Capítulo 1

Sistemas de Transporte Inteligentes basados en Ontologías

Las **Ciudades Inteligentes** (*Smart Cities*) han alcanzado gran popularidad en los últimos años. Con ellas, muchos gobiernos, tanto locales como nacionales se han beneficiado exponencialmente al proveer de manera abierta sus datos al público. Esto trae muchas ventajas no solo en el marco administrativo, sino también en el marco de la investigación, el medio ambiente y el desarrollo tecnológico. Pero, dicho crecimiento exponencial ha provocado la existencia de fuentes de datos muy diferentes entre sí, al menos en la forma de presentarlos, trayendo consigo que a la hora de diseñar sistemas para manipular y/o almacenar dichos datos lo primero que se piense es en cómo transformarlos para tratarlos de manera uniforme y no en lo que los datos brindan de por sí.

En este capítulo, se presentarán las tecnologías, los conceptos y las plataformas involucradas en dicha problemática. Primeramente, se introducirá el concepto de *Internet de las Cosas* (**IoT**) junto con sus beneficios para el desarrollo tecnológico y los principales retos que hay que enfrentar a la hora de trabajar con sensores. Seguidamente, se presentará el panorama de la *Web Semántica* junto con las *Ontologías*, como alternativa para solucionar el problema de la heterogeneidad de los datos obtenidos de sensores. De igual manera, se mencionarán las principales ontologías utilizadas en el ámbito de la **IoT** y se introducirá el paradigma fundamental para acceder a los datos a través de ontologías (**OBDA**). Luego, se presentarán algunas extensiones o variaciones del lenguaje de consulta **SPARQL** para adaptarse a la manipulación de datos en *streaming* para luego dar cabida a algunos proveedores de datos existentes en la actualidad (*Open Data*) y finalmente, se discutirá sobre trabajos relacionados con los sistemas de transporte inteligentes basados en ontologías.

1.1. Internet de las Cosas

En los últimos años, hemos sido testigos de un crecimiento explosivo en la adopción y el despliegue de tecnologías de Internet de las Cosas (**IoT**) en muchos dominios de aplicaciones como en la manufactura, el transporte, la agricultura, etc [67]. El concepto ha crecido rápidamente en importancia, sin embargo, significa cosas diferentes para diferentes personas. De hecho, Whitmore señaló en [66] que no existe una definición universal de **IoT**. Existen dos conceptualizaciones principales: la perspectiva **técnica** y la **sociotecnológica**. La primera, la perspectiva técnica, ve a la **IoT** como un conjunto y un ecosistema de artefactos técnicos. Dentro de esta se pueden encontrar la referida por Weyrich y Ebert quienes ven la **IoT** como *"una manera innovadora de alcanzar una mayor productividad al conectar dispositivos sin problemas"* [65]. En la misma línea, pero más detalladamente se encuentran Tarkoma y Katasonov, quienes definen la **IoT** como *"una red global e infraestructura de servicios de densidad variable y conectividad con capacidades de autoconfiguración basadas en protocolos y formatos estándar e interoperables que consisten en elementos heterogéneos que tienen identidades, atributos físicos y virtuales, y están integrados de manera transparente y segura en Internet"* [60]. La segunda perspectiva, la sociotecnológica, reconoce no solo el rol de los artefactos tecnológicos sino también el asociado con los actores y los procesos que interactúan con la **IoT**. Dentro de esta última se pueden citar algunas definiciones como la ofrecida Haller, quien plantea que la **IoT** es *"un mundo donde los objetos físicos se integran a la perfección en la red de información y donde los objetos físicos pueden convertirse en participantes activos en los procesos comerciales. Los servicios están disponibles para interactuar con estos "objetos inteligentes" a través de Internet, consultar su estado y cualquier información asociada con ellos, teniendo en cuenta cuestiones de seguridad y privacidad"* [30]. Mientras que Shin sostiene que la **IoT** es parte de *"sistemas sociotecnológicos más amplios, que comprenden a los seres humanos, la actividad humana, los espacios, los artefactos, las herramientas y las tecnologías"*. [58]

A pesar de existir cierta divergencia a la hora de formalizar el concepto de **IoT**, Kumar, Tiwari y Zymbler llegan a un concenso lo suficientemente abarcador para los propósitos del presente trabajo: *un paradigma emergente que permite la comunicación entre dispositivos electrónicos y sensores mediante la Internet para facilitar nuestras vidas* [38].

Este paradigma constituye un cambio radical en la calidad de vida de las personas en la sociedad, ofreciendo una gran cantidad de nuevas oportunidades de acceso a los

datos en servicios específicos en la educación, en la seguridad, en la asistencia sanitaria y en el transporte, entre otros campos. Por ejemplo, si los libros, los termostatos, los refrigeradores, las lámparas, las partes automotrices, etc, estuvieran conectados a la Internet y equipados con dispositivos de identificación se pudiera conocer en tiempo real:

→ ¿Cuál dispositivo está apagado y cuál no?

→ ¿Qué artículos están fuera de inventario?

→ ¿Qué medicinas están caducadas?

→ ¿Dónde se forman embotellamientos?

Un sistema **IoT**, como bien se ha dicho, se compone de una gran cantidad de dispositivos y sensores que se comunican unos con otros. Con el crecimiento extensivo y la expansión de redes **IoT**, el número de estos sensores y dispositivos se ha incrementado rápidamente y con ello las transferencias masivas de datos a través de la Internet. Estos datos se generan muy frecuentemente, lo cual trae consigo ciertos retos en el manejo y la recolección de datos, en el almacenamiento, en el procesamiento y en la capacidad de analizarlos al mismo ritmo que son producidos. Entre dichos retos destacan la **interoperabilidad** y la **escalabilidad**, de los cuales se ahondará en las subsecciones siguientes.

1.1.1. Interoperabilidad

La interoperabilidad es la viabilidad de intercambiar información entre diferentes dispositivos y sistemas **IoT** [38]. Este intercambio de información no depende del *software* y del *hardware* implementados. El problema de interoperabilidad surge debido a la naturaleza heterogénea de las diferentes tecnologías y soluciones utilizadas para el desarrollo de **IoT**. La interoperabilidad se centra en garantizar que dos sistemas **IoT** y sus componentes puedan participar en una red de comunicación a pesar de su composición física [61]. El problema de la interoperabilidad de los sistemas de información existe desde 1988 [18]; y posiblemente incluso antes. Hay varias definiciones de interoperabilidad en la literatura. Entre las diversas definiciones de *interoperabilidad*, se citan las relacionadas con contexto de la **IoT**. El Diccionario Oxford da una definición general de interoperabilidad como *"capaz de operar en conjunto"*. Esto implica que dos sistemas interoperables pueden entenderse y utilizar la funcionalidad del otro. **ISO/IEC** define la interoperabilidad como *"la capacidad*

de comunicar, ejecutar programas o transferir datos entre varias unidades funcionales de una manera que requiere que el usuario tenga poco o ningún conocimiento de las características únicas de esas unidades" [33]. En una visión más amplia, **IEEE** define la interoperabilidad como *"la capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información que se ha intercambiado"* [53]. Según esta definición, la interoperabilidad se realiza mediante la elaboración de estándares. En **IoT**, la interoperabilidad puede definirse como la capacidad de dos sistemas para comunicarse y compartir servicios entre sí. [37]

Considerando la interoperabilidad como un problema que merece especial atención, los investigadores han sugerido diversas alternativas para lidiar con dichos problemas de interoperabilidad. Estas alternativas, aunque alivian algo de presión sobre los sistemas **IoT**, todavía no son ni concluyentes ni mucho menos absolutas. La mayoría de ellas se pueden agrupar mediante el enfoque en que se basan en:

→ Redes virtuales

→ Arquitectura orientada a servicios

→ Superposición

→ Adaptadores

Los modelos semánticos son una de las soluciones más utilizadas para superar la heterogeneidad de las fuentes y formatos de los datos de **IoT**. Algunos de estos modelos serán presentados en secciones posteriores.

1.1.2. Escalabilidad

La escalabilidad es un aspecto crucial que surge con los nuevos y progresivos avances tecnológicos que ocurren todos los días. Un sistema es escalable si es posible añadir nuevos servicios, equipos y dispositivos sin degradar su rendimiento. La importancia de la escalabilidad yace en ayudar al sistema a funcionar correctamente sin demoras indebidas y sin consumir recursos improductivamente. En un sistema escalable, los requisitos de memoria se mantienen en niveles controlados a medida que aumenta la cantidad de datos [54]. Por lo tanto, es importante hacer que un sistema sea escalable para que sea más eficiente para el uso presente y futuro [8]. El problema principal con los sistemas **IoT** es el de soportar un gran número de dispositivos con diferente memoria, capacidad de procesamiento, poder de almacenamiento y ancho de banda [52]. Un gran ejemplo de escalabilidad son los sistemas **IoT** basados en la nube, que

proveen suficiente flexibilidad para escalar la red **IoT** añadiendo nuevos dispositivos, almacenamiento y poder de procesamiento bajo demanda; pero dicha escalabilidad trae consigo un mayor costo a medida que se van incrementando los dispositivos de la red y las necesidades de los mismos. Es por ello que la eficiencia en la obtención y procesamiento de los datos juega un papel crucial.

En el contexto de la **IoT** se tienden a definir diferentes tipos de escalabilidad: [29]

→ **Escalabilidad vertical**: También se conoce como *ampliación*, que es la capacidad de mejorar el hardware o el software existente al agregarle más recursos. Por ejemplo, al agregar potencia de procesamiento a un servidor para aumentar su velocidad. Además, se puede escalar un sistema verticalmente agregando más poder de procesamiento, memoria principal, almacenamiento o interfaces de red al nodo en el cual está contenido para satisfacer más solicitudes. La principal ventaja de la escalabilidad vertical es que consume menos energía si lo comparamos con la ejecución de varios servidores y reduce los esfuerzos administrativos ya que se necesita manejar y administrar un solo sistema. Además, la implementación del sistema se mantiene simple, reduciendo los costos de *software* y se conserva la compatibilidad de la aplicación. Si bien existen ventajas, también hay desventajas de este tipo de escalado que incluyen un mayor riesgo de falla de *hardware* haciendo que las interrupciones tengan mayores proporciones y un bloqueo severo del proveedor.

→ **Escalabilidad horizontal**: Es la capacidad de aumentar la capacidad de entidades de *hardware* o *software* para que puedan trabajar juntas como una sola unidad. La escalabilidad horizontal se puede lograr agregando más máquinas al grupo de recursos y agregando más nodos a un sistema, por ejemplo, agregando una nueva computadora a una aplicación de *software* distribuida. Los ejemplos de esto pueden ser los sistemas **SOA**[1] y los servidores web que aumentan su disponibilidad al agregar más y más servidores a la red, equilibrando la carga de solicitudes para que se puedan distribuir entre todos ellos. Los arquitectos de sistemas pueden configurar un enjambre de pequeñas computadoras en un grupo para obtener una potencia informática acumulativa que muchas veces excede la de las computadoras basadas en un solo procesador tradicional. La escalabilidad de la aplicación indica el rendimiento mejorado de la ejecución de aplicaciones en una versión ampliada del sistema.

[1] Arquitectura Orientada a Servicios

Si bien los métodos semánticos constituyen una alternativa para solucionar el problema de la interoperabilidad en los sistemas **IoT**, por otro lado traen ciertos retos que vale la pena mencionar. La esencia de los modelos semánticos yace en agregar significado a los datos, como pueden ser: el contexto y el momento en el tiempo en el que fueron obtenidos, en qué unidad de medida se encuentran, etc. Esta lista de agregados puede ser tan larga como el nivel de granularidad que se requiera por las aplicaciones cliente. Es precisamente esta característica uno de los prinicipales aspectos que atenta contra la escalabilidad de los sistemas **IoT** al utilizar modelos semánticos: el tiempo para agregar significado a los datos (incluir anotaciones a los mismos) y luego consultarlos (mediante un modelo de lenguaje de consultas al que se hará referencia posteriormente) se vuelve un cuello de botella en el procesamiento en tiempo (*cuasi*) real de los datos provenientes de entornos **IoT**. Es por ello que para sobrepasar dicho obstáculo, se deben utilizar *modelos ligeros*[2] en cuanto al número de conceptos y relaciones entre conceptos que describen los datos se refiere.

1.2. La Web Semántica

Una vez establecido que agregar significado a los datos constituye una vía para solucionar algunos de los retos que presenta el trabajo con *streams* y datos provenientes de sistemas **IoT**, se introducirá el término **Web Semántica** formalmente para luego dar cabida al concepto de **Ontología**.

La Web Semántica es la web de los datos, de fechas, títulos, números de pieza, propiedades químicas y cualquier otro dato que se pueda imaginar. [64] Día a día son muchos los datos que se utilizan y no forman parte de la web. Mediante un simple acceso a la web, cualquier individuo puede acceder a sus operaciones bancarias, a sus citas en el calendario o incluso a las fotos que tomó desde su teléfono. Pero, ¿pueden verse dichas fotos en el calendario para ver qué estaba haciendo el día que fueron tomadas o, ver todas las operaciones bancarias en un calendario? Simplemente no, esto se debe a que no se tiene una web de datos, ya que los datos están controlados por aplicaciones, y cada aplicación los provee a su forma. La Web Semántica se trata de dos cosas: se trata de formatos comunes para la integración y combinación de datos extraídos de diversas fuentes, mientras que la web original se concentra principalmente en el intercambio de documentos y además se trata del lenguaje para registrar cómo se relacionan los datos con los objetos del mundo real. Esta disposición permite que

[2]Un modelo se considera ligero si describe solamente los datos indispensables para resolver el problema para el cual fue planteado

cualquier persona o máquina, comience en una base de datos y luego se mueva a través de un conjunto interminable de bases de datos que no están conectadas físicamente, sino por el tema que tratan. [40]

La idea central de la Web Semántica consiste en añadir *metadatos*[3] semánticos y ontológicos. Estas informaciones adicionales, se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento; lo cual conduce a experiencias de cliente más inteligentes y sin esfuerzo al brindar al contenido la capacidad de comprenderse y presentarse en las formas más útiles de acuerdo a las necesidades del mismo. [42]

El precursor de esta idea fue *Tim-Berners-Lee* [4], quien intentó desde el principio incluir información semántica en su creación, la **World Wide Web**, pero por diferentes causas no fue posible. [13]

La Web Semántica no es una web separada, sino una extensión de la actual, en la que la información se presenta con un significado bien definido. Al igual que la Internet, la Web Semántica se presenta de manera descentralizada.

1.2.1. Componentes principales de la Web Semántica

A día de hoy se puede decir que a la Web Semántica, a pesar de ser un camino bastante prometedor para el futuro de la web, le quedan componentes por estandarizar. Una vez dicho esto, sí se pueden destacar aquellos que ya lo están y que a día de hoy son muy utilizados. En la figura 1.1 se pueden observar aquellas tecnologías que son estándares para la Web Semántica y que permiten que la misma sea posible. Las tecnologías que se encuentran desde el fondo del cubo mostrado en la figura 1.1 hasta el bloque etiquetado con el nombre OWL están actualmente estandarizadas y tienen gran aceptación a la hora de construir aplicaciones para la Web Semántica.

- → El bloque etiquetado con el nombre PLATAFORMA WEB se encuentra en el fondo del cubo para hacer incapié en la idea de que la Web Semántica es una extensión de la web actual y por tanto, hereda las tecnologías y estándares definidos para la misma. Entre ellas se destacan:
  - ⇀ Los **IRI**s (*Internationalized Resource Identifier*), como generalización de los **URI**s (*Uniform Resource Identifier*) que proveen una manera de identificiar unívocamente los recursos de la Web Semántica. Los **IRI** amplían

[3] Los metadatos son datos sobre los datos

[4] Científico de la computación británico conocido por ser el padre de la **World Wide Web**

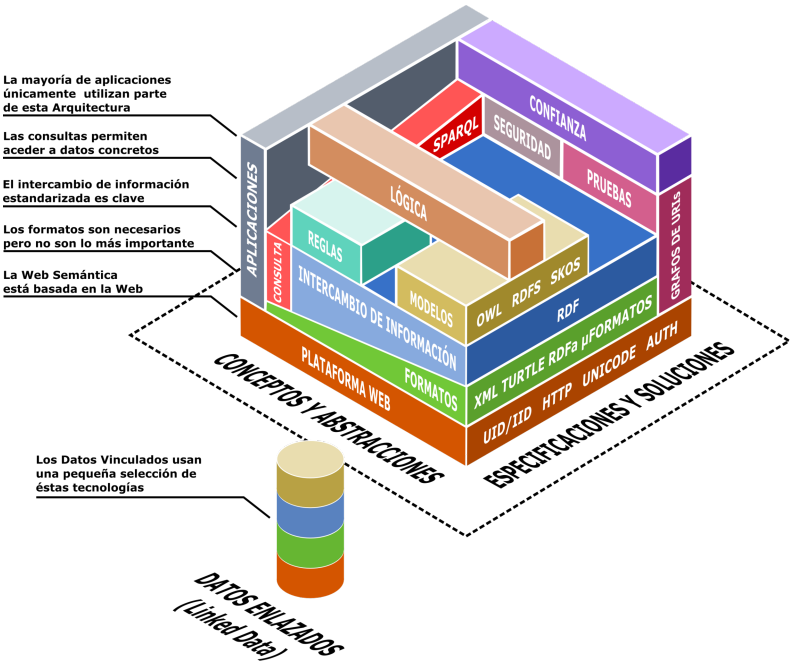


Figura 1.1: Componentes principales de la Web Semántica (basado en el esquema propuesto por *Tim Berners-Lee* en [6])

los **URI** mediante el uso del Conjunto de Caracteres Universal (**Unicode**), ya que los **URI** se limitaban a **ASCII** , con muchos menos caracteres. [24] Esta identificación unívoca se hace necesaria para permitir la manipulación demostrable con recursos en los bloques superiores.

