[1] o **Mantis**[2] para el seguimiento de errores en el desarrollo de software, generalmente mantienen la misma información sobre cada incidente.

1.2. Limitaciones de los sistemas existentes

Aunque estas herramientas han evolucionado a lo largo de los años, a tal punto de llegar a ser una herramienta indispensable en muchas organiza-

[1] Jira: https://es.atlassian.com/software/jira

[2] Mantis Bug Tracker: https://www.mantisbt.org

ciones, y podrían parecer bastante completas, creemos que no resuelven de manera eficiente y eficaz el tratamiento de incidentes de gran complejidad. La generalización de los datos que se manejan sobre los incidentes permite que la herramienta pueda ser utilizada en distintos entornos y organizaciones. Pero, por otro lado, se pierde precisión al tener datos generales. La falta de información por culpa de esta generalización no permite manejar datos que podrían ser de utilidad para usuarios o agentes informáticos que necesiten analizar estos incidentes. Por ejemplo, si se tuviera información acerca de qué áreas dentro del sistema/organización son afectadas por el incidente, o quienes son las personas que deben estar involucradas en la resolución del mismo, etc. facilitaría la comprensión de las causas y efectos que tiene el incidente. Algunos ejemplos de datos importantes del contexto que podrían ser de utilidad son:

Personas involucradas: personas involucradas, directa o indirectamente, durante todo el proceso de resolución del incidente. Nos referimos no sólo a las personas que quedan registradas en el sistema por una actividad que realizaron, sino también a aquellas que están afectadas por trabajar en el área afectada, aquellas que estuvieron en el tratamiento de incidentes pasados relacionados con éste, etc. Esta información podría servir, por ejemplo, ante una imposibilidad o complicación en la resolución, para saber a qué personas afecta, de qué manera, y así poder juntarse y llegar a una solución.

Procesos afectados: proceso/s afectado/s, directa o indirectamente, por el incidente. Dentro de todo sistema, existen muchos procesos relacionados entre sí. Generalmente los incidentes afectan a más de un proceso. Teniendo un seguimiento de los procesos (no sólo las áreas), facilita el seguimiento y la comprensión del impacto que tiene en el sistema.

Incidentes relacionados por un determinado criterio: incidentes que se relacionen con el incidente actual, ya sea por compartir áreas afectadas, personas o procesos involucrados.

Ahora bien, el faltante de información en los sistemas actuales no parecería ser motivo suficiente para realizar el cambio que proponemos en este documento. Uno podría pensar que bastaría con agregar estos datos en una nueva versión de los **SGI** para que el problema quedara resuelto. Aunque se incluyera esta información en el modelo de datos de la herramienta, es sabido que una organización o sistema cambia constantemente. Esto significa

que se podría necesitar agregar, quitar o cambiar aún más información de la mencionada anteriormente en dicho modelo. Las herramientas mencionadas anteriormente (**Jira** o **Mantis**), no presentan la flexibilidad necesaria para adaptarse a los cambios, debido a que se modelaron de manera rígida y fueron pensados para un problema/artefacto en particular (incidentes del software). Es así que empiezan a mostrar debilidades recurrentes frente a problemas genéricos y de gran complejidad, en una realidad que sufre cambios continuos.

Tenemos ahora, entonces, que no es sólo la falta de información útil que manejan los **SGI** el problema que intentamos atacar, sino también el comportamiento de la herramienta ante cambios necesarios, obligados por el contexto. Ya vimos que los **SGI** actuales no son lo suficientemente flexibles a la hora de cambios debido a las tecnologías con las que fueron construidos. Es aquí donde entra en juego nuestra propuesta.

Dicha flexibilidad en los **SGI**, puede ser conseguida utilizando tecnologías ya existentes y utilizadas en muchos otros sistemas: la Web Semántica. Estas tecnologías permitirían utilizar modelos de datos flexibles, dándole al sistema la habilidad de adaptarse ante futuros cambios. Pero esta hipótesis está en tela de juicio ya que las tecnologías de la Web Semántica todavía no fueron explotadas en problemáticas del tipo mencionado.

1.3. Otras motivaciones

1.3.1. La organización y su entorno

Una organización es un sistema socio-técnico incluido en otro sistema más amplio que es el mismo mundo en donde se encuentra; influyéndose mutuamente. Dentro de los componentes que forman a una organización podemos nombrar: el componente *psicosocial*, compuesto por individuos y grupos en interacción; el componente *técnico*, el cual se refiere a los conocimientos necesarios para el desarrollo de tareas; y el componente *administrativo*, que relaciona a la organización con su medio y establece los objetivos, desarrolla planes de integración, estrategia y operación, mediante el diseño de la estructura y el establecimiento de los procesos de control [33]. Es muy común que cualquier observador al pensar en una organización, se la imagine como una estructura jerárquica, tal cual muestra la Figura 1.1.

Este concepto teórico de organización refleja una forma de pensar el diseño del trabajo donde se pone especial importancia en la división de tareas

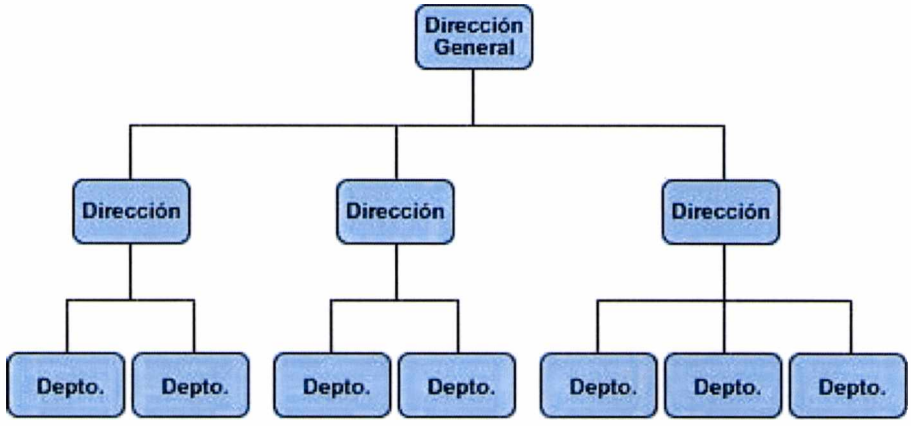


Figura 1.1: Visión jerárquica organizacional

y la consecuente especialización. Este tipo de organización ha sabido ser exitosa (todavía en muchos casos sigue siéndolo) en otros tiempos, en épocas donde las condiciones del contexto eran predecibles, los cambios llevaban un período de tiempo importante, la oferta de los productos/servicios era homogénea y en serie, los mercados predominantemente internos y los clientes con otros niveles de exigencia [33].

Cambio organizacional

La globalización y la constante innovación en campo tecnológico resultan en un ambiente de negocios continuamente en evolución. La explosión de las redes sociales y de los dispositivos móviles revolucionaron a la industria e incrementaron la necesidad de cambios.

Las compañías modernas necesitan responder rápidamente a las cambiantes condiciones de negocios de su entorno, donde los cambios suceden de manera continua y cada vez más frecuente. Estos cambios en el entorno y en las reglas de negocios, llevaron a que las organizaciones que quieran mantenerse competitivas, manejen grandes cantidades de información, integrando datos de distintas fuentes ubicadas en diferentes áreas. La habilidad para administrar y adaptarse a los cambios organizacionales es una de las características más importantes en las organizaciones actuales. De esta manera las compañías mejoran en la toma de decisiones basadas en una comprensión global a partir de estos datos. El logro de este enfoque, sin embargo, no es

una tarea sencilla debido a que la estructura, la cultura y los procesos de las organizaciones suelen ser resistentes a los cambios [33][12].

El cambio organizacional es una estrategia que hace referencia a la necesidad de un cambio. Esta necesidad se basa en la visión de la organización para que haya un mejor desempeño administrativo, social, técnico y de evaluación de mejoras. En otras palabras, el cambio organizacional es la necesidad de cambiar procesos dentro de una organización para lograr mejoras de desempeño en varios aspectos [12].

La mayoría de las organizaciones, es motivada a cambiar por innovaciones externas más que movimientos internos. Cuando estos cambios en el entorno se producen, las organizaciones que se adaptan más rápido, consiguen una ventaja importante ante sus competidores [49]. Y esto puede resultar en grandes beneficios, o grandes pérdidas.

Organizaciones dinámicas

Las organizaciones como sistema dinámico, evolucionan y generan datos. Los datos generados en un determinado punto en el tiempo, representan el estado de la organización en ese instante. Es decir, los datos pueden verse como un elemento del que se puede derivar conocimiento y por lo tanto deben tratarse como un patrimonio de vital importancia para la empresa. Ahora bien, sin la interpretación de un observador (en general suele ser una persona, hecho que cambia ante la aparición de la Web Semántica), la disposición de los datos no tiene ningún significado útil, son simples datos, valores sin sentido. Los datos sumados a la interpretación de un agente, humano o informático, forman lo que se denomina información [33].

Las organizaciones actuales tiene varias fuentes de donde sacan los datos. Muchas internas, pero también las hay externas a la empresa. La integración de datos a través de las distintas áreas o departamentos es un desafío con el que todas las organizaciones deben luchar. La integración de todos los datos dentro de una organización, no suele ser una tarea sencilla de realizar. Depende mucho de los modelos de datos utilizados pero también de las tecnologías con las que los distintos sistemas informáticos fueron desarrollados [49]. Las tecnologías líderes utilizadas en los programas informáticos suelen estar basadas en modelos relacionales.

Pero la tarea de integración de datos se complica aún más cuando no solo se manejan datos internos a la compañía. Muchas organizaciones comenzaron a utilizar modelos de negocios colaborativos, utilizando información y componentes creado por otras empresas. Muchas veces la información importada no tiene la misma estructura que la generada internamente siendo

imposible utilizar la información de ambas fuentes sin antes tener que tratar los datos externos. Esto se vuelve realmente un desafío muy importante cuando las compañías importan desde varias fuentes y deben administrar innumerables proyectos que comparten dichos datos. Muchas veces se hace inviable o injustificable este tratamiento de datos, y entonces es necesario que las compañías desarrollen una arquitectura de datos flexible con el objetivo de poder integrar datos externos [49].

Los principios de diseño y tecnologías de Web Semántica proveen un framework para hacer todo esto posible [49], de modo que las empresas continúen enfocadas en sus objetivos de negocio mientras utilizan de manera eficaz los datos (internos y externos) que les sean de utilidad.

1.4. Una solución con la Web Semántica

La Web Semántica presenta una nueva forma de estructurar los datos sobre la web actual ofreciendo beneficios como, procesamiento automático de información por computadoras, facilidad en la integración de informacion, etc. De este modo le termina dando a la web un mayor potencial [55]. A pesar de que es una tecnología relativamente nueva y que todavía falta mucho por investigar y explotar, ya se empiezan a ver pioneros que la utilizan en sus aplicaciones y mejoran considerablemente la experiencia del usuario y su potencia de procesamiento automático e integración de la información [18][23]. Las posibilidades que abre esta nueva web parecen ser muchas, aunque también implicaría un cambio en todos los agentes web utilizados en la actualidad [55].

Uno de los objetivos del presente trabajo es dar respuesta a cómo la web semántica puede ser utilizada para mejorar los sistemas de issue tracking existentes. Creemos, e intentaremos mostrar, que la utilización de las tecnologías de Web Semántica le dan mayor flexibilidad y abre un gran abanico de posibilidades a la gestión de incidentes a través de un sistema informático extensible y flexible, lo que implica una mejoría con respecto al modelo orientado a objetos/relacional.

El objetivo general del trabajo es documentar el rol, los beneficios y los desafíos en el uso de tecnologías de Web Semántica para la construcción de aplicaciones de gestión de cambio organizacional e incidentes; en particular, en lo que se refiere a la construcción y mantenimiento de los modelos de conocimiento involucrados y al soporte que los mismos proveen en los procesos de toma de decisión. Específicamente, nos concentramos en estrategias para incorporar tecnologías de Web Semántica en la construcción de un gestor de

incidentes. Este está compuesto de una red de modelos (*red de ontologías*) que brindan información acerca de la organización, del foco de los incidentes reportados (p.e., artefacto de software), del ciclo de vida de los incidentes, y de la participación de los actores involucrados en los procesos de reporte, análisis y resolución de incidentes.

La estrategia planteada se materializa en un prototipo de sistema web para la gestión de incidentes, en base a las tecnologías de la Web Semántica detalladas en el Capítulo 3. El mismo es flexible a la incorporación de nueva y variada información acerca de la organización y el foco de los incidentes.

1.5. Contribuciones de esta tesis

Además de evidenciar, con un prototipo de sistema, cómo utilizar efectivamente las tecnologías de la Web Semántica para la construcción de un gestor de incidentes, este trabajo da respuesta a las siguientes preguntas:

1. Para el caso particular de un sistema de gestión de incidencias, qué cambios implica la adopción de las propuestas de la Web Semántica (en términos de arquitectura de software y del proceso de desarrollo), respecto de las metodologías tradicionales basadas en modelos relacionales.

2. ¿Cuáles son las ontologías existentes que pueden reutilizarse en el modelado de la gestión de incidentes y de qué forma se las debe integrar?

3. ¿Cuáles son las fortalezas, debilidades, oportunidades y desafíos vinculados a la utilización de tecnologías de la Web Semántica en la construcción de herramientas de soporte al cambio organizacional? En particular, ¿cómo se compara a los métodos tradicionales de modelado relacional y persistencia en bases de datos relacionales?

1.6. Organización del documento

A continuación, en el Capítulo 2 se presentan los requerimientos de un sistemas de gestión de incidentes, que nos sirve para demostrar algunas de las conclusiones a las que llegamos. Además, se comparan los sistemas más utilizados hoy en día para decantar sus limitaciones. La estrategia general elegida para la construcción del gestor de incidentes se presenta en el Capítulo 3. En particular, la Sección 3.1 presenta la arquitectura del sistema (la cual es consistente con los principios de la Web Semántica). Puntualmente,

en la Sección 3.1.4 enfoca aquellas características del sistema (y su naturaleza semántica) que hacen a la flexibilidad ante cambios y a la integración del sistema en la organización. La Sección 3.2.1, introduce los principios fundamentales de la Web Semántica y la Sección 3.3 compara la estrategia seguida con la de modelos mas tradicionales (siendo esta una de las contribuciones principales de la tesis). El Capítulo 4 discute el estado de arte en la materia, y presenta algunos trabajos relacionados. El Capítulo 5 presenta los vocabularios seleccionados (la red de ontologías) para el modelado de la información del sistema (otra de las contribuciones principales de este trabajo). El Capítulo 6 presenta el diseño general de la aplicación, la cuál se basa en el framework Callimachus, y discute como el mismo cubre los requerimientos. El Capítulo 7 discute fortalezas y debilidades del uso de web semántica en el marco de la estrategia elegida (otra de las contribuciones principales) . Finalmente, el Capítulo 8 presenta conclusiones y líneas de trabajo futuro.

Capítulo 2

Sistema de gestión de incidentes

En este capítulo se define en detalle los sistemas de gestión de incidentes pasando por su historia, marcando sus características más importantes y finalmente, expresando cuáles son, a nuestro entender, las limitaciones que presentan. La idea central de este capítulo es presentar el problema que intentamos resolver mediante la presente investigación.

2.1. Gestores de Incidentes: historia y características

2.1.1. Incidentes

Un incidente representa una alteración en el estado normal de un objeto o servicio [45]. Cotidianamente debemos lidiar con este tipo de alteraciones en todo tipo de entornos. La correcta gestión de estos incidentes, pueden reducir mucho el impacto que estos tengan en el entorno.

Dentro de un **SGI**, el reporte de un incidente se conoce como ticket. [11][19] Un ticket contiene información necesaria para su administración y resolución. Esta información varía de un **SGI** a otro, aunque la mayoría presentan muchos datos en común, entre los que se encuentran[1]:

Type: forma en la que el incidente afecta al objeto. Por Ej.: nueva funcionalidad, cambio de funcionalidad, error, etc.

[1]Los conceptos se mantuvieron en inglés para respetar el idioma con las ontologías utilizadas

Name: texto con el cuál se puede identificar al incidente. Se deben elegir nombres cortos aunque representativos para facilitar la identificación de cada incidente.

Summary: descripción detallada del incidente. La descripción inicial es cargada por el usuario que crea el incidente en el sistema, pero puede ser modificada o ampliada a lo largo de su tratamiento.

Assignee: persona asignada a continuar con el tratamiento del incidente. Esta persona puede ir cambiando a lo largo del ciclo de vida.

Reporter: persona que cargó el incidente al sistema de seguimiento. Por lo tanto, no cambia a lo largo del ciclo de vida.

Priority: nivel de urgencia con la que se debe tratar el incidente. Generalmente, los **SGI** traen una lista de prioridades precargadas. Los usuarios encargados de la resolución usan estos campos para establecer un orden de cuál incidente tratar primero.

Status: etapa dentro del ciclo de vida en la que se encuentra el incidente. Por Ej.: *abierto*, *resuelto*, *para testear*, etc.

Fix version (solo para accidentes de software): número de versión del software en la que el incidente estará resuelto.

Create date: fecha en que el incidente fue reportado.

Last update date: fecha en que el incidente fue actualizado (cambios de estados, incorporación de archivos) por última vez.

Components: parte/s del objeto u organización afectadas, directa o indirectamente, por el incidente.Por Ej.: *área de sistemas*, *área contable*, etc.

Labels: etiquetas que categorizan al incidente y facilitan la búsqueda e indexación.

Comments: notas que los usuarios, eventualmente, agregan para dejar por escrito información relevante que no se puede cargar en otros campos. Por ejemplo, algún tipo de bloqueo en la resolución.

History: listado de actividades (actualizaciones de campos, nuevos comentarios, etc) que se realizaron sobre el incidente desde su creación.

Files: archivos que contienen información relevante que se incorporan para facilitar el seguimiento y tratamiento del incidente.

Parent task: en algunos casos, puede ser que el incidente a tratar sea una parte pequeña de un incidente más grande. Este campo contiene una referencia a dicho Incidente.

Subtasks: en el caso que el incidente pueda descomponerse en incidentes de menor tamaño para facilitar su tratamiento, se guarda una referencia a esos sub-incidentes.

Como mencionamos anteriormente, un Sistema de Gestión de Incidentes (**SGI**) es un sistema informático que administra y realiza un seguimiento de los incidentes que afectan a un determinado objeto o servicio dentro de una organización [17]. La gran mayoría de los **SGI** que se usan actualmente fueron pensados para tratar con incidentes ocurridos durante el desarrollo de software (a estos sistemas también se los conoce como bugtrackers), pero su utilización no debiera por qué limitarse a solo ese ámbito, sino que también debiera ser posible poder cargar cualquier tipo de incidente, en cualquier organización, sin que surga ningún tipo de problemas [17]. Con el fin de facilitar la comparación de los resultados obtenidos, decidimos desarrollar y describir un **SGI** cargado con incidentes de software.

La facilidad de uso de estas herramientas varía mucho, dependiendo de la herramienta en sí. Generalmente, todos los **SGI** tienen los mismos procesos, son muy parecidos a la hora de tratar los incidentes, por lo que es bastante sencillo e intuitivo usarlos una vez que los entendimos. Dentro de los casos de usos, el proceso más común y frecuente es la carga de un nuevo incidente en el sistema. Cuando un usuario encuentra un incidente en el servicio u objeto observado, lo carga en el sistema, completando toda la información que considere relevante para el tratamiento y resolución del mismo. Debe asignarse un usuario responsable para su resolución, que puede ir cambiando a lo largo del ciclo de vida del incidente. Una vez resuelto, se marca como tal en el sistema [17], y queda archivado o se elimina, según la implementación del **SGI**.

