UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

Título Modelo distribuido para el Análisis Visual de datos

Autor: Carlos Gilberto Anoceto González

Tutores: Dr. C. Carlos Pérez Risquet

Lic. David Rodríguez Mollineda

Santa Clara , junio, 2018

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Laboratorio

RESUMEN

Los sistemas distribuidos son muy usados en la actualidad debido al aprovechamiento de los recursos y la potencia de cálculo de varias computadoras enlazadas. Por ello son muy prácticos en situaciones que requieren un gran procesamiento o es necesario el manejo de grandes volúmenes de datos. Usando técnicas del campo de Visual Analytics se desarrolló un modelo para el análisis visual de datos (Mollineda, 2017) *el cual permite manejar un gran volumen de datos heterogéneos de manera intuitiva y sencilla pero no aprovecha las potencialidades de los sistemas distribuidos para esta tarea. Por tanto, el objetivo general de este trabajo de diploma consiste en desarrollar un modelo distribuido para el análisis visual de datos basado en el existente que si haga uso de las potencialidades de la computación distribuida. Los principales resultados son: (1) la creación de un modelo distribuido para el análisis visual de datos y (2) la incorporación de este a la herramienta previamente desarrollada.*

Palabras Clave: *Computación distribuida, Visual Analytics, datos heterogéneos, modelo distribuido.*

ABSTRACT

Distributed systems are very used due the exploit of resources and computer power of several linked computers. Therefore, are very practical in situations that require a huge amount of processing power or handle large data volume. Using Visual Analytics' techniques was developed a model for the data visual analysis (Mollineda, 2017) *which one allows handle a large volume of heterogeneous data in easy and intuitive way, but don't take advantage of the potential of distributed systems for this task. Therefore, the main goal of this research is develop a distributed model for the data visual analysis based in the existent one but does use distributed computation advantages. The main results are: (1) the creation of a distributed model for data visual analysis and (2) insert this in the previously developed tool.*

Keywords: *Distributed Computation, Visual Analytics, heterogeneous data, distributed model.*

Índice

Introducción

La arquitectura cliente-servidor es una rama de la computación bastante amplia que permite la descentralización del procesamiento y los recursos. Hace posible que en muchas ocasiones un servidor esté dedicado a una tarea específica, provocando una mayor eficiencia del mismo. El sistema mencionado se basa en la existencia de un recurso cliente que solicita un servicio a otro recurso servidor a través de un medio de comunicación, este servicio puede ser un acceso a un banco de información, a un determinado hardware (impresora, etc.), una solicitud de ejecución de algún programa, etc. El carácter de la solicitud está condicionado por el tipo de servidor al que se le realice, algunos de estos servidores son:

- Servidores de archivo: Sistemas de intercambio de archivos, donde existen dos esquemas básicos de localización: los que utilizan un servidor central (Ejemplo. Napster y eDonkey), y los que utilizan un modelo totalmente distribuido, en el cual todos los nodos son clientes y servidores a la vez (Ej. Gnutella) (Guillermina, Galache and Herrera, 2014). El sistema de archivos como parte del sistema operativo proporciona al usuario una interfaz sencilla, amigable y organizada, que le permite almacenar, organizar y proteger su información. Los mecanismos de protección son especialmente importantes para sistemas multiusuarios (Edmundo, Valencia and Feliciano, 2010).
- Servidor Web: es un programa que utiliza el protocolo de transferencia de híper texto ***HTTP*** (por las siglas en inglés de *Hypertext Transfer Protocol*) para proporcionar los archivos que forman páginas Web a los usuarios, en respuesta a sus solicitudes.
- Servidores de correo: Son servidores que permiten la gestión de *emails*[1]. Es un servicio de red que posibilita a los usuarios enviar y recibir mensajes mediante sistemas de comunicación electrónica. Por medio de mensajes de correo electrónico se puede enviar, no solamente texto, sino todo tipo de documentos digitales dependiendo del sistema que se use.

[1] Correos electrónicos

- Servidores de aplicaciones: Se denomina servidor de aplicaciones a un servidor en la red que ejecuta determinadas aplicaciones. Generalmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras clientes.
- Servidores de impresión: Un servidor de impresión es un software que permite que los *PCs*[2] o cualquier dispositivo que permita el trabajo con impresoras —como teléfonos inteligentes de una red local— pueda hacer uso de las impresoras de la red de una forma eficaz, ya que centraliza y facilita la gestión de las tareas de impresión. Para permitir la comunicación de los clientes con la impresora el servidor se apoya en *drivers* especializados.
- Servidores de bases de datos: Es un servidor que brinda soporte a bases de datos, Una base de datos de un sistema informático (**SI**) es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del **SI** y de sus interrelaciones. Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos (Pérez Mora *et al.*, 2005). Por otro lado (Ruiz and Vargas, 2012) lo define como un conjunto de información organizada con determinadas técnicas de acceso y tratamiento que se almacena en los dispositivos de memoria de un ordenador", esta información se convierte en un banco de datos dedicado a un tema específico o a varias disciplinas y/o recursos.

Según (Hernández, 1997) existen varias propuestas de distribución de tareas: el cliente procesa solamente el despliegue de información, el cliente procesa el despliegue de información y participa en parte del proceso y el cliente maneja el despliegue, todo el proceso, y accede a los datos del servidor. A lo largo de los años han surgido muchas variantes de implementación, basadas en estas propuestas se pueden encontrar varias clasificaciones:

- Representación remota: La lógica de la aplicación se encuentra en el servidor, el cliente solo visualiza la información al usuario.
- Representación distribuida: La interacción con el usuario se realiza en el servidor, el cliente hace de pasarela entre el usuario y el servidor.

[2] Computadoras personales

- Gestión remota de datos: El cliente realiza la interacción con el usuario y ejecuta la aplicación y el servidor es quien maneja los datos.
- Base de datos distribuidas: El cliente realiza la interacción con el usuario, ejecuta la aplicación, debe conocer la topología de la red, así como la disposición y ubicación de los datos. Se delega parte de la gestión de la base de datos al cliente.
- Lógica distribuida: El cliente se encarga de la interacción con el usuario y de algunas funciones triviales de la aplicación, —controles de rango de campos, campos obligatorios, etc.— mientras que el resto, junto con la base de datos, están en el servidor.
- Cliente servidor a tres niveles: El cliente se encarga de la interacción con el usuario, el servidor de la lógica de aplicación y la base de datos pueden estar en otro servidor.
- Niveles de una aplicación Web: El nivel de interfaz de usuario está compuesto por las páginas HTML que el usuario solicita a un servidor Web y que visualiza en un cliente Web, usualmente un navegador. El nivel de lógica de negocio está compuesto por los módulos que implementan la lógica de la aplicación y que se ejecutan en un servidor de aplicaciones. El nivel de datos está compuesto por los datos gestionados por un sistema de gestión de bases de datos (servidor de datos), que maneja la aplicación web.

Según (Coulouris, 1994) un sistema distribuido consiste en un conjunto de computadoras autónomas conectadas por una red y con soporte de software distribuido. Permite que las computadoras coordinen sus actividades y compartan recursos de hardware, software y datos, de manera tal que el usuario percibe una única facilidad de cómputo integrada, aunque esta pueda estar implementada por varias máquinas en distintas ubicaciones.

Dentro de este contexto y por necesidad surge la computación distribuida según (Lafuente, 2014): Los sistemas distribuidos suponen un paso más en la evolución de los sistemas informáticos, entendidos desde el punto de vista de las necesidades que las aplicaciones plantean y las posibilidades que la tecnología ofrece.

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora o ***mainframe***[3], a la vez que se mantiene la flexibilidad de trabajar en múltiples problemas más pequeños. Es una opción más barata para las instituciones que los

[3] Computadora central grande, potente y costosa

clústeres, puesto que permite usar los mismos ordenadores conectados como poder de cálculo y almacenamiento. Es también una alternativa que provee un sistema más escalable que favorece la ampliación o disminución de sus recursos sin dificultad. Debido a su alto nivel de abstracción cada nodo o recurso puede ser de distinta índole: un PC, un clúster, o incluso un teléfono móvil.

La programación distribuida es especialmente útil en campos como la física y la bioinformática que utilizan aplicaciones con un alto requerimiento de poder de cálculo. Permite la ejecución de un mismo algoritmo en varios o en todos sus nodos mediante *middlewares* como ***RPC***[4] (por las siglas en inglés de *Remote Procedure Call***)** o **SOAP**[5] (por las siglas en inglés de *Simple Object Access Protocol*), y, a su vez, cada nodo puede tener la capacidad de ejecutarlos de forma concurrente a nivel de procesador, lo que la convierte en una herramienta especialmente poderosa.

Como se evidenció anteriormente una de las ventajas de la computación distribuida es la capacidad de manejar grandes volúmenes de datos, los cuales son muy comunes en las aplicaciones actuales. La existencia de estas grandes masas de datos complejiza el trabajo de los científicos porque el cerebro humano no es capaz de manejarlos eficientemente (Keim Daniel and Mansmann, 2010). Para dar una solución a este problema surge el campo *Visual Analytics*[6] que según (Kerren *et al.*, 2008) se enfoca en manejar el masivo, heterogéneo y dinámico volumen de información mediante la integración del juicio humano. Para ello usa representaciones visuales y técnicas de interacción en el proceso de análisis, ya que en las aplicaciones actuales los datos se producen a un ritmo sin precedentes, y aunque la capacidad de recolectar nuevos datos y almacenarlos se incrementa rápidamente, la habilidad de analizar ese volumen de datos lo hace lentamente. Utilizando esta técnica se realizó en la Universidad Central "Marta Abreu" de Las Villas una tesis de grado, donde (Mollineda, 2017) plantea un modelo que utiliza la clasificación BCS[7] (System, 2013), un sistema de clasificación dentro de la bioinformática propuesto en el año 1995 por Gordon Amidon.

La implementación realizada por (Mollineda, 2017) tiene carácter local, es decir, el software desarrollado solo utiliza los recursos locales de un PC, por lo que no hace un uso correcto y justo

[4] Control de procesos remotos

[5] Protocolo simple de acceso a objetos

[6] Análisis Visual

[7] Sistema de Clasificación Biofarmacéutica

de las tecnologías actuales, como los recursos de red, que presentarían una mejora considerable en cuanto eficiencia y alcance. Lo que puede resumirse como:

Localidad de todos los elementos del modelo, tanto el procesamiento como el almacenamiento de datos se trata a nivel de computadora personal, lo que limita el poder de procesamiento del modelo.

Por lo que el **objetivo general** de la investigación consiste en:

Ampliar el modelo mediante un ambiente de red distribuido para aumentar sus capacidades de cálculo.

Este se desglosa en los siguientes **objetivos específicos**:

1. Identificar la arquitectura de red más apropiada para el sistema.
2. Implementar una versión prototipo del modelo distribuido propuesto.
3. Validar el modelo distribuido planteado.

Las **preguntas de investigación** planteadas sobre la base de estos objetivos son:

- ¿Cuál arquitectura o arquitecturas de red resultan apropiadas para implementar el modelo BCS en un ambiente distribuido?
- ¿Cuál sería el lenguaje de programación y las herramientas que mejor se ajusten para la implementación del nuevo sistema?
- ¿Qué técnicas de validación pudieran aplicarse?

Esta investigación posee un valor práctico y económico que se deriva del interés por lograr un modelo con más potencia de procesamiento que logre disminuir los tiempos de simulación, sin que signifique un gasto de dinero considerablemente alto.

El informe de investigación está divido en tres capítulos. El CAPÍTULO 1 consiste en un breve análisis del modelo propuesto por (Mollineda, 2017) para identificar sus principales características, algunas de las posibles arquitecturas sobre las que se podría implementar el sistema, así como un bosquejo sobre la programación paralela. En el CAPÍTULO 2 se discute la propuesta del modelo, así como algunas consideraciones a tener en cuenta a la hora de implementarlo. En el CAPÍTULO 3 se presentan los resultados obtenidos a partir de la implementación de la propuesta y su validación. Finalmente se exponen las conclusiones

obtenidas, las recomendaciones para la continuación de este trabajo en el futuro y las referencias bibliográficas que fueron empleadas.

CAPÍTULO 1. Modelo distribuido para el análisis visual de datos

El análisis de grandes volúmenes de datos para extraer información, identificar patrones interesantes, realizar clasificación, etc., es un problema desafiante desde el punto de vista computacional. A pesar de que cada vez las computadoras personales modernas tienen un poder de cálculo más potente y eficiente aún no es suficiente para los grandes volúmenes de datos con los que se trabaja en ramas de la ciencia como la biofarmacéutica, lo que lleva a la tarea de extender el modelo a un ambiente de red. Para ello primero es necesario analizar el modelo propuesto por (Mollineda, 2017), para determinar su principio de funcionamiento básico, sus formas de interacción, así como su flujo de trabajo, y, de esta forma, presentar una extensión del modelo.

1.1 Análisis del modelo

En (Mollineda, 2017) se plantea un modelo para el análisis visual de datos con determinadas características, las cuales se irán mencionando a medida que se progrese en el actual epígrafe. Para mayor documentación dirigirse a (Mollineda, 2017).

Estructura General

Según su autor (Mollineda, 2017), la estructura general del modelo consiste en un conjunto de componentes interconectados, básicamente un grafo dirigido. El modelo estaría dado por un grafo dirigido G_M compuesto por un conjunto de componentes analíticos C_A, haciendo el papel de vértices, y una serie de conexiones entre estos E_C, que determinan la forma en que fluyen los datos a través de los componentes interconectados.

En el modelo se permite la existencia de ciclos puesto que cada componente C_A tiene una función predeterminada por el propio usuario. La existencia de un ciclo significa que los datos vuelven a ser procesados pero luego de haber sido modificados por otros C_A dentro del bucle, la excepción es la conexión de un componente consigo mismo. En la **Figura 1.1** se ilustra el aspecto general de un flujo de análisis creado de acuerdo a estas especificaciones según (Mollineda, 2017).

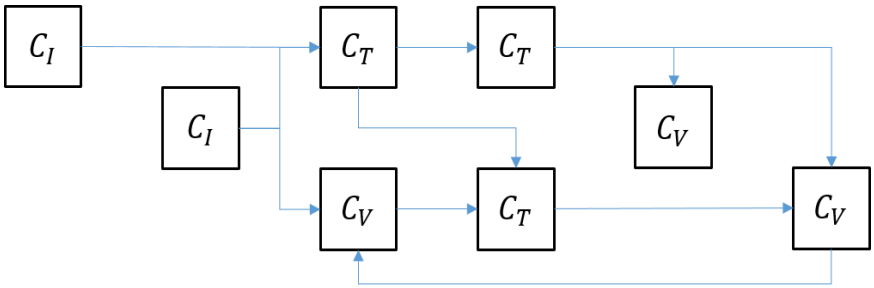

Figura 1.1 Estructura general de un flujo de análisis de acuerdo al modelo

Ejecución del flujo de análisis

De forma general, cada componente $c_i \in C_A$ tiene asociada información que conforma su estado actual s_i. Este determina su comportamiento frente a las entradas que reciba el componente a partir de otros componentes que se encuentren conectados a él, o de la interacción recibida directamente por parte del analista en su papel como usuario. Este estado incluye, además de características propias al componente, su estado de activación actual, o sea, si reacciona o no ante las entradas recibidas (Mollineda, 2017). En la extensión del modelo propuesto es necesario incluir, en el estado del componente, su localización, así como su disponibilidad a ser transferido de localización por algún criterio seleccionado.

Cada componente C_A tiene una función asociada, la cual es la encargada de modificar la vista de datos que se le suministra, por lo que pudiera definirse como:

$$B_i(I_i, s_i, U) = \begin{cases} B_{Si}(s_i), I_i = \emptyset \wedge U = U_{exe} \\ B_{Ii}(I_i, s_i), I \neq \emptyset \wedge U = \emptyset \\ B_{Ui}(I_i, s_i, U), U \neq \emptyset \end{cases}$$

Donde la entrada recibida por el componente es (I_i), (s_i) es su estado actual y (U) es la interacción del usuario.

La **Figura 1.2** ilustra el comportamiento general de la ejecución de un componente frente a los diferentes tipos de entrada e interacción, planteado por (Mollineda, 2017).

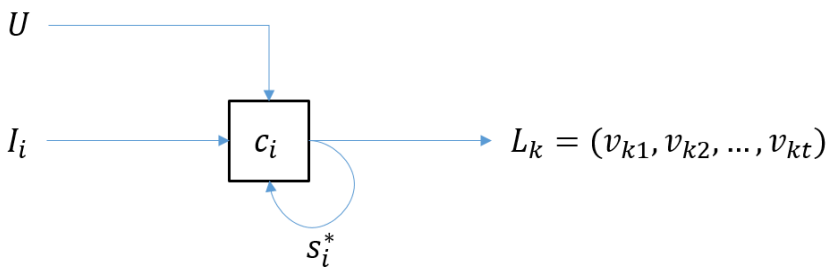

Figura 1.2 Comportamiento de la ejecución de un componente

El modelo planteado permite una alta interacción del usuario con el mismo, esto posibilita realizar pruebas más personalizadas que integren la capacidad de manejo de grandes volúmenes de datos de una PC con la habilidad de razonamiento de un especialista. Un ejemplo de estas capacidades es la posibilidad de activar o desactivar una conexión o un nodo en cualquier momento, lo que le da cierto control sobre el flujo de datos, como se pude ver en la **Figura 1.3** y la **Figura 1.4** tomadas de (Mollineda, 2017).

Como parte de la nueva propuesta se plantea que el modelo sea capaz de realizar un balance de cargas con los servidores que estén ocupados con componentes deshabilitados, es decir, el modelo debe ser capaz de ajustar la distribución del flujo de ejecución sin modificar la lógica del mismo.

La **Figura 1.3** refleja los efectos de la inhabilitación de una conexión, mientras que la **Figura 1.4** muestra los resultados de inhabilitar un componente. Los componentes marcados con el símbolo en verde son aquellos que se ejecutan satisfactoriamente. Aquellos en rojo son los que no se ejecutan a consecuencia de que el flujo de la ejecución nunca los alcanza (Mollineda, 2017).

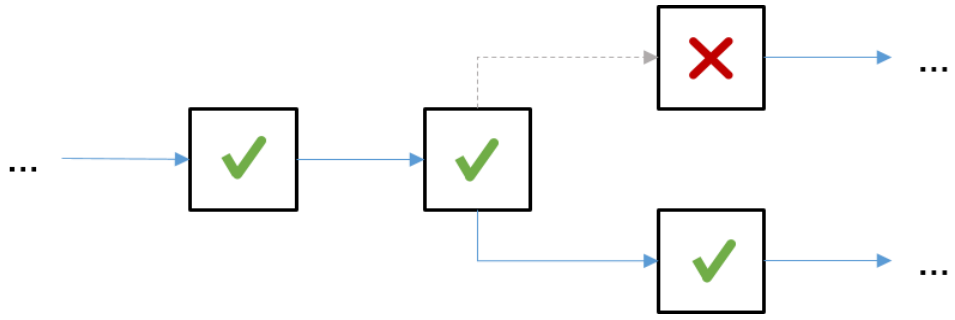

Figura 1.3 Influencia del estado de activación en la ejecución: conexión

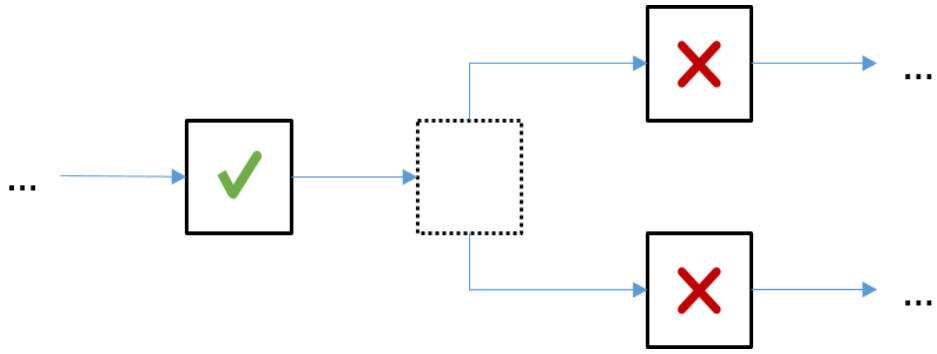

Figura 1.4 Influencia del estado de activación en la ejecución: componente

De manera general el flujo de ejecución está basado en una Red de Petri donde cada componente se ejecuta si se cumplen estas condiciones:

1. Si el componente actual está habilitado (acción del usuario).
2. Si tiene la información necesaria para ejecutarse (Red de Petri), es decir, si hay datos disponibles de alguno de los nodos que se conectan al actual, lo que evidencia un comportamiento secuencial en el flujo de análisis.

1.2 Arquitecturas de software en red

Por lo general, los sistemas distribuidos son complejas piezas de software cuyos componentes se encuentran, por definición, dispersos en diversas máquinas. Para dominar esta complejidad, resulta crucial que los sistemas se encuentren organizados adecuadamente. Existen diferentes formas de visualizar la organización de un sistema distribuido, pero un modo evidente es saber diferenciar la organización lógica de la colección de componentes del software de la organización física real. Las arquitecturas de software nos dicen cómo se organizarán los componentes de software, y cómo deben interactuar (Tanenbaum and Steen, 2008).