- *Hypertext Transfer Protocol* (**HTTP**): Es un protocolo de la *capa de aplicación* (séptimo nivel del modelo **OSI**) para la transmisión de documentos hipermedia, como **HTML**. Fue diseñado para la comunicación entre los navegadores y servidores web. Sigue el clásico modelo *cliente-servidor*[5], en el que se establece una conexión, realizando una petición a un servidor y se espera una respuesta del mismo. Se trata de un *protocolo sin estado*[6], lo que significa que el servidor no guarda ningún dato (estado) entre dos peticiones. Aunque en la mayoría de casos se basa en una conexión del tipo **TCP/IP**, puede ser usado sobre cualquier *capa de transporte* (cuarto nivel del modelo **OSI**) segura o de confianza, es decir, sobre cualquier protocolo que no pierda mensajes silenciosamente, tal como **UDP.** [49]
- **Unicode**: Es un conjunto de caracteres estándar que numera y define ca-

[5]Es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes

[6]Es un protocolo de comunicaciones que trata cada petición como una transacción independiente que no tiene relación con cualquier solicitud anterior

racteres de diferentes idiomas, sistemas de escritura y símbolos. Al asignar un número a cada caracter, los programadores pueden crear codificaciones de caracteres, para permitir que las computadoras almacenen, procesen y transmitan cualquier combinación de idiomas en el mismo archivo o programa. [50] Este estándar**,** permite representar y manipular texto en varios lenguajes. Debido a que la Web Semántica está enfocada en el significado de las cosas y no en cómo las cosas son representadas, la representación multilingüe se hace necesaria.

- ⇀ **Auth:** Se refiere al conjunto de protocolos de autenticación diseñados específicamente para la transferencia de datos de autenticación entre dos entidades. Estos protocolos permiten a la entidad receptora autenticar la entidad de conexión así como autenticarse a sí mismo ante la entidad de conexión declarando el tipo de información necesaria para la autenticación, así como la sintaxis. [17] Entre ellos se puede mencionar: **LDAP** (*Ligthweight Directory Access Protocol*), **Kerberos**, **Oauth 2**, **SAML** y **Radio**.

→ Los bloques medios, etiquetados con los nombres FORMATOS, INTERCAMBIO DE INFORMACIÓN, MODELOS, CONSULTA y REGLAS contienen las tecnologías estandarizadas por la **W3C**[7] para permitir la construcción de aplicaciones sobre la Web Semántica. Algunas de ellas son:

- ⇀ El lenguaje de marcado **XML** para permitir la creación de documentos compuestos por datos semiestructurados. La Web Semántica otorga significado (como se mencionó anteriormente) a datos semiestructurados.
- ⇀ *Terse RDF Triple Language* (**Turtle**)**:** Es un formato popular para serializar **RDF** en una forma textual compacta**.** [4] Representa la información en *tripletas*, cada una de las cuales consta de un sujeto, un predicado y un objeto.
- ⇀ *Resource Description Framework in Attributes* (**RDFa**): Es un conjunto de extensiones de **XHTML** propuestas por **W3C** para introducir semántica en los documentos. **RDFa** aprovecha atributos de los elementos *meta* y *link* de **XHTML** y los generaliza de forma que puedan ser utilizados en otros elementos. Este formato destaca por utilizar **URI**s compactos

[7] La **W3C** (*World Wide Web Consortium*) es la principal organización de estándares internacionales para la **World Wide Web**

(o **CURIE**s) para describir *sujetos*, *predicados* y *objetos*; reduciendo la cantidad de lenguaje de marcado utilizado. [57]

- **Microformatos**: Son conjuntos de formatos de datos abiertos y simples, desarrollados sobre estándares ya existentes, ampliamente adoptados, incluyendo **XHTML** (**HTML** y **XML**) y **CSS**. [14] Entre los más destacados se encuentran *hCard*, para información de contacto incluyendo direcciones postales y coordenadas geográficas; y *hCalendar* para eventos.
- **SKOS:** Es un modelo de datos común para compartir y vincular sistemas de organización del conocimiento a través de la Web. El modelo de datos **SKOS** proporciona una ruta de migración estándar y de bajo costo para trasladar los sistemas de organización del conocimiento existentes a la Web Semántica. **SKOS** también proporciona un lenguaje ligero e intuitivo para desarrollar y compartir nuevos sistemas de organización de conocimientos. Puede usarse solo o en combinación con lenguajes formales de representación del conocimiento, como el lenguaje **OWL**. [**26**]
- El **RDF** (*Resource Description Framework*), como marco de trabajo para representar y crear sentencias en la forma de *ternas*[8]. Esto da la posibilidad de representar la información sobre recursos en forma de grafos, permitiendo que todos los algoritmos diseñados para trabajar sobre una topología de grafo sean aplicables a la Web Semántica.
- El **Esquema RDF** (**RDFS** o **RDF Schema**) es una extensión semántica de **RDF.** Es un lenguaje primitivo de ontologías que proporciona los elementos básicos para la descripción de vocabularios. [11]
- El lenguaje de ontologías **OWL** (*Web Ontology Language*) extiende el **RDFS** añadiendo construcciones más avanzadas para describir las semánticas de las declaraciones en **RDF**. **OWL** es un lenguaje de la Web Semántica diseñado para representar un conocimiento rico y complejo sobre cosas, grupos de cosas y relaciones entre cosas. Está basado en la lógica computacional, de modo que el conocimiento expresado en **OWL** puede ser explotado por programas de computadora, por ejemplo, para verificar la consistencia de ese conocimiento o para hacer explícito el conocimiento implícito. Los documentos de **OWL**, conocidos como *ontologías*, se pueden publicar en la World Wide Web y pueden hacer referencia a otras ontologías de **OWL** o desde ellas [25]. **OWL** permite añadir restricciones

[8]Una terna es la entidad atómica en el modelo de datos **RDF**

adicionales, como pueden ser la cardinalidad, las restricciones sobre valores admisibles y propiedades de relaciones como la transitividad. Este lenguaje trae el poder del razonamiento y la inferencia lógica a la Web Semántica. Aunque **OWL** ha tenido éxito, se han identificado ciertas limitaciones importantes en su diseño. Una de ellas es la falta de un conjunto adecuado de datos primitivos. Es por ello que la **W3C** decide introducir **OWL 2.0** en el que se agregan varias características nuevas a **OWL 1**, incluido un mayor poder expresivo para las propiedades, soporte extendido para tipos de datos, capacidades de metamodelado simples, capacidades de anotación extendidas y claves. **OWL 2** también define varios *perfiles*: subconjuntos de idiomas de **OWL 2** que pueden cumplir mejor con ciertos requisitos de rendimiento o pueden ser más fáciles de implementar [22]. Estos *perfiles*, son versiones reducidas de **OWL 2** que corresponden a un modelo de uso específico e intercambian algo de poder expresivo por la eficiencia del razonamiento; entre ellos se pueden encontrar: [48]

- **OWL 2 EL:** Es particularmente útil en aplicaciones que emplean ontologías que contienen un gran número de propiedades y/o clases. Este perfil captura el poder expresivo utilizado por muchas de estas ontologías y es un subconjunto de **OWL 2** para el cual los problemas básicos de razonamiento se pueden realizar en un tiempo polinomial con respecto al tamaño de la ontología.
- **OWL 2 QL:** Está dirigido a aplicaciones que utilizan grandes volúmenes de datos de instancia y donde la respuesta a consultas es la tarea de razonamiento más importante. En **OWL 2 QL**, la respuesta a consultas conjuntivas se puede implementar utilizando sistemas de bases de datos relacionales convencionales. Utilizando una técnica de razonamiento adecuada, se puede realizar en $LOGSPACE$ una respuesta sólida y completa a las consultas conjuntivas con respecto al tamaño de los datos (aserciones).
- **OWL 2 RL:** Está dirigido a aplicaciones que requieren un razonamiento escalable sin sacrificar demasiado poder expresivo. Está diseñado para adaptarse a aplicaciones **OWL 2** que pueden cambiar la expresividad total del lenguaje por eficiencia, así como aplicaciones **RDF** (**S**) que necesitan algo de expresividad adicional. Los sistemas de razonamiento **OWL 2 RL** se pueden implementar utilizando motores de razonamiento basados en reglas.

- ⇀ El lenguaje de consulta **SPARQL** : un lenguaje estandarizado para realizar consultas sobre grafos **RDF**. Este lenguaje de consulta se hace imprescindible para recuperar información de aplicaciones sobre la Web Semántica. **SPARQL** se puede utilizar para expresar consultas en diversas fuentes de datos, ya sea que los datos se almacenen de forma nativa como **RDF** o se vean como **RDF** a través de un *middleware*. **SPARQL** contiene capacidades para consultar patrones de gráficos obligatorios y opcionales junto con sus conjunciones y disyunciones. También admite pruebas de valor extensibles y consultas restringidas por grafos **RDF** de origen. Los resultados de las consultas **SPARQL** pueden ser conjuntos de resultados o grafos **RDF.** [**63**]
- ⇀ El formato de intercambio de reglas **RIF.** Su existencia en el universo de la Web Semántica se debe a la necesidad de describir relaciones que no pueden ser directamente descritas utilizando la lógica descriptiva de **OWL.** Este formato fue diseñado por la **W3C** para establecer un estándar para el intercambio de reglas entre sistemas de reglas, en particular entre motores de reglas web. **RIF** se centró en el intercambio en lugar de tratar de desarrollar un único lenguaje de reglas de carácter general, a diferencia de otros estándares de la Web Semántica, como **RDF, OWL** y **SPARQL**. [36]. Además de **RIF** cabe destacar el lenguaje de reglas **SWRL** (*Semantic Web Rule Language*) que permite expresar tanto reglas como cierta lógica, combinando **OWL DL** con un subconjunto de **Datalog**[9] **[31].** Las reglas en **SWRL** tienen la forma de una implicación entre un antecedente, llamado *cuerpo* y una consecuencia, llamada *cabeza*. La mayoría de los razonadores no soportan la especificación completa de **SWRL** debido a que el razonamiento se vuelve indecidible. Ante esta situación, se toman ciertas estrategias como: traducir **SWRL** a lógica de primer orden o expandir un razonador **OWL-DL** existente basado en el algoritmo *tableaux.*[10].

→ En los bloques superiores se encuentran las tecnologías que todavía o bien no se han estandarizado o solo existen ideas de cómo deberían ser implementados para complementar la Web Semántica. Aquí se pueden encontrar:

[9]Un lenguaje de programación de lógica declarativa que sintácticamente es un subconjunto de **Prolog**.

[10]Consiste en examinar todas las posibilidades que podrían hacer falsa una proposición dada y buscar si una de estas posibilidades es lógicamente viable.

- ⇀ La criptografía (representada dentro del bloque etiquetado con el nombre SEGURIDAD) en la Web Semántica se hace necesaria para proteger y verificar que las sentencias de la misma provienen de fuentes confiables. Esta idea se puede materializar agregando firmas digitales a las sentencias **RDF**.
- ⇀ La veracidad de las sentencias derivadas (representada dentro del bloque etiquedado con el nombre CONFIANZA) estará respaldada por alguna de las siguientes variantes (no necesariamente excluyentes):
  - → La verificación de la fuente
  - → Por la lógica formal mediante la cual pudo haber sido inferida.
- ⇀ La capa de presentación que permitirá a los seres humanos utilizar las aplicaciones sobre la Web Semántica.

→ El cilindro etiquetado con el nombre DATOS ENLAZADOS, cercano a el cubo que describe a la Web Semántica en la figura 1.1, se refiere a un conjunto de prácticas recomendadas por TIM BERNERS-LEE para publicar datos estructurados en la Web. Esto se debe a que la Web Semántica no se trata solo de publicar datos en la web, sino de conectar dichos datos para que puedan ser descubiertos posteriormente por máquinas o personas. Entre dichas prácticas se pueden enumerar: [5]

1. Utilizar **URI**s como nombres para las cosas
2. Utilizar **HTTP URI**s para que las personas puedan buscar dichos nombres
3. La información identificada por un **URI** determinado debe ser útil y seguir los estándares (***RDF****, **SPARQL**)
4. Incluir enlaces a otros **URI**s para que nuevas cosas puedan ser descubiertas

1.3. Ontologías

En la sección 1.2 se introdujo informalmente el término *ontología* sin dar una definición precisa de su significado, pues solo se pretendía dar una ubicación del concepto en el panorama de la Web Semántica y no el concepto en sí. En esta sección se abordará dicho concepto con mayor profundidad.

Un problema presente en muchas aplicaciones informáticas de la actualidad es la discrepancia existente al nombrar términos del dominio que modelan dichas aplicaciones; trayendo consigo que dos bases de datos distintas puedan usar identificadores completamente distintos para identificar el mismo concepto, como puede ser el nombre o los apellidos de una persona. Un programa que requiera comparar alguno de estos conceptos necesitaría conocer que esos dos términos se utilizan para representar la misma cosa. Para satisfacer este requerimiento, las aplicaciones necesitarían tener un método capaz de descubrir dichos significados comunes para cualquier base de datos que se consulte. Este método de descubrimiento se hace disponible a través de lo que se conoce por **ontologías**. Una ontología en este sentido, se refiere a un documento o archivo que define formalmente las relaciones entre términos. [7] Una ontología típica está compuesta por:

→ Una taxonomía que define todas las clases de objetos y las relaciones que se establecen entre ellos; por ejemplo: una dirección puede ser definida como un tipo de localización.

→ Un conjunto de reglas de inferencia que permite a las aplicaciones tomar decisiones basadas en las clases proporcionadas sin la necesidad de entender la información provista. Por ejemplo, una ontología puede expresar la regla siguiente: *si un código de ciudad está asociado con un código de estado, y una dirección usa ese código de ciudad, entonces ese código de dirección tiene el código de estado asociado.* Una aplicación que utiliza estos datos puede inferir que, si un código de ciudad en particular es proporcionado, esa dirección debe estar en una provincia o estado en particular.

En resumen, todo lo que las ontologías le permiten hacer a una aplicación es manipular la información proporcionada de acuerdo a reglas predeterminadas e inferir conclusiones lógicas sobre los datos en el formato que se requiere. De igual manera las ontologías pueden ser utilizadas para realizar una variedad de funciones fuera de deducciones simples. Dado que se presenta más información sobre un concepto, pueden actuar para mejorar la precisión de las solicitudes de un motor de búsqueda y permitir que las aplicaciones realicen una amplia variedad de tareas de forma autónoma, así como abordar preguntas complicadas que los motores de búsqueda actuales no pueden responder correctamente.

1.3.1. Tipos de Ontologías

En la literatura se encuentran diversas clasificaciones de ontologías de acuerdo a distintos enfoques y a distintos autores. A continuación se exponen dichos enfoques:

- → **De acuerdo al nivel de generalidad**
  - ⇀ *Clasificación según Guarino* [28]
    - ⇁ **Ontologías de Alto Nivel**: Describen conceptos generales como espacio, tiempo, materia, objeto. Son independientes de un dominio o problema particular. Su intención es unificar criterios entre grandes comunidades de usuarios.
    - ⇁ **Ontologías de Dominio**: Describen el vocabulario relacionado a un dominio genérico (por ejemplo medicina o automotores), por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
    - ⇁ **Ontologías de Tareas**: Describen el vocabulario relacionado a una tarea o actividad genérica (por ejemplo de diagnóstico o de ventas), por medio de la especialización de los conceptos introducidos en las ontologías de alto nivel.
    - ⇁ **Ontologías de Aplicación**: Describen conceptos que pertenecen a la vez a un dominio y a una tarea particular, por medio de la especialización de los conceptos de las ontologías de dominio y de tareas. Generalmente corresponden a roles que juegan las entidades del dominio cuando ejecutan una actividad
  - ⇀ *Clasificación según Fensel* [20]
    - ⇁ **Ontologías Genéricas o de Sentido Común**: Capturan conocimiento general acerca del mundo. Proveen nociones básicas y conceptos para cosas tales como espacio, tiempo, estado, eventos. Son válidas en varios dominios.
    - ⇁ **Ontologías Representacionales**: No se comprometen con ningún dominio en particular. Proveen entidades sin establecer que deberían representar, por lo tanto definen conceptos para expresar conocimiento de manera orientada a objetos o a marcos de trabajo.
    - ⇁ **Ontologías de Dominio**: Capturan el conocimiento válido para un tipo particular de dominio (ej: electrónica, medicina, mecánica).