Los sistemas de seguimiento de incidentes son comúnmente utilizados para dar soporte a los clientes de una organización o producto dado, sobre la creación, actualización y resolución de los incidentes reportados por los clientes, o bien sobre incidentes que eventualmente son reportados por los empleados de la organización y/o empleados que están abocados al desarrollo del producto. Un sistema de seguimiento de incidentes permite mantener

registros de todas las tareas, requerimientos y mejoras que rodean a un proyecto. Además si el problema ya ha sido resuelto, avisa a los usuarios de que la solución fue encontrada [17]. Esta última característica no siempre resulta sencilla de implantar en estos sistemas, dado que no poseen la semántica necesaria para darse cuenta, como lo hace el humano.

Otros detalles de los incidentes como la fecha de incorporación, los estados por los que pasa, las descripciones o informes detallados sobre los que ha pasado y demás información relevante, deben mantenerse accesible. Es por eso que los sistemas de gestión de incidentes mantienen, para cada incidente, un historial de cambio.

Las herramientas como **Jira**[2] o **Mantis**[3], no presentan la flexibilidad para adaptarse a los cambios, debido a que se modelaron de manera rígida y exclusiva para un problema o artefacto en particular (incidentes del software) frente a un problema más genérico y frecuente en esta realidad que sufre un cambio continuo, donde por ejemplo las empresas tienen que adaptarse indefectiblemente para mantenerse competentes.

Si bien la situación actual, como se explica arriba, no es muy flexible, lo sería ante la presencia de tecnologías que pueden llegar a lograrla. Estas tecnologías podrían dar sustento a modelos flexibles, ante futuros cambios. Concretamente, la aparición de la Web Semántica ofrece la posibilidad de convertir esta hipótesis en verídica por completo.

2.1.2. Actores

Existen muchos **SGI** utilizados en el desarrollo de software (**Jira**, **BloodHound**[4], **Bugzilla**[5], **Mantis**, etc.) pero todos ellos presentan características comunes: manejan un mismo conjunto de datos sobre cada incidente, como se vio anteriormente, y los roles de las personas que interactúan con ellos.

No existen muchos actores dentro del sistema de gestión. Resumidamente, podemos separarlos de la siguiente manera [13]:

Administrador: es un usuario encargado de administrar la carga de incidentes al sistema, como también la personalización de la herramienta (si es posible). Además, se encarga de generar reportes acerca del estado de los incidentes cargados.

[2] `https://es.atlassian.com/software/jira`
[3] `https://www.mantisbt.org/`
[4] Apache BloodHound: http://bloodhound.apache.org/
[5] Bugzilla: http://www.bugzilla.org/

Desarrolladores: son los que implementaron el sistema y realizan modificaciones, en caso de ser necesario. Comunmente no se los incluye como actores del sistema, ya que no influyen una vez que éste está en funcionamiento, pero nosotros decidimos incluirlos en esta lista pues serán uno de los principales beneficiados con la propuesta de este documento.

Reportero: es la persona que realiza la carga del incidente.

2.2. Sistemas analizados

Antes de comenzar con el desarrollo de nuestro propio **SGI**, tuvimos que realizar un análisis profundo sobre las características y funcionalidades de los sistemas más utilizados. Luego de este análisis, obtuvimos una noción más clara de cuales son los puntos fuertes y débiles de estos sistemas.

En la actualidad, hay algunos **SGI** que se destacan por sobre el resto. Estamos hablando de **Jira**, **Mantis**, **Github** y **Bugzilla**. A continuación, vamos a describir brevemente cada uno de estos sistemas, marcando cuáles son sus características más relevantes.

2.2.1. Jira

Jira es una aplicación web utilizada, no solo para el seguimiento de errores e incidentes, sino para la gestión de proyectos. Además, Jira puede ser utilizado para la gestión de procesos y la mejora de estos, gracias a sus funciones para la organización de flujos de trabajo. Es propietaria, desarrollada por Attlasian.

Gracias a que fue desarrollado con tecnologías *Java EE*[6], este sistema soporta múltiples bancos de datos y es multiplataforma. También cuenta con paneles de control adaptables, filtros de búsqueda, estadísticas, función de correo electrónico, entre otros.

Jira es uno de los **SGI** más completos hoy en día. Maneja gran cantidad de información acerca de los incidentes. Algunos de los datos que contiene son: nombre, proyecto, tipo, estado, prioridad, resolución, archivos adjuntos, etc.

2.2.2. Mantis

[13] Mantis, también conocido como **MantisBT (Mantis Bug Tracker)**, es un **SGI** de código abierto simple de usar, pero sin perder potencia.

[6] Es una plataforma de desarrollo de aplicaciones orientadas a ambitos empresariales.

Desarrollado sobre **PHP**[7], soporta **Linux**[8], **Windows**[9] y sistemas operativos **MAC**[10] del lado del servidor. Mantis también incluye filtros, un sistema de búsqueda y función de correo electrónico.

Entre la información que se maneja sobre los incidentes podemos nombrar: un identificador, el nombre del proyecto, categoría, visibilidad, fecha de creación, fecha de la última actualización, informador, usuario asignado, prioridad, severidad, estado, versión del producto, etc.

2.2.3. Github

[7] Github es altamente conocido como una aplicación web para alojar repositorios utilizando el gestor de versiones **Git**[11]. Pero desde el 2009, además permite un seguimiento básico de los incidentes dentro de estos de estos repositorios.

Al no ser un sistema propiamente para el seguimiento de incidentes, no presenta muchas de las funcionalidades que otros **SGI**, como **Mantis** o **Jira**, ofrecen. Sin embargo, presenta una serie de características interesantes de remarcar, además de las usuales (búsqueda, ordenamiento y filtros):

1. Permite tratar a los incidentes de manera sencilla mediante una interfaz desarrollada en **JavaScript**[12].

2. Permite crear etiquetas para categorizar los incidentes y asignarlos a usuarios.

3. Permite a los usuarios votar cuál incidente le gustaría que se resolviese primero.

4. Permite cerrar incidentes cuando se sube código relacionado con él.

En cuanto a la información que se maneja sobre los incidentes, solo tenemos un título y una descripción. Además un incidente puede tener una persona asignada y por supuesto, un estado. Cualquier usuario con acceso al repositorio puede crear comentarios en los incidentes.

[7] http://php.net/
[8] https://www.linux.com/
[9] http://windows.microsoft.com/es-es/windows/home
[10] https://www.apple.com/es/osx/
[11] http://git-scm.com/
[12] https://developer.mozilla.org/en-US/docs/Web/JavaScript

2.2.4. Bugzilla

[4] Por último, investigamos el **SGI** Bugzilla. Es un sistema web para la gestión de incidentes gratuito pero con muchas funcionalidades. Fue originalmente desarrollado y mantenido por **Mozilla**, pero luego se separó y pasó a ser una organización aparte.

El servidor de Bugzilla fue implementado en **Perl**[13] y normalmete utiliza **MySQL**[14] como motor de base de datos[15] (aunque puede utilizar variantes como **PostgreSQL**[16], **Oracle**[17], etc).

Bugzilla también maneja una importante cantidad de datos acerca de cada incidente: alias, estado, resolución, usuario asignado, usuario que reportó, cc (usuarios que no están directamente relacionados con la resolución, pero están interesados en su seguimiento), fecha de la última modificación, comentarios, severidad, etc.

2.2.5. Comparación de los SGI

Luego de describir brevemente cada uno de los sistemas que fueron puestos bajo análisis, podemos compararlos en distintos aspectos y en la Sección 2.3 mencionamos cuales son las características deseables para estos sistemas.

	Jira	Mantis	Github ITS	Bugzilla
Lenguaje	Java	PHP	Perl	Perl
BackEnd	MySQL, PostgreSQL, Oracle, SQL Server	ADOdb (MySQL, PostgreSQL, SQL Server, etc.)	MySQL	MySQL, Oracle, PostgreSQL, SQLite
Fecha Creación	2002	2000	2009	1998
Campos customizables	Si	Si	No	Si
Heterogeneidad	No	No	No	No
Workflow customizable	Si	Si	No	Si

[13] https://www.perl.org/
[14] http://www.mysql.com/
[15] Conjunto de datos interrelacionados.
[16] http://www.postgresql.org.es/
[17] http://www.oracle.com/es/index.html

2.3. Limitaciones observadas

Luego del análisis, realizado por los autores de este documento, sobre las mejoras que podrían realizarse a los **SGI** actuales, se decidió abocarse a los siguientes aspectos que facilitarían la gestión de proyectos y permitirían relacionar los incidentes con su entorno.

Contexto: si cada uno de los incidentes estaría relacionado a la/s parte/s sobre las que incide, los incidentes dependientes y parecidos, etc. se podría optimizar el soporte de aplicaciones evitando incidentes repetidos, viendo que áreas son repercutidas, etc.

Flexibilidad/Heterogeneidad: si se tuviese un modelo de incidente lo suficientemente flexible para que se adapte a los cambios estructurales repentinos, se facilitarían los procesos de reestructuración del modelo de datos logrando una mayor productividad. Sería ideal que en un sistema puedan convivir incidentes con diferencias estructurales. Que se pueda agregar, quitar o modificar tanto la estructura como la información de un incidente sin que el resto sea repercutido.

Un buen diseño se logra con un exhausto análisis de los requerimientos de los usuarios. Pero como desarrolladores, sabemos que los requerimientos de los usuarios cambian con frecuencia. Y cambian una vez que el sistema ya está terminado. Por lo tanto, el sistema debe ser lo suficientemente flexible para aceptar cambios en el modelo subyacente. Hoy en día, estas características no son contempladas.

Estas limitaciones que tienen los gestores de incidentes actuales, son las que intentamos atacar con la utilización de la Web Semántica, según explicaremos en el Capítulo 3.

Capítulo 3

Estrategia general

En este capítulo introducimos el concepto de la web y el de la Web Semántica junto con un conjunto de tecnologías y estándares que esta última utiliza. Luego, explicamos la estrategia general utilizada para el desarrollo del sistema planteado y la comparativa con las metodologías convencionales.

Cuando se comenzó con el desarrollo de la presente tesis, lo primero que se tuvo que decidir fue cómo se iban a defender y demostrar las hipótesis planteadas. Además de presentar las pruebas y los fundamentos de manera escrita, decidimos crear un prototipo de aplicación donde se pueda ver algunas de las ventajas que nuestro trabajo intenta justificar. Además, el uso de un prototipo real, nos permitió encontrar más resultados de los que creíamos en un principio.

3.1. Arquitectura del Sistema

Se decidió desarrollar un **SGI** usando las tecnologías de la Web Semántica para demostrar las ventajas que se alcanzan al utilizar esta tecnología en contraste con los métodos tradicionales. En el Capítulo 2 se presentaron los requerimientos y las falencias de los **SGI** actuales. Ahora que tenemos conocimientos de esas falencias, y de las tecnologías involucradas en la web Semántica, podemos pasar a describir la arquitectura del **SGI** desarrollado.

Una de las características deseables en un **SGI** era su disponibilidad y accesibilidad desde cualquier lugar y en cualquier momento. Por eso, se decidió que el **SGI** iba a tener una interfaz web y debía estar montado en un servidor *HTTP* [38]. Como cualquier aplicación web típica, nuestro **SGI** (desde ahora lo llamaremos **SIT**, Semantic Issue Tracker) cuenta con dos componentes: el cliente, donde se encuentra el usuario final utilizando la

aplicación a través de un navegador, y el servidor, donde residen los datos, reglas y lógica de la aplicación.

3.1.1. Aplicación semántica

Como cualquier aplicación web, el SIT puede ser accedido desde cualquier navegador ya que utiliza las tecnologías y los estándares frecuentemente utilizados en el desarrollo web (*HTTP*, *HTML* [9], *CSS* [9], etc.). A pesar de tener muchas similitudes con las tecnologías usadas en las aplicaciones web que se usan hoy en día, también tiene diferencias importantes las cuales le permiten explotar aun más el potencial de la red. Al utilizar RDF[1], detallado en la Sección 3.2.3, el sistema también puede ser accedido por agentes (humanos o informáticos) a través de otras herramientas (no sólo mediante el navegador). Esta característica hace que el sistema tenga que cambiar el formato de la respuesta según cómo se pidieron los datos. Por ejemplo, si el servidor identifica que el requerimiento HTTP viene de un browser, devolverá una página HTML para poder ser renderizada en el browser y así, ser interpretada por las personas. En cambio, si identifica que el llamante pidió explícitamente que la respuesta sea enviada en RDF (por ejemplo, un agente informático), devolverá los datos en dicho formato. Esto permite el consumo de la información por parte de varias fuentes heterogéneas, las cuáles esperan distintos formatos de respuesta, sin la necesidad de tener lógica adicional en la aplicación. A este proceso se lo denomina negociación del contenido [43].

El trabajo fue desarrollado utilizando el framework Callimachus, explicado en el Capítulo 6, que facilita el desarrollo de aplicaciones sobre la Web Semántica. Este framework acelera el desarrollo, permitiendo al usuario concentrarse únicamente en una lógica sencilla y los datos de la aplicación. Por eso, la arquitectura de software es definida por el mismo framework, aunque permite personalizar muchos de los componentes que se utilizan.

A continuación vamos a describir la arquitectura de hardware y software que utiliza la Web Semántica, y vamos a describir algunos de los beneficios que se obtienen al usar estos modelos.

3.1.2. Componentes del sistema

Los sistemas web utilizan comúnmente la arquitectura de tres capas Cliente-Servidor-Base de datos [39]. La responsabilidad de lo que tiene que hacer cada una de las partes fue variando mucho y no hay un estándar

[1]Resource Description Framework.

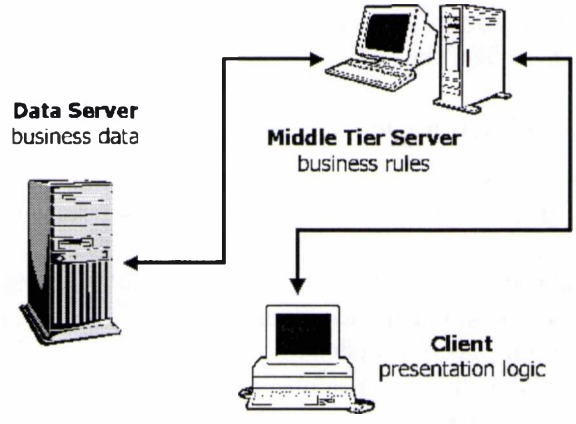


Figura 3.1: Arquitectura de tres capas clásica [42]

preferido por los desarrolladores. En un primer enfoque podemos decir que esta arquitectura separa físicamente la presentación, el procesamiento y el almacenaje de los datos. La separación en capas tiene una serie de ventajas además de las ventajas que conlleva la modularización. Por ejemplo, el cambio de tecnologías en una de las capas, solo afectará al código de dicha capa, causando un desacoplamiento importante entre las partes [14].

A continuación se define cada una de las capas [42][14]:

Cliente: presenta la interfaz de usuario. La función principal del cliente es la de transformar y mostrar los resultados que el servidor le facilita de manera que puedan ser interpretados por el usuario. El cliente más común es el navegador web, el cual corre generalmente en una PC y utiliza una interfaz de usuario gráfica. Eventualmente, en esta capa, puede aparecer cierta lógica de la aplicación.

Servidor: esta capa coordina la aplicación, procesa los datos y contiene la lógica de negocio de todo el sistema para servir al cliente. Generalmente se levanta en servidores o workstations especiales.

Datos: en la capa de datos se almacena la información en una base de datos o sistema de archivos. Esta capa suele estar en un servidor especializado en datos (Database Server).

La interacción y el flujo de mensajes entre las capas es siempre lineal y se lo denomina *Request-Response Pattern*[16]. La secuencia es la siguiente:

1. El Cliente realiza un pedido al Servidor
2. El Servidor (que se encuentra siempre escuchando por pedidos de los clientes) recibe el pedido y lo procesa.
3. Si el pedido así lo requiere, el Servidor puede pedir/enviar datos al servidor de datos.
4. El Servidor responde el pedido al Cliente

El SGI no necesitó de una arquitectura hardware distinta en cuanto a los componentes, aunque sí necesitó de algunos cambios en las tecnologías y protocolos que se usan dentro de cada componente con respecto a los SGI tradicionales.

El único componente físico que debe cambiar, por un tema de rendimiento, es la base de datos. La mayoría de las bases de datos utilizadas por los sistemas web son relacionales (aunque últimamente, con el auge del formato JSON[2] como formato preferido para envío de datos en la web, fueron tomando más fuerza las bases de datos NoSQL[3] como MongoDB[4]). Por la estructura del tipo de formato que maneja (tripletas) la Web Semántica, las bases de datos de este tipo, no son la mejor opción. Por eso aparecieron los almacenes de tripletas (triplestores), explicadas posteriormente en este documento.

3.1.3. Arquitectura de software

En cuanto a la disponibilidad de los componentes de software, los protocolos y lenguajes usados, el diseño cambió comparado al clásico.

Estándares y lenguajes

Los lenguajes y protocolos utilizados en el desarrollo fueron los estándares de la Web Semántica. Es decir, se usaron *URIs* [50] para la identificación de los recursos de manera no ambigua; estas URIs, a su vez, utilizaban el protocolo *HTTP*, popular en la web.

Una vez definidos los identificadores de los recursos dentro de la web, lo siguiente que se necesitó fue un lenguaje para poder intercambiar des-

[2] JavaScript Object Notation. `http://json.org/`

[3] `http://www.mongodb.com/nosql-explained`

[4] MongoDB (Sitio oficial: `www.mongodb.org/`): base de datos open source y basada en documentos.

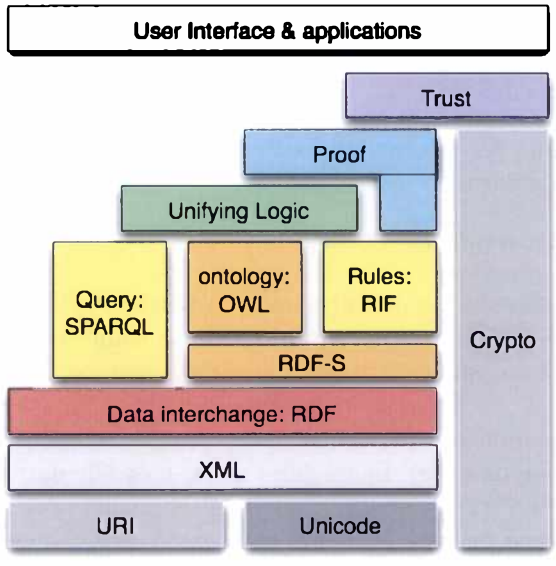


Figura 3.2: Arquitectura de capas de la Web Semántica [53]

cripciones de recursos. Respetamos el estándar de la **W3C**[5], utilizando el formato *XML* [1] para codificar los recursos en documentos.

Dado que los datos en la Web Semántica están distribuidos, la descripción de recursos debe estar codificada de manera tal que facilite la integración desde varias fuentes. El formato más conveniente es el de estructuras de grafo. La **W3C** propuso *RDF* como estándar para esta codificación. Existen varias notaciones que se pueden utilizar para escribir RDF, aunque el estándar es *XML*.

