Universidad de las Ciencias Informáticas

Facultad # 5

"Modelo para el razonamiento sobre corrientes de datos"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

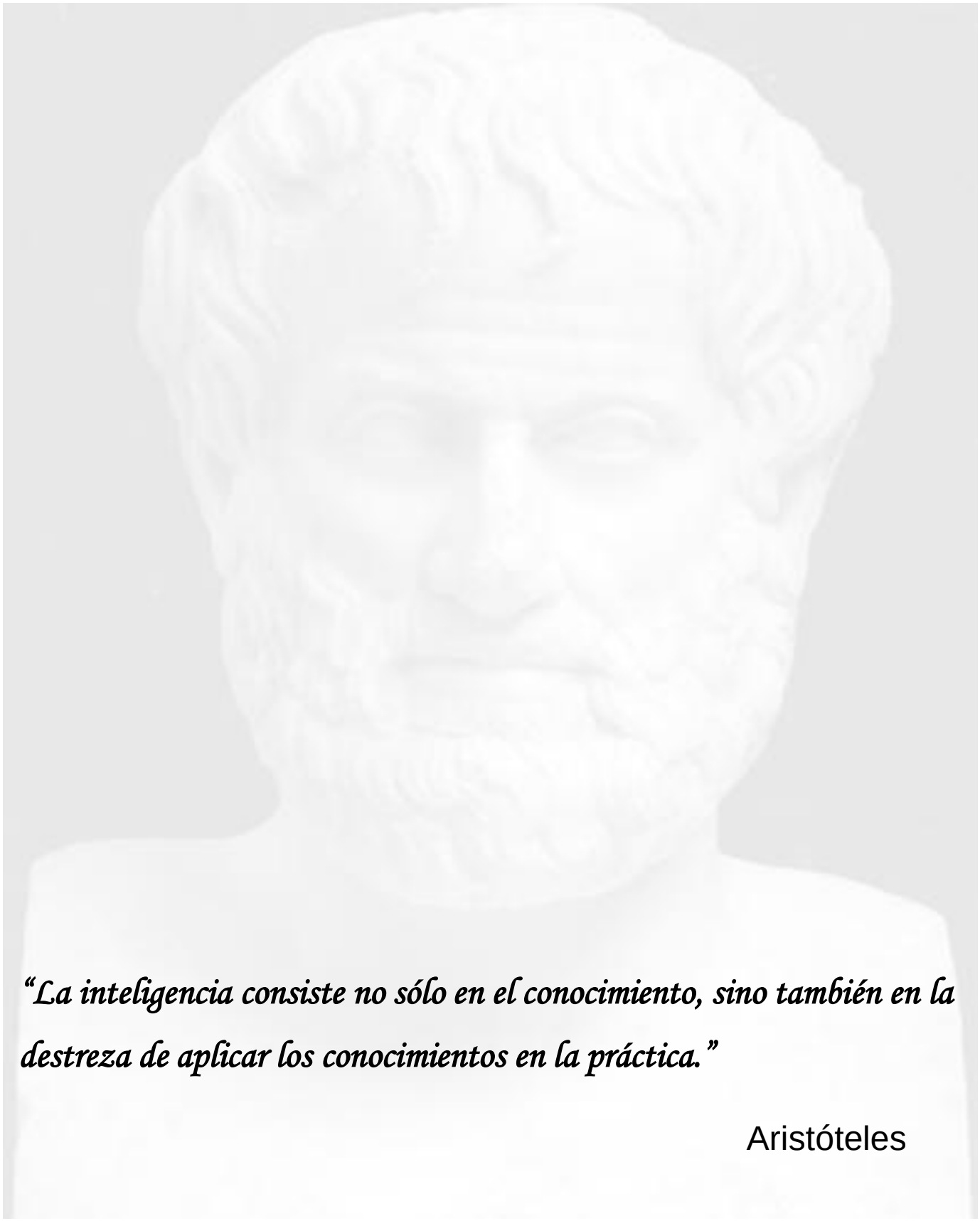
Autora: Aylín de la Concepción Caballero Weng

Tutores: MSc. Liudmila Reyes Álvarez

MSc. José Manuel Fernández Hechavarría

La Habana

Junio de 2012



"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica."

Aristóteles

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y autorizo a la Universidad de las Ciencias Informáticas, a hacer uso de la misma en su beneficio.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____.

_____ _____

Aylín de la Concepción Caballero Weng. MSc. Liudmila Reyes Álvarez.

_____

MSc. José Manuel Fernández Hechavarría.

DATOS DE CONTACTO

Tutores:

Nombre y Apellidos: M Sc. Liudmila Reyes Álvarez.
Institución: Universidad de las Ciencias Informáticas (UCI).
E-mail: lreyes@uci.cu
Graduada de la UCI en el año 2007. Profesor asistente con 5 años de experiencia docente y 8 años de experiencia en la producción. Pertenece al Departamento Integración y Despliegue del Centro de Desarrollo de Informática Industrial. Además es Máster en Ingeniería de Software e Inteligencia Artificial de la Universidad de Málaga, España.

Nombre y Apellidos: José Manuel Fernández Hechavarría.
Institución: Universidad de las Ciencias Informáticas (UCI).
E-mail: argos@uci.cu
Profesor instructor con 10 años de experiencia docente. Pertenece al Departamento de Ciencias Básicas de la Facultad 5. Además es Máster en Bioinformática.

AGRADECIMIENTOS

A mis padres y hermana por ser mi guía, siempre estar a mi lado ayudándome a ser mejor cada día y apoyarme en todo lo que me he propuesto.

Al resto de mi familia, tíos, tías, primos, abuelos por también brindarme su apoyo.

A mis viejas amigas que nunca se olvidan de mí y me apoyan en mis cosas como Leydi y Anais.

A Robert por apoyarme cuando más lo necesité y por siempre estar pendiente de mí y ayudarme cuando me hace falta.

A los nuevos amigos que conocí en esta universidad que vienen conmigo desde 1er año y me han brindado su amistad y apoyo en toda la carrera, en especial Rosy, Iselita, Sol, Liz, Jesusito, Alex, Ivan, Miguel, Adrian, Carlos.

A los que he conocido durante la carrera que también me han apoyado y me han brindado su amistad como Batista, Rosa María, Adonis, Arielito, Yaciel, Patry, Lourdes, Yissel, Silvia, Jandy, Alejandro Merzeaud, Diana, Reinier.

A mis profesores de toda la carrera por siempre tratar de que seamos mejores cada día.

A mis tutores por apoyarme en la realización de este trabajo, sin ellos no hubiese sido posible su terminación.

En fin a todos que de una forma u otra me apoyaron y ayudaron para que hoy sea INGENIERA con todas las letras, a ellos mil gracias.

DEDICATORIA

A mis padres y hermana por siempre estar a mi lado en todo momento y apoyarme en todo lo que hago y hacerme ver que todo en la vida se puede lograr.

Al resto de mi familia tías, tíos, primos y abuelos por también brindarme su apoyo.

A todas mis amistades, a los fieles que vienen conmigo desde 1er año, a los que he conocido, a los que se han ido y a los que continúan.

A todas aquellas personas que me brindaron su ayuda y apoyo en la realización de mi tesis.

A todos ustedes va dedicado este trabajo como resultado de mi esfuerzo y dedicación.

RESUMEN

Una de las principales prioridades de la inteligencia artificial es la representación del conocimiento. Para ello utiliza técnicas y lenguajes que de una forma u otra representan el mismo en información de forma estática o dinámica. Dentro de esta última se encuentran las corrientes de datos, que no son más que datos fluyentes a través del ordenador desde una entrada hacia una salida. Para consultar el conocimiento representado en ellas, se emplean diferentes lenguajes, donde se encuentra el *C-Sparql*, lenguaje utilizado para gestionar corrientes de datos en forma de marco de descripción de recursos (*RDF)*. Su característica distintiva es el soporte de las consultas continuas, es decir que las consultas son registradas y luego se ejecutan continuamente a través de ventanas sobre las corrientes de datos *RDF*. Este lenguaje presenta limitantes en el razonamiento sobre corrientes de datos en cuanto a escalabilidad y eficiencia, pero a pesar de eso, constituye la base para un futuro modelo o infraestructura de razonamiento sobre corrientes, a través de las herramientas como los razonadores que hacen frente a la evolución del conocimiento en tiempo real.

En este trabajo se propone un modelo de razonamiento sobre corrientes de datos basado en este lenguaje que mejora a su vez este razonamiento haciéndolo eficiente y escalable.

Palabras clave: corrientes de datos, *C-Sparql,* eficiencia, escalabilidad, inteligencia artificial, lenguajes, modelo, razonamiento, representación del conocimiento.

ABSTRACT

One of the main priorities of artificial intelligence is the knowledge representation. It uses techniques and languages that in one way or another represent the same information in a static or dynamic. Among the latter are flows of data, which data are only flowing through the computer from an inlet toward an exit. To consult the knowledge represented in them, different languages are used, where the C-SPARQL, language used to manage data streams in Resource Description Framework (RDF) format. Its distinctive feature is support for continuous queries, i.e. queries are recorded and then run continuously through windows on the RDF data streams. This language has limitations in reasoning about data streams for scalability and efficiency, but even so, is the basis for a future model or infrastructure of reasoning about streams, through tools like reasoners facing the evolution of knowledge in real time.

This paper proposes a model of reasoning about data streams based on this language which in turn improves reasoning and efficiency and scalability.

***Keywords**: artificial intelligence, C-SPARQL, data streams, efficiency, knowledge representation, language, model, reasoning, scalability.*

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El conocimiento puede ser definido como el conjunto de hechos y principios acumulados por la humanidad, o el acto, hecho o estado de conocer. Es la familiaridad con el lenguaje, conceptos, procedimientos, reglas, ideas, abstracciones, lugares, costumbres y asociaciones, unida a la habilidad de utilizar estas nociones en forma efectiva para modelar diferentes aspectos del universo que nos rodea. [1] Puede ser representado como imágenes mentales en nuestros pensamientos, como palabras habladas o escritas en algún lenguaje, en forma gráfica o en imágenes, como cadenas de caracteres o colecciones de señales eléctricas o magnéticas dentro de un computador. Constituye un punto importante y primordial para el comportamiento inteligente, su representación es una de las máximas prioridades de la investigación en la Inteligencia Artificial.

La investigación en el campo de la representación del conocimiento se enfoca normalmente hacia métodos que proporcionen descripciones del mundo de alto nivel con el fin de usarlas para construir aplicaciones inteligentes, o sea, sistemas capaces de encontrar conocimiento implícito a partir de conocimiento explícito. De ahí surge la necesidad de crear un lenguaje que permita analizar y resolver el problema del razonamiento sobre grandes corrientes de datos, a raíz de ello surge *C-Sparql*, lenguaje que cuenta con una infraestructura que tiene varias limitaciones al enfrentarse a encontrar conocimiento implícito a partir de conocimiento explícito en grandes corrientes de datos, estas limitaciones están dadas por:

- Pequeña escalabilidad a la hora de dar respuesta a las disímiles consultas realizadas por los usuarios sobre la gran cantidad de corrientes de datos que aparecen en la web actual.
- Las respuestas a las consultas realizadas por los usuarios a las corrientes de datos no satisfacen sus necesidades.
- El conocimiento implícito resultado del razonamiento sobre el conocimiento explícito en formato RDF presenta diversas incongruencias, por lo que es ineficiente.

Teniendo en cuenta lo anterior se hace necesario conocer el significado de escalabilidad y eficiencia. La primera de estas es la habilidad que presenta el lenguaje para terminar un número dado de operaciones

en un período específico de tiempo. Mientras que la eficiencia es la capacidad de alcanzar los objetivos y metas programadas con el mínimo de recursos disponibles y tiempo, logrando de esta forma su optimización.

Es por ello que el **problema científico** se enfoca en:

¿Cómo contribuir al desarrollo del proceso de razonamiento sobre grandes corrientes de datos?

El **objeto de estudio** son los procesos de razonamiento sobre datos dinámicos (S*tream Reasoning*). Mientras que el **campo de acción** se enmarca en el *Stream Reasoning*.

Con la finalidad de resolver el problema científico se trazó como **objetivo**:

Desarrollar un modelo que permita razonar escalable y eficientemente sobre grandes corrientes de datos (*Stream Reasoning*) basado en la infraestructura del lenguaje de consultas sobre corrientes *RDF* (*Resource Description Framework*): *C-Sparql*.

Para dar cumplimiento al objetivo propuesto se plantean las siguientes tareas investigativas:

- Elaboración del marco teórico para fundamentar el estudio del estado del arte de la investigación, asociado a los temas de proceso de razonamiento sobre datos dinámicos y razonamiento sobre corrientes de datos (*Stream Reasoning*).
- Diseño de los casos de prueba a realizar al lenguaje *C-Sparql*.
- Elaboración de pruebas funcionales al lenguaje *C–Sparql* para identificar deficiencias.
- Identificación de los mecanismos del lenguaje que permiten aumentar la escalabilidad del razonamiento sobre corrientes de datos (*Stream Reasoning*).
- Identificación de los mecanismos del lenguaje que permiten aumentar la eficiencia del razonamiento sobre corrientes de datos (*Stream Reasoning*).
- Confección del modelo que permita razonar escalable y eficiencientemente durante el razonamiento sobre corrientes de datos.