- ⇀ **Ontologías de Métodos y Ontologías de Tareas**: Las primeras proveen términos específicos para métodos particulares de resolución de problemas, mientras que las segundas proveen términos para tareas específicas. Ambas proveen un punto de vista de razonamiento sobre conocimiento del dominio

→ **De acuerdo al tipo de estructura de conceptualización** [62]

- ⇁ **Ontologías terminológicas**: Especifican términos a utilizarse para representar el conocimiento en el dominio de estudio. Intentan obtener un lenguaje unificado sobre un tema específico, por ejemplo, el Sistema de Lenguaje Médico Unificado (**ULMS**)
- ⇁ **Ontologías de información**: Especifican la estructura de los registros de una base de datos, determinando un marco para el almacenamiento estandarizado de información. Un ejemplo es un marco de trabajo para modelar los registros médicos de pacientes.
- ⇁ **Ontologías de representación de conocimiento**: Especifican conceptualizaciones del conocimiento. Comparados con las ontologías de información, estas tienen una estructura interna más rica. Suelen estar enfocadas a un uso particular del conocimiento que describen.

→ **De acuerdo a los aspectos del mundo real que intentan modelar** [35]

- ⇁ **Ontologías Estáticas**: Describen las cosas que existen, sus atributos y las relaciones entre ellos. Esta clasificación asume que el mundo está poblado de entidades que están dotadas de una identidad única e inmutable. Términos que utilizan: entidades, atributos, relaciones
- ⇁ **Ontologías Dinámicas**: Describen los aspectos que pueden cambiar en el mundo que modelan. Para modelarlas se pueden utilizar máquinas de estados finitos, redes de Petri, etc. Términos que utilizan: procesos, estados, transición de estados.
- ⇁ **Ontologías Intencionales**: Describen aspectos que tienen que se refieren al mundo de las motiva-ciones, intenciones, metas, creencias, alternativas y elecciones de los agentes involucrados. Términos que utilizan: aspecto, objetivo, soporte, agente .
- ⇁ **Ontologías Sociales**: Describen aspectos que se relacionan con lo social,

estructuras organizacionales, redes, interdependencias. Términos que utilizan: actor, posición, rol, autoridad, compromiso.

→ **De acuerdo al sujeto de conceptualización:**

- ⇀ **Ontologías de representación de conocimiento**: Capturan primitivas de representación utilizadas para formalizar conocimiento bajo un paradigma de representación de conocimiento dado.
- ⇀ **Ontologías comunes o generales**: Representan conocimiento de sentido común y reutilizable en distintos dominios, por ejemplo sobre vocabulario relacionado a cosas, eventos, tiempo, espacio, etc.
- ⇀ **Ontologías de alto nivel**: Son ontologías que describen conceptos y nociones generales bajo las cuales pueden enlazarse los términos raíces de todas las ontologías. Un problema que existe es que hay varias de estas ontologías de alto nivel que difieren en los criterios para clasificar la mayoría de los conceptos generales de la taxonomía.
- ⇀ **Ontologías de dominio**: Son aquellas ontologías reutilizables en un dominio particular (medicina, ingeniería, etc.). Proveen vocabularios sobre conceptos dentro del dominio y sus relaciones.
- ⇀ **Ontologías de tareas**: Describen vocabulario relacionado a actividades genéricas. Proveen un vocabulario sistemático de términos utilizados para resolver problemas que pueden o no pertenecer a un mismo dominio.
- ⇀ **Ontologías de tareas de dominios**: A diferencia de las ontologías de tareas, estas ontologías son reutilizables en un dominio dado, y no entre dominios diferentes.
- ⇀ **Ontologías de métodos**: Proveen definiciones de conceptos relevantes y relaciones aplicables a un proceso de razonamiento específico a fin de cumplir una tarea particular.
- ⇀ **Ontologías de aplicaciones**: Son dependientes de las aplicaciones. A menudo extienden y especializan vocabulario de una ontología de dominio o de tareas para una aplicación particular.

1.4. Principales ontologías utilizadas en IoT

La **IoT** ha introducido cambios radicales en la forma en que los datos son procesados. La cantidad de datos, la velocidad de los cambios en los mismos, la variedad

de fuentes y formatos de los dispositivos **IoT** implican nuevos retos para procesar e interoperar entre fuentes y formatos de datos heterogéneos. Especialmente en aplicaciones a gran escala sobre **IoT**, nuevas soluciones están siendo propuestas para poder lidiar con el gran volumen de datos que los dispositivos de estas aplicaciones generan [19]. La anotación con datos semánticos ha sido una de las variantes utilizadas para enfrentar los problemas antes mencionados. Para ello varios investigadores, en los últimos años, han concebido ontologías ligeras (mínimo número de conceptos y relaciones entre conceptos) para dar significado a los datos generados por los distintos dispositivos **IoT**. En las siguientes subsecciones se introducen algunas de estas ontologías.

1.4.1. Ontología *Time*

La información temporal es importante en la mayoría de las aplicaciones del mundo real. Por ejemplo, en un pedido en línea siempre debe especificarse la fecha en la cual se hizo dicho pedido. De igual manera, el proceso de renta de autos debe realizarse durante un período concreto de tiempo. Los eventos en el mundo ocurren en momentos específicos y generalmente tienen una duración finita. Las transacciones ocurren en secuencia, y el estado actual de un sistema depende del historial exacto de todas las transacciones hechas en dicho sistema. El conocimiento de las relaciones temporales entre transacciones, eventos, viajes y pedidos es a menudo fundamental. **OWL-Time** se ha desarrollado en respuesta a esta necesidad, para describir las propiedades temporales de cualquier recurso indicado, mediante un identificador web (**URI**), incluidas páginas web y elementos del mundo real si se desea. Se centra particularmente en las relaciones de ordenamiento temporal. Si bien estos están implícitos en todas las relaciones temporales, **OWL-Time** proporciona predicados específicos para respaldar o hacer explícitos los resultados del razonamiento sobre el orden o la secuencia de entidades temporales.

OWL-Time proporciona representaciones en las que los elementos de una fecha y hora se colocan en recursos direccionables por separado, lo que puede ayudar con las consultas y las aplicaciones de razonamiento. **OWL-Time** también admite otras representaciones de la posición temporal y la duración, incluidas las coordenadas temporales (posición escalada en un eje temporal continuo) y los tiempos ordinales (posiciones o períodos con nombre), así como relajar la expectativa de la versión original de que las fechas deben usar el calendario gregoriano. Sin embargo, **OWL-Time** tiene un enfoque particular en ordenar las relaciones (*topología temporal*) que

no se admite explícitamente en ninguna de las codificaciones de fecha y hora.

Esta ontología contiene conceptos temporales para describir propiedades temporales de objetos del mundo real o aquellos descritos en páginas web. La ontología proporciona un vocabulario para expresar hechos sobre relaciones topológicas (de ordenación) entre instantes e intervalos, junto con información sobre duraciones y sobre la posición temporal, incluida la información de fecha y hora. Las posiciones y duraciones de tiempo se pueden expresar usando el calendario y reloj convencional (gregoriano), o usando otro sistema de referencia temporal como el tiempo **Unix,** el tiempo geológico u otros calendarios. [41]

1.4.2. Ontología *SSN*

Los sensores son una fuente importante de datos disponibles en la web en la actualidad. Si bien los datos de los sensores pueden publicarse como meros valores, buscar, reutilizar, integrar e interpretar estos datos requiere más que solo los resultados de la observación. De igual importancia para la interpretación adecuada de estos valores son la información sobre la característica de interés estudiada, la propiedad observada y la estrategia de muestreo utilizada. Los estándares de habilitación web de sensores de la **Open Geospatial Consortium** (**OGC**) proporcionan un medio para anotar sensores y sus observaciones. Sin embargo, estos estándares no están integrados ni alineados con las tecnologías de Web Semántica del **W3C** y los Datos Vinculados (**Linked Data**) en particular, que son impulsores claves para crear y mantener un gráfico de datos global y densamente interconectado. Con el surgimiento de la **Web de las Cosas (WoT)**[11] y las ciudades y hogares inteligentes en general, los actuadores y los datos que ellos producen también se convierten en ciudadanos de primera clase de la web. Dada su estrecha relación con los sensores, las observaciones, los procedimientos y las características de interés, es deseable proporcionar una ontología común que también incluya actuadores y actuación. Por último, con la creciente diversidad de datos y proveedores de datos, las definiciones como las de sensores deben ampliarse, por ejemplo, para incluir la detección social.

Esta ontología describe los sensores y sus observaciones, los procedimientos involucrados, las características de interés estudiadas, las muestras utilizadas para hacerlo y las propiedades observadas, así como los actuadores. **SSN** sigue una arquitectura de modularización horizontal y vertical al incluir una ontología central liviana pero autónoma llamada **SOSA** (**Sensor, Observation, Sample y Actuator**), la cual se

[11]La Web de las Cosas describe un conjunto de estándares establecidos por el **W3C** para resolver los problemas interoperabilidad de diferentes plataformas y dominios de aplicaciones de **IoT**

introducirá en la siguiente subsección, para sus clases y propiedades elementales. Con su diferente alcance y diferentes grados de axiomatización, **SSN** y **SOSA** pueden soportar una amplia gama de aplicaciones y casos de uso, que incluyen imágenes satelitales, monitoreo científico a gran escala, infraestructuras industriales y domésticas y la **Web of Things** entre otras. [46]

1.4.3. Ontología *SOSA*

En su definición más amplia, los sensores detectan y reaccionan a cambios en el entorno que revelan, directa o indirectamente, el valor de una propiedad. El proceso de determinar este valor se denomina *observación*. Los procedimientos de observación proporcionan una secuencia de instrucciones para garantizar que las observaciones sean reproducibles y representativas, por lo que una evaluación individual caracteriza una característica (es decir, una entidad) de interés. Por lo general, las observaciones no se llevan a cabo en toda la característica, sino en muestras de ella, o en una región espacio-temporal detectada inmediatamente. El proceso de **muestreo** puede especificarse en sí mismo mediante un procedimiento que determina cómo obtener muestras. Algunos procedimientos de observación pueden contener procedimientos de muestreo como partes. Las acciones desencadenadas por observaciones se denominan *actuaciones* y las entidades que las realizan son *actuadores*. [34]

La ontología **SOSA** proporciona una especificación formal pero ligera de propósito general para modelar la interacción entre las entidades involucradas en los actos de *observación*, *actuación* y *muestreo*. **SOSA** es el resultado de repensar la ontología **SSN** en función de los cambios en el alcance y el público objetivo, los desarrollos técnicos y las lecciones aprendidas en los últimos años. **SOSA** también actúa como un reemplazo del núcleo del Sensor de Observación de Estímulos (*Stimulus Sensor Observation* **SSO**) de **SSN**. Ha sido desarrollado por el primer grupo de trabajo conjunto del **OGC** y el **W3C**. En contraste con la ontología original **SSN**, **SOSA** adopta una perspectiva centrada en eventos y gira en torno a observaciones, muestreos, actuaciones y procedimientos. El último es un conjunto de instrucciones que especifican cómo llevar a cabo uno de los tres actos antes mencionados. Este modelo centrado en eventos está alineado con las expectativas de la comunidad, en particular la comunidad *Schema.org* que solo se preocupa por la representación digital de un evento y no el proceso del mundo real que en el que subyace tal evento. [34]

1.4.4. Ontología *GEO*

La ontología **GEO** comienza una exploración de las posibilidades de representar datos geoespaciales en **RDF**, y no intenta abordar mucho en los problemas cubiertos en el mundo profesional, en particular por el **OGC.** En su lugar, proporciona solo algunos términos básicos que se pueden utilizar cuando es necesario describir *latitudes*, *longitudes* y *altitudes* de acuerdo a la especificación de referencia **WGS84**[12]. La motivación para usar **RDF** como portador de información latitud-longitud es la capacidad de **RDF** para mezclar datos entre dominios. Se pueden describir no solo mapas, sino las entidades que se posicionan en el mapa. Y también se puede usar cualquier vocabulario **RDF** relevante para hacerlo, sin la necesidad de una precoordinación costosa o de cambios en un esquema mantenido de forma centralizada.

Su diseño permite que la información básica sobre los puntos se amplíe con metadatos más sofisticados o específicos de la aplicación en la que se utilice.

En ella se define una clase *Punto*, cuyos miembros son puntos. Los puntos se pueden describir utilizando las propiedades *lat*, *long* y *alt*, así como con otras propiedades **RDF** definidas en otros lugares. Por ejemplo, se podría usar una propiedad definida externamente como *bornNear* o *withinFiveMilesFrom*, o quizás otras propiedades para representar lat / long / alt en sistemas que no son **WGS84**. Las propiedades *lat* y *long* toman valores literales (es decir, valores textuales), cada uno en grados decimales. La propiedad *alt* son metros decimales sobre el elipsoide de referencia local. [10]

Esta ontología destaca por su simplicidad en cuanto a las entidades y relaciones que describe, lo cual la hace muy acertada para ser utilizada en el mundo de la **IoT.**

1.4.5. Ontología *QU*

Las cantidades y unidades, como la eslora de un barco medida en metros, son vitales para las ciencias y las ingenierías exactas. Grandes cantidades de datos cuantitativos se utilizan y se producen en experimentos científicos y en el diseño de artefactos diariamente; los cuales se almacenan en representaciones formales para que puedan ser manipulados por herramientas de análisis y diseño. En algunas ocasiones dichos datos necesitan ser integrados con otras fuentes, lo cual trae consigo problemas de interoperabilidad, no solo por las diferencias existentes en la forma de obtener los datos, sino también por la manera informal de anotarlos. [55] Es por ello que para

[12]Un sistema geodésico de coordenadas geográficas usado mundialmente, que permite localizar cualquier punto de la Tierra por medio de tres unidades dadas (x,y,z)

mejorar la anotación e integración de datos se han desarrollado varias ontologías, entre las cuales se encuentra **QU.**

La ontología **QU** se basa parcialmente en el modelo conceptual de *Cantidades*, *Unidades*, *Dimensiones* y *Valores* (**QUDV**) para definir sistemas de unidades y cantidades para su uso en modelos de sistemas propuestos por el *Grupo de Trabajo de Revisión* (**SysML** 1.2[13]) que trabaja en estrecha coordinación con el grupo de especificaciones **OMG MARTE**[14]**.** [27]

Esta ontología se desarrolla como complemento del ejemplo de meteorología agrícola que muestra la ontología desarrollada por el grupo de incubadoras *W3C Semantic Sensor Networks* (**SSN-XG**).

La entidades (cantidades, unidades, etc) descritas en este modelo son un subconjunto de las definidas por el *Sistema Internacional de Medidas* (**SI**) que son formalmente estandarizadas a través del **ISO31** y del **IEC60027.**

1.4.6. Ontología *IoTStream*

Si bien existen ontologías que formalizan los elementos principales de un sistema **IoT**, también existen ontologías que describen el flujo de datos que emana de ellos; entre dichas ontologías se encuentra **IoT Stream**. Esta ontología se basa en la ontología **SOSA** (1.4.3) y por consecuencia en **SSN** (1.4.2). La idea principal detrás de **IoT-Stream** es la simplicidad del modelo de información, y especialmente los flujos individuales, que son la parte más pesada de las anotaciones, ya que representan la mayor parte de la información anotada. Por lo tanto, cada observación de flujo se compone solo de *un valor* y una *marca de tiempo*. En ella se separan todos los metadatos necesarios para fines de búsqueda y rastreo, pero no necesarios para el procesamiento de datos en tiempo (*cuasi*) real. Además, permite anotar datos sin procesar, así como datos procesados, y ambos podrían anotarse como flujos y mantenerse livianos. [19]

El modelo de datos principal de esta ontología se enfoca en representar las observaciones de un flujo, su análisis y los eventos que se detectan a partir de ella, que se capturan en cuatro clases. Estas clases reflejan los conceptos de un flujo, *IoTStream*; una observación, *StreamObservation*; un proceso de analíticas, *Analytics*; y el evento en sí, *Event*. En la figura 1.1 se puede ver claramente como la clase *IoTStream* es la clase central, a la cual se vinculan las demás. Esta abstracción representa un flujo de datos que se origina de una fuente de datos de **IoT.** Tiene propiedades como *streamS-*

[13]**SysML** es un lenguaje de especificación de sistemas, un subconjunto ampliado de **UML 2.0** y desde el 2007 un estándar de la **OMG**

[14]Define las bases para la descripción basada en modelos de sistemas embebidos y en tiempo real

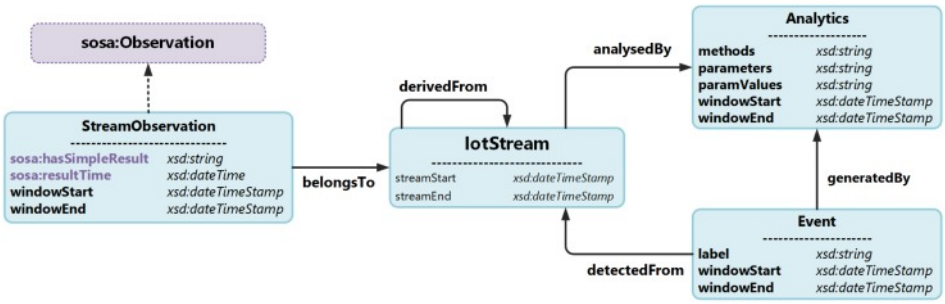


Figura 1.2: Clases y propiedades de **IoT-Stream**

tart y *streamEnd* que capturan la vida útil del flujo. Destacar aquí que un flujo de datos puede ser la fuente o el destino de otro flujo de datos, y es por ello que existe la relación *derivedFrom***.** La clase de mayor interés resulta ser *StreamObservation*, a la cual un flujo *IoTStream* pertenece (*belongsTo*). Estas observaciones pueden ser el resultado de las lecturas de un sensor, o la salida de un proceso de análisis (*Analytics*)**.** En el caso de que un flujo se analice mediante un proceso de análisis de datos (*analysedBy*), la clase *Analytics* captura la información relevante como los métodos de análisis utilizados (*methods*) y los parámetros con los cuales dichos métodos fueron aplicados (*parameters*). La clase *Analytics* también se puede utilizar exclusivamente para definir el proceso de análisis de datos que es utilizado para generar eventos que se detectan (*detectedFrom*) de un flujo de datos **IoT**. La clase *Event* contiene propiedades para capturar una descripción precisa del evento en sí, como puede ser el intervalo de tiempo en el cual el evento se produjo.