Teniendo los datos integrados en grafos, utilizamos *SPARQL* (el servidor consta con un endpoint SPARQL, explicado en la Sección 3.2.9, al que se le puede mandar consultas y este responde en formato RDF) para consular la información de los distintos grafos.

3.1.4. Flexibilidad e Interoperabilidad

El uso de estas tecnologías y lenguajes en una aplicación donde no se usa la Web Semántica con la intención de publicar datos, era una incógni-

[5]El World Wide Web Consortium es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la web a largo plazo.

ta al empezar la tesis. Luego de investigar las tecnologías y los conceptos relacionados, creímos que sería posible agregar expresividad a las aplicaciones web, si usasemos dichas herramientas. El resultado fue el esperado, y además, encontramos otros beneficios en el camino, que facilitan enormemente el desarrollo a los programadores.

Una de las primeras ventajas que encontramos fue la posibilidad de trabajar sobre las estructuras de datos, la base de datos y sobre las interfaces de usuario sin la necesidad de tener que bajar el servidor, o reiniciarlo luego de realizados los cambios. Para los desarrolladores que no están acostumbrados al manejo de infraestructura y de ambientes de desarrollo, esta cualidad presenta un beneficio grande que permite ahorrar mucho tiempo.

Esta flexibilidad se debe en su totalidad a los lenguajes que se utilizan en el desarrollo. RDF, y las bases de datos triplestore, no tienen una estructura rígida de datos para cada clase. Posteriormente, en este documento se explicarán estos conceptos más en detalle, pero por ahora basta con saber que las entidades son un conjunto de tripletas que contienen la información y las relaciones de los recursos. Esta flexibilidad permite agregar, editar y borrar información y relaciones de las instancias sin que el sistema sufra las inconsistencias. Además, esta característica de RDF permite que dos instancias de una misma *clase RDF* tengan estructuras distintas, de acuerdo al momento en que fueron creadas [22].

Otra ventaja importante de la aplicación, es la facilidad para integrar datos de otras aplicaciones. Esta es otra de las principales características de la Web Semántica. En nuestro caso, obtenemos la información de las personas dentro de la organización desde una aplicación separada (que por motivos de facilidad de desarrollo, también fue desarrollada usando Callimachus). La aplicación tiene dos formas de publicar datos, a través del endpoint SPARQL, o a través de los archivos RDF, en sus diferentes formatos de serialización, que se obtienen al consultar una URI de un recurso en particular.

3.2. Tecnologías utilizadas

Ahora que el lector tiene una idea general de lo que se construyó, debemos citar algunas definiciones de las tecnologías involucradas en el proceso de desarrollo para facilitar el entendimiento de los capítulos venideros.

3.2.1. La Web Semántica

La Web es un sistema distribuido de documentos de hipertexto accesibles a través de Internet, que permite compartir e intercambiar información entre diferentes sistemas. Sin embargo, estos documentos están escritos para poder ser interpretados principalmente por el humano. Iniciativas emergentes como la Web Semántica o LOD[6], explicado en la Sección 3.2.2, tratan de cambiar la forma en que se relacionan los datos en la Web y darles semántica para que estos puedan ser interpretados por agentes informáticos. Estas iniciativas se centran en la publicación, intercambio, recolección y consulta de los contenidos existentes en los sistemas web actuales.

Podemos definir la Web Semántica como una colección de estándares y tecnologías que permiten a las computadoras entender el significado de la información que tienen los documentos en la Web actual [55]. Junto con el crecimiento de la Web Semántica, aparecieron un conjunto de estándares, tecnologías y herramientas necesarias, relacionadas con el concepto de Web Semántica. En un primer intento de relacionarlas, podemos decir que la Web Semántica combina un mecanismo de nombramiento y direccionamiento distribuido (URIs con las que se identifican a los recursos), basada en la web actual, con una representación formal del conocimiento (*RDF*, *RDFS* y *OWL*), un mecanismo para mostrar dialectos en RDF (*GRDDL*) y un lenguaje de consulta común (*SPARQL*). En el desarrollo de este capítulo explicaremos bien en detalle estas tecnologías y su relación.

Para entender el concepto de Web Semántica, definiremos antes algunos conceptos básicos necesarios para su comprensión final.

Un recurso en la web es una entidad que puede ser identificada de alguna manera (actualmente con una URI) [50]. Por lo tanto, un documento, una imagen son todos recursos web, pero también lo son conceptos abstractos como la representación de una persona en una aplicación web.

Definiremos la integración de datos en la web como el proceso de combinar y agregar recursos en la red de tal forma que faciliten la comprensión de la información para los usuarios. En la web tradicional, la integración de datos corre por parte del usuario. Es decir, la computadora no tiene forma de saber la relación que tienen dos recursos disponibles en la web. La web es una gran red de recursos conectados, pero la interpretación de las relaciones y el significado es tarea de las personas que la utilizan. Una computadora sólo ve un montón de recursos heterogéneos interconectados. Queda claro que para lograr que una computadora tenga el poder de interpretar la información en un sitio web, no alcanza con las tecnologías existentes. Un

[6]Linked Open Data.

enfoque posible para resolver este problema es agregar contenido semántico (que no modifica en absoluto la vista final de las páginas) a los documentos de la web. En particular, deberíamos realizar los siguientes cambios en cada sitio:

1. Debe existir una forma de representar la información en la Web. Y esta representación tiene que poder ser interpretada por una computadora.
2. Esta representación debe seguir un estándar que todos los sitios deben respetar.
3. Debe existir una forma de crear estas representaciones y agregarlas a los sitios.
4. Deben existir términos y relaciones compartidos entre la información utilizada en las distintas páginas relacionadas.
5. Debe existir una forma de crear estos términos y relaciones.

La Web Semántica aparece con la intención de resolver todos estos puntos. Formalmente, la Web Semántica es un conjunto de tecnologías y estándares que que permiten a las máquinas entender el significado (*semántica*) de la información disponible en la web, permitiendo así a las computadoras realizar tareas automáticas de gran escala, que de otra manera tendrían que hacerse manualmente por un usuario.

Pero la idea de agregar contenido semántico a la Web, trae varios conceptos relacionados. A continuación vamos a ver algunos de ellos.

3.2.2. Linked Data

La idea de Linked Data fue propuesta por **Tim Berners-Lee** [51]. Es un movimiento respaldado por la **W3C** para conectar bancos de datos distribuidos en la Web. Al agregar contenido semántico en una página, un recurso dado puede estar conectado con un recurso en otro lado de la Web; y este a su vez, con otro recurso en otro servidor. Así se van formando relaciones semánticas entre los distintos recursos a lo largo de toda la Web. La publicación de datos estructurados en la red y la interconección entre esos datos en los distintos banco de datos son los conceptos básicos que permiten la aparición de Linked Data.

Técnicamente hablando, Linked Data se refiere a la publicación de datos en la Web de modo que puedan ser interpretados por una computadora, su significado esté explícitamente definido, esté interconectado con un recurso

en un banco de datos externo y pueda ser usado en conexiones en objetos externos.

La definición es muy similar a la de la Web Semántica. De hecho muchas veces se pueden ser intercambiados [55]. Para clarificar la relación entre la Web Semántica y Linked Data podemos decir que:

1. Los datos en Linked Data son publicados utilizando tecnologías y estándares de la Web Semántica.

2. Las relaciones entre los datos en Linked Data se establecen utilizando tecnologías y estándares de la Web Semántica.

3. La relación entre los dos conceptos se puede ver como: la Web Semántica es el objetivo a conseguir, y Linked Data provee lo necesario para lograrlo.

3.2.3. RDF: Resource Description Framework

Con el propósito de construir una Web Semántica, es necesario contar con un lenguaje y un estándar para representar los datos. Este lenguaje debe ser lo suficientemente flexible para poder expresar cualquier información que se quiera representar, como así también contar con un mecanismo para conectar información distribuida.

RDF fue propuesto en 1999 como modelo básico para la creación y procesamiento de *meta-datos*[7]. Define un mecanismo para describir recursos pero no asume nada con respecto al dominio de la aplicación (es decir, es independiente del dominio), y por lo tanto puede ser usado para describir información de cualquier dominio. Entonces, una definición de RDF puede ser la siguiente: RDF es un estándar publicado por **W3C**, que puede ser usado para representar información distribuida de tal forma que una aplicación pueda usarla y procesarla automáticamente [55].

En la Figura 3.3 se puede ver un ejemplo de una persona modelada utilizando RDF. En este diagrama se pueden distinguir cinco recursos identificados con una URI (los óvalos de color azul). Los óvalos colorados representan valores literales (texto, números) de las relaciones que tienen los recursos. Así, se puede entender que el recurso que representa a **Berners-Lee**, tiene una propiedad *name* con valor *"Tim Berners Lee"*.

Un aspecto fundamental de RDF es su modelo abstracto. Este modelo es el que utiliza para descomponer cualquier información en pequeñas piezas

[7]Datos que describen datos.

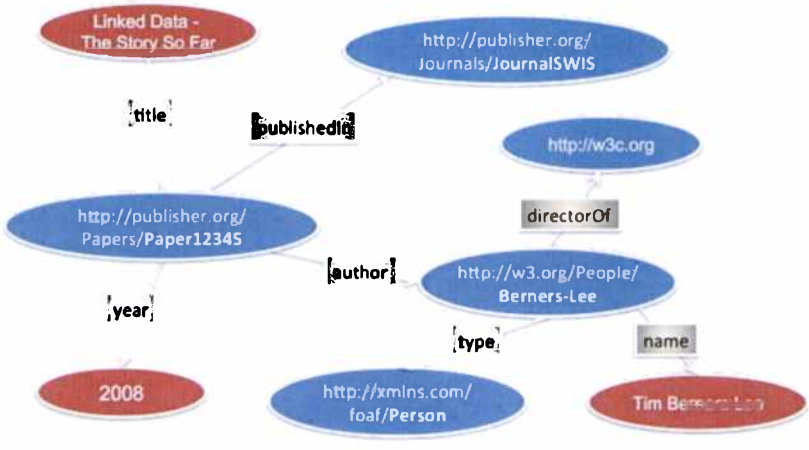


Figura 3.3: Modelo de ejemplo RDF

llamadas sentencias. Toda información puede ser descompuesta en una lista de sentencias. Cada sentencia tiene la forma sujeto-predicado-objeto, y este orden nunca cambia. El sujeto y el objeto son nombres de dos cosas en el mundo, y el predicado es la relación entre dichas cosas [55].

Como una sentencia siempre consta de los mismos tres componentes, se llama también tripleta. Cada sentencia o tripleta representa un hecho; una colección de sentencias representan una porción de información y se conoce como grafo RDF.

El sujeto en una sentencia denota un recurso de la web identificado con una *URI*. El objeto, a diferencia del sujeto, puede denotar tanto un recurso como el valor de un recurso expresado con un literal. El predicado denota la relación entre los dos recursos y también es identificado por una *URI*. Cuando un recurso no es identificado con una *URI*, se lo conoce como *nodo blanco*. Un nodo blanco puede estar tanto en el sujeto como en el objeto, pero nunca en el predicado.

Sujeto	Predicado	Objeto
URI o nodo blanco	URI	URI, literal o nodo blanco

Cuadro 3.1: Modelo abstracto de RDF

En cuanto al formato de serialización de los documentos RDF, la reco-

mendación de **W3C** es *XML*, aunque han ido apareciendo varios formatos alternativos que son más fáciles de escribir para los programadores y más fácil de entender. Algunas de esas alternativas son *Notation-3*, *Turtle* y *N-triples* [55].

3.2.4. RDFa

RDFa es una especificación de atributos para expresar datos estructurados en cualquier lenguaje de marcado [55]. Es decir, forma una capa angosta de marcado que se puede agregar a las páginas web para que las máquinas puedan interpretar la información en ellas, tal como las personas.

Para simplificar el entendimiento del concepto, podemos decir que es otra forma de agregar datos con contenido semántico dentro de documentos *XHTML*. Pero a diferencia de los microformatos, este provee un conjunto de atributos que pueden ser usados para el marcado de datos. Otra diferencia con los microformatos es que RDFa si es un estándar de la **W3C** [35].

Para explicar de forma simple la relación entre RDF y RDFa, veamos a RDFa como la forma de expresar tripletas RDF dentro de un documento XHTML. De esta forma los servicios web y los motores de búsqueda pueden utilizar ese nuevo marcado para realizar mejores búsquedas y ofrecerle al usuario un resultado más rico en contenido.

Los atributos que se utilizan para embeber la inforamción son:

typeof: indica de qué tipo es la instancia descripta.

about: es la URI del recurso al cual se está describiendo. Remite al documento actual por defecto.

rel, rev, href y resource: establecen una relación con otro recurso.

property: aporta una propiedad para el contenido de un elemento.

content: atributo opcional que se sobrepone al contenido del elemento. cuando se usa el atributo property.

datatype: atributo opcional que indica el tipo de datos del contenido.

Un ejemplo claro del uso de RDFa es el siguiente:

Listing 3.1: Ejemplo de RDFa

```
1  <div vocab="http://schema.org/" typeof="Person">
2    <a property="image"
       href="http://manu.sporny.org/images/manu.png">
3      <span property="name">Manu Sporny</span>
4    </a>,
5    <span property="jobTitle">Founder/CEO</span>
6    <div>
7      Phone: <span property="telephone">(540)
         961-4469</span>
8    </div>
9    <div>
10     E-mail: <a property="email"
         href="mailto:msporny@digitalbazaar.com">
         msporny@digitalbazaar.com</a>
11   </div>
12   <div>
13     Links: <a property="url"
         href="http://manu.sporny.org/">Manu's
         homepage</a>
14   </div>
15 </div>
```

A partir del código podemos ver:

1. en la línea 1, que se va a describir una instancia de *Person*, que está difinido en el vocabulario `http://schema.org/`.

2. en la línea 2, se agrega una propiedad *image* a la instancia que se está describiendo.

3. en la línea 3, se agrega una propiedad *name* a la instancia que se está describiendo con valor *Manu Sporny*.

4. en la línea 5, se agrega una propiedad *jobTitle* a la instancia que se está describiendo con valor *Founder/CEO*.

5. en la línea 7, se agrega una propiedad *telephone* a la instancia que se está describiendo.

6. en la línea 10, se agrega una propiedad *email* a la instancia que se está describiendo.

7. en la línea 7, se agrega una propiedad *url* a la instancia que se está describiendo.

3.2.5. RDFS

RDFS (*RDF-Schema*) es un lenguaje que puede ser utilizado para crear vocabularios, para que cuando se creen documentos RDF para un dominio determinado, se puedan utilizar los términos definidos en el mismo.

Es decir, RDFS es un lenguaje descriptivo de vocabularios para el estandár RDF especificado a través de la **W3C** [44]. Presenta algunos constructores para describir clases, propiedades, etc dentro de un dominio. Cabe mencionar que los constructores son a su vez clases y propiedades; es decir, RDFS provee clases y propiedades por defecto que pueden ser usadas para describir clases y propiedades de un dominio en particular.

Una característica importante de RDFS, es que facilita la inferencia en el documento que usa el vocabulario.

3.2.6. Inferencia

Llanamente hablando, la inferencia en la Web Semántica trata de descubrir nuevas tripletas implícitas, a partir de las definidas explícitamente. Formalmente, es el proceso de deducción de nuevas aceptaciones basandose en reglas ya conocidas [52]. Más adelante explicaremos la definición de ontología, pero por ahora basta con saber que definen el vocabulario de un determinado dominio. Al crear ontologías, el usuario puede definir reglas de inferencia para que una computadora, a través de un motor de inferencia, pueda generar las nuevas tripletas. Un ejemplo es el de chequeo de subsunción, en el cual se puede determinar si un recurso es *sub-clase de* o *super-clase de* a través de relaciones definidas por OWL como *Subclass Of*, etc. De esta manera se podria obtener una clasificación de jerarquía automática [31].

Veamos un ejemplo concreto. Supongamos que tenemos una sentencia que dice *"Laika es-un Perro"*. Además, una ontología dice que *"Perro es-a-su-vez Mamífero"*. Esto significa que una computadora capaz de interpretar la notación *"es-a-su-vez"* puede inferir que Laika es un mamífero, sin que esta sentencia esté expresada explícitamente.

Otro ejemplo se puede ver con las relaciones bidireccionales. Si se define una relación bidireccional *"es-hermano-de"* y la tripleta *"Bart es-hermano-de Lisa"*, se infiere que *"Lisa es-hermano-de Bart"*.

La inferencia es una de las cualidades más atractivas y poderosas de la Web Semántica, que sirve para mejorar la calidad en la integración de los datos de la web. También se utiliza esta técnica para encontrar posibles inconsistencias en los datos. Además, puede ahorrar mucho código que de otra manera el usuario tendría que escribir. Si bien esta característica es muy poderosa, necesitamos de la especificación de relaciones semánticamente ponderadas (*sameAs*, etc.) provistas por lenguajes diseñados para soportar inferencia, como lo es el caso de *OWL* [48].

3.2.7. OWL

El propósito de OWL (*Web Ontology Language*) es el mismo que **RDFS**: definir una ontología que incluya clases, propiedades y sus relaciones para un dominio en particular. Cuando alguien necesita describir un recurso, utiliza estos términos en documentos RDF para permitir la interpretación de la información por máquinas. Sin embargo, en comparación con RDFS, OWL provee la capacidad de expresar relaciones mucho más ricas y complejas [55]. Al momento de la escritura de la tesis, la versión más reciente del lenguaje era *OWL2*[8].

OWL introduce a su vez algunos otros conceptos:

Un *axioma* es una sentencia básica en OWL. Representa una pequeña porción de conocimiento. Dentro de un axioma, los componentes que aparecen (*sujeto*, *objeto*, *relación*) se conocen como *entidades*. Una de las propiedades de OWL es la posibilidad de combinar entidades para crear nuevas. Estas combinaciones son conocidas como *expresiones*.

3.2.8. TripleStore

Un triplestore es una base de datos especialmente diseñada para almacenar y devolver tripletas (*sujeto - predicado - objeto*) [55]. A diferencia de las bases de datos relacionales, el lenguaje de consulta utilizado es SPARQL.

En algunos casos, cada sentencia (*tripleta*) agrega un cuarto miembro para identificar el contexto, o el grafo RDF al cuál pertenece. Así, en vez de tripletas, las sentencias tienen cuatro miembros y la base de datos donde se almacenan se conoce como *quadstore*.

[8]OWL2 Overview: `http://www.w3.org/TR/2012/REC-owl2-overview-20121211/`

3.2.9. SPARQL

Ante la aparición de la Web Semántica y la introducción de datos semánticos en formato RDF, apareció la necesidad de una herramienta que pudiese buscar información sobre dicha web, ya que las herramientas que existían al momento no alcanzaban. En concreto, se debía construir un nuevo tipo de motor de búsqueda que trabajase eficientemente con este formato y que pudiese devolver resultados más precisos y completos.

A ese nuevo motor se lo denominó SPARQL (el nombre es un acrónimo recursivo, ***SPARQL Protocol And RDF Query Language***). Por cuestiones de accesibilidad y eficiencia, estos motores suelen residir en servidores accesible via HTTP. Formalmente, SPARQL 1.1[9] es un conjunto de especificaciones que definien lenguajes y protocolos para consultar y manipular el contenido de los grafos RDF en la Web o en un triplestore [55]. El estándar incluye las siguientes especificaciones [54]:

SPARQL 1.1 Lenguaje de consulta: un lenguaje de consulta para RDF.

