Universidad de las Ciencias Informáticas

Facultad No. 5

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

DISEÑO ARQUITECTÓNICO PARA LA BÚSQUEDA SEMÁNTICA EN LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Autor: *Pedro Oramas Peña*

Tutor(es): *Ing. Liliana del Carmen Castellanos Montero*

Ing. Michel David Suarez

La Habana, mayo 28 de 2012

"Año del 54 aniversario del Triunfo de la Revolución"

Declaración de Autoría

Declaro que soy autor del trabajo de diploma "**Diseño arquitectónico para la búsqueda semántica en la Universidad de las Ciencias Informáticas**" y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma a los __ días del mes de _____ del año _____.

Autor

_____
Pedro Oramas Peña

Tutores

_____ _____
Ing. Liliana del Carmen Castellanos Montero Ing. Michel David Suárez

Resumen

"Diseño arquitectónico para la búsqueda semántica en la Universidad de las Ciencias Informáticas" surge con la idea de obtener una interfaz única para relacionar la información por su significado a través de la semántica y obtener conocimiento sobre datos heterogéneos. Implica el estudio de la capa de dominio de usuario de la plataforma LarKC, que permite el razonamiento sobre millones de datos y la primera concepción de una arquitectura para introducir semántica en los procesos sustantivos de la Universidad, tomando como punto de partida el proceso de expediente académico. Se realiza el diseño de la vista lógica, de despliegue e implementación en conjunto con el desarrollo de un prototipo funcional de prueba, que luego fue validado mediante el método MECABIC.

Palabras clave: flujo de trabajo, componentes, semántica, LarKC.

Contenido

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

La forma de acceder y recuperar la información en la historia ha sufrido numerosos cambios. Las bibliotecas en un principio fueron la principal vía para acceder a ella, teniendo como función principal facilitar a los usuarios el acceso a través de libros, revistas y otros documentos mediante diferentes sistemas de recuperación y distribución. Debido al crecimiento alcanzado en la Industria de la información y al avance tecnológico, la información ya se encuentra de forma digital permitiendo su procesamiento, almacenamiento y distribución, brindando la posibilidad de poder relacionarla y recuperarla descubriendo conocimiento.

El volumen de información que abarca la red es inimaginable. En ella existen numerosas fuentes de datos que aumentan su tamaño cada día debido al desarrollo científico que existe. Algunas de estas fuentes se pueden encontrar en sitios web, *blogs*, sistemas de gestión, repositorios, entre otras, conteniendo información de forma desordenada y no estructurada. Los sistemas de búsquedas y recuperación no manejan las ambigüedades en los criterios de búsquedas introducidos obteniendo como resultado, documentos cuyo contenido no guarda relación alguna con lo que se quiere encontrar. Estos sistemas no tienen en cuenta la preferencia del usuario desplazando los resultados más relevantes a las últimas posiciones.

El contexto internacional se pronuncia cada vez más en función de una nueva alternativa que permita relacionar la información por significado para descubrir conocimiento, esto se conoce como web semántica.

En la web semántica los datos se encuentran enlazados de forma inteligente utilizando ontologías[1] como mecanismo para estructurar la información. También, permite búsquedas más refinadas con la finalidad de encontrar resultados más relevantes en relación con lo que demanda el usuario.

[1]Es una especificación de una conceptualización. Se encarga de definir los términos para describir y representar un área de conocimiento; incluye conceptos básicos en un campo determinado y establece sus relaciones de forma que las computadoras pueden codificar el conocimiento y el conocimiento extendido haciéndolo reutilizable [17].

En las instituciones o empresas existe una propagación de información que en ocasiones hace imposible su recuperación por parte de quienes la necesitan en el momento oportuno. Dichos datos en su mayoría son importantes y generan grandes flujos de información y conocimiento manifestándose como un recurso intangible.

La Universidad de las Ciencias Informáticas (UCI) desde sus inicios se ha caracterizado por tener una producción creciente de información, resultado del cúmulo de aplicaciones que participan en sus procesos sustantivos. En los últimos cursos académicos la red universitaria ha acogido múltiples sistemas que permiten transmitir información haciendo accesible documentos de tesis, publicaciones de investigaciones, repositorios de código fuente, herramientas para la gestión de proyectos y académica, entre otras. El proceso de búsqueda en estos sistemas se hace difícil porque:

- Los motores de búsqueda existentes no conocen el significado de los criterios que se introducen.
- La información recuperada; mediante un buscador, presenta imprecisiones cuando una palabra tiene varios significados y cuando varias palabras significan lo mismo.
- Los datos son inutilizados y poco manejables por los ordenadores cuando existe una ausencia de supervisión humana.
- Los usuarios casi siempre realizan la búsqueda con lenguaje natural haciendo poco uso de las palabras clave que conoce el motor de búsqueda.

Todo esto trae consigo la insatisfacción del usuario como consecuencia de las respuestas inexactas dadas por los sistemas de recuperación. Por este motivo se ha planteado el siguiente **problema de investigación**: ¿Cómo relacionar la información por significado para descubrir conocimiento produciendo mejores resultados en las búsquedas?

El **objeto de estudio** en el que está centrado la investigación es: el proceso de descubrir conocimiento enmarcándose en el **campo de acción**: el diseño de arquitecturas de *software* basadas en componentes para descubrir conocimiento. Como **objetivo general** se plantea

diseñar una arquitectura basada en componentes de recuperación de información por significado en la Universidad de las Ciencias Informáticas.

Para dar cumplimento al objetivo general se plantearon las siguientes **tareas investigativas**:

- *Caracterización* de las principales tendencias sobre el proceso de descubrir conocimiento sobre datos heterogéneos.
- *Determinación* de las principales arquitecturas que pueden ser utilizadas en la propuesta.
- *Diseño* de la arquitectura de software de la c*apa de dominio de usuario* para la plataforma LarKC.
- *Evaluación* del prototipo de la arquitectura.

Como **idea a defender** se plantea diseñar una arquitectura basada en componentes de recuperación de información por significado para mejorar el proceso de búsquedas en la Universidad de las Ciencias Informáticas.

Toda investigación se rige por una metodología acorde a su objeto de estudio. Para el presente trabajo se emplea la combinación dialéctica de los **métodos teóricos** y **empíricos**.

Métodos teóricos

Analítico-Sintético, el cual facilitó identificar, analizar y seleccionar los conceptos y definiciones más importantes relacionadas con la presente investigación.

Análisis Histórico-Lógico, el cual permitió determinar la evolución y desarrollo hasta la actualidad de las arquitecturas de *software* para de esta forma determinar las tendencias actuales del *software*.

Métodos empíricos

Modelado, mediante el cual se crean abstracciones sobre las arquitecturas con vistas a explicar la realidad, operar de forma práctica o teórica con un objeto, no en forma directa, sino utilizando cierto sistema intermedio, auxiliar, natural o artificial.

Entrevista, mediante el cual se hace una recolección de información a través de formularios, en los cuales tienen aplicación aquellos problemas que se pueden investigar por métodos de observación, análisis de fuentes documentales y demás sistemas de conocimientos.

Estructura de capítulos:

El contenido a desarrollar en el presente trabajo está estructurado en tres capítulos:

Capítulo No. 1: *Fundamentación teórica*

Se definirán un conjunto de conceptos y definiciones que permitirán obtener una panorámica del contexto de la investigación, incluye un estudio sobre el estado del arte del tema que se aborda para conformar un sólido basamento teórico.

Capítulo No. 2: Propuesta del diseño arquitectónico.

Expone la intención de la propuesta, definiendo los elementos técnicos de ella: los patrones, el diseño de clases, los medios empleados, el modelo de despliegue de la arquitectura, las restricciones de diseño así como otros elementos.

Capítulo No. 3: Evaluación y aceptación de la arquitectura

Evaluación basada en demostración que explica cómo la arquitectura debe dar garantías de que la solución diseñada es realizable de acuerdo con los atributos de calidad.

CAPÍTULO 1

Los datos, la información y el conocimiento son términos que están estrechamente relacionados en el proceso de descubrir conocimiento. La tarea de concebir un software –al menos en su diseño arquitectónico– que maneje los elementos de ese proceso no resulta fácil, es por ello que a continuación se realiza un estudio de las principales valoraciones de diversos autores, con el fin de lograr el basamento teórico necesario para diseñar tal arquitectura de software.

Como requisito básico para comprender el trabajo se hace necesario revisar el concepto que le da inicio al proceso.

1.1 Información

Múltiples autores de se han dedicado al estudio de la información y su importancia como recurso en la sociedad que ha alcanzado un gran desarrollo rebasando cualquier propósito realizado años atrás. Su industria se ha convertido en un factor esencial para la sociedad moderna. Jeremy Campbell comentó que no fue hasta los años 40 del siglo XX, que se definió el término desde una perspectiva científica, [1] en el contexto de la era de la comunicación electrónica.

Algunos especialistas definen la información desde el punto de vista de su existencia como fenómeno “en sí”, pero que a la vez, es inseparable de la conciencia humana – información “para sí”; otros desde su perspectiva cognitiva, la describen como parte de la mente de las personas en forma de configuraciones de las cosas y por otro lado, existen quienes ven la información como proceso social al emplearse de modo consciente y planificado para informar o informarse [2]. Estos enfoques concuerdan con la primera y más simple de las acepciones registradas en el diccionario de la Real Academia de la lengua Española es la “acción y efecto de informar o informarse” [3].

La información en sí misma, como la palabra, es al mismo tiempo significado y significante, este último es el soporte material o simbología que registra o encierra el significado y el contenido [2]. Puede considerarse que la información transita por dos estados o momentos: el

primero cuando la mente humana asimila, procesa e interpreta, es decir, la transforma en conocimiento; el cual según Páez Urdaneta consiste en un conjunto de "estructuras informacionales que al internalizarse se integran a los sistemas de relacionamiento simbólico de más alto nivel y permanencia" [4] y el segundo cuando se registra en forma documental, que actúa como fuente de información mediante lenguaje. Una fuente de información no es más que cualquier objeto o sujeto que genere, contenga, suministre o transfiera otra fuente de información [5].

La siguiente definición puede sintetizar lo abordado anteriormente: "La información puede entenderse como la significación que adquieren los datos como resultado de un proceso consciente e intencional de adecuación de tres elementos: los datos del entorno, los propósitos y el contexto de aplicación así como la estructura del conocimiento" [6].

Para obtener información, se necesita extraer datos distribuidos en múltiples fuentes, heterogéneas entre sí, combinándolos para facilitar al usuario una vista unificada de esos datos distribuidos, conociéndose ese proceso *integración de datos*[2]. Existen múltiples ejemplos de la importancia que tiene la *integración de datos* en la vida diaria en todos los ámbitos. A continuación se muestran dos de ellos:

- La cantidad de fuentes de información que se encuentra en la web es cada vez más extensa y más heterogénea. Aquí, el problema de la integración de la información, está presente en cada consulta que realiza el usuario.

- La cantidad de empresas que nacen como unión de empresas más pequeñas, tienen la necesidad de integrar y compartir mucha información procedente de numerosas fuentes de datos diferentes [7].

[2] Es una de las principales áreas tanto de investigación como de negocio, que se encarga de permitir a los usuarios acceder a datos almacenados en fuentes de datos heterogéneas, presentando una única vista unificada de esos datos, de forma que el usuario no llegue a percibir esa heterogeneidad [50].

En la investigación, el término información, es un conjunto de datos relacionados por el humano para construir su conocimiento. Es un fenómeno que da significado o sentido a las cosas que indica los modelos del pensamiento humano mediante códigos o conjuntos de datos. Con la nueva era tecnológica crece cada vez más debido a su fácil creación, por lo que los enormes volúmenes de información digital que existen son cada vez más complejos. Esto ha provocado el surgimiento de nuevas técnicas y mecanismos más potentes que faciliten su acceso, recuperación, análisis y procesamiento.

1.2 Conocimiento

El conocimiento no es más que un conjunto de información almacenada mediante la experiencia y el aprendizaje (a posteriori), o a través de la introspección (a priori) [8]. Es la interrelación que existe entre los datos que se perciben mediante los sentidos que permite tomar decisiones para realizar las acciones cotidianas. La ciencia considera que, para alcanzarlo, es necesario seguir un método. El conocimiento científico no solo debe ser válido y consistente desde el punto de vista lógico, sino que también debe ser probado mediante el método científico o experimenta [8].

Según Andreu y Sieber [2000], el conocimiento tenía tres características fundamentales:

- El conocimiento es personal, en el sentido de que se origina y reside en las personas, que lo asimilan como resultado de su propia experiencia (es decir, de su propio "hacer", ya sea físico o intelectual) y lo incorporan a su acervo personal estando "convencidas" de su significado e implicaciones, articulándolo como un todo organizado que da estructura y significado a sus distintas "piezas".

- Su utilización, que puede repetirse sin que el conocimiento "se consuma" como ocurre con otros bienes físicos, permite "entender" los fenómenos que las personas perciben (cada una "a su manera", de acuerdo precisamente con lo que su conocimiento implica en un momento determinado) y también "evaluarlos", en el sentido de juzgar la bondad o conveniencia de los mismos para cada una en cada momento.

- Sirve de guía para la acción de las personas, en el sentido de decidir qué hacer en cada momento porque esa acción tiene en general por objetivo mejorar las consecuencias, para cada individuo, de los fenómenos percibidos (incluso cambiándolos si es posible) [9].

La relación indisoluble que se establece entre la información, el conocimiento, el pensamiento y el lenguaje se explica a partir de comprender que la información es la forma de liberar el conocimiento que genera el pensamiento humano. Dicha liberación se produce mediante el lenguaje (oral, escrito, gesticular, entre otros), un sistema de señales y símbolos que se comunican de alguna manera [10].

Para la investigación se puede concluir que el conocimiento no es más que los datos o información adquiridos por una persona a través del transcurso de su vida, esto no significa que tenga que ser cierto o verdadero. En la actualidad, el conocimiento más valorado por la sociedad es el científico ya que debe ser válido y consistente desde el punto de vista lógico y debe ser probado mediante el método científico o experimental.

Para lograr un conocimiento es necesario el uso de la semántica que da significado a las palabras, es por esto que a continuación se realiza un estudio del término.

1.3 Semántica

La semántica proviene de un vocablo griego que puede traducirse como “significativo”. Se refiere al significado de las palabras y al estudio del significado de símbolos lingüísticos y por ende las combinaciones entre ellos [11]. Se vincula al significado, sentido e interpretación de palabras, expresiones o símbolos.

La semántica trata el significado de los datos, y a diferencia de la sintaxis, no solo tiene la adecuación de los datos y la estructura en la que se encuentran guardados. Es la interpretación que las personas atribuyen a los datos, de acuerdo con su entendimiento del mundo real. En el área de las bases de datos, la semántica se refiere a la interpretación que las personas dan a los datos y a los atributos que referencian esos datos dependiendo del contexto [12]. La *integración semántica* es el área de investigación que se centra en contrastar

y decidir sobre la similitud de los datos provenientes de distintas fuentes, usando herramientas basadas en la semántica [13].

No solo conocer el significado de la semántica resulta de importancia, sino su manejo en la actualidad en diversas áreas destacándose entre ellas la web.

La web semántica

A la web semántica, se le denomina también 'web de los datos' ya que hace posible que los usuarios en Internet puedan encontrar respuestas a sus preguntas de forma más rápida y sencilla debido a que la información está mejor definida, o lo que es lo mismo, dotada de mayor significado. Se basa principalmente en añadir metadatos semánticos y ontológicos a la World Wide Web [14]. Consiste en realizarle mejoras a la web ampliando su interoperabilidad entre sistemas y reduciendo los operadores humanos. Es una red de información relacionada con el objetivo de ser procesable por los ordenadores a escala global.

Tim Berners-Lee su precursor, intentó desde el principio incluir informaciones semánticas en la creación de la web, pero no fue posible. Años después introdujo la semántica para recuperar esta omisión y satisfacer las expectativas de los usuarios que requieren respuestas más sencillas.

Componentes de la web semántica

La Web semántica tiene como principales componentes a los metalenguajes[3] y estándares de representación *XML (*Extensible Markup Language*)*, *XMLSchema (*Extensible Markup Language Schema*)*, *RDF (Resource Description Framework)*, *RDFSchema (Resource Description Framework Schema)* y *OWL* (Ontology Web Language). La *"OWL Web Ontology Language Overview"* describe la función y relación de cada uno de estos componentes de la Web Semántica:

[3] Son lenguajes como XML y HTML que sirven para definir otros lenguajes.

- ***XML*** aporta la sintaxis para los documentos estructurados, pero sin contener restricciones sobre el significado.
- ***XMLSchema*** es un lenguaje para restringir la estructura de los documentos XML y extiende de él.
- ***RDF*** es un modelo de datos para los recursos y las relaciones entre ellos. Introduce una semántica básica para este modelo de datos que puede representarse mediante XML.
- ***RDFSchema*** es un vocabulario para describir las propiedades y las clases de los recursos RDF, con una semántica para establecer jerarquías de generalización entre dichas propiedades y clases.
- ***OWL*** añade más vocabulario para describir propiedades y clases: tales como relaciones entre clases, cardinalidad[4], igualdad, tipologías de propiedades más complejas, caracterización de propiedades o clases enumeradas [15].

Según Juan Pablo Palacios son lenguajes universales de la web semántica, que sirven para etiquetar la información semántica, mediante los cuales, los dispositivos no solo son capaces de transmitir datos sino también de entenderlos. Estos lenguajes, extienden XML para incorporar aspectos semánticos que lo doten de mayor capacidad expresiva, para permitir inferir conocimiento y lo más importante, para ser entendidos por las máquinas (además de por los humanos) [16].

