Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

Me gustaría dar agradecimientos a mi familia y facultad, ya que sin ellos esto no habría sido posible nada de estos.

Es importante agradecer también a CIC Consulting Informático por permitirme la oportunidad de realizar el desarrollo del proyecto en su empresa, sin olvidarme de mis compañeros de LUCA, que han sido un gran apoyo durante el mismo..

Para finalizar, me gustaría gradecer a mi mentor Pablo, por guiarme durante el desarrollo del proyecto con eficacia y ayudarme a afrontar este trabajo de fin de grado.

Resumen

Las empresas actuales no utilizan un único sistema de información que de soporte a sus procesos de trabajo, sino un ecosistema de sistemas información que dan soporte a diferentes procesos de negocio ejecutados dentro de dicha organización. Como consecuencia de esta nueva situación, cuando un usuario quiere obtener una información concreta cuyos datos residen en varios de estos sistemas, necesita acceder a cada uno de estos sistemas, extraer de cada sistema la información que precisa, filtrarla y unificarla para finalmente obtener los datos requeridos.

Por ejemplo, una tienda de electrodomésticos podría tener sistemas informáticos diferentes para el departamento de atención al cliente, para el departamento técnico de postventa y para el departamento de compras y adquisiciones.Por tanto, para conocer el estado actual de una reparación, podríamos necesitar:

- Acceder al primer sistema para obtener el identificador de la incidencia y en qué fase de su gestión se encuentra.
- Comprobado que la incidencia está actualmente en reparación, recuperaríamos otro sistema el estado detallado de la reparación, comprobando que está a la espera de una pieza.
- Finalmente accederíamos al sistema de compra y adquisiciones para comprobar cuando está prevista la entrega de dicha pieza. Los sistemas de almacenamiento de la información pueden ser diversos, incluyendo desde un servicio web, una base de datos relacional, un repositorio de ficheros accesible vía FTP o una base de datos NoSQL.

El objetivo de este proyecto es facilitar dicho proceso de composición al usuario mediante el desarrollo de un mecanismo gráfico para la especificación de estos procesos de composición de consultas.

Palabras clave:

Preface

Keywords:

Capítulo 1

Introducción

Este primer capítulo describe de forma general la información principal y relevante para poder comprender el funcionamiento del producto actual LUCA, así como los objetivos del proyecto a desarrollar junto con la explicación y motivación del mismo.

1.1. Introducción

El Trabajo Fin de Grado descrito en este documento ha consistido en ...

Dado que el presente trabajo se enmarca dentro del proyecto LUCA, para poder comprender los objetivos de este Trabajo Fin de Grado, se hace necesario describir primero dicho proyecto, lo cual se realiza en la siguiente sección.

1.2. LUCA

1.2.1. Motivación

En los últimos años, el volumen de datos recogidos y manipulados por las empresas ha aumentado de forma vertiginosa. Estos datos se han ido almacenando en diferentes tipos de fuentes conforme las empresas crecían y sus sistemas evolucionaban y se fusionaban. Como resultado de este proceso no es extraño actualmente encontrar empresas que tengan sus datos almacenados en sistemas tan dispares como bases de datos relacionales, hojas XML o repositorios FTP.

Como consecuencia de esta nueva situación, cuando un usuario quiere obtener una información concreta cuyos datos residen en varios de estos sistemas, éste necesita acceder a cada uno de estos sistemas, extraer de cada

sistema la información que precisa, y finalmente filtrarla y unificarla para finalmente obtener los datos requeridos.

Por ejemplo, una cadena de venta de electrodomésticos podría tener sistemas informáticos diferentes para el departamento de atención al cliente, para el departamento técnico de postventa y para el departamento de compras y adquisiciones. Por tanto, para conocer con precisión el estado actual de una reparación, podríamos necesitar:

1. Acceder al sistema de atención al cliente para obtener el identificador de la incidencia y en qué fase de su gestión se encuentra.

2. Una vez corroborado que la incidencia está actualmente siendo atendida, recuperaríamos del sistema de gestión de reparaciones el estado detallado de la reparación. Como resultado de esta operación, supongamos que averiguamos que la reparación está a la espera de recibir una pieza que se ha de sustituir.

3. Finalmente, para poder hacer una estimación de cuando podría estar lista la reparación, accederíamos al sistema de compra y adquisiciones para averiguar cuando está prevista la entrega de la pieza solicitada.

Como hemos comentado anteriormente, a cada uno de estos sistemas podría accederse de manera diferente. Por ejemplo, el primero podría consultarse utilizando un servicio web. La información del segundo podría recuperarse accediendo directamente a una base de datos relacional, mientras que la información del tercero se obtendría analizando órdenes de compra en formato *pdf* almacenadas en un repositorio de ficheros compartido. Por tanto, el usuario, para poder realizar este proceso, necesita conocer las particularidades de cada sistema y de su forma de acceso.

Para aliviar esta situación, dentro de la empresa CIC, se está desarrollando una aplicación denominada LUCA, a la cual contribuye este Trabajo Fin de Grado. Para facilitar este proceso de recuperación de información, LUCA proporciona un lenguaje común para todas las fuentes de datos a unificar, permitiendo al usuario abstraerse de los detalles de cada fuente.