Este trabajo está dirigido **a defender la siguiente idea**: con la creación de un modelo se sentarán las bases de una teoría formal para la implementación de una arquitectura *Stream Reasoning.*

Como resultado de la investigación, se quiere lograr la definición y aplicación de estos mecanismos para el procesamiento automatizado en tiempo real de los datos provenientes de los sensores, lo cual responde a uno de los objetivos de la tesis doctoral de la tutora de este trabajo de diploma.

Para dar cumplimiento a las tareas, la investigación se basará en los marcos de los métodos científicos de investigación **Teóricos** y **Empíricos**.

Dentro de los métodos teóricos:

- *Análisis y síntesis:* Para analizar y extraer los elementos más importantes que se relacionan con el objeto de estudio.
- *Análisis histórico-lógico*: Para analizar toda la evolución del problema que se estará estudiando.
- *Modelación*: Para la creación de modelos, propuestas y alternativas, donde serán representadas las características y relaciones fundamentales del problema que se estará estudiando.

Métodos empíricos que serán referenciados:

- *Observación:* Permite adquirir información necesaria y puede utilizarse en cualquier fase de la investigación, además de que permite ver la posible solución del problema desde diferentes puntos de vista.
- *Revisión bibliográfica:* Para fundamentar el estudio del estado del arte mediante consultas a un conjunto de fuentes de información referidas a los temas de proceso de razonamiento sobre datos dinámicos y razonamiento sobre corrientes de datos, como libros, artículos, revistas, publicaciones, boletines y toda una variedad de materiales escritos y digitales.
- *Experimentación:* Para evaluar y mejorar el lenguaje una vez aplicadas las pruebas y detectadas las deficiencias.

El presente documento está estructurado de la siguiente manera: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, glosario de términos y anexos.

En el **Capítulo1**: *FUNDAMENTACIÓN TEÓRICA*, se realiza un estudio del estado del arte donde se investiga todo lo relacionado con la representación del conocimiento y razonamiento sobre datos dinámicos, enfatizando sobre el razonamiento sobre corrientes de datos, viendo conceptos importantes, definiciones y lenguajes utilizados.

En el **Capítulo 2**: *PROCESO DE RAZONAMIENTO SOBRE CORRIENTES DE DATOS,* se realizan pruebas al lenguaje utilizado para el razonamiento y se describen las mismas.

En el **Capítulo 3**: *PROPUESTA DEL MODELO*, se propone un modelo para el razonamiento sobre corrientes de datos basado en el lenguaje.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se muestra un estudio de los conceptos relacionados con representación del conocimiento y razonamiento sobre datos dinámicos dados por diferentes autores. Se parte de la web semántica, la cual cuenta con diferentes componentes, capas, lenguajes de consulta y motores de razonamiento, que le favorecen en su funcionamiento. Se exponen conceptos sobre representación del conocimiento, donde se explican los lenguajes que se utilizan para representar el mismo, como también los utilizados en una corriente de datos. Por último se puntualizan conceptos relacionados con proceso de razonamiento, donde se realiza una descripción del mismo, se explican los tipos de razonamientos que existen, así como sus aplicaciones. Sirviendo como base todo lo anterior para la realización del trabajo.

1.2 Web Semántica

La web semántica es una extensión de la *world wide web (www)*, creada por Tim Berners-Lee con el objetivo de obtener un medio universal que permita el intercambio de datos y brindar un mayor significado a los mismos para que puedan ser interpretados por las máquinas u ordenadores.

Es una web que contiene mucha información donde podemos relacionar diferentes recursos. Con la misma podemos solucionar problemas habituales como son los de realizar búsquedas en Internet. [2]

Sus contenidos pueden ser expresados mucho más que en un lenguaje natural, y también en un formato que pueda ser entendido, interpretado y usado por diferentes software, permitiéndoles buscar, compartir e integrar información más fácil.

La web semántica está dotada de un mayor significado, se desarrolla con lenguajes universales que permiten a los usuarios encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a la mejor estructuración de la información. Como también delegar tareas a las herramientas que utiliza, las cuales podrán ser capaces de procesar la información. Para poder obtener información en la web semántica, se utiliza el *framework* para metadatos[1] *RDF*, el cual constituye un marco de descripción de

[1] Datos sobre datos que permiten describir, identificar y localizar contenidos en documentos de la web.

recursos creado por W3C[2] "*World Wide Web Consortium*". 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). Mediante el mismo podemos extraer la información de la base de datos para crear un formato más comprensible para las máquinas. Con esta información podemos realizar deducciones lógicas, combinar información, generar información nueva a partir de una ya existente y realizar consultas complejas en buscadores.

Entre los principales componentes de la web semántica podemos encontrar: [2]

- **XML (*Extensible Markup Language*):** Provee una sintaxis elemental para las estructuras de contenidos dentro de documentos.
- **XML Schema:** Es un lenguaje para proporcionar y restringir la estructura y el contenido de los elementos contenidos dentro de documentos *XML*.
- **RDF (*Resource Description Framework*):** Es un lenguaje simple para expresar modelos de los datos, que refieren a los objetos "recursos" y a sus relaciones. Un modelo de *RDF-based* se puede representar en sintaxis de *XML*.
- **RDF Schema:** Es un vocabulario para describir propiedades y clases de recursos *RDF-based*, con semántica para generalizar-jerarquías de las propiedades y clases.
- **OWL *(Web Ontology Language)*:** Es un mecanismo para desarrollar temas o vocabularios específicos en los que podamos asociar esos recursos.

La web semántica está estructurada por diferentes capas (Figura 1) que cumplen una función muy importante para el buen funcionamiento de la misma. Entre ellas se encuentran: [2]

- **Unicode:** Es un estándar cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático.

[2] Consorcio internacional que produce recomendaciones para la *world wide web*, dirigida por Tim Berners-Lee.

- **URI *(Uniform Resource Identifier)*:** Son cadenas que permiten acceder a cualquier recurso de la web. Son las encargadas de identificar objetos.
- **XML+NS+xmlschema:** En ella se encuentran agrupadas las diferentes tecnologías que posibilitan la comunicación entre agentes.
- **RDF+rdfschema:** Está basada en la capa anterior, define el lenguaje universal con el que podemos expresar diferentes ideas en la web semántica.
- **Ontology (Ontologías):** Nos permite clasificar la información, así como extender la funcionalidad de la web semántica agregando nuevas clases y propiedades para describir los recursos.
- **Logic (Lógica):** Además de ontologías se precisan reglas de inferencia.
- **Proof (Pruebas):** Se intercambiarán "pruebas" escritas en el lenguaje unificador de la web semántica. Este lenguaje posibilita las inferencias lógicas realizadas a través del uso de reglas de inferencia.
- **Trust (Confianza):** Hasta que no se haya comprobado de forma exhaustiva las fuentes de información, los agentes deberían ser muy escépticos acerca de lo que leen en la web semántica.
- **Digital Signature (Firma digital):** Utilizada por los ordenadores y agentes para verificar que la información ha sido ofrecida por una fuente de confianza**.**

1.2.1 Lenguajes de consulta y motores de razonamiento utilizados en la web semántica

Para que la web semántica sea una realidad, precisa tanto de un lenguaje de consulta estándar y de un protocolo de recuperación. [3] Para esto se han desarrollado diferentes lenguajes. A continuación se muestran algunos de ellos:

- ***Sparql*:** Consiste en tres especificaciones separadas, que contienen diferentes partes de su funcionalidad: un lenguaje de consulta, un formato para las respuestas, y un medio para el transporte de consultas y respuestas. Estos son:[4]
  - ✓ *Sparql Query Language*: Núcleo de *Sparql*. Explica la sintaxis para la composición de sentencias y su concordancia.

- ✓ *Sparql Protocol*: Formato utilizado para la devolución de los resultados de las búsquedas (sentencias *SELECT* o *ASK*), a partir de un esquema de *XML*.
- ✓ *Sparql Query XML Results Format:* Describe el acceso remoto de datos y la transmisión de consultas de los clientes a los procesadores. Utiliza *WSDL* [5] para definir protocolos remotos para la consulta de bases de datos basadas en *RDF*.

- ❖ ***Triple*:** Lenguaje *RDF*[3] para la transformación hacia la web semántica. [3] Se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto. El elemento de construcción básica en *RDF* es la "tripleta" o sentencia, que consiste en dos nodos (sujeto y objeto) unidos por un arco (predicado), donde los nodos representan recursos, los arcos propiedades y el objeto es el valor de la propiedad o el otro recurso con el que se establece la relación.[4]

- ❖ ***RQL*** (*RDF Query Language*): Lenguaje de consulta a nivel semántico. Debe ser sensible a la semántica de las primitivas *RDF*. Es un lenguaje de consulta declarativo para *RDF* que explícitamente captura esta semántica en su diseño.

  Fue desarrollado en el instituto ICS-FORTH, y su potencia semántica está basada en la evaluación de caminos de expresiones sobre grafos *RDF*. Permite el uso de variables tanto para denotar clases y propiedades. Permite consultar esquemas *RDF*, *RDF-S*[4], y descripciones *RDF* en una misma consulta. *RQL* está definido por medio de un conjunto de consultas básicas, e iteradores que se permiten construir otras consultas a través de una composición funcional con base en la teoría planteada por *OQL*.[5]

  Sin embargo la semántica de *RQL* no es completamente compatible con la semántica de *RDF*, existen un número de restricciones adicionales que deben ser puestas sobre los modelos *RDF* para poder llevar a cabo las consultas sobre los documentos con entera confiabilidad; un ejemplo de tales restricciones es que cada propiedad debe tener exactamente un dominio y un rango específico. [4]

[3] *Resource Description Framework*. Es una base para procesar metadatos y para la implementación de la web semántica.

[4] Conjunto de clases con ciertas propiedades utilizando el *RDF* extensible de representación del conocimiento del lenguaje.

[5] Lenguaje de consulta estándar para bases de datos orientadas a objetos.

- ***eRQL*** (*easy RDF Query Language*): Constituye una manera fácil e intuitiva de utilizar el lenguaje de consulta *RQL*. Su implementación se basa en *RQL* y cumple una serie de características como:

  - ✓ Simplicidad: El usuario debe tener la posibilidad de hacer consultas simples, incluso sin ningún conocimiento previo.
  - ✓ Independencia del esquema: Las consultas deben poder realizarse sin conocer la base del esquema *RDF* de los datos correspondientes.
  - ✓ Poder: Debe permitir la realización de consultas de gran alcance, las cuales deberán ser generadas mediante la concatenación de lo más simple.
  - ✓ Independencia de dominio: El lenguaje de consulta debe ser independiente de un dominio específico.

- ***SeRQL*** (*Sesame RDF Query Language*): Lenguaje de recuperación para *RDF/RDF-S* desarrollado por Aduna como parte del software *Sesame*. Combina características de otros lenguajes (principalmente *RQL, RDQL, N-Triples* y *N3*) y añade otras propias. Algunas de sus características más importantes para la recuperación y organización de la información son:
  - ✓ Tranformación de grafos.
  - ✓ Soporte de *RDF Schema*.
  - ✓ Soporte de los tipos de datos de *XML Schema*.
  - ✓ Emparejado de caminos opcionales.

La sintaxis de *SeRQL* es similar a la de *RQL*, añadiendo algunas modificaciones. Al igual que *RQL* se basa en una interpretación de los grafos modelados con *RDF*.

1.3 Representación del conocimiento en una corriente de datos

1.3.1 ¿Qué es representar el conocimiento?

El conocimiento es adquirido por el hombre por muchas vías. Existen varios tipos de conocimiento como son:

- **Declarativo** (o no procedimental): Lógica. Donde la lógica puede representar el mismo tipo de procedimientos de un lenguaje de programación. Requiere relaciones o predicados explícitos para expresar la secuencia, mientras que los lenguajes procedimentales dependen de la secuencia implícita en la estructura del programa. Dentro de este tipo de conocimiento se encuentran:
    - ✓ Conocimiento relacional simple: Conjunto de relaciones del mismo tipo que las de una base de datos.
    - ✓ Conocimiento heredable: Se puede observar en una estructuración jerárquica, en las relaciones de programación es-un (clase-clase), instancia-de (clase-instancia), en la herencia de propiedades y valores (Herencia simple y múltiple, y en Valores por defecto).
    - ✓ Conocimiento inferible: Describe la vía lógica tradicional y un ejemplo de su utilización es la resolución.
- **Procedimental**: Representado por funciones, reglas de producción, lenguajes de programación convencionales, etc.

Una vez que se adquiere el conocimiento, es necesario encontrar una representación simbólica, clara, precisa y completa del mismo. Una representación del conocimiento (RC) o un esquema de representación se define como: [6]

1. Un **substituto** de lo que existe en el mundo real o imaginario.

2. Un conjunto de **cometidos ontológicos**:

- ¿En qué términos hay que pensar acerca del mundo?

3. Una **teoría fragmentaria del razonamiento inteligente**:

- ¿Qué es la inteligencia?
- ¿Qué se puede inferir de lo que se conoce?
- ¿Qué se debería inferir de lo que se conoce? (¿Qué inferencias son recomendadas?)

4. Un medio para la **computación eficiente**:

- ¿Cómo se debería organizar la información para facilitar la manera de pensar y razonar?