El aprovechamiento de la web y su usabilidad aumentará gracias a que los documentos estarán etiquetados con información semántica para ser interpretados por los ordenadores. Incluye vocabularios de metadatos descriptivos que permiten a quienes elaboran los

[4] Número mínimo y máximo de ocurrencias de una entidad que pueden ser relacionadas a una ocurrencia de otra entidad por medio de un relacionamiento dado.

documentos disponer de una noción de cómo deben ser etiquetados para que los agentes automáticos puedan realizar sus tareas de razonamiento [15].

De la información al conocimiento mediante la semántica

La técnica más usada para convertir la información en conocimiento es mediante el desarrollo de estructuras formalizadas llamadas ontologías. Según Gruber, son una especificación de una conceptualización en un marco común o una estructura conceptual sistematizada y de consenso, no solo para almacenar la información, sino también para poder buscarla y recuperarla [17]. Una ontología especifica términos y relaciones elementales para la compresión de un área del conocimiento, así como reglas para poder combinar los términos para definir las extensiones de este tipo de vocabulario controlado.

Otras técnicas como la extracción de texto sirven en el contexto de la semántica, sin embargo, estas técnicas deben introducirse dentro de herramientas que permita realizar el proceso de descubrir conocimiento

1.4 Sistemas para razonamiento de datos a escala web

El contexto internacional se pronuncia cada vez más en función de una nueva alternativa que permita relacionar la información por significado para descubrir conocimiento debido a que el internet está lleno de información heterogénea y desestructurada que necesita que a través de técnicas y mecanismos provean una cómoda localización, facilitando las búsquedas de los usuarios. Una de las técnicas más usada para esto, es el procesamiento distribuido que propone ejecutar varias instancias de un razonador. Existen varios artículos en donde se considera y se evalúa esta opción, ya que se afirma que una sola máquina no es suficiente para procesar tal cantidad de datos y que por el contrario una solución basada en un sistema distribuido es potencialmente más escalable. Se consideran tres enfoques actuales de razonamiento paralelo para datos a escala web, como son los sistemas LarKC, MaRVIN y Reasoning-Hadoop [18].

Reasoning-Hadoop

Es un framework que permite el procesamiento distribuido de grandes conjuntos de datos mediante un conjunto de equipos basados en un modelo de programación simple. Implementa un paradigma computacional llamado MapReduce, donde la aplicación se divide en pequeños fragmentos de trabajo mapper y reduce y cada uno de los cuales se pueden ejecutar o volver a ejecutar en cualquier nodo del clúster. Manipula los datos de varios servidores para realizar procesamiento paralelo y masivo. Se utiliza para el razonamiento de RDFS/OWL mediante MapReduce. Además, proporciona un sistema de archivos distribuido que almacena los datos en los nodos de cómputo, produciendo un alto ancho de banda agregado en todo el clúster. Ambos, MapReduce y el sistema de archivos distribuidos, están diseñados de manera que las fallas de nodo se gestionan automáticamente mediante él [19].

Hadoop no es adecuado para cargas de trabajo OLTP (OnLine Transaction Processing) y tampoco para procesamiento analítico en línea o para sistemas de toma de decisiones donde se acceden a los datos de forma secuencial en datos estructurados como bases de datos relacionales.

MaRVIN

Massive RDF Versatile Inference Network, es una plataforma paralela y distribuida para el procesamiento de grandes cantidades de datos RDF. Utiliza una arquitectura peer-to-peer para lograr la escalabilidad masiva mediante la adición de los recursos computacionales a través del principio divide-vencerás. Marvin garantiza la integridad final del proceso de inferencia y produce sus resultados en forma gradual. Gracias a su diseño modular y su instrumentación integrada proporciona una plataforma de experimentación versátil con muchas configuraciones [20].

LarKC

El Gran Colisionador de Conocimiento por sus siglas en inglés, LarKC más conocido por lark, es una plataforma para el razonamiento incompleto distribuido, que soporta inferencia a gran escala sobre millones de datos estructurados en fuentes heterogéneas, removiendo barreras

de escalabilidad perteneciente a los motores de razonamiento actuales [21]. Presenta varias funcionalidades esenciales para el funcionamiento de la plataforma en las que se encuentra:

- ***Razonamiento***

Los investigadores en su mayoría han desarrollado métodos de razonamiento en dominios estáticos, haciéndole frente a los axiomas, (definiciones de conceptos), pero la escalabilidad es deficiente para grandes volúmenes de datos. Los motores de programación lógica lidian con reglas similares, pero representan solo conclusiones simples. Ambos son una interesante área de investigación y temas como su combinación, representan una de las áreas más interesantes en el razonamiento de datos. Sin embargo, existe un abismo muy grande en cuanto a escalabilidad en el razonamiento web y algoritmos de razonamiento eficientes sobre dominios restringidos.

Razonamiento y la búsqueda Web: la única forma de enlazar el raciocinio y la web es vinculando el proceso de razonamiento con el de establecer los hechos y axiomas relevantes a través de la recuperación (selección) y abstracción (comprendiendo y transformando información.

Precisión / Recuperación: LarKC llena uno de los abismos existentes en cuanto a los métodos de indexación en las búsquedas y el razonamiento. El método convencional usado en las tareas de búsqueda en términos de precisión y recuperación tienen una gran precisión y poca recuperación o viceversa. Para un buen razonamiento se necesita un 100% de precisión y un 100% de recuperación [22]. En la práctica este tipo de perfección tiene altos costos de instalación (la codificación de experiencia en el campo en una ontología y semánticamente la estructuración del espacio de información completa) y computación. El Gran Colisionador de conocimiento explota el espacio de posibilidades por encima de la curva de precisión[5] /

[5] Mide el número de elementos correctamente identificados como un porcentaje del número de elementos identificados o sea la mejor precisión es la medida en que un sistema asegura que sus datos son correctamente identificados [22].

recuperación[6]. Gráficamente, el proyecto se posiciona en el espacio entre la búsqueda tradicional y el razonamiento tradicional.

A continuación se muestra una figura que refleja una comparación del estado del arte de las plataformas de razonamiento existentes con respecto a LarKC la cual expresa una notable superioridad de razonamiento del gran Colisionador.

Características	Estado del Arte	LarKC
Método de Razonamiento	Solamente la lógica	Múltiples Métodos
Heurísticas	Hardwired	Plug-ins configurables
Precisión/ recuperación	100/100	Configurable trade-off
Axiomas dinámicos y hechos	No soporta	Soporta
RDF escala de recuperación	O(10^9)	O(10^12)
RDF escala de inferencia	O(10^8)	O(10^10)
Comportamiento	Ninguno	Configurable
Paralelismo	Ninguno	Clúster local O(10^2) máquinas de gran área de distribución O(10^3)

Figura 1. Comparación del estado del arte de plataformas con respecto a LarKC [7]

- ***Repositorios semánticos escalables***

[6] Mide el número de elementos correctamente identificados como un porcentaje del número total de elementos correctos y la cantidad de elementos que fueron identificados de los que se tenían que identificar, teniendo que mientras mayor sea la taza de recuperación significa que al sistema no le faltan elementos correctos [22].

[7] Tomado de: Towards LarKC: a Platform for Web-scale Reasoning [22].

Un repositorio semántico no es más que un componente de *software* para almacenar y manipular datos RDF. Está compuesto por tres componentes distintos:

- Una base de datos RDF para almacenar, recuperar, actualizar y eliminar sentencias en RDF.
- Un motor de inferencia que utiliza reglas para inferir en el conocimiento de las declaraciones explícitas.
- Un motor de consulta de gran alcance para el acceso a los conocimientos explícitos e implícitos.

Ontotext[8] produce un alto rendimiento de RDF, almacenamiento de OWL y tecnología de inferencia, supervisa de forma activa el estado de arte en esta área. Proporciona un rendimiento extenso (ver figura #2) que facilita una comparación de las herramientas en términos de escalabilidad, velocidad y capacidad de inferencia, en la cual se manifiestan los mejores resultados obtenidos correspondientes a dichas herramientas.

El motor de inferencia Cycorp, es capaz de hacer frente a las bases de conocimiento pues contiene varios millones de aserciones, utilizando de manera complementaria la tecnología Ontotext con respecto a las tareas de razonamiento. La funcionalidad almacenamiento y consulta de LarKC se basa en la tecnología Ontotext detrás de SwiftOWLIM y BigOWLIM y en la tecnología suministrada por CycEur [21].

[8] Un desarrollador líder de tecnología semántica. Sostiene aplicaciones semánticas como: OWLIM, ORDI, KIM entre otras.

Herramientas	Escala (mil declaraciones)	Inferencia	Velocidad de carga en 100 st/sec	Hardware (GB RAM)
KAON2	~10	OWL DL + rules	20	0.5
RacerPro	1	OWL DL	-	0.5
Minerva (IBM)	2	OWL Lite+/-	>1	0.5
Triple20	40	OWL Lite+/-	6	2
SwiftOWLIM	10-80	OWL Lite+/-	20-60	1-16
Sesame 2.0 NS	70	RDFS+	6	0.8
ORACLE10R2	100	RDFS+	>1	2
Jena v2.1 ½.3	7-200	-	?-6	2-?
KOWARI	235	Ninguno	4	?
RDFGateway	262	OWL Lite+/-	>1	?
AlegroGraph	1000	RDFS-	20	2
OpenLink	1060	Ninguno	12	8
BigOWLIM	1060	OWL Lite+/-	4	12

Figura 2. Repositorios semánticos escalables[9]

En trabajos recientes, DERI (Digital Enterprise Research Institute) ha implementado un almacenamiento distribuido en RDF que permite reportes en tiempo real de más de 7 billones de tripletas. Este sistema se encuentra limitado en la recuperación de datos y no ejecuta ninguna inferencia. Claramente esta herramienta es una excelente base para la implementación de indexación y técnicas de búsqueda, pero no es suficiente. LarKC tiene por objetivo la inferencia, en las operaciones de búsqueda como paradigma de la ciencia, en comparación con la recuperación. Es evidente que ninguna mejora en los sistemas de indexación eficiente será capaz de derrotar a la exponencial búsqueda del espacio generado por la inferencia.

- **Búsqueda Web**

[9] Tomado de: Towards LarKC: a Platform for Web-scale Reasoning [22].

Para el procesamiento de volúmenes a escala web, GYM (Google, Yahoo, Microsoft) utiliza una combinación de una de las más antiguas y sencillas estructuras de recuperación de datos (un archivo invertido que se relaciona con los términos de búsqueda a los documentos) y un algoritmo de clasificación, cuyo componente más importante se deriva de la estructura de enlaces de la Web. En general cuando se especifica una búsqueda, los usuarios entran un pequeño número de términos como una consulta, sobre la base de las palabras que la gente espera que se produzca en los tipos de documentos que buscan. Esto da lugar a un problema fundamental, conocida como "indicador de términos de sinonimia", donde no todos los documentos usarán las mismas palabras para referirse al mismo concepto. Por lo tanto, no todos los documentos que tratan sobre el concepto van a ser recuperada por una búsqueda basada por palabras clave. Por otra parte, los términos de consulta, por supuesto, puede tener múltiples significados, este problema se llama "la polisemia término de consulta". La ambigüedad de la consulta puede llevar a la recuperación de la información irrelevante o el fracaso para recuperar información relevante. El objetivo de LarKC es mostrar como las técnicas de alto valor pueden escalar grandes volúmenes de información de lo que es factible en la actualidad.

Luego de un análisis de los sistemas anteriormente expuestos, tanto MaRVIN como Hadoop se utilizan para el procesamiento de grandes cantidades de datos RDF, sin embargo, ambos enfoques se basan en modelos de razonamiento radicalmente diferentes:

- Hadoop es más rápido que MaRVIN.
- MaRVIN puede adaptarse con más rapidez a otras lógicas.
- La coordinación de MaRVIN es descentralizada, mientras que Hadoop es centralizada.

Ambos enfoques pueden ser desplegados dentro de la arquitectura de LarKC ya que contribuyen de alguna forma dentro del funcionamiento de la misma [23].

1.5 El proceso de descubrir conocimiento

Luego de un estudio del proceso de descubrir conocimiento, se puede definir tomando como base el concepto de Guillermo Matos pero realizándole modificaciones necesarias para esta investigación que:

El proceso de descubrir conocimiento a partir de los datos se puede definir como el conjunto de técnicas tales como la ontología y la extracción de textos y herramientas que las utilizan tales como los sistemas de razonamiento a escala web aplicadas al proceso no trivial de presentar conocimiento implícito, previamente desconocido, potencialmente útil y humanamente comprensible, a partir de grandes conjuntos de datos, con objeto de predecir de forma automatizada tendencias y comportamientos y/o descubrir de forma automatizada modelos previamente desconocidos [24].

La necesidad de extraer conocimiento sobre grandes bases de datos se vuelve cada vez más engorroso debido a que el volumen de la información que se maneja continuamente necesita más recursos. Es por esto que el conocimiento también se puede definir como aquellos patrones que se han aprendido a detectar y que se han guardado porque permiten aplicarlos a nuevos datos y por tanto, predecir el comportamiento de los fenómenos que nos rodean [25].

La cantidad de información que administran las organizaciones crece no solo en cantidad, sino también en variedad en consecuencia al desarrollo tecnológico que existe. Trayendo consigo la posibilidad de descubrir conocimiento sobre diversas fuentes de datos. Estas fuentes se encuentran organizadas en servidores corporativos de forma estructurada, semiestructurada y no estructurada. Las organizaciones, en su labor diaria intentan trasformar la información en conocimiento entonces, cuando se descubre conocimiento sobre fuentes de datos heterogéneas, se refiere, al descubrimiento sobre cantidades de datos de diversas naturalezas que pueden ser combinados para a partir de ellos extraer información relevante.

1.6 Arquitectura de software

Para descubrir conocimiento en un sistema capaz de recuperar información por su significado, utilizando las técnicas antes mencionadas, se hace necesario diseñar su arquitectura. Para

esto se realiza un análisis de los estilos y patrones que pueden ser utilizados en esta investigación.

1.6.1 Orígenes - Definiciones

La Arquitectura de *Software* (AS) como concepto fue identificada en 1968 por Edsger Dijkstra, de la universidad tecnológica de Eindhoven y Premio Turing 1972. Propuso que se estableciera una estructura adecuada a los sistemas de *software* antes de empezar a programar. Dijkstra, sostenía que la ciencia de la computación era una rama aplicada de las matemáticas y planteaba seguir pasos formales para descomponer problemas de gran tamaño. David Parnas en los años 70, manifestó que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso varios principios de diseño que debían seguirse para obtener una estructura adecuada [26].

Brooks, el diseñador del sistema operativo OS/360 y Premio Turing 2000, en el año 1975, utilizó el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” [27], también diferenciaba entre arquitectura e implementación, la primera decía ***qué*** hacer y la segunda se ocupa de ***cómo.***

La AS con el paso de los años siguió tomando auge, pero no es hasta 1992 que se define como disciplina de *software* en la publicación “*Foundations for the study of software architecture*” escrito por Perry y Wolf donde planteaban: “La década de 1990, creemos, será la década de la arquitectura de *software*” [28]. Dándole cumplimiento a lo planteado por Perry y Wolf, en los años 90, se logró la de la consolidación y diseminación de la AS a gran escala.

En el actualidad existen diversas definiciones de la AS, una de las primeras es propuesta por Perry y Wolf en 1992: “Un conjunto de elementos arquitectónicos que tienen una forma particular" [28]. Ellos plantean tres tipos de elementos: procesamiento, datos y conexión. Otras definiciones clásicas de la arquitectura de *software* son de Garlan y Shaw en 1994 y en 1996 donde proponen: “una colección de componentes y conectores, junto con una descripción de las interacciones entre los componentes y conectores” [29]. También, según Kazman, Clements y Bass plantean que "la arquitectura de *software* de un programa o un sistema de

computación, es la estructura o estructuras del sistema, que comprenden componentes de *software*, las propiedades externamente visibles de esos componentes y las relaciones entre ellos" [30].

La IEEE en el año 2000, publica el documento IEEE 1471 donde se introduce la definición de las AS que ha sido adoptada en el mundo:

"*La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución*" [31].

Esta definición, deja claro que la arquitectura de *software* no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se lleva a cabo durante todo el ciclo de desarrollo de *software*.

1.6.2 Corrientes arquitectónicas

- *Arquitectura como una etapa de ingeniería y diseño orientada a objetos*: James Rumbaugh, Grady Booch, Ivar Jacobson, y otros, proponen un modelo que se vincula al UML (Lenguaje Unificado de Modelado) y RUP (Proceso Unificado de Desarrollo). La arquitectura se limita a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está sistemáticamente relacionada al requerimiento que llega antes o a la composición del diseño que llega después. No considera la simplicidad de los diagramas sino la complejidad de estos. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo.

- *Arquitectura estructural, basada en un modelo estático de estilos, ADL (*Arquitecture Description Language*) y vista*: Carnegie Mellon en Pittsburgh: Mary Shaw, David Garlan, Paul Clements, Robert Allen, John Ockerbloom, Gregory Abowd entre otros proponen que es la corriente fundamental y clásica de la disciplina. En toda la corriente, el diseño arquitectónico no solo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones,

pocas veces se encuentran referencias a lenguajes de programación o piezas de código y en general nadie habla de clases o de objetos.