1.2.2. Funcionamiento de LUCA

A continuación, se detalla brevemente el funcionamiento de LUCA. Para ello, utilizaremos como ejemplo el proceso de recuperación de información de un servicio REST. Concretamente, se utilizará el servicio REST que proporciona información de la flota de autobuses de Santander (TUS). Dicho

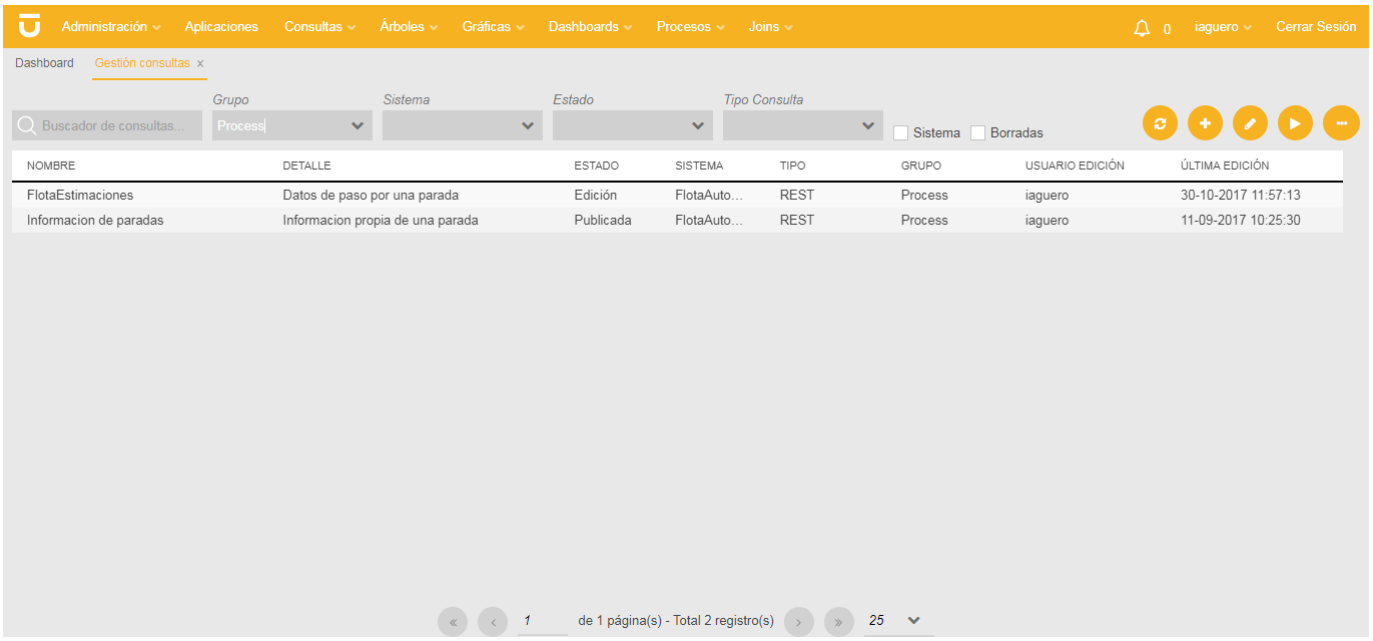

Figura 1.1: Vista de Gestión de Consultas

servicio permite obtener, a partir del identificador de una parada de autobús, información relativa a dicha parada, como, por ejemplo, su localización.

Al abrir LUCA, nos encontramos con una primera vista (Figura 1.1)que proporciona una serie de consultas predefinidas que el usuario tiene disponibles para ejecutar. Estas consultas predefinidas representan procesos de recuperación de información que el usuario realizará habitualmente contra diversos sistemas. En este ejemplo introductorio, se muestran sólo dos consultas: *Flota Estimaciones* e *Información Parada*.

Al seleccionar una consulta, se iniciaría el proceso de ejecución de la misma, para lo cual se abre una nueva interfaz. En primer lugar, debemos proporcionar a la consulta requeriría los datos de entrada necesarios para su ejecución. Por ejemplo, en el caso de la consulta *Información Parada* (Figura 1.1, etiqueta 1) sería necesario proporcionar el identificador de la parada de la que queremos obtener información. Una vez introducido dicho identificador, se seleccionaría el botón de ejecución de la consulta (Figura 1.1, etiqueta 2). La consulta tiene definido internamente cómo acceder al servicio REST, por lo que dicho proceso se realiza de manera transparente al usuario. Finalmente, se muestra el resultado de la consulta, el cuál puede ser visualizado de diferentes formas en función del recurso al que se llama. En nuestro caso, hemos optado por visualizar el resultado de la consulta en formato JSON (Figura 1.1, etiqueta 3).

La principal ventaja que aporta LUCA es que el proceso de ejecución de consultas es opaco para el usuario que la ejecuta. El usuario sólo tiene que proporcionar los parámetros necesarios de entrada y seleccionar un formato de salida. Por tanto, el proceso de ejecución de consultas es exactamente el mismo con independencia de la fuente a la cual se accede.

No obstante, para que dichas consultas pueden ser ejecutadas, cómo llevar a cabo las mismas debe haber sido especificado previamente. Para ello, un usuario con conocimientos suficientes para ello, al que denominaremos en adelante el *creador de consultas*, debe haber especificado cómo ejecutar cada consulta a bajo nivel.

Para poder realizar esta tarea, el creador de consultas accedería a interfaz dedicada a esta tarea (ver Figura **??**). En esta interfaz, definiría primero las variables de entrada y salida de la consulta (Figura **??**, etiqueta 1). A continuación, especificaría cómo llevar a cabo dicha consulta a bajo nivel. Para ello puede utilizar una serie de facilidades y primitivas proporcionadas por LUCA. Para ejemplo, en el caso de la consulta *Información Parada*, utilizando estas facilidades se especificaría el el recurso a obtener y el tipo de llamada HTTP a realizar, un *GET* en nuestro caso.