1.5. OBDA

Si bien las ontologías constituyen una manera efectiva de modelar un dominio determinado, acceder mediante ellas a los datos también resulta un acierto. El **Acceso a Datos Basado en Ontologías** (**OBDA** por sus siglas en inglés) es un enfoque destacado para consultar bases de datos que utilizan una ontología para exponer datos de una manera conceptualmente clara al abstraerse de los detalles técnicos a nivel de esquema de los datos subyacentes. La ontología está *"conectada"* a los datos a través de asignaciones que permiten traducir automáticamente las consultas planteadas sobre la ontología en consultas a nivel de datos que pueden ser ejecutadas por el sistema de gestión de base de datos subyacente. A pesar de la gran atención de la comunidad de investigadores, todavía hay pocos casos de uso industrial de sistemas **OBDA** en el mundo real.

Desde su introducción, el **OBDA** se ha convertido en un tema de investigación ampliamente aceptado en diferentes áreas como la Web Semántica, la Teoría de Bases de Datos y las Lógicas Descriptivas. La idea impulsora detrás del acceso a datos basado en ontologías es mantener los datos de origen donde están y acceder a ellos con una interfaz continua sobre una base de conocimiento declarativa (ontología).

La idea clave de **OBDA** es proporcionar a los usuarios acceso a la información en sus fuentes de datos a través de una arquitectura de tres niveles, constituida por *la ontología*, *las fuentes* y *el mapeo entre los dos*, donde la ontología es una descripción formal del dominio de interés, y es el corazón del sistema. [59] A través de esta arquitectura, **OBDA** proporciona una conexión semántica de extremo a extremo entre los usuarios y las fuentes de datos, lo que permite a los usuarios consultar directamente datos distribuidos en múltiples fuentes distribuidas, a través del vocabulario familiar de la ontología: el usuario formula consultas **SPARQL** sobre la ontología que son transformados, a través de la capa de mapeo, en consultas **SQL**[15] sobre las bases de datos relacionales subyacentes.

→ La capa de ontología en la arquitectura es el medio para aplicar un enfoque declarativo a la integración de la información y, de manera más general, a la gobernanza de datos. La base de conocimiento del dominio de la organización se especifica a través de una descripción formal y de alto nivel tanto de sus aspectos estáticos como dinámicos, representados por la ontología. Al hacer explícita la representación del dominio, se logra la reutilización del conocimiento adquirido, lo que no se logra cuando el esquema global es simplemente una descripción unificada de las fuentes de datos subyacentes.

→ La capa de mapeo conecta la capa de ontología con la capa de fuente de datos definiendo las relaciones entre los conceptos de dominio, por un lado, y las fuentes de datos, por otro lado. Estas asignaciones no solo se utilizan para el funcionamiento del sistema de información, sino que también pueden ser un activo importante para fines de documentación en los casos en que la información sobre los datos se distribuye en piezas separadas de documentación que a menudo son de difícil acceso y rara vez se ajustan a los estándares comunes.

→ La capa de fuente de datos está constituida por las fuentes de datos existentes de la organización.

[15]Es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales

El panorama para el procesamiento de flujos de datos (*streaming data*) dentro del paradigma **OBDA** todavía es prematuro. Existen pocos motores de *streams* [40, 12, 3] y estos aún están en desarrollo. Además se ha demostrado que carecen de una u otra funcionalidad básica. Sin embargo, el núcleo de todos parece ser la adición de operadores de ventanas deslizantes (*sliding windows*) sobre los *streams*, que son adaptados de los lenguajes de consulta sobre *streams* relacionales.

En la figura 1.3 se presenta la propuesta de [12] en la que puede observarse como el servicio recibe consultas especificadas en términos de las clases y propiedades de la ontología (capa de la ontología en el esquema **OBDA**) usando **SPARQL**$_{Stream}$, (el cual será presentado en la subsección 1.7.1), una extensión de **SPARQL** que incluye operadores para trabajar sobre *streams* en **RDF.** Luego, para transformar dicha consulta, expresada en términos de la ontología, en términos de las fuentes de datos se utiliza un conjunto de mapeos (capa de mapeo en el esquema **OBDA**) expresados en S_2O, una extensión de R_2O[16]. El resultado de este proceso, llamado **traducción de consulta** (*query translation* en la figura 1.3) tiene como objetivo el lenguaje de consulta **SNEEql**[17]**,** el cual es lo suficientemente expresivo para lidiar tanto con el *streaming* como con las fuentes almacenadas. Una vez la *consulta continua*[18] ha sido generada, comienza la fase de procesamiento de la consulta (*query processing*) y el evaluador usa diferentes técnicas de procesamiento de consultas distribuidas para extraer datos relevantes de las fuentes de datos y ejecutar la consulta en sí (selección, proyección o reunión). El resultado del procesamiento de las consultas es un conjunto de tuplas que el proceso de traducción (*data translation*) transforma en instancias de ontologías que son devueltas al cliente que inicialmente hizo la petición.

1.6. Datos en streaming

Los *streams* de datos surgen en muchos dominios de aplicaciones como el procesamiento de sensores, el monitoreo de redes y el análisis financiero. Los *streams* de diferentes dominios pueden significar cosas muy diferentes: una señal discreta, un evento de *log*, una combinación de series de tiempo, etc. Esto trae consigo que la interpretación de lo que realmente es un *stream* varíe de aplicación en aplicación.

[16]Un lenguaje declarativo para describir mapeos entre esquemas de bases de datos relacionales y ontologías implementadas en **RDF** u **OWL**

[17]**SNEEql** es un lenguaje de consulta declarativo continuo sobre *streams* de datos detectados por redes de sensores. Se destaca por su gran expresividad.

[18]Una consulta continua es aquella se ejecuta automáticamente y periódicamente sobre datos en tiempor real y almacena sus resultados en una medida específica

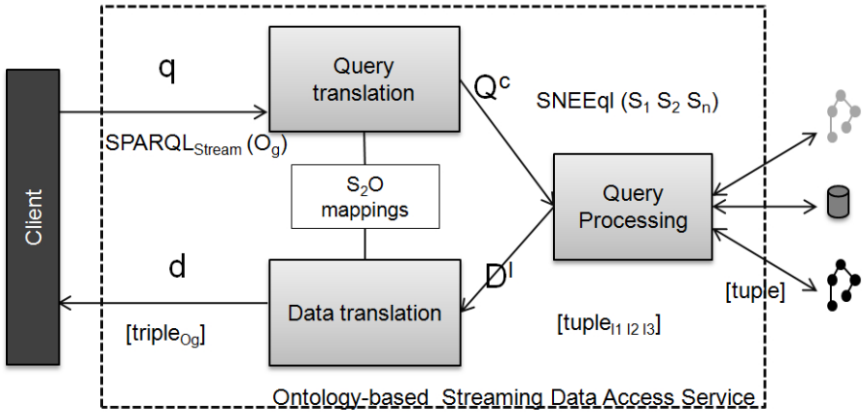


Figura 1.3: Propuesta presentada en [12]

Law, Wang y Zaniolo en [39] definen los *streams* de datos como una *"bolsa"* de tuplas ordenadas donde solamente se pueden incorporar datos, o, alternativamente, "bolsas" en las cuales se pueden adjuntar elementos de manera ilimitada cuando existe una marca de tiempo explícitamente asociada a cada tupla. Dicha definición combina el tipo de estructura que denota un *stream* (**una secuencia ilimitada**) con una representación en particular (**tuplas con marcas de tiempo**) y con la función para recuperar uno de otro (**adjuntar**). Esta definición deja muchas preguntas sin responder. Por ejemplo, ¿se consideran las marcas de tiempo como parte del contenido de los elementos del *stream*? o ¿la bolsa de tuplas debe ser considerada como un conjunto y por ende ignorar las tuplas duplicadas? Es por ello que resulta importante distinguir entre la *denotación* de un stream de su *representación* en particular. La *denotación* es una interpretación abstracta de qué representa el *stream,* como estructura matemática, en algun dominio; mientras que la *representación* es una codificación particular que se utiliza para los elementos de dicho dominio. [44]

Por otro lado, Margara y Rabl en [45] definen un *stream* de datos como una secuencia numerable de elementos que es usada para representar datos que están disponibles a lo largo del tiempo. Esta definición, a diferencia de la anterior, se abstrae de la representación del *stream* y por tanto resulta más acertada. Formalizando esta última se puede decir que: un *stream* de datos es un par ordenado (s, Δ) donde:

1. s es una *secuencia*[19] de *tuplas*[20]

2. Δ es una secuencia de intervalos de tiempo reales y positivos

[19]Una colección enumerada de objetos con repeticiones admitidas y donde interesa el orden
[20]Una lista finita y ordenada de elementos

1.6.1. Desafíos

Los *streams* ofrecen flujos continuos de datos que se pueden consultar para obtener información. Generalmente, estos datos deben estar en orden, pero, debido a que los datos pueden provenir de diferentes fuentes, o incluso de la misma fuente pero moviéndose a través de un sistema distribuido, surge el problema de manener los datos en orden para luego entregarlos al consumidor. Es por ello que los *streams* de datos enfrentan directamente lo estipulado en la **Conjetura de Brewer**, o como es más comúnmente conocido, **Teorema CAP** presentado en [9]. Este teorema enuncia que es imposible para un sistema de cómputo distribuido garantizar simultáneamente:

→ **Consistencia** (***C****onsistency*)**:** Cualquier lectura recibe como respuesta la escritura más reciente o un error

→ **Disponibilidad** (**A**vailability)**:** Cualquier petición recibe una respuesta no errónea, pero sin la garantía de que contenga la escritura más reciente

→ **Tolerancia al particionado** (***P****artition Tolerance*)**:** El sistema sigue funcionando incluso si un número arbitrario de mensajes es descartado (o restrasado) entre nodos de la red

En la figura 1.4 se puede apreciar la relación existente entre estos tres elementos, a los que se tiene que enfrentar un arquitecto de datos a la hora de elegir un sistema de persistencia determinado. La **X** en la intersección de los tres conjuntos presentados en 1.4 indica la imposibilidad de alcanzar las tres características a la vez; solamente se pueden asegurar a lo sumo dos de manera simultánea.

1.7. Consultas sobre *streams*

Una vez que se ha presentado una manera más apropiada que los tradicionales sistemas de bases de datos relacionales para acceder a los datos, se hace imprescindible presentar cómo es posible acceder a ellos y poder manipularlos tal y como se hacía con **SQL**. En esta sección serán presentados algunos de los lenguajes de consulta utilizados para acceder y manipular *streams* de datos. Destacar que algunas de dichas variantes solo se encuentran en la literatura o al menos no existe una implementación estable (hasta la fecha) de motores capaces de interpretar y ejecutar dichos lenguajes de consulta.

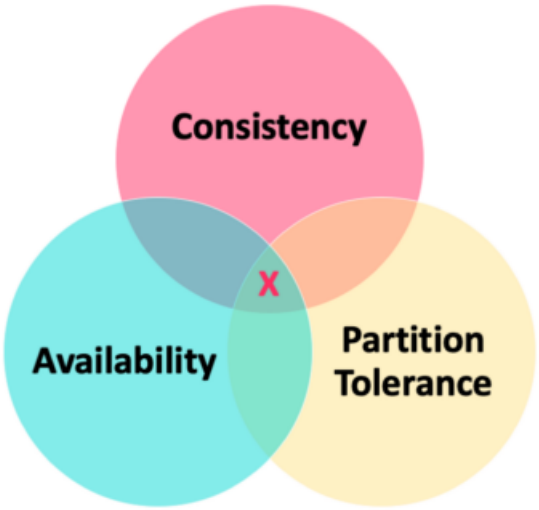


Figura 1.4: **Teorema CAP**

1.7.1. SPARQLstream

Para permitir el acceso a los *streams* de datos basado en ontologías, se requiere un modelo de datos general capaz de representar tanto las conceptualizaciones presentes en las ontologías como los aspectos temporales y del *stream* de los datos. Este modelo también requiere de un lenguaje de consulta capaz de acoplarse con estas características. Los *streams* **RDF** surgen en la literatura para dar cumplimiento a dicho modelo de datos mientras que, variantes de lenguajes de consulta para dicho modelo de datos han sido propuestos. Sin embargo, todas estas extensiones asumen que los *streams* son representados nativamente en **RDF** mientras que, en muchos escenarios, se utilizan motores **DSMS** (*Data Stream Management Systems*) o **CEP** (*Complex Event Processing*) como intermediarios para manejar y almacenar los datos dinámicamente.

SPARQL$_{stream}$ surge como una extensión de **SPARQL** para consultar datos **RDF** en *streaming*. Este lenguaje permite consultar y exponer los datos en términos de conceptos de alto nivel de una ontología mientras que, internamente, los datos son manejados por estructuras o esquemas entendibles por el motor **DSMS** o **CEP** que actúa como intermediario subyacente.

La sintaxis de **SPARQL**$_{stream}$ es muy similar a la de **SPARQL 1.1** añadiendo los constructores necesarios de ventanas (*sliding windows*) para grafos de *streams*

RDF y los modificadores adicionales para la solución. En **SPARQL**$_{stream}$ cada *stream* **RDF** virtual es identificado por un **IRI**, de manera que pueda ser usado o referenciado en cualquier lugar de la consulta. Para indicar que un determinado grafo *stream* es usado en una consulta, su **IRI** tiene que ser referenciado en la cláusula **FROM**, precedido por las palabras clave **NAMED STREAM**. Las definiciones de ventanas de tiempo de la forma [*start* **TO** *end* **SLIDE** *slide*] pueden ser aplicados a los grafos agregándolas al final del **IRI**. Opcionalmente se puede agregar la palabra clave **SLIDE** para especificar cuán a menudo la ventana será evaluada (ej. cada minuto, cada hora). [12]

1.7.2. C-SPARQL

Los repositorios **RDF** con el transcurso del tiempo se están volviendo cada vez más densos. Si bien **SPARQL** soporta consultas complejas sobre múltiples fuentes, a la hora de lidiar con combinaciones de conocimiento estático (o lentamente cambiantes) y datos rápidamente cambiantes (datos en *streaming*) presenta dificultades. Ante estas dificultades surge **Continuous SPARQL** o simplemente **C-SPARQL** como extensión a **SPARQL** para consultar **RDF streams**. Este lenguaje de consulta une los *streams* de datos con el razonamiento permitiendo el razonamiento en *streaming,* una nueva e inexplorada área de alto impacto debido al incremento sustancial de las redes de sensores en los últimos años.

Los **RDF streams** son análogos a los grafos **RDF**. Cada **RDF stream** se identifica usando un **URI,** pero en vez de ser una colección de *ternas,* se considera una secuencia de ternas **RDF** que son continuamente producidas y anotadas con marcas de tiempo.