- *Arquitectura basada en patrones*: No se encuentra tan rígidamente vinculada a UML en el modelado ni a CMM (Capability Maturity Model) en la metodología. Principalmente se basa en la redefinición de los estilos como patrones POSA (Pattern-Oriented *Software* Architecture), el diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

- *Arquitectura procesual y metodologías*: Esta surge desde comienzos del siglo XXI, con centro en el SEI (*Software* Engineering Institute) y con la participación de algunos arquitectos de Carnegie Mellon de la primera generación y varios nombres nuevos de la segunda: Len Bass, Paul Clements, Rick Kazman, Félix Bachmann, Charles Weinstock, Jeromy Carriére, Mario Barbacci, Fabio Peruzzi, intenta establecer modelos de ciclo de vida y técnicas de diseño, selección de alternativas, análisis, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de *software* [32].

1.6.3 Estilos arquitectónicos

Los “estilos arquitectónicos” de *software* son marcos de referencias arquitectónicas, formas comunes o clases de sistemas. En el texto de la arquitectura de *software*, Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica [28].

Los estilos de arquitectura se definen como las 4C:

- *Componentes*
- *Conectores*
- *Configuraciones*

- *Restricciones (Constraints)*

A la hora de definir un estilo se debe tener en cuenta el tipo de aplicación ya que puede imponer restricciones que acotan la interacción de los componentes, además se tiene en cuenta el patrón de organización general.[10]

Los principales estilos que se usan en la actualidad están divididos por clases de estilos que abarcan una serie de estilos arquitectónicos específicos:

- *Estilos de flujo de datos*: esta familia de estilos enfatiza la reutilización y la modificabilidad. Se usa principalmente para sistemas que implementan transformaciones de datos en pasos sucesivos.
    - *Arquitectura tuberías y filtros*: una tubería es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que el procesamiento de datos se ejecuta como un flujo. Los datos se transportan a través de las tuberías entre los filtros, convirtiendo gradualmente las entradas en salidas. Este estilo se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada.
- *Estilos centrados en datos:* esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.
    - *Arquitecturas de pizarra o repositorio:* esta arquitectura está compuesta por dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Estos sistemas se usan en aplicaciones que requieren interpretaciones complejas de proceso de señales (reconocimiento de patrones, reconocimiento de habla).

[10] Para describir los estilos y arquitecturas en el trabajo se usó como referencia la de, Juan Carlos de la Cruz [33] aunque en aspectos importantes se utilizan otros autores.

- *Estilos de llamada y retorno*: esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

  - *Arquitecturas en capas*: es este estilo arquitectónico cada capa presta servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se reducen las dependencias entre capas, resulta más fácil remplazar la implementación de una capa sin afectar al resto del sistema.

  - *Arquitecturas orientadas a objetos*: los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de Mary Shaw y David Garlan, los objetos representan una clase de componentes que ellos llaman controladores debido a que son responsables de preservar la integridad de su propia representación [29]. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos.

  - *Arquitectura basada en componentes*: los sistemas de *software* basados en componentes se apoyan en principios definidos por una ingeniería de *software* específica. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser reveladas y utilizadas en tiempo de ejecución.

- *Estilos de código móvil*: esta familia de estilos enfatiza la portabilidad. Ejemplo de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguajes de comandos.

- *Arquitectura de máquinas virtuales*: esta arquitectura se conoce como intérpretes basados en tablas y sistemas basados en reglas. Se puede decir que estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen, sin duda, un extenso espectro que está comprendido los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación [32].

- *Estilos peer - to - peer:* esta familia se conoce también como componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes.

  - *Arquitectura basada en eventos*: estas se han llamado también arquitectura de invocación implícita, se vinculan con sistemas basados en actores en publicación-suscripción. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciarse mediante de difusión uno o más eventos.

  - *Arquitecturas orientadas a servicios*: esta arquitectura más conocida por SOA construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces. Es una relación entre servicios y consumidores de servicios, ambos los suficientemente amplios como para representar una función de negocio completa.

    "La mayoría de las definiciones de SOA identifican la utilización de servicios web (empleando SOAP (*Simple Object Access Protocol*) y WSDL (*Web Services Description Language*)) en su implementación, no obstante, se puede implementar una SOA utilizando cualquier tecnología basada en servicios [33]."

    Existen cuatro elementos básicos para la construcción de una arquitectura orientada a servicios:

    - Operación: es la unidad de trabajo o procesamiento en una arquitectura SOA.

- Servicio: es un contenedor de lógica compuesto por un conjunto de operaciones, que ofrecerá a sus usuarios.

- Mensaje: para poder ejecutar una determinada operación, es necesario un conjunto de datos de entrada. Una vez ejecutada la operación, esta devolverá un resultado. Los mensajes son los encargados de encapsular esos datos de entrada y de salida.

- Proceso de negocio: son un conjunto de operaciones ejecutadas en una determinada secuencia (intercambiando mensajes entre ellas) con el objetivo de realizar una determinada tarea.

Para esta investigación un servicio encapsula funcionalidad de negocios y proporciona dicha funcionalidad mediante interfaces públicas definidas. Los componentes del estilo (o sea los servicios) están débilmente acoplados. Este puede recibir requerimientos de cualquier origen. Su funcionalidad se puede ampliar o modificar sin rendir cuentas a quienes lo requieran.

Al contrario de las arquitecturas orientado a objetos, las SOAs están formadas por servicios de aplicación altamente interoperables y débilmente acoplados. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación (ejemplo WSDL). La definición de la interfaz oculta las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo. Con esta arquitectura, se pretende que los componentes de *software* desarrollados sean muy reutilizables, ya que la interfaz se define siguiendo un estándar; así, un servicio C# podría ser usado por una aplicación Java y viceversa [33].

Una aplicación SOA está integrada por un conjunto de procesos de negocio, que a su vez estarán compuestos por aquellos servicios que proporcionan las operaciones que se necesitan ejecutar para que el proceso llegue a un buen término. Estas

operaciones se producen mediante el envío de los datos a través de mensajes. El elemento fundamental de una arquitectura SOA es el servicio. La arquitectura orientada a servicios propone el uso de los estándares como XML, SOAP, WSDL, entre otros. Aunque no siempre es necesaria la implementación de todos ellos, es muy recomendable que se utilicen todos.

- *Arquitecturas basadas en recursos*: este estilo arquitectónico conocido como Representational State Transfer (REST), define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URIs identifican los recursos y HTTP (HyperText Transfer Protocol. Protocolo) es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. Este estilo fue originalmente introducido para construir largos y escalables sistemas de hypermedia distribuidos.

1.6.4 Patrones

Los patrones fueron sugeridos por Christopher Alexander en 1977, el cual escribió varios artículos y libros referentes a ¿qué son los patrones?, los textos encontrados se refieren a patrones arquitectónicos en cuanto a arquitectura real, o sea, de edificios y ciudades y no de arquitectura de aplicaciones de *software*. Una de las ideas más brillantes de la década de los 90 fue la de vincular las estructuras recurrentes, aquellas entidades que ocurrían una y otra vez, a problemas en determinados contextos [34]. Un patrón:

- Soluciona un problema en un contexto.
- Codifica conocimiento específico acumulado por la experiencia en un dominio.
- Se encuentra dentro de un sistema bien estructurado.

- Describe un problema que ocurre una y otra vez en nuestro ambiente y luego escribe el núcleo de la solución a este problema de manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.

Los patrones pueden dividirse o clasificarse en:

- *Patrones de arquitectura*: relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad, acoplamiento.

- *Patrones de diseño*: fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución: Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC). Se enfrentan a problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, entre otros. Se ubican en el refinamiento del diseño. Existen diversos tipos de patrones como: los GRASP (General Responsibility Assignment Software Patterns), que son patrones generales para la asignación de responsabilidades y los GoF (Gang Of Four) que clasifican en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

- *Patrones de análisis*: usualmente específicos de aplicación.

- *Patrones de proceso o de organización*: que tratan lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.

Relación entre estilos y patrones

El tema más importante y famoso en la arquitectura de *software* es, sin duda, el de los patrones, tanto en los patrones de diseño como en los de arquitectura. Sin embargo, solo en pocas literaturas técnicas existe el vínculo entre estilos y patrones. La relación entre ellos es que una vez que se ha decidido que estilos se van a usar en la aplicación los patrones que tienen relación con estos estilos ayudarán a derivar el diseño y posteriormente la programación correspondiente.

1.7 Métodos de evaluación de la arquitectura

El objetivo de la evaluación de una arquitectura es para cumplir con las necesidades de los stakeholders[11] y habilitar los requerimientos, atributos de calidad y restricciones. Las mediciones que se realizan pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Las mediciones se clasifican en [35]:

- La ***medición cualitativa*** se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.

- La ***medición cuantitativa*** busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de *software*. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación.

- La ***medición de máximo y mínimo teórico*** contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

¿Por qué evaluar una arquitectura?

Se debe evaluar una arquitectura para analizar los riesgos en sus propiedades y estructura, que puedan ocasionar posibles daños al sistema de *software*. Se deben verificar que los requerimientos no funcionales se encuentran en la arquitectura.

¿Cuándo una arquitectura puede ser evaluada?

[11] Término inglés, para referirse a «quienes pueden afectar o son afectados por las actividades de una empresa» [51].

La evaluación de la arquitectura puede ser realizada en distintas etapas, Kazman propones dos variantes, la temprana y la tardía. La primera puntualiza que no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo. Y la segunda define que se debe realizar cuando la arquitectura se encuentra establecida y la implementación se ha completado [35].

La evaluación de esta arquitectura se realiza en una etapa temprana para probar que las decisiones tomadas son viables y que puede ser desplegada en un ambiente real.

¿Quiénes participan en una evaluación?

Las evaluaciones a la arquitectura casi siempre la realizan los miembros del equipo de desarrollo, arquitecto, diseñador, entre otros. Sin embargo, puede realizarse por personas especialistas en el tema.

1.7.1 Técnicas de evaluación

Existen un grupo de técnicas para evaluar que se clasifican en cualitativas y cuantitativas:

- Técnicas de cuestionamiento o cualitativas: utilizan preguntas cualitativas para preguntarle a la arquitectura.
- Cuestionarios: en momentos tempranos.
- Escenarios: específicas del sistema para arquitecturas avanzadas.
- Listas de chequeo: específicas del dominio de la aplicación.
- Técnicas de mediciones: sugiere hacerle medidas cuantitativas a la arquitectura.
- Utiliza métricas arquitectónicas, como acoplamiento, cohesividad en los módulos, profundidad en herencias, modificabilidad.
- Simulaciones, prototipos y experimentos.

Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada [36].

El planteamiento anterior muestra las técnicas para la evaluación de las arquitecturas y se decide realizar una evaluación cualitativa que permita medir los atributos de calidad del arquetipo y los componentes implementados.

1.7.2 ¿Por qué cualidades puede ser evaluada una arquitectura?

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías:

- Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla #1 (Descripción de atributos de calidad observables vía ejecución).
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla #2 *(*Descripción de atributos de calidad no observables vía ejecución).

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso
Confidencialidad	Es la ausencia de acceso no autorizado a la información
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria

Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos

Tabla 1. Descripción de atributos de calidad observables vía ejecución [35]

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados
Integridad	Es la ausencia de alteraciones inapropiadas de la información
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad*
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida

	y a bajo costo
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, *software* o una combinación de los dos
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental
Capacidad de Prueba	Es la medida de la facilidad con la que el *software*, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el *software* fallará en su próxima ejecución de prueba

Tabla 2. Descripción de atributos de calidad no observables vía ejecución [35]

Métodos de evaluación de arquitecturas

- *ATAM (Architecture Trade-off Analysis Method)*: está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (*Software* Architecture Analysis Method). ATAM surge del hecho de que revela la forma en que una arquitectura determinada satisface ciertos atributos de calidad y provee una visión de cómo los atributos de calidad se relacionan con otros. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos constituyen los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la manera en la que el sistema puede crecer, responder a cambios e integrarse con otros sistemas, entre otros [35]. El método de evaluación ATAM comprende nueve

pasos, agrupados en cuatro fases (Presentación, Investigación y Análisis, Pruebas y Reporte).

- *Bosch (2000*) plantea que: “el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Cuando la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requerimientos. Luego de transformada es evaluada de nuevo” [35]. Este método consta de 5 pasos divididos en dos etapas.

- ADR (Active Design Review). “se utiliza para la evaluación de diseños detallados de unidades del *software* como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto” [35].

- *ARID (Active Reviews for Intermediate Design).* es un método de bajo costo y gran beneficio, es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo [35]. Es un híbrido entre ADR y ATAM [37]. Se basa en ensamblar el diseño de los stakeholders para articular los escenarios de usos importantes o significativos y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Evaluación) [38].

- *Losavio (2003):* es un método para evaluar y comparar arquitecturas de *software* candidatas, que hace uso del modelo de especificación de atributos de calidad adaptado del modelo ISO/IEC 9126. La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares para efectos de la evaluación. El método contempla siete actividades [35].

Comparación entre métodos de evaluación

	ATAM	SAAM	ARID	*Bosch (2000).*	*Losavio (2003)*
Atributos de Calidad Contemplados	Modificabilidad Seguridad Confiabilidad Desempeño	Modificabilidad Funcionabilidad	Conveniencia del diseño evaluado	Seleccionados por el arquitecto, de acuerdo a la importancia sobre el sistema	Funcionabilidad Confiabilidad Usabilidad Eficiencia Mantenimiento Portabilidad
Objetos Analizados	Estilos Arquitectónicos, Documentación, Flujo de Datos y Vistas Arquitectónicas	Documentación y Vistas Arquitectónicas	Especificación de los componentes	Estilos Arquitectónicos, Vistas Arquitectónicas, Patrones Arquitectónicos, Patrones de Diseño y Patrones de Idioma	Especificación de Atributos de Calidad
Etapas del Proyecto en las que se Aplica	Luego que el diseño de la arquitectura ha sido establecido	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos	A lo largo del diseño de la arquitectura	Luego que el diseño de la arquitectura ha sido establecido	Luego que el diseño de la arquitectura ha sido establecido
Enfoques Utilizados	Árbol de Utilidad y lluvia de ideas para articular los requerimientos	Lluvia de ideas para escenarios y articular los requerimientos de calidad	Revisiones de diseño, lluvia de ideas para	Análisis de perfiles (profiles)	Análisis y comparación con de los resultados para

	de calidad Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos	Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios	obtener escenarios		las arquitecturas candidatas

Tabla 3. Comparación de métodos de evaluación [35]

Hasta el momento se han presentado varios métodos de evaluación de arquitectura algunos más completos que otros, pero independientemente de esto todos tienen algo en común y es que utilizan la técnica de escenarios como vía de constatar en qué medida la arquitectura responde a los atributos de calidad requeridos por el sistema. Para esto ver la tabla #3 (Comparación de métodos de evaluación). El método ATAM evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura.

1.8 Análisis del capítulo

En el estudio del manejo de la información en grandes volúmenes de datos y el avance tecnológico que existe en la actualidad, se puede evidenciar la evolución del proceso de descubrir conocimiento. Desarrollar un sistema que incluya este proceso en él no es tarea fácil pues involucra diversas herramientas y técnicas para realizar la inferencia sobre los datos que componen la información. Como cumplimiento de los objetivos se tiene:

- Descubrir conocimiento en datos heterogéneos es el conjunto de técnicas y herramientas para extraer datos en grandes volúmenes para realizar inferencia y predicciones sobre ellos.

- Los términos que componen el proceso de descubrir conocimiento son las técnicas: ontología y extracción de textos y las herramientas, los sistemas para el razonamiento de datos a escala web como: LarKC, MaRVIN y Hadoop.

- La característica que distingue el proceso de descubrir conocimiento de otro, radica en la extracción del conocimiento sobre grandes volúmenes de datos que se encuentran de forma estructurada, semiestructurada y no estructurada.

- La arquitectura de *software* es una estructuración de un sistema compuesto por sus componentes, las relaciones entre ellos y los elementos principales que guían su diseño y creación

- El estilo arquitectónico peer-to-peer permite la construcción de componentes desacoplados y altamente interoperables que se comunican mediante mensajes, facilitando el uso de estos en diversos sistemas informáticos.

- El método ATAM propone una manera de evaluar la arquitectura apoyándose en los atributos de calidad y la forma en que estos interactúan.

CAPÍTULO 2

Propuesta del diseño arquitectónico

LarKC es una plataforma para el razonamiento incompleto distribuido que sirve como herramienta en el proceso antes estudiado. Sin embargo, para relacionar la información por significado para descubrir conocimiento en el proceso de búsqueda en la universidad, se hace necesario diseñar una arquitectura. En esta investigación solo se realiza el diseño de la capa de dominio de usuario y se crea un prototipo ejecutable para demostrar que la solución es viable.

2.1 Análisis del problema

La investigación en sus inicios arrojó el siguiente problema científico. ¿Cómo relacionar la información por significado para descubrir conocimiento produciendo mejores resultados en las búsquedas? Para esto, se debía investigar el proceso de descubrimiento de conocimiento sobre datos heterogéneos realizado en el capítulo uno. Luego, siguiendo lo planteado por el objeto de estudio, el campo de acción y el objetivo general, la investigación fue acotando sus límites y trasladó el centro de atención a lograr una arquitectura basada en componentes capaz de poder recuperar información por su significado en la Universidad de las Ciencias Informáticas.