5. Un medio de **expresión humana**: Un lenguaje que las personas usan para hablar entre ellas y con las máquinas.

(1). Es un substituto. [7]

- Es un substituto de los objetos del mundo real o imaginario.
- Las operaciones sobre la representación del conocimiento substituyen acciones en el mundo.
- El mismo razonamiento es un substituto de la acción. Inversamente, las acciones pueden sustituir el razonamiento.

(2). Conjunto de cometidos ontológicos (COs).

- Los substitutos son inevitablemente imperfectos. La selección de una RC inevitablemente hace un CO. Determina lo que se puede conocer, enfocando una parte del mundo y desenfocando las otras.
- El cometido ocurre incluso al nivel de las técnicas de representación del conocimiento.

(3). Fragmento de una teoría de razonamiento inteligente.

- ¿Cuáles son todas las inferencias que está permitido hacer?

  Ejemplos:

  - ✓ Lógica formal clásica: Inferencias bien fundamentadas.
  - ✓ Reglas: Inferencias plausibles.
  - ✓ Ontologías: Expectaciones, valores por defecto.

- Estilos de respuestas.

- ✓ **Precisas**, formuladas en términos de un lenguaje formal.
- ✓ **Imprecisas**, no basadas en un lenguaje formal.

- ¿Cuáles inferencias están especialmente impulsadas?

  Ejemplos:

  - ✓ Ontologías: propagación de valores, enlaces.
  - ✓ Reglas: encadenamiento, asociaciones.
  - ✓ Lógica: proposiciones subordinadas (lemas), grafos de conexión.

- Explosión combinatoria.
  - ✓ Necesidad de guía sobre lo que se debería hacer, no sólo lo que se puede hacer.

(4) Medio para la computación pragmáticamente eficiente.

- Razonar con una RC significa hacer algún tipo de computación.
- ¿Cómo se puede organizar la información para facilitar el razonamiento?
- Ejemplos:
  - ✓ Marcos, ontologías: *triggers*, jerarquías taxonómicas.
  - ✓ Lógica: *theorem provers* de grafos de conexión.

(5) Medio de expresión y comunicación.

- ¿Cómo se expresan las personas acerca del mundo?
- ¿Cómo se comunican las personas entre ellas y con el sistema de razonamiento?
- RC como medio de expresión:
  - ✓ ¿Cómo es de general, preciso? ¿Proporciona una expresividad adecuada?

- RC como medio de comunicación:
  - ✓ ¿Cómo es de transparente? ¿Los humanos pueden entender lo que se está diciendo?
  - ✓ ¿Se pueden generar las expresiones que interesan?

Desde el punto de vista informático el conocimiento puede ser representado a través de estructuras de datos, las cuales representan el dominio y el problema de manera estática; y procedimientos, los cuales manipulan las estructuras de manera dinámica con la utilización de operaciones (procedimientos para crear, modificar o destruir las representaciones o sus elementos) y predicados (procedimientos para acceder a campos concretos de información).

1.3.2 Lenguajes para representar el conocimiento

A partir de las técnicas de representación del conocimiento, se han diseñado a lo largo de los años diversos formalismos y lenguajes que permiten modelar de un modo formal el conocimiento representado mediante ellas. Entre los lenguajes más destacados que se han utilizado y que se siguen utilizando hoy en día en diferentes ámbitos se encuentran: [8]

- ❖ **Lenguajes basados en lógica formal:** Muchos de los sistemas de representación del conocimiento de la actualidad se basan en algún tipo de lógica formal. Esta lógica aporta un buen número de ventajas para la representación del conocimiento y su manejo, partiendo de una sintaxis y semántica bien definidas que detallan perfectamente la forma de construir sentencias y razonamientos sobre ellas. Entre estos tipos de lenguajes se encuentran:
  - ✓ Lógica Proposicional: Una proposición es una sentencia que puede decirse que es falsa o verdadera. Se asigna símbolos a cada sentencia y se utilizan operadores lógicos como: *AND* (∧), *OR* (∨), *NOT* (¬), *IMPLIES* (→ o ⇒), y *EQUIVALENCE* (⇔). Partiendo de los símbolos y utilizando los diferentes operadores se construyen proposiciones complejas, las cuales es posible obtener si son ciertas o falsas operando a partir de los valores de verdad de cada uno de los símbolos iniciales, utilizando el cálculo proposicional.
  - ✓ Lógica de Primer Orden: Es una ampliación de la lógica proposicional a partir de dos operadores más, el cuantificador universal ∀ y el existencial ∃. Utiliza también símbolos para

representar conocimiento y operadores lógicos para construir sentencias más complejas, pero a diferencia de la lógica proposicional, los símbolos pueden representar constantes, variables, predicados y funciones.

- ✓ Lógicas Descriptivas: Están muy relacionadas con el desarrollo de las ontologías tal como se usan en la actualidad en la web semántica. Se basan en representar el conocimiento, utilizando por un lado una terminología o vocabulario del dominio *(TBOX)* y por otro un conjunto de afirmaciones *(ABOX).* El vocabulario consiste en conceptos y roles; donde los conceptos corresponden a conjuntos de elementos y los roles a relaciones binarias entre elementos.

- ❖ **Lenguajes basados en *Frames*[6] o Marcos:** Estos lenguajes son similares a los lenguajes de programación orientados a objetos, en el sentido de que modelan el conocimiento utilizando clases, atributos, objetos y relaciones, y utilizan relaciones de generalización y especialización para representar la organización jerárquica de los conceptos.

- ❖ **Lenguajes basados en Reglas:** Han sido durante mucho tiempo posiblemente los más usados, principalmente debido a su estrecha relación con los Sistemas Expertos utilizados en Inteligencia Artificial. Estos lenguajes son fáciles de entender debido a su sencillez conceptual y a su paralelismo con las estructuras de control más simples utilizadas en programación.

  Estos tipos de lenguajes han recibido también un fuerte impulso a partir de la aparición de la web semántica, ya que se piensa en ellos como herramientas para definir servicios web, y como herramienta base que permita definir la forma en la que pueden interactuar las aplicaciones de comercio electrónico.

- ❖ **Ontologías:** El concepto de Ontología en el ámbito de la informática se obtuvo del campo de la filosofía, con el objetivo de abarcar cualquier tipo de representación de conocimiento. Es por ello que es muy amplio y se ha venido usando prácticamente para cualquier tipo de modelado de dominio de conocimiento. Una de las definiciones más consensuadas es la proporcionada por Grubber, **"*An ontology is a formal explicit specification of a shared conceptualisation*"**. En esta definición, "*explicit*" se refiere a la necesidad de detallar los diferentes conceptos que forman

[6] Son redes semánticas estructuradas. Conforman una colección de atributos y la descripción de sus características.

la ontología, "*formal*" indica que el conocimiento que modelemos debe representarse según un lenguaje formalizado e interpretable fácilmente y "*shared*" indica que la ontología modela el conocimiento común sobre la materia, para las diferentes personas que deben usarla. Este último punto referido al conocimiento compartido es una de las principales diferencias sobre otros modelos de conocimiento, en los que muchas veces no se tenía en cuenta el concepto de conocimiento compartido. Aún así, esta definición es muy amplia, y engloba el modelado de conocimiento con muy diferentes técnicas, formalismos y lenguajes.

1.3.2.1 Lenguajes de consulta sobre el conocimiento representado en una corriente de datos

Un primer paso hacia el *Stream Reasoning* (en el español una posible traducción de este término es razonamiento sobre corrientes de datos, pero en la investigación se decide utilizar de esta forma) es, sin duda tratando de combinar el poder de los actuales sistemas de gestión de flujos de datos y la web semántica. La idea clave es mantener los datos fluyentes en formato relacional el mayor tiempo posible y llevarlos a nivel semántico como eventos agregados. Los modelos de datos existentes, protocolos de acceso, y lenguajes de consulta para los sistemas de gestión de flujos de datos y la web semántica no son suficientes para hacerlo, sino que deben ser combinados. Es por ello que se han creado diversos lenguajes entre los que se encuentran: *Sparql-ST, EP-Sparql, Streaming Sparql* y *Continuous Sparql (C-Sparql)*, que son propuestas de ampliación de *Sparql* para gestionar corrientes de datos. Los cuales introducen la noción de corrientes *RDF* como la extensión natural del modelo de datos *RDF* para este escenario, y luego extender *Sparql* para consulta de corrientes *RDF*.

- *Sparql-ST* es una extensión de *Sparql* para consultas espaciales complejas y temporales. El lenguaje y una correspondiente implementación tratan datos temporales (y posible razonamiento sobre estos datos). Esto se aplica para otros acercamientos de *Sparql* como *Temporal Sparql, stSparql* y *T-Sparql*. Los datos de la corriente son representados en un formato *RDF* con un propósito de explotación en aplicaciones de web semántica (semánticamente los datos son anotados y razonados con servicios). Para esto se propone un modelo *RDF Time-Annotated* y un *Time-Annotated Sparql*. Sin embargo las consultas continuas, como un requisito típico de razonamiento sobre corrientes de datos, no son consideradas en este lenguaje. [9]

- El Procesamiento de Eventos, se preocupa por el descubrimiento oportuno de eventos compuestos dentro de las corrientes de datos de eventos simples. Este procesamiento proporciona el análisis de corrientes de datos de eventos, pero no puede combinar las corrientes de datos con la base de conocimiento, y no puede realizar tareas de razonamiento. Para lograr esto se crea el lenguaje, *Event Processing Sparql (EP-Sparql),* que constituye un nuevo lenguaje para eventos complejos y razonamiento sobre corrientes de datos. Este presenta una sintaxis y una semántica formal, y un modelo efectivo de ejecución del formalismo propuesto. La ejecución de este modelo está inmerso en la programación lógica y funcionalidades efectivas del procesamiento de eventos y capacidades de inferencia sobre conocimiento estático y temporal.[9]

- *Streaming Sparql* es una extensión de *Sparql* introducida en el álgebra correlativa temporal y provee un algoritmo para transformar las consultas *Sparql* en esta álgebra. Este no detecta secuencias de tripletas *RDF* que ocurren en un orden específico y para ello propone un razonamiento sobre corrientes de datos basado en mantenimiento de materializaciones incremental. Las tripletas *RDF* de la corriente (cuando ocurren) realizan un procedimiento de inferencia que mantiene las materializaciones. Aun no está claro cómo este lenguaje trabaja para múltiples consultas con definiciones de ventanas de tiempo diferentes. [9]

Todas estas propuestas almacenan de una forma u otra la corriente de datos permanentemente y la procesan a través de consultas. Pero a diferencia *C-Sparql* da la noción de procesamiento continuo. Tiene como característica distintiva, el soporte de las consultas continuas, lo cual significa que las consultas se registran en los flujos de datos *RDF* y luego se ejecutan de forma continua. Estas consultas tienen en cuenta las ventanas de tiempo*,* es decir, se seleccionan las tripletas a utilizar a través de ellas, mientras que los datos observados están fluyendo continuamente. [10]

Las consultas se dividen en la parte estática y la parte dinámica. La parte estática se evalúa por medio de una tripleta *RDF* almacenada, mientras que la corriente procesa el artefacto que se evalúa en la parte dinámica de la consulta. En estas escenas, estas dos partes actúan como cajas negras y *C-Sparql*, la ventaja de tomar una consulta pre-procesada y/o una optimización sobre el espacio unificado de datos no es posible. [9] Lo cual representa una limitación de eficiencia para este lenguaje. También presenta dificultades en cuanto a distribución de los datos a la hora de dar respuesta a las consultas, lo cual

constituye una limitación de escalabilidad, porque para ser lograda la misma deben combinarse: distribución y paralelismo, y este último lo realiza en la ejecución de las consultas.

1.3.3 Importancia de razonar sobre una corriente de datos

Las corrientes de datos se producen ampliamente en diversas aplicaciones del mundo real. La investigación sobre el flujo de datos se centra principalmente en la gestión de datos, evaluación y optimización de consultas sobre estos datos. Sin embargo el trabajo sobre procedimientos de razonamiento para la transmisión de las bases de conocimiento, es muy limitada, ya que el razonamiento es normalmente aplicado a las bases de conocimiento estático y, a menudo ignorado por los datos que cambian rápidamente, tanto a nivel terminológico como afirmativo. Pero, existe una clara necesidad de diseño y aplicación de métodos de razonamiento para las bases de conocimiento dinámico motivado por su uso aumentado en los datos provenientes de los sensores, corrientes de datos provenientes de una web y datos multimedia, que se utilizan para cambiar rápidamente datos financieros y médicos en tiempo real.[11] Es por ello que el razonamiento sobre corrientes de datos se ha convertido en un tema popular en la investigación de las base de datos, afrontando cada día el desafío de responder preguntas más eficientes realizadas a datos continuos. Mientras que los datos continuos se han hecho más y más importantes como base para los procesos de nivel superior, los cuales precisan de un razonamiento más complejo y una base de conocimiento enriquecida.[12]

1.4 Proceso de razonamiento

1.4.1Razonamiento

Podría definirse como un conjunto de operaciones cognoscitivas que nos permiten como personas expresar alguna opinión, algún juicio, alguna conclusión. (Definición Empírica)

El proceso de razonamiento en un sistema basado en reglas es una progresión desde un conjunto inicial de afirmaciones y reglas hacia una solución, respuesta o conclusión. (Definición técnica)

El razonamiento es una actividad psíquica de orden cognitivo por la que se asocia un sujeto a un predicado, cuyo nexo no es del todo evidente. Un ejemplo sería: “Los sistemas operativos que no se bloquean son sistemas robustos, es así que Linux no se me ha bloqueado, por lo tanto Linux es un sistema operativo robusto”.