Arquitectura par-a-par

En mayo de 1999, Shawn Fanning introdujo la aplicación para compartir música y archivos llamada **Napster**. Esta aplicación dio comienzo de las redes *peer-to-peer*, como las conocemos hoy día, donde "los usuarios que participan establecen una red virtual, totalmente independiente de la red física, sin tener que obedecer a cualquier autoridad administrativa o restricciones"

Una red par a par o ***P2P***, por su nombre en inglés, es básicamente una red informática que no tiene clientes y servidores fijos, sino una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red (González, 2008).

Las redes ***P2P*** permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados. Normalmente este tipo de redes se implementan como redes superpuestas construidas en la capa de aplicación de redes públicas como **Internet**.

Las redes *peer-to-peer* aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, y obtienen así más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales. Esto se debe a que una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación. Aunque existen casos en que es todo lo contrario, por ejemplo, la segunda generación de estas redes mantiene una estructura descentralizada, por lo que realizar búsquedas conlleva a un elevado tráfico de red (Fernández and Arias, 2012).

Las redes ***P2P*** son útiles para diversos propósitos. A menudo se usan para compartir ficheros de cualquier tipo (por ejemplo, audio, vídeo o *software*). Este tipo de red también suele usarse en telefonía ***VoIP***[8] y en sistemas dedicados a las video conferencias como *Skype*, para hacer más eficiente la transmisión de datos en tiempo real, y en la mensajería instantánea como es el caso de *Jabber*. Además, se encuentran presentes en aplicaciones encaminadas al almacenamiento y distribución de contenidos digitales tales como: *Gnutella*, *Napster*, *BitTorrent*, entre otros.

Es importante tener en cuenta que la eficacia de los nodos en el enlace y transmisión de datos no solo depende de los sistemas o modelos que se empleen, puede variar según su configuración local (cortafuegos, NAT, *ruters*, etc.), velocidad de proceso, disponibilidad de ancho de banda de su conexión a la red y capacidad de almacenamiento en disco.

[8] Voz sobre protocolo de internet

Como todo sistema, las redes ***P2P*** poseen determinas características que la identifican del resto. A continuación, se mencionarán seis características deseadas para este tipo de red:

- **Escalabilidad**. Las redes ***P2P*** tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos estén conectados a una red P2P, mejor será su funcionamiento. Así, cuando los nodos llegan y comparten sus propios recursos, los recursos totales del sistema aumentan. Esto es diferente en una arquitectura del modo servidor-cliente con un sistema fijo de servidores, en los cuales la adición de clientes podría significar una transferencia de datos más lenta para todos los usuarios.
- **Robustez**. La naturaleza distribuida de las redes *peer-to-peer* también incrementa la robustez en caso de haber fallos en la réplica excesiva de los datos hacia múltiples destinos, —y en sistemas ***P2P*** puros— permitiendo a los *peers* encontrar la información sin hacer peticiones a ningún servidor centralizado de indexado. En el último caso, no hay ningún punto singular de falla en el sistema.
- **Descentralización**. Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red.
- **Distribución de costes entre los usuarios**. Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.
- **Anonimato**. Es deseable que en estas redes quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo, siempre que así lo necesiten los usuarios. Muchas veces el derecho al anonimato y los derechos de autor son incompatibles entre sí, y la industria propone mecanismos como el ***DRM***[9] para limitar ambos.
- **Seguridad**. Es una de las características deseables de las redes ***P2P*** menos implementada. Los objetivos de un ***P2P*** seguro serían identificar y evitar los nodos maliciosos, evitar el contenido infectado, evitar el espionaje de las comunicaciones entre nodos, creación de

[9] Gestión de derechos digitales

grupos seguros de nodos dentro de la red, protección de los recursos de la red, etc. La mayor parte de los nodos aún están bajo investigación, pero los mecanismos más prometedores son: cifrado multiclave, cajas de arena, gestión de derechos de autor (la industria define qué puede hacer el usuario; por ejemplo, la segunda vez que se oye la canción se apaga), reputación (permitir acceso sólo a los conocidos), comunicaciones seguras, comentarios sobre los ficheros, etc.

De manera general si hacemos una comparación rápida con la distribución cliente/servidor podemos decir que los sistemas basados en ***P2P*** brindan modelos con una mejor tolerancia a fallos. Gracias a que el contenido está distribuido entre varios nodos responde bien ante caídas de algunos de sus nodos, en caso de nodos con pocos recursos (CPU, Memoria, ancho de banda) estos pueden publicar contenido debido a esa misma distribución de nodos en la red que sirve de apoyo, lo que posibilita una mayor escalabilidad. Mientras que a modo de desventaja se puede mencionar un mayor consumo de recursos en el cliente, tiempos de descarga elevados (para ficheros distribuidos en pocos nodos) y protocolos mucho más complejos.

Las redes ***P2P*** desde su surgimiento hasta el momento, han atravesado por tres fases conocidas como generaciones (Fernández and Arias, 2012). Cada una de ellas representa un paso de evolución con respecto a su antecesora.

- **Primera Generación:** La primera generación estaba presidida por *Napster*, representa una red centralizada donde un servidor central sirve de enlace entre los *peers.* La presencia de un servidor central hacia que la búsqueda de recursos o *peers* fuera rápida, además mantiene un registro de los recursos que comparte cada *peers.* Por otro lado, la existencia de este servidor es un punto central de fallos y es un objetivo sensible a ataques, también representa un problema al momento de escalar el sistema, por su característica en común con los sistemas basados en cliente/servidor.
- **Segunda Generación:** La segunda generación toma vida con **Gnutella**[10], y tiene como objetivo principal eliminar el punto central de fallos de la primera generación. El objetivo se alcanza creando redes descentralizadas donde no existe un servidor central como punto

[10] Software distribuido para distribución de archivos entre pares, sin un servidor central.

de contacto entre los pares, por lo que todas las comunicaciones se realizan de par a par, o incluso a través de otros pares. El nuevo diseño da una carencia de estructura, lo que condiciona una mayor robustez y una menor sensibilidad a ataques o caídas de nodos. Como efecto negativo se obtiene un mayor tráfico en la red cuando se realizan búsquedas de información, —y puede que haya casos en lo que no encontremos lo que buscamos, aun cuando está en la red— como efecto secundario de la descentralización de la información.

- **Tercera Generación:** La tercera generación representa una actualización de la segunda. Su objetivo es mantener las ventajas de su antecesora minimizando sus inconvenientes. Normalmente se introduce cierta estructura en la red para hacer más eficiente la distribución de consultas y/o contenidos, un ejemplo sería **Gnutella 2** con la presencia de supernodos.

Arquitectura cliente-servidor

El principal motivo detrás de su creación es la necesidad que tienen las organizaciones (empresas o instituciones públicas o privadas) de realizar sus operaciones más ágil y eficientemente, debido a la creciente presión competitiva a la que están sometidas. Esto se traduce en la necesidad de que su personal sea más productivo, que se reduzcan los costos y gastos de operación, al mismo tiempo que se generan productos y servicios con mayor rapidez y calidad.

En este contexto es necesario establecer una infraestructura de procesamiento de información que cuente con los elementos requeridos para proveer información adecuada, exacta y oportuna en la toma de decisiones, y para proporcionar un mejor servicio a los clientes. El modelo cliente/servidor reúne las características necesarias para proveer esta infraestructura, independientemente del tamaño y complejidad de las operaciones. De acuerdo con (Bochenski, 1994), son diez las características que definen un ambiente cliente/servidor. Las cinco primeras son de carácter obligatorio, mientras que las otras cinco son de cumplimiento opcional.

a) Una arquitectura cliente/servidor consiste de un proceso cliente y un proceso servidor que pueden ser distinguidos uno de otro y que pueden interactuar bastante independientemente

b) Las partes cliente y servidor pueden operar, aunque no necesariamente, en plataformas computacionales diferentes.

c) Tanto la parte cliente como la del servidor pueden ser actualizadas individualmente sin que la otra deba serlo también.

d) El servidor es capaz de dar servicio a múltiples clientes en forma concurrente. En algunos sistemas pueden acceder a múltiples servidores.

e) Un sistema cliente/servidor incluye algún tipo de capacidad de red.

f) Una porción significativa (a veces la totalidad) de la lógica de la aplicación reside en el cliente.

g) El procesamiento es iniciado usualmente en el lado del cliente, no del servidor. Sin embargo, los servidores de bases de datos pueden iniciar acciones basadas en "disparos automáticos", "reglas del negocio" o procedimientos almacenados.

h) Una interfaz gráfica de usuario amigable generalmente reside en el lado del cliente.

i) La capacidad de un lenguaje estructurado de consultas es una característica de la mayoría de los sistemas cliente/servidor.

j) El servidor de base de datos debería proporcionar seguridad y protección a los datos.

Por su parte (Orfali, 1994) resume así las características de los sistemas cliente/servidor:

a) Servicio. El ambiente cliente/servidor conforma una relación entre procesos que se ejecutan en equipos separados. El proceso servidor ofrece servicios, mientras que los clientes los solicitan.

b) Recursos compartidos. Los servidores regulan el acceso a los recursos comunes por parte de los clientes.

c) Protocolos asimétricos. Los clientes pueden pertenecer a una amplia variedad de tecnologías, y son los responsables por iniciar las solicitudes de servicios. Un servidor es un ente pasivo que se encuentra en espera permanente por dichas solicitudes.

d) Transparencia de localización. Aun cuando un proceso servidor puede residir en el mismo equipo que los procesos clientes, es indispensable que los sistemas cliente/servidor oculten a estos la localización física de los servidores, redireccionando apropiadamente las llamadas a los servicios requeridos.

e) Apertura. El software cliente/servidor debe ser lo más independiente posible de las plataformas de hardware y sistemas operativos involucrados.

f) Intercambios basados en mensajes. Los servidores y los clientes participan de sistemas "débilmente acoplados", cuya relación se implementa con mecanismos de paso de mensajes.

g) Encapsulación de servicios. Los procesos servidores deben poder ser actualizados sin que esto provoque cambios en los clientes. Para lograr lo anterior solo es necesario mantener inalterada la interfaz de comunicación entre ambos componentes.

h) Escalabilidad. El escalamiento de los sistemas cliente/servidor puede ser horizontal (adición o eliminación de clientes sin afectar significativamente el rendimiento global de un sistema) o vertical (crecimiento hacia configuraciones más grandes y eficientes).

i) Integridad. Los servicios y datos de un servidor se ubican en un lugar centralizado, lo que simplifica su mantenimiento y protección.

Dentro de la estructura cliente/servidor —la cual puede ser de dos capas o n-capas— existen dos enfoques principales: el primero es conocido como Cliente Ligero o delgado y es aquel que posee toda la lógica de la aplicación en el lado del servidor, por lo que en la parte del cliente solo se tiene una interfaz gráfica y los medios necesarios para establecer comunicación con su servidor; el segundo se conoce como Cliente Pesado o grueso y consiste en que gran parte de la lógica está en el cliente y solo utiliza el servidor como apoyo o para algunas funciones específicas como por ejemplo almacenamiento (no se aconseja su uso en ambientes empresariales).

Esta clasificación (arquitectura de dos capas o niveles) es la distribución básica donde solo existe el cliente y un servidor, mientras que el de n-capas tiene servidores adicionales a su disposición (ejemplo: Servidores de bases de datos). La unión de las características y capacidades hacen que tenga ciertas ventajas tales como la centralización del control. Esto significa que los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.

La centralización también facilita la tarea de poner al día datos u otros recursos, escalabilidad, que se traduce en la posibilidad de aumentar la capacidad de los clientes y los servidores por separado. También posibilita que el mantenimiento sea más fácil, ya que, al estar las funciones y responsabilidades distribuidas entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor sin que sus clientes se vean afectados por ese cambio (o se afecten mínimamente).

1.3 Estilos arquitectónicos

Se debe iniciar esta explicación sobre arquitecturas considerando primero la organización lógica de los sistemas en componentes de software, también conocida como arquitectura de software (Bass, Clements and Kazman, 2003). La investigación sobre arquitecturas de software ha madurado considerablemente, según (Tanenbaum and Steen, 2008) la idea de estilo arquitectónico es importante, tal estilo se formula en términos de componentes, la forma en que los componentes interactúan entre sí, el intercambio de datos entre los componentes y, por último, en cómo es que estos elementos se configuran juntos en un sistema.

Un componente es una unidad modular con las interfaces requeridas bien definidas, dicha unidad es reemplazable dentro de su ambiente (Object Management Group, 2004). La cuestión importante sobre un componente para sistemas distribuidos es que pueda ser reemplazado, con la condición de que se respeten sus interfaces. Un concepto más difícil de entender es el de conector, el cual generalmente se describe como un mecanismo que media la comunicación, coordinación o cooperación entre componentes (Mehta, Medvidovic and Phadke, 2000). Por ejemplo, un conector puede formarse por los medios disponibles para hacer llamadas a procedimientos remotos (RPC), paso de mensajes, o flujo de datos.

Por medio de componentes y conectores podemos lograr varias configuraciones, las cuales se han clasificado en estilos arquitectónicos. Varios estilos ya están identificados y los más importantes para sistemas distribuidos son:

- Arquitecturas en capas.
- Arquitecturas basadas en objetos.
- Arquitecturas centradas en datos.
- Arquitecturas basadas en eventos.

La idea básica para el estilo en capas es sencilla según (Tanenbaum and Steen, 2008): los componentes se estructuran (organizan) a modo de capas, donde al componente de la capa L_i se le permite llamar a componentes de la capa subyacente L_{i-1}, pero no del resto de capas, como ilustra la **Figura 1.5(a).** Este estilo se ha adoptado ampliamente en la comunidad de redes. Una

observación clave es que el control generalmente fluye de capa a capa: las peticiones se mueven hacia abajo en la jerarquía mientras que los resultados se mueven hacia arriba.

Una organización bastante libre es la que siguen las arquitecturas basadas en objetos, la cual aparece en la **Figura 1.5(b)**. En esencia, cada objeto corresponde a lo que hemos definido como componente, y estos componentes se conectan a través de un mecanismo de llamadas a procedimientos remotos (***RPC***). Sin ser sorprendente, esta arquitectura de software coincide con la arquitectura de sistemas cliente-servidor que describimos antes. La arquitectura basada en capas y la basada en objetos aún son los estilos más importantes utilizados para implementar grandes sistemas de software (Bass, Clements and Kazman, 2003).

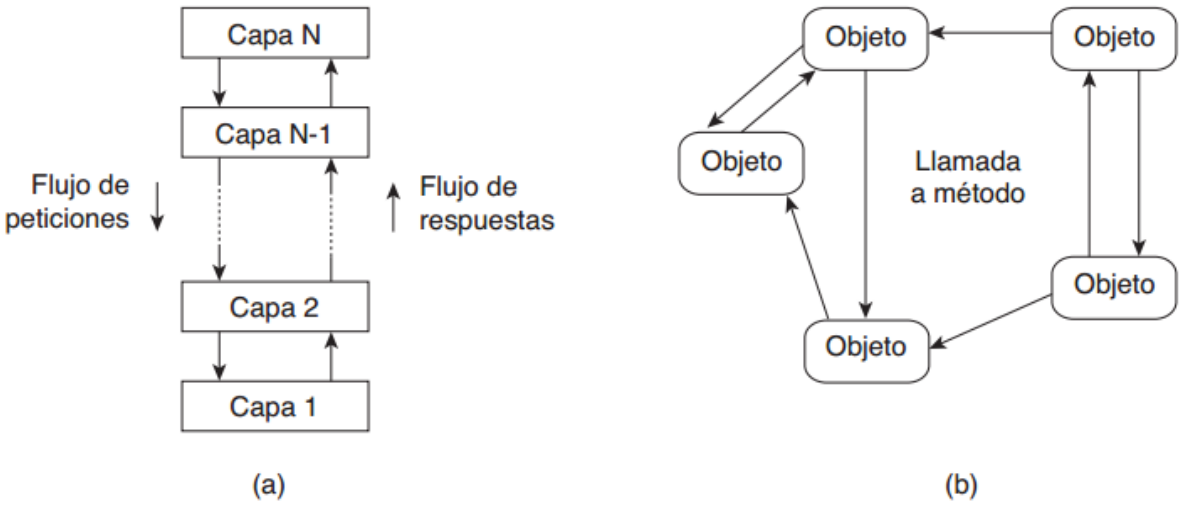

Figura 1.5 Estilos arquitectónicos (a) en capas, y (b) basado en objetos

Las arquitecturas centradas en datos evolucionaron alrededor de la idea de que los procesos se comunican a través de un repositorio común (activo o pasivo). Se puede argumentar que, para sistemas distribuidos, estas arquitecturas son tan importantes como las basadas en capas y objetos. Por ejemplo, un punto a favor que han desarrollado las aplicaciones en red es que se basan en un sistema de archivos distribuidos compartidos donde casi todas las comunicaciones se realizan a través de archivos (Tanenbaum and Steen, 2008). Se muestra una ilustración del estilo en la **Figura** 1.6**(b)**.

En las arquitecturas basadas en eventos**,** los procesos se comunican básicamente a través de la propagación de eventos, los que opcionalmente transportan datos, como ilustra la **Figura 1.6(a).** Para sistemas distribuidos, la propagación de eventos se ha asociado con lo que se conoce como sistemas de publicación-subscripción (Eugster *et al.*, 2003)**.**

La idea básica es que los procesos publican eventos después de los cuales el *middleware* asegura que solo aquellos procesos suscritos a tales eventos los recibirán. La principal ventaja de los sistemas basados en eventos es que los procesos están libremente acoplados. En principio, no necesitan referirse uno a otro explícitamente. A esto se le conoce también como desacoplado en el espacio o referencialmente desacoplado (Tanenbaum and Steen, 2008).

Las arquitecturas basadas en eventos pueden combinarse con arquitecturas centradas en datos, y arrojan lo que conocemos como espacios de datos compartidos. La esencia de los espacios de datos compartidos es que los procesos ahora también están desacoplados en el tiempo: no es necesario que ambos estén activos cuando la comunicación se lleva a cabo. Además, muchos espacios de datos compartidos utilizan una interfaz similar a la SQL para el repositorio compartido lo que hace posible acceder a los datos utilizando una descripción, en lugar de una referencia explícita, como en el caso de los archivos.

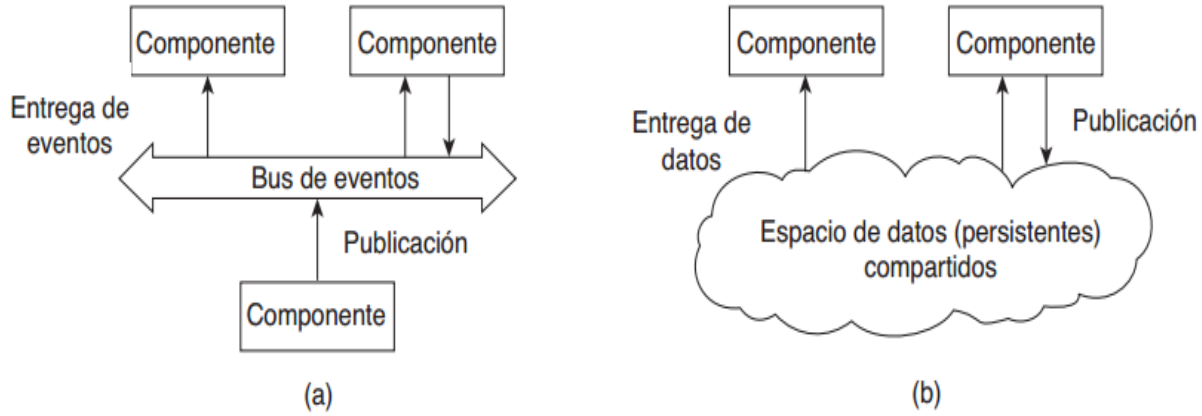

Figura 1.6 Estilo arquitectónico (a) basado en eventos, y (b) espacio

1.4 Arquitecturas de sistemas

Según (Tanenbaum and Steen, 2008) el modelo cliente/servidor ha estado sujeto a diversas controversias y debates, una de las principales cuestiones es cómo distinguir entre un cliente y un servidor, a pesar de esto mucha gente defiende la existencia de tres niveles que lo permitirían:

- El nivel de interfaz de usuario
- El nivel de procesamiento
- El nivel de datos

Basadas en estos niveles y en las diversas formas que pueden ser distribuidas han surgido varias formas de sistemas distribuidos, de los cuales se mencionaran a continuación algunas, así como las ventajas que presentan. pero antes se mostrara en la **Figura 1.7** un bosquejo básico de la estructura de un sistema distribuido.