En esta etapa de la investigación se visitó primeramente la biblioteca del centro para encontrar los procedimientos que rigen la universidad para el manejo de la información y sus flujos de trabajo, pero los especialistas no disponían de esta documentación. La búsqueda en este caso fue fallida, pero se dispuso a visitar las áreas pertenecientes a la UCI para identificar cuáles pertenecían a los procesos sustantivos de la institución. En esta pesquisa se encontró a la Dirección de Desarrollo Institucional, creada en el 2010 que tiene entre sus objetivos principales conocer el estado de los procedimientos y flujos de trabajo con los que cuenta la UCI. Esta área se encuentra trabajando, pero aún no cuenta con una descripción de los procesos aprobados por la rectoría de la ciudad universitaria, pero sí cuentan con una propuesta que es la que se utilizará para esta investigación. Se identificaron los siguientes procesos sustantivos para la universidad en cada una de sus áreas.

Gestión

- Auditoría

  Contabilidad
- Jurídico
- Recursos humanos

Reclutamiento de recursos humanos
Nómina y control de expedientes
Seguridad y salud

- Recursos materiales

Selección de proveedores, compras y control de proveedores
Almacenes y su control
Elaboración y distribución de alimentos

- Recursos financieros

  Planificación financiera
  Control financiero
  Estadística de las finanzas

- Recursos tecnológicos

  Planificación y uso de laboratorios
  Selección de proveedores de compras y control de proveedores
  Distribución y mantenimiento
  Red

- Seguridad

  Seguridad de los bienes informáticos

Seguridad informática

- Ambiente de vida y trabajo

  Mantenimiento constructivo
  Transporte y su mantenimiento
  Servicios generales
  Beca

- Cooperación internacional

  Cooperación internacional
  Atención a delegaciones
  Trámites para viajes internacionales

Procesos Productivos

- Formación
- Postgrado
- Investigación
- Extensión universitaria
- Desarrollo

En el siguiente flujograma se refleja una vista más generalizada de los procesos anteriormente mencionados:

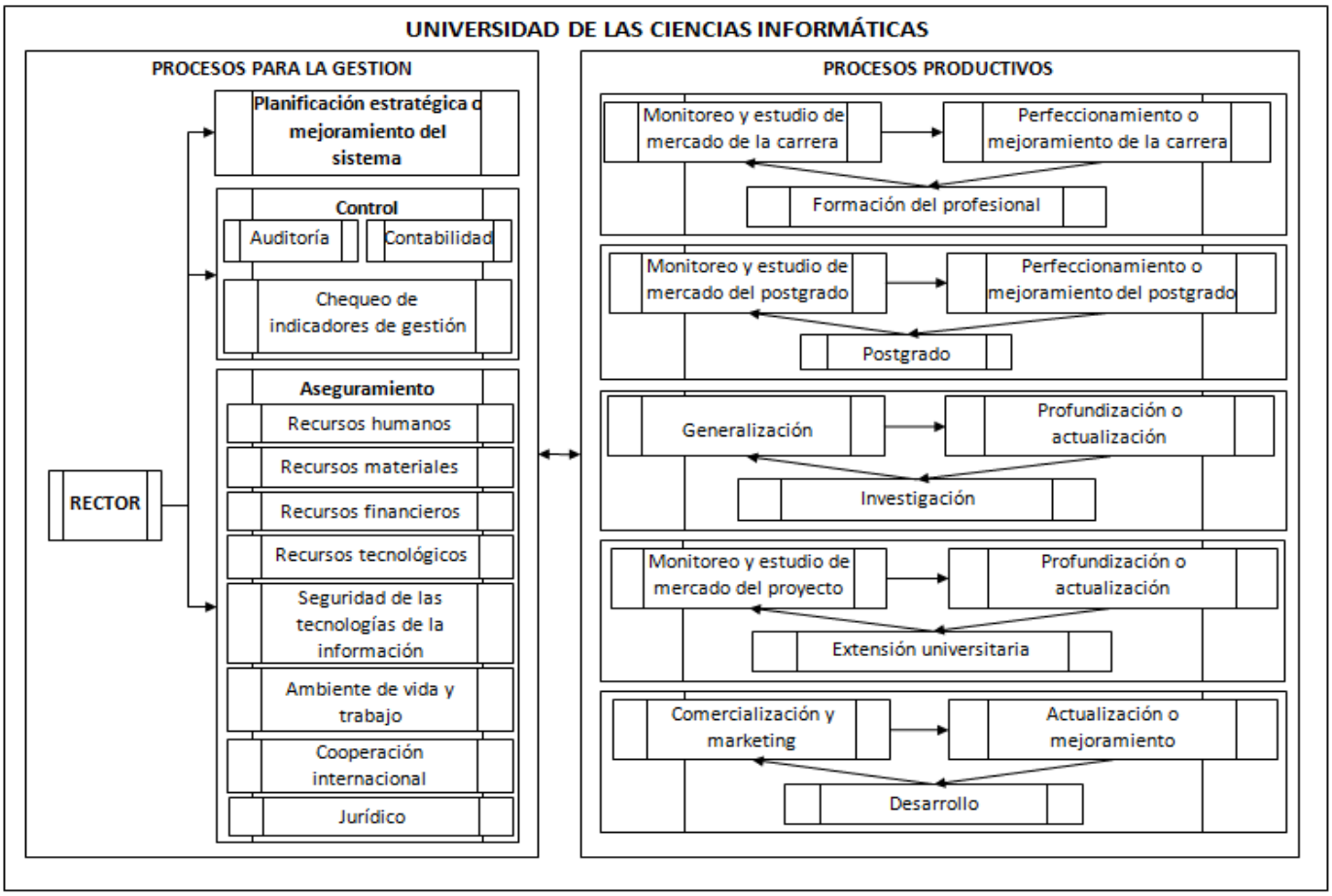

Figura 3. Procesos UCI[12]

Teniendo en cuenta todas las áreas anteriormente expuestas que contribuyen con los procesos sustantivos de la UCI, se seleccionó una muestra para su posterior análisis: recursos humanos, formación profesional, cooperación internacional, seguridad de las tecnologías de información y el sector jurídico. Estos lugares fueron visitados con el propósito de identificar si contaban con un sistema automatizado que pudiera contribuir con la solución del trabajo de investigación, reduciendo el alcance a las siguientes: formación profesional, seguridad de las tecnologías de información, recursos humanos y el sector jurídico. Debido a la cantidad de procesos que se establecen en cada área se seleccionó solo el conjunto de procedimientos

[12] Tomado de: La Dirección de Desarrollo Institucional de la Universidad de Ciencias Informáticas.

pertenecientes a expediente académico para esta investigación[13]. Se presenta un mapa conceptual de este procedimiento (ver anexo #2).

Para darle solución al problema planteado se pretende diseñar una arquitectura basada en la plataforma LarKC, pues es la más idónea en cuanto al razonamiento de datos heterogéneos a gran escala.

2.1.1 Aspectos generales de la arquitectura de LarKC

Arquitectura LarKC

La plataforma LarKC fue diseñada para ser un componente de arquitectura flexible que permita el diseño y ensayo de nuevas técnicas de razonamiento. El razonamiento en LarKC se divide a través de flujos de trabajo (workflows dentro de LarKC) de varios pasos. Cada paso corresponde a un tipo de componente (plug-in dentro de LarKC) y la unión de varios de ellos conforman un flujo de trabajo. Se puede manejar también el razonamiento sobre bases de conocimiento de gran tamaño. El flujo de trabajo requiere un poco de apoyo a su composición, creación de instancias, seguimiento y control, el cual es proporcionado por la plataforma.

El diseño de esta arquitectura permite la superación de las limitaciones conocidas y el cumplimiento de las nuevas solicitudes de los usuarios. Al ser una plataforma permite utilizar aplicaciones para realizar razonamiento sobre diversos datos para así descubrir conocimiento.

[13] Esta selección se realizó mediante el método de selección de muestras: *muestreo de juicio*, que realiza la selección mediante el juicio personal.

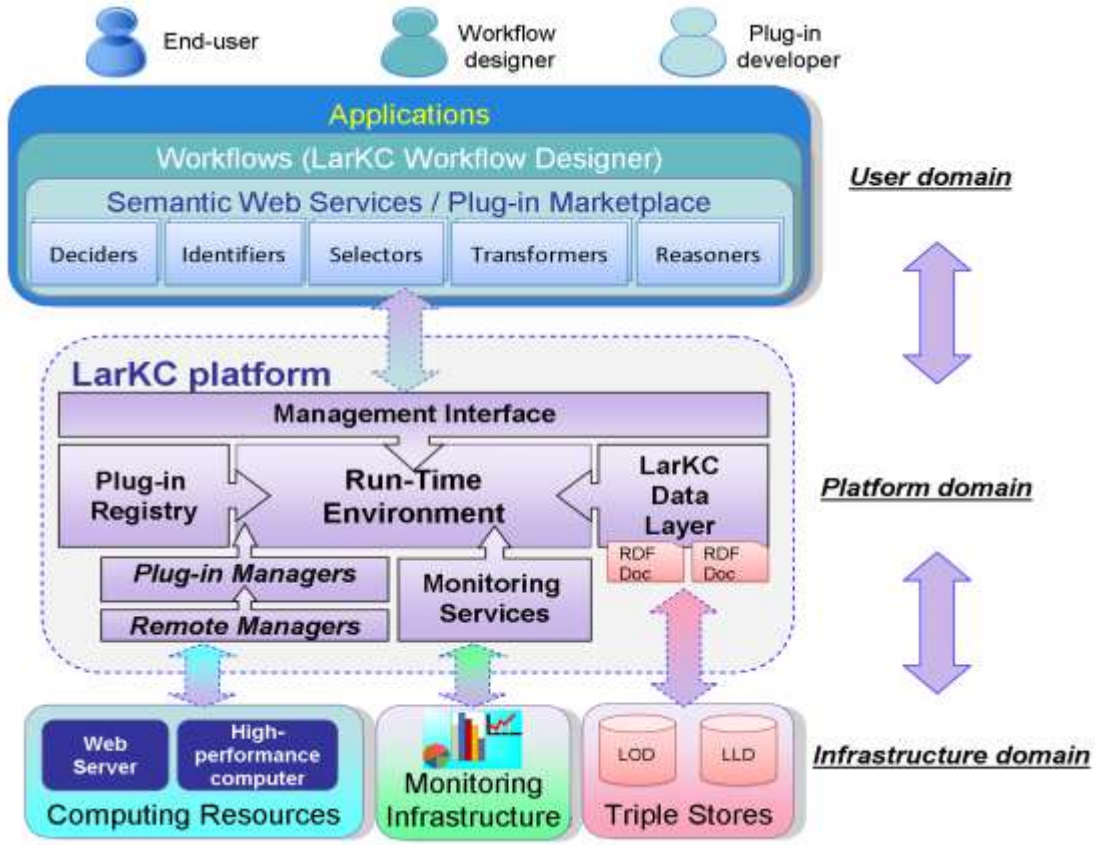

Figura 4. Arquitectura LarKC[14]

La figura anterior es una vista general de la arquitectura de LarKC, que desde el punto de vista de la organización del sistema, posee una infraestructura de *hardware* y *software* que ofrece un *framework* para desarrollar componentes desacoplados, así como la construcción de flujos de trabajo basados en estos. Un entorno de desarrollo y ejecución para el lanzamiento de los flujos de trabajo sobre un grupo dinámico de datos heterogéneos y sistemas distribuidos y una infraestructura para el almacenamiento de datos.

La arquitectura presenta 3 subsistemas principales:

- Dominio de usuario.
- Dominio de la plataforma.

[14] Tomado de: D5.3.3 Final LarKC architecture and design.

- Dominio de infraestructura.

Este trabajo de diploma se centra en la capa de *dominio de usuario* para llegar a una posible solución, pero se realiza una breve descripción de las restantes capas pues resultan de vital importancia para el entendimiento de la investigación.

Dominio de usuario

Desde la perspectiva del usuario, la arquitectura habilita diversos bloques operacionales y actúa como una capa de consolidación para varias categorías de usuarios. Envuelve tres grupos fundamentales:

- *desarrolladores de componentes* que diseñan e implementan módulos y los despliegan en la plataforma o los publican si están listos para ser usados para colaborar con el Marketplace de LarKC. Web habilitada con un repositorio donde los componentes están disponibles para ser descargados y utilizados por los

- *diseñadores de los flujos de trabajo* con el propósito de construir un flujo de trabajo, combinándolos para resolver una tarea determinada de la manera más certera posible, seleccionando los más adecuados y por último los

- *usuarios finales* que usan los servicios que provee la plataforma.

El núcleo de la aplicación de LarKC está constituido por módulos de programación desacoplados, ligeros y flexibles y servicios que implementan alguna parte específica de la lógica de razonamiento, ya sea selección, identificación, transformación o algoritmos de razonamiento. Estos componentes están agrupados en categorías correspondientes a los elementos comunes de su funcionalidad:

- ***Identificador:*** se utiliza para reducir el alcance de una tarea de razonamiento. Dada una consulta es capaz de identificar toda la información que puede ser usada para resolverla defiendo solamente aquellos documentos potencialmente relevantes.

- ***Transformador***: transforma los datos de una representación a otra. Estas transformaciones pueden ser simples o complejas. Existen dos tipos de transformadores, los de consulta y los de un conjunto de información. El primero recibe una consulta como entrada en SPARQL y las transforma de una representación a otra ya sea a Triple Pattern Query, Keyword Query, entre otras. El segundo toma como entrada un conjunto de información y la transforma en dependencia de la implementación que tenga.

- ***Selector***: es el responsable de la selección que identifica las declaraciones de entrada que se deben utilizar para el razonamiento hacia una respuesta a la consulta de los usuarios. Usando como entrada un conjunto de datos RDF y retornándolo a otro más preciso.

- ***Razonador***: es destinado para aplicar diferentes tipos de razonamientos (deductivo, inductivo, entre otros) a los datos antes identificados, seleccionados y transformados.

- ***Decididor***: es un componente de supervisión utilizado para gestionar, controlar y si es necesario generar nuevos flujos de trabajo en tiempo de ejecución. Esto último se logra por medio de los llamados *Decididores* inteligentes, que utilizan los servicios especiales de la plataforma para crear instancias de nuevos flujos de trabajo sobre la marcha, si así lo exige la lógica de razonamiento.

Estos componentes actúan como servicios que ofrece la plataforma, son llamados plug-ins en la terminología LarKC que fácilmente pueden ser conectados a un flujo de trabajo. Para esta investigación se referirá a ellos como componentes o módulos. Dentro de un flujo se comunican entre ellos por medio de mensajes que contienen las declaraciones RDF, es decir, aceptan los argumentos y devuelven los resultados como conjuntos de sentencias. La topología del flujo de datos está definida por la configuración del flujo de trabajo.

Estructura de un módulo

Los cinco tipos de componentes de LarKC difieren solo conceptualmente y en sus descripciones, mientras tienen la misma estructura de Java. Extienden de una clase común que proporciona una abstracción entre el dominio de la plataforma y el dominio del módulo y proporcionan el soporte básico de las funcionalidades de ellos mismos, tales como el almacenamiento en caché, conexiones y mensajes con los otros componentes en un flujo de trabajo. La siguiente figura refleja como está compuesto el esqueleto de un componente.

```
1  package eu.larkc.plugin.select;
2
3  import org.opencyc.cycobject.CycConstant;
4  import org.opencyc.cycobject.Guid;
5  import org.openrdf.model.URI;
6
7  import javax.jms.Message;
8
9  import eu.larkc.core.data.SetOfStatements;
10 import eu.larkc.plugin.Plugin;
11
12 public class ExamplePlugin extends Plugin {
13
14   public ExamplePlugin(URI pluginName) {
15     super(pluginName);
16   }
17
18   @Override
19   protected void initialiseInternal(SetOfStatements workflowDescription) {
20       }
21
22   public SetOfStatements invokeInternal(SetOfStatements input) {
23     return null;
24   }
25
26   @Override
27   public void shutdownInternal() {
28   }
29 }
```

Figura 5. El esqueleto de un componente de LarKC[15]

Método *initialiseInternal()*:

Este método es llamado al principio de la creación de un flujo de trabajo, cuando el componente es instanciado. Se debe utilizar para manejar la inicialización del módulo antes de que realmente se llame desde la plataforma.

[15] Tomado de: D5.3.3 Final LarKC architecture and design.

Método *invokeInternal()*:
Es el método principal del componente, que es llamado cada vez que algunos componentes o la plataforma soliciten algunos datos del mismo.

Método *shutdownInternal()***:**
Este método es llamado cuando un flujo de trabajo es destruido por el usuario y se debe utilizar para la limpieza de algunos datos, si es necesario.

Cada componente necesita tener dos archivos especiales incluidos en el paquete, que describen el tipo y la funcionalidad. El archivo *wsdl* que describe el servicio y el archivo *rdf* que contiene el módulo de la ontología, que provee un vocabulario extenso, compartido con el vocabulario de flujo de trabajo.

La implementación de cada método se vuelve muy engorrosa, debido a que poseen una alta complejidad. La identificación y la selección son necesarias para soportar razonamiento sin límites. Su función principal consiste en reducir o expandir el conjunto de datos con el que se trabaja en dependencia de factores tales como, costo de procesamiento o resultados confidenciales. La tarea es retornar las tripletas de más relevancia en cuanto al contexto en un repositorio. Para esto, explota métodos de *recuperación de información*[16] (IR), *máquina de aprendizaje*[17] y *ciencias cognitivas*[18]. Existe cierta similitud en el proceso de recuperación

[16] Recuperación de información: Área de la ciencia y la tecnología que trata de la adquisición, representación, almacenamiento, organización y acceso a elementos de información. Desde un punto de vista práctico, dada una necesidad de información del usuario, un proceso de IR produce como salida un conjunto de documentos cuyo contenido satisface potencialmente dicha necesidad.