El vehículo normal del razonamiento es la lógica. Utilizando una lógica se elaboran razonamientos. Es importante tener en cuenta esto, ya que es lo que hace posible simular el razonamiento. Esta es la base teórica para que algunas aplicaciones de Inteligencia Artificial funcionen.

1.4.2 Proceso de razonamiento

El proceso de razonamiento en un sistema basado en reglas es una progresión desde un conjunto inicial de afirmaciones y reglas hacia una solución, respuesta o conclusión.

1.4.2.1 Como llegar a obtener el resultado

Existen dos formas de obtener el resultado en el proceso de razonamiento, estas son:

- Se puede partir considerando todos los datos conocidos y luego ir progresivamente avanzando hacia la solución. Este proceso se lo denomina guiado por los datos o de **encadenamiento progresivo** (*forward chainning***)**.

- Se puede seleccionar una posible solución y tratar de probar su validez buscando evidencia que la apoye. Este proceso se denomina guiado por el objetivo o de **encadenamiento regresivo** (*backward chainning*).

1.4.3 Tipos de Razonamiento

1.4.3.1 Razonamiento progresivo

En el caso del razonamiento progresivo, se empieza a partir de un conjunto de datos colectados a través de observación y se evoluciona hacia una conclusión. Se examina cada una de las reglas para ver si los datos observados satisfacen las premisas de alguna de las reglas. Si una regla es satisfecha, es ejecutada derivando nuevos hechos que pueden ser utilizados por otras reglas para derivar hechos adicionales. Este proceso de chequear reglas para ver si pueden ser satisfechas se denomina: interpretación de reglas. La cual es realizada por una máquina de inferencia en un sistema basado en conocimiento.

La interpretación de reglas, o inferencia, en el razonamiento progresivo involucra la repetición de los pasos que se indican a continuación (Figura 2).

1. **Unificación (*Matching*):** En este paso, en las reglas en la base de conocimientos se prueban los hechos conocidos al momento para ver cuáles son las que resulten satisfechas.
2. **Resolución de Conflictos:** La resolución de conflictos involucra la selección de la regla que tenga la más alta prioridad de entre el conjunto de reglas que han sido satisfechas.
3. **Ejecución:** La ejecución puede dar lugar a uno o dos resultados posibles: nuevo hecho (o hechos) pueden ser derivados y añadidos a la base de hechos, o una nueva regla (o reglas) pueden ser añadidas al conjunto de reglas (base de conocimiento) que el sistema considera para ejecución.

1.4.3.1.1 Aplicaciones

Un conjunto de aplicaciones adecuadas al razonamiento progresivo incluye supervisión y diagnóstico en sistemas de control de procesos en tiempo real, donde los datos están continuamente siendo adquiridos, modificados y actualizados. Estas aplicaciones tienen dos características importantes:

1. Necesidad de respuesta rápida a los cambios en los datos de entrada.
2. Existencia de pocas relaciones predeterminadas entre los datos de entrada y las conclusiones derivadas.

Otro conjunto de aplicaciones está formado por: diseño, planeamiento y calendarización, donde ocurre la síntesis de nuevos hechos basados en las conclusiones de las reglas. En estas aplicaciones hay potencialmente muchas soluciones que pueden ser derivadas de los datos de entrada.

1.4.3.2 Razonamiento regresivo

El mecanismo de inferencia, o intérprete de reglas para el razonamiento regresivo, difiere significativamente del mecanismo de razonamiento progresivo. Si bien es cierto, ambos procesos involucran el examen y aplicación de reglas, el razonamiento regresivo empieza con la conclusión deseada y decide si los hechos que existen pueden dar lugar a la obtención de un valor para esta conclusión.

El razonamiento regresivo sigue un proceso muy similar a la búsqueda primero en profundidad. El sistema empieza con un conjunto de hechos conocidos que típicamente está vacío. Se proporciona una lista

ordenada de objetivos (o conclusiones), para las cuales el sistema trata de derivar valores. El proceso de razonamiento regresivo utiliza esta lista de objetivos para coordinar su búsqueda a través de las reglas de la base de conocimientos. Esta búsqueda consiste de los siguientes pasos:

1. Conformar una pila inicialmente compuesta por todos los objetivos prioritarios definidos en el sistema.
2. Considerar el primer objetivo de la pila.
3. Para cada una de estas reglas examinar en turno sus antecedendentes:
    a. Si todos los antecedentes de la regla son satisfechos (esto es, cada parámetro de la premisa tiene su valor especificado dentro de la base de datos), entonces ejecutar esta regla para derivar sus conclusiones.
    b. Si alguna premisa de la regla no puede ser satisfecha, buscar reglas que permitan derivar el valor especificado para el parámetro utilizado en esta premisa.
    c. Si en el paso (b) no se puede encontrar una regla para derivar el valor especificado para el parámetro actual, entonces preguntar al usuario por dicho valor y añadirlo a la base de datos.
4. Si todas las reglas que pueden satisfacer el objetivo actual se han probado y todas no han podido derivar un valor, entonces este objetivo quedará indeterminado. Removerlo de la pila y retornar al paso (2). Si la pila está vacía parar y anunciar que se ha terminado el proceso.

1.4.3.2.1 Aplicaciones

El razonamiento regresivo es mucho más adecuado para aplicaciones que tienen mucho mayor número de entradas, que de soluciones posibles. Una excelente aplicación es el diagnóstico, donde el usuario dialoga directamente con el sistema basado en conocimiento y proporciona los datos a través del teclado.

1.5 Conclusiones

Luego de haber realizado el estudio del estado del arte relacionado con la web semántica, los diferentes lenguajes de consulta y motores de razonamiento que en ella se utilizan, la representación del

conocimiento, los lenguajes utilizados para representar el conocimiento en una corriente de datos, el proceso de razonamiento, así como los tipos de razonamiento que existen, se llega a las siguientes conclusiones:

- Las corrientes de datos surgen con la web semántica y el proceso de razonamiento sobre ellas presenta problemas en cuanto a escalabilidad y eficiencia por lo que deben ser mejoradas.
- De los lenguajes para consultar el conocimiento representado en una corriente de datos, se selecciona *C-Sparql* por sus características y las facilidades que brinda en el razonamiento sobre corrientes de datos.
- De los tipos de razonamiento estudiados, se escoge el razonamiento progresivo, por las aplicaciones que ofrece para datos en tiempo real, y es aplicable a corrientes de datos.

CAPÍTULO 2: PROCESO DE RAZONAMIENTO SOBRE CORRIENTES DE DATOS

2.1 Introducción

En este capítulo se realiza una descripción del proceso de razonamiento sobre corrientes de datos *(Stream Reasoning)*, abordando conceptos, definiciones y cosas fundamentales del lenguaje *C-Sparql*, escogido como base fundamental para la confección del modelo. Se realiza una descripción de las pruebas realizadas a este lenguaje y se analizan los resultados de las mismas. Contribuyendo todo lo anterior a la confección del modelo.

2.2 Stream Reasoning

El uso de la Internet como una fuente mayor de información ha creado nuevos desafíos para la informática y ha permitido una innovación significativa en áreas como las bases de datos, recuperación de información y tecnologías semánticas.

Actualmente, estamos enfrentando un cambio en la manera de que proviene la información. Tradicionalmente la información era principalmente estática, siendo en algunos casos la excepción de la regla. Hoy en día, la información es más dinámica, usada para ser escondida dentro de sistemas especializados, poniéndose solo disponible a fabricantes. [12]

El proceso continuo de corrientes de datos ha sido principalmente investigado en las comunidades de bases de datos. Este proceso junto con una base de conocimiento enriquecida, requiere de razonadores especializados, pero trabaja en las tecnologías semánticas, que todavía están enfocadas en los datos estáticos. Existen trabajos en cuanto a creencias cambiantes en base a las nuevas observaciones, pero las soluciones propuestas en esta área son demasiadas complejas para ser aplicadas a grandes corrientes de datos. Por lo que es de gran necesidad de cerrar el hueco entre las soluciones existentes para la actualización de la creencia, las necesidades reales de proceso de decisión de apoyo basada en las corrientes de datos y el enriquecido conocimiento de fondo. [12]

La combinación de las técnicas de razonamiento con las corrientes de datos da lugar al ***Stream Reasoning***, un inexplorado tema con alto impacto en esta área de investigación.

Este se puede caracterizar por tres aspectos importantes:

Definición1. *Stream Reasoning*: *Logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.* [12]
Una posible traducción de este concepto al español para un mejor entendimiento es: razonamiento lógico en tiempo real en grandes e inevitables corrientes de datos para soportar el proceso de decisión de números sumamente grandes de usuarios coexistentes.

Se puede entender el impacto potencial del *Stream Reasoning*, considerando emblemático el caso de la informática urbana[7]. La naturaleza de la misma puede explicarse por medio de las *data streams* (en el español una posible traducción de este termino es corrientes de datos y en la investigación se decide utilizar así), representando objetos reales que se supervisan dada una determinada situación como: los automóviles, los trenes, las muchedumbres, las ambulancias, espacios de estacionamiento, y así sucesivamente. Razonando sobre estas corrientes puede ser muy eficaz en la reducción de costos, por ejemplo: en la búsqueda de parques de estacionamiento en grandes ciudades puede costar un 40% del consumo de combustible diario; pero gracias a la informática urbana los ciudadanos pueden encontrar un parque de estacionamiento en un mínimo de tiempo, evitando congestiones de tráfico global. [13]

Definición 2. *Windows* (ventana de tiempo): Los problemas de razonamiento tradicionales son basados en la idea de que toda la información disponible debe alojarse para considerar el problema a resolver. En el *Stream Reasoning* se elimina este principio y se restringe el mismo a una cierta ventana de tiempo que introduce las *RDF streams* (en el español una posible traducción de este término es corriente de datos RDF o corriente RDF y en la investigación se decide utilizar sin traducir) como un nuevo tipo de datos de entrada con la capacidad para identificar y aplicar criterios de selección sobre ellos, mientras son observados en la corriente. La selección o extracción puede ser física (un número dado de tripletas) o lógica (un número variable de tripletas en un tiempo determinado).

Esto es necesario por diferentes razones, ya que en primer lugar, ignora las declaraciones más viejas permitiéndonos ahorrar recursos informáticos en términos de memoria y tiempo de procesamiento para reaccionar a eventos importantes en tiempo real. [12]

[7] Aplicación de la informática a ambientes urbanos.

Definición 3. *Continuous Processing* (procesamiento continuo): Los acercamientos del razonamiento tradicional son basados en la idea de que el proceso de razonamiento tiene un principio y un fin (cuando el resultado se entrega por el sistema) bien definido. En el *Stream Reasoning*, se cambia de este modelo tradicional a un modelo de proceso continuo dónde las consultas son registradas en el razonador y continuamente evaluadas contra una base de conocimiento que constantemente está cambiando. [12,14]

2.2.1 Aplicaciones de Stream Reasoning

El *Stream Reasoning* puede beneficiar numerosos ámbitos. Control de tráfico y detección de patrón de tráfico, parecen proporcionar un área de aplicación natural. A continuación podemos observar dos aplicaciones concretas del mismo. [5]

Razonamiento para aplicaciones móviles

La movilidad es una de las características definitorias de la vida moderna. La tecnología puede apoyar y acompañar a nuestra movilidad de varias maneras, tanto para los negocios y entretenimiento. Los teléfonos móviles son muy populares y generalizados, que establecen un territorio bueno para cuestionar el concepto de *Stream Reasoning*. Para estar inmerso en nuestra vida diaria, las aplicaciones móviles deben cumplir con requisitos de tiempo real, sobre todo si vamos a utilizarlos para tomar decisiones a corto plazo.

A partir de datos procedentes de sensores, que es probable que en las corrientes, las aplicaciones móviles deban encontrar respuestas a los problemas de razonamiento sobre corrientes de datos como: hacer frente a datos ruidosos que tratan los errores; moviendo pesados cálculos en el servidor en lugar de hacerlos en los dispositivos móviles, entre otras.

Lidiando con la corriente de los usuarios de experiencia, las aplicaciones móviles razonan qué parte de la información fluyente es relevante y cuál es su significado. Por ejemplo, tienen para abstraer a partir de información cuantitativa sobre latitud y longitud a la información cualitativa acerca de lugares tales como el hogar, oficina y gimnasio. Después, tienen que elevar el nivel de abstracción para razonar sobre problemas concretos, como la manera de llegar a esos lugares: medios de transporte, una ruta que evita los atascos de tráfico, y así sucesivamente. Con el tratamiento de usuarios de teléfonos móviles a sí mismos como los sensores, las aplicaciones móviles pueden proporcionar una comprensión del medio ambiente urbano y su estructura.

Vigilancia de la Salud Pública-Riesgos

La detección temprana de potencial peligroso para eventos de salud pública tales como los brotes y epidemias, es una de las principales prioridades nacionales y de las organizaciones internacionales relacionadas con la salud. Ejemplos recientes son las infecciones como el SRAS, la gripe aviar H5N1 y el virus H1N1. Tratar con esta prioridad se requiere mejorar las capacidades de detección temprana, al permitir la adquisición más relevante oportuna y completa de los datos y el avance de las tecnologías de información casi en tiempo real y la identificación foco automático.