SPARQL 1.1 Formatos JSON, CSV y TSV para resultados de consultas: además del formato estándar para los resultados (XML), SPARQL 1.1 permite tres formatos populares y alternativos: JSON, CSV y TSV.

SPARQL 1.1 Consulta Distribuida: una especificación que define una extensión del SPARQL Lenguaje de consulta para realizar consultas sobre endpoints SPARQL distribuidos.

SPARQL 1.1 Entailment Regimes: define la semántica para las consultas sobre RDFS, OWL o RIF[10].

SPARQL 1.1 Lenguaje de actualización: un leguaje de actualización para grafos RDF.

SPARQL 1.1 Protocolo para RDF: un protocolo que define la manera de envíar consultas SPARQL o actualizaciones contra un endopoint SPARQL.

SPARQL 1.1 Descripción de Servicio: una especificación que define un método para descubrir servicios como también el vocabulario para describirlos.

[9] SPARQL 1.1: `http://www.w3.org/TR/rdf-sparql-query/`

[10] `http://www.sindikos.com/2011/08/capas-de-la-web-semantica/`

SPARQL 1.1 Protocolo HTTP en Grafos: a diferencia del protocolo SPARQL entero, esta especificación define como manejar el contenido de grafos RDF directamente via operaciones HTTP.

SPARQL 1.1 Casos de Uso: un conjunto de tests.

Ejemplo de consulta:

Listing 3.2: Ejemplo de SPARQL

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
        ?person foaf:name ?name .
        ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

En el ejemplo anterior se obtiene el nombre y la cantidad de amigos de todas las personas que tengan nombre y amigos.

3.2.10. Software para la Web Semántica

Para que las organizaciones puedan beneficiarse por completo del enfoque que la Web Semántica impone, se necesita de nuevas herramientas de software. *Bases de datos*, *middleware*[11], y demás aplicaciones implicadas en un sistema de información, deben ser mejoradas de manera tal que sean compatibles con las nuevas tecnologías y estándares como *RDF*, *OWL*, *SPARQL*, etc.

Triplestores

La mayoria de los triplestores son de código abierto. Los proveedores de tecnologías adoptaron enfoques ligeramente variados en lo que respecta a la arquitectura. Por ejemplo, **3store**[12] emplea una representación en memoria, mientras que **Sesame**[13] y **Kowari**[14] usan una solución basada en almacenamiento físico secundario. Recientemente un número de triplestores comerciales han dado a luz, como **RDF Gateway de Intellidimension**[15],

[11] Es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos.

[12] `http://threestore.sourceforge.net/`

[13] `http://sourceforge.net/projects/sesame/`

[14] `http://kowari.org`

[15] `http://www.intellidimension.com/`

Virtuoso de OpenLink[16], **Profium Metadata de Profium Server**[17] y **Oracle Database de Oracle**[18]. Una característica interesante de **Oracle** y **OpenLink** es que permiten almacenar RDF/OWL en triplestores junto con datos XML y relacionales. Este tipo de solución híbrida permite que una única consulta abarque diferentes representaciones de datos [49].

Frameworks

Algunas organizaciones han desarrollado un conjunto de herramientas para construir aplicaciones, o más conocidas como frameworks, para la Web Semántica. Entre los más conocidos están Callimachus, el cual provee un marco de trabajo para crear aplicaciones basadas en manejos de datos sobre principios concernientes a *Linked Data*. Además permite a los desarrolladores crear fácil y rápidamente aplicaciones basadas en la Web Semántica. Es decir que con Callimachus se puede almacenar datos *RDF* y proveer de un sistema de plantillas para la visualización de los mismos. Las plantillas son escritas en estándares ya conocidos como *XHTML* y *RDFa*. También se provee de un servidor *SPARQL* y una *API*[19] REST[20]. Otro framework conocido es **Jena** framework para Java[21]. Este provee un entorno de programación para leer, escribir, y consultar RDF y OWL. Además dispone de un motor de inferencia basado en reglas, como también mecanismos estándares para usar otros motores de razonamiento, tales como **Pellet**[22] o **FaCT++**[23]. Otro framework conocido es **Haystack Semantic Web Browser** que esta basado en la plataforma **Eclipse**[24] [49].

[16] `http://virtuoso.openlinksw.com`

[17] `http://www.profium.com/index.php?id=485`

[18] `http://www.oracle.com/technology/tech/semantic_technologies`

[19] Interfaz de Programación de Aplicaciones.

[20] Transferencia de Estado Representacional: `http://www.restapitutorial.com/lessons/whatisrest.html`.

[21] `http://jena.sourceforge.net/`

[22] `http://pellet.owldl.com/`

[23] `http://owl.man.ac.uk/factplusplus/`

[24] `http://haystack.csail.mit.edu/staging/eclipse-download.html`

Otros

Si una organización ha podido consistentemente asignar *URIs* a través de sus fuentes de datos, luego la integración de datos es simple con RDF. Sin embargo, si diferentes departamentos de la organización inicialmente asigna diferentes *URIs* para una misma entidad, luego las construcciones *OWL* pueden ser usadas para relacionar las *URIs* equivalentes. La mayoría de los vendedores proveen un *middleware* asignado para integrar datos basados en ontologías. Ejemplos de estos pueden ser, **OntoBroker**[25], **TopBraid Suit**[26], **Protege**[27], y **Cogito Data Integration Broker**[28] [49].

3.3. Comparación con los Métodos Tradicionales

Conociendo ahora la arquitectura del sistema propuesto (explicado en la sección 3.1) y las tecnologías usadas (sección 3.2) tenemos una idea de las tecnologías involucradas en la Web Semántica y cómo se estructuran y relacionan las partes dentro de un sistema con datos modelados con RDF y almacenados en una triplestore. Ahora que conocemos las características del modelo semántico, realizaremos una comparación con los métodos tradicionales, en particular con aquel que modela con objetos y almacena en base de datos relacionales.

Las ideas básicas en las que se basan ambos modelos son similares. Los objetos (recursos) son descritos por propiedades y relaciones con otros objetos formando grafos. Las diferencias no se ven en la información que puede modelarse, sino en el cómo se modela esta información y las flexibilidades que tienen los esquemas ante los cambios. Veamos algunas de las diferencias entre ambos enfoques:

3.3.1. Diferencias

1. La primer diferencia es en lo concerniente al modelo de los datos. Una propiedad en el modelo RDF es un *ciudadano de primer clase*, mientras que en el modelo OO (orientado a objetos) se define en el contexto de una clase. Es decir, en RDF es posible agregar una propiedad solo a un recurso dentro de una clase, sin definirla en el resto de los recursos de la misma clase [22].

[25] `http://www.ontoprise.de/content/e1171/e1231/`
[26] `http://www.topquadrant.com/tq_topbraid_suite.htm`
[27] `http://protege.stanford.edu/`
[28] `http://www.cogitoinc.com/databroker.html`

2. Otra diferencia entre los modelos es que el modelo RDF no tiene el concepto de método para la comunicación entre instancias y además, todos los recursos del grafo son públicos, al contrario del modelo clásico donde se pueden definir miembros privados.

3. Para integrar la información proveniente de múltiples bancos de datos usando el esquema semántico, no se necesita conocer de antemano la estructura del modelo o el formato en el que se presenta (ya que se tiene un lenguaje en común, RDF) [23]. En el modelo OO el panorama puede no presentarse tan a favor, al no existir estándares en los lenguajes usados ni en las estructuras de los modelos.

4. Una de las ideas principales de la Web Semántica es el uso de URIs para identificar a los recursos [55][8][48][15]. Además, se pueden agregar reglas de inferencia que permitan distinguir cuando dos recursos son el mismo (por ej. si dos recursos tienen el mismo email personal). De esta manera, los motores de inferencia[29] resuelven las posibles ambigüedades que existen entre la información de distintos bancos de datos. Existen lenguajes que proveen propiedades para indicar relaciones semánticas ponderadas, como *sameAs* de OWL tratado en una sección posterior. Por su parte, en el modelo clásico se debe agregar lógica en el motor de base de datos o en el sistema mismo (o al menos utilizar una utilidad que se encargue de ello) para identificar objetos repetidos.

5. El lenguaje de consulta estándar usado en la Web Semántica es SPARQL, mientras que el más popular en el modelo clásico es SQL[30]. Ambos lenguajes permiten consultar, agregar, eliminar y actualizar datos de las bases de datos [55].

6. En el modelo clásico, un cambio en un objeto significa tener que cambiar la tabla en la base de datos donde se almacenan y esto se debe hacer con el servidor parado. Así, si originalmente tenemos una clase Persona con un único atributo *nombre* y una tabla en la base de datos con una única columna *nombre*, si agregamos un nuevo atributo *apellido* en la clase Persona, en la tabla de la base de datos también debemos que cambiar la estructura. Y si además este nuevo atributo no permite valores nulos, debemos setear un valor en todas las instancias de la clase Persona. En el modelo semántico esto no sucede.

[29] También denominados razonadores.

[30] `http://databases.about.com/od/sql/a/What-Is-Sql.htm`

Distintas instancias de la misma clase pueden utilizar los atributos de la clase que ellos quieran. No existen los atributos obligatorios a nivel de base de datos [23][22].

7. La Web Semántica presenta dos formas de acceder a los datos de un repositorio remoto: consultas a un endpoint SPARQL o mediante una consulta HTTP a una URL con los datos RDF, especificando que se quiere los resultados en dicho formato. En el modelo clásico, se puede crear un endpoint SQL para compartir datos a través de HTTP.

8. Mientras que SPARQL tiene una única estandarización (realizada por la W3C) [55][27], SQL tiene muchas variantes (**ANSI**, **ISO**, etc.) [36][21].

9. SPARQL te permite, a través de los denominados *quadstore*, llevar el rastro de donde proviene cada porción de la información, y te permite almacenar metadatos sobre el origen, permitiendo filtrar los resultados según su procedencia [20]. En SQL, se debería crear un campo especial en todas las tablas de todas las bases de datos para conseguir el mismo esquema, algo que no es escalable ni eficiente.

10. La Web Semántica todavía no está popularizada. Mientrás que **SQL** aparecío en el año 1984 [21], SPARQL fue estandarizado en el año 2008 [55]. Esto refleja una amplia ventaja de **SQL** en cuanto a maduración.

11. SQL y SPARQL tienen un conjunto de funciones de agregación analogas usando el operador *GROUP BY* [34].

12. Tanto SQL como SPARQL soportan la operacion *join*. SQL prevee dicha funcionalidad a través de el operador *JOIN* y sus derivados, mientras que SPARQL la provee mediante el operador *OPTIONAL* [34].

Capítulo 4

Trabajo Relacionado

En este capítulo describiremos brevemente trabajos del ámbito científicos que se relacionan directa o indirectamente con nuestra propuesta.

4.1. Web Semántica Corporativa

El objetivo de la *Web Semántica Corporativa* es el de establecer el uso de las tecnologías semánticas y metodologías de administración de la información en las empresas. Enfocándose en los ambientes controlados que las organizaciones presentan, se evitan los conocidos problemas de escalabilidad, privacidad y confianza que la Web Semántica global tiene que enfrentar. Considerando un ambiente organizacional, estos problemas no suelen presentarse. Mientras que la Web Semántica global se centra en tecnologías, la **CSW**[1] explota el uso de esas tecnologías semánticas dentro de las organizaciones [6]. No solo embarca la realización de aplicaciones semánticas sino también los aspectos económicos que conlleva [49]. Además direcciona tanto el lado interno como el lado externo de la organización, los cuales pueden ser humanos o servicios automatizados [6].

El primer equipo de investigación que se dedicó al estudio de la **CSW** fue **ACACIA** en 2002. Desde 2008, se creó un equipo de investigación en una universidad de Berlin, el cuál dividió a la Web Semántica Corporativa en tres áreas importantes: *Corporate Semantic Search*, *Corporate Semantic Collaboration* y *Corporate Ontology Engineering* [6].

Corporate Semantic Search: combina métodos de búsqueda tradicionales con las tecnologías semánticas.

[1] Corporate Semantic Web

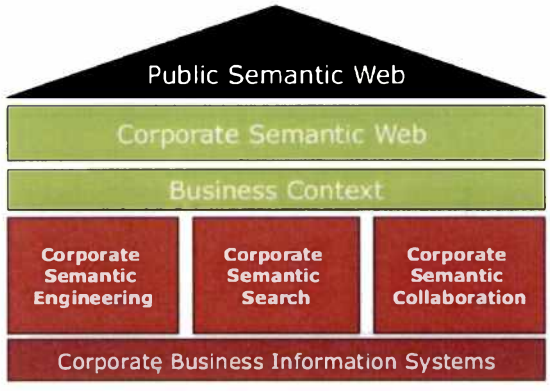


Figura 4.1: Web Semántica Corporativa

Corporate Semantic Collaboration: se investigan métodos y herramientas para modelar la información de manera colaborativa y que pueda ser compartida dentro de la organización.

Corporate Ontology Engineering: trata el desarrollo eficaz y eficiente de ontologías para bajar el costo de desarrollo y mantenimiento. Se investigan y tratan de desarrollar metodologías y herramientas para este fin.

4.2. Otras formas de integrar la información

Como ya explicamos, la Web Semántica presenta una nueva forma de integrar información heterogénea desde distintos bancos de datos. Pero no es la única manera de integración de datos.

La integración de información es uno de los principales desafíos en la **Ingeniería del Software**[2]. Esto se debe principalmente a:

1. La ausencia o disparidad de mecanismos para postular la información en un formato común procesable por las maquinas.

2. Las discrepancias entre los modelos de datos utilizados por los sistemas a integrar.

Habitualmente, las herramientas o sistemas informáticos están orientados hacia el usuario final para el desempeño de sus actividades envueltas

[2] `http://definicion.de/ingenieria-de-software/`

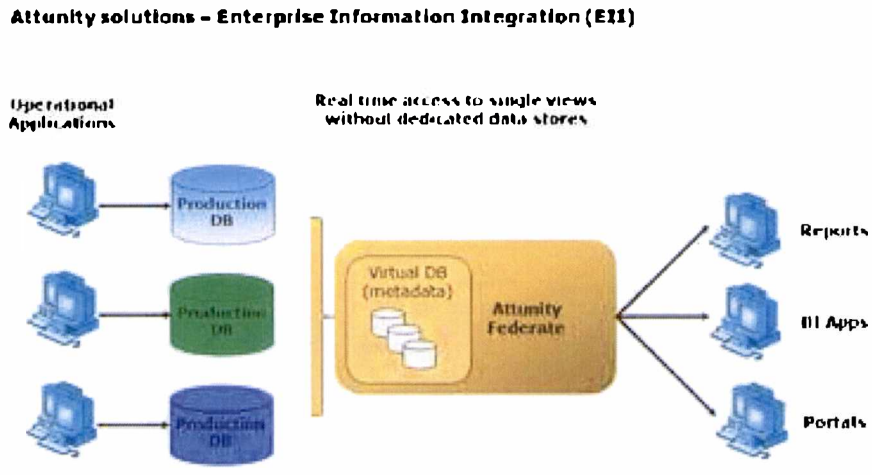


Figura 4.2: Esquema *EII*

por un contexto organizacional. Sin embargo, no todas están concebidas para una explotación automatizada de los datos por otras aplicaciones y sin intervención del usuario [46]. Desde el punto de vista de la integración de información, existen dos estrategias o patrones principales para su implementación [46]:

Enterprise Information Integration (EII): se basa en una arquitectura compuesta por un esquema global y una serie de fuentes externas de datos como muestra la Figura 4.2. Las fuentes externas almacenan los datos reales, mientras que el esquema global ofrece una vista virtual e integrada de la información diseminada en cada una de las fuentes subyacentes.

Extract, Transform and Load (ETL): esta estrategia no utiliza una vista virtual, sino un repositorio central o data warehouse [3] que almacena los datos previamente extraídos de las fuentes externas para luego transformarlos en un modelo común como muestra la figura 4.3.

Con independencia de la estrategia de integración utilizada, toda solución para integrar datos requiere de:

1. Envoltorios o wrappers que permitan extraer o publicar los datos de los sistemas.

[3] Es un sistema usados para almacenar, reportar y analizar datos.

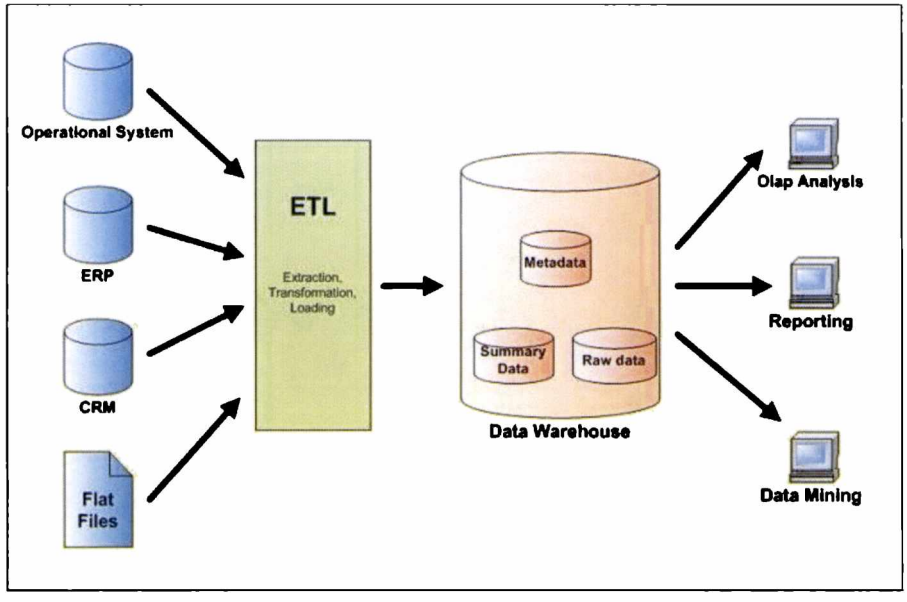


Figura 4.3: Esquema *ETL*

2. Procesos de integración para transformar y transportar los mensajes de datos.

3. Contenedores de información basados en vistas lógicas, en el caso de *EII*, o en almacenes de datos, en el caso de *ETL*.

4. Interfaces para la consulta de los datos.

4.3. Framework para el despliegue y evaluación de procesos software

Este trabajo fue introducido como tesis doctoral de la Universidad de Cadiz, España [46]. Propone un marco de trabajo para el despliegue y evaluación de los procesos software. Este marco de trabajo se basa en la aplicación de las técnicas de la Ingeniería del Software dirigida por modelos y de la integración de información mediante datos abiertos enlazados. Utilizando las primeras, se consigue la adaptación semi-automática de las herramientas de soporte mediante la transformación sucesiva de modelos, partiendo desde el modelo de procesos. Con los datos abiertos enlazados, se consigue que las

herramientas expongan de manera controlada la información que gestionan, para así facilitar la construcción de soluciones de integración destinadas a la evaluación de los procesos. El framework incluye, además de un método sistemático para el despliegue y evaluación, un conjunto de modelos y relaciones, así como una serie de herramientas de apoyo.

Además, implementó un vocabulario que define características comunes de incidentes, basándose en herramientas de gestión de incidentes conocidas, como **Jira**, **Redmine**[4], **Trac**[5], etc.

En nuestro desarrollo, utilizamos este vocabulario mencionado para el modelado de los incidentes, aunque se debieron agregar un conjunto adicional de relaciones. Estas nuevas relaciones fueron agregadas con fines demostrativos sobre características de la Web Semántica.