La introducción de dichos **RDF streams** trae consigo la necesidad de identificar dichos tipos de datos y aplicar un criterio de selección sobre ellos. Para lograr la identificación se asume que cada *stream* de datos esta asociado con un **IRI** distinto y para la selección, debido a que los *streams* son intrínsecamente infinitos, se introduce la noción de *ventana*. Una *ventana* considera las *ternas* más recientes de dichos *streams*, observados mientras los datos fluyen continuamente. [3] El soporte de *streams* en formato **RDF** garantiza la interoperabilidad y permite su utilización en diversas áreas, en las que los razonadores pueden lidiar con el conocimiento que evoluciona con el tiempo. Ejemplos de tales dominios de aplicación incluyen razonamiento en tiempo real sobre sensores, computación urbana y datos semánticos sociales. [15]

1.7.3. EP-SPARQL

Los *streams* de eventos aparecen cada vez más en varias aplicaciones web como *blogs*, *feeds*, *streams* de datos de sensores, información geoespacial, datos financieros en línea, etc. El procesamiento de eventos (**EP**) se ocupa de la detección oportuna de eventos compuestos dentro de *streams* de eventos simples. El **EP** de última generación proporciona un análisis sobre la marcha de *streams* de eventos, pero no puede combinarlos con conocimientos previos y no puede realizar tareas de razonamiento. Por otro lado, las herramientas semánticas pueden manejar de manera efectiva el conocimiento previo y realizar el razonamiento al respecto, pero no pueden lidiar con los datos que cambian rápidamente proporcionados por los *streams* de eventos. Para cerrar esta brecha, **EP-SPARQL** surge como un nuevo lenguaje para eventos complejos y el razonamiento sobre *streams* (*Stream Reasoning*). El modelo de ejecución de **EP-SPARQL** se basa en la programación lógica y presenta capacidades efectivas de procesamiento e inferencia de eventos sobre el conocimiento temporal y estático.

EP-SPARQL especifica eventos complejos ubicando temporalmente los datos en *streaming* en tiempo real y utiliza ontologías de fondo para permitir el razonamiento del *stream* basándose en reglas de encadenamiento hacia atrás (*event-driven backward chaining rules*). [2]

1.7.4. Instants

Los entornos inteligentes requieren la colaboración de sensores multiplataforma operados por múltiples partes. Las soluciones de procesamiento de eventos patentadas carecen de flexibilidad de interoperación, lo que lleva a funciones superpuestas que pueden desperdiciar hardware y recursos de comunicación. **Instants** surge como una plataforma de alto rendimiento basada en **Rete**[21] para la ejecución continua de consultas **SPARQL** interconectadas. Esta plataforma permite el procesamiento en tiempo (*quasi*) real de eventos complejos y heterogéneos. **Instants** realiza una evaluación continua de los datos **RDF** entrantes frente a múltiples consultas **SPARQL**. Los resultados intermedios se almacenan en una red de nodos y cuando todas las condiciones de una consulta coinciden, el resultado está disponible instantáneamente.

La estructura de **INSTANTS** se ilustra en la figura 1.5. El sistema consta de un motor **Rete** y de conectores de entrada y salida, que pueden interactuar con la red, almacenes de ternas, archivos u otros procesos. El motor de **Rete** tiene cuatro

[21] **Rete** es un eficiente algoritmo de reconocimiento de patrones para implementar un sistemas basados en reglas.

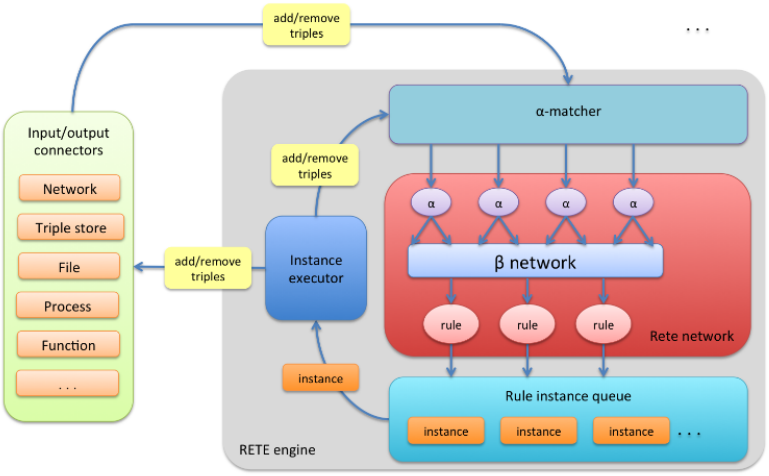


Figura 1.5: Estructura de **INSTANTS** presentada en [56]

componentes:

1. La red **Rete**
2. Un analizador de coincidencias llamado $\alpha - matcher$
3. Una cola de instancias de reglas
4. Una instancia ejecutora

El analizador de coincidencias ($\alpha - matcher$) y la red **Rete** son capaces de encontrar todas las reglas **SPARQL** que se cumplan en el conjunto actual de ternas. En tiempo de ejecución, el analizador de coincidencias recibe comandos para añadir o eliminar ternas. Este analizador encuentra todos los nodos α ($\alpha - nodes$) de **Rete** que coinciden con las ternas y llama a los métodos de añadir o eliminar de dichos nodos. Los cambios se propagan a través de la red β ($\beta - network$) y, finalmente, las condiciones de las reglas completamente satisfechas ingresan a los nodos de reglas, que agregan nuevas intancias de reglas (con enlaces variables) a la cola de instancias de reglas. El ejecutor de instancias ejecuta las instancias de cada regla, lo que hace que los comandos de agregar o eliminar ternas alimenten los conectores de salida. La ejecución de una instancia de regla también alimenta de comandos de adición y eliminación de ternas al analizador de coincidencias, dando como resultado nuevas instancias de reglas.

Este modelo ha permitido que el rendimiento de **INSTANTS** sea superior comparado con otros sistemas basados en ejecución repetida de consultas en intervalos fijos de tiempo (o conteo de ternas), los cuales prácticamente no pueden competir con él, cuyos retrasos en las notificaciones son del orden de los milisegundos. Esta

eficiencia se debe a que **INSTANTS** evita el cálculo redundante, donde cada evento se procesa inmediatamente a su llegada y solo una vez a través de la red **Rete**, las estructuras de la red son compartidas a través de consultas similares y los resultados intermedios se almacenan en caché. [56]

1.7.5. CQELS

Usualmente se suele considerar que los **Datos Enlazados** (*Linked Data*) cambian con poca frecuencia. Primeramente, los datos son recopilados y luego almacenados en un repositorio centralizado antes de ser procesados. Las actualizaciones en los datos, una vez almacenados, se limitan a una pequeña fracción del conjunto de datos y ocurren esporádicamente, o el conjunto de datos es reemplazado por la nueva versión. El procesamiento de consultas sobre dichos datos, siguiendo el modelo presentado por las bases de datos relacionales, consta de tres pasos:

1. Leer los datos del disco
2. Ejecutar la consulta
3. Entregar los datos que satisfacen la consulta en forma de conjunto

Este proceso se inicia una vez por cada consulta ejecutada en el tiempo. En contraste, cuando se trabaja con *streams* de **Datos Enlazados**, los nuevos datos se producen continuamente, los cuales son válidos solamente durante una ventana de tiempo y son periódicamente enviados al procesador de consultas. Las consultas, son registradas una vez y son evaluadas continuamente a medida que transcurre el tiempo debido a la constante actualización de los datos. Los resultados de dichas consultas son actualizados a medida que aparecen nuevos datos. Por tanto, es fácil darse cuenta de que los motores de procesamiento de consultas existentes para **Datos Enlazados** no son adecuados para manejar *streams* de dichos datos. Soluciones que tratan de abordar este problema como **C-SPARQL** (1.7.2), **SPARQL**$_{stream}$ (1.7.1) y **EP-SPARQL**(1.7.3) usan un modelo de *caja negra*, que delega el procesamiento a otros motores como aquellos dedicados al procesamiento de eventos en *streaming* y los procesadores de consultas **SPARQL** haciendo las respectivas traducciones a los lenguajes de dichos motores en los que se delega el procesamiento de la consulta. Esta dependencia introduce una sobrecarga tanto en la traducción de la consulta como en la transformación de los datos. Las consultas primero necesitan ser traducidas en los lenguajes que entienden los sistemas en los cuales se hace la delegación. Y luego los datos necesitan ser transformados de vuelta para ser entregados al sistema que delega

en sí. **CQELS** (*Continuos Query Evaluation over Linked Streams*) surge para remediar estas desventajas, al proveer un motor unificado para procesar consultas sobre *streams* de **Datos Enlazados** y **Datos Enlazados** en sí. En contraste a los sistemas existentes, **CQELS** usa una aproximación de *caja blanca*, que define su propio modelo de procesamiento, que es implementado en el propio motor de consulta. Siguiendo esta filosofía, **CQELS** provee de un *framework* de ejecución de consultas flexible donde el procesador de dichas consultas se adapta dinámicamente a los cambios de los datos de entrada. Cuando la consulta está siendo ejecutada, los operadores de dicha consulta se están reordenando continuamente de acuerdo a ciertas heurísticas para mejorar el tiempo de ejecución de la consulta en términos de tiempo y complejidad. De igual manera, el acceso a disco en grandes colecciones de **Datos Enlazados** se reduce codificando los datos y almacenando en caché los resultados intermedios, y mejorando el tiempo de acceso a los datos utilizando técnicas de indexación. [40]

1.7.6. STARQL

Todos los sistemas recientes para procesar *streams* que utilizan modelos conceptuales tienen en común que utilizan una semántica de *"bolsa de aserciones"* para la *ventana deslizante*, mediante la cual todas las aserciones entrantes en un *stream* (independientemente de su etiqueta) son colocadas en una bolsa. A pesar de que esto constituye una adaptación directa de la semántica para *streams* relacionales, debe tenerse en cuenta que esta pérdida de información temporal, es decir, el marco de tiempo en que los eventos ocurren, conlleva a problemas debido a posibles inconsistencias lógicas que pueden ocurrir con respecto al modelo conceptual (ontología). Es por ello que el lenguaje de consulta **STARQL** implementa una semántica diferente basada en el concepto general de secuenciación de *ABox*[22]. Las aserciones en una ventana dependen de las marcas de tiempo y la estrategia de secuencia elegida agrupada en *ABox*s, dando como resultado una secuencia de *ABox*s en cada punto de tiempo.

STARQL se encuentra en la intersección del **OBDA** clásico y el procesamiento de *streams*. Este lenguaje de consulta amplía los conceptos de ventana deslizante, que se conocen en muchos lenguajes para sistemas de gestión de datos de *streams* relacionales, así como sistemas recientes para **RDFS**, con constructores de secuenciación de **ABox**. La ventaja de utilizar una metodología basada en secuencias sobre otros enfoques es que, en primer lugar, la secuencia establece un contexto estándar en el que se pueden aplicar los servicios de razonamiento **OBDA** estándar, y en segundo lugar,

[22]Un hecho asociado con un modelo conceptual u ontología dentro de una base de conocimiento

que el lenguaje de consulta puede equiparse con una semántica ordenada basada en cierta semántica de respuesta. La gran expresividad de **STARQL** a nivel conceptual con respecto a los cálculos aritméticos y estadísticos, así como la especificación de eventos se puede implementar de manera segura para alcanzar la independencia del dominio. Esto deja el terreno preparado para una completa y correcta transformación de lenguajes de consulta sobre *streams* en las fuentes de datos del *backend. [51]*

1.8. Servicios para la detección de eventos basados en ontologías

La gran utilidad que han demostrado tener los dispositivos **IoT** ha traído consigo la proliferación de conceptos como **OpenData** y **Smart Cities**, que si bien ya existían idealmente, la **IoT** los ha hecho concretarse. La instalación de dichos dispositivos en distintas ciudades ha permitido monitorear en tiempo real por ejemplo, la temperatura en un invernadero, el ritmo cardíaco de pacientes en sus hogares, la velocidad con la que viajan los automóviles por una autopista, etc. De igual manera, su apertura a todo el público ha desprendido un sin número de proyectos e investigaciones con grandes aplicaciones en la medicina, la ingeniería y las finanzas. En este capítulo se presentan algunas iniciativas de distintos gobiernos que servirán de ejemplo a lo anteriormente planteado.

1.8.1. ACTIVAGE

ACTIVAGE es un piloto europeo multicéntrico a gran escala sobre entornos de vida inteligentes. El objetivo principal es construir el primer ecosistema de **IoT** europeo en nueve Sitios de Despliegue (*Deployment Sites* **DS**) en siete países europeos, reutilizando y ampliando las plataformas, las tecnologías y los estándares de **IoT** subyacentes ya sean libres o patentados, e integrando nuevas interfaces necesarias para proporcionar interoperabilidad a través de estas plataformas heterogéneas, que permitirá la implementación y operación a gran escala de soluciones y servicios basados en **IoT** para el envejecimiento activo y saludable, apoyando y extendiendo la vida independiente de los adultos mayores en sus entornos de vida y respondiendo a las necesidades reales de los cuidadores, proveedores de servicios y autoridades públicas. [1]

La visión de **ACTIVAGE** es ser la referencia a nivel mundial para proporcionar la evidencia de que los ecosistemas de **IoT** estándar, seguros e intraoperativos permiten

nuevos modelos comerciales y soluciones rentables para el envejecimiento activo y saludable, contribuyendo a la sostenibilidad de los sistemas de salud y atención, la competitividad de la industria europea a través de la innovación y la mejora de la calidad de vida y la autonomía de los adultos mayores en forma de vida independiente.

Otros objetivos de **ACTIVAGE** son:

→ Ofrecer **ACTIVAGE IoT Ecosystem Suite** (**AIOTES**), un conjunto de técnicas, herramientas y metodologías para la interoperabilidad en diferentes capas entre plataformas de **IoT** existentes heterogéneas y un marco abierto para proporcionar interoperabilidad semántica de plataformas de **IoT** para **AHA**, abordando la confiabilidad, la privacidad, la protección de los datos y la seguridad

→ Proporcionar un marco de cocreación que permita la identificación, la medición, la comprensión y la predicción de las demandas y necesidades del ecosistema de **IoT** en los usuarios de **AHA**: adultos mayores, cuidadores, profesionales y proveedores de atención sanitaria y social, evaluando sus necesidades, preferencias y percepciones con respecto a aceptación del usuario, confianza, confidencialidad, privacidad, protección de datos y seguridad. El objetivo principal es plantear e identificar algunos factores clave de éxito desconocidos relacionados también con el despliegue y la ampliación de las actividades.

→ Establecer y poner en funcionamiento un programa de comunicación que permita la difusión mundial de las actividades y logros del proyecto, para hacer de **ACTIVAGE** un marco de referencia global de valores basados en evidencia de **IoT** para **AHA**.

1.8.2. Londonair

Londonair es el sitio web de *London Air Quality Network* (**LAQN**) que muestra la contaminación del aire en la ciudad de Londres y el sureste de Inglaterra. El sitio web proporciona información para el público, los usuarios de política y los científicos. **LAQN** se formó en 1993 para coordinar y mejorar el monitoreo de la contaminación del aire en Londres. La red proporciona evaluaciones y mediciones científicas independientes. Este sitio no monitorea toda la zona en sí, ya que establecer y administrar un sitio de monitoreo requiere una gran inversión, y no es posible colocar estaciones de monitoreo en todas partes, en cambio el sito sí presenta mapas y modelos para predecir los niveles de contaminación. [43]

Este sitio surge debido a la preocupación concerniente a la contaminación del aire por parte del departamento de salud pública de Londres. Las mediciones que aquí se obtienen son utilizadas, además de para evaluar la contaminación del aire, para rastrear las tendencias de la misma a lo largo del tiempo y para crear modelos que pueden evaluar cómo las diferentes políticas gubernamentales afectan la contaminación del aire. Estas mediciones además ayudan a cumplir con las obligaciones legales de las autoridades locales con respecto a la contaminación del aire.

Londonair es mantenido por el Grupo de Investigación Ambiental (**Environmental Research Group**) del *Imperial College* de Londres. El monitoreo es propiedad y está financiado por las autoridades locales, los Distritos de Mejoramiento de Negocios (**Business Improvement Districts**), la **TFL**[23] y el **Defra**[24].

1.8.3. Open Data DK

Open Data DK es una asociación de municipios y regiones danesas que desde 2016 han colaborado para abrir sus datos, es decir, en un portal común de datos abiertos. [16]

Los datos abiertos son datos no personales a los que cualquier persona puede acceder y utilizar de forma gratuita. Pueden ser cualquier cosa, desde datos sobre la infraestructura de los municipios hasta la composición socioeconómica. El propósito de los datos abiertos es:

→ Crear transparencia en la administración pública

→ Crear un terreno fértil para el crecimiento y la innovación basados en datos

→ Garantizar un mayor grado de utilización de los datos ya recopilados.