Lo importante de este proceso es que, una vez definida la consulta, ésta se puede ejecutar fácilmente sin conocer los detalles internos de la misma, incluso hasta el tipo de sistema al que se accede.

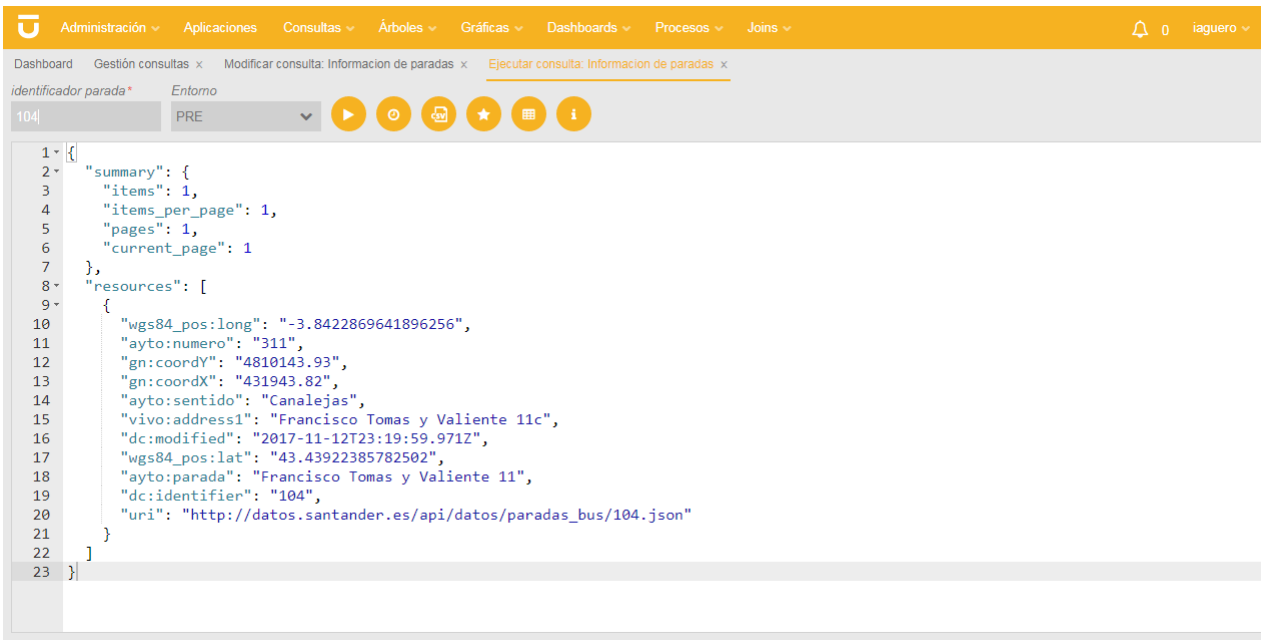

Figura 1.2: Vista de Ejecución de Consultas

1.2.3. Limitaciones actuales de LUCA

Actualmente, LUCA proporciona mecanismos para permitir al usuario recuperar de manera uniforme información de diferentes fuentes de datos. No obstante, LUCA por el momento sólo es capaz recuperar información de una única fuente de datos a la vez. Por tanto, cuando es necesario combinar información procedente de distintas fuentes, el propio usuario es el que debe realizar dicho proceso de composición a mano, ejecutando él cada consulta, y utilizando las salidas de cada una de ellas como entradas para las siguientes.

Un ejemplo de dicho proceso de composición sería la necesidad de un dependiente de una tienda de electrodomésticos de obtener la edad de los usuarios que compraron lavadoras durante el mes pasado. Actualmente, la secuencia de consultas que debería de realizar serían las siguientes:

- Primero necesitaría obtener el registro de compras del mes pasado del sistema.
- Después, tras guardar dicho registro, tendría que, uno por uno, seleccionar los que se corresponden con lavadoras.
- Una vez que el usuario tiene las lavadoras compradas el mes pasado, éste tendría que extraer que usuarios han comprado las lavadoras.
- Por último, debería buscar en el sistema cada usuario que ha realizado la compra, a partir del nombre obtenido en el punto anterior, y anotar su edad.

En adelante, estas cadenas de consultas para obtener un resultado concreto las denominaremos *procesos*. El problema actual de LUCA, tal como ilustra el ejemplo anterior, es que no soporta el concepto de *proceso*. Por tanto, para ejecutar un proceso, el usuario tiene que realizar una larga y compleja secuencia de acciones.

1.3. Objetivos del Trabajo de Fin de Grado

El objetivo general de este Trabajo Fin de Grado es integrar en LUCA el concepto de *proceso*. Para ello, hay que dar soporte a dos cuestiones diferentes: (1) la ejecución de los procesos; y (2) la especificación de procesos. Por tanto, el objetivo general de este trabajo se descompone en estos dos subobjetivos principales.

El primer objetivo implica poder tratar procesos en LUCA de la misma forma que se trata las consultas. Es decir, los procesos deberán aparecer en la

interfaz de consultas (ver Figura**??**), y ejecutarse mediante el mismo procedimiento utilizado para las consultas simples (ver Apartado**??**). Obviamente, la complejidad de ejecutar una proceso es mayor que la de ejecutar una consulta, ya que necesitamos ejecutar varias consultas, guardar resultados intermedios y utilizar estos resultados como entradas para otras consultas.

El segundo objetivo, que es el que implica una mayor complejidad, consiste en facilitar la especificación de procesos en LUCA. Para que un proceso pueda ser ejecutado, primero debe ser especificado, indicando qué consultas lo componen y cómo se relacionan. De acuerdo con los deseos expresados por los responsables del proyecto LUCA y la empresa CIC, dicho mecanismo de especificación debía ser gráfico, permitiendo así componer consultas de manera visual mediante la interconexión de las salidas de unas con las entradas de otras.