Esto requiere un evento de salud pública con una plataforma de detección integrada que controle una gran variedad de datos para detectar eventos y situaciones que puedan, cuando se interpretan en el contexto adecuado, significar una amenaza potencial para la salud pública. Esta plataforma dinámica debe identificar, integrar e interpretar heterogéneos distribuidos flujos de datos con información que surge de estas fuentes de datos analizados automáticamente y son expresados sobre la base de conocimiento previamente. Cuando esta plataforma calcule una probabilidad de amenaza mayor, tendrá que agilizar las notificaciones a los organismos públicos sobre distintos canales de comunicación y entregar las huellas del proceso de razonamiento y los datos que llevaron al cálculo, por lo que las autoridades puedan evaluar y utilizar la información adecuadamente.

Los sistemas existentes, como el ya clásico *Google Trends* contra la gripe, procesa altos volúmenes de flujos de datos, pero todo el procesamiento semántico de estos datos tiene un lugar predefinido, en el código de forma-a priori (al integrar corrientes) o a posteriori (al interpretar los resultados). El reto consiste en hacer la transición de estos sistemas hechos a mano con el razonamiento automático sobre las corrientes de datos de magnitudes similares.

2.3 RDF

Por sus siglas en inglés *RDF* (*Resource Description Framework*) es un marco de descripción de recursos, este es un *framework* para procesar metadados en la *world wide web*, creado por W3C "*World Wide Web Consortium*". [2]

Proporciona interoperabilidad entre aplicaciones que intercambian información legible por máquina en la web. Uno de sus objetivos es hacer posible especificar la semántica para las bases de datos en *XML* de

una forma normalizada e interoperable. Mientras que su objetivo general es definir un mecanismo para describir recursos que no cree ningún enlace sobre un dominio de aplicación particular, ni defina (a priori) la semántica de algún dominio de aplicación. [15]

Este modelo se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto. [2] Su elemento de construcción básica es la "tripleta" o sentencia, que consiste en dos nodos (sujeto y objeto) unidos por un arco (predicado), donde los nodos representan recursos, es decir, es lo que se está describiendo, y los arcos son las propiedades o relaciones que se desean establecer acerca del recurso. Por último, el objeto es el valor de la propiedad o el otro recurso con el que se establece la relación. [15]

La combinación de *RDF* con otras herramientas como *RDF Schema* y *OWL* permite añadir significado a las páginas, y es una de las tecnologías esenciales de la web semántica. [2]

- **Sujeto:** Es el recurso al cual nos estamos refiriendo.
- **Predicado:** Es el recurso que indica lo que estamos definiendo.
- **Objeto:** Puede ser un recurso que puede considerarse el valor definido.

En el siguiente ejemplo veremos la página de W3C en Wikipedia:

```
<http://en.wikipedia.org/wiki/World_Wide_Web_Consortium>
<http://purl.org/dc/elements/1.1/title> "W3C".

<http://en.wikipedia.org/wiki/World_Wide_Web_Consortium>
<http://purl.org/dc/elements/1.1/publisher> "Wikipedia".
```

Expresado en RDF/XML sería:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="
http://en.wikipedia.org/wiki/World_Wide_Web_Consortium ">
```