De esta manera, los datos abiertos pueden utilizarse en el desarrollo de aplicaciones y servicios o ser un punto de partida para el análisis, las evaluaciones de tendencias, la investigación, etc. **Open Data DK** tiene un portal de datos desde el cual todos pueden usar los datos de forma gratuita sin necesidad de registrarse. El portal de datos se basa en el software de código abierto **CKAN**[25] de la **Open Knowledge**

[23] *Transport for London* (**TFL**) es el organismo del gobierno local responsable de la mayoría de los aspectos del sistema de transportes de Londres

[24] El *Department for Environment, Food and Rural Affairs* (**Defra**) es el departamento de gobierno responsable por la protección del medio ambiente, la producción de alimentos y sus estándares, así como la agricultura, la pesca y las comunidades rurales en el Reino Unido

[25] Una aplicación web de código abierto para el almacenamiento y la distribución de datos

Foundation[26].

Todos los conjuntos de datos del portal **Open Data DK** están agrupados para crear una descripción general. Se dividen en las mismas categorías que en el **Portal Europeo de Datos**. Entre algunos de estos conjuntos de datos que son ofrecidos están los relacionados con:

→ Población y sociedad, que incluyen: datos demográficos, migración, empleo, socioeconomía, etc.

→ Energía, que inlcuyen: consumo de energía, fuentes de energía, etc.

→ Salud, que inlcuyen: aparatos de asistencia, asistencia odontológica, tratamientos por drogas / alcohol, etc.

→ Transporte, que incluyen: movilidad, aparcamiento, carreteras, mantenimiento invernal, transporte público, etc.

1.9. Arquitecturas basadas en ontologías para sistemas de transporte inteligentes

Los sistemas de transporte inteligentes son un conjunto de soluciones tecnológicas que se utilizan para mejorar el rendimiento y seguridad del transporte por carretera. Un elemento crucial para el éxito de estos sistemas es el intercambio de información, no solo entre vehículos, sino también entre otros componentes de la infraestructura vial a través de diferentes aplicaciones. Una de las fuentes de información más importantes en este tipo de sistemas son los sensores. Los sensores pueden estar dentro de los vehículos o como parte de la infraestructura, como puentes, carreteras, señales de tráfico, semáforos, etc. Dichos sensores pueden proporcionar información relacionada con las condiciones meteorológicas y la situación del tráfico, lo que es útil para mejorar el proceso de conducción [21]. Para facilitar el intercambio de información entre las diferentes aplicaciones que utilizan datos de sensores, se necesita un marco común de conocimiento que permita la interoperabilidad, tal y como se expresó en la sección 1.1.1.

En los últimos años, ha habido un interés creciente en las ontologías para los sistemas de transporte por carretera. Gorender y Silva en [23] desarrollaron una

[26]Una fundación sin ánimos de lucro que apoya la difusión del conocimiento abierto en su sentido más amplio

ontología para representar el tráfico en las carreteras. Su objetivo fue la construcción de un sistema de información de tráfico confiable que proporcione información sobre carreteras, tráfico y escenarios relacionados con los vehículos en las carreteras. También ayuda al sistema de información de tráfico a analizar qué tan crítica es una situación específica. Por ejemplo, es posible que una ambulancia necesite conocer el estado de congestión de una plaza de peaje. Solicitar esta información es fundamental si la ambulancia se traslada al lugar del accidente. Por otro lado, si un vehículo normal se mueve por una carretera sin prisa, entonces esta información solicitada no es tan crítica.

Morignot y Nashashibi en [47] propusieron una representación de alto nivel de un vehículo automatizado, otros vehículos y su entorno, que puede ayudar a los conductores a tomar decisiones de relajación "poco ortodoxas" pero prácticas (por ejemplo, cuando un automóvil dañado no permite la circulación, tomar la decisión de moverse a otro carril cruzando una línea continua y adelantar al automóvil detenido, si el otro carril está despejado). Esta representación de alto nivel incluye conocimiento topológico y reglas de inferencia, con el fin de calcular el próximo movimiento de alto nivel que debe realizar un vehículo automatizado, como ayuda para la toma de decisiones del conductor. La principal debilidad de este enfoque es la falta de reglas que representen las normas de tráfico anteriores. Acaban de definir un conjunto de infracciones a las normas de tránsito, que permiten clasificar la moción dada como "legal" o "ilegal".

Zhao, Ichise y Sasaki en [68] introdujeron una base de conocimientos basada en ontologías, que contiene mapas y normas de tráfico. Puede estar al tanto de situaciones de velocidad y tomar decisiones en las intersecciones para cumplir con las normas de tránsito, pero no considera elementos importantes como las señales de tránsito y las condiciones climáticas.

El trabajo propuesto en [32] es un enfoque para crear una descripción de situación genérica para sistemas avanzados de asistencia al conductor utilizando razonamiento lógico en una base de conocimiento de la situación del tráfico. Contiene múltiples objetos de diferente tipo como vehículos y elementos de infraestructura como carreteras, carriles, intersecciones, señales de tránsito, semáforos y relaciones entre ellos. La inferencia lógica se realiza para verificar y ampliar la descripción de la situación e interpretar la situación, por ejemplo, razonando sobre las reglas de tráfico. Las capacidades de este enfoque de descripción de situación ontológica se muestran en el ejemplo de intersecciones complejas con varias carreteras, carriles, vehículos y diferentes combinaciones de señales de tráfico y semáforos. Como restricción, en este

trabajo, la carretera de destino que pasa sobre la intersección debe ser conocida para cada vehículo,por lo que no es posible modelar diferentes posibilidades de acuerdo con la situación real de la intersección.

Capítulo 2

Traffic, un sistema para la detección de eventos de tráfico basado en ontologías

En este capítulo se presentará un modelo de solución para implementar un sistema inteligente para la detección de eventos de tráfico basado en **IoT** y anotaciones semánticas. Las primeras secciones introducen los componentes del sistema, mientras que las últimas se encargarán de ejemplificar cómo se comunican esos componentes entre ellos y conceptualmente, mediante datos enlazados, con otras definiciones de dominio. Finalmente se ejemplificará cómo consultar semánticamente los datos anotados mediante el lenguaje **SPARQL** presentado en 1.2.1.

2.1. Diseño de componentes

En la figura 2.1 pueden observarse los componentes principales del sistema, enumerados del **1** al **6** con círculos verdes. El componente número 1, llamado *Productor,* es el responsable de registrar los *streams* y publicar las observaciones generadas por los sensores. Adicionalmente, este componente almacena información propietaria de los sensores en sí y los *streams* que todavía siguen abiertos. En esta variante de implementación, se ha decidido obtener la información relacionada con los sensores y sus observaciones de terceras fuentes y no de sensores físicos como tal. Pero, en cualquier caso, la implementación se abstrae de dichas situaciones y es capaz de, con cambios mínimos, pasar de una a otra. Para el sistema presentado se escogió la *API*[1]

[1] *Application Programming Interface*

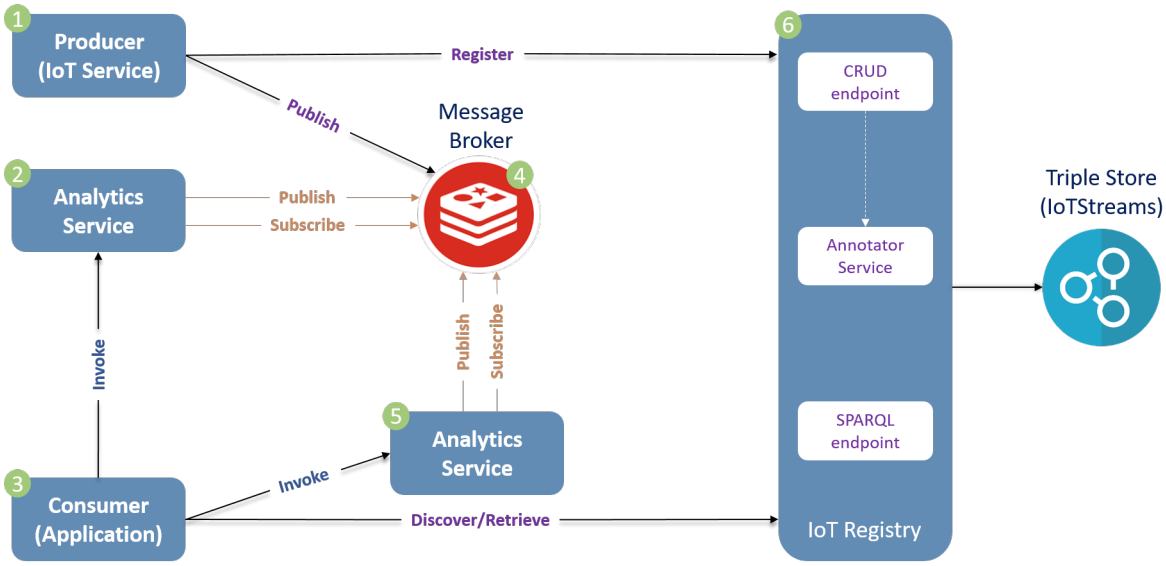


Figura 2.1: Esquema general del sistema

https://data.mobility.brussels/traffic/api/counts/. Dicha decisión estuvo avalada por la buena documentación que presenta y el buen diseño de la misma, en cuanto a usabilidad y flexibilidad se refiere. Los sensores ofrecidos por dicho servicio están ubicados en carriles de distintas carreteras de la ciudad de Bruselas de Bélgica y, para cada sensor se ofrecen sus observaciones en intervalos variables de 1, 5, 15 y 60 minutos. Una vez iniciado el servicio **1**, se gestiona una primera petición a la *API* antes mencionada y se obtiene la información necesaria sobre los sensores para luego registrarla en el componente **6** y enriquecer así los *streams*. Este servicio, se encarga además de asociar cada *stream* con su sensor correspondiente y de mantener actualizado el *stream* con las observaciones producidas por sus sensores. La *API* ofrece tres datos en tiempo real, a saber: **cantidad de autos**, **velocidad promedio** y **porcentaje de ocupación del sensor**, que no es más que el tiempo que un sensor estuvo ocupado por un vehículo; para nuestro sistema solo se utilizaron los dos primeros para realizar el posterior análisis de datos.

Este componente fue escrito totalmente utilizando el lenguaje *Java* y el kit de herramientes de Vertx por su naturaleza asíncrona y uso eficiente de recursos, requisito indispensable para un sistema basado en **IoT**. El repositorio para dicho componente puede ser encontrado en GitHub siguiendo este enlace.

Los componentes **2** y **6**, se encargan del análisis y el preprocesamiento de los datos respectivamente. Para el funcionamiento del sistema se puede prescindir del componente **6** debido a que su próposito está encaminado a refinar los datos que luego consumirá el componente **2**. Dichos componentes están implementados en el lengua-

je *Python* debido a la infinidad de bibliotecas para el análisis y la manipulación de datos que contiene. El despliegue de dichos componentes es bajo demanda; es decir, que en cualquier instante de tiempo dichos servicios pueden estar ejecutándose o no, todo depende de si se requiere análisis de datos y con ello detección de eventos. El componente **2** debe tener un clasificador previamente construido para poder detectar los eventos de los stream a los cuales fue suscrito. Para construir dicho componente se recopilaron los datos de todos los sensores ofrecidos por la *API* cada 5 min durante un período de una semana. Una vez hecho esto, se utilizaron dichos datos para conformar grupos (*clústers*) mediante el algoritmo **GMM** (*Gaussian Mixture Models*) y luego, a cada grupo, se le asoció una etiqueta en dependencia del estado del tráfico que representó dicho grupo, a saber: *BAJO*, *NORMAL* o *ALTO*. Luego se construyó un clasificador con dichos datos ya etiquetados, que es el encargado de detectar los eventos de tráfico bajo demanda. Estos servicios, además de el análisis de datos, se encargan de publicar sus descubrimientos en el componente **4** para luego ser consumidos por otros servicios.

El componente **3** también es opcional y pueden existir varias implementaciones e instancias del mismo, todo está en dependencia de cómo se quieran presentar los eventos detectados y las observaciones de los sensores. Por ejemplo, puede ser que se requiera un sistema para monitorizar la presencia de dichos eventos de tráfico, el cual será utilizado por varios operarios humanos; en dicho caso un cliente web sería ideal como implementación para el componente **3**. Por otro lado, si lo que se quiere es utilizar el sistema para ofrecer dichos eventos a otros sistemas, entonces hay dos variantes, primero no implementar el componente **3** y consumir los eventos directamente desde el componente **4** o restringir el consumo de dichos eventos a través de una *API GATEWAY*[2]. Para el sistema en concreto, se decidió implementar la primera variante, mediante una application web escrita en Angular para poder visualizar el estado de los sensores, sus observaciones, los *streams* y los eventos generados.

En el componente **6** reside todo el peso semántico del sistema. Este componente expone una interfaz *REST* para interactuar con los siguientes recursos: **sensores** y **streams**. Los metadatos son almacenados en un *triplestore*[3], en particular el ofrecido por Stardog. Se decidió utilizar esta implementación por su simpleza y útil panel de adminstración visual al cual se puede acceder desde una instancia ya sea local o

[2] *Es un sistema intermediario que proporciona una interfaz API REST o WebSocket para hacer de enrutador desde un único punto de entrada, el API Gateway, hacia un grupo de microservicios y/o API de terceros definidos*

[3] *Una triplestore o almacén de RDF es una base de datos especialmente diseñada para el almacenamiento y recuperación de ternas a través de consultas semánticas*

remota mediante el siguiente enlace. En este componente, puede verse la existencia de un subcomponente (el servicio de anotaciones), el cual se encarga de transformar los datos recibidos sobre los sensores y los *streams* en documentos **RDF** para luego ser suministrados al *triplestore*, así como de proporcionar los **IRI**s correspondientes a las definiciones de dominio expuestas en las ontologías del capítulo 1. Notar que, este componente se llama *registro* y no servicio porque su acceso es libre para cualquier tipo de consulta y enlace semántico; en situaciones normales, los servicios requieren algún tipo de autenticación. El *triplestore* ofrecido por **Stardog** implementa *out of the box*[4] el protocolo **SPARQL 1.1,** por tanto no se requirió implementación alguna para exponer los grafos de conocimiento almacenados mediante el lenguaje de consulta **SPARQL**.

El componente **4** está diseñado para permitir la comunicación entre los demás componentes del sistema. Consiste en una base de datos Redis en la que se publican y se consumen observaciones y eventos. Se eligió dicha base de datos por su gran eficiencia y velocidad (debido a que persiste los datos en memoria) y por las facilidades que brinda para utilizar el patrón *publish–subscribe*[5]. Este componente es indispensable para el funcionamiento del sistema, pues sin él la detección y publicación de eventos en tiempo real no fuese posible.

2.2. Sistema de anotaciones

El sistema de anotaciones se muestra en la figura 2.2. La frecuencia de generación de datos por los sensores es de 5 minutos. Dicha frecuencia puede ser ajustada mediante dos vías:

→ Solicitar una frecuencia distinta a la API consumida para obtener las observaciones de los sensores

→ Agrupar las observaciones en ventanas de tiempo de tamaño acorde a la frecuencia deseada

Destacar que este proceso ocurre cuando se solicita al registro guardar o modificar alguna entidad; por ejemplo, cuando se solicita comenzar algún servicio de analíticas este automáticamente instanciará un nuevo stream que será enviado al registro. En dicha petición el servicio de analíticas enviará toda la información necesaria para anotar dicho stream apropiadamente, incluyendo el stream del cual se deriva. En la

[4] *Un anglicismo que indica una característica o funcionalidad lista para usar*
[5] *https://en.wikipedia.org/wiki/Publish %E2 %80 %93subscribe_pattern*

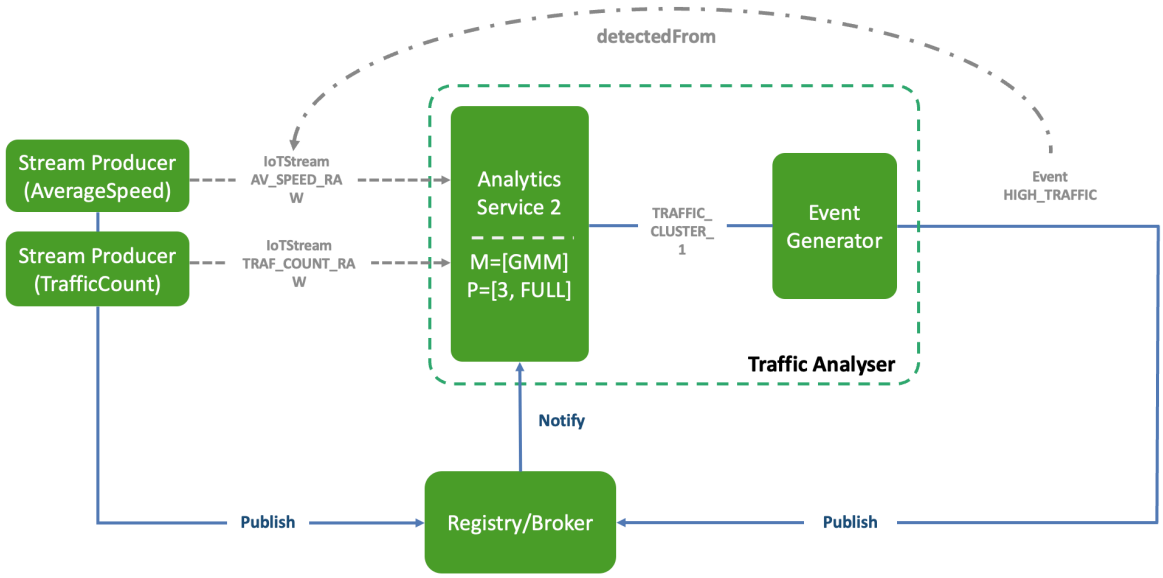


Figura 2.2: Anotaciones durante el proceso de generación de eventos de tráfico

figura 2.2 el flujo directo de los datos está representado por saetas azules, mientras que el flujo de anotaciones queda representado por saetas grises. Este proceso indirecto ocurre cuando se generan alguno de los siguientes eventos:

→ Se registra algún sensor

→ Se registra algún stream

→ Se indica que algún stream ya no recibe observaciones

→ Se inicia algún servicio de analíticas

Para el caso del sistema presentado, se decidió no incluir ni las observaciones ni los eventos en las anotaciones debido a que la frecuencia de generación de dichas entidades es muy alta y hacerlo generaría una carga significativa en el procesamiento de las consultas hechas al *triplestore* por la naturaleza no estructurada en la que almacena sus datos. Para los streams derivados de otros streams; es decir, aquellos que se producen como resultado de algún servicio de analíticas mediante la aplicación de cierto algoritmo y cierto conjunto de parámetros en el proceso de anotación se incluyen como metadatos el método y los parámetros utilizados en el componente de analíticas. Aquí **M** y **P** hacen referencia a dicho método y a dichos parámetros respectivamente. Para la instancia del sistema en particular, **M** haría referencia al

método *Gaussian Mixture Models*, que se encargaría de conformar los clústers y **P** haría referencia a la cantidad de clústers esperados, que a los efectos del sistema sería 3.