Para refinar estos dos grandes objetivos en una serie de requisitos más concretos, se llevó a cabo en primer lugar una reunión con el Jefe y el Gerente del proyecto. El objetivo de dicha reunión era conocer LUCA en profundidad. A continuación, dado que la fase de Ingeniería de Requisitos para este proyecto ya había sido realizada por la propia empresa, se nos proporcionaron unos documentos técnicos con los requisitos técnicos tanto para la ejecución de procesos como para el desarrollo del componente gráfico de especificación de procesos. Estos documentos pueden encontrarse en el Anexo adjunto a la memoria.

Como ya se ha mencionado, en estos documentos se pueden encontrar los requisitos técnicos atribuidos al proyecto, pero, de forma resumida, se centran en tres pilares o requisitos principales:

- Concatenación de las consultas entre si pertenecientes a un mismo proceso.
- Visualización del progreso de ejecución del proceso.
- Aplicar criterios de navegación a partir de los resultados de salidas.

1.4. Arquitectura LUCA

Esta sección explica la actual arquitectura de LUCA, mostrada en la Figura**??** con objeto de que se pueda entender su funcionamiento y cómo el trabajo descrito en este documento se integra en dicha arquitectura. LUCA sigue una arquitectura de tres capas donde la vista de presentación se estructura de acuerdo al patrón *Model-View-Presenter* (MVP).

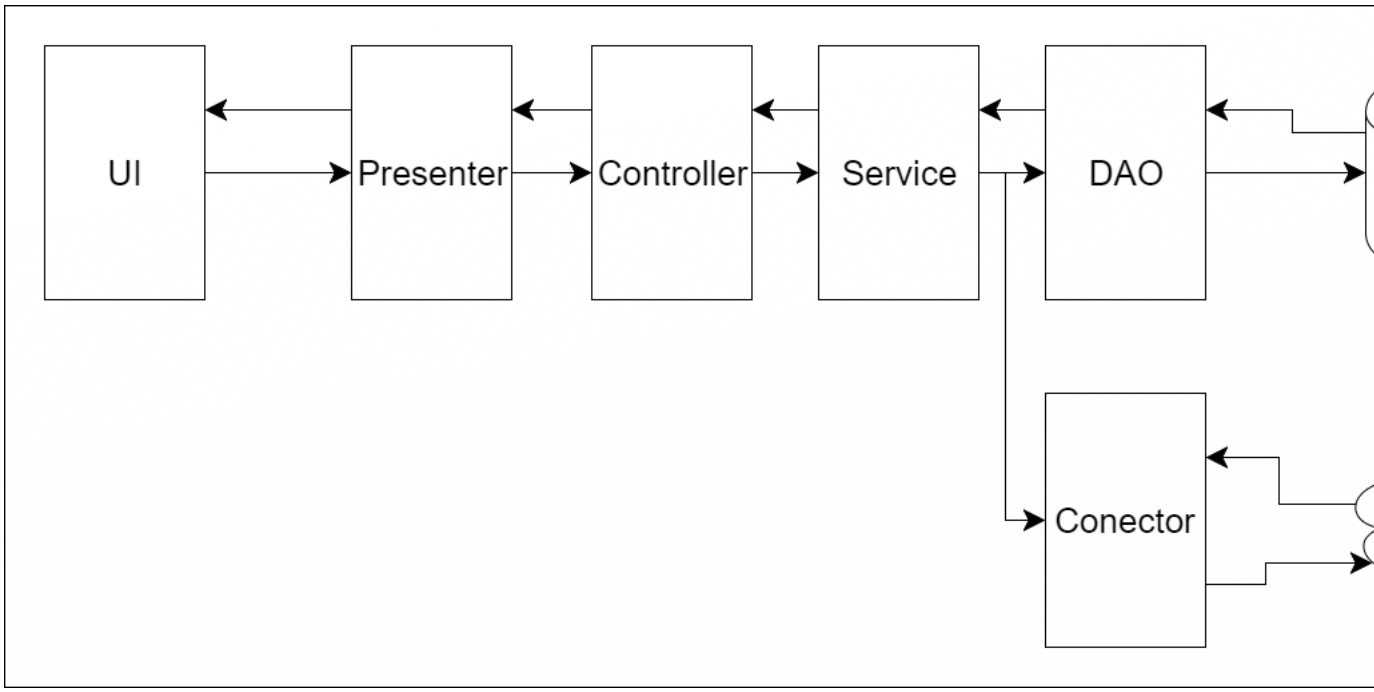

Figura 1.3: Arquitectura de LUCA

1.4.1. Arquitectura del Conector

El conector de LUCA es un componente que se encarga de recibir o recoger los datos de los diferentes recursos albergados en las diferentes fuentes de datos externas. Puede ser de diferentes tipos, como es un conector REST, SOAP o de acceso a bases de datos

En el ejemplo posterior podemos ver un diagrama que describe dicha interacción. El conector en función del tipo va a comunicarse con un cliente diferente, ya sea el HTTPClient [1] de Apache o JDBC [6] de Oracle, para realizar la comunicación y recepción de datos con los diferentes recursos.