```
  <dc:title>W3C</dc:title>
  <dc:publisher>Wikipedia</dc:publisher>
</rdf:Description>
</rdf:RDF>
```

2.3.1 RDF streams

Una *RDF stream* se puede definir mediante la extensión de tipo de datos *RDF* con una noción de fecha y hora. Constituye una secuencia ordenada de pares, donde cada par es constituido por una tripleta *RDF* y su marca de tiempo T. [14] Es decir:

...

$$(<Subj_{yo}\ Pred_{yo,}\ Obj_{yo}>,\ T_{yo})$$

$$(<Subj_{i+1},\ Pred_{i+1,}\ Obj_{i+1}>,\ T_{i+1})$$

...

Las marcas de tiempo se pueden considerar en el contexto de tripletas *RDF*. Estas son monótonamente no decrecientes en la corriente ($T_{yo} \leq T_{i+1}$), y son estrictamente crecientes porque no están obligadas a ser únicas. Cualquier número de tripletas consecutivas pueden tener la misma marca de tiempo, lo cual significa que se producen al mismo tiempo, aunque la secuencia en la corriente dependa de su orden posicional. [16]

Las *RDF streams* garantizan la interoperabilidad y abren importantes aplicaciones, en las que los razonadores pueden hacer frente a la evolución del conocimiento a través del tiempo. Un ejemplo de ello es el razonamiento sobre los sensores, la informática urbana y social de datos semánticos. [10]

2.4 C-Sparql

Continuous Sparql (*C-Sparql*) es un nuevo lenguaje para consultas continuas sobre gráficos *RDF* y corrientes de datos en formato *RDF*. Constituye una extensión de *Sparql* cuya característica distintiva es el soporte de las consultas continuas, es decir las consultas son registradas y luego se ejecutan continuamente sobre ventanas de tiempo abiertas en las corrientes de datos en formato *RDF* y gráficos *RDF* instantáneos. [10,17]

C-Sparql se presenta por medio de una especificación completa de la sintaxis, la semántica formal, y un conjunto completo de ejemplos, en relación con las aplicaciones de la informática urbana, que sistemáticamente cubren las extensiones de *Sparql*. [10,17]

Este lenguaje está basado en dos tipos de datos: [13]

1. Corrientes de datos *RDF* (*RDF stream)*
2. Gráfico *RDF* instantáneo *(tgraph),* que no es más que un conjunto de tripletas *RDF* o conjunto de datos *RDF*.

Estos datos son un mapeo directo de las mismas corrientes *(streams).*

Las consultas *C-Sparql* son ejecutadas como árboles de operadores, realizando selección y abstracción sobre corrientes de datos. Para ello este lenguaje utiliza los siguientes operadores: [13]

- *tgraph-to-tgraph*: Operador que toma uno o más gráficos *RDF (tgraph)* como entrada y produce un gráfico *RDF (tgraph)* como salida.
- *RDF stream-to-tgraph*: Operador que toma las declaraciones de una corriente de datos *RDF (RDF stream)* como entrada y produce un gráfico *RDF (tgraph)* como salida.
- *tgraph-to-RDF stream*: Operador que toma un gráfico *RDF (tgraph)* como entrada y produce una corriente de datos *RDF (RDF stream)* como salida.

2.4.1 Consultas continuas

La expresión de consultas significativas en las corrientes de datos, está estrictamente vinculada con la disponibilidad de las primitivas de agregación, por lo que *C-Sparql* incluye extensiones al respecto. Las consultas continuas, que hacen uso de los agregados, son particularmente relevantes.

Una consulta de *C-Sparql* se registra utilizando la extensión de la gramática proporcionada por la regla de producción siguiente:

Registration → *'REGISTER QUERY' QueryName 'AS' Query*

Como salida, las consultas de *C-Sparql* producen las mismas salidas que las consultas Sparql: respuestas booleanas, selecciones de asignaciones de variables, descripciones *RDF* de recursos involucrados o construcciones de nuevas tripletas *RDF*. Estas salidas son continuamente renovadas en cada ejecución de la consulta.

Además, las consultas de *C-Sparql* pueden ser registradas para producir nuevas corrientes *RDF*, usando la extensión de la gramática siguiente:

Registration → *'REGISTER STREAM' QueryName 'AS' Query*

En este caso solo consultas de *CONSTRUCT* y *DESCRIBE* pueden ser registradas, cuando producen las tripletas *RDF* que una vez asociadas con una marca de tiempo, pueden ser manejadas por *C-Sparql* en corrientes de datos RDF. [16]

2.4.2 Windows

La noción de *Windows* (ventana de tiempo) en *data streams*, es introducida con los tipos y características de las ventanas de tiempo de los lenguajes de consulta continua como, *CQL*.

La identificación y la selección, son expresadas en *C-Sparql* por medio de la cláusula *FROM STREAM,* teniendo en cuenta que cada dato en la corriente está asociado a un *IRI*[8].

Esta selección e identificación se realiza como se muestra a continuación.

FromStrClause → *'FROM' ['NAMED'] 'STREAM' StreamIRI '[RANGE 'Window '] '*

De la corriente identificada por *StreamIRI*, una ventana extrae las últimas tripletas que son consideradas por la consulta. La selección o extracción puede ser física (un número dado de tripletas) o lógica (un número variable de tripletas que ocurren dado un intervalo de tiempo determinado). [16]

2.4.3 Agregación

C-Sparql permite múltiples agregaciones independientes dentro de la misma consulta. Las cláusulas de agregación tienen la siguiente sintaxis: [16]

[8] Localizador del recurso del dato actual de la corriente. Representa más bien la dirección y el puerto para acceder al dato.

AggregateClause → *('AGGREGATE {(' var ',' Fun ',' Group ')' [Filter] '}')**

Fun → *'COUNT' | 'SUM' | 'AVG' | 'MIN' | 'MAX'*

Group → *var | '{' var (',' var)* '}'*

Una cláusula de agregación empieza con una nueva variable que no ocurre en la cláusula *WHERE*, seguido por una función de agregación y cerrado por un conjunto de una o más variables, ocurridas en la cláusula *WHERE*, que expresan un grupo de criterios; y una cláusula *FILTER* opcional.

La semántica de una consulta con función de agregación consiste en agregar lazos entre las variables de la cláusula *WHERE* y otras no producidas dentro de esta, introducidas por la cláusula *AGGREGATE*. La solución construida en esta manera puede ser iterada a través de la cláusula *FILTER*. Las evaluaciones de las funciones de agregación son totalmente independientes unas de otras.

2.4.4 Función de marca de tiempo

La marca de tiempo de un elemento de la corriente puede ser recuperada o limitada a una variable usando una función de marca de tiempo.

Las funciones de marcas de tiempo tienen dos argumentos:

1. El nombre de una variable, introducida en la cláusula *WHERE* y limitada a una tripleta *RDF* de la corriente como resultado del modelo de macheo.
2. La *URI* de una corriente, que puede ser obtenida a través de una cláusula *GRAPH* de *Sparql*.(argumento opcional)

La función retorna la marca de tiempo del elemento *RDF* de la corriente, que resulta de la evaluación de la consulta.

- Si la variable no es limitada, la función es indefinida y una comparación involucrando esta evaluación tiene un comportamiento no determinado.
- Si la variable es limitada, la función retorna el valor de marca de tiempo más reciente relativo con el tiempo de evaluación de la consulta.

2.4.5 Ventajas de C-Sparql

C-Sparql y su correspondiente infraestructura constituye un punto de partida excelente para el *Stream Reasoning*. Ofrece múltiples ventajas entre las que se encuentran: [10,17]

- Proporciona un *RDF* básico (*RDF-based)* a la hora de representar *RDF streams* heterogéneas, resolviendo el desafío de dar a los razonadores un protocolo de acceso para este tipo de corrientes heterogéneas parcialmente.
- Como *RDF* es el formato más aceptado y utilizado por los razonadores, *C-Sparql* permite extender los diferentes mecanismos de razonamiento existentes más allá, para apoyar el *Stream Reasoning* y la base de conocimiento enriquecida.
- Las consultas de *C-Sparql* pueden ser evaluadas usando una arquitectura simple basada en decisión que guarde la administración de la corriente por un lado y la evaluación de la misma por otro.
- Constituye la única extensión de *Sparql* que soporta agregados, permitiendo siempre posibles optimizaciones en agregaciones como cierre de posibles corrientes de datos.

2.4.6 Desventajas de C-Sparql

Este lenguaje presenta como desventajas:

- No puede tomar ventaja de una consulta pre-procesada y/o una optimización sobre el espacio unificado de datos.
- No realiza una correcta distribución de los datos, a la hora de dar la respuesta de las consultas.

2.5 Casos de prueba al lenguaje C-Sparql

Las pruebas que a continuación se describen se realizaron basadas en los requisitos funcionales del lenguaje *C-Sparql*. A continuación se presentan los mismos:

Requisitos Funcionales:

RF1-Identificar la corriente de datos.

RF2- Seleccionar datos de la corriente.

RF3- Registrar la corriente de datos.

RF4- Registrar consulta.

RF5- Ejecutar consulta.

RF6- Agregar marca de tiempo.

Requisitos no Funcionales:

RNF1- Se requiere de conexión a Internet.

A continuación se describen los casos de pruebas realizados.

Caso de prueba al requisito identificar la corriente de datos

1. Descripción general

Este caso de prueba permite al lenguaje identificar la corriente de datos con la cual evaluará la consulta.

2. Condiciones de ejecución

Debe existir una corriente de datos *RDF*.

Tabla 1. Secciones de identificar la corriente de datos

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Identificar la corriente de datos	EC 1.1: Identificar corriente de datos	El lenguaje identifica la corriente de datos con la cual realizara la consulta.	El lenguaje a través de la cláusula *FROM STREAM* identifica la corriente de datos a utilizar para evaluar la consulta.

Tabla 2. Identificar la corriente de datos

ID del escenario	**Escenario**	**Variable 1** Corriente *RFD*	**Variable 2** *StreamIRI*	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Identificar corriente de datos	V Tripleta	V Dirección	Identifica la corriente de datos.	Satisfactorio, ya que para identificar la corriente el lenguaje debe contar con una corriente de datos y una dirección *(StreamIRI)* y en este caso las dos son validas.
		I Tripleta	I Dirección	No identifica la corriente de datos	Insatisfactorio, ya que para identificar la corriente el lenguaje debe contar con una corriente de datos y una dirección *(StreamIRI)* y en este caso las dos no son válidas.
		V Tripleta	I Dirección	No identifica la corriente de datos	Insatisfactorio, ya que para identificar la corriente el lenguaje debe contar con una corriente de datos y una dirección *(StreamIRI)* y en este caso la corriente de datos es válida pero la dirección no.

		I Tripleta	V Dirección	No identifica la corriente de datos	Insatisfactorio, ya que para identificar la corriente el lenguaje debe contar con una corriente de datos y una dirección *(StreamIRI)* y en este caso la dirección es valida pero la corriente de datos no.

Caso de prueba al requisito seleccionar datos de la corriente

1. Descripción general

Este caso de prueba permite al lenguaje seleccionar los datos adecuados de la corriente de forma lógica o física que serán utilizados en la consulta.

2. Condiciones de ejecución

Debe existir una corriente de datos de tipo *RDF*.

Tabla 3. Secciones de seleccionar datos de la corriente

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Seleccionar datos de la corriente	EC 1.1: Seleccionar datos de forma física.	El lenguaje selecciona los datos de la corriente.	El lenguaje a través de la ventana de tiempo selecciona los datos de forma física utilizando un número dado de tripletas a seleccionar.
	EC 1.2: Seleccionar datos de forma lógica.	El lenguaje selecciona los datos de la corriente.	El lenguaje a través de la ventana de tiempo selecciona los datos de forma lógica, seleccionando una cantidad de tripletas dado un tiempo determinado.

Tabla 4. Seleccionar datos de la corriente

ID del escenario	**Escenario**	**Variable 1** Corriente *RFD*	**Variable 2** Número de tripletas	**Variable 3** Tiempo	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Seleccionar datos de forma física.	V Tripleta	V Número	NA	Selecciona la tripleta *RDF*.	Satisfactorio ya que para seleccionar los datos de la corriente de forma física el lenguaje debe contar con una corriente de datos y un número de tripletas a seleccionar y en este caso los dos son válidos.
		I Tripleta	I Número	NA	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma física el lenguaje debe contar con una corriente de datos y un número de tripletas a seleccionar y en este caso los dos no son válidos.

		V Tripleta	I Número	NA	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma física el lenguaje debe contar con una corriente de datos y un número de tripletas a seleccionar y en este caso la corriente es válida pero el número de tripletas no.
		I Tripleta	V Número	NA	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma física el lenguaje debe contar con una corriente de datos y un número de tripletas a seleccionar y en este caso el número es válido pero la corriente no.

1.2	Seleccionar datos de forma lógica.	V Tripleta	NA	V Número	Selecciona la tripleta RDF.	Satisfactorio ya que para seleccionar los datos de la corriente de forma lógica el lenguaje debe contar con una corriente de datos y un tiempo determinado para poder seleccionar las tripletas, y en este caso los dos son válidos.
		I Tripleta	NA	V Número	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma lógica el lenguaje debe contar con una corriente de datos y un tiempo determinado para poder seleccionar las tripletas, y en este caso el tiempo es válido pero la corriente no.

		V Tripleta	NA	I Número	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma lógica el lenguaje debe contar con una corriente de datos y un tiempo determinado para poder seleccionar las tripletas, y en este caso la corriente es válida pero el tiempo no.
		I Tripleta	NA	I Número	No debe seleccionar la tripleta.	Insatisfactorio ya que para seleccionar los datos de la corriente de forma lógica el lenguaje debe contar con una corriente de datos y un tiempo determinado para poder seleccionar las tripletas, y en este caso los dos no son válidos.

Caso de prueba al requisito registrar la corriente de datos

1. Descripción general

Este caso de prueba permite al lenguaje registrar o almacenar la corriente de datos resultante luego de

evaluada la consulta para ser posteriormente utilizada.

2. Condiciones de ejecución

Debe existir una corriente de datos de tipo *RDF*, para poder evaluar la consulta.