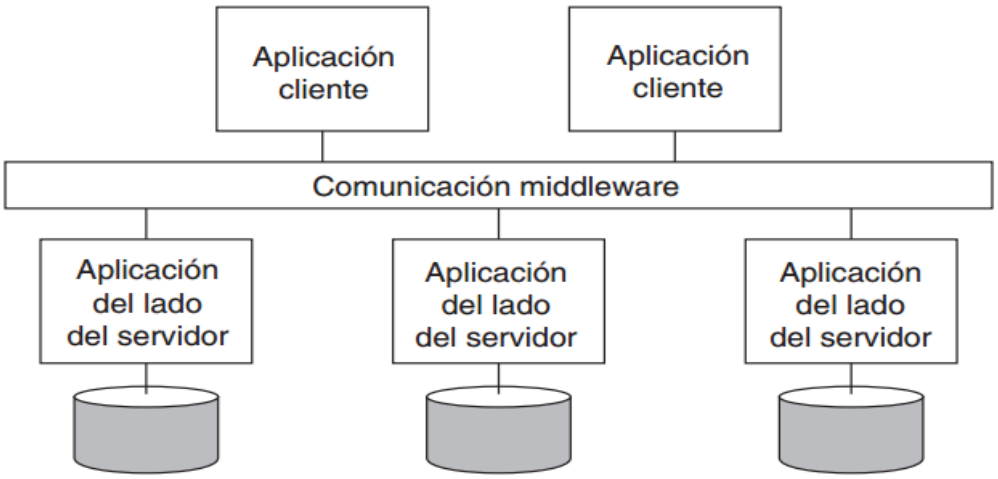

Figura 1.7 Estructura de Sistema Distribuido

Presentación Distribuida:

Un sistema distribuido se define como una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones; cada máquina posee sus componentes de hardware y software que el programador percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El programador accede a los componentes de software (objetos)

remotos, de la misma manera en que accedería a componentes locales, en un grupo de computadoras que usan un *middleware* entre los que destacan ***RPC*** y ***SOAP*** para conseguir un objetivo. Aquí se distribuye la interfaz entre el cliente y la plataforma servidora, donde se encuentra la aplicación y los datos mientras que el PC solo se aprovecha para mejorar la interfaz gráfica del usuario.

Las ventajas de esta distribución son la revitalización de los sistemas antiguos y un bajo costo de desarrollo, pero por otra parte desaprovecha la GUI y LAN y mantiene la interfaz de usuario en muchas plataformas.

Lógica Distribuida:

La interfaz está en el cliente, terceros servicios, como bases de datos, están en el servidor y la lógica de la aplicación distribuida está entre el cliente y el servidor. Presenta como ventajas una arquitectura que puede manejar todo tipo de aplicaciones, pueden utilizarse en sistemas existentes y los programas del sistema pueden distribuirse al nodo más apropiado. Sus debilidades radican en la dificultad a la hora de diseñar, así como las pruebas y el mantenimiento, sobretodo si los programas del cliente y el servidor están hechos en distintos lenguajes de programación.

Aunque solo se han mencionado dos, en la **Figura 1.8** se muestran algunas de las distribuciones mencionadas.

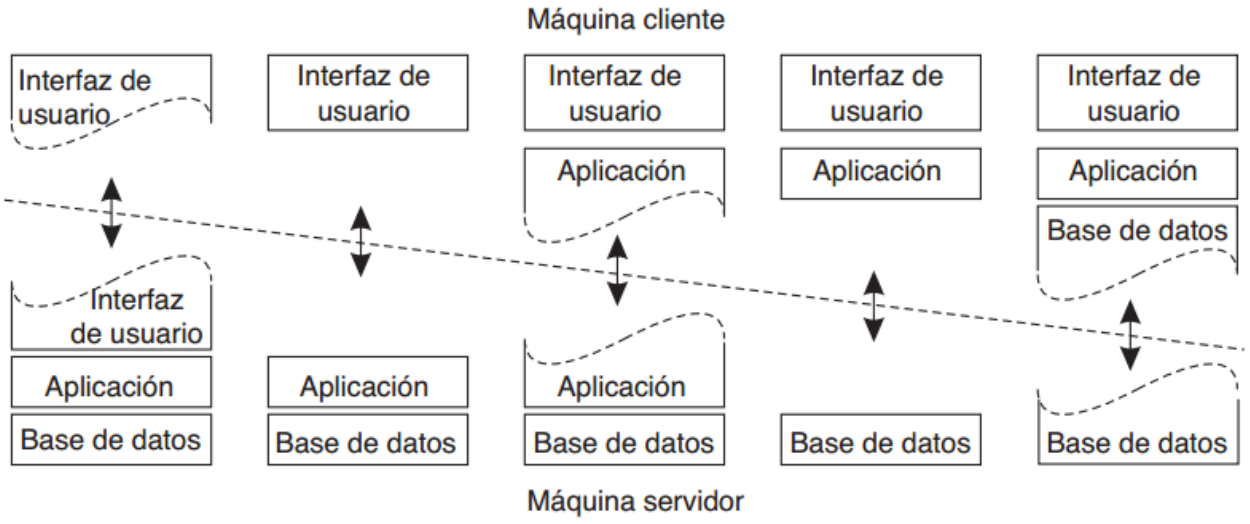

Figura 1.8 Posibles distribuciones de arquitecturas de sistemas

1.5 Programación Paralela

Como la naturaleza del modelo es asincrónica y concurrente es necesario aplicar técnicas de programación paralela para su desarrollo. La programación paralela ha sido abordada en numerosos artículos científicos desde el siglo pasado y aún deja que hablar, y esto está dado por el permanente progreso de las tecnologías que le dan soporte. En la actualidad los sistemas con cierto grado de paralelismo o concurrencia como es conocida también son tan comunes que muchas personas no se percatan de su existencia en sus vidas, dos ejemplos básicos serían un teléfono inteligente y nuestro propio ordenador.

Según (Foster, 1994) la computación paralela usa un conjunto de procesadores que son capaces de cooperar en la solución de problemas computacionales. A continuación, se hablará acerca del tema, presentando algunos conceptos relacionados, pero sin profundizar demasiado.

Clasificación de Flynn.

Probablemente la clasificación más popular de computadores sea la clasificación de Flynn. Esta taxonomía de las arquitecturas está basada en la clasificación atendiendo al flujo de datos e instrucciones en un sistema. Un flujo de instrucciones es el conjunto de instrucciones secuenciales que son ejecutadas por un único procesador, y un flujo de datos es el flujo secuencial de datos requeridos por el flujo de instrucciones (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012). Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías, las cuales se evidencian a continuación y en la **Figura 1.9**.

- ***SISD*** (por las siglas en inglés de *Single Instruction stream, Single Data stream*): Flujo único de instrucciones y flujo único de datos. Este el concepto de arquitectura serie de Von Neumann donde, en cualquier momento, sólo se está ejecutando una única instrucción. A menudo a los ***SISD*** se les conoce como computadores serie escalares.

  Todas las máquinas ***SISD*** poseen un registro simple que se llama contador de programa que asegura la ejecución en serie del programa. Conforme se van leyendo las instrucciones de la memoria, el contador de programa se actualiza para que apunte a la siguiente instrucción a procesar en serie.

  Prácticamente ningún computador puramente ***SISD*** se fabrica hoy día, ya que la mayoría de procesadores modernos incorporan algún grado de paralelización como es la

segmentación de instrucciones o la posibilidad de lanzar dos o más instrucciones a un tiempo (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012).

- ***MISD*** (por las siglas en inglés de *Multiple Instruction stream, Single Data stream*): Flujo múltiple de instrucciones y único flujo de datos. Esto significa que varias instrucciones actúan sobre el mismo y único trozo de datos. Este tipo de máquinas se puede interpretar de dos maneras. Una es considerar la clase de máquinas que requerirían que unidades de procesamiento diferentes recibieran instrucciones distintas operando sobre los mismos datos.

  Esta clase de arquitectura ha sido clasificada por numerosos arquitectos de computadores como impracticable o imposible, y en estos momentos no existen ejemplos que funcionen siguiendo este modelo. Otra forma de interpretar los ***MISD*** es como una clase de máquinas donde un mismo flujo de datos fluye a través de numerosas unidades procesadoras.

  Arquitecturas altamente segmentadas, como los *arrays* sistólicos *o* los procesadores vectoriales, son clasificados a menudo bajo este tipo de máquinas. Las arquitecturas segmentadas, o encauzadas, realizan el procesamiento vectorial a través de una serie de etapas, cada una ejecutando una función particular produciendo un resultado intermedio. La razón por la cual dichas arquitecturas son clasificadas como ***MISD*** es que los elementos de un vector pueden ser considerados como pertenecientes al mismo dato, y todas las etapas del cauce representan múltiples instrucciones que son aplicadas sobre ese vector (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012).
- ***SIMD (***por las siglas en inglés de *Single Instruction stream, Multiple Data stream)*: Flujo de instrucción simple y flujo de datos múltiple. Esto significa que una única instrucción es aplicada sobre diferentes datos al mismo tiempo. En las máquinas de este tipo, varias unidades de procesado diferentes son invocadas por una única unidad de control. Al igual que las ***MISD***, las ***SIMD*** soportan procesamiento vectorial (matricial) asignando cada elemento del vector a una unidad funcional diferente para procesamiento concurrente.

  Por ejemplo, el cálculo de la paga para cada trabajador en una empresa es repetir la misma operación sencilla para cada trabajador; si se dispone de una arquitectura ***SIMD*** esto se puede calcular en paralelo para cada trabajador. Por esta facilidad en la paralelización de vectores de datos (los trabajadores formarían un vector) se les llama también procesadores matriciales (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012)**.**

- ***MIMD*** **(**por las siglas en inglés de *Multiple Instruction stream, Multiple Data stream***):** Flujo de instrucciones múltiple y flujo de datos múltiple. Son máquinas que poseen varias unidades procesadoras en las cuales se pueden realizar múltiples instrucciones sobre datos diferentes de forma simultánea. Las MIMD son las más complejas, pero son también las que potencialmente ofrecen una mayor eficiencia en la ejecución concurrente o paralela. Aquí la concurrencia implica que no sólo hay varios procesadores operando simultáneamente, sino que además hay varios programas (procesos) ejecutándose también al mismo tiempo.

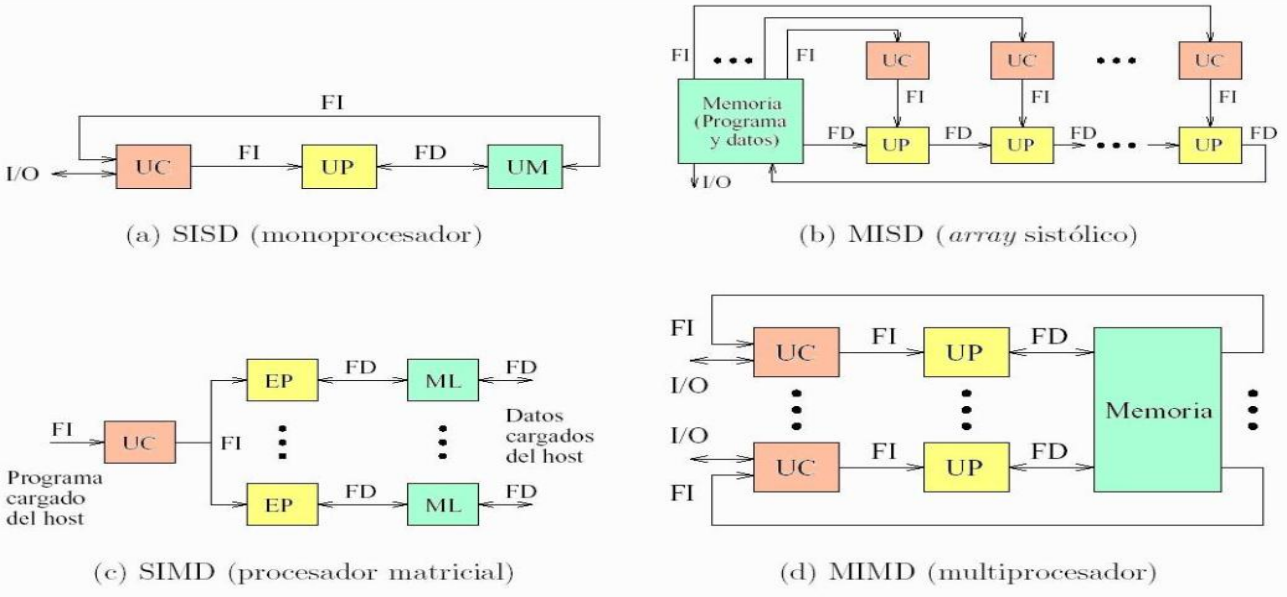

Figura 1.9 Clasificación de Fynn

Multiprocesadores.

Un multiprocesador se puede ver como un computador paralelo compuesto por varios procesadores interconectados que pueden compartir un mismo sistema de memoria. Los procesadores se pueden configurar para que ejecute cada uno una parte de un programa o varios programas al mismo tiempo.

Un multiprocesador está generalmente formado por *n* procesadores y *m* módulos de memoria. La red de interconexión conecta cada procesador a un subconjunto de los módulos de memoria. Dado que los multiprocesadores comparten los diferentes módulos de memoria, pudiendo acceder varios procesadores a un mismo módulo, a los multiprocesadores también se les llama sistemas de

memoria compartida (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012). Dependiendo de la forma en que los procesadores comparten la memoria, podemos hacer una subdivisión de los multiprocesadores:

- ***UMA*** (Acceso de memoria uniforme)**:** la memoria física está uniformemente compartida por todos los procesadores. Esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de memoria. Cada procesador puede tener su cache privada y los periféricos son también compartidos de alguna manera.
- ***NUMA*** **(**Acceso de memoria no uniforme**):** Es un sistema de memoria compartida donde el tiempo de acceso varía según el lugar donde se encuentre localizado el acceso. La ventaja de estos sistemas es que el acceso a la memoria local es más rápido que en los UMA, aunque un acceso a memoria no local es más lento. Lo que se intenta es que la memoria utilizada por los procesos que ejecuta cada procesador, se encuentre en la memoria de dicho procesador para que los accesos sean lo más locales posible.

Multicomputadores.

Un multicomputador se puede ver como un computador paralelo en el cual cada procesador tiene su propia memoria local. La memoria del sistema se encuentra distribuida entre todos los procesadores y cada procesador solo puede direccionar su memoria local, para acceder a las memorias de los demás procesadores debe hacerlo por paso de mensajes**.** Esto significa que un procesador tiene acceso directo solo a su memoria local, el acceso al resto de memorias del resto de procesadores es indirecto. Este acceso local y privado a la memoria es lo que diferencia los multicomputadores de los multiprocesadores.

La transferencia de datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de unos procesadores a otros se realiza por tanto por múltiples transferencias entre procesadores conectados dependiendo de cómo esté establecida la red. Dado que la memoria está distribuida entre los diferentes elementos de proceso, a estos sistemas se les llama distribuidos, aunque no hay que olvidar que pueden haber sistemas que tengan la memoria distribuida pero compartida y por lo tanto no ser multicomputadores (‘Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.’, 2012)**.**

1.6 Herramientas de Desarrollo

En este epígrafe se realizará un resumen de las principales alternativas y herramientas utilizadas para el desarrollo del presente trabajo, ya sea el lenguaje de programación usado, e cual influye a la ahora de escoger los modulos y/o *frameworks* apropiados para la realización del ambiente de red dentro del modelo, y por último los principales aspectos a tener en cuenta al momento de escoger los módulos para la interfaz gráfica.

Lenguaje de programación

Debido al amplio uso y el auge que tiene, asi como la gran cantidad de herramientas que posee gracias a su extensa comunidad de desarrolladores, sin obviar las facilidades para un desarrollo ágil, se decidió emplear Python 3 como lenguaje para desarrollar la implementación, específicamente la versión 3.6.2. La decisión también está influenciada por la implementación del trabajo previo que está desarrollado en el mismo.

A pesar que es un lenguaje interpretado y tipado dinámico, lo que podría suponer una disminución de la velocidad de ejecución con respecto a alternativas como C/C++, Fortran o incluso Java (Johansson, 2016), no es significativa como para despreciarlo y quitarle el puesto ganador como lenguaje seleccionado, sobre todo porque según (Kuhlman, 2012) las versiones más recientes del intérprete de Python cachean el código compilado a *bytecode* después de que este se ejecuta por primera vez, aumentado considerablemente la velocidad de ejecuciones subsecuentes. Además la simplicidad y elegancia del código Python permite un fácil mantenimiento y lectura del código.

Como cualquier otro proyecto, este se desea que sea portable lo que lleva a tener en consideración la portabilidad lenguaje. Python, al ser interpretado puede correr en cualquier sistema operativo que tenga un intérprete, y además, la mayoría de la familia Linux, incluso los Mac OS X de Apple, vienen con el intérprete integrado, y para el sistema operativo Windows existen múltiples distribuciones que cuentan con todos los paquetes necesarios para tareas científicas, como es el caso de Anaconda y Pythonxy.

Ambiente Red

Para la actual versión de Python existen numerosas bibliotecas, frameworks y módulos especializadas en el desarrollo de aplicaciones que hagan uso de recursos de la red, por lo tanto, no es tarea fácil seleccionar alguno de ellos sin un previo estudio de las capacidades de cada una. Por lo mencionado con anterioridad vamos a centrarnos en las más usadas y famosas.

El uso de recursos de software viene condicionado por la necesitad que se tenga, por ejemplo, si necesitamos un servidor que este dedicado al cómputo, es decir, que ejecute código dedicado al procesamiento, es conveniente el uso de ***RPC*** que en caso de Python viene por defecto en la biblioteca *xmlrpc*. Este es un recurso que permite la invocación de métodos remoto mediante el uso del protocolo ***HTTP***. Los sistemas ***RPC*** permiten llamar a funciones en otros procesos o en un servidor remoto utilizando la misma sintaxis que se emplearía en una llamada a una rutina de una ***API*** (por las siglas en inglés de *Application Programming Interface*) local o una biblioteca (Rhodes and Goerzen, 2016). Como se mencionó su uso es conveniente en el uso de servidores en la red, específicamente en dos situaciones:

- El programa tiene mucho trabajo que hacer, y se desea distribuir en varias máquinas a través de la red, sin tener que cambiar el código que realiza las llamadas, que ahora es remoto.
- Se necesita datos o información que solo está disponible en otro disco duro o red, y una interfaz RPC permitiría enviar fácilmente consultas a otro sistema y recibir una respuesta.

Uno de los *frameworks* más famoso en los tiempos actuales es Django. Posee numerosas funciones y constantemente se amplía hacia nuevos campos, es uno de los más completos ya que permite su expansión mediante extensiones y *plugins*. Django es una herramienta muy poderosa para el desarrollo de software ya que nos brinda la posibilidad de usar recursos de red, base datos, manipulación de procesos en paralelo y varios otros de una manera muy sencilla e intuitiva. Aunque a pesar de todas sus ventajas tiene un enfoque mayormente dirigido hacia el desarrollo de servicios web (Holovaty and Kaplan-Moss, 2008).

Otro *frameworks* famoso es Twisted, desarrollado por *Twisted Matrix Lab* es otro de los *frameworks* relacionados con el tema, inicialmente fue usado y comercializado en juegos, aunque es una plataforma para el desarrollo de aplicaciones de internet (Labs, 2018). Twisted posee

implementación de todos los protocolos de red existentes hasta el momento, y nos permite el empleo de uno o todos en nuestras aplicaciones, lo que aumenta la escalabilidad de nuestro sistema considerablemente, además maneja ejecución concurrente, lo que es especialmente útil en el desarrollo de servidores que dan servicio a cientos de cliente.

Si ninguna de las herramientas resulta convincente del todo o no cumple las condiciones que se precisan, se puede recurrir a las bibliotecas *urllib*, *urllib3* (*urllib2* para versiones de Python 2), *http* y *socket*, que vienen por defecto en Python y así desarrollar otra propuesta. Básicamente los módulos mencionados dan soporte a casi todos los protocolos y funciones que podamos necesitar, es el nivel más bajo de abstracción en este aspecto del lenguaje. Es necesario tener presente que esto requeriría un trabajo intenso y complejo.

Por último, para concluir esta breve lista de propuestas, se hablará de ***RPyC***, que, si bien esta es una ampliación de ***RPC***, brinda un conjunto nuevo de posibilidades. Llamada Remota de Python o **RPyC** es una herramienta que permite la creación de aplicaciones que no son posibles mediante ***RPC***. Maneja el *object-proxing*, una técnica que emplea la naturaleza dinámica de Python para sobrepasar los límites físicos entre procesos y máquinas (Filiba, 2018), de tal manera que los objetos remotos pueden ser manipulados como si fueran locales.

Esta biblioteca representa una herramienta en el campo del procesamiento distribuido, puesto que además de permitir la invocación de métodos remotos maneja la creación de servidores multihilo, y toda una serie de clases encaminadas a facilitar el trabajo en la red. Esta enorme ventaja combinada con las que ya poseía ***RPC*** hacen que este sea el modulo seleccionado para el desarrollo de nuestra propuesta.

Concurrencia

Python se ha ido consolidando poco a poco dentro de la comunidad de ***HPC***[11] con su sintaxis simple, gran cantidad de bibliotecas estándar y poderosas bibliotecas de terceros para la computación científica. Mientras que la ***HPC*** está monopolizada por Fortran y C, otros lenguajes están esforzándose para ganar terreno, como es el caso de Python que a pesar de ser un lenguaje de alto nivel su autoproclamado objetivo es ser rápido y fácil (Wagner *et al.*, 2017).

[11] Computación de alto rendimiento

Para la programación paralela en Python existen, entre otros, paquetes de sus librerías estándares como el módulo multiprocessing e interfaces externas para rutinas paralelas como ***MPI4Py***[12] que posee interfaces orientadas a objetos similares a C++, además soporta la comunicación optimizada de segmentos de *buffer* (arreglos NumPy, objetos *bytes*/*string*/*array*) con velocidades cercanas a C o Fortran (Dalcin, 2017). La principal característica es que maneja la concurrencia mediante procesos pesados, mientras que el módulo multiprocessing lo realiza mediante los hilos ligeros de Python.