[17] Máquina de aprendizaje: Es la disciplina científica relacionada con el diseño y desarrollo de algoritmos que permiten a los ordenadores evolucionar basados en datos empíricos, tales como, datos de los sensores o bases de datos. Un aspecto importante de la máquina de aprendizaje es aprender a reconocer automáticamente los patrones complejos y tomar decisiones inteligentes sobre la base de datos.

[18] Ciencias cognitivas: Estudio interdisciplinario de cómo la información es representada y transformada en la mente / cerebro. Es el conjunto de disciplinas que surgen de la convergencia transdisciplinaria de investigaciones científicas y tecnológicas, en torno a los fenómenos funcionales y emergentes, dados a partir de las actividades neurofisiológicas del encéfalo y del sistema nervioso, incorporados y que típicamente se les denomina como: mente y comportamiento.

(identificación) de la información y la selección; para las tareas de selección se requieren métodos que se basan en tres factores fundamentales, costo / beneficio, origen y contexto, entre estos se tiene agrupamiento, clasificación, entre otros. La tarea de trasformación implica realizar una abstracción de un formato a otro que pueda ser entendido para realizar las tareas de razonamiento donde se infieren resultados sobre una muestra, utilizando para esto algoritmos de este tipo.

Flujo de trabajo LarKC

En términos de LarKC un flujo de trabajo es una aplicación de razonamiento que se construye de los componentes combinándolos de una manera útil. Sin embargo, para realizar dicha tarea el módulo de razonamiento debe ser siempre involucrado en el flujo. El propósito de este es especificar como los datos son procesados mediante los componentes. Una descripción del mismo, es un grafo dirigido donde los vértices son módulos y puntos finales (endpoint dentro del LarKC) y las aristas representan las conexiones entre estos. Un flujo de trabajo puede tener múltiples caminos (path dentro del LarKC) y un camino puede ser visto como la manifestación de un flujo de datos que debe tener al menos una entrada y exactamente una salida. Para sustentar la entrada del mismo y recuperar la salida, un punto final tiene que estar conectado a un solo camino, que va a ser el responsable de proporcionar el módulo de entrada para las consultas y el de salida para la recuperación de los resultados. Los flujos de trabajo son aplicaciones independientes, que pueden ser sometidos a la plataforma y ejecutados por medio de herramientas tales como el Diseñador de flujo de trabajo (Workflow Designer) (ver figura #6).

Dominio de la plataforma

Esta capa es el módulo principal del LarKC que consolida a los servicios clave que habilitan los componentes y las características de los flujos de trabajos que están íntimamente asociados con los datos LarKC y el modelo de ejecución. Además, sirve un entorno de desarrollo y ejecución para los módulos y los flujos de trabajo; también establece y gestiona la capa de datos para el manejo eficiente de flujo de datos y ofrece medios para el monitoreo del

rendimiento y evaluación. A continuación se mencionaran los principales servicios que son ofrecidos por el dominio de la plataforma:

Entorno de desarrollo y ejecución (RTE): es el "centro de control" de la plataforma LarKC y responsable de la inicialización y la invocación de los flujos de trabajo. Sus componentes principales son el Ejecutor[19] que contiene el id de todos los caminos y los puntos finales correspondientes en la parte superior de la misma.

Interfaz de administración: implementa un "puente" hacia los usuarios, así como a los desarrolladores que les permite presentar su flujo de trabajo en una descripción específica en RDF.

Componente de registro: se basa principalmente en el motor OpenCyc2 y es responsable de la gestión de los componentes. Permite el registro de componentes, incluyendo sus descripciones semánticas.

Componente de administración: facilita la integración de componentes dentro de los flujos de trabajo y la gestión de su ejecución.

Capa de datos: es una realización OWLIM[20] (Online Writing Lab) que soporta los componentes y aplicaciones con respecto al almacenamiento, la recuperación (incluida la transmisión) y la inferencia de peso ligero en la parte superior de grandes volúmenes de datos RDF. En particular, la capa de datos se utiliza para el almacenamiento de los datos transmitidos entre los componentes.

Dominio de infraestructura

[19] Es una clase que tiene como función principal la invocación del flujo de trabajo y los puntos finales.

[20] Repositorio Semántico, un componente de software para almacenar y manipular grandes cantidades de datos RDF. OWLIM se empaqueta como un dispositivo de almacenamiento y una capa inferencia (SAIL) para el marco de Sesame OpenRDF.

La ejecución eficiente de un flujo de trabajo a menudo requiere de recursos especiales de infraestructura que sean accesibles para los flujos. Estas instalaciones de recursos constituyen la infraestructura de dominio. La plataforma ofrece todo el apoyo necesario para un acceso fácil y transparente de la computación externa, almacenamiento y monitoreo de instalaciones.

2.2 El entorno de desarrollo

Cuando se va a desarrollar un *software* de cualquier índole, es buena idea planear e implementar un entorno de desarrollo desde el principio. De forma tal que todo el que se involucre en él sepa cómo construir el proyecto, corregir errores, cómo está organizado, cómo desplegarlo y cómo probarlo.

Composición del equipo de desarrollo

Comprender la composición del equipo de desarrollo da una idea del alcance que puede tener la propuesta de arquitectura de *software*. El equipo que mantendrá y desarrollará la primera versión de este *software*, tendrá que poseer amplios conocimientos en lenguajes como JAVA para el desarrollo del sistema y SPARQL para la realización de consultas, deberá dominar también RDF, TURTLE y OWL para el uso de las ontologías y familiarizarse con los de marcado como HTML, XML. Además, deberá poseer un amplio dominio de la metodología de desarrollo de *software* RUP.

Metodología de desarrollo de software

RUP (*Proceso Unificado de Desarrollo*): es una metodología de desarrollo de *software* que está basado en componentes e interfaces bien definidas y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es un proceso que puede especializarse para una gran variedad de sistemas de *software*, en diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto [39]. Es iterativo e incremental, dirigido por los casos de uso y centrado en la arquitectura.

Herramientas horizontales

Sistemas Operativos

La solución estará desarrollada en JAVA, lo que da la posibilidad de que el *software* pueda ser desarrollado sobre muchos sistemas operativos, en particular los basados en UNIX, cualquiera de la familia GNU/Linux y los de la familia Windows NT tales como Windows 7, Vista y XP. Como principal recomendación se propone el uso de las plataformas en GNU/Linux, ya que es un sistema operativo portable, multitarea, multiusuario y puede funcionar tanto en modo gráfico como en modo consola.

Herramientas verticales

Herramientas de compilación y construcción

Dentro de la solución del sistema, desde el ámbito de desarrollo, se encuentra el Apache ANT que es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente se utiliza durante la fase de construcción y compilación, es la más utilizada por la mayoría de los proyectos de desarrollo de Java, para construir productos OpenSource [40].

El diseñador de flujo de trabajos de LarKC, representando por una aplicación web, véase la siguiente figura para un mejor entendimiento de esta herramienta.

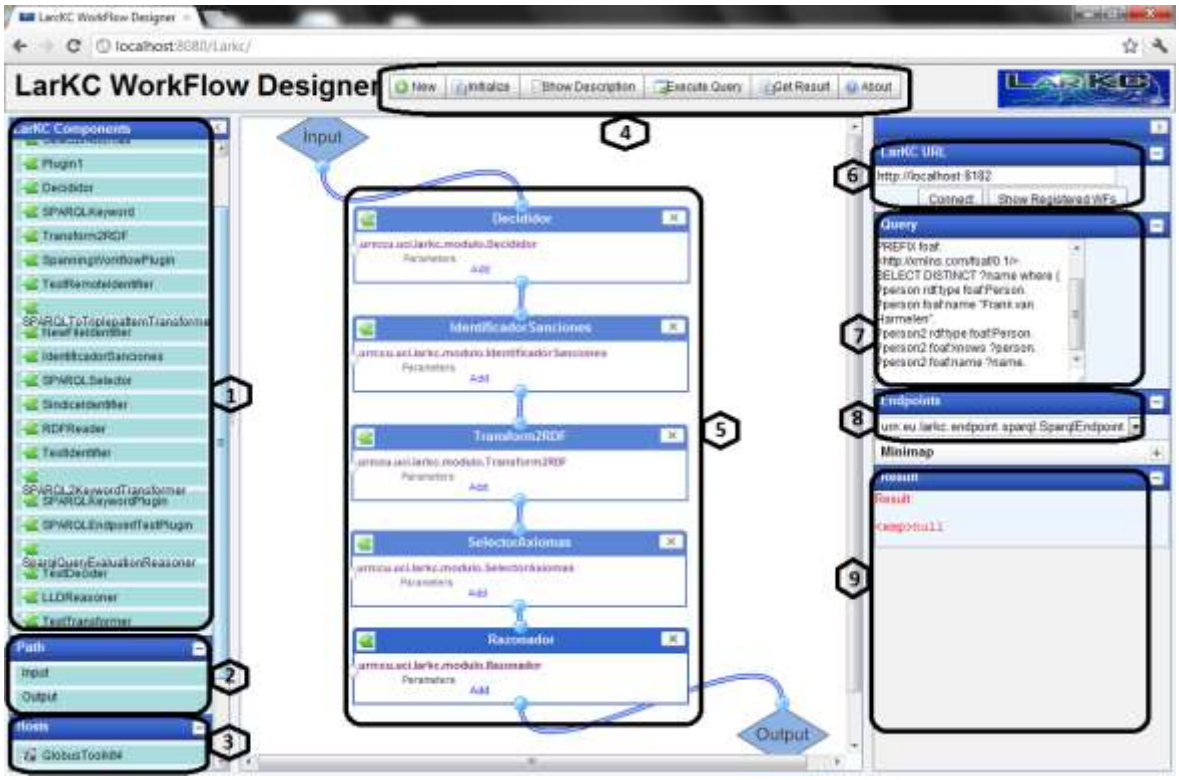

Figura 6. Diseñador de flujos de trabajo de LarKC

1. Se muestra una lista de todos los módulos creados, los cuales están disponibles para conformar un flujo de trabajo.

2. Se utiliza para definir el módulo de entrada y de salida.

3. Se muestra una lista de todas las planillas de host remotos disponibles.

4. Es una barra de herramientas compuesta por varias funcionalidades como:

    - Nuevo *(New)*: se utiliza para crear un nuevo flujo de trabajo.
    - Inicializar *(Initialize):* tiene como función inicializar el flujo de trabajo.
    - Mostrar descripción *(Show Description):* muestra la descripción del flujo de trabajo ejecutado en formato TURTLE.
    - Ejecutar consulta *(Execute Query):* ejecuta la consulta representada en SPARQL.

- Obtener resultado *(Get Result):* muestra el resultado obtenido de la ejecución del flujo de trabajo, representado en formato XML.

5. Es donde se crean los flujos de trabajos de forma visual, su composición resulta muy fácil ya que para conformarlos solo hay que arrastrar los componentes y soltarlos.
6. Es donde se realiza la conexión con la plataforma y refleja la URI mediante la cual se podrá acceder a la interfaz de administración
7. Es donde se va a realizar la consulta en SPARQL.
8. Se muestran todos los puntos finales disponibles de los cuales se elegirán los necesarios para dar respuesta a la consulta.
9. Es donde se visualizarán los resultados de la consulta en formato XML.

Entorno de desarrollo integrado (IDE)

Eclipse: es un entorno de desarrollo integrado de código abierto y multiplataforma. Es una potente y completa plataforma de programación, de desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones en Java. No es más que un entorno de desarrollo integrado (IDE) que está compuesto por herramientas y funciones necesarias para un buen desarrollo, recogidas además en una atractiva interfaz que lo hace fácil y agradable de usar. La compilación es en tiempo real. Permite añadir plugins para pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, test y refactorización [41].

Lenguajes

- *Java*: es un lenguaje de propósito general, concurrente, basado en clases y orientado a objetos. Su diseño fue concebido para que los programadores puedan lograr fluidez con el lenguaje. Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos. Una de las características más importantes es que posee una arquitectura neutral, es decir, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) pude ejecutar ese código, sin importar en modo alguno la máquina en que ha sido

generado. Más allá de la portabilidad básica por ser una arquitectura independiente, implementa otros estándares de portabilidad para facilitar el desarrollo [42].

- *SPARQL*: se trata de un lenguaje estandarizado para la consulta de grafos RDF. Al igual que sucede con SQL, es necesario distinguir entre el lenguaje de consulta y el motor para el almacenamiento y recuperación de los datos. Por este motivo, existen múltiples implementaciones de SPARQL, generalmente ligados a entornos de desarrollo y plataformas tecnológicas. En un principio únicamente incorpora funciones para la recuperación de sentencias RDF. Sin embargo, algunas propuestas también incluyen operaciones para el mantenimiento (creación, modificación y borrado) de datos [43].

- *RDF*: es la base de la mayoría de los lenguajes ontológicos de la actualidad, se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto (conocidas en términos RDF como tripletas). El sujeto es el recurso, es decir, aquello que se está describiendo. El predicado es la propiedad o relación que se desea establecer acerca del recurso y por último, los objetos son los valores que se le asignan a las propiedades mediante los cuales se establecen relaciones. La combinación de RDF con otras herramientas como RDF Schema y OWL permite añadir significado a las páginas [44].

- *OWL*: en la actualidad es el lenguaje que más predomina en el mundo para representar ontologías, que permite publicar y compartir datos mediante el uso de estas, es un lenguaje basado en RDF aunque expresa relaciones más complejas. Posee tres variantes OWL Lite que solo utiliza restricciones simples y una cardinalidad de 0 a 1. OWL DL permite una máxima expresividad sin perder el uso de algoritmos de razonamiento y OWL Full que mantiene su compatibilidad con RDF y posibilita el aumento del vocabulario definido anteriormente [45].

- *TURTLE*: es un formato de serialización de RDF, como una sintaxis textual de RDF que permite a los grafos estar completamente escritos en compacto y en texto de forma natural, con las abreviaturas comunes para patrones de usos y tipos de datos. La sintaxis RDF/XML tiene ciertas restricciones impuestas por XML y el uso de espacios de

nombres XML que impiden la codificación de todos los grafos RDF, estas restricciones no se aplican a TURTLE. Todo RDF escrito en TRUTLE debe ser usado dentro la de la consulta en SPARQL, que utiliza una sintaxis de estilo Turtle/N3 para los patrones de triples y tripletas RDF. Esto posibilita el uso de RDF escrito en TURTLE para permitir la formación de consultas [46].

Herramientas de modelado

Visual Paradigm: es una herramienta CASE para el modelado UML muy potente, gratuita, fácil de instalar, utilizar y actualizar. Soporta el ciclo de vida completo de desarrollo de *software*, análisis y diseño orientado a objetos, construcción, pruebas y despliegues. Permite dibujar diversos tipos de diagramas UML, revertir y generar código fuente desde los diagramas UML. Incluye los objetos más recientes de este lenguaje, además de diagramas de casos de uso, diagramas de clase, diagramas de componentes. Conjuntamente soporta Java, C++, DotNet Exe/dll, XML, XML Schema y Corba IDL, ofrece soporte para Rational Rose, integración con Microsoft Visio, además permite generar reportes y documentación en HTML/PDF [47].

Servidores de Aplicaciones

Servidor Web Apache Tomcat 6.0.x: Tomcat es un servidor web con soporte de servlets escrito totalmente en Java y portable a cualquier plataforma. El motor de servlets del mismo a menudo se presenta en combinación con el servidor web apache. En sus inicios existió la percepción de que su uso de forma autónoma era solo recomendable para entornos de desarrollo, con requisitos mínimos de velocidad y gestión de transacciones, pero hoy en día ya no existe esa idea y es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad [48].

2.3 Requerimientos, restricciones de diseño y atributos de calidad

A continuación se mencionan los requerimientos, restricciones y atributos de calidad de la capa dominio de usuario que tienen un impacto significativo en la arquitectura.

Requerimientos

Los requerimientos serán los siguientes:

- Los componentes identificadores deberán reducir el alcance de una tarea de razonamiento, dada una consulta deben ser capaces de identificar toda la información que puede ser usada para resolverla, defiendo solamente aquellos documentos potencialmente relevantes.
- Los componentes transformadores deberán realizar transformación de los datos de una representación a otra.
- Los componentes selectores se encargarán de la selección que identifica las declaraciones de entrada que se deben utilizar para el razonamiento hacia una respuesta a la consulta de los usuarios.
- Los componentes razonadores, deberán implementar cualquiera de los tipos de razonamientos.
- El decididor deberá gestionar, controlar y si es necesario generar nuevos flujos de trabajo en tiempo de ejecución.
- La implementación de un flujo de trabajo deberá realizar una recuperación de información basada en el uso de la semántica.
- Para la implementación de los componentes se deberá realizar en la medida de lo posible con métodos heurísticos.

Restricciones de diseño

- Solo se podrá realizar consultas en SPARQL.
- El envío de mensajes entre componentes será en RDF.

Atributos de calidad

Los atributos de calidad forman parte de los requerimientos no funcionales de la aplicación y como tal deben ser específicos para llenar alguna necesidad dada o llegar a una meta específica.

Constructibilidad:

- La arquitectura debe poder ser diseñada en un plazo de 6 meses.

Confiabilidad:

- La información a la que accede cada componente debe ser totalmente confiable.

Eficiencia:

- La solución para cada componente no debe afectar el funcionamiento de los recursos del sistema dicho sea, utilización del disco duro, memoria, procesador.