4.4. SEON

Esta página [26] publica distintas ontologías *OWL* que describen conceptos relacionados con la ingeniería, evolución y mantenimiento de software. En otras palabras, SEON (acrónimo de ***Software Evolution ONtologies***) consiste en múltiples vocabularios para describir entidades como *stakeholders*[6], *actividades*, *objetos abstractos* (por ejemplo *incidentes*), etc. Este conjunto de ontologías permanece categorizado bajo una estructura piramidal, la cual tiene la siguiente clasificación, de arriba hacia abajo:

Conceptos generales: contiene todas aquellas ontologías que implican una característica, ya sea propiedad o relación, común a todos los conceptos y relaciones omnipresentes en el proceso de desarrollo del software. Por ejemplo; una actividad, un archivo, etc.

Conceptos de dominio abarcador: esta categoría incluye conceptos menos abstractos que la categoría mencionada arriba, es decir que los conceptos que se alcanzan abarcan un número limitado de subdominios. Por ejemplo, la ontología de código defectuoso (*Flawed Code*) está principalmente relacionado con características de código fuente y seguimiento de incidentes.

Conceptos de dominio específico: contempla conceptos asociados a dominios específicos pero independientes de la tecnología, vendedor y

[4] `http://www.redmine.org/`

[5] `http://trac.edgewall.org/`

[6] Todas aquellas personas u organizaciones que afectan o son afectadas por el proyecto.

versión utilizada. Por ejemplo, la mayoría de los incidentes se pueden dividir en *Bugs*, *Mejora* y *Agregado*. Los incidentes son reportados por alguien y asignados a alguien para ser resueltos. Dentro de esta categoría, SEON define una ontología para el seguimiento de incidentes [7]. Si bien esta ontología, encontrada luego del desarrollo del prototipo, especifica una modelo bastante completo, faltarían contemplar casos como la visión global de la organización, etc.

Conceptos de sistema específico: esta capa extiende la de arriba, acaparando conceptos específicos y dependientes del lenguaje de programación, vendedor y versión. Por ejemplo, la mayoría de las herramientas de seguimiento de incidentes conciben el concepto de severidad, pero la implementación concreta podría llegar a variar sustancialmente. Depende concretamente de la implementación de la herramienta, como Jira, Mantis, etc.

La Web Semántica fue diseñada principalmente para que las máquinas consuman información. Pese a esta concepción, existen muchas ocasiones donde los humanos necesitan crear interfaces con los datos de la Web Semántica. Es por eso que SEON agrega una capa de etiquetado para el lenguaje natural.

4.5. Semantic Issue Tracker Paper

Este artículo [17] presenta un modelo basado en tecnologías semánticas para la creación y consulta de incidentes en Purple Numbers[32] con un mecanismo rápido y eficiente. El modelo es llamado Semantic Issue Tracker (con su acrónimo **SIT**), es decir que es una instancia de **MediaWiki**[8] con **Semantic MediaWiki Extension** (SMW) y **Purple MediaWiki Extension** (PMWX) instalados. Su objetivo es abordar los problemas recurrentes en los sistemas de gestión de incidentes convencionales de manera inteligente y utilizando *RDF*, *OWL* y *XML*. El modelo es planteado para ser implementado con *Semantic forms*[9], *AJAX scripting*[10] y *SPARQL endpoints* sobre **Purple Semantic MediaWiki** (PSMW). Si bien se trata de

[7] `https://seal-team.ifi.uzh.ch/seon/ontologies/domain-specific/2012/02/issues.owl`

[8] Es un software que permite la gestión de wikis (nombre que recibe un sitio web cuyas páginas pueden ser editadas directamente desde el navegador a través de los usuarios).

[9] `https://semantic-mediawiki.org/wiki/Semantic_Forms`

[10] Es una técnica de desarrollo web para crear aplicaciones interactivas mediante llamados asincrónicos.

abordar las falencias, mediante tecnologías semánticas, por las cuales están pasando las herramientas de administración de incidentes actuales, no se define ni menciona nada en lo concerniente a las ontologías que se deberían utilizar; tópico que en esta tesis se aborda debido al fuerte impacto de cambio arquitectónico implicado en implementaciones con sustento semántico en lugar de un sustento relacional.

Capítulo 5

Red de Ontologías

En este capítulo se describe con mayor detalle el concepto de ontología, mostrando los beneficios que se obtienen a partir de su utilización. En la segunda parte del capítulo describimos las ontologías utilizadas en nuestro sistema y cómo se relacionan entre sí. Por último, se citan algunas decisiones que debieron tomarse a la hora de elegir dichos vocabularios.

5.1. Visión general

La definición mas popular indica que una ontología es la formalización de la conceptualización [55]. Como la Web Semántica es una *estandarización*, la **W3C** define formalmente, en el documento *"W3C's OWL Use cases and Requirements Documents"*, el concepto de ontología como un conjunto común (*estándar*) de términos que son usados para describir y representar un dominio determinado [37]. Si vemos un ejemplo de la vida real quizás se pueda entender mejor:

- Imagine a la **RAE**[1] como la conjunción de la W3C y el documento *"W3C's OWL Use cases and Requirements Documents"*. Ambos juegan un rol de ente regulador de estándares.
- Imagine al alfabeto como a las URIs, RDF, RDFS y OWL. Ambas partes forman una herramienta para la creación de vocabularios dentro de un ámbito.
- Por último imagine a las palabras que la RAE admite, componiendo conceptos comunes para que las personas se comuniquen de tal forma

[1] Real Academia Española.

que se entiendan, como los conceptos que define una ontologías. Por un lado, las palabras admitidas por la RAE forman conceptos y por el otro, las ontologías componen conceptos dentro un ámbito dado. Ambas partes sustentan el entendimiento entre seres humanos y/o computadoras.

Con este ejemplo sencillo se puede apreciar el objetivo de la Web Semántica. Es decir, los documentos distribuidos por la web podrían ser comprensibles por las computadoras a través de agentes informáticos capaces de comprender vocabularios en común.

Hay cosas importantes para esclarecer a partir de la definición de ontología. En principio una ontología es un dominio específico, y es usado para describir y representar un área de aceptación. Un dominio es simplemente un área temática específica, tal como el área de fotografía, medicina, educación, etc [55].

En segunda instancia, una ontología define términos y relaciones entre ellos que sirven tanto para los humanos como para los sistemas informáticos implicados en ciertas áreas de conocimiento. Los términos son, a menudo denominados *clases*, o *conceptos*. Las relaciones entre esas clases pueden ser expresadas usando una estructura jerárquica: las *super-clases* representan conceptos de alto nivel mientras que las *sub-clases* representan conceptos mas concretos. A su vez, los conceptos concretos tienen todos los atributos y características que las clases de nivel superior tienen [55] (concepto identificado como *herencia*).

En tercer lugar, y basándonos en las relaciones entre clases como se mencionó anteriormente, hay otro nivel de relaciones expresadas por el uso de un grupo especial de términos: las *propiedades*. Dichos términos describen varias características de las clases. Entonces podemos decir que las clases poseen *relaciones*, algunas para relacionarse unas con otras y el resto expresadas en términos de propiedades que las describen [55].

Teniendo las clases y las relaciones claramente definidas, la ontología codifica o especifica un dominio de tal manera que el mismo puede ser entendido por las computadoras [55].

Es, pues necesario, que los documentos generados para la web, posean una semántica normalizada, apoyada en ontologías con el fin de que los conceptos y conocimientos del documento puedan ser intercambiados y entendibles por los agentes de software. De este modo estos documentos pueden ser usados por herramientas automatizadas para potenciar los servicios brindados. Por ejemplo, mejores resultados de búsquedas que resuelven los motores disponibles en la web, como Google, etc.

Podemos distinguir entre dos tipos de ontologías [46]:

1. Las de dominio
2. Las de nivel superior

Las primeras representan exclusivamente conceptos aplicables a una determinada parte del conocimiento universal, es decir que actúan o aplican en un ámbito en particular, como la informática, la medicina, etc. Mientras que las segundas están destinadas a representar conceptos aceptados en un amplio rango de dominios, ergo aplican sobre conceptos compartidos en diferentes ámbitos o espacios del mundo real como el tiempo, el dinero, las personas.

Otros vocabularios se irán viendo en el desarrollo de esta investigación, pero a continuación se describen algunos vocabularios relacionados con el software encontrados hasta la fecha [46]:

Description Of A Project (DOAP): vocabulario para describir proyectos de software, en especial para software de fuentes abiertas.

Asset Description Metadata Schema for Software (ADMS.SW): vocabulario creado con propósitos similares al anterior.

SourceForge Trove Map: vocabulario utilizado por la plataforma SourceForge para categorizar proyectos con respecto a su licencia, lenguaje de programación y otros aspectos.

Software Package Data Exchange (SPDX): vocabulario dirigido a estandarizar la forma en la cual las organizaciones publican los metadatos relativos a las licencias de distribución y uso del software.

ISO 19770-2: especificación desarrollada por la ISO con el objetivo de optimizar la identificación y gestión del software a través de un etiquetado común.

Open Services for Lifecycle Collaboration (OSLC): conjunto de especificaciones orientadas a mejorar la integración de plataformas ALM[2] (*Application lifecycle management*).

Las ontologías juegan un rol muy importante para la Web Semántica (de hecho junto con otras tecnologías y lenguajes para generar marcado y

[2]Es la administración del ciclo de vida de un software.

permitir procesamiento semántico, la harán posible), al igual que los conceptos adquiridos por los humanos para establecer relaciones sociales. Por este motivo en la sección que sigue se detallan los beneficios desencadenados a partir del uso y reutilización de ontologías.

5.2. Beneficios del uso de las Ontologías

Los beneficios de utilizar ontologías son los siguientes [55]:

- Provee una *definición globalmente común* acerca de un dominio específico.
- Ofrece términos que se pueden utilizar cuando se crea un documento *RDF* para cierto dominio relacionado a la misma. Por ejemplo, para dos sistemas totalmente diferentes que manejan usuarios, el concepto de *nickName*, semánticamente hablando, es el mismo en ambos.
- Provee un medio para *reutilizar* representaciones de un dominio dado.
- Expresa explícitamente las características del *dominio*.
- Junto con los lenguajes de descripción de ontologías (tal como *RDF* explicado en la Sección 3.2.3, *RDFS* explicado en la Sección 3.2.5 y *OWL* explicado en la Sección 3.2.7), presentan los datos de manera tal que las computadoras puedan çomprenderlos".
- Hace posible el *procesamiento automático por parte de las computadoras*.

[55] A esta altura queda claro que *RDF* promueve el desarrollo y uso de vocabularios comunes. A continuación, se detallan algunos de los beneficios de usar ontologías y RDF:

- Sin un vocabulario compartido, siempre sera posible que una *misma palabra* tenga *significados diferentes* y/o que diferentes palabras puedan referirse al mismo concepto.
- Sin un vocabulario compartido, la *información* distribuida en la web estará *aislada*, es decir que no tendrían una conexión alguna para poder relacionarla. Ergo la integración de informacion seria una complicación. Sin embargo esta característica es uno de los puntos sobre los cuales, la Web Semántica, pone énfasis.

- Sin un vocabulario compartido, la *inferencia*; presentada en la Sección 3.2.6; sería *difícil de implementar*, aunque de por sí descubrir nuevas aceptaciones es complicado.

Hoy en día existen muchas ontologías de uso cotidiano, entre las que se destacan *FOAF*, *SKOS*, *Dublin Cor* (DC), *Schema*, etc.

5.3. Descripción de la red de ontologías utilizadas

Clases	Propiedades
rdfs:Literal	dc:created
owl:Thing	dc:modified
skos:Concept	dc:relation
dc:Agent	dc:creator
foaf:Person	rdf:type
schema:Comment	foaf:name
schema:Text	foaf:nick
itm:Issue	itm:completedDate
rdfs:Member	itm:category
itm:Category	itm:responsible
ownns:Issue	itm:priority
ownns:Category	itm:status
ownns:IssueComment	itm:owner
ownns:Person	itm:description
-	itm:name

Cuadro 5.1: Visión general de la red de ontologías

A continuación se detalla la red de ontologías utilizadas para el desarrollo del prototipo, justificando debidamente la elección de cada una. Previo a esto, vamos a explicar qué es el dominio y el rango de una propiedad.

Se suele denominar *dominio* de una propiedad al conjunto de clases que describe o caracteriza dicha propiedad. Por otro lado, el *rango* de una propiedad son las clases sobre las cuales puede desembocar la propiedad. Por ejemplo, si tuviésemos una propiedad denominada *"produce"*, una clase dentro de su dominio podría ser *Bodega* mientras que *Vino* podría ser una clase del rango de la misma.

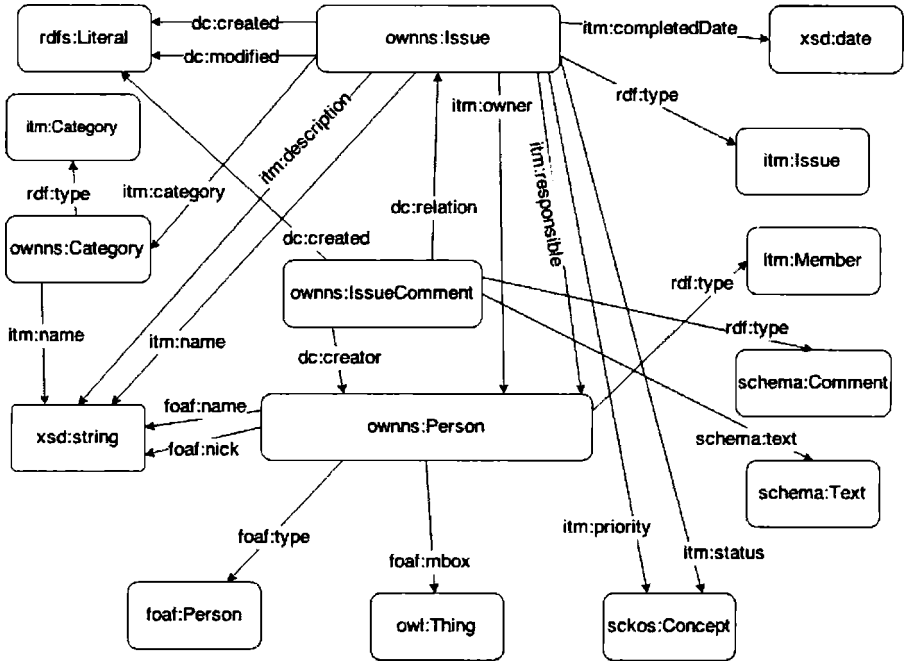


Figura 5.1: Red de ontologías utilizadas para realizar el prototipo

5.3.1. FOAF

El nombre FOAF[29] es un acrónimo de *Friend of a Friend* (Amigo de un Amigo). Es un proyecto abocado a enlazar personas e información usando la web. La misma integra tres tipos de redes: redes sociales, amigos y asociaciones, redes representacionales que describen una vista simplificada de dibujos animados y redes de información que usan enlazado basado en web (*LOD*) para compartir descripciones publicadas de forma independiente de este mundo interconectado. Es una de los vocabularios más conocidos y nos fue clave para representar una persona.

Person: sirve para representar a personas del mundo real. Es *sub-clase* de un *foaf:Agent*, entre otras.

mbox: especifica la dirección de correo de una persona única. Poseer esta propiedad también implica que el dominio y el rango sean *foaf:Agent* y *foaf:Thing* respectivamente. Cabe aclara que el propietario podría tener ninguna, una o mas direcciones de correo.

name: esta propiedad define el nombre de una cosa, como una persona, un proyecto, etc. Su dominio especifica que debe ser de tipo *foaf:Thing* y su rango un simple *literal*.

nick: propiedad que relaciona a una persona (*foaf:Person*) con su nombre corto usado en contextos donde existen cuentas de usuario, logins, chats, etc.

5.3.2. SKOS

SKOS[41] es también un acrónimo, que hace referencia a *Simple Knowledge Organization System* (Conocimiento simple en sistemas de organizaciones). Es un modelo de datos común para compartir y enlazar conocimientos sobre sistemas de organizaciones a través de la Web Semántica. Más que nada esta ontología se utilizó para modelar componentes predefinidas por Callimachus del tipo "`http://www.w3.org/2004/02/skos/core#Concept`", tales como las prioridades, los estados, etc.

Concept: esta clase puede verse como una idea o noción. Es útil cuando se quiere describir una estructura conceptual o intelectual específica. Esta clase fue utilizada indirectamente para representar prioridades y estados, los cuales forman un conjunto finito de valores.

prefLabel: define una etiqueta preferida para un *skos:Concept* mediante una cadena de caracteres *UNICODE*[3]. Es útil cuando se crean representaciones de partes de un sistema legibles o comprensibles por los humanos.

5.3.3. ITM

ITM[46] son las siglas para *Issue Tracking tool Model* (Modelo para herramientas de Gestión de Incidentes). Este vocabulario define características comunes de herramientas para la administración de tareas o sistemas de registro de incidentes tales como **Jire**, **Redmine**, **Mantis**, **GitHub**, etc. Es el vocabulario sobre el cual nos apoyamos para realizar el prototipo debido a que su dominio coincidía con el de nuestra propuesta. La representación de un issue es un claro ejemplo para el cual nos sirvió.

[3]Es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad.

Issue: es una clase para describir una tarea o unidad concreta de trabajo necesaria para evolucionar un sistema informático, como añadir una nueva característica, arreglar un fallo, etc.

Member: clase utilizada para asociar un determinado usuario a un proyecto, pudiendo hacer uso de un rol determinado. Se lo puede usar como dominio de *itm:owner*, *itm:responsible*, etc. En el prototipo **SGI** se lo utilizó con esas dos propiedades nombradas.

IssueCategory: clase que permite realizar una clasificación específica de los incidentes de un determinado proyecto.

name: es una propiedad para especificar el nombre de un proyecto, tarea (incidente a resolver), una categoría de incidentes, etc. Su dominio puede ser algo de tipo *itm:Issue*, *itm:Project*, entre otros y su rango es un *literal*.

category: especifica la categoría de la tarea. El dominio y el rango son *itm:Issue* e *itm:IssueCategory* respectivamente.

description: detalla la descripción de un proyecto, tarea, entre otros. Su dominio es *itm:Project*, *itm:Issue*, etc. Su rango es un *itm:Issue*.

status: estado de una versión o tarea de un proyecto. Tiene como dominio *itm:Version* e *itm:Issue* y además como rango tiene *skos:Concept*, clase de la cual nos basamos para modelar los diferentes estados.

priority: es la prioridad con la que una tarea debe ser realizada. Su dominio es *itm:Issue*, siendo su rango *skos:Concept*, clase de la cual nos basamos para modelar las diferentes prioridades en el prototipo del **SGI**.

owner: indica quien es el propietario de la tarea. Su dominio es un *itm:Issue* y su rango es un *itm:Member*, clase de la que es tipo una persona en el **SGI**.

responsible: es el miembro que se responsabiliza de la realización de la tarea. Exige como dominio una instancia de la clase *itm:Issue* y como rango una instancia de la clase *itm:Member*.

5.3.4. SCHEMA

SCHEMA[28] es una ontología introducida por Google, Microsoft y Yahoo, que provee una colección de términos que los desarrolladores pueden utilizar

para mejorar la indexación por parte de los motores de búsqueda y así presentarle al usuario una mejor respuesta a la búsqueda realizada.