Otra opción sería los hilos de QT llamados *QThreads,* debido a que Python maneja los hilos ligeros de forma interna, donde no emplea hilos verdaderos, sino que el propio interprete maneja sus ejecuciones, haciendo función de *dispatcher*. Este módulo de QT es accesible desde Python mediante el uso del *framework* PyQt, este crea un *binding*[13] entre los programas en Python y los módulos de QT. Los *QThreads* pueden ser ejecutados en varios procesadores simultáneamente, a diferencia de los hilos Python, lo que resulta en un mejor rendimiento.

Interfaz gráfica

En estos momentos se encuentran disponibles varios módulos para el desarrollo de interfaces visuales, tales como ***PyQt*** (en sus distintas versiones), ***Tkinter***, ***PyGTK*** o ***wxPython***. Aunque todas tienen sus características propias, es válido aclarar que presentan soporte para todos los sistemas operativos, así básicamente sus diferencias se encuentran a nivel estético.

Tkinter se considera la biblioteca estándar de Python para el trabajo con interfaces gráficas, es un *binding* de la biblioteca Tcl/Tk creada por John Ousterhout. Esta viene instalada por defecto en la plataforma Windows. PyGTK es un *binding* de la biblioteca GTK[14] que se usa para desarrollar el entorno gráfico GNOME[15], así como sus aplicaciones. Por último, wxPython es también un *binding* sobre la biblioteca wxWidgets que se caracteriza por ser multiplataforma.

[12] Implementación para Python de MPI (Interfaz de Paso de Mensaje) en su versión 4

[13] Enlace

[14] GIMP Tool Kit, Conjunto de Herramientas de GIMP

[15] ***G****NU* ***N****etwork* ***O****bject* ***M****odel* ***E****nvironment:*

Por último, ***PyQt*** en su versión 5, por diversos motivos. Primeramente, el trabajo previo está realizado sobre este mismo módulo, lo que evita la programación desde cero de la interfaz visual o la unión de dos módulos de los mencionados anteriormente. Básicamente este módulo es un *binding* sobre la biblioteca de Qt, la cual por sí sola es ampliamente usada hoy día, además esta biblioteca le da un aspecto homogéneo a la aplicación sin importar la plataforma en la que se ejecute, y, por último, posee una gran eficiencia por su implementación en C++.

1.7 Conclusiones parciales

Las potencialidades de los modelos distribuidos sobre una arquitectura cliente/servidor como vía de aumento de la potencia de cálculo para los sistemas representa una herramienta idónea para ser aplicadas al problema de la sobrecarga de procesamiento de la computadora personal en el modelo de (Mollineda, 2017). También determinadas características de la arquitectura ***P2P*** que permiten su escalabilidad y robustez serían útiles a la hora de concebir la interacción entre los servidores del modelo.

La disponibilidad de herramientas para el desarrollo de sistemas distribuidos ofrece un amplio espectro de donde escoger de acuerdo a las necesidades particulares planteadas por el problema que se aborde. Ello pone un peso especial en su selección de acuerdo a la audiencia y plataforma a la que se apunte, por lo que es necesario considerar cuidadosamente los pros y los contras de cada una de las alternativas disponibles.

CAPÍTULO 2. Propuesta de un modelo distribuido

En este capítulo se abordará la concepción de un nuevo modelo y se trataran las características básicas que se necesitan modificar para la obtención del mismo. Se incluyen además consideraciones de diseño específicas para algunas clases de componentes, apuntando a servir como guía para la implementación de estas.

2.1 Propuesta del modelo

A continuación, se plantea las principales características del modelo distribuido, así como las nuevas formas de interacción, incluyendo la distribución del flujo.

Estructura general

Dadas las características deseadas para el modelo, que se traducen en el deseo de mantener todas las ventajas que ya posee y solo añadir aquellas funciones y características que resulten de provecho, como es el caso de la transición hacia un modelo distribuido, por lo que no es necesario modificar la forma conceptual en la que está prevista el modelo base. La forma más natural que puede adoptar este conceptualmente es la de un conjunto de componentes interconectados, básicamente un grafo dirigido.

Partiendo de este principio, la forma general del modelo (Mollineda, 2017) para cualquiera de los escenarios de análisis considerados estaría dada por un grafo dirigido G_M compuesto por un conjunto de componentes analíticos C_A, haciendo el papel de vértices, los cuales pueden estar tanto en la máquina local donde se está diseñando el flujo de análisis como en un servidor al otro lado del mundo. Además una serie de conexiones entre estos C_A, que las determinamos de forma analítica como E_C, que establecen la forma en que fluyen los datos a través de los componentes interconectados.

Los componentes pueden ser a su vez divididos en conjuntos de acuerdo a su responsabilidad o función primordial. De esta forma se tiene un conjunto de elementos de entrada $C_I \subset C_A, C_i \neq \emptyset$ cuya función principal es la entrada y cierto nivel de limpieza y filtrado de los datos que entran al

sistema. Generalmente, los componentes encargados de esta función no deberían recibir entradas desde otros componentes.

En el caso de los componentes de entrada, se hace la distinción de que el conjunto que los contiene no puede ser vacío. Esto se debe a que, sin una entrada de datos externa de algún tipo, no puede procederse a realizar ningún análisis, por lo que estos componentes de entrada serían el punto de inicio de todo el flujo de análisis.

En segundo lugar, se tienen los componentes que representan procesos de análisis automatizado, aplicación de técnicas estadísticas o filtrado y transformación de los datos de forma general, contenidos en el conjunto $C_T \subset C_A$. Por regla general, los datos entrados al sistema serán transformados y refinados en este tipo de componentes, posiblemente siendo disgregados en conjuntos con una granularidad más fina. Dentro de este subconjunto se ubicarían aquellos componentes que poseen un mayor nivel de procesamiento sobre los datos(Mollineda, 2017).

Finalmente, los componentes de visualización $C_V \subset C_A$ son los encargados de recibir la mayor parte de la interacción del usuario, permitiendo la manipulación de los datos y focalizar los elementos que le resulten interesantes a este, así como ser los principales responsables de la presentación de información y la generación de hipótesis. Los componentes de visualización interactuarían con el flujo general de los datos permitiendo, por ejemplo, limitar las operaciones posteriores a ser realizadas o centrarse en los conjuntos de datos con los que el usuario haya interactuado dentro de ellos.

Aunque los conjuntos han sido tratados separadamente para ilustrar mejor las responsabilidades que caracterizan a sus integrantes, es necesario mencionar que estos no tienen por qué ser excluyentes: el diseño y función de un componente puede contemplar los tres tipos de comportamiento simultáneamente o cualquier combinación de ellos, maximizando la flexibilidad a la hora de diseñar componentes para una implementación de este modelo. La distinción entre qué comportamiento es el apropiado en cada caso puede realizarse en base a los estímulos externos recibidos por el componente, como se discute más adelante en la sección dedicada a las características de la ejecución de un flujo de análisis diseñado de acuerdo a este modelo.

Como ya se mencionó el modelo se desea que esté basado en un sistema distribuido, lo que conlleva a algunos cambios dentro del sistema en general y el flujo de análisis en particular.

Básicamente el flujo de análisis tiene que ser capaz de ejecutarse de forma remota, lo que implica la creación del modelo de la parte servidor.

La variante más sencilla sería hacer una copia de todos los módulos que intervienen en la lógica de aplicación y ubicarla en cada uno de los servidores, pero esto constituye una ineficiencia de programación debido a la alta presencia de código duplicado, sería necesario un trabajo de sincronización bastante fuerte y además no sería un modelo distribuido sino muchos modelos conectados.

Se propone que en cada servidor solo se encuentre el código asociado a cada C_A , es decir, la función que el componente va a aplicar al flujo de datos, a excepción de los módulos de visualización puesto que carece de sentido ubicarlos en una máquina que no sea la local. Estos componentes no realizan procedimientos demasiado costosos computacionalmente como para que sea necesaria su distribución. Los servidores deben comunicarse entre ellos simulando una red P2P para aumentar la robustez del sistema.

La disponibilidad de estos módulos estaría dada por la presencia de un servidor multihilo, que permitiría múltiples conexiones de varios usuarios. Mientras que del lado del cliente, el modelo identifica si el componente actual se desea que se ejecute local, lo sería igual modelo de (Mollineda, 2017) o remoto, que se haría invocando a un servidor. Las formas de interactuar de los clientes con los servidores, he incluso el del flujo de análisis con el servidor se explicará en los siguientes epígrafes.

Las conexiones entre los elementos que conforman el modelo se hayan restringidas únicamente por la capacidad del componente determinado como final de recibir entradas a partir de otros componentes y por las conexiones ya existentes, dado que al igual (Mollineda, 2017) se asume que no existen múltiples conexiones en el mismo sentido entre el mismo par de componentes ni del componente consigo mismo, por lo que las conexiones tienen la forma siguiente: $e_j \in E_c = (c_p, c_q)$ con $c_p, c_q \in C_A, p \neq q$.

Existe la posibilidad de existencia de ciclos en el flujo de análisis, los cuales pueden interpretarse intuitivamente como una forma de iterar y refinar los resultados del proceso o herramienta representado por un componente luego de haber sido transformados por la acción combinada de

otros. El único caso excluido es el de una conexión de un componente consigo mismo porque esta no tiene sentido en la mayoría de los casos considerados (Mollineda, 2017).

La **Figura 2.1** ilustra el aspecto general de un flujo de análisis creado por (Mollineda, 2017).

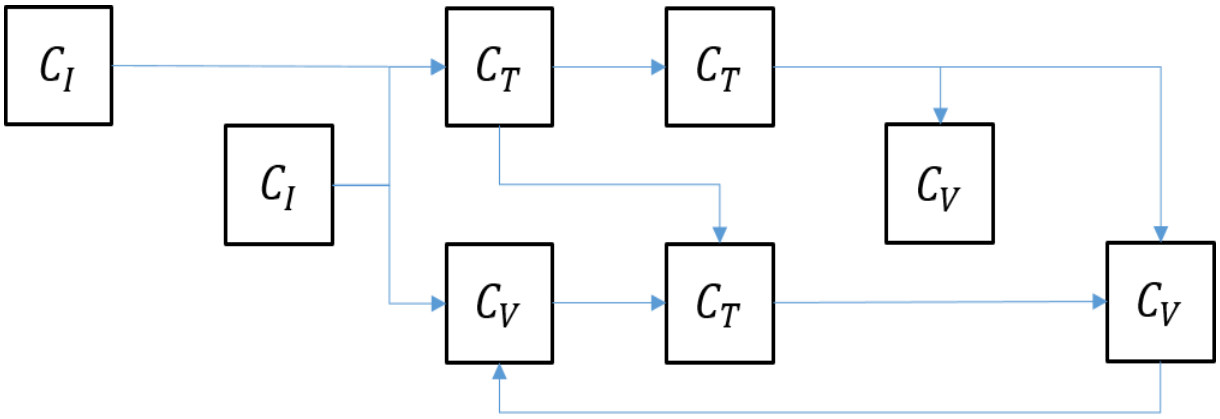

Figura 2.1 Estructura general de un flujo de análisis

Visto que aspecto general posee el flujo de análisis planteado por (Mollineda, 2017) en su modelo, se ha creado un diagrama que se muestra en la **Figura 2.2** donde muestra una representación de un flujo de análisis muy sencillo del modelo distribuido, el cual permite observar las diferencias entre ambos. En el gráfico mostrado, los componentes representados solo por un cuadrado identifican a los que serán ejecutados de forma local, mientras que el resto será en los servidores asignados.

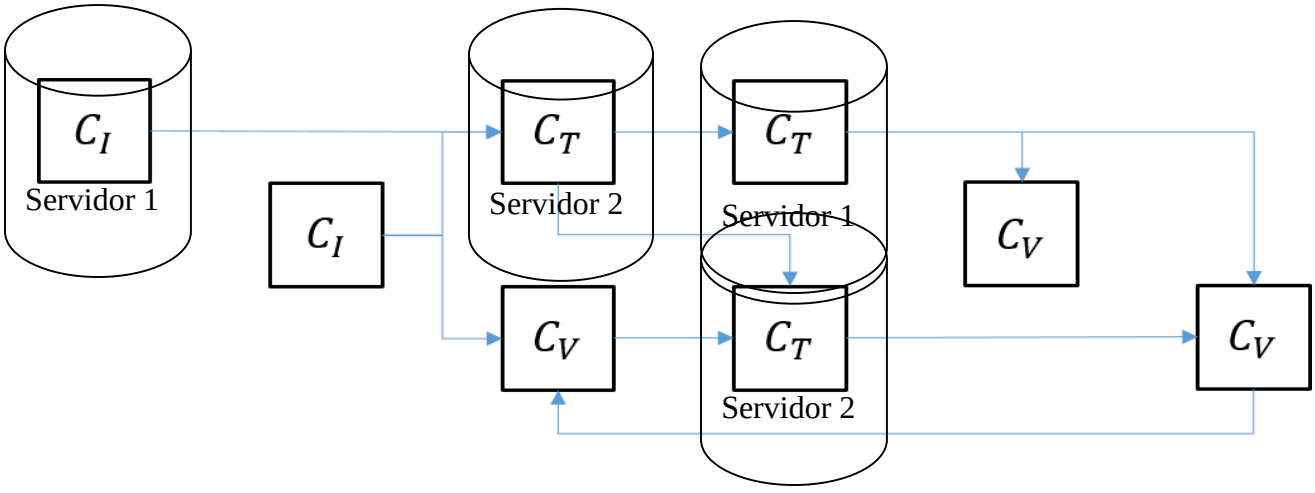

Figura 2.2 Estructura general del flujo de análisis del modelo distribuido

Ejecución del flujo de análisis

De forma general, cada componente $c_i \in C_A$ tiene asociada información que conforma su estado actual s_i, el cual determina su comportamiento frente a las entradas que reciba el componente a partir de otros componentes que se encuentren conectados a este, o de la interacción recibida directamente por parte del analista en su papel como usuario. Este estado incluye, además de características propias al componente, su estado de activación actual, o sea, si reacciona o no ante las entradas recibidas, así como su disponibilidad a ser ejecutada en el lugar designado por el usuario, que estaría dado en caso de ser ubicado en un recurso remoto por las posibles dificultades a las que se pudiera enfrentar el mismo, como por ejemplo un error interno del software en el lado del servidor, lo que en caso de no avisar al modelo a tiempo pudiera conllevar a la corrupción de los datos, colapso del modelo, etc.

El comportamiento de un componente puede por tanto ser definido como una función en ramas B_i de la entrada recibida por el componente (I_i), su estado actual (s_i) y la interacción del usuario (U) de la forma siguiente (Mollineda, 2017):

$$B_i(I_i, s_i, U) = \begin{cases} B_{Si}(s_i), I_i = \emptyset \wedge U = U_{exe} \\ B_{Ii}(I_i, s_i), I \neq \emptyset \wedge U = \emptyset \\ B_{Ui}(I_i, s_i, U), U \neq \emptyset \end{cases}$$

El resultado de la evaluación de esta función, equivalente a la ejecución del componente, es siempre una lista de vistas de una fuente de datos $B_i(I_i, s_i, U) = L_k = (v_{k1}, v_{k2}, \dots, v_{kt})$.

Las vistas de datos son la estructura en la que se considera que se mueven los datos a través del flujo de análisis y la naturaleza de las entradas recibidas por los componentes, siendo susceptibles de ser creadas o modificadas de acuerdo con la interacción o función de estos. Formalmente, dada una fuente de datos externa al componente F_d para la cual se encuentra disponible un conjunto de dimensiones Dm_d y un conjunto de datos Dt_d, una vista de datos v_d correspondiente a F_d es un subconjunto de los datos disponibles en Dt_d en base a la selección de un conjunto de propiedades $Dm_{v_d} \subset Dm_d$. Los datos serían proyectados sobre este subconjunto mediante la aplicación de una función $p_{v_d}: Dm_d \rightarrow Dm_{v_d}$ y la función de pertenencia $b_{v_d}: Dt_{v_d} \rightarrow \{True, False\}$. Para los componentes de entrada, F_d es una fuente de datos completamente externa no solo al componente sino al sistema en su conjunto, mientras que para los demás componentes casi siempre se tendría que $F_d = I_i$ (Mollineda, 2017).

El papel de estas funciones es reducir las propiedades consideradas para un dato a las que se contendrán en la vista resultante y determinar si el dato en particular pertenece o no a esa vista de datos. Normalmente el caso es que b_{v_d} es en cierta forma equivalente a B_i.

Como punto a considerar en una implementación real del modelo, es necesario tener en cuenta que alguna representación de la función que generó la selección de propiedades que pertenecen a la vista y metadatos correspondientes, por ejemplo, al rol de cada una de esas variables deberían estar disponibles para ser usadas por los demás componentes. Conceptualmente, el modelo asume que esta información se encuentra universalmente disponible para todos los componentes y que estos la toman en consideración a la hora de efectuar sus salidas.

Aunque se asume que siempre se produce una salida por parte de un componente, se contempla la posibilidad de que la lista generada sea vacía ($L_k = \emptyset$). Esto refleja apropiadamente el comportamiento que podría tener un componente cuya función sea filtrar las entradas recibidas en base a algún criterio: si para una entrada en particular no existen datos que cumplan las condiciones que representa el componente, entonces no se produce ninguna salida (Mollineda, 2017).

De igual manera, la ejecución del componente puede, como efecto secundario, modificar su estado interno, de tal manera que ante la misma entrada se pueda obtener un resultado diferente dependiendo de la naturaleza del componente y las ejecuciones anteriores. Esto incluye la posibilidad de ignorar entradas ya procesadas anteriormente, o refinar resultados anteriores en base a conocimiento obtenido con posterioridad.

Realizando un análisis de la función planteada que define el comportamiento de cada componente podemos entender el papel de cada una de las ramas dentro de la misma, pero como no es objetivo hacer un nuevo análisis del modelo de (Mollineda, 2017) se concentrará el interés en la última (B_{Ui}), que define la interacción con el usuario. Es aquí donde se da más frecuentemente la ejecución del componente. Esta interacción produce alteraciones en su estado interno.

Es en esta rama de la función donde se concentran la mayoría de los casos en los que la salida del componente en respuesta a una entrada es vacía, como resultado casi siempre a interacciones que tienen como objetivo configurar el componente.

A la configuración de ese componente se le añade otro detalle, que es la ubicación donde se desea que se procese la información inherente al componente. Esta tarea se le asigna al usuario, aunque

sería ideal el diseño de un modelo donde se realice de forma automática, pero esto requeriría un estudio fuerte de balances de carga y otros temas, lo que conlleva a un alto grado de complejidad, por lo que se optó por la primera variante. En la **Figura 2.3** se muestra como el usuario interactúa con esta característica, y para el caso particular del ejemplo allí mostrado, el usuario determina que el nodo Kmean seleccionado deber ser ejecutado en el servidor1.

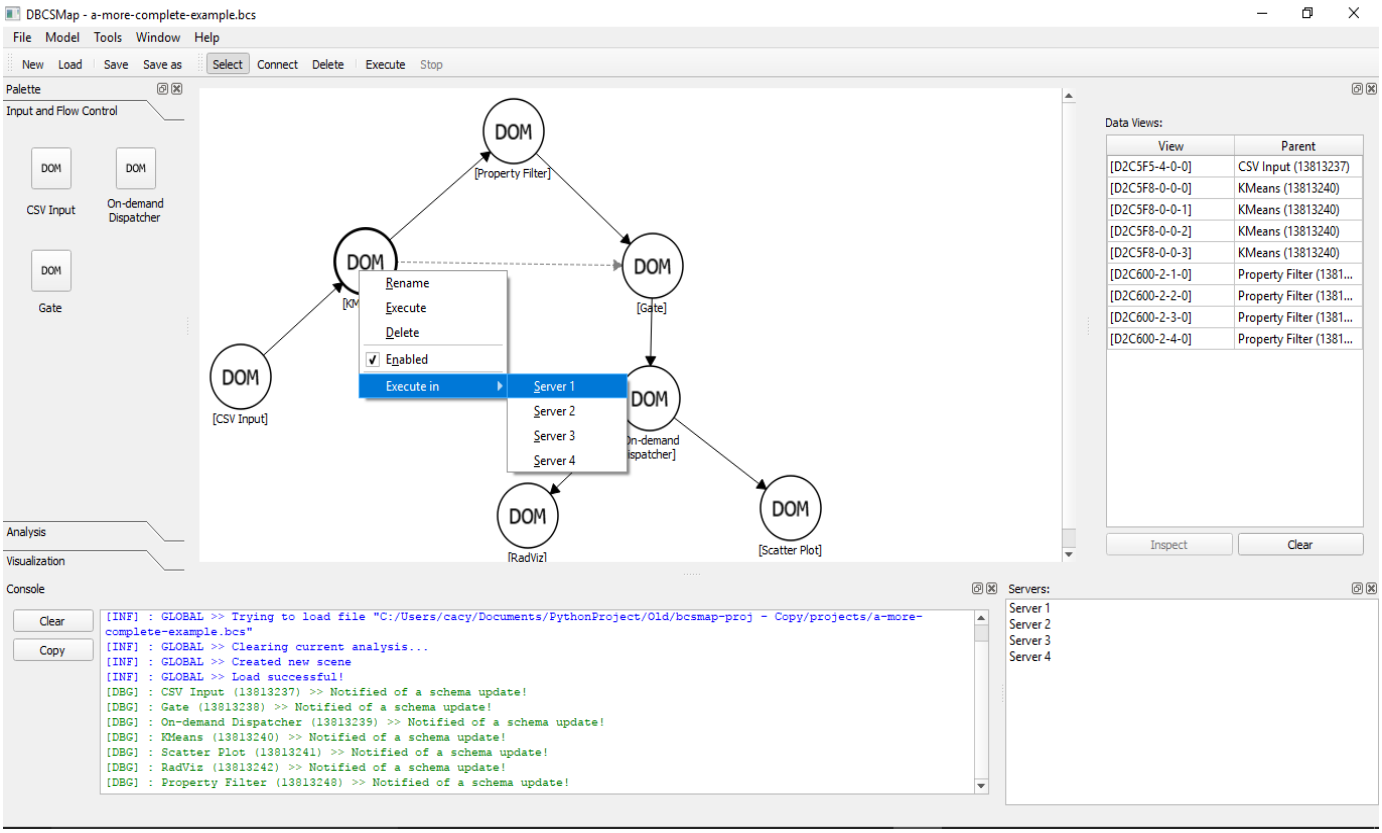

Figura 2.3 Asignación del componente a un servidor

El modelo al igual que su predecesor contempla dos escenarios para iniciar su ejecución, dependiendo de si esta es global o detonada a partir de un componente específico. En el primero de los casos, se produce la ejecución simultánea de todos los componentes de entrada existentes en el sistema, transfiriendo su salida a cada uno de los componentes que se encuentren directamente conectados a estos.