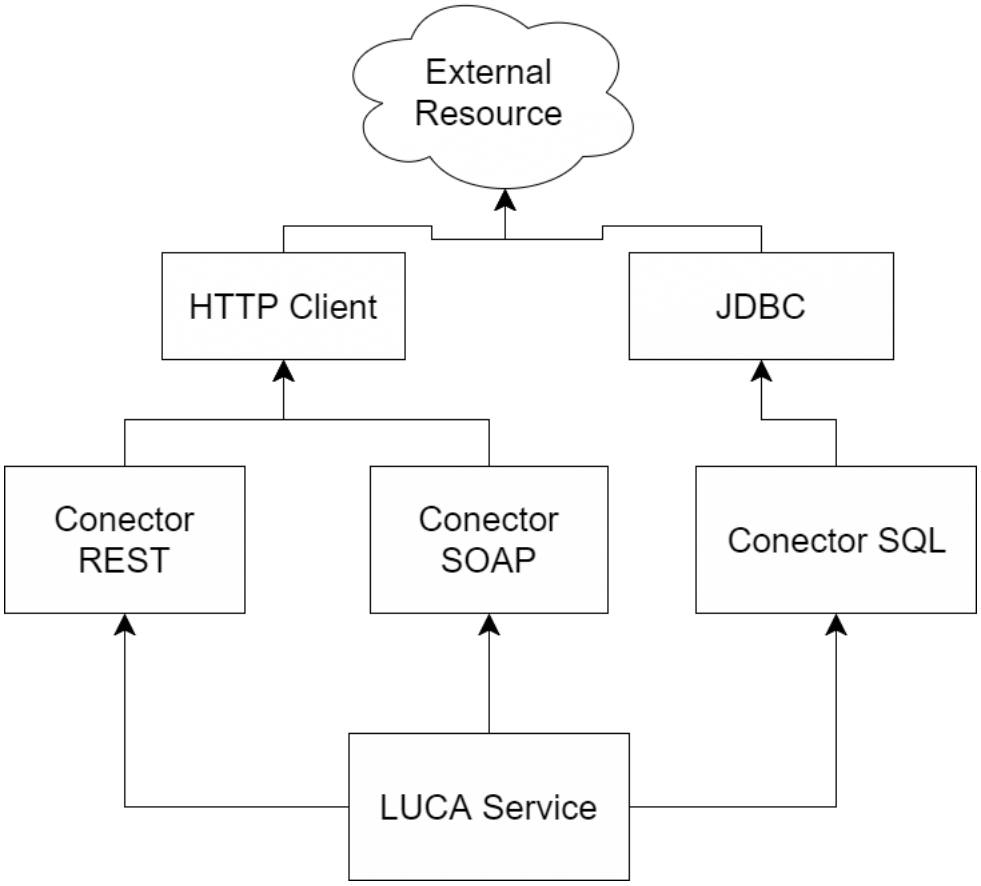

Figura 1.4: Arquitectura del Conector

1.4.2. Arquitectura del Process-Component

El Process-Component es un proyecto abstracto (debe de ser importado e implementado por un proyecto o componente padre) encargado de recibir y comunicar los eventos realizados sobre una interfaz construida a partir de la librería GO.JS.

La arquitectura del Process-Component se ostenta en dos pilares. El primero es la implementación de la lógica del propio componente y el segundo es la implementación de un conector que se comunica con una librería de GO.JS que también debe de ser definida.

La lógica del componente se basa en un estado (el cuál alberga todos los elementos para poder formar la vista), y en una serie de acciones o comandos que se pueden realizar sobre él, y donde tras cada acción, se realiza una comunicación con el conector para que este se encargue de modificar el estado

de los elementos de GO.JS.

El conector internamente se compone de la librería encargada de definir el ámbito gráfico con el que se va a trabajar, de una serie de eventos que serán trasladados a la lógica del Process-Component, y de modificar los elementos en activo en la vista bajo la orden especificada.

A continuación, se muestra una figura explicativa de dicha interacción interna entre los componentes:

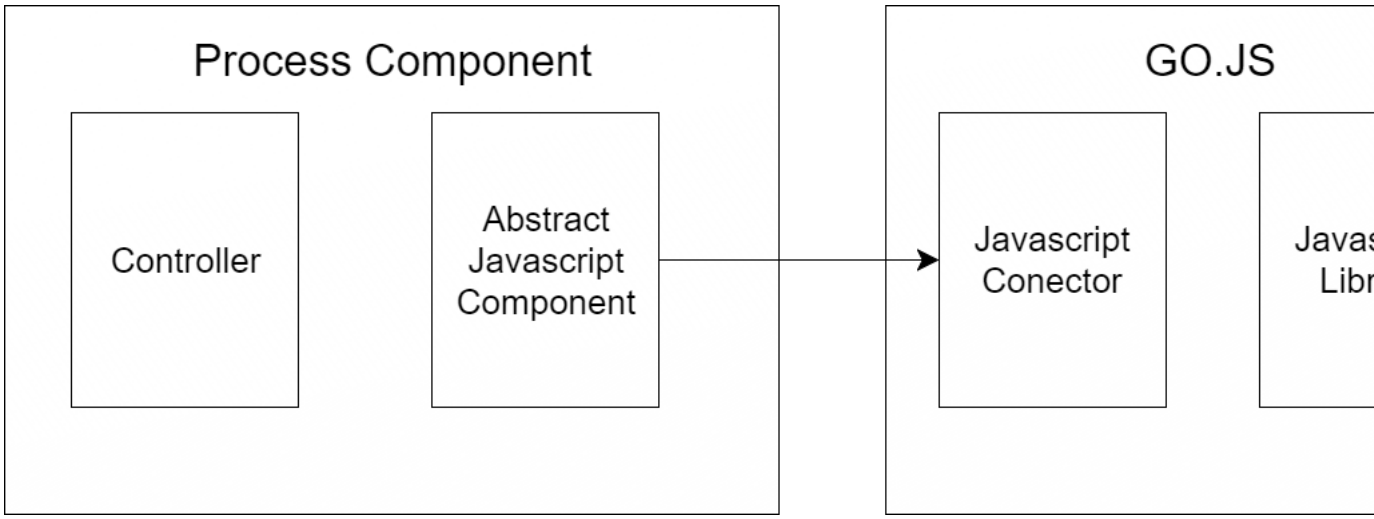

Figura 1.5: Arquitectura del Process-Component

Como resumen del funcionamiento del Process-Component, éste es el encargado de recibir eventos realizados sobre la interfaz gráfica, a través del conector hasta llegar a la lógica del componente para publicarlos o abstraerlos a un conjunto de elementos escuchadores que deberán de ser atendidos por el componente implementador de este Process-Component.