2.3. Flujo de datos

En esta sección se explicará cómo fluyen los datos entre los distintos componentes presentados en la sección 2.1. De igual manera se ahondará un poco más en el valor semántico de las acciones representadas mediante flechas en la figura 2.1.

2.3.1. Productor → Registro

Los datos entre el *productor* y el *registro* fluyen en una sola dirección; es decir, el registro se mantiene en todo momento agnóstico de cualquier productor; mientras que el productor sí debe conocer la dirección del registro para poder registrar las entidades de su dominio. Las figuras 2.3a y 2.3b muestran un sensor y un stream respectivamente enviados al registro. A nivel de sistema, se eligió el formato de serialización **JSON** para la comunicación entre todos los servicios y *turtle* para los documentos **RDF**. Debido a que los sensores obtenidos de la API de Bruselas emiten más de un dato (cantidad de autos y velocidad promedio) es posible encontrar más de un stream asociado a un mismo sensor mediante el campo *generatedBy*. Una vez obtenidos los datos de la API de Bruselas, se realizan tantas peticiones al registro como sensores y streams se obtengan. En la versión actual del componente 1, las peticiones al registro no se hacen en lotes, esto trae consigo de que si se obtienen muchos sensores y por ende streams de la API de Bruselas, se realicen muchas peticiones HTTP al registro; razón por la cual esta acotación debe considerarse para futuras versiones.

2.3.2. Productor → Bróker de mensajería

El *bróker de mensajería* al igual que el *registro*, se mantiene agnóstico del *productor* (y de todo componente en general). La comunicación entre este par esta totalmente enfocada en publicar las observaciones de los sensores en tiempo real para que cualquier otro servicio interesado en consumirlas pueda hacerlo suscribiéndose al stream correspondiente. El bróker contiene un canal suscribible por cada stream, cada uno de los cuales recibe observaciones periódicamente. Si en algún momento, un sensor deja de funcionar, los streams asociados al mismo dejan de actualizarse y seguidamente se cierra dicho stream en el registro; el efecto a nivel de bróker sería un canal que no

```json
{
  "id": "TDO-01-SPEED",
  "latitude": 0.2345,
  "longitude": -0.1234,
  "quantityKind": "speed",
  "unit": "km"
}
```


(a) Sensor serializado a JSON


```json
{
  "id": "4edb2-aa236-SPEED",
  "streamStart": "2021/11/03 16:22",
  "generatedBy": "TDO-01-SPEED",
  "feature": "speed",
  "location": {
    "latitude": 0.2345,
    "longitude": -0.1234
  }
}
```


(b) Stream serializado a JSON

Figura 2.3: Observación serializada a JSON

recibe actualizaciones. Destacar aquí que las publicaciones a los distinos canales del bróker son del tipo *fire-and-forget* lo que implica que si en algún momento se emiten observaciones a algún canal y no hay nadie suscrito al mismo, dicha observación se pierde; de igual manera el bróker solo mantiene una observación por canal en el momento de publicación y luego de ser entregada a todos los consumidores interesados en dicho canal esta desaparece.

2.3.3. Servicio de Analíticas ↔ Bróker de mensajería

El flujo de datos entre los servicios de analíticas y el *bróker de mensajería* es muy similar al mencionado en la sección 2.3.2; pero en este caso no se envían observaciones sino eventos. Los eventos están formados por un *tipo* (indicando el tipo del evento), una *marca de tiempo* y el *stream* al que pertenece. Un ejemplo de evento puede verse en la figura 2.4.

Figura 2.4: Ejemplo de evento enviado al bróker de mensajería

2.3.4. Servicio de Analíticas ↔ Consumidor

Los servicios de analíticas y el consumidor(es) se comunican bidireccionalmente. Como se había planteado en la sección 2.1, los servicios de analíticas se despliegan

bajo demanda. En este caso, cualquier consumidor puede solicitar iniciar un servicio de analíticas haciendo la petición correspondiente, este proceso se ilustra en la figura 2.5. Si la petición se gestionó correctamente, entonces dicha petición recibirá en el cuerpo de la respuesta el nombre del canal (en el bróker de mensajería) en el cual el servicio de analíticas comenzará a publicar los eventos detectados. De esta manera, el consumidor solo tiene que suscribirse al canal devuelto por el servicio de analíticas para presentar los eventos detectados al usuario.

```
POST /analytics?streamId=ffff-eee-aa45-speed HTTP/1.1
Host: iot-traffic.org
Content-Type: application/json; charset=utf-8
```

(a) Petición para comenzar a analizar el stream **ffff-eee-aa45-speed**

```
{
  "channel": "242e-fd44-9738-events"
}
```

(b) Respuesta del servicio de analíticas

Figura 2.5: Ejemplo de petición (2.5a) y respuesta (2.5b) sobre un servicio de analíticas

2.3.5. Consumidor ← Registro

El consumidor solo necesita hacer operaciones de lectura sobre el registro, lo que implica que la comunicación sea también en un solo sentido: desde el registro hacia el consumidor. Este flujo existe en esta dirección porque una vez iniciado el componente *consumidor* este necesita descubir sobre qué sensores se tiene información, así como qué streams están actualmente tramitando observaciones. De igual manera, el consumidor necesita conocer metadatos sobre dichas entidades para poblar las vistas que, una vez conformadas, contendrán información relevante para el usuario. En la figura 2.6 puede apreciarse un fragmento de la respuesta que puede recibir un consumidor al solicitar datos al registro sobre los sensores y los streams disponibles.

```
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix iot-lite: <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix qu: <http://purl.oclc.org/NET/ssnx/qu/qu#> .

<http://localhost:9987/api/v1/sensors/ARL_103_SPEED> a sosa:sensor;
  iot-lite:hasUnit "km";
  iot-lite:hasQuantityKind "speed" .

<http://localhost:9987/api/v1/sensors/GH_103_COUNT> a sosa:sensor;
  iot-lite:hasUnit "unit";
  iot-lite:hasQuantityKind "count" .

<http://localhost:9987/api/v1/sensors/VP_103_COUNT> a sosa:sensor;
  iot-lite:hasUnit "unit";
  iot-lite:hasQuantityKind "count" .
```

(a) Fragmento de respuesta al solicitar los sensores disponibles

```
@prefix iot-stream: <http://purl.org/iot/ontology/iot-stream/> .
@prefix wgs84_pos: <https://www.w3.org/2003/01/geo/wgs84_pos/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://localhost:9987/api/v1/streams/cc4ded1a-0a41-4102-aeda-c839b656e180-speed>
 a iot-stream:IotStream;
 iot-stream:windowStart "2021-11-03T15:28:00"^^xsd:dateTime;
 iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/CIN_TD1_SPEED>;
 wgs84_pos:location <http://localhost:9987/api/v1/points/0f4a4498-631f-4eb0-981a-9f755fd8b9c6> .

<http://localhost:9987/api/v1/points/0f4a4498-631f-4eb0-981a-9f755fd8b9c6> a wgs84_pos:Point;
 wgs84_pos:lat 4.39448722891203 9E0;
 wgs84_pos:long 5.08402207235609 2E1 .

<http://localhost:9987/api/v1/streams/5fc80177-cb3a-4b29-898c-e4661a1a413b-count>
 a iot-stream:IotStream;
 iot-stream:windowStart "2021-11-03T15:28:00"^^xsd:dateTime;
 iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/GH_203_COUNT>;
 wgs84_pos:location <http://localhost:9987/api/v1/points/c6449111-26ac-46d2-bde0-13037545edf5> .

<http://localhost:9987/api/v1/points/c6449111-26ac-46d2-bde0-13037545edf5> a wgs84_pos:Point;
 wgs84_pos:lat 4.40646945845846 52E0;
 wgs84_pos:long 5.08422366718352 E1 .

<http://localhost:9987/api/v1/streams/74fb5156-6f7f-411f-b7ba-80f6a079ec76-count>
 a iot-stream:IotStream;
 iot-stream:windowStart "2021-11-03T15:28:00"^^xsd:dateTime;
 iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/LOU_110_COUNT>;
 wgs84_pos:location <http://localhost:9987/api/v1/points/7d33007d-0d8e-40d5-bd9f-1adea9650418> .

<http://localhost:9987/api/v1/points/7d33007d-0d8e-40d5-bd9f-1adea9650418> a wgs84_pos:Point;
 wgs84_pos:lat 4.35470097195641 6E0;
 wgs84_pos:long 5.08356140340850 5E1 .

<http://localhost:9987/api/v1/streams/709688c3-cf6c-4a7f-87f5-b7b3876badfb-count>
 a iot-stream:IotStream;
 iot-stream:windowStart "2021-11-03T15:28:00"^^xsd:dateTime;
 iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/BAI_TD1_COUNT>;
 wgs84_pos:location <http://localhost:9987/api/v1/points/72b4c115-e327-4722-ac1e-f1a7683b450e> .

<http://localhost:9987/api/v1/points/72b4c115-e327-4722-ac1e-f1a7683b450e> a wgs84_pos:Point;
 wgs84_pos:lat 4.36302375091686 4E0;
 wgs84_pos:long 5.08281804402939 5E1 .
```

(b) Fragmento de respuesta al solicitar los stream disponibles

Figura 2.6: Fragmentos de respuestas recibidas por el consumidor al solicitar datos sobre los sensores (2.6a) y streams (2.6b) disponibles al registro

2.4. Recuperación de la información

Como se había explicado en la sección 2.1, el componente 6 es el que almacena la información referente a las entidades del sistema. Dicho componente puede ser consultado de dos maneras:

1. Mediante la **API REST** que ofrece dicho componente
2. Mediante el lenguaje de consulta **SPARQL**

La opción 1 permite consultar información específica de los sensores y de los streams

individualmente, siempre en formato **RDF**; mientras que la opción 2 ofrece una mayor flexibilidad a la hora de recuperar la información. Para el primer caso solo sería necesario hacer una petición **HTTP** utilizando el verbo **GET** hacia la ruta que se requiera; por ejemplo, si se quisiera obtener información referente a todos los sensores almacenados por el sistema la petición se haría hacia la ruta **/api/v1/sensors**, mientras que si lo que se quiere es obtener la información referente a los streams la petición se haría hacia la ruta **/api/v1/streams**. Fragmentos de respuesta a dichas peticiones fueron presentados en la subsección 2.3.5. Para el segundo caso se necesitaría elaborar una consulta **SPARQL** que exprese lo que se quiere obtener. Por ejemplo, supongamos que se quiere:

1. Conocer cuándo comenzó el stream **784fb2df-cc95-41f4-bf95-259515a06fc0-speed**
2. Obtener los **URI**s de los streams generados por el sensor **ARL_103**

Para la primer situación, podemos construir una consulta como la que se muestra en la figura 2.7a mientras que para la segunda situación, la consulta mostrada en la figura 2.7b mostrará el resultado esperado.

```
PREFIX iot-stream: <http://purl.org/iot/ontology/iot-stream/>

SELECT ?started
WHERE {
    <http://localhost:9987/api/v1/streams/784fb2df-cc95-41f4-bf95-259515a06fc0-speed> iot-stream:windowStart ?started.
}

```

(a) Consulta para obtener la fecha de inicio del stream **784fb2df-cc95-41f4-bf95-259515a06fc0-speed**

```
PREFIX iot-stream: <http://purl.org/iot/ontology/iot-stream/>
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT ?stream
WHERE {
    { ?stream rdf:type iot-stream:IotStream;
              iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/ARL_103_SPEED> }
    UNION
    { ?stream rdf:type iot-stream:IotStream;
              iot-stream:generatedBy <http://localhost:9987/api/v1/sensors/ARL_103_COUNT> }
}

```

(b) Consulta para obtener los streams generados por el sensor **ARL_103**

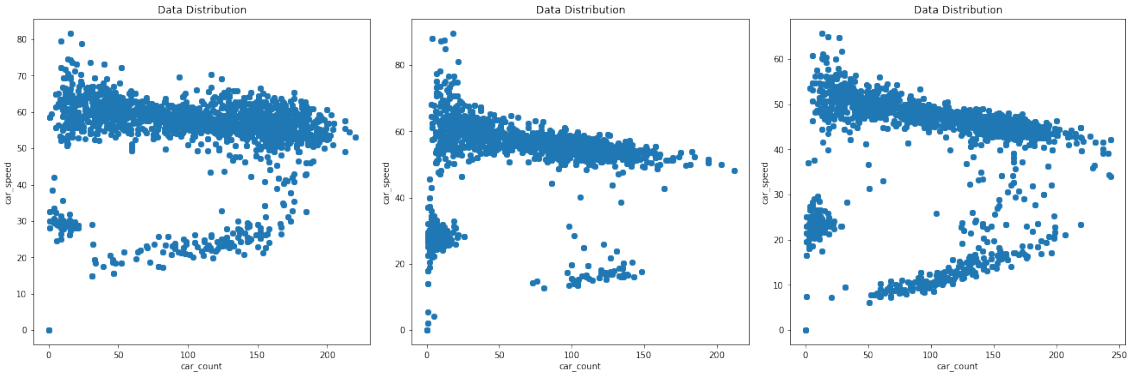
Figura 2.7: Algunas consultas que se pueden realizar para recuperar información específica sobre los streams

En ambas consultas se utilizó la cláusula **PREFIX** para simplificar los predicados

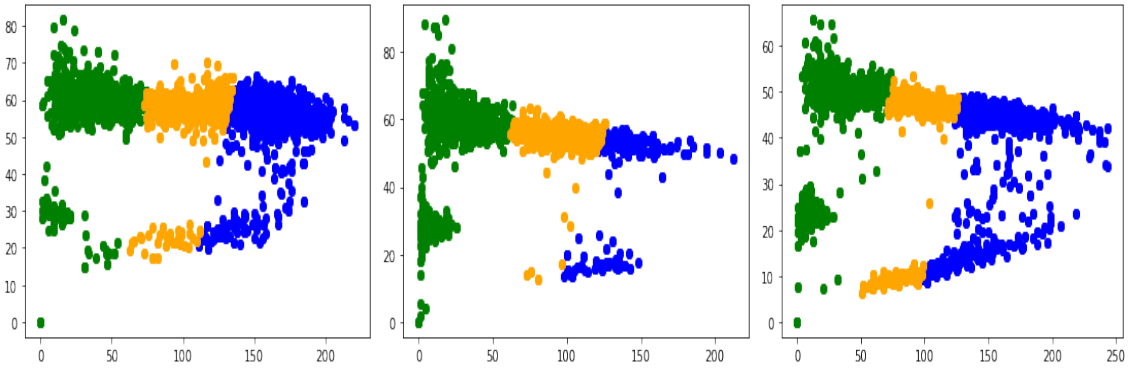
de las cláusulas **WHERE**; de no hacerlo, hay que especificar el **URI** completo a la propiedad o predicado que se quiera incluir en la consulta.