Esta primera salida debe consistir normalmente de una única vista correspondiente a la totalidad de los datos existentes en la fuente manejada por el componente de entrada que la generó, teniendo en cuenta los filtros y condiciones que este haya impuesto (representados por su estado interno).

En el segundo caso, la ejecución puede empezar a partir de una localización específica en vez de detonar todas las entradas al mismo tiempo, mediante la interacción del usuario. Para que la ejecución no se limite únicamente al componente inicial, este debería pertenecer al conjunto C_I. En esta forma de ejecución es menos probable que la ejecución se propague por todo el modelo, siendo por tanto ideal para trabajar separadamente con subprocesos de análisis.

En la **Figura 2.3** el ejemplo mostrado puede servir para explicar lo anterior, y es que para ese ejemplo si ejecutamos el modelo de forma global, la ejecución comenzaría por el nodo ***CSVinput*** y luego se propagaría hacia el resto, mientras que si se selecciona la segunda forma de ejecución, se tendría que activar el mismo nodo pero ya de forma manual, no siendo el caso para el ejemplo de la **Figura 2.4**, que en presencia de la primera selección se activan dos flujos de ejecución (siempre que estén configurados los componentes), mientras que de lo contrario seleccionaríamos el flujo que se desea que sea ejecutado.

Una vez comenzado el flujo, la ejecución continúa propagando los datos a través de todos los componentes que se hallen conectados. En el caso de componentes pertenecientes al conjunto C_T, su ejecución puede traer como consecuencia que una entrada recibida sea convertida en múltiples vistas, de ahí la necesidad de que el caso general contemple que la salida de los componentes siempre se trate como una lista.

Para permitir la circulación de los datos es necesario crear un mecanismo nuevo que sea capaz de gestionar y mantener el correcto funcionamiento del sistema en la red, lo que implica una modificación del flujo de ejecución. Básicamente se está en presencia de una red de Petri sobre un sistema distribuido que emplea como arquitectura de software la distribución cliente/servidor, lo que posibilita tener cada nodo un servidor distinto de la red.

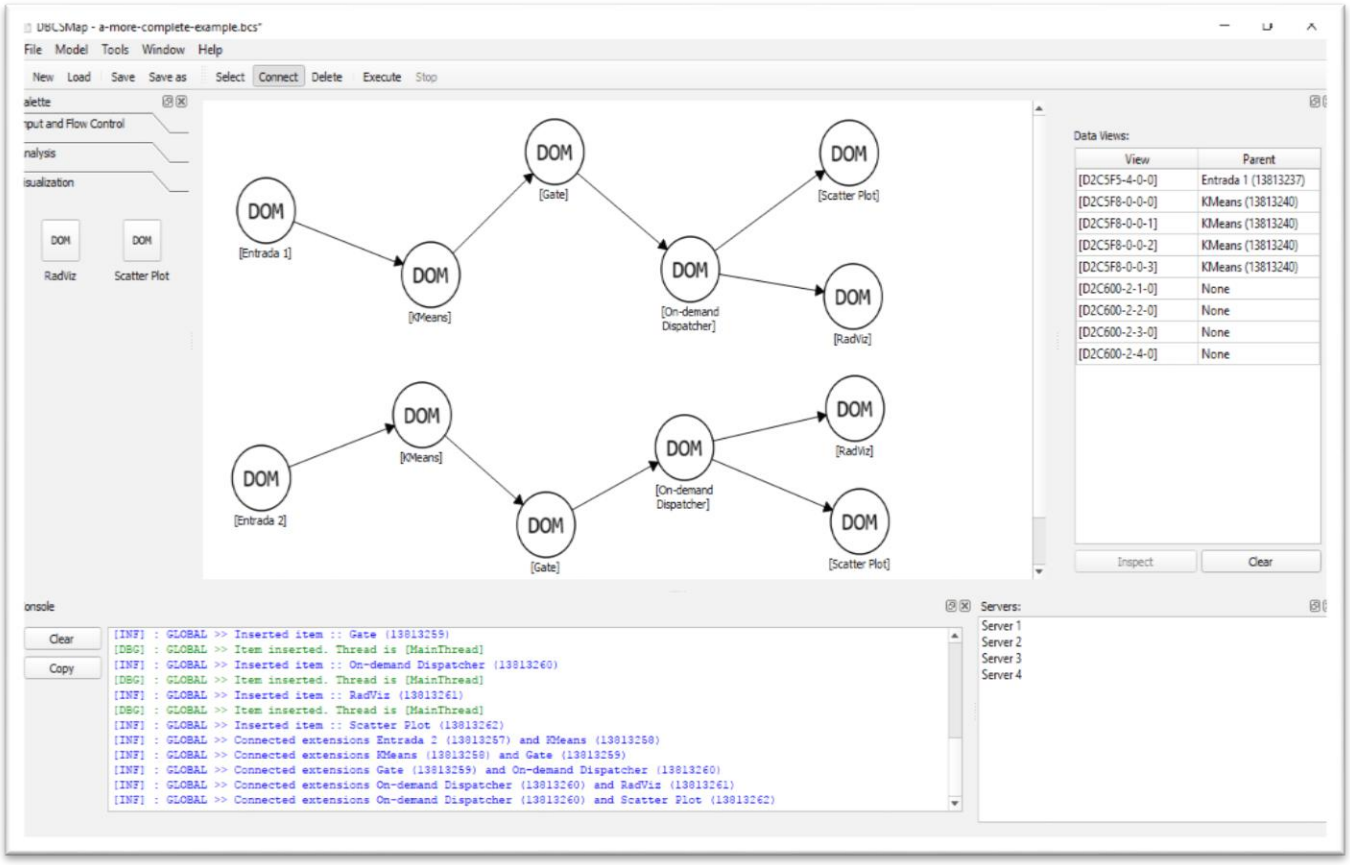

Figura 2.4 Ejemplo con dos flujos de ejecución

Para permitir el flujo de datos se mantiene la estructura de datos que alberga la información inherente al modelo, la cual es utilizada para enviar los datos al servidor que contenga el nodo que requiere ser ejecutado.

Al comenzar el flujo de ejecución, el sistema tiene que identificar si el nodo que solicita permiso para ejecutarse está local o remoto, en caso de que este local procede a ejecutarlo tal y como lo haría el modelo (Mollineda, 2017) y en caso contrario se ve nuevamente ante una encrucijada donde identifica como el modelo debe tratar el flujo: Secuencial, o por *frames*[16], cómo será tratado es una decisión que le corresponde al especialista, puesto que es el que decide la forma de configuración general del sistema, teniendo en cuenta si los datos con los que está trabajando pueden ser tratados por *frames*. Independientemente de cual sea la selección se establece comunicación con el servidor seleccionado y se le notifica qué nodo se le ha asignado para su

[16] Marcos, fragmentos de los datos

ejecución, puesto que cada servidor mantiene una copia de cada nodo, por consiguiente, le envía todos los datos que son necesarios (ejemplo: configuración del componente) y pasa a esperar de manera no bloqueante a que el servidor le entregue una respuesta, la cual puede ser de terminal o parcial (con respecto al flujo diseñado). En caso de ser parcial publica su resultado y continúa esperando por el resto de *frames* para realizar la misma acción. Una vez que el nodo ha concluido su ejecución (parcial o total) comunica al sistema de sus resultados y los envía al resto de componentes que estén conectados a él, en caso de que los resultados fueran parciales continuaría con el procesamiento de la información restante.

Es importante notar que este modelo de ejecución asume que los componentes procesan una entrada a la vez en el mismo orden en el que fueron recibidas, reaccionando a un estímulo externo (llegada de una entrada proveniente de otro componente o interacción del usuario), por lo que en principio la ejecución de cada componente es asíncrona (Red de Petri).

El estado de activación de un componente o conexión incluido en su estado interno puede tener un impacto significativo en la forma final que adquiere el flujo de análisis. Esto se debe a que un componente o conexión inhabilitado prevendrá que la ejecución se extienda más allá de él si no existe un camino alternativo. Básicamente, cambiar el estado de un componente a inactivo implica sustituir completamente B_i de tal manera que el resultado de evaluar esta siempre sea vacío, independientemente de la entrada recibida (Mollineda, 2017).

Es importante tener en cuenta principalmente al momento de la implementación que flujo de análisis puede que no se realice de la forma que fue configurado. Esto se evidencia si en el momento de ejecución alguno de los servidores tiene un error o simplemente se cae, el sistema tiene que ser capaz de redireccionar el flujo hacia otro servidor sin la intervención del usuario, esto posibilita la recuperación de errores.

Formas de interacción

Las interacciones del usuario con el modelo son varias, puesto que posee todas las heredadas del modelo de (Mollineda, 2017) y se le añaden algunas nuevas características del modelo actual, por tal motivo no se hará un profundización en las formas de interacción heredadas, solo aquellas que sufren cambios con las modificaciones deseadas.

A pesar de haber sido mencionado indirectamente, resulta importante definir apropiadamente las formas en las que el modelo contempla la interacción del usuario. Su forma más básica ya había sido abordada y se expresa en la alteración de los parámetros representados en el estado interno de un componente, lo que en otros términos se refiere a su configuración. Hay que tener presente además que, en el caso de una acción que responda puramente a un cambio de configuración, la salida correspondiente por parte del componente es vacía. Sin embargo, el efecto de la interacción con el usuario descrito en B_{Ui} no se encuentra limitado estrictamente a este tipo de acciones, este incluye además los casos donde la entrada por parte del usuario sea imprescindible para el funcionamiento de un componente, así como para contribuir con su conocimiento en la realización de las funciones de este.

Es vital tener en cuenta que el papel principal del analista, como la palabra lo dice, es el de entender y analizar los resultados obtenidos, tanto parciales como finales, en caso de que sean parciales puede que analista tenga que intervenir para una obtención de mejores resultados.

Un ejemplo interesante que pone de manifiesto la naturaleza más general de esta interacción puede plantearse a partir de una situación con un componente que realice alguna forma de clasificación: si el componente no cuenta con criterios suficientes para dar dicha clasificación con un grado de certeza suficiente, la decisión final es delegada al analista. Como resultado de esta interacción, el criterio que fue empleado para romper el empate existente en este caso puede ser empleado para modificar el estado interno del componente. De esta manera, entradas subsiguientes de casos similares serán eventualmente clasificadas sin necesidad de la interacción directa del usuario, habiendo efectivamente incorporado su conocimiento a la solución del problema (Mollineda, 2017).

La interacción directa con las visualizaciones contenidas en los componentes que pertenecen al conjunto C_V es una de las más intuitivas, expresándose sobre todo en forma de selección o refinamiento de conjuntos de datos. En el caso de los componentes de visualización contenidos en representa no solamente los parámetros de la configuración del componente, sino que debería contener además las vistas de datos que han transitado por ellos o información extraída a partir de estas (datos estadísticos, por ejemplo). Esta información adicional es la que se emplea para crear la visualización correspondiente al componente.

Una de las interacciones más poderosa disponible para el analista en este caso se refiere a la posibilidad de ajustar y rediseñar el flujo de análisis de acuerdo a los resultados obtenidos con cada ejecución. La distinción entre ajuste y rediseño se realiza en base a si la interacción modifica o no la estructura del flujo de análisis existente.

En el caso de un ajuste, se refiere por lo general a un cambio en el estado de activación de algún conjunto de componentes o conexiones. Como consecuencia de este cambio, la ejecución sigue una ruta diferente, posiblemente cambiando las entradas recibidas por los componentes en una etapa posterior del flujo. Sin embargo, la estructura subyacente se mantiene intacta y el cambio puede ser revertido fácilmente, puesto que los componentes y la información contenida en su estado todavía se encuentra disponible.

Cuando se trata de una operación de rediseño, se considera que la estructura del flujo de análisis actual es alterada, ya sea mediante la inserción o eliminación de componentes o conexiones. Particularmente en el caso de la eliminación, a diferencia de cuando solamente se cambia el estado de activación del componente, el estado asociado al componente se pierde irreversiblemente cuando este es eliminado. Esto implica que, incluso si se vuelve a insertar otro componente del mismo tipo que el que fue eliminado, los resultados obtenidos a partir de la ejecución del flujo pueden ser distintos de los que se habrían obtenido originalmente a menos que el estado interno del componente pueda ser reproducido perfectamente.

Además de todo lo mencionado hasta el momento, el especialista debe hacer una distribución de su flujo, es decir, sabiendo que el modelo permite la ejecución de sus nodos tanto locales como remotos, reside en el analista la decisión si utiliza o no los recursos que se le brindan. Es válido tener en mente que sea cual sea la decisión, tendrá un impacto sobre la eficiencia, ya sea para bien o para mal, debido, por ejemplo, a que si se desarrolla un flujo donde el nivel de cálculo necesario para su ejecución excede las posibilidades de un ordenador personal la decisión más sabia sería distribuir sus nodos por la red, pero como se ha mencionado el usuario puede tomar la decisión de dejarlo totalmente local.

En contraparte al ejemplo anterior se puede ver el caso donde se trabaja con pocos datos y se requiere poco poder de cálculo, para este caso en particular la mejor distribución posible sería dejar todo el modelo local, puesto que si se distribuye alguno o todos sus nodos se estaría aumentando

el tiempo que toma el sistema en ejecutar el flujo, que se traduce en la adición del tiempo que toma le envió y recepción de la información hacia sus respectivos servidores de ejecución.

A pesar de las implicaciones de lo mencionado anteriormente, la distribución de un componente no es más que la modificación de su configuración, cuando se distribuye básicamente se le asigna un lugar de destino, lugar donde se ejecutará.

Otra de las interacciones es ya relacionada directamente con el flujo de ejecución, y es que, como se mencionó en el epígrafe anterior, el analista debe seleccionar el modo en que se va a tratar los flujos de datos a través del sistema. Como es de esperar cada selección tiene sus particularidades, pero dada que la novedad se presenta en los flujos por *frames,* solo se mostrará un ejemplo de su utilidad.

A la hora de trabajar con metadatos, donde la información a minar posee tamaños significativos incluso para una supercomputadora y se tiene una serie de recurso esperando por otros para procesar estos datos, es necesario encontrar una forma más eficiente de trabajar estos datos, es por tal razón que se le da la posibilidad al usuario de que pueda realizar todo el proceso mediante *frames.* Al trabajar los datos de esta forma le permite al sistema ejecutar nodos con información parcial, que de lo contrario estarían esperando a que le llegue toda la información, y por tanto desperdiciando tiempo.

En caso de trabajar de esta manera el especialista debe seleccionar el tamaño de los *frames* en base a sus conocimientos para que no se afecte la información global o lo haga lo menos posible, es válido aclarar que no siempre es conveniente su utilización, así como no todos permiten su manipulación de esta forma, sin afectar la veracidad de la información extraída.

Finalmente, la forma de interacción restante se relaciona directamente con la preservación y externalización del conocimiento obtenido a partir del proceso de análisis. Esta se realizaría a través de los componentes, pero actuaría sobre las vistas de datos que circulan por el sistema. La idea es que los conjuntos de datos con características interesantes determinados a través de la interacción puedan ser recuperados y reutilizados. En una implementación real del modelo, esto podría verse como guardar a un soporte persistente (fichero, base de datos, etc.) las características que definen a esas secciones interesantes dentro de los datos.

De esta forma se refuerza la naturaleza iterativa del modelo y la retroalimentación en base al conocimiento extraído, permitiendo que un mismo diseño de flujo de análisis sea aplicado sucesivamente para entradas diferentes incorporando en cada ocasión nuevas entradas correspondientes al conocimiento extraído de iteraciones anteriores.

Teniendo en cuenta todas las características discutidas, el modelo propuesto cumple con todos los requerimientos que fueron identificados en la sección anterior, resultando, de hecho, lo suficientemente general como para ser adaptado a otros escenarios de análisis fuera del objetivo general perseguido originalmente.

2.2 Diseño de componentes

Habiendo definido las características del modelo en su conjunto, esta sección se propone abundar en particularidades correspondientes a tipos de componentes específicos. Por ejemplo, se desarrolla más profundamente el comportamiento esperado por parte de los componentes, la forma que deberían tomar sus interacciones y posibles casos especiales a ser considerados. Particularmente, se pretende que estas propuestas de diseño puedan servir como guía para una implementación del modelo.

Componentes de entrada

Previamente se había discutido la naturaleza de las vistas de datos que constituyen el elemento fundamental del flujo de análisis que se trabaja en el modelo. Dado que los componentes de entrada son el medio por el que dichos datos entran a formar parte del flujo de análisis, es natural que sean los más directamente involucrados en la creación de dichas estructuras y en donde se han de tomar la mayor parte de decisiones con respecto a la estructura de los datos.

Estos componentes son los puntos de entra al flujo de ejecución, siendo en varios casos los encargados de obtener la información, que a posteriori convierten en vistas de datos, de disimiles fuentes externas tales como archivos CSV (por las siglas en inglés de *Comma Separated Value*) y bases de datos. Dichos componentes tienen que permitir que los usuarios seleccionen donde van a ser ejecutados: si remoto o local, por lo que al momento de su diseño hay que tener presente que en caso de que la selección local los componentes tienen que identificarlo y mandar a ejecutar como el modelo de (Mollineda, 2017), mientras que en caso contrario tiene que recoger la información necesaria para actuar y enviarla a su homólogo en el servidor, un ejemplo de ello sería

identificar la dirección a un archivo ***CSV*** que se quiera importar y enviarla al servidor para que este proceda a la carga del mismo.

Componentes de análisis

Al contrario que para los componentes de entrada o los de visualización, la función principal es la de procesar y transformar las salidas generadas por otros componentes. Debido a que esta transformación debe necesariamente asumir una cierta estructura de los datos con los que se trabaja (que ya ha sido mencionada en algunos momentos), pero las vistas de datos que arriban a un componente no son necesariamente homogéneas en su estructura, los componentes de análisis se ven afectados por el mismo problema que los componentes de entrada en cuanto a la necesidad de definir una política para tratar con datos para los cuales la operación definida en el componente no sea aplicable.

En el caso particular de los componentes de análisis, las dos opciones más sencillas son propagar la vista de datos hacia los demás componentes conectados en el flujo sin alterarla, o impedir que esta continúe propagándose. La elección de una opción sobre otra es muy dependiente del escenario de análisis que se esté trabajando, por lo que en última instancia sería conveniente, para una implementación concreta, darle al analista la posibilidad de escoger el comportamiento que estime más apropiado (Mollineda, 2017).

La ejecución de un componente para una entrada puede traer consigo un cambio en el estado interno de este, lo que se refleja en que una futura ejecución empleando la misma entrada como argumento podría producir una salida diferente. Este es el comportamiento más intuitivo para componentes que posean alguna forma de aprendizaje o cuyo funcionamiento esté basado en un criterio aleatorio generado cada vez que se ejecuta el componente, ejemplo: componentes que representen técnicas de agrupamiento o clasificación con parámetros susceptibles de ser refinados automáticamente o con un elemento no determinista en su función.

El igual que los componentes de entrada, estos identifican la decisión del usuario y toman las mismas acciones, variando solamente la recopilación de la información que será enviada al servidor, puesto que esta es muy particular de cada componente. Un ejemplo sería que se seleccione un componente *Kmean* para que sea ejecutado de forma remota, el nodo establecería conexión con el servidor elegido y le notifica que va a ejecutar un nodo *Kmean*, para el cual le envía su configuración (creada por el analista) acompañada de los datos a ser procesados. Ya

obtenidos los resultados los publica para el usuario y los envía al siguiente nodo en caso de ser necesario.

Componentes de visualización

Desde el punto de vista de la interacción con el usuario, son los componentes de visualización los que tienen el mayor grado de responsabilidad. Debido a ello, la naturaleza y extensión de las principales interacciones disponibles para un componente de visualización ya fueron discutidas en la sección correspondiente a las formas de interacción con el flujo de análisis.

La principal tarea de los componentes de visualización es mostrarle al analista los datos obtenidos por el modelo de una forma que tenga más sentido para él y sea más sencillo de analizar. A pesar de que el procesamiento fuerte se encuentra en los componentes de análisis puede ocurrir el caso de que se necesite un poder de cálculo un poco más potente que el ofrecido por una PC para procesar un volumen de datos considerable. Obviando el caso anterior se decidió que estos componentes no serán modificados, lo que se traduce a que no será posible su ejecución en la red, aunque se sugiere para futuras implementaciones la migración de este hacia un sistema de red.