Debe haberse ejecutado la consulta y haber obtenido una corriente de datos como resultado.

Tabla 5. Secciones de registrar la corriente de datos

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Registrar la corriente de datos	EC 1.1: Registrar la corriente de datos como resultado de consulta.	El lenguaje registra la corriente de datos.	El lenguaje a través de la cláusula *REGISTER STREAM* registra la corriente resultante luego de evaluada la consulta sobre la corriente de datos original.

Tabla 6. Registrar la corriente de datos

ID del escenario	**Escenario**	**Variable 1** Corriente *RFD* obtenida	**Variable 2** Ejecución de Consulta	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Registrar la corriente de datos como resultado de consulta	V Tripleta	V Ejecución	Se registra la corriente resultante.	Satisfactorio ya que para registrar la corriente de datos como resultado de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido una corriente de datos como resultado y en este caso los dos son válidos.
		I Tripleta	I Ejecución	No se debe registrar la corriente.	Insatisfactorio ya que para registrar la corriente de datos como resultado de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido una corriente de datos como resultado y en este caso los dos no son válidos.

		I Tripleta	V Ejecución	No se debe registrar la corriente.	Insatisfactorio ya que para registrar la corriente de datos como resultado de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido una corriente de datos como resultado y en este caso se ejecutó la consulta pero no se obtuvo una corriente de datos como resultado.

Caso de prueba al requisito registrar consulta

1. Descripción general

Este caso de prueba permite al lenguaje registrar el resultado de una consulta luego de ser evaluada la misma.

2. Condiciones de ejecución

Debe existir una corriente de datos de tipo *RDF*, para poder evaluar la consulta.

Debe haberse ejecutado la consulta y haber obtenido un resultado.

Tabla 7. Secciones de registrar consulta

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Registrar la consulta	EC 1.1: Registrar el resultado de la consulta.	El lenguaje registra el resultado de la consulta.	El lenguaje a través de la cláusula *REGISTER QUERY* registra el resultado de la consulta evaluada.

Tabla 8. Registrar consulta

ID del escenario	**Escenario**	**Variable 1** Dato *RFD* obtenido	**Variable 2** Ejecución de Consulta	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Registrar el resultado de la consulta.	V Tripleta	V Ejecución	Se registra el resultado de la consulta.	Satisfactorio ya que para registrar el resultado de la consulta el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido un resultado y en este caso los dos son válidos.
		I Tripleta	I Ejecución	No se debe registrar el resultado.	Insatisfactorio ya que para registrar el resultado de la consulta el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido un resultado y en este caso los dos no son válidos.
		I Tripleta	V Ejecución	No se debe registrar el resultado.	Insatisfactorio ya que para registrar el resultado de la consulta el lenguaje debe haber ejecutado la consulta satisfactoriamente y obtenido un resultado y en este caso se ejecutó la consulta pero no se obtuvo un resultado.

Caso de prueba al requisito ejecutar consulta

1. Descripción general

Este caso de prueba permite al lenguaje ejecutar la consulta sobre los datos seleccionados previamente y lo puede hacer de forma estándar o continuamente a través de una frecuencia de tiempo.

2. Condiciones de ejecución

Deben haberse seleccionado los datos de la corriente correspondiente para la ejecución de la consulta.

Tabla 9. Secciones de ejecutar consulta

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Ejecutar consulta	EC 1.1: Ejecutar consulta con datos seleccionados	El lenguaje ejecuta la consulta con los datos seleccionados de la corriente de datos.	El lenguaje a través del conjunto de cláusulas como *SELECT, WHERE*, entre otras, realiza la evaluación de la consulta sobre los datos seleccionados de la corriente.
	EC 1.2: Ejecutar consulta con datos seleccionados continuamente	El lenguaje ejecuta la consulta con los datos seleccionados de la corriente de datos continuamente.	El lenguaje a través del conjunto de cláusulas como *SELECT, WHERE*, entre otras, realiza la evaluación de la consulta sobre los datos seleccionados de la corriente y lo realiza continuamente con una frecuencia de tiempo especificada en la cláusula *REGISTER*.

Tabla 10. Ejecutar consulta

ID del escenario	**Escenario**	**Variable 1** Datos *RDF* seleccionados	**Variable 2** Frecuencia de Tiempo	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Ejecutar consulta con datos seleccionados	V Tripletas	NA	Se ejecuta la consulta.	Satisfactorio ya que para ejecutar la consulta el lenguaje debe haber seleccionado los datos para poder evaluar la consulta y en este caso es válido.
		I Tripletas	NA	No se ejecuta la consulta.	Insatisfactorio ya que para ejecutar la consulta el lenguaje debe haber seleccionado los datos para poder evaluar la consulta y en este caso no es válido.
1.2	Ejecutar consulta con datos seleccionados continuamente	V Tripletas	V Número	Se ejecuta la consulta.	Satisfactorio ya que para ejecutar la consulta el lenguaje debe haber seleccionado los datos para poder evaluar la consulta y contar con la frecuencia de tiempo con que va a realizar la consulta y en este caso los dos son válidos.

		I Tripletas	I Número	No se ejecuta la consulta.	Insatisfactorio ya que para ejecutar la consulta el lenguaje debe haber seleccionado los datos para poder evaluar la consulta y contar con la frecuencia de tiempo con que va a realizar la consulta y en este caso los dos no son válidos.
		I Tripletas	V Número	No se ejecuta la consulta.	Satisfactorio ya que para ejecutar la consulta el lenguaje debe haber seleccionado los datos para poder evaluar la consulta y contar con la frecuencia de tiempo con que va a realizar la consulta y en este caso la frecuencia es válida pero no se seleccionaron los datos.

Caso de prueba al requisito agregar marca de tiempo

1. Descripción general

Este caso de prueba permite al lenguaje agregar marcas de tiempo a los datos resultantes luego de ser ejecutada la consulta.

2. Condiciones de ejecución

Debe haberse ejecutado la consulta y obtenido un resultado.

Debe haberse seleccionado la variable que se utilizará en la función de marca de tiempo.

Tabla 11. Secciones de agregar marca de tiempo

Sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
1 Agregar marca de tiempo	EC 1.1: Agregar marca de tiempo a elementos.	El lenguaje agrega marcas de tiempo a los elementos resultantes luego de ejecutarse la consulta.	El lenguaje a través de una función de marca de tiempo agrega las marcas de tiempo a los elementos resultantes de la ejecución de la consulta.

Tabla 12. Agregar marca de tiempo

ID del escenario	**Escenario**	**Variable 1** Ejecución de la Consulta	**Variable2** Variable de función de marca de tiempo	**Respuesta del sistema**	**Resultado de la prueba**
1.1	Agregar marca de tiempo a elementos.	V Ejecución	V Limitada	Se agrega la marca de tiempo al resultado de la consulta.	Satisfactorio ya que para agregar la marca de tiempo a los elementos resultantes de la evaluación de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y haber evaluado la

					función de marca de tiempo con una variable limitada introducida en la cláusula *WHERE* y en este caso los dos son válidas.
		V Ejecución	I Ilimitada	No se agrega la marca de tiempo.	Insatisfactorio ya que para agregar la marca de tiempo a los elementos resultantes de la evaluación de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y haber evaluado la función de marca de tiempo con una variable limitada introducida en la cláusula *WHERE* y en este caso la consulta se ejecuta pero la variable no es limitada.
		I Ejecución	I Ilimitada	No se agrega la marca de tiempo.	Insatisfactorio ya que para agregar la marca de tiempo a los elementos resultantes

					de la evaluación de la consulta, el lenguaje debe haber ejecutado la consulta satisfactoriamente y haber evaluado la función de marca de tiempo con una variable limitada introducida en la cláusula *WHERE* y en este caso las dos no son válidas.

2.6 Análisis de los resultados de las pruebas

Al realizar un análisis de los resultados de las pruebas al lenguaje basadas en sus requisitos funcionales, podemos observar que:

- Para que el lenguaje pueda realizar un buen razonamiento debe contar principalmente con una corriente de datos, la cual se identificará y de ella seleccionarán los datos correspondientes para evaluar la consulta.
- Para lograr un buen registro ya sea de una consulta o de una corriente, debe ejecutarse la consulta satisfactoriamente y haber obtenido un resultado en ambos casos.
- Para lograr una buena ejecución de la consulta, deben seleccionarse los datos de la corriente y en dependencia si la evaluación es de forma estándar o continua, constar con una frecuencia de tiempo que es la que posibilitará que la consulta se ejecute continuamente.
- Para agregar las marcas de tiempo a los resultados obtenidos de la consulta, debe ejecutarse la misma satisfactoriamente obteniendo un resultado y contar con una variable limitada introducida en

la cláusula *WHERE*, la cual se evaluará en la función de marca de tiempo y devolverá la marca de tiempo correspondiente al elemento.

2.7 Conclusiones

Luego de analizar el proceso de razonamiento sobre corrientes de datos y los resultados obtenidos al realizar las pruebas al lenguaje *C-Sparql* se llega a las siguientes conclusiones:

- Las pruebas realizadas al lenguaje son la guía para la confección del modelo, porque ofrecen un mejor entendimiento de cómo razona el mismo lenguaje sobre corrientes de datos.
- Los requisitos funcionales del lenguaje y las cosas esenciales del proceso de razonamiento sobre corrientes de datos (*Stream Reasoning),* deben estar reflejados en el modelo a desarrollar.

CAPÍTULO 3: PROPUESTA DEL MODELO

3.1 Introducción

En este capítulo se realiza la propuesta del modelo obtenido para el razonamiento sobre corrientes de datos basado en el lenguaje *C-Sparql*, y se describe el funcionamiento interno del mismo.

3.2 Propuesta de modelo

El modelo propuesto para razonar sobre corrientes de datos, está basado en la arquitectura interna del lenguaje de consultas sobre corrientes *RDF*: *C-Sparql*. Este lenguaje para razonar sobre corrientes de datos, utiliza un Analizador de Consultas que toma como entrada una consulta *C-Sparql* y provee la información necesaria a las capas posteriores para la ejecución de la misma.

El razonamiento sobre los datos se compone de dos partes fundamentales: la parte continua y la parte estática donde se realiza la consulta *Sparql* como tal. En la parte continua (Capa de Administración de Corrientes de Datos) es donde se registran los datos de la corriente especificada en la consulta (la que se encuentra de forma completa en el Analizador de Consultas *C-Sparql*) a través del *FROM STREAM* y se aplica una ventana de tiempo física o lógica.

Como resultado de ese registro se obtiene uno o más gráficos *RDF* (se utiliza el operador ***RDFstream-to-tgraph D***) y se pasa a la parte de realización de la consulta (Capa Final de *Sparql*) sobre estos gráficos *RDF* como tal (se utiliza el operador ***tgraph-to-tgraph D***). Se ejecuta la consulta *Sparql*, pero con una frecuencia de tiempo especificada en la cláusula *REGISTER* de la parte continua, que es lo que hace que la consulta se realice continuamente o periódicamente. Como resultado de esto se obtiene uno o más gráficos *RDF* pero con una marca de tiempo que se toma del tiempo de ejecución de la consulta. Por último se pasa a procesar nuevamente la corriente (se utiliza el operador ***tgraph-to-RDFstream***).

Lo novedoso de este modelo es que incorpora operadores para la transformación de la información, tomando como base los operadores definidos para el lenguaje C-Sparql pero con la diferencia que van a trabajar los datos de forma distribuida, es decir que a la hora de realizar una determinada transformación se realizará distribuyendo los mismos. Para ello utilizan el *framework MapReduce*, que se encarga de distribuir la información de tal forma que esté lógicamente relacionada facilitando el manejo y tratamiento de la misma. Este *framework, es* introducido por *Google* para dar soporte a la computación paralela sobre

grandes colecciones de datos en grupos de computadoras. [18] Utiliza dos funciones, ambas definidas con respecto a datos estructurados en pares (clave, valor):

- *Función Map*: Toma uno de estos pares de datos de entrada con un tipo en un dominio de datos, y devuelve una lista de pares del mismo tipo pero con dominios diferentes: Map (k1, v1) -> list (k2, v2). Esta función se aplica en paralelo para cada ítem en la entrada de datos.[18]

Luego de aplicar la función *Map,* el *framework* junta todos los pares con la misma clave de todas las listas y los agrupa, creando un grupo por cada una de las claves diferentes generadas, y se pasa a la función *Reduce*.

- *Función Reduce*: Es aplicada en paralelo para cada grupo. Produce una colección de valores para cada dominio, fusionando estos valores para formar un grupo posiblemente más pequeño de valores. Reduce (k2, list (v2)) -> list(v2). [18]

Los operadores para la transformación de los datos se definen de la siguiente forma:

- ***tgraph-to-tgraph D*:** Operador que toma uno o más gráficos *RDF* (tgraph) como entrada, los transforma utilizando *MapReduce* y produce un gráfico *RDF (tgraph)* como salida.
- ***RDF stream-to-tgraph D*:** Operador que toma como entrada una corriente de datos *RDF* (*RDF stream*), la trasnforma utilizando *MapReduce* y produce un gráfico RDF (*tgraph*) como salida.
- ***tgraph-to-RDF stream*:** Operador que toma un gráfico *RDF (tgraph)* como entrada y produce una corriente de datos *RDF (RDF stream)* como salida.

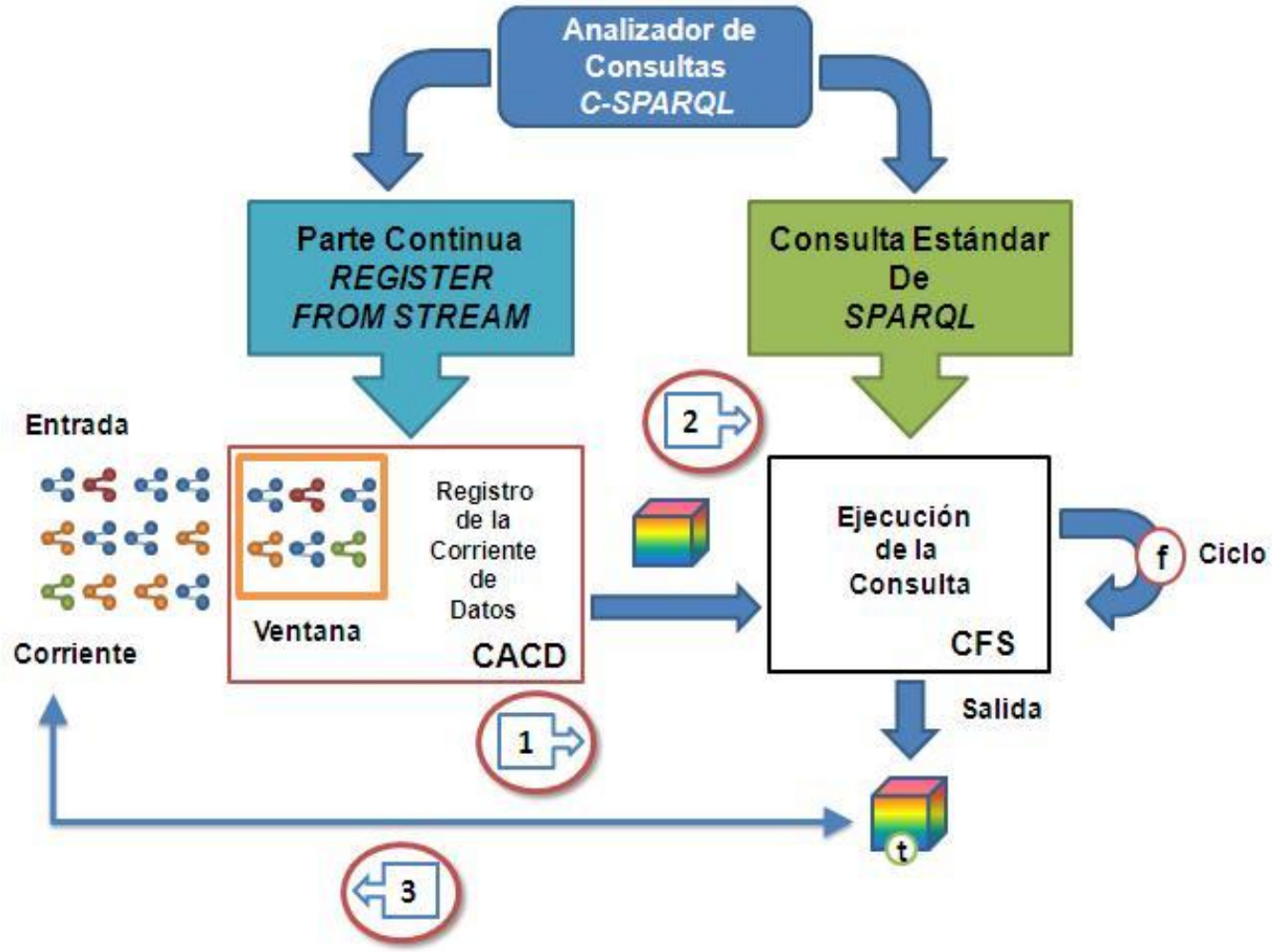
Para la transformación de gráfico *RDF (tgraph)* a corriente de datos (*RDF stream*), no es necesario realizar distribución ya que lo que se transforma es el resultado de la consulta evaluada para nuevamente procesar la corriente y es por ello que se utiliza el operador *tgraph-to-RDFstream.* Con la introducción de estos nuevos operadores, se logra que el razonamiento sea escalable y eficiente al mismo tiempo.