Comment: clase con el fin de representar un comentario de un item, por ejemplo un comentario en un blog. El contenido de los comentarios se expresa mediante la propiedad *schema:text* y su tópico se expresa mediante la propiedad *schema:about*, propiedades compartidas por todos los *schema:CreativeWork*.

text: es el contenido textual de un *schema:CreativeWork*, que es un tipo genérico de trabajo creativo, como un libro, una película, software, etc.

5.4. Determinaciones ambiguas

Durante el modelado de la aplicación, nos encontramos que algunos de los términos o conceptos que queríamos modelar, podían ser modelados con diferentes clases de distintas ontologías. Estas son algunas de las situaciones ambiguas sobre las que tuvimos que tomar una decisión y el porqué de la decisión:

- A la hora de elegir el tipo de las instancias de los comentarios de los incidentes se eligió *Schema: Comment*[4] en lugar de *Sioc: term_ Post*[5] debido a que el primero es más específico. Es decir, el primero hace referencia a un comentario, sencillo y concreto, en un blog/discusión y el segundo abarca conceptos más grandes y complejos como lo es el de un artículo.

- Se eligió *Schema: text*[6] para mantener la homogeneidad con la clase elegida para representar un comentario. Además, es; semánticamente hablando; más específico que *DC: description*[7] debido a que este último incluye representaciones gráficas y lo que se buscó fue algo más concreto y simple, como lo es el texto plano.

- Se eligió *DC: creator*[8] debido a que posee un nivel de restricción menor en cuanto a dominio y rango. Esta flexibilidad semántica nos ayudo ya que el rango para esta propiedad en nuestra aplicación es un usuario

[4] `http://schema.org/Comment`
[5] `http://rdfs.org/sioc/spec/#term_Post`
[6] `http://schema.org/text`
[7] `http://purl.org/dc/elements/1.1/description`
[8] `http://purl.org/dc/elements/1.1/creator`

de Callimachus. En cambio *Sioc: term_ has_ creator*[9] define estrictamente un dominio y un rango. Esta restricción semántica nos hubiera llevado a definirle explícitamente un tipo más al recurso que representa un usuario de Callimachus o resolverlo mediante inferencia de manera que se represente un modelo compatible respetando los estándares.

[9] `http://rdfs.org/sioc/spec/#term_has_creator`

Capítulo 6

Diseño de la aplicación

En este capítulo veremos en detalle las tecnologías utilizadas como también cuestiones de diseño, accesibilidad y usabilidad de la aplicación desarrollada. Explicaremos en profundidad el framework elegido, **Callimachus**, y brevemente describiremos su par Jena, el cual se había tomado en consideración en un primer momento. Además, trataremos la interfaz de usuario elegida y la experiencia de usuario del **SIT**.

6.1. Tecnologías para el desarrollo

En las etapas iniciales del desarrollo, luego de definido el modelo de datos de la aplicación, tuvimos que probar y decidir qué tecnologías (*framework*, *lenguajes*, *herramientas*, etc.) íbamos a utilizar.

Se analizaron varias tecnologías candidatas. Entre las herramientas descartadas, vale la pena remitirse brevemente a algunas de ellas. En particular, hablaremos de *HAL* y *Jena*.

6.1.1. HAL

HAL[1] [40] es un formato simple que posibilita conectar recursos de una API mediante hipervínculos de manera sencilla. La utilización de este formato permite a los usuarios de una API, explorar la estructura y los recursos que este publica. Provee un conjunto de reglas y convenciones para expresar estas relaciones en formato *JSON*[2] [10] o *XML*.

[1] Hypertext Application Language
[2] JavaScript Object Notation

Inicialmente, pensamos la aplicación con un servidor *REST* exponiendo servicios en formato *JSON* y un cliente *JavaScript* que consumiese dichos servicios. HAL nos brindaba la capacidad de desacoplamiento entre cliente y servidor que nosotros necesitábamos. La flexibilidad de los modelos en el servidor, hacían cambiar la estructura de los servicios REST, y el cliente necesitaba algún mecanismo para adaptarse ante estos cambios. Mediante la exploración de la API, esto era posible. Así también, se podía llegar a generar herramientas que crearan código *HTML* automáticamente a partir de un archivo *HAL-JSON*.

HAL se terminó descartando cuando se eligió Callimachus como framework de desarrollo, cambiando la idea original. Con este framework, no utilizamos una API REST en el servidor, sino que directamente se devuelven los archivos estáticos. Por eso, fue imposible la utilización de esta tecnología en el proyecto.

6.1.2. Jena

[47] Jena es un framework libre para construir aplicaciones usando Web Semántica y Liked Data para Java[3]. Presenta una API para extraer y escribir datos en un grafo RDF. Esos grafos se representan como modelos abstractos y pueden ser populados desde archivos, bases de datos, URLs o alguna combinación de los anteriores. Ademas se puede consultar los datos RDF a través de SPARQL.

Capa de Grafo (SPI): es la capa base. Es en realidad una *SPI*[4]; una descripción de las clases, interfaces y métodos a implementar y extender. Es una capa muy chica donde se almacena el grafo. La funcionalidad es básica, el corazón de las funciones reside en otras capas, por lo que permite una gran variedad de implementaciones de esta capa, como in-memory o triplestores. En la práctica, los desarrolladores no interactúan contra esta capa, sino más bien con la de Modelo.

Capa de Modelo (API): la capa de Modelo extiende la funcionalidad base de la capa de Grafo, abstrayendo al desarrollador de términos como *nodo*, *tripleta* y facilitando objetos como *recurso*, *sentencia* o *propiedad*.

Capa de Ontología (API): la tercera y última capa de Jena es la capa de Ontología. Esta brinda la funcionalidad de inferencia. Es decir, la

[3] `https://www.java.com/es/`
[4] Interfaz de Proveedor de Servicios

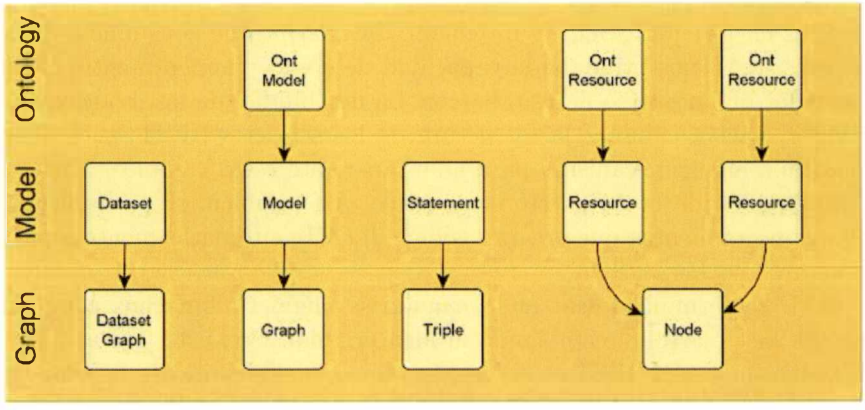


Figura 6.1: Arquitectura del framework Apache Jena

habilidad de trabajar con tripletas implícitas, además de aquellas que fueron explícitamente definidas.

A pesar de que se trata de un framework popular y fácil de usar, no elegimos Jena para el desarrollo. Esta decisión no fue tomada por desventajas o problemas en el uso de Jena, sino que Callimachus nos brindaba algo que Jena no: una manera fácil de crear una aplicación web teniendo en cuenta la base de datos la interface, etc.

6.1.3. Callimachus

[5] Callimachus es una framework para la creación y ejecución de sitios web basados en la administración de los datos, utilizando *Linked Data* y *Web Semántica*. Permite crear aplicaciones web semánticas de manera fácil y rápida; usando datos de un grafo RDF ya existente o, también, creando nueva información.

A continuación veremos algunos de los puntos importantes del framework en cuestión.

Recursos Callimachus

Una aplicación en Callimachus está construida a partir de recursos Callimachus. Estos últimos incluyen archivos, carpetas, clases, dominios, con-

ceptos, entre otras cosas. A continuación veremos solo los más importantes para conocer la estructura de las aplicaciones [5]:

Archivo: los archivos binarios y de texto son usados para almacenar muchos de los recursos de Callimachus y son el componente principal de la aplicación.

Clase: una clase describe un conjunto de recursos con características en común. En Callimachus, una clase es un conjunto de recursos RDF que usan los mismos templates.

Dominio: un dominio representa un conjunto de agentes de red (por ejemplo, todos los usuario logueados). Se utiliza para dar permisos a los recursos Callimachus. Callimachus provee un servicio de control de acceso, manejando la creación de usuarios y sus permisos. Cada usuario puede tener acceso a una clase de recursos o a un recurso en particular. Los permisos que se les puede dar a los usuarios son: de lectura, de escritura, de contribución o de administración.

Documento de Grafo: un Documento de Grafo es un documento RDF en formato Turtle. Se usa para agregar información a la base de datos. Por ejemplo, cuando se quiere importar datos de un grafo, basta con agregar el documento del Grafo en Callimachus para que se agreguen las tripletas a la base.

Archivos de consulta: contiene una consulta SPARQL que puede ser ejecutada sobre la triplestore.

Pipeline: un pipeline ejecuta una secuencia de operaciones en documentos XML determinados, como por ejemplo la transformación en RDF para poder ser añadidos al triplestore.

Triplestore

Como mencionamos anteriormente, Callimachus permite popular la base de datos de dos maneras; creando los datos directamente en la aplicación o importándolos de un archivo RDF. En nuestro caso, decidimos usar ambos métodos para demostrar una de las características del modelo semántico, que es la coexistencia de datos en la misma base, que pueden tener más o menos atributos que otras instancias de la misma clase.

El archivo importado en nuestra aplicación fue obtenido de un repositorio de **GitHub**. Se exportaron los issues de dicho repositorio y se integró con los

datos previamente cargados en el sistema sin necesidad de realizar cambio alguno en la estructura del modelo, ni detener el servidor.

Para poder lograr la persistencia de esta información semántica, Callimachus provee un endpoint *SPARQL* contra el que se pueden realizar consultas con el fin de almacenar, manipular o consultar los datos en formato *RDF*. Además este framework provee una *API REST* con el fin de poder acceder a cada uno de los recursos almacenados. Si bien existen diferentes endpoints *SPARQL*, como *Virtuoso* y otros, Callimachus utiliza *Sesame*. Por defecto, existe un Datasource creado, pero pueden crearse varias instancias dentro de la misma aplicación sin problemas [5].

A pesar que el desarrollador que utiliza Callimachus no interactua directamente con *Sesame*, es importante conocer sus principales características para comprender el funcionamiento de Callimachus.

Sesame

[24] Sesame es un framework abierto en Java para gestionar datos *RDF*. El mismo es fácil de extender y configurar con respecto a mecanismos de almacenado, motores de inferencia, formatos de los archivos RDF, etc. aunque si lo utilizamos bajo Callimachus, no será necesario configurar ninguna de estos parámetros.

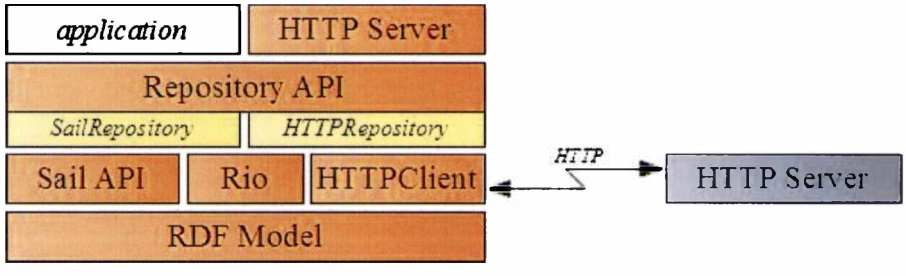


Figura 6.2: Modelo en bloques de la arquitectura del framework Sesame

Entre sus componentes principales se encuentran implementaciones de triplestore en memoria y de triplestore en disco, junto con dos procesos que sirven para administrar y proveer acceso a esos triplestores. Además, Sesame soporta dos lenguajes de consulta: *SPARQL* y *SeRQL*[5], siendo el primero el utilizado en las implementaciones de Callimachus. Por último, cabe destacar que Sesame cuenta con un componente llamado *AliBaba*, una API confor-

[5] `http://rdf4j.org/sesame/2.7/docs/users.docbook?view#chapter-serql`

mada por diferentes modulos que abstraen el almacenamiento RDF para acelerar el desarrollo y facilitar el mantenimiento de las aplicaciones [2].

En la Figura 6.2 se puede observar una vista completa acerca de la composición y las dependencias de este framework. En la parte inferior se encuentra el modelo RDF, la base sobre la cual se sostiene toda la arquitectura. Al ser un framework orientado a RDF, todas las partes de alguna manera dependen de este modelo, el cual define interfaces e implementaciones para las entidades RDF: *URI*, *literales*, etc.

[24] Rio, que significa *RDF I/O*, consiste en un conjunto de parseadores y escritores para diferentes formatos de archivo RDF. Los parseadores y los escritores pueden ser utilizados para traducir archivos RDF en un conjunto de declaraciones, y los escritores para la operación inversa. Ambos con la ayuda adicional de AliBaba.

[25] La capa denominada *Sail API* es una API de bajo nivel para almacenar e inferir datos RDF. Este genera una abstracción de los detalles de las operaciones de almacenamiento y de inferencia, como también poder utilizar, abajo, diferentes implementaciones. Hay varias implementaciones de esta API, por ejemplo *MemoryStore* el cual almacena datos RDF en memoria y *NativeStore* el cual utiliza estructuras de datos sobre discos para poder lograr la persistencia. En la Figura 6.3 se puede apreciar con mayor detalle las partes con las que se compone dicha API.

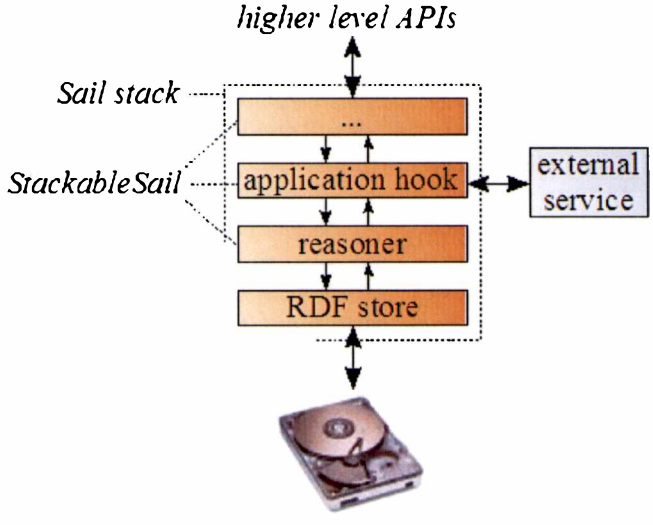


Figura 6.3: Modelo en bloques de la arquitectura de la API Sail

[24] La capa de *Repository API* es una API de alto nivel que ofrece un gran número de métodos orientados al desarrollador para el manejo de datos RDF. El principal objetivo de esta API es hacer lo más sencillo posible la vida de los desarrolladores de aplicaciones. La misma ofrece varios métodos para subir archivos de datos, consultar, extraer y manipular datos. Hay muchas implementaciones de la misma, las que se muestran en la figura son *SailRepository* y *HTTPRepository*. El primero traduce las llamadas a una implementación de SAIL, mientras que el ultimo ofrece una transparencia en la comunicación entre el cliente y el servidor con un servidor Sesame sobre HTTP.

[24] En la parte superior de la figura se encuentra el componente *HTTP Server*. El mismo consiste en un conjunto de procesos Java que implementan un protocolo para poder acceder a los repositorios Sesame a través de HTTP. La mayoría de las personas pueden utilizar una librería cliente para manejar la comunicación. El componente *HTTPClient* que es utilizado por el componente *HTTPRepository* es una de esas librerías

Templates

Este framework permite crear no solo la información para ser usada por personas y máquinas, sino que también facilita la creación de vistas *HTML* para que esa información pueda ser creada, actualizada y mostrada de manera legible por las personas.

Las vistas se crean utilizando *XHTML* y *RDFa*, explicado en la sección 3.2.4, para unir los datos RDF con el template [5]. Si se desea usar datos del grafo en un template (por ejemplo, para listar todos los issues del sistema) se deberán obtener a partir de consultas SPARQL contra la base a través del endpoint que Callimachus brinda.

6.1.4. Interfaz de usuario

Si bien el objetivo de la tesis no es la realización de un **SGI** con una interfaz de usuario simple e intuitiva, una *UI*[6] bien diseñada ayuda al rápido entendimiento de la aplicación y sus funciones por parte de los usuarios.

Para facilitar el desarrollo, Callimachus trae incluido por defecto el framework Bootstrap para el armado de documentos **HTML**. **Bootstrap**[7]

[6] Iterfaz de Usuario.

[7] Bootstrap: `http://getbootstrap.com/`

[3] es un framework **HTML**, **CSS**[8] y **JS**[9] para desarrollar proyectos en la web para todos los dispositivos, es decir, para computadoras de escritorio, portátiles y móviles[10]. Bootstrap le da estilo a nuestros sitios para lograr una interfaz amigable, pero además nos brinda una serie de componentes desarrollados puramente en JS y HTML que podemos utilizar directamente en nuestras páginas.

A continuación se mostrara una serie de imágenes sobre diferentes puntos operables del **SGI** para dar una leve noción de como operar con él.

Menú principal

En la Figura 6.4 se da a conocer el menú con el que el usuario de la aplicación se toma en una primera instancia. A partir de el podrá ir navegando hacía los diferentes puntos más globales de la aplicación y así seguir navegando por las páginas más concretas. Como se podrá apreciar, tendrá acceso al listado de los *incidentes* reportado pudiéndolos filtrar, al listado de las *personas* y a la creación de nuevas *categorías* en las cuales encuadrar a los diferentes incidentes.

Listado de Incidentes

En la Figura 6.5 queda demostrado como el usuario podrá visualizar todos los *incidentes*. Adicionalmente podrá realizar una filtración por el nombre completo, o incluso solo una parte, del creador. La información que se facilita es el nombre que se le asignó al incidente, la fecha de creación y la posibilidad de cambiarle el responsable que tiene asociado.

Alta de Incidente

La Figura 6.6 detalla un formulario, el cual posibilita al usuario para la creación de un nuevo incidente detectado. En este punto, la aplicación solicitará un nombre con el cual identificar rápidamente el incidente por parte de los usuarios, la asignación de la categoría a la que pertenece, una descripción opcional para explayarse, el estado del ciclo de vida en el que se encuentra, la prioridad con la que se requiere atender el mismo y el responsable que tiene que resolverlo. Una vez que el usuario completa todos los campos y presiona en el botón de confirmación del alta, la aplicación redireccionará automáticamente a la página de la vista del incidente.

[8] Cascading Style Sheets.
[9] JavaScript.
[10] Concepto conocido como Responsive Web.

Figura 6.4: Captura de pantalla de la página con la que se tiene el acceso a la aplicación

Vista de Incidente

Finalmente en la Figura 6.7 queda plasmado que el usuario podrá saber el estado en el que se encuentra el incidente, la categoría en la que encuadra, el nivel de prioridad que posee, su nombre identificatorio, una descripción pertinente, por quien fue creado y a quien se lo asignaron. Otra característica de esta vista es la inclusión del listado de comentarios que eventualmente fueron agregados al mismo. Cabe aclarar que desde la misma se podrá seguir navegando a diferentes partes de la aplicación, como crear un nuevo comentario, ver los perfiles del usuario creador y el usuario al que se lo asignado, etc.