1.5. Planificación

La sección presente describe el proceso llevado a cabo para realizar el desarrollo tanto del componente gráfico (Process-Component) como del componente de Luca (Luca-Process).

El proceso que se llevó a cabo para realizar el desarrollo del proyecto se basó en dos fases:

- Una primera en la que se estudió el funcionamiento de la herramienta gráfica de GO.JS y después se realizó la implementación del Process-Componet.

- Una segunda en la que utilizando el componente descrito anteriormente, se implementa un incremento utilizando la lógica previa de LUCA.

1.5.1. Implementación

A continuación se explica brevemente el proceso de implementación llevado a cabo para ambas fases:

- Process-Component

  El primer paso para empezar la implementación del Process-Component, fue aprender y practicar con ejemplos y con pruebas de concepto para ver como funcionaba y aprender la sintaxis de la librería de GO.JS.

  Tras este primer paso, se realizó la implementación del modelo (se puede encontrar en los documentos técnicos), para posteriormente generar una primera prueba de concepto conjunta realizando una comunicación con la librería GO.JS.

  Por último se implementaron los eventos y escuchadores para posteriormente centrarse en completar dicho componente.

- Luca-Process

  En este componente se sigue un metodología similar al anterior. Primero se crea una prueba de concepto sencilla para ver la comunicación entre los componentes para posteriormente empezar a montar todo el sistema de recepción de eventos y llamadas al componente gráfico.

  El siguiente paso fue seguir el estilo arquitectónico ya existente en LUCA, utilizando el patrón MVP [3] para crear toda la jerarquíca de capas necesarias para la comunicación con los servicios ya existentes de LUCA y la propia base de datos de LUCA.

1.5.2. Pruebas

Este apartado describe el proceso de pruebas llevado a cabo para la comprobación y verificación de las diferentes funcionalidades del proyecto.

Las pruebas se centraron principalmente en el Luca-Process, ya que es el componente que alberga la mayor lógica del proyecto debido al conjunto de servicios que lo compone y a la lógica sobre los presenters desplegada.

Profundizando en las pruebas implementadas, solo se han realizado pruebas de integración por varios motivos. El primero es que las pruebas unitarias no son necesarias hacerlas ya que se centran sobre la capa de repositorio y esta capa ha sido implementada con Spring Data Jpa [7] y ofrece ya una fiabilidad. El motivo por el que no se realizaron pruebas de sistema, aunque en una primera planificación estaban previstas hacerlas integrandolas con Selenium [9] fue la complejidad que lleva la integración junto con Vaadin [2], ya que la ventaja de programar mediante una captura de eventos sobre la vista toda la secuencia de movimiento sobre dicha vista pasa a ser programática, y debido a que había que ceñirse a unas fechas de entrega y al tiempo que llevaría dicha implementación se decidió omitirlas.

Las pruebas de integración que se llevaron a cabo se implementaron con Spring Test [8] y JUnit [4]. Desglosando la composición de los mismos, se utilizó en cada test un fichero sql que declaraba las instrucciones de inserción de datos en la base de datos de pruebas necesarios para el correcto funcionamiento de los mismos. La base de datos de pruebas es una base de datos Mysql en local. relacional

Capítulo 2

Antecedentes

Este capítulo describe el proceso de adquisición de conocimientos necesarios para poder llevar a cabo el proceso de diseño arquitectónico y de construcción o implementacíon de la aplicación. De esta fomra se puede realizar una planificación mas cercana a la realidad y partir de unos conocimientos mínimos para empezar a elaborar el proyecto.

Contents

2.1. GO.JS

Go.JS [5] es una biblioteca de JavaScript para implementar editores gráficos dentro de interfaces web. GoJS facilita la implementación de funciones tales como definición de símbolos gráficos, gestión de paletas de símbolos, arrastrar y soltar (*drag and drop*), copiar y pegar, edición de etiquetas texto asociadas a símbolos gráficos, menús contextuales, función de deshacer o gestión de eventos, entre muchas otras funcionalidades.

2.2. Vaadin

Figura 2.1: Vaadin Logo

2.2.1. ¿Qué es GO.JS?

Vaadin [2] es un framework de desarrollo de SPA que permite escribir el código de dichas aplicaciones en Java o en cualquier otro lenguaje soportado por la JVM 1.6+. Esto permite la programación de la interfaz gráfica en lenguajes como Java 8, Scala o Groovy, por ejemplo.

2.2.2. Características

Uno de las características diferenciadores de Vaadin es que, contrario a las librerías y frameworks de JavaScript típicas, presenta una arquitectura centrada en el servidor, lo que implica que la mayoría de la lógica es ejecutada en los servidores remotos. Del lado del cliente, Vaadin está construido encima de Google Web Toolkit, con el que puede extenderse.