Al *C-Sparql* trabajar los datos *RDF* en dos partes (la parte continua y la estática), sin tener que almacenar la corriente de datos, es decir, coge los datos seleccionados a través de la ventana, los transforma, realiza

la consulta sobre ellos y devuelve un resultado; y vuelve a procesar la corriente nuevamente según el tiempo especificado de frecuencia. Esto hace que sea más eficiente en cuanto a tiempo de respuesta, ya que la selección de las tripletas *RDF* significativas se realiza mucho mas rápido que en comparación con otros lenguajes de consulta sobre corrientes de datos como por ejemplo el Sparql, que solamente selecciona las tripletas, las procesa y devuelve un resultado, y para procesar nuevamente la corriente de datos y seleccionar las tripletas hay que ejecutar nuevamente la consulta. Que al razonar el *C-Sparql* lo realice continuamente posibilita que devuelva muchos más resultados en menos tiempo.

Como simbología del modelo propuesto tenemos:

A continuación podemos observar el modelo antes descrito.



Este modelo constituye un modelo de infraestructura *Stream Reasoning,* porque cumple con los requisitos planteados para este tipo de infraestructura como son:

1. Selección de datos.
2. Abstracción por extracción de la información, transformando la misma.
3. Razonamiento sobre conocimiento.
4. Decidir si una nueva iteración es necesitada.

Estos requisitos se evidencian de la siguiente forma: la selección de los datos se realiza a través de la ventana de tiempo física o lógica, se transforman los datos obtenidos aplicando los operadores, se razona sobre ellos obteniéndose un resultado y se decide según la frecuencia de tiempo especificada si una nueva iteración es precisa.

3.3 Casos de estudio

Los siguientes casos de estudio muestran la aplicación del modelo propuesto al analizar corrientes de datos.

3.3.1 Parques de estacionamiento

La informática urbana, trata del uso (y generación) de información física y digital sobre la ciudad, brindando aplicaciones que puedan ser usadas por sus ciudadanos a través de dispositivos fijos o móviles que leen esa información. Para ello utiliza las corrientes de datos, representando objetos reales que se supervisan dada una determinada situación como: automóviles, trenes, muchedumbres, ambulancias, espacios de estacionamiento, y así sucesivamente. Razonar sobre estas corrientes puede ser muy eficaz en la reducción de costos y un ejemplo de ello es en la búsqueda de parques de estacionamiento en grandes ciudades.

Aplicando el modelo a la solución de este problema podemos observar que se recibe la corriente de datos con los parques de estacionamiento de la ciudad, su ubicación en la misma y la capacidad en cuanto a autos. Primeramente se seleccionan los datos a través de una ventana lógica, ya que el resultado se espera en un tiempo limitado. Se transforman los mismos a través de los operadores permitiendo que la consulta se evaluada fácilmente, la cual seleccionará el parque de estacionamiento que se encuentre más cerca a la ubicación del dispositivo por el cual el ciudadano solicita el servicio, pero teniendo en cuenta que exista capacidad en él. Esta consulta se ejecutará continuamente hasta cumplirse la frecuencia de tiempo, logrando que la solución sea bastante acierta, en comparación con un razonamiento tradicional que al no procesar los datos continuamente obvia resultados relevantes, por lo que la efectividad de este modelo es mucho mayor. Como resultado de la evaluación de la consulta se obtiene el parque de estacionamiento con capacidad más cercano al ciudadano que solicita del servicio.

La realización de esta operación de encontrar el parque de estacionamiento físicamente puede costar un 40% del consumo de combustible diario; pero gracias a la informática urbana los ciudadanos pueden encontrar un parque de estacionamiento en un mínimo de tiempo, evitando congestiones de tráfico global y ahorrando combustible.

3.3.2 Informe de riesgo volcánico

La totalidad de volcanes activos que pueden afectar a la Argentina se encuentran ubicados en la cadena Andina. Durante una erupción los volcanes emiten dos tipos de productos de riesgo que según su alcance se denominan: proximales (lavas, flujos piroclásticos y lahares o flujos de barro), que afectan en una distancia aproximada de 100 Km desde el volcán a pequeñas localidades cordilleranas; y distales (lluvia de cenizas) que afectan grandes áreas del territorio y pueden alcanzar distancias del orden de los miles kilómetros. Estos productos son de alto riesgo para la aeronavegación. Para el estudio y seguimiento de ambos productos así como para la elaboración de los correspondientes mapas de riesgo se utiliza la información provista por las imágenes satelitales.

Aplicando el modelo propuesto a la detección de riesgo volcánico podemos observar que se reciben los datos procesados a través de un satélite. Se tiene una corriente con los datos de los volcanes (altura, temperatura, masa) que constituyen la pluma eruptiva y si el volcán posee nube de cenizas. Los datos se seleccionan a través de una ventana lógica, ya que el resultado se espera en un tiempo determinado. Se transforman los mismos a través de los operadores del modelo para poder se evaluados fácilmente en la consulta, que seleccionará el o los volcanes con riesgo a erupción, teniendo en cuenta que el satélite detecte una nube de cenizas, pluma volcánica y su temperatura sea mayor de 500 grados. La consulta se ejecutará con una frecuencia de tiempo que permitirá que la consulta se ejecute continuamente y que se obtengan más resultados que en un razonamiento tradicional, que al no ser continuo puede pasar por alto resultados relevantes.

La utilización de este modelo de razonamiento permite que se detecte de una manera más fácil y temprana el riesgo a erupción de cierto volcán y con ello se puedan tomar las medidas precisas en cuanto a la aeronavegación en esa zona, como también en las zonas urbanas cercanas al mismo, evitando catástrofes.

3.4 Conclusiones

Luego de obtener el modelo para razonamiento sobre corrientes de datos (*Stream Reasoning*) basado en el lenguaje *C-Sparql,* se llega a las siguientes conclusiones:

- El modelo constituye la base para una futura arquitectura *Stream Reasoning* por los requisitos que cumple.
- El modelo es eficiente en el razonamiento sobre corrientes de datos (*Stream Reasoning*) ya que al estar basado en el lenguaje *C-Sparql* y realizar el razonamiento continuamente, lo hace más efectivo en comparación con otros lenguajes de consulta.
- El modelo al incorporar operadores que trabajen la información de forma distribuida, permite que el manejo y tratamiento de la misma sea mucho más rápido.

CONCLUSIONES GENERALES

Luego de la culminación del trabajo se llegan a las siguientes conclusiones:

- De los lenguajes de consulta para razonar sobre corrientes de datos se escoge *C-Sparql* para la confección del modelo por las herramientas que brinda para razonar sobre corrientes de datos.
- Se desarrolló un modelo de razonamiento sobre corrientes de datos basado en el lenguaje *C-Sparql* como modelo de infraestructura de razonamiento corriente (*Stream Reasoning)* que mejora el razonamiento sobre estas corrientes.
- El modelo al incorporar operadores que transformen la información de forma distribuida posibilita que el razonamiento sobre corrientes de datos sea escalable y eficiente al mismo tiempo.

RECOMENDACIONES

Se recomienda:

- Utilizar el presente trabajo como punto de partida para próximas investigaciones en el mejoramiento del razonamiento sobre corrientes de datos, ya que constituye un tema importante, novedoso y que constantemente está evolucionando.

- Probar y aplicar el modelo en la infraestructura creada en el grupo de investigación Kahos de la Universidad de Málaga, España, a raíz de la tesis doctoral de la tutora de este trabajo de diploma, la MSc. Liudmila Reyes.

REFERENCIAS BIBLIOGRÁFICAS

1. **Becerra Z. S.,** (1999) *Bases de datos inteligentes.* Tesis que para obtener el grado de Maestro en Ciencias, Área de Computación, Facultad de Ingenería Mecánica y Eléctrica, Universidad de Colima, Colima, Coquimatlán.

2. **Pérez V. D.,** (26 jun. 2007). *Web Semántica y sus principales características*, Maestros del Web, Artículo. Extraído el 20 de nov. 2011 de http://www.maestrosdelweb.com/editorial/web-semantica-y-sus-principales-caracteristicas/.

3. **Lapuente L. M. J.,** *Hipertexto: El nuevo concepto de documento en la cultura de la imagen*. [en línea]: *Hacia la Web Semántica.*19 Nov. 2011. < http://www.hipertexto.info/documentos/web_semantica.htm > [Consultada: 23 Nov. 2011].

4. **Giraldo P.J.E.,** (feb. 2009) *Estudio Comparativo de Lenguajes para la Búsqueda y Recuperación de Información Semántica*. Politécnico Colombiano Jaime Isaza Cadavid, Medellín – Colombia. GRINSOFT: Grupo de Investigación en Desarrollo de Software. Revista Digital Sociedad de la Información. Edita Cefalea. Nº 15.

5. **Della V. E., Ceri S., Harmelen F., Fensel D.,** (2009) *It's a Streaming World! Reasoning upon Rapidly Changing Information* .IEEE de Sistemas Inteligentes de 24 (6): 83-89 BibTeX.

6. **Davis, R, et al**. (1993.)*What is a Knowledge representation?* AAAI Journal. Primavera.

7. **Ceccaroni L.,** (2008). *Inteligencia Artificial: Introducción a la representación del conocimiento.* Primavera.

8. **Echarte P.,** *Técnicas y Lenguajes para la Representación del Conocimiento*, [en línea] *eslomas.com.*14dec.2006 <http://www.eslomas.com/2006/12/tecnicas-y-lenguajes-para-la-representacion-del-conocimiento/#comments >,[Consultada: 10 de ene. 2012].

9. **Anicic D., Fodor P., Rudolph S., Stojanovic N.,** (2010) *EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning.* FZI Research Center for Information Technology Karlsruhe, Germany.

10. **Francesco B. D., Braga D., Ceri S., Della V. E., Grossniklaus M.,** (2009). *Querying RDF Streams with C-SPARQL.* Departamento de Electrónica e Informática. Politécnico de Milano. ISWC 2009. BibTeX.

11. **Unel G., Roman D.,** (2009) *Stream Reasoning: A Survey and Further Research Directions.* Proc. Web as a Stream Workshop, FQAS Semantic. Technology Institute (STI) Innsbruck, University of Innsbruck, Austria.

12. **Stuckenschmidt H., Ceri S., Della V. E., Harmelen F.,** (2010) *Towards Expressive Stream Reasoning.* University of Mannheim, Politechnico di Milano, Vrije Universiteit Amsterdam.

13. **Della V. E., Ceri S., Francesco B. D., Braga D., Campi A.,** (2009) *A First Step Towards Stream Reasoning.* Departamento de Electrónica e Informática. Politécnico de Milano, Milano, Italy.

14. **Barbieri D., Braga D., Ceri S., Della V. E., Huang H., Tresp V., Rettinger A., Wermser H.,** (2010) *Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics*. Magazine Social Media Analytics and Intelligence. Polytechnic of Milan. IEEE Intelligent Systems: 32-40.

15. **Brickley D.,** (2004), *Interest Group Chair and Core Working Group co-chair. Resource Description Framework (RDF).* World Wide Web Consortium (W3C), Publicación. Extraído el 20 de enero 2012 de <http://www.w3.org/rdf>.

16. **Francesco B. D., Braga. D., Ceri .S., Della. V. E., Grossniklaus M.,** (2009). *Continuous Queries and Real-time Analysis of Social Semantic Data with C-SPARQL.* Departamento de Electrónica e Informática. Politécnico de Milano. ISWC 2009. BibTeX.

17. **Francesco B. D, Braga D., Ceri S., Della V. E., Grossniklaus M.,** (2009) *C-SPARQL: SPARQL for Continuous Querying*. WWW 2009: 1061-1062 BibTeX.

18. **Dean J., Ghemawat S.,** (2004) *MapReduce: Simplified Data Processing on Large Clusters.* OSDI. Google, Inc.

BIBLIOGRAFÍA CONSULTADA

Anicic D., Fodor P., Rudolph S., Stojanovic N., (2010) *EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning.* FZI Research Center for Information Technology Karlsruhe, Germany.

Barbieri D., Braga D., Ceri S., Della V. E., Huang H., Tresp V., Rettinger A., Wermser H., (2010) *Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics*. Magazine Social Media Analytics and Intelligence. Polytechnic of Milan. IEEE Intelligent Systems: 32-40.

Della V. E., Ceri S., Francesco B. D., Braga D., Campi A., (2009) *A First Step Towards Stream Reasoning.* Departamento de Electrónica e Informática. Politécnico de Milano, Milano, Italy.

Francesco B. D., Braga D., Ceri S., Della V. E., Grossniklaus M., *C-SPARQL: SPARQL for Continuous Querying*. WWW 2009: 1061-1062 BibTeX.

Francesco B. D., Braga D., Ceri S., Della V. E., Grossniklaus M., (2009). *Continuous Queries and Real-time Analysis of Social Semantic Data with C-SPARQL.* Departamento de Electrónica e Informática. Politécnico de Milano.ISWC 2009. BibTeX.

Francesco B. D., Braga D., Ceri S., Della V. E., Grossniklaus M., (2009). *Querying RDF Streams with C-SPARQL.* Departamento de Electrónica e Informática. Politécnico de Milano. ISWC 2009. BibTeX.

Stuckenschmidt H, Ceri S, Della V. E., Harmelen F, (2010) *Towards Expressive Stream Reasoning.* University of Mannheim, Politechnico di Milano, Vrije Universiteit Ámsterdam.

GLOSARIO DE TÉRMINOS

Entre los términos más importantes encontrados a lo largo del trabajo tenemos:

- ✓ **Interoperabilidad:** Habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.
- ✓ **Inferencia:** Evaluación que realiza la mente entre expresiones bien formadas de un lenguaje (EBF), que al ser relacionadas intelectualmente como abstracción, permiten trazar una línea lógica de condición o implicación lógica entre las diferentes EBF. Determinando la verdad o falsedad de algunas de estas expresiones.
- ✓ **Conocimiento Explícito:** Es aquel conocimiento que sabemos que tenemos y somos plenamente conscientes cuando lo ejecutamos. Es más fácil de compartir con los demás ya que se encuentra estructurado y muchas veces esquematizado.
- ✓ **Conocimiento Implícito:** Es aquel conocimiento que sabemos que tenemos, pero no nos damos cuenta que lo estamos utilizando, simplemente lo ejecutamos y ponemos en práctica de una manera habitual.
- ✓ **Recurso:** Cualquier entidad de la cual se pueda dar información.
- ✓ **Framework:** Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

ANEXOS

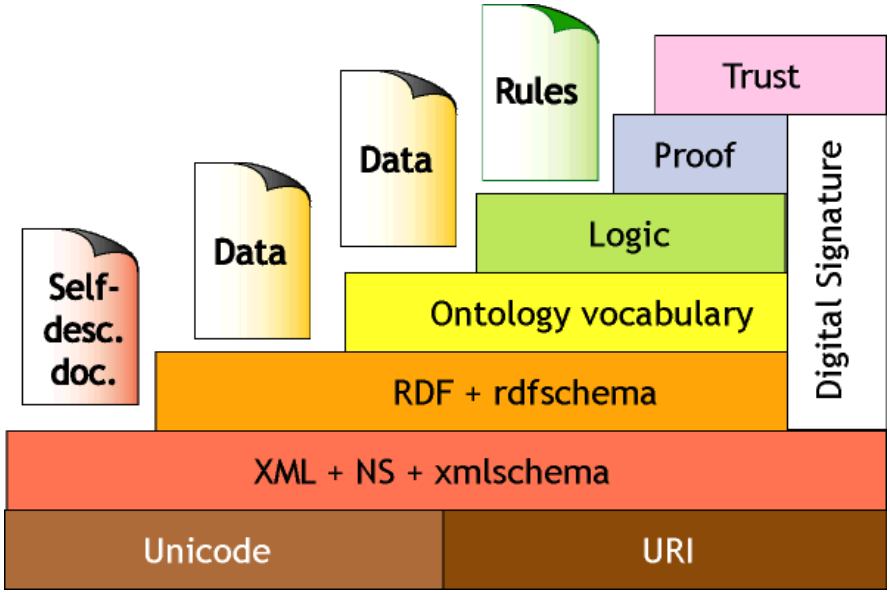


Figura.1. Arquitectura de la web semántica.

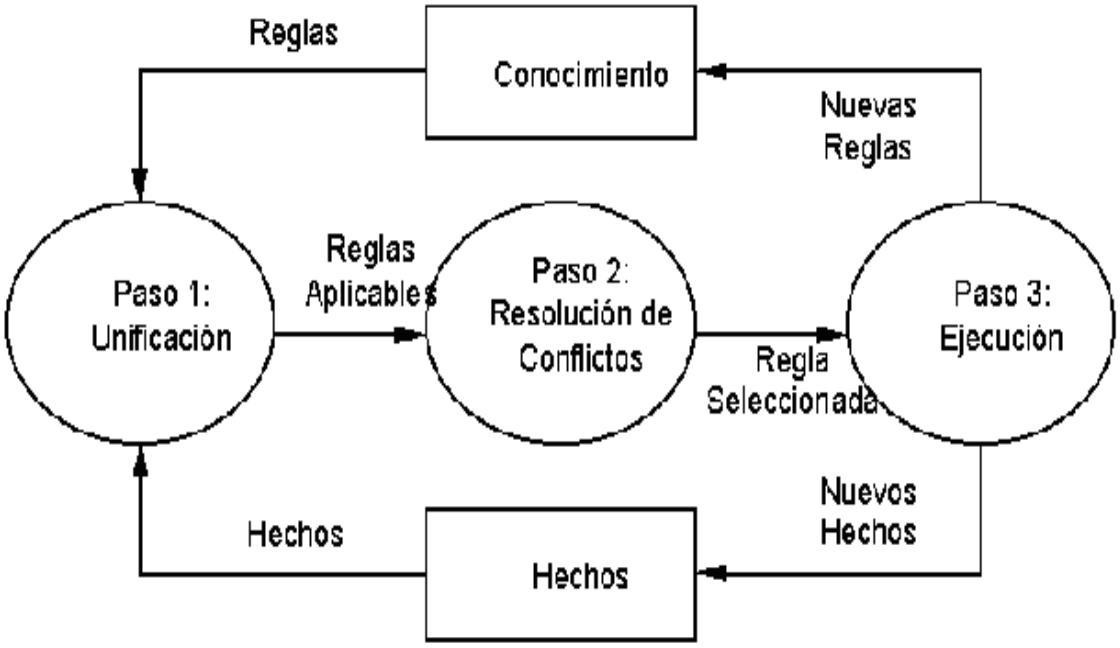


Figura 2. Razonamiento progresivo