Figura 6.5: Captura de pantalla de la página con la que podrá ver todos los incidentes e incluso filtrarlos

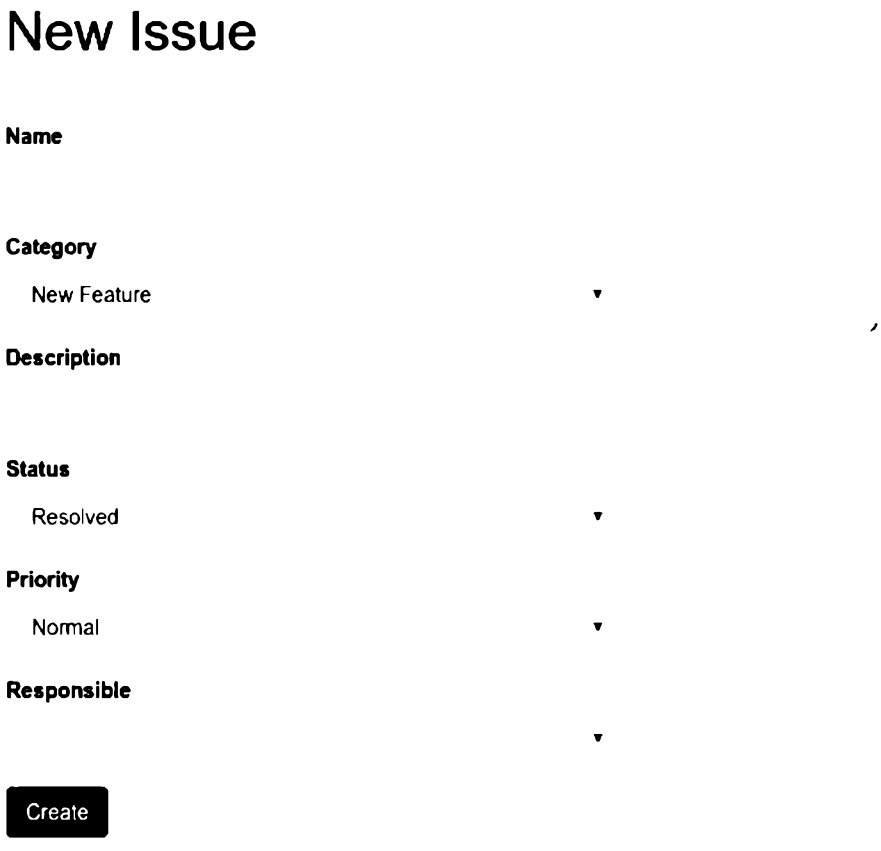


Figura 6.6: Captura de pantalla de la página con la que se provee el formulario para dar de alta un incidente

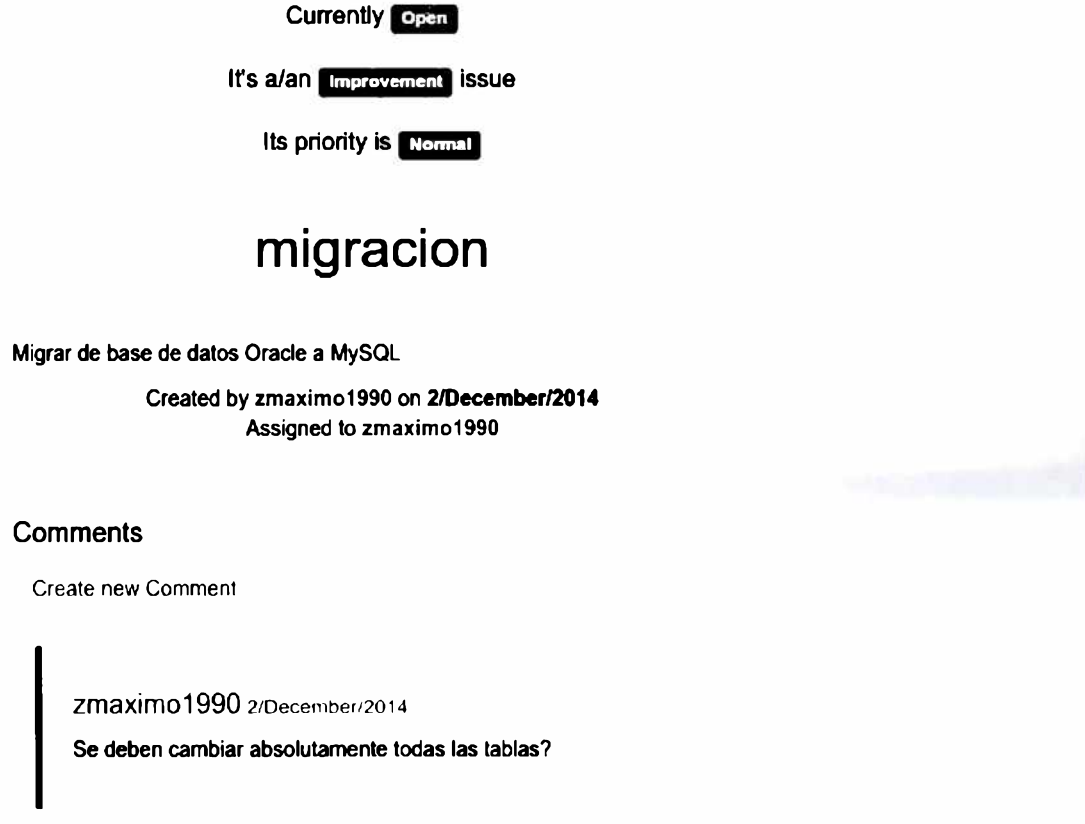


Figura 6.7: Captura de pantalla de la página con la que se visualiza la información completa del incidente

Capítulo 7

Fortalezas y Debilidades

En este último capítulo antes de exponer las conclusiones del trabajo, intentamos señalar los puntos fuertes y los puntos débiles que presenta la utilización de las tecnologías de la Web Semántica comparado a la utilización de las tecnologías más tradicionales: paradigma orientado a objetos y bases de datos relacionales.

7.1. Web Semántica vs tecnologías tradicionales

A medida que desarrollábamos el prototipo propuesto, nos fuimos encontrando con dificultades y puntos a favor en cuanto a la utilización de la Web Semántica. Ahora que ya describimos las tecnologías, la arquitectura del sistema, y tenemos una idea más amplia del modelo semántico utilizado, procederemos a describir las ventajas y desventajas del uso de la Web Semántica dentro del marco propuesto. Estas ventajas y desventajas saltan a la vista al comparar el sistema y su desarrollo con sus semejantes que utilizan un modelo orientado a objetos y bases de datos relacionales.

En la Sección 3.3 ya hicimos una comparación de los modelos y mostramos las diferencias. Pero no dijimos explícitamente si esas diferencias son ventajas o desventajas del modelo. Algunas pueden inferirse, ya que pueden resultar obvias a simple vista, pero otras no tanto. Por eso en este capítulo, usaremos las diferencias encontradas anteriormente y mostraremos de qué manera resulta beneficiado, o no, el usuario y/o desarrollador.

7.1.1. Fortalezas

1. El modelo semántico facilita la *integración* de la información proveniente de múltiples bancos de datos con ontologías diferentes, ya que existe un único lenguaje para representarlos, RDF [23][30]. En el paradigma orientado a objetos, la integración puede no ser tan sencilla si los datos que utilizamos están definidos utilizando lenguajes diferentes. Además, para utilizar dicha información, la estructura de los objetos debe ser la misma (o al menos conocer la estructura que usa el otro modelo). En cambio, RDF evade dicha restricción con la re-utilización de ontologías.

2. SPARQL soporta consultas de tipo *ASK*, *DESCRIBE* y *CONSTRUCT*, mientras que SQL no. Si bien operadores como *ASK* o *DESCRIBE* pueden ser implementados de alguna manera en SQL, las mismas no existen de forma nativa.

3. SPARQL permite hacer *consultas federadas*, mientras que con SQL no es posible. Una consulta federada consiste en realizar una consulta sobre varios repositorios remotos. En SQL un acercamiento a este tipo de consultas son las consultas que involucran más de un esquema de datos, que a diferencia de SPARQL los esquemas residen en el mismo servidor. Otra solución que utilizan los diferentes motores SQL para emular este tipo de consultas es la realización de consultas por separado para luego unir la respuesta localmente [30].

4. Una de las ideas principales de la Web Semántica es el uso de *URIs* para identificar a los recursos [55][8][48][15]. Además, se pueden agregar reglas de inferencia que permitan distinguir cuando dos recursos son el mismo (por ej. si dos recursos tienen el mismo email personal). De esta manera, los motores de inferencia resuelven las posibles ambigüedades que existen entre la información de distintos bancos de datos. Existen lenguajes que proveen propiedades para indicar relaciones semánticas ponderadas, como *sameAs* de OWL [48]. Por su parte, el modelo clásico no tiene estas capacidades, sino que se debe agregar lógica en el motor de base de datos o en la aplicación misma para identificar objetos repetidos.

5. En el modelo clásico, un cambio en un objeto significa tener que *cambiar* la tabla en la base de datos donde se almacenan. Así, si originalmente tenemos una clase Persona con un único atributo *nombre* y una

tabla en la base de datos con una única columna *nombre*, y luego agregamos un nuevo atributo *apellido* en la clase Persona, en la tabla de la base de datos también debemos cambiar la estructura. Y si además este nuevo atributo no permite valores nulos, debemos setear un valor en todas las instancias de la clase Persona. En el modelo semántico esto no sucede. Distintas instancias de la misma clase pueden utilizar los atributos de la clase que ellos quieran. No existen los atributos obligatorios a nivel de base de datos [23][22].

6. La Web Semántica permite acceder a los datos de dos maneras: consultando a un endpoint *SPARQL* o accediendo a una *URL*. Así, en caso de que haya una baja en el servidor del endpoint, se podría seguir accediendo a los datos, aunque no se podrían manipular.

7. Mientras que SPARQL tiene una única *estandarización* (realizada por la **W3C**) [55][27], SQL tiene muchas variantes (*ANSI*, *ISO*, etc.) [36][21].

8. SPARQL te permite, a través de los denominados *quadstore*, llevar el rastro de donde proviene cada porción de la información, y te permite almacenar metadatos sobre el origen, permitiendo filtrar los resultados según su procedencia [20]. En SQL, se debería crear un campo especial en todas las tablas de todas las bases de datos para conseguir el mismo esquema, algo que no es escalable ni eficiente.

7.1.2. Debilidades

1. La Web Semántica todavía no está popularizada. Mientrás que SQL aparecío en el año 1984 [21], SPARQL fue estandarizado en el año 2008 [55]. Esto refleja una amplia ventaja de SQL en cuanto a *maduración*.

2. La maduración de SQL ante SPARQL se sustenta con las herramientas disponibles en el mercado y las funciones que este provee. Por ejemplo, las *transacciones* en SPARQL todavía no son estables en su totalidad.

3. Si bien la flexibilidad que presenta la Web Semántica aporta determinadas ventajas, también a su vez delega mayor *responsabilidad* sobre el desarrollador, debido a que debe tener en cuenta posibles inconsistencias por falta de una estructura rígida subyacente.

Capítulo 8

Conclusiones y trabajo futuro

Luego de presentar un prototipo de **SGI**, Sistema de Gestión de Incidentes, es momento de detallar las conclusiones de este trabajo teniendo en cuenta diferentes puntos evaluados y los trabajos futuros que deja esta investigación.

8.1. Conclusiones sobre la Web Semántica

Como se vio en capítulos anteriores, la Web Semántica puede verse como una colección de estándares y tecnologías que permiten a las computadoras entender el significado de la información en la web, al igual que lo hacen los seres humanos. Presenta una nueva forma de estructurar la web (*datos*) para que la información pueda ser procesada automáticamente por computadoras a gran escala dándole a la web otro potencial.

A pesar de que es una tecnología relativamente nueva, y que todavía falta mucho por investigar y explotar, ya se empiezan a ver aplicaciones que la utilizan mejorando considerablemente la experiencia de usuario y plasmando su potencia de procesamiento automático.

Junto con el crecimiento de la Web Semántica, aparecieron un conjunto de estándares, tecnologías y herramientas necesarias y relacionadas con el concepto de Web Semántica. La Web Semántica combina un mecanismo de nombramiento y direccionamiento distribuido (**URIs**) con una representación formal, **sujeto-relación-objeto**, especificados por **RDF**. **OWL** permite desarrollar un modelo mucho más rico facilitando capacidades de inferencia. También se revela una técnica para obtener datos RDF desde

páginas con formato XML o XHTML denominado **GRDDL**, un lenguaje de consulta común **SPARQL**, con el que se podrán realizar operaciones del tipo *SELECT*, *ASK*, *DESCRIBE* y *CONSTRUCT*. Por último se necesita de un paradigma de persistencia para los datos **Quad/Triple-Store**. Agregado a todo esto, la Web Semántica permite serializar un modelo de datos en una variedad de formatos entre los que se encuentran **XML**, **Turtle**, etc.

Para asegurar que las soluciones de integración sean lo más independientes posible de las herramientas concretas que se desean integrar, es preciso utilizar vocabularios (también conocidas como **ontologías**) compartidos. Una premisa fundamental en el enfoque **LOD** es que a la hora de publicar datos en RDF es conveniente reutilizar, siempre que sea posible, los términos procedentes de vocabularios ya existentes, en lugar de reinventarlos. De esa forma, se aumenta la probabilidad de que las aplicaciones desarrolladas con vocabularios ya existentes puedan consumir nuevos datos, ya compatibles, sin necesidad de realizar modificaciones adicionales.

Como ya se evidenció, el desarrollo de aplicaciones empresariales con sustento semántico es reciente, por lo tanto es un paradigma sobre el cual queda mucho por andar e investigar. Como la presente investigación presenta; uno de los campos en donde se lo podría aplicar es en el desarrollo de aplicaciones empresariales, donde las compañías cada vez más tienden a moverse hacia modelos colaborativos, los cuales tienen que ser flexibles y fácil de integrar. Es este uno de los tópicos en donde la Web Semántica pisa fuerte. Con este ideal, es necesario que las compañías tengan una arquitectura de datos flexibles con el objeto de que puedan integrar datos desde colaboradores, con datos generados internamente.

8.2. Conclusiones sobre las Contribuciones

Esta tesis fue motivada por ciertas incógnitas que impulsaron al desarrollo de la misma. Se deseaba intentar dar respuestas a las mismas con el fin de indagar la aplicación de tecnologías y estándares que abarca la Web Semántica sobre software empresarial. Estas son:

> Para el caso particular de un sistema de gestión de incidentes, ¿qué cambios implica la adopción de las propuestas de la Web Semántica (en términos de arquitectura de software y del proceso de desarrollo), respecto de las metodologías tradicionales basadas en modelos relacionales?

Como se ha visto en los capítulos desarrollados durante esta investigación, es necesario agregar contenido semántico a los documentos desperdigados por la web, para lo cual se deben tener en cuenta los siguientes puntos:

1. Debe existir una forma de representar la información en la web. Y esta representación tiene que poder ser interpretada por una computadora.
2. Esta representación debe seguir un estándar que todos los sitios deben respetar.
3. Debe existir una forma de crear estas representaciones y agregarlas a los sitios.
4. Deben existir términos y relaciones compartidas entre la información utilizada en las distintas páginas de un mismo dominio, formando un modelo de datos.
5. Debe existir una forma de crear estos términos y relaciones.

Una vez sabido esto, se debe proceder con el armado de un modelo de datos utilizando, en primer medida ontologías existentes y que apliquen al dominio tratado, de lo contrario se deberán crear tantos vocabularios como se lo requiera utilizando los estándares que la Web Semántica provee para ello. Luego se podrá proceder con la realización de aplicaciones con un modelo semántico subyacente que deberán integrarse con SPARQL para consultar y manipular los datos almacenados en un almacén de tripletas.

Para anotar los campos de las páginas XHTML se podrá utilizar RDFa, explicado en la Sección 3.2.4, sin que las mismas tengan un aspecto diferente a lo que hoy en día se aprecia.

> ¿Cuáles son las ontologías existentes que pueden reutilizarse en el modelado de la gestión de incidentes y de qué forma se las debe integrar?

Acorde a lo investigado previamente al desarrollo del prototipo, se encontró la ontología especificada en una tesis de postgrado [17] cuya dirección es `http://spi-fm.uca.es/spdef/models/genericTools/itm/1.0#`.

Sobre el final del desarrollo nos topamos con una ontología propuesta por **SEON**, como se detalla en el Capítulo 4, la cual contiene conceptos que podrían ser utilizados en un dominio de herramientas para el seguimiento de incidentes.

Una de las ventajas con la que juega la Web Semántica son las **URIs** como identificadores de recursos. Como se ha explicado, dan una gran ventaja comparado con otro tipo de identificadores, como los de los modelos relacionales. Como una URI es de acceso global, solo resta utilizarla. Ergo, la acción de integrar un vocabulario en una aplicación Web Semántica resulta sencilla.

> ¿Cuáles son las fortalezas, debilidades, oportunidades y desafíos vinculados a la utilización de tecnologías de la Web Semántica en la construcción de herramientas de soporte al cambio organizacional? En particular, ¿cómo se compara a los métodos tradicionales de modelado relacional y persistencia en bases de datos relacionales?

En este punto se detalla brevemente algunas de las ventajas y debilidades del modelo semántica respecto del modelo relacional. En el Capítulo 7 se lo vio más detallado.

Como este trabajo deja en claro, las corporaciones hoy en día están migrando sus aplicativos de soporte hacia modelos colaborativos en donde la integración de la información es crucial. Haciendo foco en un modelo relacional y pensando en que la información puede provenir de bancos de datos diferentes (por un lado un banco de datos de determinada área y por el otro un banco de datos de otro área diferente) la integración puede complicarse si los mismos no poseen un formato común o si no se utilizan plataformas compatibles. Por ejemplo, en un programa Java si consulto por una persona de tipo *Persona* con determinado identificador, requiero importar la estructura de dicho tipo, la cual a su vez debe ser facilitada. Esto sucede porque es un modelo rígido. Cambiando el foco hacia un modelo semántico la integración es más sencilla. Por empezar el identificador de la persona que deseamos obtener es una **URI**, accesible directamente a través de la red. En segundo lugar podríamos pensar en almacenar esa persona en nuestro servidor **SPARQL** y si días después se necesita actualizarla, simplemente se ejecuta la sentencia correspondiente en dicho servidor, indistintamente de la cantidad de propiedades de la cual la persona, en ese momento, disponga. Esta es una de las características que logran convertir a la Web Semántica en un modelo flexible.