Interoperabilidad:

- Los componentes deberán ser capaces de dejar que otros sistemas trabajen con ellos logrando la comunicación entre sistemas o aplicaciones.

2.4 Patrones de diseño

Los patrones de diseños escogidos para esta primera iteración se encuentran dentro los patrones GRAPS y GoF:

GRAPS

- *Bajo acoplamiento*: los componentes no dependen de otros en cuanto a funcionalidad, esto se logra con la arquitectura orientada a componentes.
- *Alta cohesión:* las clases no dependen y no interfieren en la lógica de otra.
- *Experto:* cada clase está especializada para una funcionalidad específica.

GOF

- *Command:* todos los componentes implementan el método InvokeInternal() que recibes los comandos.
- *Factory:* implementado con la familia de DataFactory para crear instancias de las *querys.*

2.5 Vista lógica

La vista lógica de la arquitectura apoya principalmente los requerimientos funcionales, desde el ámbito en donde ellos son proveídos en términos de servicios a sus usuarios. En esta sección el sistema es descompuesto en un conjunto de abstracciones claves, tomadas principalmente del dominio de usuario. Se explotan los principios de la abstracción, encapsulamiento y la herencia. Esta descomposición no es solo por el bien del análisis funcional, sino también sirve para identificar los mecanismos comunes y elementos de diseño a través de las diferentes partes del sistema.

Subsistemas y capas

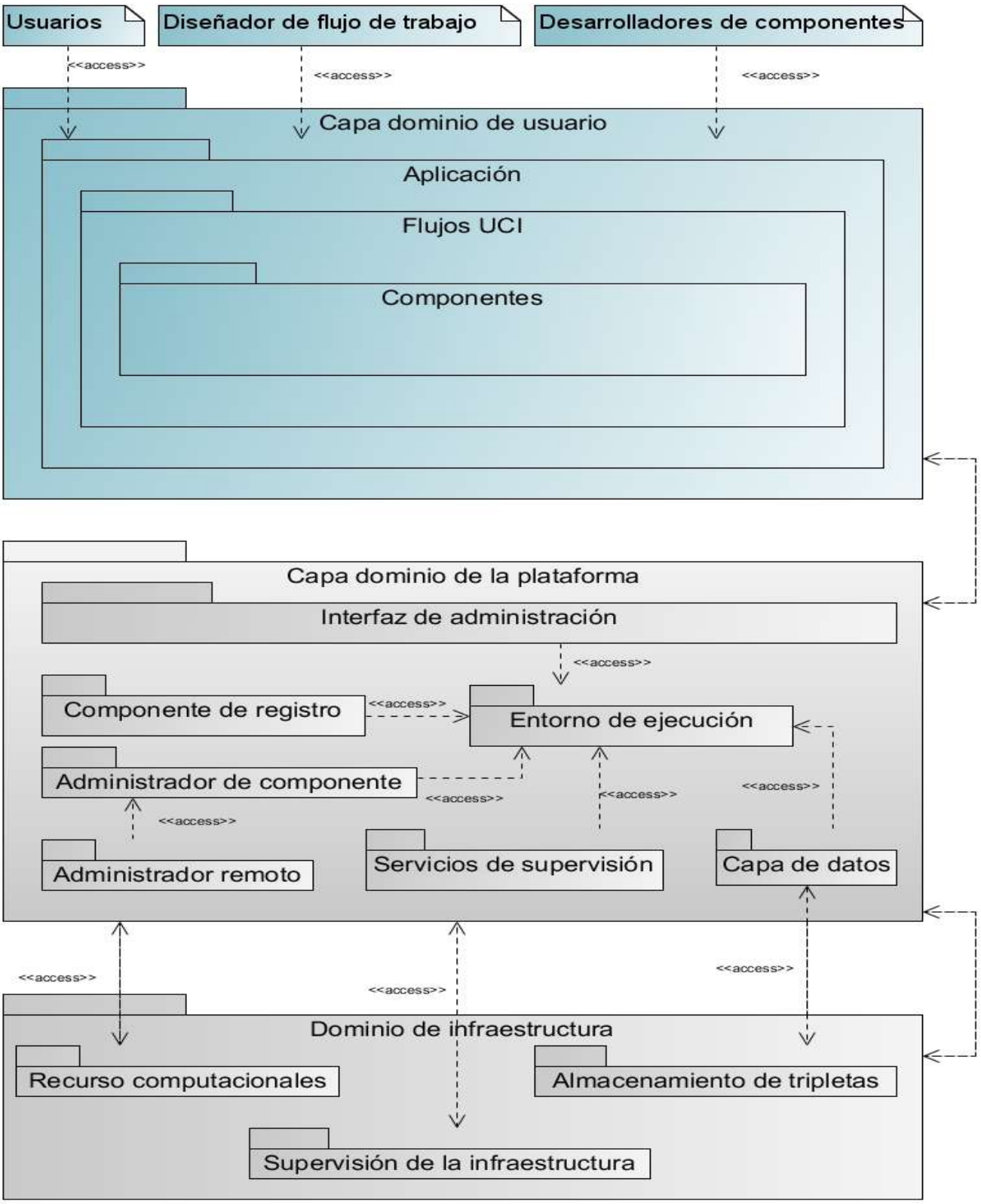
Usuarios
Diseñador de flujo de trabajo
Desarrolladores de componentes
<<access>>
<<access>>
<<access>>
Capa dominio de usuario
Aplicación
Flujos UCI
Componentes
Capa dominio de la plataforma
Interfaz de administración
<<access>>
Componente de registro
<<access>>
Entorno de ejecución
Administrador de componente
<<access>>
<<access>>
<<access>>
<<access>>
Administrador remoto
Servicios de supervisión
Capa de datos
<<access>>
<<access>>
<<access>>
Dominio de infraestructura
Recurso computacionales
Almacenamiento de tripletas
Supervisión de la infraestructura

Figura 7. Representación gráfica de la vista de paquetes

La figura mostrada anteriormente, refleja una vista simplificada de cómo queda estructurada la arquitectura propuesta para dar solución al problema de investigación; centrándose solo en la primera capa que corresponde a la de *Dominio de Usuario* la cual está compuesta por los siguientes paquetes:

Capa de aplicación: constará con una interfaz en la que los usuarios podrán interactuar con el sistema o sea, posibilitará al cliente y a la aplicación establecer la comunicación.

Capa flujos UCI: contendrá todos los procesos sustantivos de la UCI de manera estructurada, cada flujo de trabajo estará representado por componentes que se conectarán entre sí para responder a una tarea específica, la cual es realizada por el usuario.

Capa de componente: contendrá todos los componentes necesarios para conformar un flujo de trabajo.

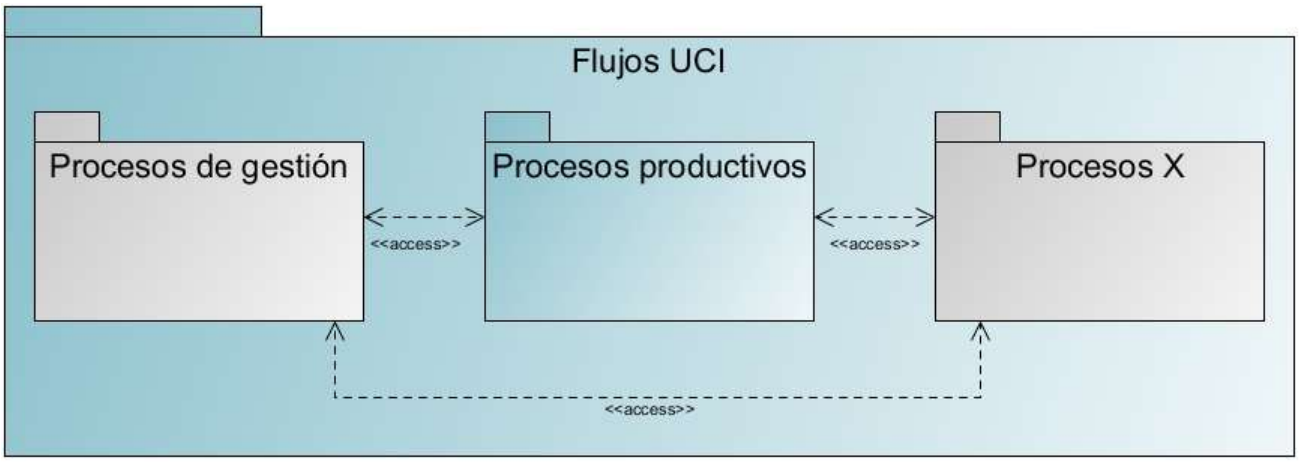

Figura 8. Vista ampliada de flujos UCI

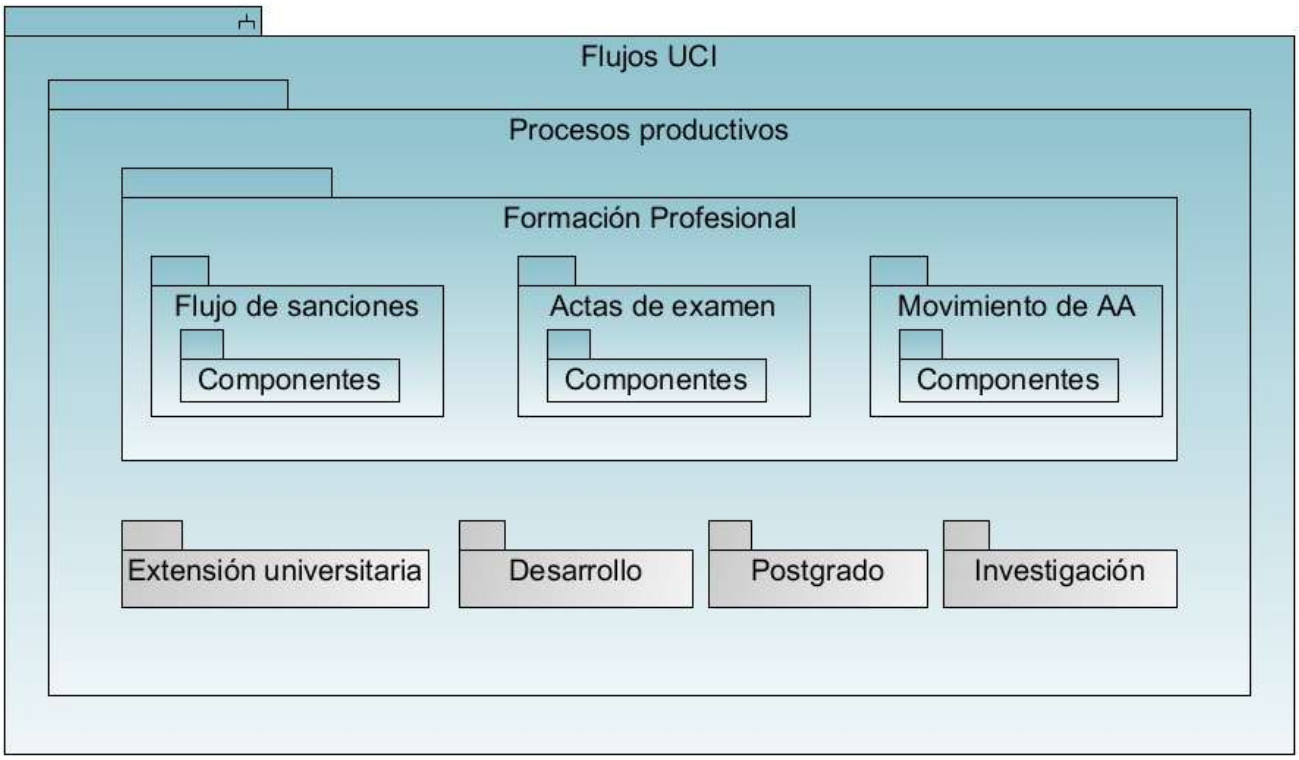

Figura 9. Vista ampliada de procesos productivos

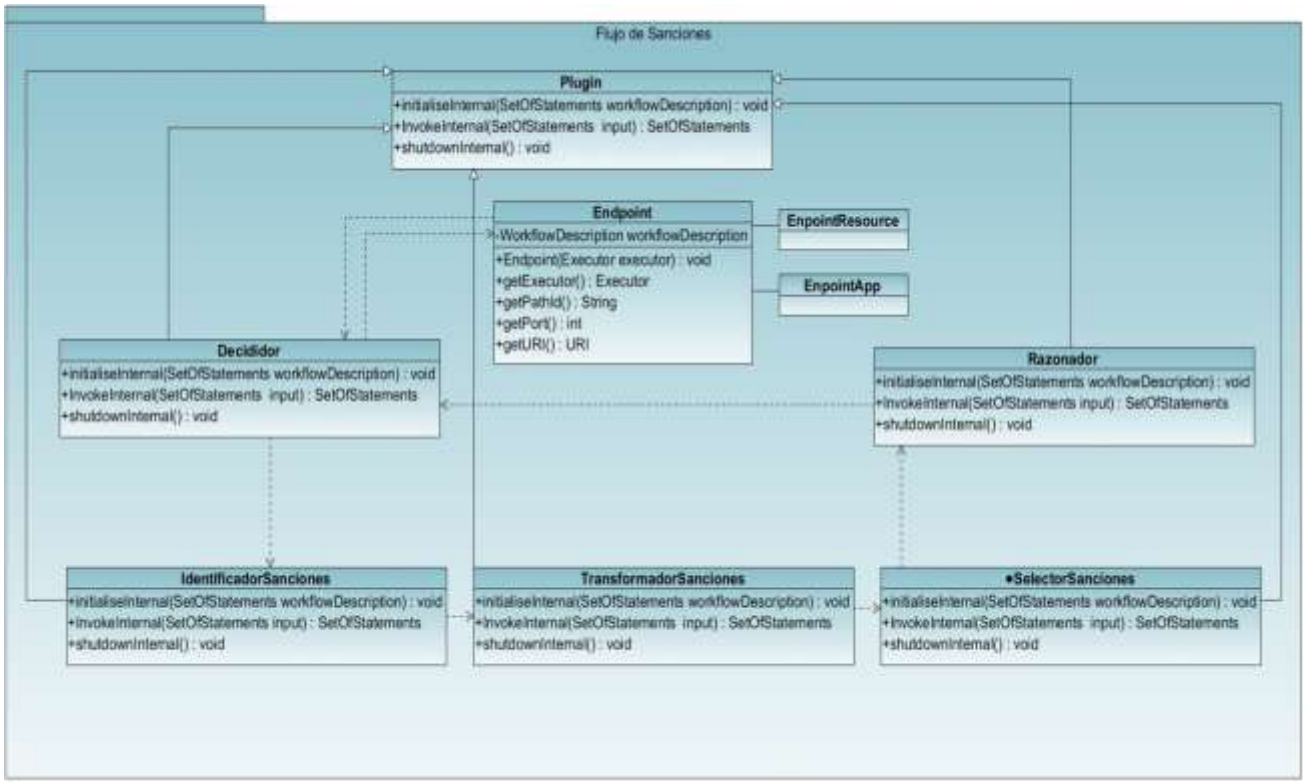

Figura 10. Vista ampliada flujo de sanciones

- *Endpoint:* es el encargado de enviar y recibir la consulta entrada por el usuario.

- *Decididor:* evalúa la consulta en SPARQL para determinar en tiempo de ejecución que flujo utilizar para dar respuesta a la consulta, ejecutando una descomposición de esta, realizando un árbol sintáctico del cual se obtienen las palabras claves para luego conformar el flujo de trabajo definiendo los endpoints necesarios para esto y cuando el resultado es devuelto a él, verifica que esté correcta la respuesta.

- *IdentificadorSanciones:* recibe el objeto de una tripleta con una lista de palabras clave y accede a un sistema externo que puede o no tener una semántica implícita identificando y recuperando todos los recursos que puedan ser relevantes para responder a la consulta, estos son extraídos de fuentes de datos heterogéneas que se encuentren de manera desestructurada y estructurada.

- *TransformadorSanciones:* transforma todos los documentos antes identificados en RDF, conformando un modelo de datos compuestos por grafos RDF.
- *SelectorSanciones:* selecciona una muestra de grafos en RDF del modelo de datos, la cual es usada para realizar el razonamiento.
- *Razonador:* realiza inferencia sobre la muestra anteriormente seleccionada a través un motor de inferencia, generando el resultado final para responder a la consulta, el cual es enviado al decididor.

2.6 Vista de despliegue

La vista de despliegue se refiere a la implementación en módulos y fragmentación en muchas capas. Colecciona las categorías de clases y grupos, describe el mapeo del *software* en el *hardware* y toma en cuenta los requerimientos funcionales del sistema, tales como, confiabilidad, respuesta y escalabilidad.