2.3 Consideraciones sobre componentes existentes

En este punto ya son evidente las nuevas características del modelo, así como un comportamiento general de cómo se desea que funcione, tal nivel de entendimiento del sistema conduce a pensar que son necesarios ciertos cambios en los componentes existentes para que se garantice la compatibilidad.

De la forma que están diseñados los componentes existentes posibilitan que cada uno se encargue de su ejecución, haciendo uso de la concurrencia, lo que permite que estén varios nodos estén procesando información a la vez. De modo que todos los nodos que cumplan con determinas restricciones pasan a procesar sus datos. Si bien este comportamiento es conveniente en el marco de un modelo local, no lo es tanto para uno repartido por la red.

Para que lograr que sean compatibles los dos flujos de ejecución, y el nodo sea capaz de ejecutarse en cualquiera de los ambientes se debe hacer una unión de ambos conceptos. Es decir, cada nodo debe ser modificado para que identifique cuál es su posición actual (local o remota) y es comporte en base a su selección.

En caso de estar local procede de la misma manera prevista por (Mollineda, 2017) y de lo contrario necesita aprovisionarse de toda la información necesaria inherente al servidor, que sería la ubicación del mismo en la red, una vez que este todo listo invoca al servidor y le notifica que tiene que ejecutar un nodo con sus características, por lo cual al invocar ese servidor además de suministrarle el flujo de datos tiene que enviar su configuración.

Luego de obtener los resultados del servidor el componente sigue un comportamiento como si todo el proceso se hubiera realizado local, esto permite que las modificaciones realizadas al modelo sean totalmente transparentes para la lógica del modelo de (Mollineda, 2017).

2.4 Servidor del sistema

La mayoría de los servidores de hoy día funcionan para dar servicios a múltiples usuarios, como es el caso de los servidores de Google, Amazon, Microsoft, GitHub, etc. Todos con diferentes servicios, pero con el mismo propósito, llegar a la mayor cantidad de usuarios posible con el mejor rendimiento que puedan, que son las mismas metas que se aspiran con el modelo planteado. Para que el servidor sea capaz de manejar varios usuarios simultáneamente es necesario que este sea multihilo, es decir, cada conexión que se establezca será procesada en un nuevo hilo de ejecución distinto al principal. A pesar de que existe una variante al servidor multihilo, que serían los servidores ***multiforking***[17], este no es funcional para sistemas que operen con *Windows* como sistema operativo.

De esta manera se da solución a la problemática de la disponibilidad permanente del servidor, y por consiguiente el manejo de varios usuarios a la vez, pero ya resuelto este problema surge uno nuevo, y es que cada una de estas conexiones se hizo con el objetivo de realizar un procesamiento determinado, que puede ser tan complejo y pesado como se quiera. Por lo que podría llegar incluso a paralizar el sistema, razón que hace nuevamente acudir a la concurrencia como modo de solución. Como se ha mencionado cada servidor tiene una función asociada a un componente en específico, función que será invocada desde el modelo local según sea necesario. Dicha función tiene que emplear alguna de las técnicas de concurrencia mencionada en el CAPÍTULO 1, pese a que todas

[17] Bifurcación, consiste en la duplicación o bifurcación del código a partir de ese punto

las variantes son factibles en mayor o menor grado (respecto a eficiencia) la seleccionada ha sido los procesos ligeros de ***PyQt*** por la simplicidad que conlleva su uso.

Los archivos en los que están implementados los nodos deben ser ubicado en la carpeta del proyecto correspondiente a la clasificación que le da el modelo en cuanto a su función dentro de este, para que de esta manera el sistema sea capaz de cargar las funciones asociadas a los módulos dinámicamente.

Un servidor que brinde determinado servicio no es de utilidad si no está accesible o visible para los usuarios, es decir, no cumple ningún objetivo tener servidor potente brindando un servicio necesario si este es invisible para el mundo, si no hay forma ubicarlo. Para el caso presente la decisión es crear un servicio que posibilite la publicación de los servidores en la red, por lo que el modelo tiene en todo momento los servicios disponibles. Este servicio estará ubicado en algún servidor seleccionado por el usuario que configure la red.

Cada servidor tiene la posibilidad de estar visible o no, en caso de estar visible podrá ser descubierto por cualquier cliente que emplee el modelo, de lo contrario seguirá estando disponible pero solo será usado por aquellos usuarios que conozcan su dirección. Esta característica brinda la posibilidad de que los usuarios puedan tener a su disposición servidores personales o a nivel de institución. Para ello es necesario añadir los servidores manualmente al sistema, en cuyo caso sistema obtiene la responsabilidad de mantener actualizada la disponibilidad del servicio. Esto evitaría algunos errores que se darían por ejemplo al intentar conectar con un servidor que en realidad ya no está disponible. Por otro lado, el servidor tiene que ser capaz de notificar al modelo de cualquier error que pudiera poner en riesgo la integridad o veracidad de la información.

Como fue mencionado en la estructura general del modelo, la relación entre los servidores simula una red ***P2P***, es decir, al momento de un servidor pasar sus resultados a otros este lo hace mediante conexiones directas. Y lo que se pretende es obtener todas las ventajas de una red de pares.

2.5 Mecanismo de escalabilidad

La conectividad mundial a través de internet es en los tiempos actuales sumamente común, y es que prácticamente todas transacciones de datos, información y cualquier otro tipo de interacción se realiza mediante internet, por lo que es importante tener en cuenta a la hora de desarrollar un sistema distribuido la escalabilidad del mismo.

Según (Tanenbaum and Steen, 2008) un sistema distribuido debe hacer que los recursos sean fácilmente accesibles; debe ocultar de manera razonable el hecho de que los recursos están distribuidos por toda la red; debe ser abierto, y debe ser escalable.

La escalabilidad de un sistema se puede medir de acuerdo con al menos tres dimensiones (Neuman, 1994). Primero, un sistema puede ser escalable con respecto a su tamaño, lo cual significa que podemos agregarle fácilmente usuarios y recursos. Segundo, un sistema escalable geográficamente es aquel en el cual usuarios y recursos pueden radicar muy lejos unos de los otros. Tercero, un sistema puede ser escalable administrativamente.

Aunque desafortunadamente, con frecuencia un sistema escalable en una o más de estas dimensiones exhibe alguna pérdida de rendimiento al escalarlo (Tanenbaum and Steen, 2008) pero ese tema ya fue tratado.

Cuando un sistema requiere ser escalado, hay que resolver problemas inherentes a las tres clasificaciones mencionadas. Consideremos primero la escalabilidad relativa al tamaño y número de usuario. El problema es que, si más usuarios o recursos requieren soporte, con frecuencia se enfrentan limitaciones en los servicios, datos, y algoritmos centralizados. Por ejemplo, muchos servicios están centralizados en el sentido de que se implementan en términos de un solo servidor que se ejecuta en una máquina específica localizada en el sistema distribuido, con este esquema resulta evidente: el servidor se puede convertir en un cuello de botella mientras el número de usuarios y aplicaciones crece.

Según (Tanenbaum and Steen, 2008) incluso si contamos con una capacidad de almacenamiento y procesamiento ilimitada, en algún momento la comunicación con el servidor prohibirá cualquier crecimiento posterior. Esto está dado además de la posible creación de cuellos de botella, a medida que aumenta la distancia crece la latencia de la red que si viene acompañada de un sistema que trabaja de forma sincrónica traería consecuencias nefastas.

Esta problemática no afecta a este modelo en particular puesto que no se accede a única base de datos centralizada, de hecho, el acceso a los datos es tan general que puede cargarse incluso desde un archivo en la máquina local del usuario.

Por otro lado, el aumento de recursos (servidores disponibles) es relativamente fácil, solo conlleva la selección del equipo que se desea usar (como servidor), instalar el módulo diseñado para el

servidor y realizar unos pasos básicos de configuración, y eso es todo lo necesario para agregar nuevos recursos al modelo, mientras que para aumentar el número de usuarios, estos solo necesitan tener aplicación desarrollada y el servidor de publicación (mencionado anteriormente) se ocupa del resto, puesto que además de permitir la existencia de varios servidores, el módulo instalado en cada uno permite la interacción con múltiples clientes.

En el caso de la escalabilidad geográfica por su lado tiene sus propios problemas según (Neuman, 1994) al escalar un sistema geográficamente sus introducen a los sistemas algunos problemas tales como un aumento en las probabilidades de fallo (disminución de fiabilidad), puesto que las conexiones son más propensas a caídas y errores a medida que aumente la distancia en los dos extremos, además estas largas distancias entre componentes posibilitan un incremento de la latencia de las comunicaciones.

La manera que se platea de enfrentar los problemas de la escalabilidad geográfica es mediante la simulación de una red P2P entre los servidores. Como fue mencionado en el CAPÍTULO 1 dos características presentes en este tipo de redes son la robustez y la escalabilidad, condicionadas por una red de nodos que se conectan entre ellos sin la presencia de servidores.

De esta forma los datos no tienen que ser enviados a un servidor (cliente) para que este lo reenvíe luego hacia otro nodo, sino que se envían directamente a su lugar de destino y se notifica al servidor del modelo la operación realizada. Con esta variante se disminuye el tiempo de respuesta del sistema y se le da una mayor resistencia ante problemas y caídas de la red.

La conjunción de estos problemas tiene un impacto principalmente en el rendimiento lo que puede ser contradictorio con los objetivos de un sistema distribuido ya que lo que se busca es un aumento del mismo, por lo que es necesario tener presentes todos los problemas que pueden ocurrir y erradicarlos en medida de lo posible.

Una de las razones por las cuales es difícil escalar sistemas distribuidos existentes diseñados para redes de área local es que se basan en la comunicación síncrona (Tanenbaum and Steen, 2008). En esta forma de comunicación, una parte que solicita un servicio, por lo general llamada cliente, bloquea el servicio hasta que obtiene una respuesta. Por lo común, este método funciona bien en redes de área local donde la comunicación entre dos máquinas es, en el peor de los casos, de unos cuantos cientos de microsegundos. Sin embargo, dentro de una red de área amplia, es necesario tomar en cuenta que la comunicación interproceso puede ser de cientos de milisegundos, tres veces

más pequeña en magnitud, y si se le añaden los problemas mencionados de escalabilidad se obtiene una situación bastante desfavorable.

Teniendo esto en mente se mantiene el carácter asincrónico del modelo, lo que, unido a la simulación del comportamiento de una red de pares, permite obtener un modelo que facilita una mayor dispersión de sus recursos y maximizan la fiabilidad.

2.6 Conclusiones parciales

Debido a la unión de las características presentes en las arquitecturas de red y estilos arquitectónicos empleados, y la interacción del modelo con estas tecnologías, le brindan al sistema robustez y escalabilidad. Además, el apoyo de servidores distribuidos para la realización de tareas agrega una buena potencia de cálculo al sistema y que al estar acompañado de programación concurrente se optimiza la ejecución del flujo, lo que posibilita una mayor eficiencia en su procesamiento.

CAPÍTULO 3. Implementación del modelo distribuido

En este capítulo se discuten las características de la implementación desarrollada a partir del modelo propuesto en el CAPÍTULO 2. En primer lugar, se analizan los actores involucrados y los casos de uso principales de la aplicación identificados en base al diseño conceptual del modelo. Posteriormente se abordan las características y funcionalidades de las principales clases involucradas en el funcionamiento de la herramienta desarrollada. Finalmente, se presenta un ejemplo de un caso de uso real de la aplicación.

3.1 Análisis de actores y casos de uso

Debido al propósito específico de la aplicación, se puede considerar que existen dos actores para los casos de uso de esta: el analista o especialista y un administrador, el cual pudiera o no estar presente. Se asume que el analista tiene un conocimiento extenso del dominio del problema, es capaz de interpretar la información representada en los valores de las propiedades de los fármacos y está familiarizado con el uso de herramientas informáticas para el análisis, pero no necesariamente posee conocimientos de programación.

El segundo actor denominado administrador tiene un papel secundario, este se encarga de escalar el sistema agregándole nuevos servidores y chequearlos cuando ha ocurrido algún error. Puede darse el caso en el que no sea necesario su presencia, debido a que, por la simpleza de su interacción con el sistema, el propio analista puede realizar sus funciones en determinado momento. No obstante, lo ideal sería una persona con conocimiento de redes informáticas o al menos con alguna experiencia en el tema, para que trabaje con mayor eficiencia.

Por otro lado, los casos de uso para la aplicación se corresponden en su mayor parte con las interacciones que fueron descritas en el desarrollo conceptual del modelo.

Al igual que en (Mollineda, 2017) en el manejo de archivos y operaciones similares se encuentran contenidos los casos correspondientes a crear un nuevo flujo de análisis, cargar un flujo de análisis previamente guardado o guardar el flujo de análisis actual. Además, la creación, configuración y

guardado de esquemas de entrada como forma de interactuar con la estructura de los datos del flujo de análisis.

El analista es el encargado de diseñar un flujo de análisis y ejecutar el mismo. El primero de estos dos casos puede a su vez subdividirse de acuerdo al tipo de acción que se tome en particular: añadir un nuevo componente al diseño actual, eliminar un componente previamente insertado, conectar y desconectar componentes y configurar un componente, que es donde viene una nueva tarea para el usuario y es que como parte de esa configuración se encuentra la posible distribución del nodo en la red, es decir, tiene que decidir si ese componente será procesado por una máquina remota o local. También se incluye el caso de cambiar el diseño del flujo, o sea, alterar su representación visual sin hacer modificaciones que influyan en su ejecución, así como decidir cómo será tratado este flujo, secuencial o por lotes (*frames*).

En el caso relacionado con la ejecución, quizá sería más apropiado hablar en términos de cambiar el estado de ejecución del flujo de análisis, pudiendo también subdividirse en tres casos más específicos: iniciar la **ejecución global** del flujo, iniciar la ejecución localizada a partir de un componente específico y detener la ejecución del flujo (Mollineda, 2017).

En particular, la implementación realizada a partir del modelo respeta esta posibilidad: los cambios en la distribución visual del modelo, la modificación de los estados de activación de componentes y conexiones y cambios en la configuración que representa el estado interno de estos son realizables durante el proceso de ejecución del flujo de análisis, siempre y cuando el nodo modificado no se encuentre en ejecución o este de forma local, que entonces sería tratado de igual forma que en el modelo de (Mollineda, 2017).

Sin embargo, operaciones tales como la inserción y eliminación de componentes y conexiones entre estos se encuentran inhabilitados durante la ejecución del flujo de análisis y requieren que este sea detenido primero para poder ser realizadas. De igual manera, debido a que los componentes pueden modificar su estado como consecuencia de su ejecución, tampoco es posible emplear ninguna de las operaciones representadas por los casos de uso relacionados con operaciones con archivos cuando el flujo de análisis se encuentra en proceso de ejecución.

3.2 Diseño y comportamiento de la interfaz de usuario

El diseño de la interfaz de usuario para la aplicación desarrollada como implementación del modelo consiste de una ventana principal cuyos elementos principales son una paleta de componentes separados en tipos por pestañas, una consola de eventos, un área principal correspondiente a la escena de análisis y una lista con los servidores activos y visibles para el sistema.

La ventana principal de la aplicación cuenta con varios menús que permiten acceder a sus funcionalidades principales, aunque la mayor parte de estas, sobre todo aquellas que tienen relación directa con tareas como la configuración o ejecución de componentes del flujo de análisis también se hallan disponibles mediante menús contextuales. De igual manera, algunas de las opciones más comunes también se hallan disponibles en forma de barra de herramientas para permitir el acceso rápido a algunas de las funcionalidades más comunes: en muchos casos, es posible realizar una acción concreta empleando cualquiera de estas tres variantes, de acuerdo a las preferencias del usuario. En la **Figura 3.1** se muestra una captura de la ventana principal de la aplicación y los elementos de la interfaz de esta.

A continuación, se describen los principales elementos que componen la interfaz principal de la aplicación y sus responsabilidades brevemente, haciendo hincapié en aquellos que son nuevos o sufren modificaciones:

Paleta de componentes: Esta se encuentra dividida en tres pestañas, cada una contiene los componentes correspondientes a uno de los tipos de entrada, análisis o visualización. Los elementos que se muestran aquí son reflejo de las extensiones cargadas dinámicamente al iniciar la aplicación. Mediante la interacción con estos, el usuario puede seleccionar un tipo de componente para insertar en la escena de análisis actual. Este elemento de la interfaz puede ocultarse si no es necesario en un momento determinado, además de que puede estar acoplado a la izquierda (por defecto) o a la derecha de la ventana principal, o ser usado como una ventana flotante.

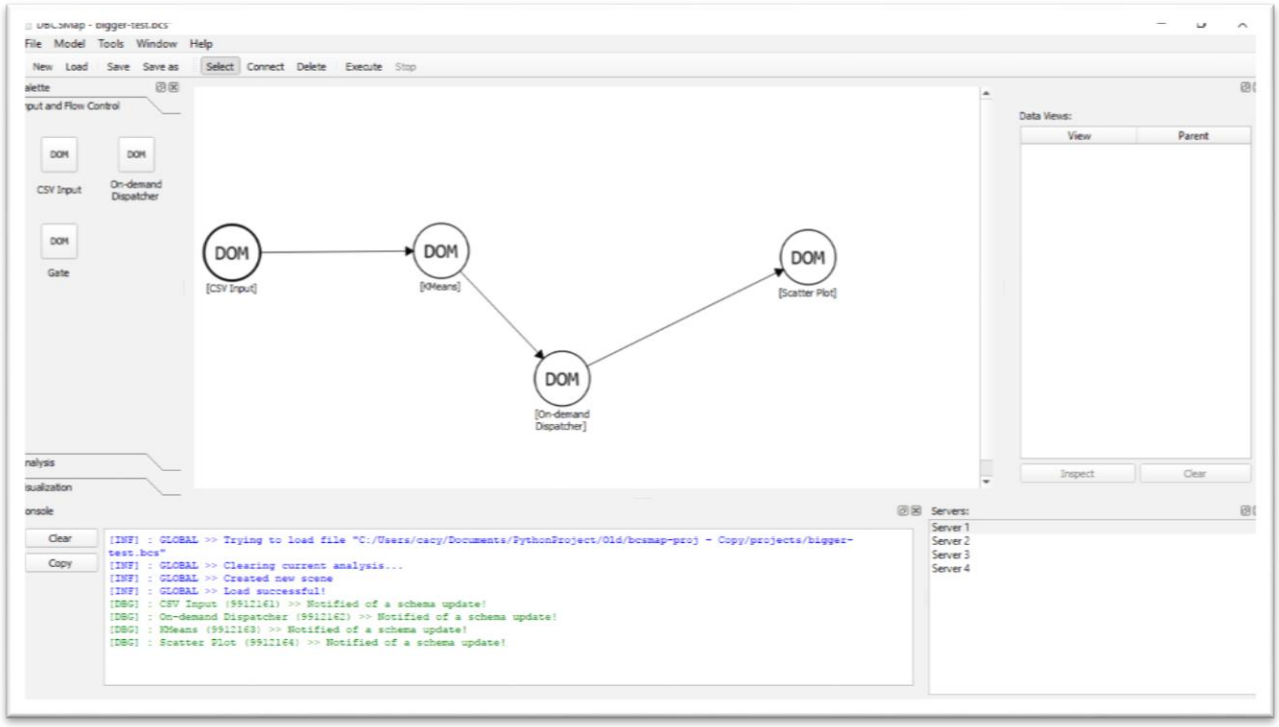

Figura 3.1 Vista general de la interfaz de usuario de la aplicación

Consola de registros: Ubicada en la sección inferior izquierda de la ventana principal de la aplicación, esta, al igual que la paleta de componentes, también puede ser ocultada o usada como una ventana flotante. La función primordial de este componente es mostrar los mensajes generados por la aplicación y los componentes en respuesta a las interacciones (con el usuario y entre componentes), así como cualquier mensaje de error resultado de alguna excepción en el comportamiento de alguno de los elementos o servidores.

Interfaces de componentes: Cada uno de los componentes o elementos que conforman un flujo de análisis poseen una interfaz asociada a ellos. Esta puede mostrarse en forma de diálogo mediante la interacción con el objeto que representa visualmente al elemento en cuestión. Por regla general, estas interfaces deben contener al menos los elementos que permitan configurar el estado interno del componente.