2.5. Analíticas

El proceso de generación de eventos consta de dos partes fundamentales: la primera, que se realiza una sola vez y consiste en construir los clústers de tráfico a partir de datos históricos; y la segunda que también se realiza una sola vez pero se utiliza cada vez que se va a analizar un stream y consiste en construir el clasificador para, dada una observación de tráfico, discriminar entre las etiquetas: *ALTO*, *NORMAL* y *BAJO* de acuerdo al nivel de tráfico que exista en el intervalo de tiempo fijado para la aplicación. Los datos históricos para construir los clústers se recolectaron durante dos semanas, para ello se escribió un script que se adjuntará en el repositorio correspondiente.



(a) Observaciones de los sensores **ARL_203**, **CIN_TD1**, **ROG_TD1**



(b) Algoritmo **GMM** aplicado a los tres conjuntos de observaciones anteriores

Figura 2.8: Conjuntos de datos y clústers formados a partir de dichos conjuntos

En la figura 2.8a pueden observarse los datos recopilados de los sensores **ARL_203**,

CIN_TD1, **ROG_TD1** durante el período de tiempo especificado anteriormente. Por el eje de las abscisas se presenta la cantidad de vehículos medidos por el sensor en unidades, mientras que por el eje de las ordenadas se encuentra la velocidad promedio de dichos vehículos en *km*. Por otro lado, en la figura 2.8b se muestran los clústers encontrados por el algoritmo **GMM**; en verde se destaca el clúster que representa el estado de tráfico *BAJO* y en naranja y azul los estados de tráfico *NORMAL* y *ALTO* respectivamente.

Para el proceso de construcción del clasificador se decidió utilizar todas las observaciones obtenidas del sensor **ROG_TD1** como conjunto de entrenamiento; mientras que para el conjunto de evaluación se decidió utilizar todas las observaciones del sensor **ARL_203**. Una vez establecidos el conjunto de entrenamiento y el conjunto de evaluación, se comenzó a constuir el clasificador como tal, experimentando con distintos métodos. En el cuadro 2.1 se muestran los resultados obtenidos respecto a la precisión de los clasificadores mientras que en el cuadro 2.2 se encuentran las matrices de confusión de dichos clasificadores. Como puede apreciarse, de manera general la precisión de los clasificadores obtenidos es bastante buena, en especial, el clasificador obtenido utilizando regresión logística fue el que presentó el mejor resultado en esta categoría. Debido a que este método en general se utiliza para problemas de clasificación binaria, hubo que especificar en el parámetro *multi_class* el valor *multinomial* para utilizar la generalización de este método para problemas multiclase: *la regresión logística multinomial*. En adición hubo que especificar una mayor cantidad de iteraciones debido a que las que utilizaba el método por defecto (100) no permitían alcanzar la convergencia. Por otro lado, si se observan las matrices de confusión de los dos últimos clasificadores se puede ver que el clasificador obtenido utilizando el soporte de regresión vectorial es mejor clasificador para la clase *high* que el obtenido utilizando regresión logística; pero de manera general ambos demostraron ser lo suficientemente buenos. En consecuencia, para construir el componente de analíticas se eligió el último clasificador.

Método	Parámetros	Precisión
k-nearest neighbors	$k \leftarrow 3$	95,70 %
k-nearest neighbors	$k \leftarrow 5, weights \leftarrow distance$	94,88 %
k-nearest neighbors	$k \leftarrow 7$	93,63 %
random forests	*Parámetros por defecto*	95,90 %
support vector regression	*Parámetros por defecto*	96,82 %
logistic regression	$multiclass \leftarrow multinomial, maxiter \leftarrow 110$	97,99 %

Cuadro 2.1: Eficacia del los distintos clasificadores probados

KNN, $k \leftarrow 3$

	high	*low*	*normal*
high	1931	0	0
low	0	1621	16
normal	133	61	1123

KNN, $k \leftarrow 5, weights \leftarrow distance$

	high	*low*	*normal*
high	1931	0	0
low	0	1621	16
normal	218	16	1083

,

KNN, $k \leftarrow 3$

	high	*low*	*normal*
high	1931	0	0
low	0	1621	16
normal	133	61	1123

Random Forests

	high	*low*	*normal*
high	1922	0	9
low	0	1637	0
normal	165	26	1126

,

SVR

	high	*low*	*normal*
high	1931	0	0
low	0	1637	0
normal	81	74	1162

Logistic regression

	high	*low*	*normal*
high	1910	0	21
low	0	1637	0
normal	8	69	1240

Cuadro 2.2: Matrices de confusión para los clasificadores mostrados en 2.1

Bibliografía

[1] ACTIVAGE. 2021. About ACTIVAGE. (may 2021). Retrieved may 11, 2021 from https://www.activageproject.eu/activage-project/#About-ACTIVAGE

[2] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. 2021. EP-SPARQL: a unified language for event processing and stream reasoning. In *the 20th international conference*. ACM Press, Hyderabad, India, 635. https://doi.org/10.1145/1963405.1963495

[3] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. 2010. C-SPARQL: A CONTINUOUS QUERY LANGUAGE FOR RDF DATA STREAMS. *International Journal of Semantic Computing* 04, 01 (2010), 3–25. https://doi.org/10.1142/S1793351X10000936

[4] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. 2014. Terse RDF Triple Language. (feb 2014). Retrieved June 9, 2021 from https://www.w3.org/TR/2014/REC-turtle-20140225/

[5] Tim Berners-Lee. 2006. Linked Data. (mar 2006). Retrieved June 10, 2021 from https://www.w3.org/DesignIssues/LinkedData

[6] Tim Berners-Lee. 2007. Semantic Web: Linked Data on the Web. (mar 2007). Retrieved June 10, 2021 from https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)

[7] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web : a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American* 284, 5 (2001), 34–43.

[8] André Bondi. 2000. Characteristics of Scalability and Their Impact on Performance. *Proceedings Second International Workshop on Software and Performance WOSP 2000*, 195–203. https://doi.org/10.1145/350391.350432

[9] Eric A. Brewer. 2000. Towards robust distributed systems. https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[10] Dan Brickley. 2006. Introduction to Bayesian Statistics. (mar 2006). Retrieved June 8, 2021 from https://www.w3.org/2003/01/geo/

[11] Dan Brickley, R.V. Guha, and Brian McBride. 2014. RDF Schema 1.1. (2014). https://www.w3.org/TR/rdf-schema/

[12] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair JG Gray. 2010. *Enabling ontology-based access to streaming data sources*. Springer, 96–111.

[13] Andy Carvin. 2006. DDN Articles - Tim Berners-Lee: Weaving a Semantic Web. (2006-04-21 2006). https://web.archive.org/web/20060421130339/http://www.digitaldivide.net/articles/view.php?ArticleID=20

[14] Janet Daly, Marie-Claire Forgue, and Yasuyuki Hirakawa. 2007. El W3C establece un puente entre HTML/Microformatos y la Web Semántica. (mar 2007). Retrieved June 9, 2021 from https://www.w3c.es/Prensa/2007/nota070911_grddl.html

[15] Daniele Dell'Aglio and Emanuele Della Valle. 2021. C-SPARQL Stream Reasoning. (2021). http://streamreasoning.org/resources/c-sparql

[16] Open Data DK. 2015. ¿Qué es Open Data DK? (mar 2015). Retrieved May 15, 2021 from https://www.opendata.dk/hvad-er-open-data-dk

[17] Richard Duncan. 2002. An Overview of Different Authentication Methods and Protocols. (2002), 8.

[18] F. Eliassen and J. Veijalainen. 1988. A functional approach to information system interoperability. (1988).

[19] Tarek Elsaleh, Shirin Enshaeifar, Roonak Rezvani, Sahr Thomas Acton, Valentinas Janeiko, and Maria Bermudez-Edo. 2020. IoT-Stream: A Lightweight Ontology for Internet of Things Data Streams and Its Use with Data Analytics and Event Detection Services. *Sensors (Basel, Switzerland)* 20, 4 (2020). https://doi.org/10.3390/s20040953

[20] Dieter Fensel. 2004. *Ontologies: a Silver Bullet for Knowledge Management and Electronic Commerce*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[21] Susel Fernandez, Rafik Hadfi, Takayuki Ito, Ivan Marsa-Maestre, and Juan R. Velasco. 2016. Ontology-Based Architecture for Intelligent Transportation Systems Using a Traffic Sensor Network. *Sensors* 16, 8 (2016). https://doi.org/10.3390/s16081287

[22] Christine Golbreich and Evan K. Wallace. 2012. OWL 2 Web Ontology Language Primer (Second Edition). (dec 2012). Retrieved June 9, 2021 from http://www.w3.org/TR/2012/REC-owl2-new-features-20121211/

[23] Sérgio Gorender and Ícaro Silva. 2013. An ontology for a fault tolerant traffic information system.

[24] Network Working Group. 2005. Internationalized Resource Identifiers (IRIs). (jan 2005). Retrieved June 9, 2021 from https://datatracker.ietf.org/doc/html/rfc3987

[25] OWL Working Group. 2013. Web Ontology Language (OWL). (mar 2013). Retrieved June 9, 2021 from https://www.w3.org/OWL/

[26] Semantic Web Deployment Working Group. 2009. Simple Knowledge Organization System (SKOS). (aug 2009). Retrieved June 9, 2021 from https://www.w3.org/2001/sw/wiki/SKOS

[27] Working group of the SysML 1.2 Revision Task Force (RTF) and W3C Semantic Sensor Network Incubator Group. 2010. Introduction to Bayesian Statistics. (september 2010). Retrieved June 8, 2021 from https://www.w3.org/2005/Incubator/ssn/wiki/QU_Ontology

[28] Nicola Guarino. 1998. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. Vol. 46. IOS press.

[29] A. Gupta, R. Christie, and Prof. R. Manjula. 2017. Scalability in Internet of Things : Features , Techniques and Research Challenges.

[30] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. 2008. The internet of things in an enterprise context. In *Future Internet Symposium*. Springer, 14–28.

[31] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. (mar 2004). Retrieved June 10, 2021 from https://www.w3.org/Submission/SWRL/

[32] Michael Hülsen, J Marius Zöllner, and Christian Weiss. 2011. Traffic intersection situation description ontology for advanced driver assistance. IEEE, 993–999.

[33] ISO. 2021. ISO/IEC 2382-1:1993 Information Technology – Vocabulary – Part 1: Fundamental terms. International Organization for Standardization (ISO). (2021). Retrieved June 5, 2021 from https://translate.google.com/website?sl=auto&tl=es&ajax=1&u=http://www.iso.org/iso/catalogue_detail.htm?csnumber%3D7229

[34] Krzysztof Janowicz, Armin Haller, Simon J. D. Cox, Danh Le Phuoc, and Maxime Lefrancois. 2019. SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators. *Journal of Web Semantics* 56 (2019), 1–10. https://doi.org/10.1016/j.websem.2018.06.003

[35] Igor Jurisica, John Mylopoulos, and Eric Yu. 1999. Using ontologies for knowledge management: An information systems perspective, Vol. 36. 482–496.

[36] Michael Kifer and Harold Boley. 2013. RIF Overview (Second Edition). (2013). https://www.w3.org/TR/rif-overview/

[37] Jussi Kiljander, Alfredo D'elia, Francesco Morandi, Pasi Hyttinen, Janne Takalo-Mattila, Arto Ylisaukko-Oja, Juha-Pekka Soininen, and Tullio Salmon Cinotti. 2014. Semantic interoperability architecture for pervasive computing and internet of things. *IEEE access* 2 (2014), 856–873.

[38] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. 2019. Internet of Things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data* 6, 1 (2019), 111. https://doi.org/10.1186/s40537-019-0268-2

[39] Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. 2004. Query Languages and Data Models for Database Sequences and Data Streams. 492–503. https://doi.org/10.1016/B978-012088469-8/50045-0

[40] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. 2011. *A Native and Adaptive Approach for Unified Processing of Linked*

Streams and Linked Data. Vol. 7031. Springer Berlin Heidelberg, Berlin, Heidelberg, 370–388.

[41] Chris Little and Simon Cox. 2020. OWL-Time - Summary | NCBO BioPortal. (2020). https://bioportal.bioontology.org/ontologies/TIME/?p=summary

[42] Simple A LLC. 2021. What Is the Semantic Web? (mar 2021). Retrieved June 9, 2021 from https://simplea.com/Articles/what-is-the-semantic-web

[43] Imperial College London. 2021. London Air Quality Network. (2021). https://www.londonair.org.uk/LondonAir/General/about.aspx

[44] David Maier, Jin Li, Peter Tucker, Kristin Tufte, and Vassilis Papadimos. 2005. Semantics of Data Streams and Operators. In *Database Theory - ICDT 2005*, Thomas Eiter and Leonid Libkin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 37–52.

[45] Alessandro Margara and Tilmann Rabl. 2019. *Definition of Data Streams*. Springer International Publishing, Cham, 648–652. https://doi.org/10.1007/978-3-319-77525-8_188

[46] Matthew B. Jones Mark Schildhauer. 2016. OBOE: the Extensible Observation Ontology, version 1.1. (2016 2016).

[47] Philippe Morignot and Fawzi Nashashibi. 2012. An ontology-based approach to relax traffic regulation for autonomous vehicle assistance. *arXiv:1212.0768 [cs]* (2012).

[48] Boris Motik, Bernardo Cuenca, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. OWL 2 Web Ontology Language Profiles (Second Edition). (dec 2012). Retrieved June 9, 2021 from https://www.w3.org/TR/owl2-profiles/

[49] Mozilla and individual contributors. 2021. HTTP. (mar 2021). Retrieved June 9, 2021 from https://developer.mozilla.org/es/docs/Web/HTTP

[50] Mozilla and individual contributors. 2021. HTTP. (mar 2021). Retrieved June 9, 2021 from https://developer.mozilla.org/es/docs/Glossary/Unicode

[51] Christian Neuenstadt. 2014. OBDA Stream Access Combined with Safe First-Order TemporalReasoning. *Institute for Software, Technology & Systems* (2014).

[52] Carlos Pereira and Ana Aguiar. 2014. Towards Efficient Mobile M2M Communications: Survey and Open Challenges. *Sensors* 14, 10 (2014), 19582–19608. https://doi.org/10.3390/s141019582

[53] J Radatz, A Geraci, and F Katki. 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std. 610.12-1990. *Computer Society of the IEEE* (1990).

[54] Rodrigo Righi, Márcio Gomes, and Cristiano André da Costa. 2014. Internet of things scalability: Analyzing the bottlenecks and proposing alternatives. *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops* 2015. https://doi.org/10.1109/ICUMT.2014.7002114

[55] Hajo Rijgersberg, Mark van Assem, and Jan Top. 2013. Ontology of units of measure and related concepts. *Semantic Web* 4, 1 (2013), 3–13. https://doi.org/10.3233/SW-2012-0069

[56] Mikko Rinne and Esko Nuutila. 2012. INSTANS: High-Performance Event Processing with Standard RDF and SPARQL *(ISWC-PD'12)*. CEUR-WS.org, Aachen, DEU, 101–104.

[57] Toby Segaran, Colin Evans, and Jamie Taylor. 2009. *Programming the Semantic Web: Build Flexible Applications with Graph Data*. .º'Reilly Media, Inc.".

[58] Donghee Shin. 2014. A socio-technical framework for Internet-of-Things design: A human-centered design for the Internet of Things. *Telematics and Informatics* 31, 4 (2014), 519–531.

[59] OBDA Systems S.R.L. 2021. OBDA: A three-level architecture. (mar 2021). Retrieved June 4, 2021 from https://www.obdasystems.com/obda

[60] Sasu Tarkoma and Artem Katasonov. 2011. Internet of things strategic research agenda (IoT–SRA). *Finnish Strategic Centre for Science, Technology, and Innovation: For Information and Communications (ICT) Services, Businesses, and Technologies, Finland* (2011).

[61] Trialog. 2021. IoT systems and interoperability. (2021). Retrieved June 5, 2021 from https://www.trialog.com/en/iot-systems-and-interoperability/

[62] Gertjan Van Heijst, A Th Schreiber, and Bob J Wielinga. 1997. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies* 46, 2-3 (1997), 183–292.

[63] W3C. 2013. SPARQL Query Language for RDF. (mar 2013). Retrieved June 9, 2021 from https://www.w3.org/TR/rdf-sparql-query/

[64] W3C. 2015. What is Linked Data? (mar 2015). Retrieved June 9, 2021 from https://www.w3.org/standards/semanticweb/data

[65] Michael Weyrich and Christof Ebert. 2016. Reference Architectures for the Internet of Things. *IEEE Software* 33, 1 (2016), 112–116. https://doi.org/10.1109/MS.2016.20

[66] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. 2015. The Internet of Things—A survey of topics and trends. *Information systems frontiers* 17, 2 (2015), 261–274.

[67] Sherali Zeadally and Oladayo Bello. 2021. Harnessing the power of Internet of Things based connectivity to improve healthcare. *Internet of Things* 14 (2021), 100074. https://doi.org/10.1016/j.iot.2019.100074

[68] Lihua Zhao, Ryutaro Ichise, Seiichi Mita, and Yutaka Sasaki. 2015. Ontologies for Advanced Driver Assistance Systems.