Otra característica importante de la Web Semántica es la posibilidad de inferir nuevas aceptaciones. En un modelo relacional este concepto no existe. En cambio, este nuevo paradigma ofrece la posibilidad de ejecutar motores de inferencia; los cuales, por ejemplo se jactan de relaciones como *sameAs*

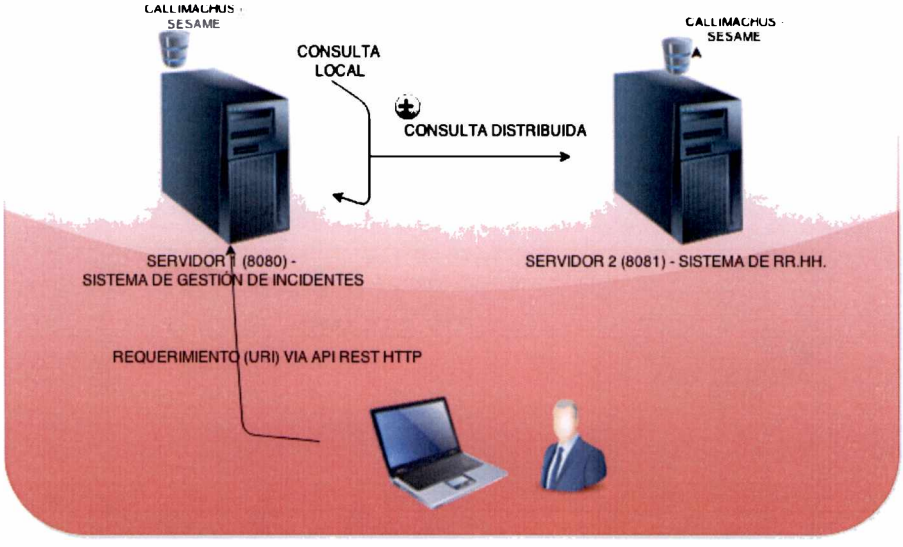


Figura 8.1: Visión general del manejo de datos del prototipo

de **OWL**; para que de esa manera se genere nueva información.

Además la Web Semántica permite realizar consultas SPARQL distribuidas pudiendo relacionar datos entre servidores diferentes. En el mundo relacional este concepto se ve un tanto distorsinado.

Entre otras ventajas que posee esta web, también se encuentran las desventajas. Como se ha mencionado en ciertas partes de este documento, la Web Semántica está en sus comienzos, aún tiene mucho por recorrer. Debe ser testeada y modificada de forma exhaustiva para que sea un estándar con pisada firme en los desarrollo de aplicaciones empresariales futuras.

El prototipo impulsado por esta tesis para tratar de responder las incógnitas planteadas, plasma algunas de las ventajas que se pueden adquirir al elegir un modelo semántico a la hora de desarrollar aplicaciones empresariales. En la Figura 8.1 se muestra el proceso de resolución de una consulta de un recurso en una hipotética organización.

8.3. Conclusiones sobre la Arquitectura Empleada

En la Figura 8.2 se puede apreciar como esta compuesta y dividida la arquitectura de la aplicación. En un primer nivel se encuentra el servidor, el cual contiene una instancia de Callimachus funcionando dentro de la cual se

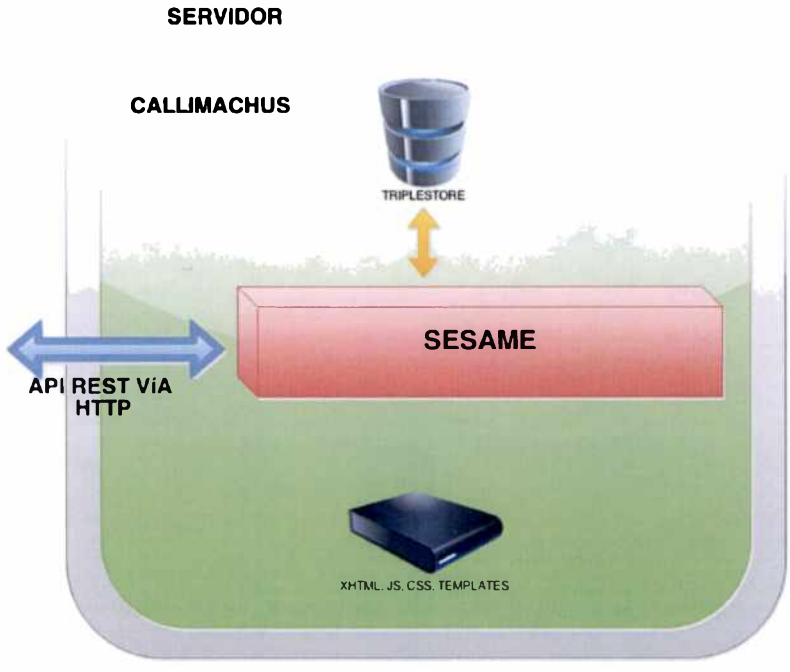


Figura 8.2: Visión en bloques de la arquitectura del prototipo

encuentra **Sesame** para interactuar con el almacén de tripletas. Además Se almacenan archivos para la visualización (**XHTML**, **JS**, etc.) persistidos en *memoria secundaria*[1].

A continuación se detallan las ventajas y desventajas encontradas en el framework que se utilizó como base para el desarrollo del prototipo:

8.3.1. Ventajas:

- Permite la creación de aplicaciones web que manejan datos semánticos en cuestión de semanas.
- Provee de un entorno sencillo y completo para la creación de recursos y su visualización basado en plantillas. Provee *scaffolding*[2].

[1] Disco rígido, por ejemplo.

[2] Es una técnica, soportada por los frameworks que respetan el patrón *MVC* (`https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf`), que genera de manera automática el código necesario para la lectura, creación, actualización y eliminación de los recursos implicados en el modelo.

- Se acerca a lo que es un **CMS**[3].
- Posee mecanismo para *pipelineing*[4] y transformaciones XML mediante **XSLT**, lo cual se torna una herramienta potente.

8.3.2. Desventajas:

- Se requiere un manejo excesivo de JavaScript con resoluciones engorrosas para ciertas situaciones como el manejo de datos entre diferentes sistemas residentes en distintos servidores.
- No se tiene demasiado control sobre el mecanismo de persistencia, complicando la personalización del mismo.
- No presenta una manera sencilla de establecer equivalencias entre dos clases.
- Se detallan ciertos problemas que tuvimos:

  1. ¿Cómo se indica que dos clases son equivalentes?: estuvimos investigando y realizando pruebas pertinentes a la incógnita. Primero que nada, se pueden linkear a través de *Equivalent* clases externas, pero solo al momento de crear una clase. Una vez que ya esta creada, no sabemos el motivo, pero Callimachus no lo permite, es decir no muestra el campo *Equivalent*. Los intentos realizados resultaron fallidos (drag and drop, etc.). Además aunque se puede indicar esta relación, no es utilizada al momento de inferir.

  2. ¿Cómo y dónde defino herencia de atributos y tipos?: la herencia se define mediante *Super* tanto al momento de la creación como de la edición. Se realizaron pruebas para ver el comportamiento de inferencia al momento de realizar una consulta, y se apreció que la inferencia no es aplicada implícitamente. Sesame soporta inferencia, el problema es que al estar en Callimachus nos encontramos en un nivel alto. Se probó una consulta con *@infer true* pero no se obtuvieron resultados positivos.

  3. ¿Callimachus siempre guarda en archivos las cosas o se persisten en un almacén de tripletas?: Callimachus almacena la definición e instancias de de las clases en un almacén de tripletas, denominado

[3]Content Manager Service.

[4]Consta de un conjunto de elementos, conectados en serie, que procesan datos en donde la salida de un elemento es la entrada al siguiente

SESAME, identificadas mediante un nombre único y global, dado por el usuario al momento de crear el recurso, y que formará parte de la URI para accederlo mediante la **API REST**[5] que presenta.

8.3.3. Experiencia de usuario

Callimachus tiene una curva de aprendizaje particular. Al principio se aprende rápido y fácil, logrando grandes avances, pero luego, esa rapidez y sencillez disminuye. Esa disminución se debe principalmente a la falta de documentación clara y a la simplicidad de los tutoriales de Callimachus que hay en la web. Luego de el período de aprendizaje, Callimachus es muy útil y potente para desarrollar aplicaciones orientadas a datos, facilitando el trabajo de los desarrolladores frente a otros frameworks.

En cuanto al usuario final, la experiencia de usuario depende pura y exclusivamente de las vistas creadas por el desarrollador. De todas formas, gracias al potencial que provee Bootstrap, es bastante simple realizar un interface amigable y fácil de utilizar.

8.4. Contribuciones

Se listan las contribuciones realizadas por esta investigación:

- Conocimiento interno acerca del framework Callimachus para todos aquellos proyectos que lo utilicen.
- Además de evidenciar, con un prototipo de sistema, cómo utilizar efectivamente las tecnologías de la Web Semántica para la construcción de un gestor de incidentes, este trabajo da respuesta a las siguientes preguntas:
    1. Para el caso particular de un sistema de gestión de incidencias, ¿qué cambios implica la adopción de las propuestas de la Web Semántica (en términos de arquitectura de software y del proceso de desarrollo), respecto de las metodologías tradicionales basadas en modelos relacionales?.
    2. ¿Cuáles son las ontologías existentes que pueden reutilizarse en el modelado de la gestión de incidentes y de qué forma se las debe integrar?

[5] `http://callimachusproject.org/docs/1.4/callimachus-reference.docbook?view\#Callimachus_REST_API`.

3. ¿Cuáles son las fortalezas, debilidades, oportunidades y desafíos vinculados a la utilización de tecnologías de la Web Semántica en la construcción de herramientas de soporte al cambio organizacional? En particular, ¿cómo se compara a los métodos tradicionales de modelado relacional y persistencia en bases de datos relacionales?

8.5. Trabajos Futuros

Luego de realizada la corriente investigación se detallan, para lograr extender la misma, ciertas posibles mejoras no realizadas por falta de tiempo y otros motivos.

- Existe la posibilidad de que el vocabulario de **SEON** para gestores de incidentes, encontrado sobre el final, se incorpore al modelo especificado en esta investigación. Por ejemplo, la utilización de la propiedad ''`http://se-on.org/ontologies/domain-specific/2012/02/issues.owl#dependsOnIssue`'' podría ser posible.

- El prototipo desarrollado no utiliza *inferencia* ya que la versión utilizada del *framework Callimachus* no proveía un mecanismo transparente. Agregar esta capacidad incrementaría el potencial de la aplicación.

- En este trabajo se presentó un prototipo sencillo y básico con la intención de responder a las incógnitas surgidas. Se podría realizar una implementación de la aplicación utilizando un modelo más completo para que de esta manera se lograse definitivamente el desarrollo de un aplicativo, para la gestión de incidentes semántico y genérico, utilizable. Por ejemplo se podrían agregar las siguientes propiedades:

  Fix version (solo para incidentes de software): número de versión del software en la que el incidente estará resuelto.

  Components: parte/s del objeto u organización afectadas. Ej. Área de sistemas o área contable.

  Labels: etiquetas que categorizan al incidente y facilitan la búsqueda e indexación.

  History: listado de actividades (actualizaciones de campos, nuevos comentarios, etc.) que se realizaron sobre el incidente desde su creación.

Files: archivos que contienen información relevante que se incorporan para facilitar el seguimiento y tratamiento del incidente.

Parent task: en algunos casos, puede ser que el incidente a tratar sea una parte pequeña de un incidente más grande. Este campo contiene una referencia a dicho Incidente.

Subtasks: en el caso que el incidente pueda descomponerse en incidentes de menor tamaño para facilitar su tratamiento, se guarda una referencia a esos sub-incidentes.

People involved: personas involucradas, directa o indirectamente, durante todo el proceso de resolución del incidente. Por ejemplo, las personas que están afectadas por trabajar en el área sobre la cual esta relacionada el incidente, aquellas personas que estuvieron en el tratamiento de incidentes antiguos relacionados con este, etc.

Processes involved: proceso/s de negocio afectados de la organización, directa o indirectamente, por el incidente.

Issue Associated by specific criteria: incidentes que se relacionen con el incidente actual, ya sea por compartir áreas afectadas, personas de la organización o procesos de negocio involucrados.

Bibliografía

[1] A Technical Introduction to XML. http://www.xml.com/pub/a/98/10/guide0.html?page=2. Visto por última vez: Marzo 2015.

[2] AliBaba. https://bitbucket.org/openrdf/alibaba. Visto por última vez: Marzo 2015.

[3] Bootstrap. http://getbootstrap.com/. Visto por última vez: Marzo 2015.

[4] Bugzilla. https://www.bugzilla.org/features/. Visto por última vez: Marzo 2015.

[5] Callimachus. http://www.callimachusproject.org. Visto por última vez: Marzo 2015.

[6] Corporate Semantic Web. http://www.corporate-semantic-web.de/. Visto por última vez: Marzo 2015.

[7] GitHub Issue Tracker. https://github.com/blog/411-github-issue-tracker. Visto por última vez: Marzo 2015.

[8] Hacia la Web Semántica. http://www.hipertexto.info/documentos/web_semantica.htm. Visto por última vez: Marzo 2015.

[9] HTML and CSS. http://www.w3.org/standards/webdesign/htmlcss. Visto por última vez: Marzo 2015.

[10] Introducing JSON. http://json.org/. Visto por última vez: Marzo 2015.

[11] JIRA Documentation. https://confluence.atlassian.com/display/JIRA/JIRA+Documentation. Visto por última vez: Marzo 2015.

[12] Managing Organizational Change. http://www.inc.com/encyclopedia/managing-organizational-change.html. Visto por última vez: Marzo 2015.

[13] Mantis Bug Tracker Administration Guide. http://www.mantisbt.org/docs/master-1.2.x/en/administration_guide.pdf. Visto por última vez: Marzo 2015.

[14] Number of Tiers (N-tier). http://www.techopedia.com/definition/16194/number-of-tiers-n-tier. Visto por última vez: Marzo 2015.

[15] Pubby - A Linked Data Frontend for SPARQL Endpoints. http://wifo5-03.informatik.uni-mannheim.de/pubby/. Visto por última vez: Marzo 2015.

[16] Request/Response Pattern. http://www.servicedesignpatterns.com/ClientServiceInteractions/RequestResponse. Visto por última vez: Marzo 2015.

[17] Semantic Issue Tracker Paper. http://project.cim3.net/wiki/Semantic_Issue_Tracker_Paper. Visto por última vez: Marzo 2015.

[18] Semantic MediaWiki. https://semantic-mediawiki.org/. Visto por última vez: Marzo 2015.

[19] Using Mantis Bug Tracker. http://www.suitable.com/docs/mantisbt.html. Visto por última vez: Marzo 2015.

[20] What is the difference between rdf-, triple and quad store? http://fadyart.com/en/index.php?option=com_content&view=article&id=48:what-is-the-difference-between-rdf-triple-and-quad-store&catid=35:semantic-data-stores. Visto por última vez: Marzo 2015.

[21] Consejos para escribir enunciados SQL portables, 2003. http://www.lacorona.com.mx/fortiz/adodb/tips_portable_sql-es.htm. Visto por última vez: Marzo 2015.

[22] Down with the Data Warehouse! Long Live the Semantic Data Warehouse!, 2011. http://www.dataversity.net/down-with-the-data-warehouse-long-live-the-semantic-data-warehouse/. Visto por última vez: Marzo 2015.

[23] Making the Argument for Semantic Technologies, 2011. http://www.mkbergman.com/974/making-the-argument-for-semantic-technologies/. Visto por última vez: Marzo 2015.

[24] Sesame, 2014. http://rdf4j.org/sesame/2.7/docs/users.docbook?view. Visto por última vez: Marzo 2015.

[25] The SAIL API, 2014. http://rdf4j.org/sesame/2.7/docs/system.docbook?view#The_SAIL_API. Visto por última vez: Marzo 2015.

[26] SEON, 2313. http://www.se-on.org/. Visto por última vez: Marzo 2015.

[27] Arto. How RDF Databases Differ from Other NoSQL Solutions, 2010. http://blog.datagraph.org/2010/04/rdf-nosql-diff. Visto por última vez: Marzo 2015.

[28] Dan Brickley and Libby Miller. Schema. http://schema.org/. Visto por última vez: Marzo 2015.

[29] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99, 2014. http://xmlns.com/foaf/spec/. Visto por última vez: Marzo 2015.

[30] Semantics Cambridge. SPARQL vs. SQL - Intro, 2015. http://www.cambridgesemantics.com/semantic-university/sparql-vs-sql-intro. Visto por última vez: Marzo 2015.

[31] Oberle Daniel. How Ontologies Benefit Enterprise Applications. http://www.semantic-web-journal.net/system/files/swj212_2.pdf. Visto por última vez: Marzo 2015., 2009.

[32] Eugene Eric Kim. An Introduction to Purple, 2001.

[33] Emilio Ing. Lorenzon. *Sistemas y Organizaciones.* 2011.

[34] Polikoff Irene. Comparing SPARQL with SQL, 2014. http://www.topquadrant.com/2014/05/05/comparing-sparql-with-sql/. Visto por última vez: Marzo 2015.

[35] Sporny Manu Herman Ivan, Adida Ben, and Birbeck Mark. RDFa 1.1 Primer - Second Edition. W3c note, W3C, August 2013.

[36] JCC Consulting Inc. SQL Standards, 2013. http://www.jcc.com/resources/sql-standards. Visto por última vez: Marzo 2015.

[37] Heflin Jeff. OWL Web Ontology Language Use Cases and Requirements, 2004.

[38] Yannakopoulos John. HyperText Transfer Protocol: A Short Course. http://condor.depaul.edu/dmumaugh/readings/handouts/SE435/HTTP/http.pdf. Visto por última vez: Marzo 2015.

[39] Fowler Martin. *Patterns of Enterprise Application Architecture*, volume 48 of *The Addison-Wesley signature series*. Addison-Wesley Professional, 2002.

[40] Kelly Mike. JSON Hypertext Application Language, 2013. http://tools.ietf.org/html/draft-kelly-json-hal-06. Visto por última vez: Marzo 2015.

[41] Alistair Miles of STFC Rutherford Appleton Laboratory University of Oxford and Sean Bechhofer of University of Manchester. SKOS Simple Knowledge Organization System Namespace Document HTML Variant, 2009. http://www.w3.org/2004/02/skos/core. Visto por última vez: Marzo 2015.

[42] Ramirez Ariel Ortiz. Three-Tier Architecture, 2000. http://www.linuxjournal.com/article/3508. Visto por última vez: Marzo 2015.

[43] Fillottrani Pablo R. Fundamentos de la Web Semántica: Aplicaciones RDF, 2013. http://www.cs.uns.edu.ar/~prf/teaching/FSW13/downloads/rdfApp.pdf. Visto por última vez: Marzo 2015.

[44] Guha Ramanathan and Dan Brickley. RDF Schema 1.1. Technical report, W3C, 2014.

[45] Quetglas Patricia Riera. *Memoria Final*. PhD thesis, Universitat Oberta de Catalunya. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/18966/8/prieraqTFC0113memoria.pdf. Visto por última vez: Marzo 2015.

[46] Rube Iván Ruiz. Un framework para el despliegue y evaluación de procesos software. 2013.

[47] Foundation The Apache Software. Apache Jena, 2015. https://jena.apache.org/. Visto por última vez: Marzo 2015.

[48] Staab Steffen. The Semantic Web Revisited. http://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf. Visto por última vez: Marzo 2015., 2006.

[49] Susie Stephens. The Enterprise Semantic Web - Technologies and Applications for the Real World.

[50] Berners-Lee T. Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web. *Network Working Group , Category: Informational, June 1994*, 1994.

[51] Berners-Lee Tim. Linked Data - Design Issues, 2006.

[52] TopQuadrant. Semantic Web Solutions at Work in the Enterprise. http://www.topquadrant.com/docs/marcom/TopQuadrant_Whitepaper_online.pdf. Visto por última vez: Marzo 2015.

[53] W3C. Semantic Web: Data on the Web, 2006. http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/#(19). Visto por última vez: Marzo 2015.

[54] W3C. SPARQL, 2013. http://www.w3.org/TR/sparql11-query/. Visto por última vez: Marzo 2015.

[55] Liyang Yu. *A Developer's Guide to the Semantic Web.* 2011.