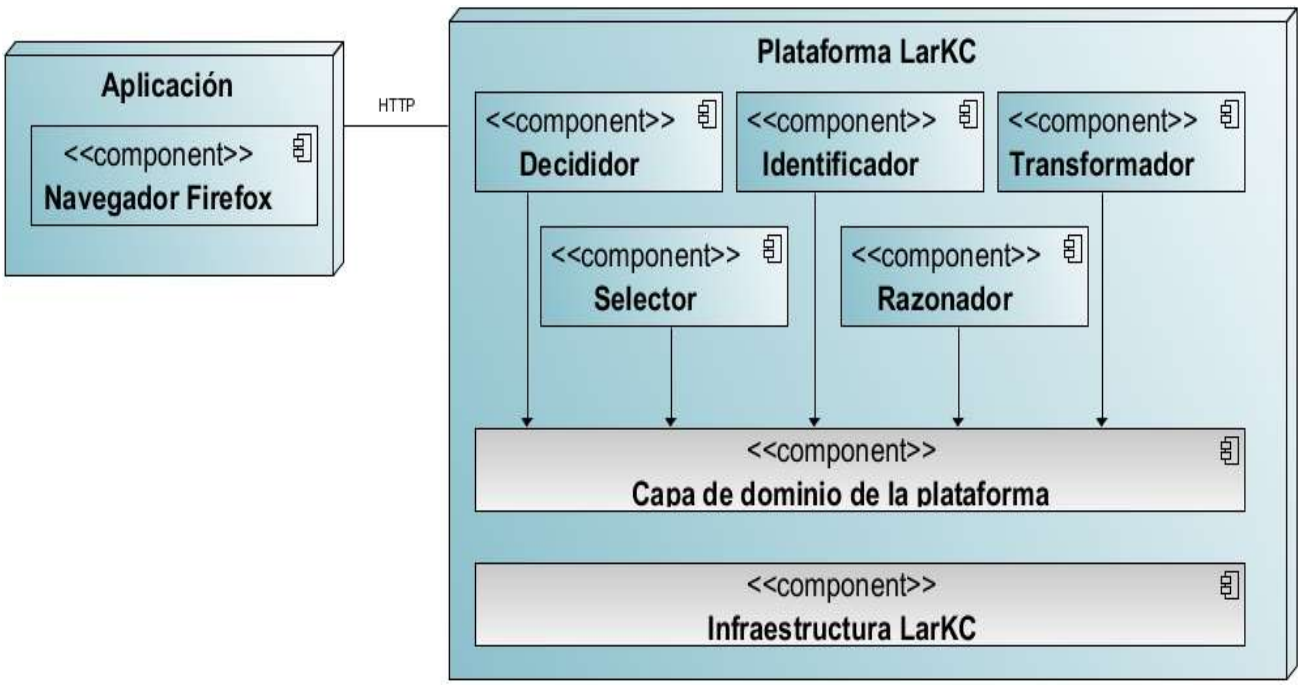

Figura 11. Representación gráfica de la vista de despliegue

Estación cliente

Se refiere a las estaciones de trabajo que el usuario utilizará para acceder a la aplicación y transcribir sus datos. En la estación cliente debe residir un navegador web.

Plataforma

Se refiere al núcleo del sistema, donde se establece la lógica de la aplicación para lograr la conexión del sistema con la estación cliente, se utiliza HTTP como protocolo de comunicación. Dentro del nodo de la plataforma residen los componentes que sostendrán a esta plataforma.

Requisitos mínimos de hardware para el despliegue en un ambiente de prueba

Nodos físicos de despliegue	Requerimientos mínimos de Hardware
Estación cliente	64 MB de memoria RAM
Plataforma	1.0 GB de memoria RAM Microprocesador Intel Pentium o compatible 10 GB de Capacidad de Disco Tarjeta de red 10/100bps

Tabla 4. Requerimientos mínimos de Hardware[21]

2.7 Vista de implementación

La vista de implementación contiene la organización de los módulos en términos de paquetes y capas, pueden incluirse también la trazabilidad de la vista lógica. Es representada por un diagrama de componentes o especificaciones de paquetes que son básicamente un subconjunto del modelo de despliegue. Se obtiene cuando se realiza el flujo de trabajo de RUP de implementación.

[21] La funcionalidad de esta arquitectura se está realizando a modo de prueba.

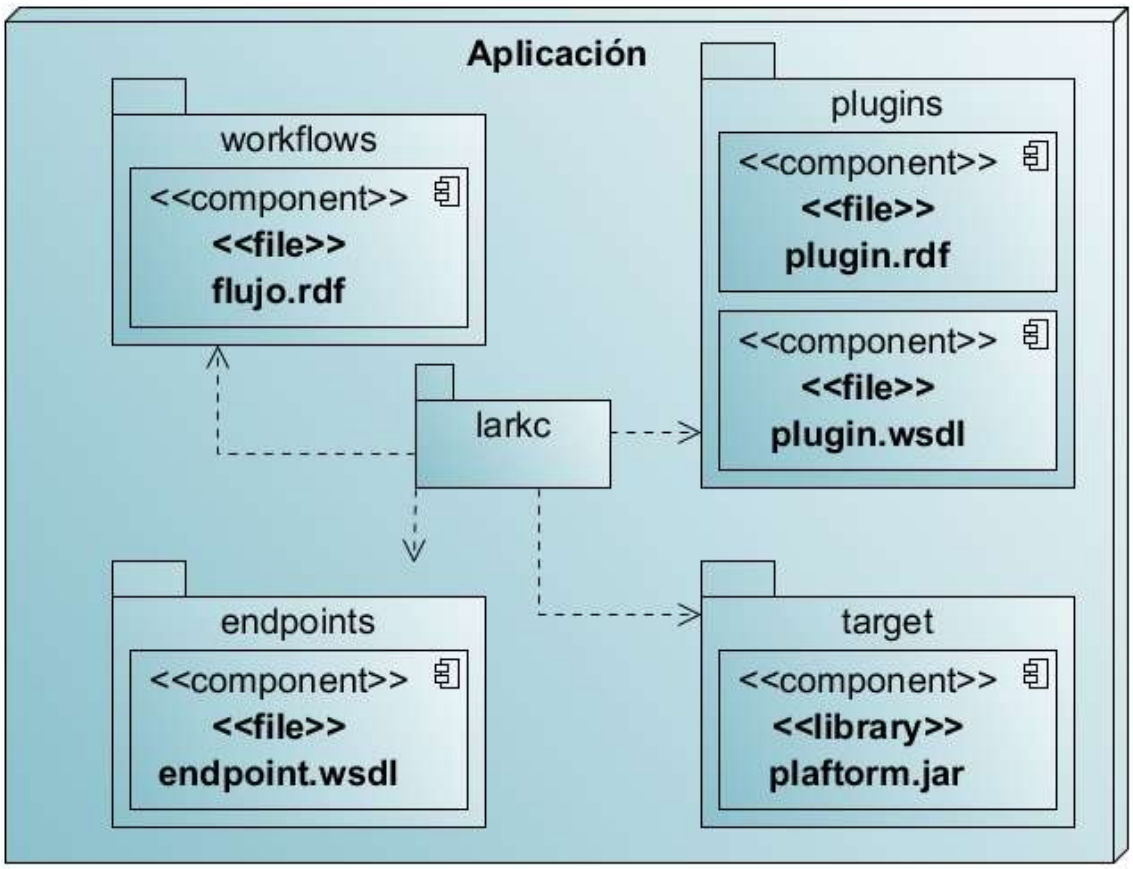

Figura 12. Representación gráfica de la vista de implementación

Descripción de los paquetes

- *larkc:* es el directorio que contiene la implementación particular del sistema.
- *plugins:* es el directorio de componentes donde están habilitados todos los módulos listos para conformar un flujo de trabajo.
- *target:* este directorio contiene las librerías necesarias para el desarrollo de los componentes.
- *workflows:* en el directorio workflows se encuentran todos los flujos de trabajos en formato TURTLE.

- *endpoints:* es el directorio que contiene todos los endpoints creados para la ejecución de un flujo determinado.
- *plugin.rdf:* es el archivo que contiene el módulo de la ontología, que provee un vocabulario extenso, compartido con el vocabulario de flujo de trabajo, su ubicación se encuentra en el paquete plugin *src/main/resources/plugin.rdf.*
- *plugin.wsdl:* es el archivo encargado de describir el tipo de servicio y la funcionalidad del plug-in, su ubicación se encuentra en el paquete plugin *src/main/resources/plugin.wsdl.*
- *platform.jar:* es el archivo que contiene las bibliotecas con propósitos generales, que se almacenan dentro del paquete target, *target/plugin.cu.uci.larkc.ejemplo.jar.*
- *endpoint.wsdl:* es el archivo encargado de describir el tipo de servicio y la funcionalidad del endpoint.
- *flujo.rdf:* es el archivo donde se encuentra reflejado un flujo de trabajo en formato TURTLE, se encuentra en el paquete workflows.

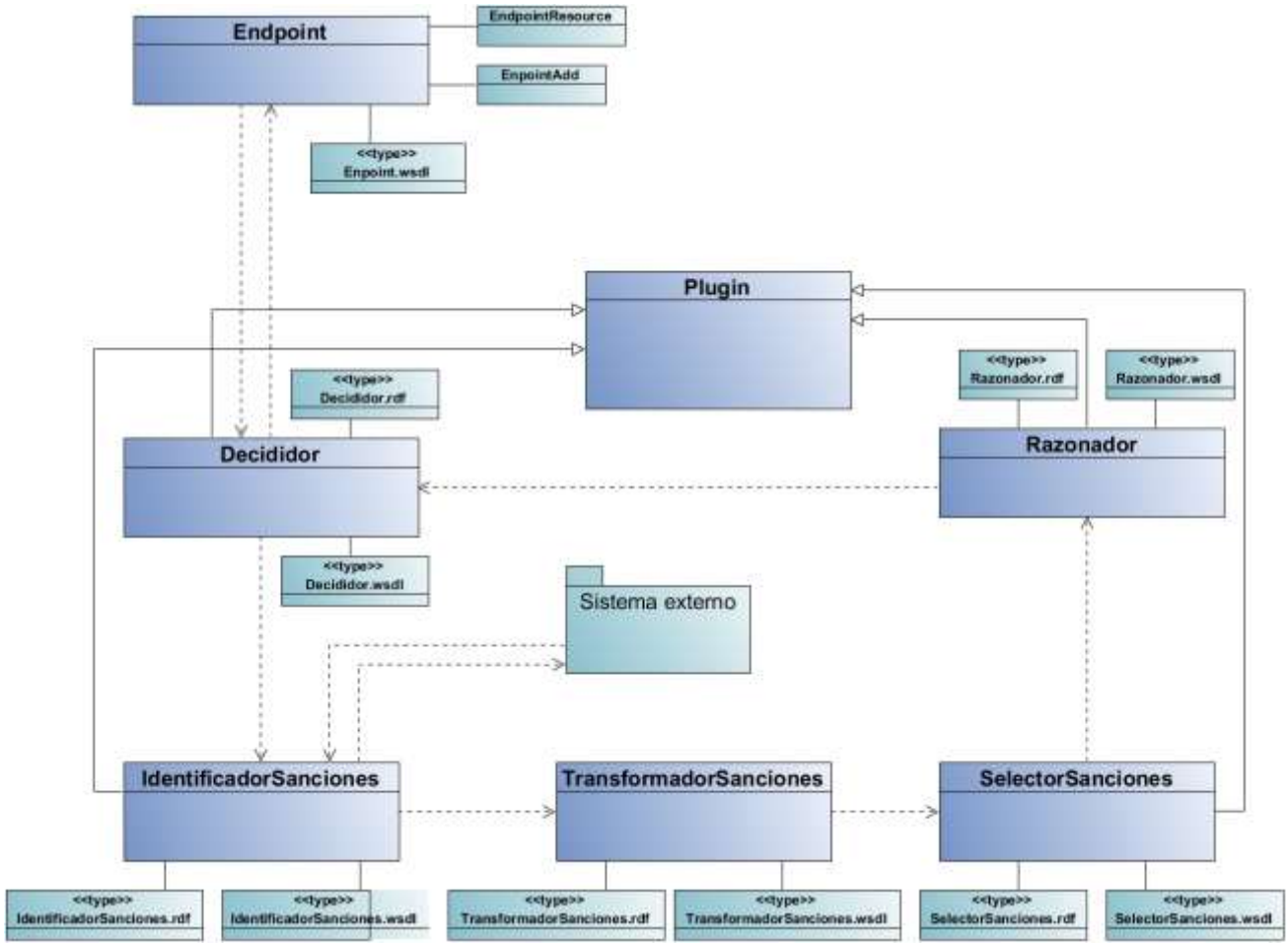

Figura 13. Representación gráfica del flujo de sanciones

2.8 Análisis del capítulo

El diseño de una arquitectura de un software proporciona una visión global del sistema a construir y marca las decisiones de diseño. La concepción de esta involucra la definición de herramientas, lenguajes, patrones de diseño, atributos de calidad, entre otros. Todo esto a través de varias vistas que permiten tener una representación en diversos niveles para su compresión y posterior desarrollo.

Incorporar al diseño arquitectónico el componente:

Identificador permite reducir el alcance de una tarea de razonamiento, y define los documentos relevantes para resolver una consulta.

Transformador permite realizar transformaciones de datos de una representación a otra.

Selector facilita la selección de las declaraciones de entrada que se deben utilizar para el razonamiento.

Razonador permite ejecutar diferentes tipos de razonamientos como el inductivo, deductivo, entre otros.

Decididor permite la gestión y el control de los flujos de trabajo.

CAPÍTULO 3
Evaluación de la arquitectura propuesta

La evaluación de la arquitectura constituye una actividad de vital importancia, ya que proporciona un alto grado de confianza y seguridad, no solo en el producto sino en los resultados que se obtienen en implantarlo. Realizar una evaluación es un requisito que debe ser cumplido con rigor. Para lograrla se requiere tiempo y esfuerzo y la conclusión de que la arquitectura es correcta, debe estar respaldada por evidencias recolectadas por un conjunto de especialistas a partir de actividades planificadas de un método de evaluación.

3.1 Evaluación y aceptación de la arquitectura propuesta

La razón principal que justifica la evaluación de una arquitectura se debe a la necesidad de demostrar que la solución (el diseño) es viable y correcta para resolver el objetivo (los requisitos). En la construcción de la arquitectura de un software es donde se toman las decisiones más importantes del proyecto. La arquitectura constituye la materialización temprana de las decisiones más importantes en la construcción de un proyecto. El prototipo ejecutable debe demostrar que la solución ya es madura y que las decisiones tomadas son efectivas para resolver el problema. La arquitectura debe dar garantías de que la solución diseñada es realizable dentro de las restricciones de tiempo, personal y presupuestos, o sea, que el proyecto es viable.

Método de la evaluación

Como resultado del análisis de la comparación de diferentes métodos, se observó que el Architecture Tradeoffs Analysis Method (ATAM), tiene un conjunto de pasos, un equipo de evaluación y un conjunto de salidas mejor definidas y no presenta ninguna restricción con respecto a la característica de calidad a evaluar. El MECABIC[22] está inspirado en ATAM, aunque con el objetivo de facilitar su aplicación sobre arquitecturas de software basadas en componentes. Incluye en algunos de sus pasos un enfoque de integración de elementos de

[22] Método de evaluación de la calidad de arquitecturas basadas en la integración de componentes.

diseño, inspirado en la construcción de partes arquitectónicas adoptado por el ABD (Architecture Based Design). Además se propone un árbol de utilidad inicial basado en el modelo de calidad ISO-9126 instanciado para Arquitectura de *Software* propuesto por Losavio. Adoptar este modelo por parte del MECABIC permite concentrarse en características que dependen exclusivamente de la arquitectura y al ser un estándar facilita la correspondencia con características de calidad consideradas por los métodos estudiados.

El equipo de la evaluación

El equipo de la evaluación está compuesto por tres grupos de trabajo. La siguiente tabla muestra la composición del equipo de desarrollo

Equipo	Definición	Fases
Arquitectos	Responsables de generar y documentar una arquitectura de *software* para el sistema estudiado	Todas
Evaluador	Integrado por personas especialistas en asuntos de calidad quienes guiarán el proceso de evaluación de la arquitectura	Todas
Relacionados	Son las personas involucradas de alguna manera con el sistema, programadores, usuarios, gerentes, entre otros	Fases 1, 3, 4.

Tabla 5. Composición del equipo de desarrollo

Instrumentos y técnicas de evaluación

Para la especificación de la calidad se hace uso de un árbol de utilidad, el cual tiene como nodo raíz la “bondad” o “utilidad” del sistema. En el segundo nivel del árbol se encuentran los atributos de calidad. Las hojas del árbol de utilidad son escenarios, los cuales representan mecanismos mediante los cuales extensas (y ambiguas) declaraciones de cualidades son hechas específicas y posibles de evaluar. La generación del árbol de calidad incluye un paso

que permite establecer prioridades. Para el análisis de la arquitectura se utiliza la técnica de escenarios, soportada por cuestionarios, para identificar lo que desean los participantes [49].

3.2 Fases del método

El método MECABIC presenta cuatro fases, a continuación se explica que se realizó en cada una:

En la *primera fase*, se dió a conocer el método, la organización y el sistema a todos los involucrados en la evaluación.

En la *segunda fase,* se llevó a cabo la investigación y el análisis, donde se determinó de qué manera se iba a estudiar la arquitectura. Se pidió a los stakeholders que expresaran de una manera precisa qué escenarios de calidad se deseaban y se analizó si la arquitectura era apropiada o se requerían modificaciones.

MECABIC proporciona un árbol de utilidad inicial específico para la arquitectura de software basada en componentes, a partir del cual se seleccionan un conjunto de escenarios de interés. Este árbol está basado en el modelo de calidad ISO 9126-1 para arquitecturas de *software* propuesto por Losavio.