Ventana de edición de esquemas de entrada: Este componente, que se encuentra disponible como un diálogo accesible desde el menú de herramientas de la ventana principal de la aplicación, permite realizar la carga, guardado y edición de los esquemas de entrada que usan los componentes como medio para conocer cómo se hallan conformadas las vistas de datos con las que han de

interactuar, permitiendo definir los nombres, el tipo de variable (continua o nominal) que representa la propiedad y el rol que esta cumple. El diálogo correspondiente a esta funcionalidad puede apreciarse en la **Figura 3.2**

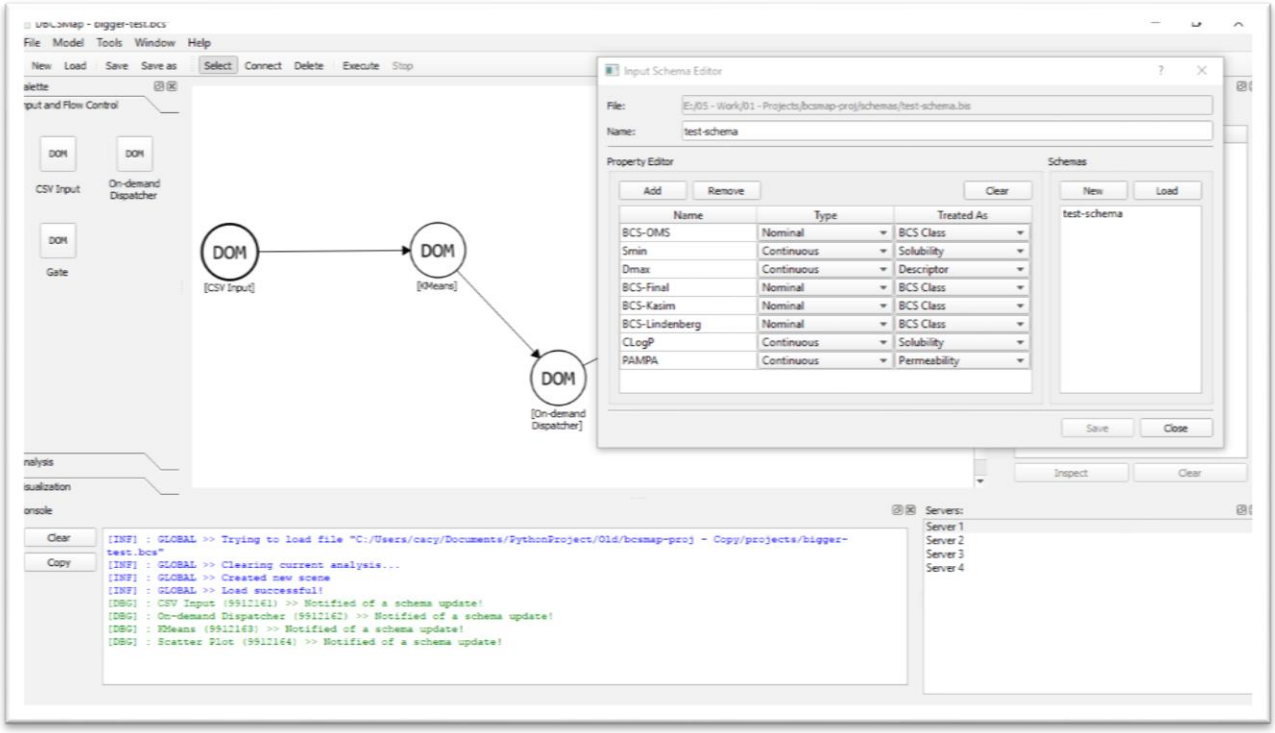

Figura 3.2 Ventana de edición de esquemas de entrada

Inspector de vistas de datos: Accesible mediante la interacción con el listado de las vistas de datos existentes en el flujo de análisis (ubicado por defecto a la derecha en la ventana principal de la aplicación), este elemento permite interactuar directamente con una vista de datos. Las posibilidades incluyen la edición de los metadatos asociados a la vista, una descripción estadística de los valores de las propiedades que se encuentran contenidas en esta.

Área principal de la escena de análisis: Este es el componente fundamental de la interfaz de usuario de la aplicación. Sobre esta área es que se diseña y construye un flujo de análisis, mostrando la representación visual de este para permitir la interacción con el usuario, además de reflejar los cambios que ocurren como consecuencia de la ejecución u otras interacciones. El menú contextual invocado al hacer *click* secundario sobre una parte vacía de esta área contiene, como ya se había mencionado, las acciones referentes a los cambios de modo que dictan el comportamiento

de la escena de análisis y las destinadas a iniciar, parar la ejecución global del flujo de análisis, así como establecer el modo en que será tratado el flujo (secuencial o *frames*).

La representación visual de los elementos del flujo de análisis se trata como un cuadrado con bordes negros sólidos, el ícono o imagen asociada al tipo de componente en el centro de este y una etiqueta identificadora en la parte inferior. El color y tipo de línea del borde del componente representan su estado de activación o ejecución: gris sólido para el estado inhabilitado y verde discontinuo para el estado correspondiente a la ejecución del componente.

Tanto las representaciones de los elementos como las de las conexiones entre estos permiten la invocación de un menú contextual desde el cual se puede inhabilitar o eliminar ese elemento. Además, en el caso de los componentes, el menú incluye la posibilidad de cambiar la etiqueta asociada a este, iniciar la ejecución a partir de ese componente y distribuir el mismo mediante la asignación de los servidores mostrados.

Lista de servidores: Se encuentra por defecto en la zona inferior derecha, aunque puede reubicarse en cualquier lugar seleccionado por el analista, y al igual que otras ventanas de la aplicación puede ser cerrada si se desea y luego activarla nuevamente mediante el menú. Su papel principal es brindar información acerca de los servidores activos y visibles en la red, identificado cada uno por una fila en la lista. También sirve como instrumento de distribución de los nodos, puesto que mediante *drag and drop* permite al analista asignarle un servidor, de una manera sencilla e intuitiva.

Añadir servidor: Otra de las interfaces disponibles desde el menú es la relacionada con adición de un nuevo servidor. Esta interfaz mostrada en la

Figura 3.3 permite agregar servidores de forma manual, así como asignarle un nombre y de un manera muy sencilla comprobar si el servidor deseado está funcional.

Servidor: Como medio para facilitar la interacción con el servidor se desarrolló una interfaz visual, mostrada en la **Figura 3.4**. Como muestra la figura, existen tres botones dedicados a iniciar, detener y reiniciar el servicio respectivamente. Se cuenta con un cuarto botón cuya función es hacer una solicitud al servidor de publicación y mostrar los resultados (otros servidores disponibles) en la consola que aparece en la zona inferior, la cual está dedicada a mostrar información genérica, que va desde el estatus del servicio hasta errores en el sistema. Por último,

en zona superior derecha se encuentra una lista conformada por los clientes que están haciendo uso del servidor.

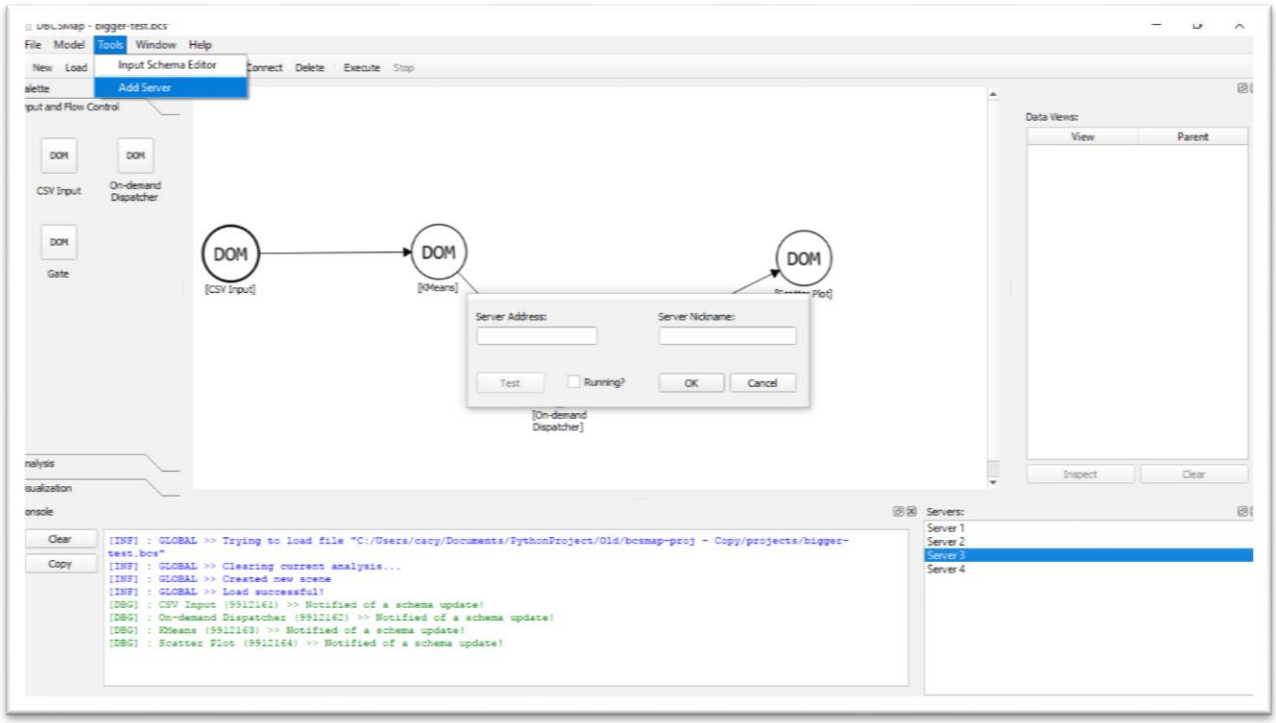

Figura 3.3 Interfaz para añadir un servidor manual

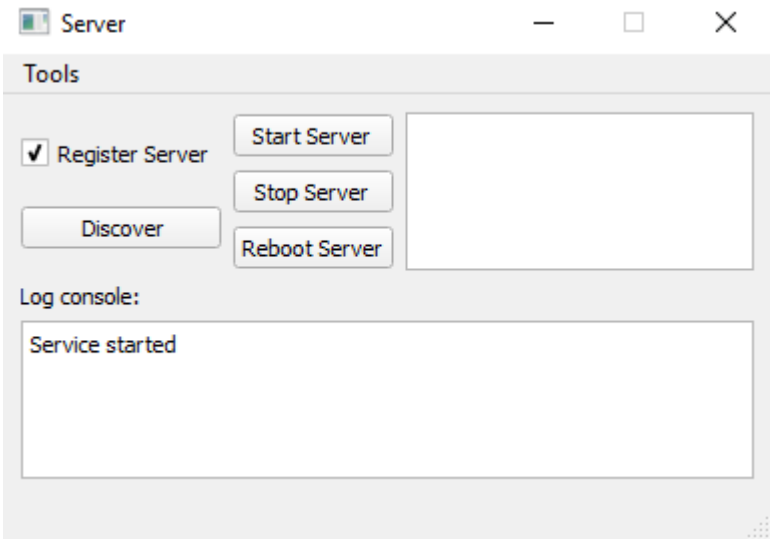

Figura 3.4 Interfaz del servidor

3.3 Diseño de la implementación

El diseño adoptado para la implementación se basa fundamentalmente en crear la estructura que permita el desarrollo de un sistema distribuido, pero la manteniendo la lógica base del modelo de (Mollineda, 2017), para ello puede ser dividido en tres partes fundamentales: el cliente, que contiene toda la lógica de la aplicación y es el encargado de regular el flujo incluso sino está local; el servidor, que se encarga del procesamiento fuerte y la comunicación entre los dos primeros.

Desarrollo del servidor

A la hora crear el servidor hay que tener en cuenta que este brindará servicio a varios clientes, por lo que se tiene que lograr que esté disponible, aunque este siendo empleado por otros usuarios. Para el desarrollo de la aplicación se decidió usar la biblioteca ***RPyC*** que contienen clases sumamente útiles para el desarrollo de sistemas distribuidos.

Para garantizar la extensibilidad y mantenibilidad del sistema, dentro de la implementación se maneja el concepto de una extensión como la suma de los elementos que definen un componente del flujo de análisis en el modelo conceptual, considerando su interfaz, alguna representación visual de este y su función, además de un conjunto de metadatos que describen el propósito del componente. La idea es que la implementación define la estructura que debe poseer una extensión, permitiendo agregar estas y utilizarlas inmediatamente sin necesidad de alterar el resto del código de la aplicación. La gestión de este proceso se realiza separadamente de la ejecución del modelo.

Debido a que los módulos de los servidores no tienen interacción directa con el usuario estos no poseen interfaces visuales, por lo que solo disponen del código dedicado al procesamiento de los datos. En caso de la actual implementación, la ejecución de dicho código se realiza en un hilo aparte utilizando la clase *QThread* de QT. Se tomó esta decisión por las limitaciones que presentan los hilos de Python en el caso de los procesos ligeros, y por parte de los procesos pesados, la naturaleza heterogénea de los datos, aunque no se descarta la segunda opción en caso de futuras implementación de otros tipos de nodos.

Estos nodos creados tienen que ser ubicados en los paquetes correspondientes a su clasificación (*analysis*, *input* o *viz*), de donde serán cargados dinámicamente, y mediante inyección de código publicados en la clase *ExecutionService* del módulo ***connection***, concatenándole el prefijo '*exposed_*' para que sea reconocible por el servicio. Dicha clase se encarga de manejar la conexión

con un cliente, y es aquí donde se colocan aquellos métodos que serán accesibles fuera del servidor. Una vez que *ExecutionService* sea instanciado por la conexión de un nuevo usuario esta creará un servicio donde pondrá a disposición del cliente todos los métodos que fueron inyectados. La **Figura 3.5** muestra el diagrama de clases asociado al módulo.

El servicio se mantiene disponible mediante la clase *ThreadedServer*, también perteneciente a la biblioteca ***RPyC***. Dicho servidor maneja cada conexión en un hilo aparte, que posibilita que siempre se esté escuchando por nuevas solicitudes. Una vez que se ha establecido la comunicación entre el cliente y el servidor, se invoca a *ExecutionService* para que realice su función y se desplaza el proceso hacia otro hilo.

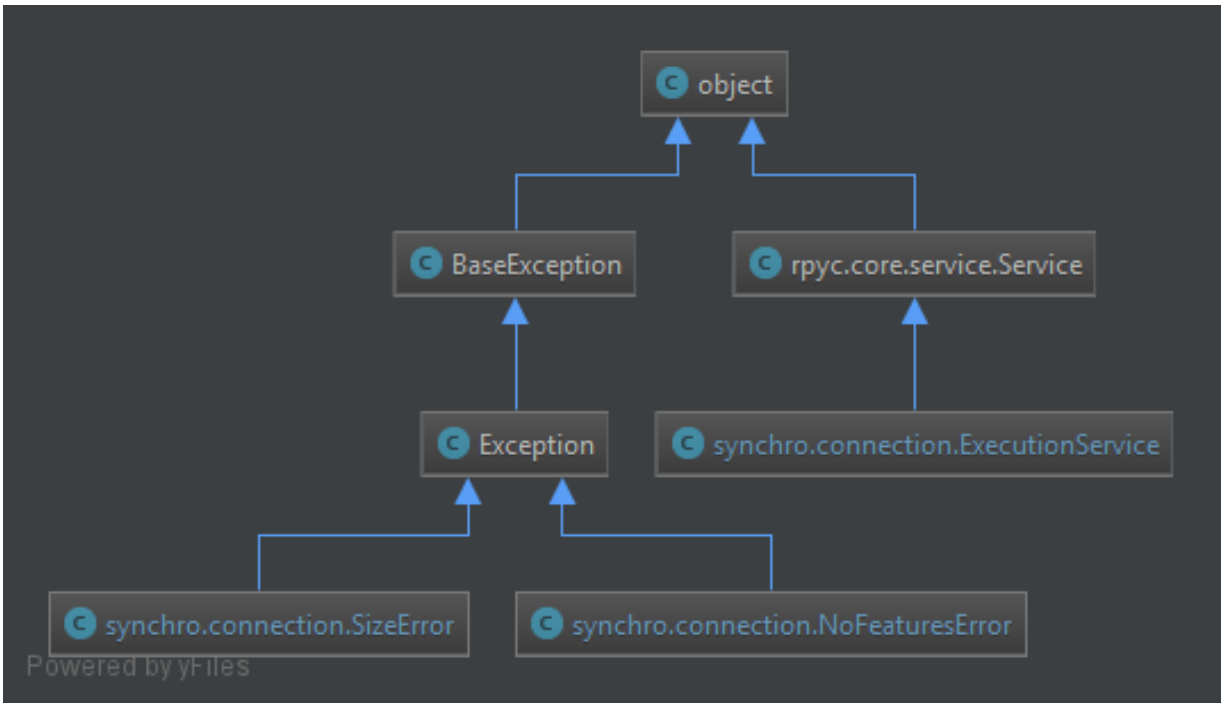

Figura 3.5 Diagrama de clases: connection

En el módulo **mainui** se encuentran los métodos asociados al control de la interfaz visual, algunas de sus funciones es iniciar, detener y reiniciar el servicio, como ya fue mencionado el usuario que instala el servidor determina si este albergará el servicio de publicación, decisión que se refleja en la configuración del servidor (*ThreadedServer*). La **Figura 3.6** muestra el diagrama de clases asociado a mainui. Por último, el módulo **serverui** alberga la interfaz visual del servidor. Para ver la relación que existe entre los paquete y clases en la **Figura 3.7** se muestra su estructura general.

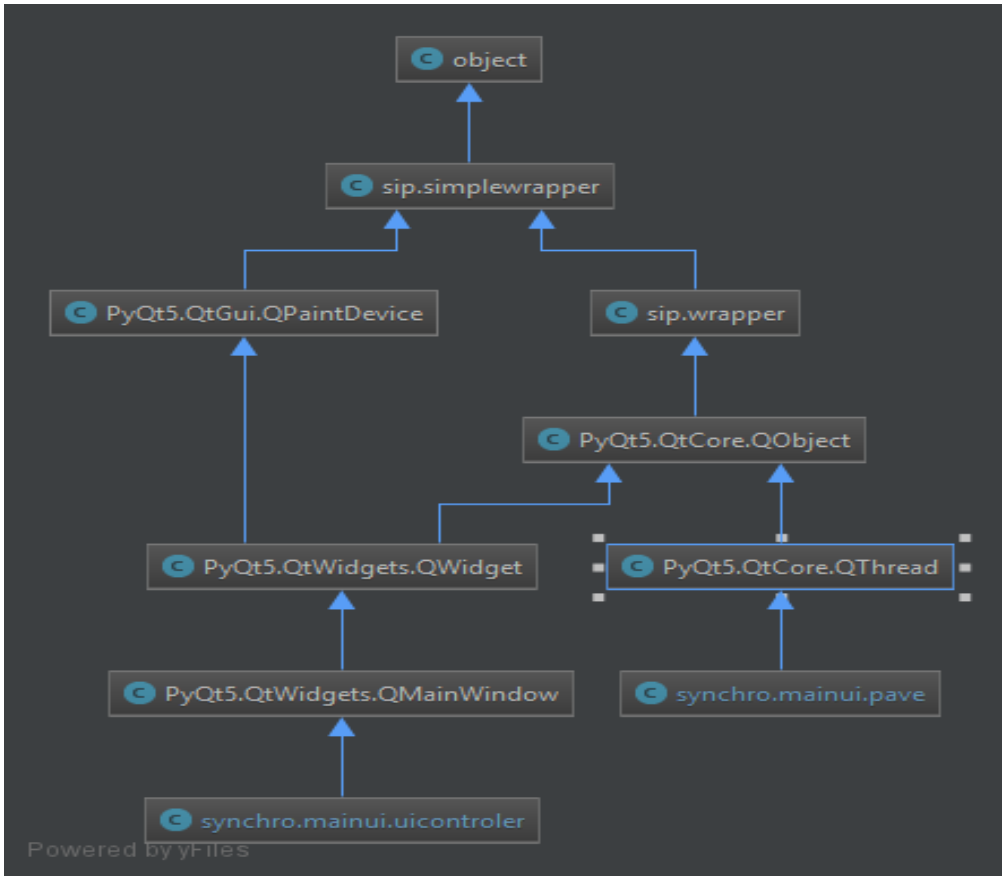

Figura 3.6 Diagrama de clases: mainui

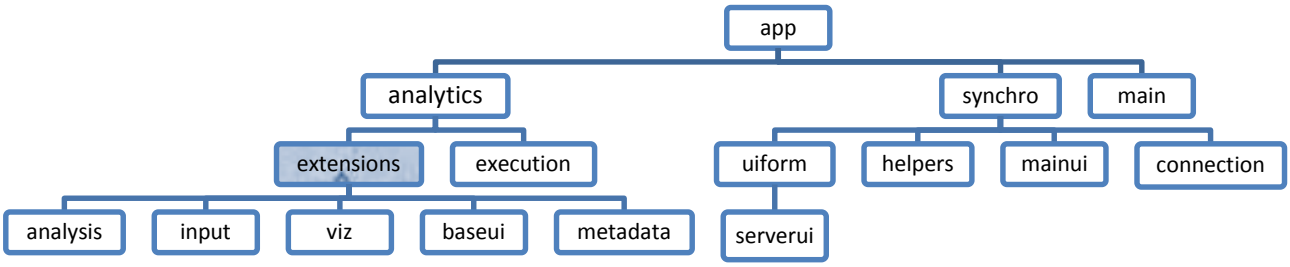

Figura 3.7 Estructura de paquetes y módulos de la aplicación

Desarrollo del cliente

Por su parte el desarrollo del cliente no conlleva a tantos cambios. El primero es la creación de una nueva interfaz visual dedicada a la registración de un nuevo servidor en el modelo, definida en el

módulo **addserverdialog**. Su función es proporcionar una herramienta visual para modificar los parámetros necesarios (IP y nombre).

En el caso de los nodos distribuidos se modificó el método *do_execute*, primero se identifica si el nodo será ejecutado local o remoto. En dependencia de la selección se ejecutará de un modo u otro, en presencia de la segunda variante se conecta con el servidor y le notifica el tipo de nodo que se requiere. Una vez identificado, si este es un punto de entrada al modelo (ejemplo: CSVinput) los datos necesarios serán enviados al servidor, de lo contrario solo se invoca al servidor y se espera la respuesta. También se tiene que modificar la forma en que se trata el flujo de datos, hay que añadirle capacidad de trabajar por lotes, que no es más que subdividir los datos en grupos más pequeños.