Árbol de utilidad

Característica	Subcaracterística	Escenario
Funcionalidad	Interoperabilidad	El sistema posee componentes capaces de leer datos provenientes de otros sistemas
		El sistema posee componentes capaces de producir datos para otro sistema
	Precisión	Los resultados ofrecidos por los componentes del sistema son exactos
		La comunicación entre los componentes no altera la exactitud de los datos

	Seguridad	El sistema detecta la actuación de un intruso e impide acceso a los componentes que manejen información sensible
		El sistema asegura que los componentes no pierdan datos ante un ataque (interno o externo)
	Obediencia (Compliance)	Los componentes respetan un estándar de fiabilidad
		La comunicación entre los componentes no viola los estándares de fiabilidad
Fiabilidad	Madurez	Los componentes del sistema manejan entradas de datos incorrectas
	Tolerancia a fallas	Todas las operaciones ejecutadas por los componentes se realizan correctamente bajo condiciones adversas
	Capacidad de restablecimiento o recuperación	Los componentes del sistema no fallan bajo ciertas condiciones especificadas
		Ante problemas con el ambiente un subconjunto determinado de los componentes puede continuar prestando sus servicios
Eficiencia	Tiempo de comportamiento	El sistema debe recibir los servicios de sus componentes en el transcurso de un tiempo indicado
	Recursos utilizados	Los componentes pueden compartir recursos adecuadamente
		El sistema controla que ningún componente se quede sin recursos cuando los necesita
Mantenibilidad	Habilidad de cambio, estabilidad, prueba	Es posible verificar el estado de los componentes del sistema
		El sistema brinda facilidad para adaptar un componente

		El sistema debe facilitar la sustitución/adaptación de un componente
Portabilidad	Adaptabilidad	El sistema debe continuar funcionando correctamente aun cuando los servicios de los componentes provistos por el ambiente varíen
	Capacidad de Instalación	Los componentes pueden instalarse fácilmente en todos los ambientes donde debe funcionar
	Co-existencia	Los componentes manejan adecuadamente recursos compartidos del sistema

Tabla 6. Características a medir por el grupo de especialistas [49]

Seleccionándose los siguientes escenarios:

Precisión:

- La comunicación entre los componentes no altera la exactitud de los datos.
- Los resultados ofrecidos por los componentes del sistema son exactos.

Interoperabilidad:

- *El sistema posee componentes capaces de producir datos para otro sistema.*
- El sistema posee componentes capaces de leer datos provenientes de otros sistemas.

Seguridad:

- El sistema detecta la actuación de un intruso e impide acceso a los componentes que manejen información sensible

Madurez:

- Los componentes del sistema manejan entradas de datos incorrectas

Tolerancia fallas:

- Todas las operaciones ejecutadas por los componentes se realizan correctamente bajo condiciones adversas

Habilidad de cambio, estabilidad, prueba:

- Es posible verificar el estado de los componentes del sistema

La *tercera fase* es de prueba, consistió en la revisión de la *segunda fase* y en ella participaron todos los grupos. La fase anterior muestra la relación entre las principales características por las cuales el grupo de expertos puede medir un prototipo ejecutable de arquitectura. Para realizar esta medición se formula un conjunto de preguntas clasificables dentro de cada característica o subcaracterística.

Característica	Subcaracterística	Preguntas del Análisis
Funcionalidad	Precisión	¿Puede la comunicación entre los componentes introducir imprecisiones en los servicios ofrecidos?
	Interoperabilidad	¿Los componentes pueden ser usados por otro sistema?
		¿Los componentes pueden leer datos de otros sistemas?
	Seguridad	¿Se puede detectar la actuación de un intruso?
Fiabilidad	Madurez	¿Los componentes del sistema manejan las entradas de datos incorrectas?
	Tolerancia a fallas	¿Cómo se detecta el funcionamiento incorrecto de un

		componente?
Mantenibilidad	Habilidad de cambio, estabilidad, prueba	¿Cómo se detecta el funcionamiento incorrecto de un componente? ¿Cómo se verifica el funcionamiento correcto de un componente?

Tabla 7. Clasificación de las preguntas de acuerdo a las diferentes características

Luego de elaboradas las preguntas, se responden para luego contrastar en dependencia de los resultados obtenidos por la evaluación a los especialistas.

¿Puede la comunicación entre los componentes introducir imprecisiones en los servicios ofrecidos?:

- No, porque la comunicación entre componentes no incide sobre los datos que se envían. Cuando se realiza una llamada a un componente o plug-in es a través de un endpoint descrito formalmente en un fichero WSDL. Esto crea un canal de comunicación interno de LarKC que abstrae el cuerpo del mensaje del modo de comunicación, permitiendo que mediante el uso del Patrón *command pattern* la plataforma gestione el manejo de la comunicación entre un componente y otro, evitando que se introduzcan impresiones entre los servicios. Esto no dice que si falla la comunicación no fallen los servicios ofrecidos, sino que la comunicación no origina imprecisiones.

¿Los componentes pueden ser usados por otro sistema?:

- Sí, siempre que se use la misma base tecnológica, con la debida configuración y el correcto envío de parámetros es posible.

¿Los componentes pueden leer datos de otros sistemas?:

- Si, la plataforma le ofrece a cada componente muchas herramientas para la comunicación externa, como acceso remoto por el protocolo http, consulta a bases de datos semánticas, a repositorios rdf, cliente de servicios SOAP entre otros.

¿Se puede detectar la actuación de un intruso?:

- Creando endpoints personalizados se puede acceder a los encabezados de las peticiones por el protocolo HTTP, lo cual permitirá usar mecanismos de control de acceso como servicios autenticados, firma de cabeceras de peticiones, o cualquier otro mecanismo para garantizar la autenticidad del usuario.

¿Los componentes del sistema manejan las entradas de datos incorrectas?:

- Si, validando los argumentos desde un método de acceso HTTP desde un endpoint específico o evaluando la consulta SPARQL.

¿Cómo se detecta el funcionamiento incorrecto de un componente?:

- Se detecta en tiempo de prueba mediante el empleo de pruebas unitarias de otra forma, habría que implementar un mecanismo de *feedback*.

¿Cómo se verifica el estado de una comunicación entre componentes?:

- No tiene soporte. Queda a recomendación.

¿Cómo se verifica el funcionamiento correcto de un componente?:

- Mediante los logger definidos en cada componente, los cuales mediante la API se verifica que se inicializan y cumplen con su funcionalidad.

¿Es viable la arquitectura propuesta?[23]:

[23] Esta es una pregunta de comprobación, no se incluye en ningún escenario pues es para verificar con certeza si la arquitectura es viable o no.

- Sí.

4. En la *última fase* se lleva a cabo los resultados de la prueba.

Resultado de la evaluación

Preguntas	Especialistas		
	Arquitecto de BISMED	Arquitecto de CICPC	Arquitecto de SCADA
¿Puede la comunicación entre los componentes introducir imprecisiones en los servicios ofrecidos?	No, porque la comunicación no tiene que ver con la implementación interna que tenga el componente	Probablemente sí	Debe estar definido bien el dominio del plugin y las tareas a realizar. Por otro lado, la comunicación lo que afectaría el servicio pero no causaría imprecisiones
¿Los componentes pueden ser usados por otro sistema?	Sí, siempre y cuando este otro sistema satisfaga las necesidades de entrada que necesita el componente	Sí	Sí
¿Los componentes pueden leer datos de otros sistemas?	Sí	Sí, el sistema proporciona mecanismos para hacer enlaces a otros sistemas	Sí, se ve claramente que el identificador se debe comunicar a un sistema para poder extraer la información

¿Se puede detectar la actuación de un intruso?	No con lo que tiene actualmente, en el futuro sí	No, no tiene mecanismos de detección de intruso, pero en esta primera fase no la necesitan	Pudieran tenerlo
¿Los componentes del sistema manejan las entradas de datos incorrectas?	Sí, a través de los endpoint	Sí, a través de las validaciones	Sí, si se le realizan validaciones a ellos
¿Cómo se detecta el funcionamiento incorrecto de un componente?	A través de pruebas.	Revisando la entrada y salida de los datos en cada componente	Realizando pruebas unitarias
¿Cómo se verifica el estado de una comunicación entre componentes?	No lo hace	No lo hace	No lo hace
¿Cómo se verifica el funcionamiento correcto de un componente?	A través de los log	Utilizando la API para obtener logs	Con los logger
¿Es viable la arquitectura propuesta?:	Sí	Sí	Sí

Tabla 8. Respuestas del análisis de la evaluación

3.3 Análisis del capítulo

MECABIC proporciona un árbol de utilidad basado en escenarios. Permite formular un conjunto de preguntas para el análisis de las arquitecturas basadas en componentes que se realizan a especialistas a través de entrevistas. Como cumplimiento de los objetivos se tiene:

- MECABIC al ser un método para evaluar arquitecturas basadas en componentes es el más adecuado para la evaluación.
- Las preguntas de análisis permiten evaluar la arquitectura demostrando que es viable.
- El prototipo ejecutable confirma la calidad de las decisiones tomadas en la construcción de la arquitectura.

CONCLUSIONES

El proceso de descubrir conocimiento en datos heterogéneos comprende un conjunto de técnicas y herramientas como el LarKC que permiten la extracción de datos en grandes volúmenes de información de forma estructurada, semiestructurada y no estructurada.

La arquitectura basada en componentes permite crear sistemas fácilmente escalables que tengan la reutilización como principio, donde los componentes son construidos como unidades independientes y pueden ser incorporados a otras aplicaciones.

La implementación de un flujo trabajo de la arquitectura y los componentes que lo conforman: identificador, selector, transformador, razonador y decididor, sugieren que introducir la semántica en los procedimientos que rigen la universidad permitirá obtener mejores resultados en las búsquedas.

La evaluación mediante MECABIC muestra que el prototipo ejecutable de la arquitectura puede ser desplegado en un entorno real, facilitando el proceso de búsqueda en la Universidad de las Ciencias Informáticas.

RECOMENDACIONES

Luego de realizada la investigación se propone:

- Implementar todos los componentes de identificación, selección, transformación, razonamiento y el decididor para conformar los flujos del área de formación profesional (ver anexo #2).
- Formar nuevos arquitectos con los conocimientos necesarios para el diseño de las capas de dominio de la plataforma y dominio de infraestructura.
- Realizar una prueba en un ambiente real utilizando los procesos sustantivos de la universidad.

REFERENCIAS BIBLIOGRÁFICAS.

1. **Campell, J.** *El hombre gramatical. Información, entropía, lenguaje y vida.* México D.F : Fondo de Cultura económica, 1989.

2. *Algunas reflexiones sobre el concepto de información y sus implicaciones para el desarrollo de las ciencias de información.* **Camejo, Ivis Goñe.** La Habana, Cuba : ACIMED, 2000, Vols. 8(3):201-7.

3. *Real Academia Española.Diccinario de la lengua española.* Madrid : P.721, 1936.

4. *Lenguaje e información.* **Vizcaya, Alejandro.** s.l. : Ciencia Información, 1997, Vols. 28(2):109-17.

5. **Paez, Urdaneta I.** *Gestión de la Inteligencia, aprendizaje tecnológico y medernización del trabajo informacional. Retos y opurtunidades.* Caracas : Universidad Simon Bolivar, 1992.

6. **M, Morales Morejon.** *Informetría.* s.l. : PROINFO/DICT, 1995.

7. *Información: una nueva propuesta conceptual.* **Angulo, Marcial N.** 4, 1996, Vol. 27. 190-5.

8. *INVESTIGACIÓN CIENTÍFICA, POR QUÉ Y PARA QUÉ INVESTIGAR.* **Noruega, Mg. Jorge Sánchez.** Colombia : s.n., 06-02-2012, Vols. MPI-001.

9. *"La Gestión Integral del Conocimiento y del Aprendizaje". Economía Industrial.* **Andreu, R y Sieber, S.** 326, 2000, págs. 63-72.

10. **Fayyad, U., G. Piatetsky-Shapiro y P. Smyth.** *Data Mining and Knowledge Discovery in Databases: An overview, Communications of ACM.* 1996.

11. **Akmajian, Adrian, Richard Demers, Ann Farmer, and Robert Harnish.** *Linguistics: an introduction to language and communication.* s.l. : Cambridge, MIT Press, 1985.

12. **HAKIMPOUR, F. y GEPPERT, A.** *Resolving Semantic Heterogeneity in Schema Integration: an Ontology Based Approach, FOIS.* 2001. págs. 297-308.

13. **R, Tous.** Data Integration with XML and Semantic Web Technologies, Doctoral. 2006.

14. **Macías, Lyssania y Michán, Layla.** *Los recursos de la Web 2.0 para el manejo de información académica.* Diciembre, 2009.

15. **World Wide Web Consortium W3C.** OWL Web Ontology Language Overview. *World Wide Web Consortium.* [En línea] mayo de 2012. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

16. **Palacios Escalona, Juan Pablo.** *Modelo de Unificación Semántica de Ontologías, aplicado al dominio de los archivos digitales.* s.l. : Departamento de ingeniería de sistemas telemáticos, 2005.

17. **Gruber.** *A Translation Approach to Portable Ontology Specification. Knowledge Acquisition 5.* 1993. págs. 199-220.

18. *Estrategias de Paralelización de Algoritmos de Razonamiento para Ontologías Biomédicas.* **Eduardo J. Cepas, Gines D. Guerrero, Jose M.** 2011.

19. Apache™ Hadoop™. [En línea] The Apache Software Foundation, 2011. http://hadoop.apache.org/.

20. MaRVIN:a distributed platform for massive RDF inference. [En línea] mayo de 2009. http://www.larkc.eu/marvin/.

21. **Matthias Assel, Alexey Cheptsov, Georgina Gallizo, Emanuele Della Valle.** *Large Knowledge Collider - a Service-Oriented Platform.* s.l. : 7th Framework Programme, 2011. ICT-FP7-215535.

22. *Annex I - "Description of Work".* **Atanas Kiryakov, Michael Witbrock, Ning Zhong , Emanuele Della Valle, Frank van Harmelen.** s.l. : EU 7th framework, 2007, Vols. FP7-ICT-2007-1. FP7-ICT-2007-1.

23. **Georgina Gallizo, Mick Kerrigan, Barry Bishop,.** *D5.3.2 Overall LarKC architecture.* 2009. D5.3.2.

24. *Metodología para la Extracción del Conocimiento Empresarial a partir de los Datos.* **Matos, Guillermo.** N°2-2006, Cuba : s.n., Información tecnológica, Vol. 17. 0718-0764.

25. **Liebowitz, J.** *Knowledge Management Handbook.* s.l. : CRC Press, 1999.

26. **Edsger, Dijkstra.** *The Structure of the THE Multiprogramming system.* 1983. págs. 49-52. 26(1).

27. **Brooks, Frederick Jr.** *The mythical man-month. Reading, Addison-Wesley.* 1975.

28. **Perry, Dewayne y Wolf , Alexander.** *Foundations for the study of software architecture.* s.l. : SOFTWARE ENGINEERING NOTES, 1992. Vol. 17.

29. *An introduction to software architecture.* **David Garlan y Mary Shaw.** s.l. : CMU Software Engineering Institute, 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

30. **Bass, Len, Clements, Paul y Kazman, Rick.** *Software Architecture in Practice.* s.l. : Addison-Wesley Professiona, 1998.

31. **IEEE.** *IEEE Recommended Practice for Architectural Description of Software Intensive Systems.* s.l. : IEEE, 2000. 1471-2000.

32. *Introducción a la Arquitectura de Software.* **Reynoso, Carlos Billy.** 1.0 , s.l. : UNIVERSIDAD DE BUENOS AIRES, Marzo, 2004.

33. **Santiago, Juan Carlos de la Cruz.** ORQUESTADOR DE SERVICIOS WEB PARA EL DESARROLLO DE APLICACIONES DISTRIBUIDAS. Mexico : Centro de Investigación de la Computación, 2006.

34. **Alexander, Christopher.** *A Pattern Language: Towns, Buildings, Construction (APL).* s.l. : Oxford University Press, 1977.

35. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.** Arquitecturas de Software. 2004.

36. *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* **Gomez, Omar.**

37. **P. Clement.** "Sei (software engeniering institute).".

38. *Evaluando Arquitecturas de Software. Parte 2. Métodos de Evaluación.* **Gomez, Omar.**

39. **Rofman, Paul.** *Metodologías de Desarrollo "Proceso Unificado de Desarrollo"'.* 2004.

40. Apache Ant™. *The Apache Ant Project.* [En línea] mayo de 2012. http://ant.apache.org/index.html.

41. **Hufman, Steve.** *Entornos de desarrollo comprendidos en JAVA .* Madrid: s.n., 2008.

42. *The Source for Java Technology Collaboration.* [En línea] 2012. http://home.java.net.

43. SPARQL CURRENT STATUS. *World Wide Web Consortium W3C.* [En línea] mayo de 2012. http://www.w3.org/standards/techs/sparql#w3c_all.

44. RDF CURRENT STATUS. *World Wibe Web Consortium W3C.* [En línea] abril de 2012. http://www.w3.org/standards/techs/rdf#w3c_all.

45. OWL WEB ONTOLOGY LANGUAGE CURRENT STATUS. *World Wide Web Consortium W3C.* [En línea] mayo de 2012. http://www.w3.org/standards/techs/owl#w3c_all.

46. Terse RDF Triple Language. *World Wide Web Consortium W3C.* [En línea] mayo de 2012. http://www.w3.org/TR/turtle/.

47. Visual Paradigm for UML. [En línea] mayo de 2012. http://www.visual-paradigm.com/.

48. Apache Tomcat. *The Apache software fundation.* [En línea] mayo de 2012. http://tomcat.apache.org/.

49. *Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC).* **Aleksander González, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez. Colimia, Mexico.** 2005, Vol. Vol1.

50. **Batini C, Scannapieco M .** Data Quality: Concepts, Methodologies and. Berlin, Springer-Verlag Berlin Heidelberg.: s.n., 2006.

51. **Freeman, R. E.** *Strategic Management: A Stakeholder Approach.* 1984.