Proceso de comunicación

El mecanismo de comunicación que emplea ***RPyC*** es el mismo que ***RPC***, el cual se basa en el protocolo de red **HTTP**, esto posibilita la existencia de una comunicación asincrónica. Lo primero a realizar, es una comunicación con el servidor de publicación he identificar todos los servidores visibles, una variante sería la consulta siguiente: `rpyc.utils.factory.discover('EXECUTION')`, donde *EXECUTION* es el nombre del servicio; ya obtenida la información se comparte con el visual para que este se actualice. Ese es un proceso que se ejecuta frecuente para mantener la información actualizada.

Un proceso importante en el sistema es la comunicación del cliente con los servidores y los servidores entre ellos. Cuando comienza la ejecución, los componentes distribuidos se comunican con sus respectivos servidores y estos se preparan para la tarea que van a desarrollar. En esta comunicación se les informa a los componentes remotos quiénes son sus sucesores, de esta forma cuando un recurso remoto culmine su asignación, este envía los resultados al modelo para su publicación en el sistema, pero también establece una comunicación directa con los servidores que le siguen en el proceso de ejecución y le pasa sus resultados. Así se evita que el sistema local tenga que enviar los datos nuevamente a través de la red, y puesto que las probabilidades de que los servidores estén esperando los datos para comenzar su ejecución, es considerable se ahorra en tiempo de ejecución final. Es válido aclarar que cada componente local se encarga de la

configuración de su homólogo remoto y de obtener sus resultados. En la **Figura 3.8** se realiza una muestra muy sencilla de funciona la comunicación entre servidores, y con clientes.

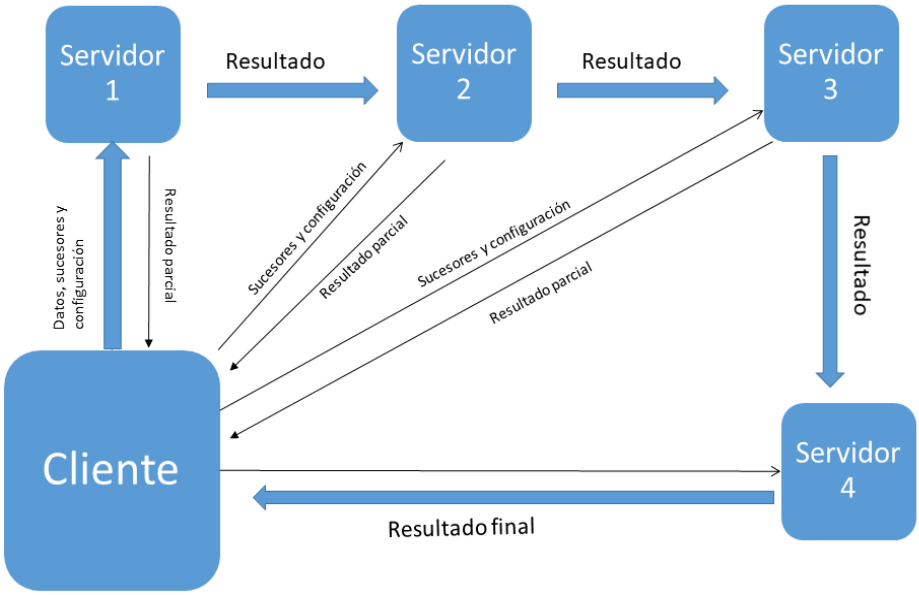

Figura 3.8 Interacción del flujo

3.4 Caso de estudio planteado por (Mollineda, 2017): propiedades que influyen sobre la permeabilidad

Para probar la capacidad del sistema distribuido en la solución de problemas, su aplicabilidad y la implementación de este, desarrollada sobre la aplicación ***BCSMap***, se sometió a un uso práctico mediante un caso de estudio diseñado en (Mollineda, 2017) por su autor y un grupo de especialistas de la rama biofarmacéutica. Este consiste en constatar, mediante el uso nodos distribuidos, la relación existente entre un conjunto de propiedades y la permeabilidad de los compuestos sometidos a estudio.

Como fuente para los datos empleados en la demostración se empleó un extracto de una base de datos biofarmacéutica, en formato CSV[18] y conteniendo 322 compuestos y 184 propiedades para cada uno de estos. Para adaptar la entrada al subconjunto de propiedades interesantes para este caso de análisis en particular, se creó un esquema de entrada conteniendo 17 propiedades que influencian o son influenciadas por los valores de permeabilidad de un fármaco.

El diseño del flujo de análisis contempla un componente de entrada del tipo CSVinput al que se le asigna el fichero: "test-input-no-smiles.csv". Además se configuró un componente de filtrado en base a los valores de propiedades, configurado para aplicar la regla de cinco de Lipinski (Lipinski, 2004), un criterio biofarmacéutico empleado para discriminar fármacos que pueden presentar problemas potenciales en su absorción (Mollineda, 2017), seguidos de dos componentes de análisis que aplican una implementación de k-medias, también ubicó un componente para el paso selectivo de vistas de datos en base a las demandas del analista (*dispatcher*).

Por último, para poder realizar la exploración visual, un componente de visualización (radviz) equipado con una técnica destinada a la identificación de estructura en base a un conjunto de propiedades y etiquetas (Mollineda, 2017).

Posteriormente es necesario definir las conexiones entre los componentes los cuales finalmente termina conectados de la manera siguiente:

Desde el *CSVinput* salen dos conexiones, una hacia el componente de visualización (radviz) y otra hacia el componente de filtrado (regla de 5), de este sale dos conexiones nuevamente, la primera hacia un componente de *Kmean*, el que se encuentra junto a la otra conexión con el *dispatcher*. De aquí sale una conexión deshabilitada hacia un *Kmean* y otra para el componente de visualización. En la **Figura 3.9** se muestra el grafo resultante antes de ser ejecutado.

[18] Elementos separados por coma

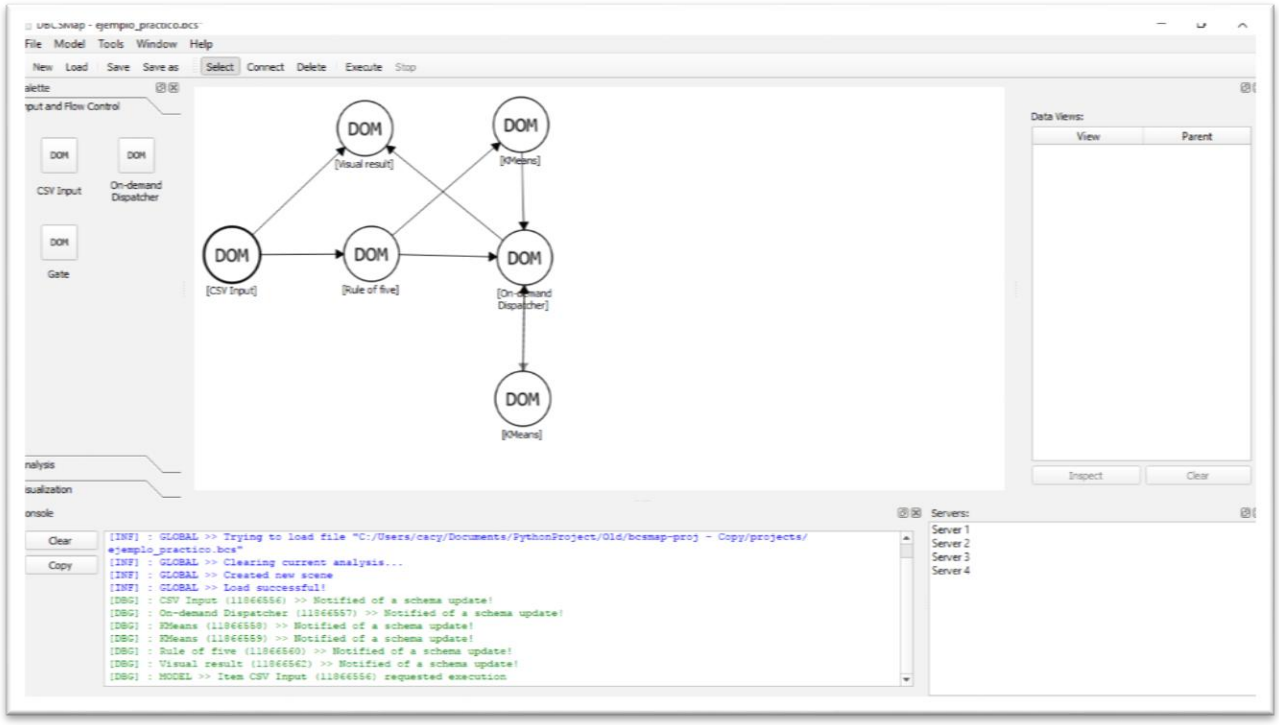

Figura 3.9 Ejemplo práctico

Luego de haber iniciado la ejecución y después de la aplicación del filtro configurado con la regla de cinco, la ejecución global del flujo produce en primera instancia un conjunto de datos filtrados donde se reduce la cantidad de compuestos a considerar a 252, concentrados principalmente en fármacos de la clase I, II y III, con elevada permeabilidad y solubilidad respectivamente. La mayor parte de los fármacos pertenecientes a la clase IV fueron eliminados en este paso, dado que en su mayor parte violan algún punto de la regla de Lipinski (Mollineda, 2017).

Como salida del procesamiento del primer componente de k-medias, se obtienen cinco vistas de datos, de las cuales una se corresponde a la vista de entrada filtrada, pero con la información de agrupamiento agregada a ella. Las restantes cuatro se corresponden con los grupos identificados. Para el caso de estudio aplicado resulta de particular interés la vista correspondiente a la clasificación III/I, en la que se experimenta la mayor variación en la permeabilidad (Mollineda, 2017).

Al pasar esa vista hacia el componente de visualización "*visual result*" mediante la activación del componente de control de flujo, es posible estudiar el comportamiento de la estructura visible en

las clases BCS con respecto a las propiedades seleccionadas lo que permite una generación de hipótesis por parte del especialista que se encuentra conduciendo el análisis.

Al aplicar esta misma selección de propiedades sobre los datos originales sin filtrar, es posible observar la misma regularidad, lo que aumenta el número de elementos que apoyan que este conjunto de propiedades en particular es una buena elección para discriminar entre compuestos con elevada y baja permeabilidad (Mollineda, 2017)

Luego se aplica un nuevo paso de agrupamiento en el componente *Kmean* que previamente no estaba accesible, y mediante el empleo del componente de visualización, se puede visualizar que los datos filtrados han sido divididos en dos grupos correspondientes a elementos de baja y alta permeabilidad respectivamente. La visualización del nuevo agrupamiento y su comparación con la distribución de las clases propiamente dichas apoya fuertemente la conclusión obtenida (**Figura 3.10**).

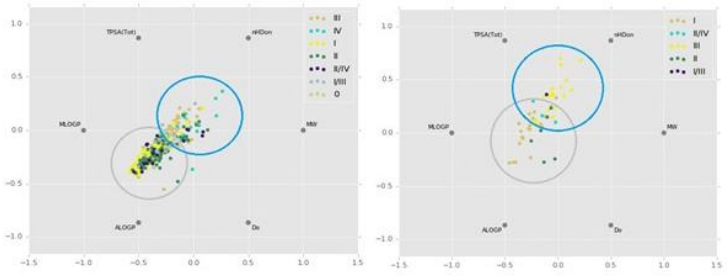

Figura 3.10 Comparación de la distribución de clases

3.5 Comparación de resultados

Usando el modelo de (Mollineda, 2017) se definió un flujo de trabajo de prueba donde las imágenes resultantes se almacenaron en formato ***jpeg***[19], para realizar una comparación entre estas y las resultantes usando el modelo distribuido. El modelo mencionado consta de siete componentes, y es el que está representado en la **Figura 2.3**. Luego de realizar la ejecución del modelo se obtienen nueve vistas de datos, de las cuales las últimas cuatro son las que llegan a los componentes de visualización. Luego se realiza el mismo flujo de ejecución, pero esta vez usando el modelo distribuido e igualmente se obtienen los mismos resultados. Al comparar estas vistas

[19] Método de compresión de imágenes

(solo tres de ellas) se evidencia que efectivamente se obtiene el mismo resultado como se muestra en las figuras: **Figura 3.11**, **Figura 3.12** y **Figura 3.13**.

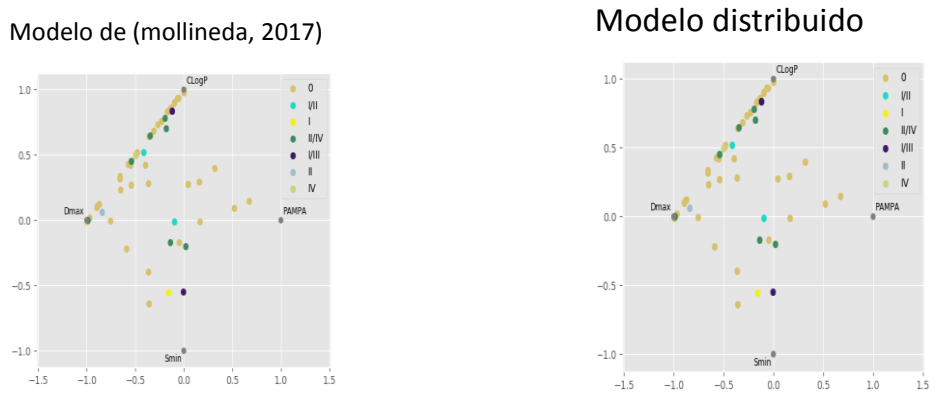

Figura 3.11 Primera vista

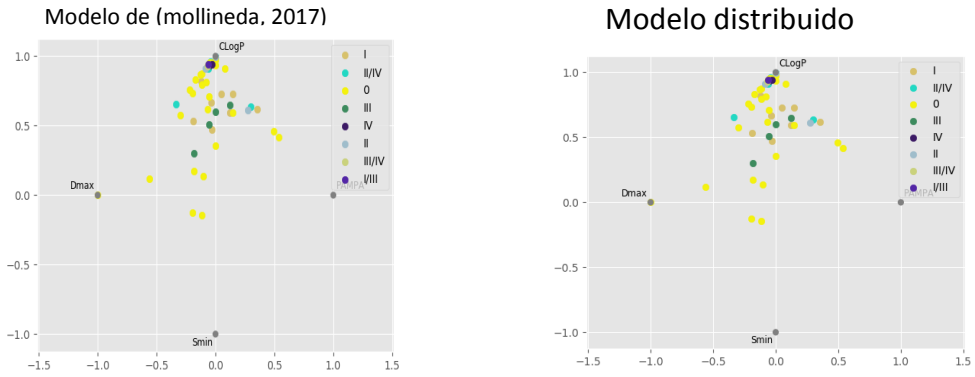

Figura 3.12 Segunda vista

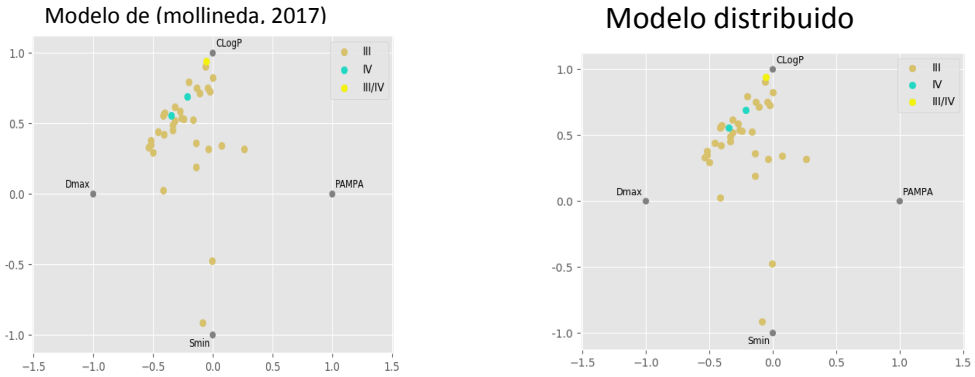

Figura 3.13 Tercera vista

3.6 Conclusiones parciales

Siguiendo el proceso de análisis planteado por (Mollineda, 2017), la implementación del modelo propuesto arrojó resultados satisfactorios en un ejemplo real. Se obtuvo un nuevo conocimiento en forma de un conjunto de propiedades potencialmente útil para discriminar nuevos fármacos (Mollineda, 2017). Además, se demostró que la implementación no introduce errores ni variaciones en los resultados.

CONCLUSIONES

Como resultado de este trabajo se identificó una arquitectura de red para las conexiones entre el cliente y los servidores, donde los servidores utilizan la arquitectura ***P2P***, siendo de esta manera apropiada para las características deseadas, así como un lenguaje de programación y algunas bibliotecas útiles. Haciendo uso de lo anterior se planteó un modelo distribuido y se desarrolló una herramienta basada en este, cumpliéndose de esta forma el objetivo general del proyecto. Además, se demostró que el empleo del modelo como herramienta para enfrentar el problema de la búsqueda de conocimiento mediante el análisis visual de datos es aplicable. Fue demostrado también la aplicación de esta herramienta a un caso de estudio real.

RECOMEDACIONES

A lo largo del trabajo se fueron mencionando algunas sugerencias, las cuales son:

1. Diseñar una función heurística que maneje el balance de cargas a la hora de la distribución de los nodos por la red, para que se logre de esta manera que este proceso sea transparente para los usuarios.
2. Extender los componentes visuales al modelo en futuras implementaciones
3. Implementar nodos que hagan uso de concurrencia mediante procesos pesados en vez de ligeros.
4. Probar la efectividad del modelo distribuido para flujos de datos grandes donde se muestre la disminusion en tiempo que se obtiene al usar el modelo distribuido.
5. Permitir la publicación de los resultados de análisis en un ambiente web, para que estén disponibles a otros especialistas.

De forma general se recomienda continuar con la ampliación del modelo propuesto, y realizar futuras implementación donde se busque mejorar su rendimiento.

BIBLIOGRAFÍA

Bass, L., Clements, P. and Kazman, R. (2003) *Software Architecture in Practice.* 2a edn. Addison-Wesley.

Bochenski, B. (1994) *Implementing Production-Quality Client/Server Systems*. John Wiley.

'Capítulo 4. Introducción al paralelismo y al rendimiento. 4.1.' (2012) in, pp. 1–23.

Dalcin, L. (2017) 'MPI for Python'.

Edmundo, R., Valencia, C. and Feliciano, A. (2010) *Autores : José Luis Hernández Hernández*.

Eugster, P. *et al.* (2003) *The Many Faces of Publish/Subscribe.* ACM Comput.

Fernández, N. and Arias, J. (2012) 'Peer to peer (I) Introducción Introducción', (I), pp. 1–13.

Filiba, T. (2018) 'RPyC Documentation'.

Foster, I. (1994) 'Designing and Building parallel programs: concepts and tools for parallel software engineering'.

González, A. S. (2008) 'Sistemas Operativos Distribuidos'.

Guillermina, A., Galache, G. and Herrera, P. A. (2014) 'Sistema de Intercambio de Archivos : Análisis de Requerimientos de Sistemas Distribuidos Sistema de Intercambio de Archivos : Análisis de Requerimientos de Sistemas Distribuidos', (May).

Holovaty, A. and Kaplan-Moss, J. (2008) *El libro de Django*.

Johansson, R. (2016) *Introduction to Scientific Computing in Python*.

Keim Daniel, K. J. and Mansmann, G. rey E. and F. (2010) *Mastering the Information Age Solving Problems with Visual Analytics, Mastering the Information Age Solving Problems with Visual Analytics*. doi: 10.1016/j.procs.2011.12.035.

Kerren, A. *et al.* (2008) *Information visualization: human-centered issues and perspectives*. Springer.

Kuhlman, D. (2012) *A Python Book : Beginning Python , Advanced Python , and Python Exercises*.

Labs, T. M. (2018) 'Twisted Documentation', p. 487.

Lafuente, A. (2014) '1 Introducción a los sistemas distribuidos', in, pp. 1–35.

Lipinski, C. A. (2004) 'Lead-and drug-like compounds: the rule-of-five revolution', *Drug Discovery Today: Technologies*. Elsevier, 1(4), pp. 337–341.

Mehta, N., Medvidovic, N. and Phadke, S. (2000) *Towards A Taxonomy Of Software Connectors.* Proc. 22nd Int'l Conf. on Software Engineering, (Limerick, Irlanda): NY:ACM Press.

Mollineda, D. (2017) *Tesis Final*. Universidad Matha Abreu de las Villas.

Neuman, B. C. (1994) *Scale in Distributed Systems, Readings in Distributed Computing Systems*.

Object Management Group, F. (2004) *The Common Object Request Broker: Core Specification, revision 3.0.3*.

Orfali, R. (1994) *Essential Client/Server Survival Guide*. Estados Unidos.

Pérez Mora, O. *et al.* (2005) *Bases de datos*. Fundació per a la Universitat Oberta de Catalunya. Available at: http://www.uoc.edu/masters/oficiales/img/913.pdf.

Rhodes, B. and Goerzen, J. (2016) *Foundations of Python Network Programming*.

Ruiz, S. and Vargas, J. (2012) 'BASES DE DATOS'.

System, A. (2013) 'BCS Biowaivers'.

Tanenbaum, A. S. and Steen, M. Van (2008) *Principios y Paradigmas*. Segunda Ed. Pearson Education.

Wagner, M. *et al.* (2017) 'Performance Analysis of Parallel Python Applications'.