2.2.3. Aplicación

En este proyecto, Vaadin se encargara de realizar la comunicación entre el cliente y el servidor. De esta forma, será capaz de enviar y recibir datos, eventos y peticiones entre el componente Javascript (cliente) y el servidor.

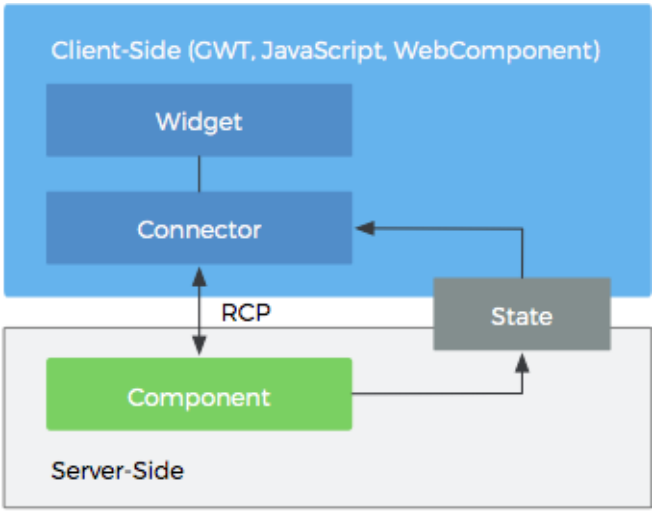

Figura 2.2: Esquema Cliente-Servidor

Capítulo 3

Process-Component

Este fragmento del proyecto o componente se dedica exclusivamente al apartado gráfico, el cuál posee una sintaxis ya definida en el apartado del diseño arquitectónico.

Este componente es una herramienta que se dedica a proveer métodos para interactuar con él y además es capaz mediante escuchadores de avisar a los clientes que lo requieran sobre los eventos ocurridos sobre la interfaz gráfica.

Su implementación se centra en dos pilares centrales. Por una parte se ha realizado un proyecto Vaadin encargado de mantener el estado de la aplicación gráfica, y por otro lado un conjunto de ficheros Javascript implementados sobre GO.JS que se encargan de modificar el entorno gráfico mediante sentencias .

- Proyecto Vaadin

  Este proyecto es el encargado de crear los metodos necesarios para interactuar desde el exterior con el esquema creado previamente. Además debe de permitir insertar escuchadores para los eventos proporcionados desde GO.JS, de forma que se pueda establecer dos direcciones de comunicaciones. Una desde el exterior con el proyecto Vaadin y este con el conector y directamente con el entorno gráfico de GO.JS, y otro desde interacción con los eventos (por parte del usuario) desde el fichero Javascript de configuración (explicado en el apartado posterior), con el conector y este con el estado, es decir, con el proyecto Vaadin.

- Ficheros Javascript

Existe un primer fichero que permite configurar el esquema gráfico que se va a llevar a cabo (Procesos, Subprocesos, InputVariables, OutputVariables ...), así como todo el resto de propiedades gráficas, ademas de ser capaz de lanzar eventos preconfigurados.

El segundo fichero esencial para el funcionamiento de esta estructura es el fichero conector. Este es el encargado de declarar y configurar todos los eventos que se pueden lanzar, además de ser el encargado de realizar todos los metodos CRUD [1]necesarios para que puedan interactuar con las propiedades configuradas en el primer fichero citado previamente.

Esta imagen trata de resumir el esquema de actuación que se lleva a cabo para la comunicación entre los distinto elementos del Process-Component.

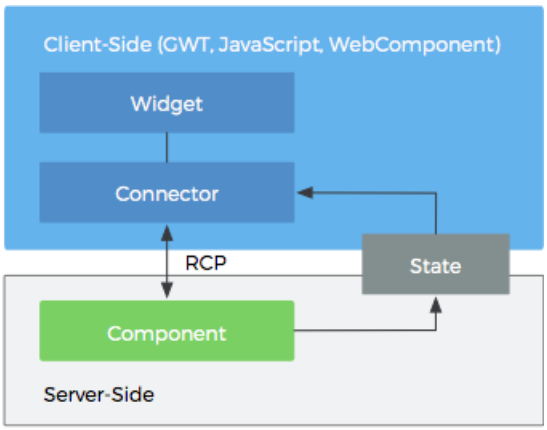

Figura 3.1: Estructuración en módulos

[1]CRUD es el acrónimo de crear, leer, actualizar y borrar, en esete contexto significa el conjunto de métodos para poder realizar dichas acciones sobre los distintos elementos existentes.

Capítulo 4

Luca-Process

Este capitulo describe el proceso de desarrollo e integración del componente gráfico (Process-Component). Se analizarán los aspectos del diseño y arquitectura a realizar así como las etapas de desarrollo del mismo.

4.1. Arquitectura

La arquitectura de Luca-Process tras la integración del Process-Component, cambia ligera o mínimamente respecto a la arquitectura demostrada en la introducción. La siguiente figura describe la diferencia respecto a la figura anterior:

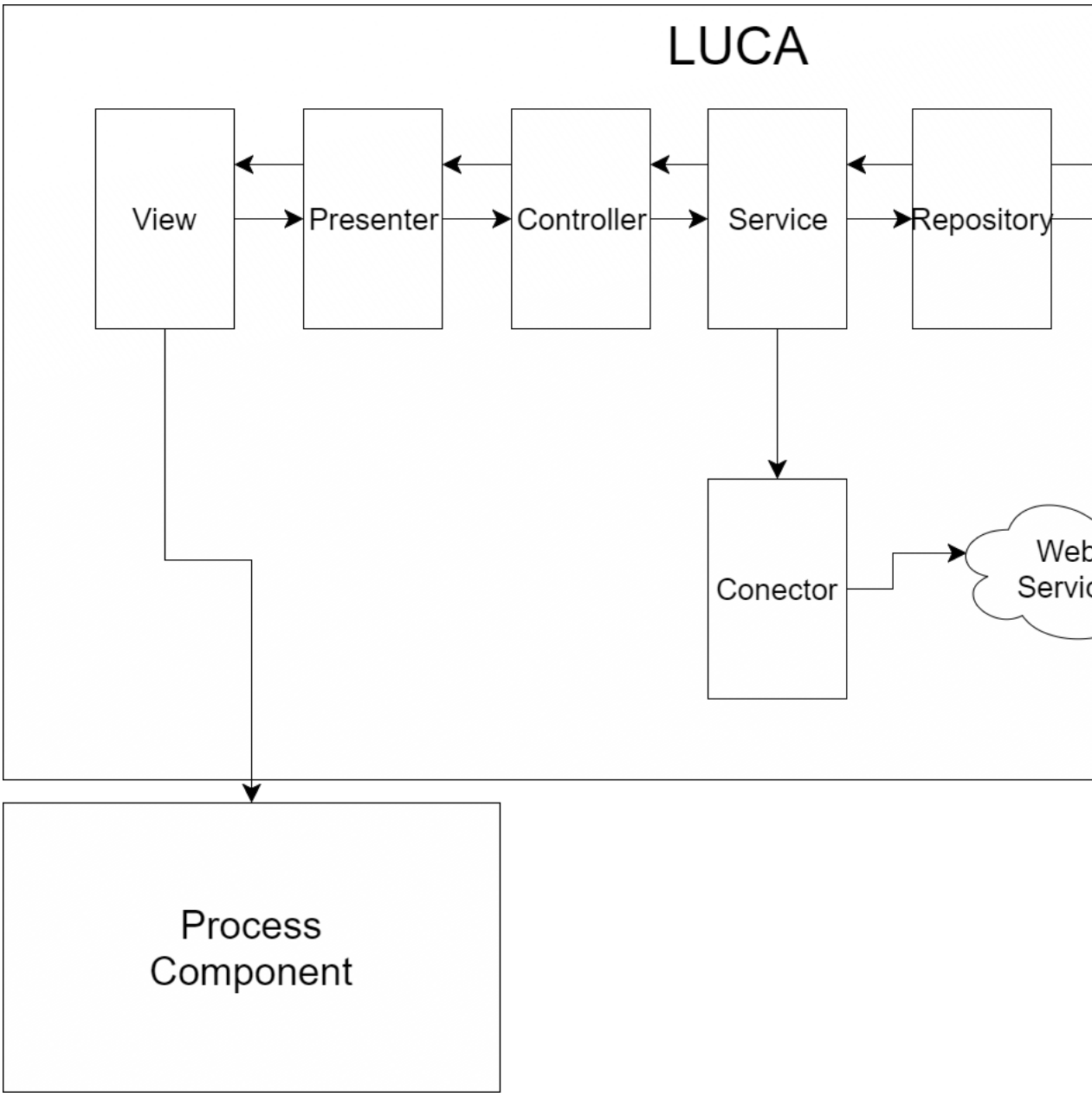

Figura 4.1: Nueva Arquitectura de LUCA

Como se puede apreciar, la única diferencia reside en que en esta citada nueva funcionalidad se utiliza otro proyecto aparte, el cuál se encarga de forma exclusiva de proveer componentes de visualización para poder abordar la sintaxis de procesos y subprocesos que se enlazan entre sí mediante entradas y salidas.

4.2. Desarrollo

En esta sección se describe el proceso de desarrollo o implementación del componente de LUCA que integra el componente gráfico.

El primer paso para empezar a comprender el diseño y arquitectura de LUCA, fue realizar una pequeña prueba de concepto aplicando el patrón MVP [3] e importando el componente gráfico para mostrar un proceso provisional, comprobando así el comportamiento del componente.

Una vez comprobado el funcionamiento, se comienza la implementación por capas del proyecto, creando la capa controladora junto con sus anotaciones de seguridad, la capa de servicio que realiza llamadas a la capa de repositorio (implementada con Spring Data [7]) para acceder a la base de datos, así como, al resto de servicios ya existentes en LUCA para la obtención de datos y ejecución de consultas. También se realizaron e implementaron, como se citó en la introducción, el conjunto de pruebas sobre la capa de servicio.

Con la solidez de las capas de datos implementadas, se prosiguió con el diseño de la capa de presentación para componer todas las vistas y los presentadores necesarios. Para ello hubo que, siguiendo el diseño preestablecido, diseñar para cada vista principal(aquellas vistas que albergan al resto y que forman una unidad de interacción completa, por ejemplo la vista de gestión de procesos, de creación o de ejecución) un conjunto de vistas de acciones y de una vista de visualización de datos, cada una implementada con una pareja de vista presentador.

Finalmente, tras realizar la implementación de componente, se realizó un comprobación y satisfacción de los requisitos generales descritos en la introducción.

Capítulo 5

Conclusiones

Para finalizar, a continuación se relatan las conclusiones sacadas tras el transcurso y ejecución del proyecto.

El objetivo principal del proyecto consistía en crear un proyecto que fuese capaz de permitir al usuario de forma gráfica y sencilla crear una jerarquía de procesos para enlazar las diferentes consultas existentes en el producto LUCA.

Tras completar con éxito la implementación del proyecto, se implanto en el producto LUCA, para comprobar el correcto funcionamiento e integración y comprobar así la correcta y eficiente satisfacción de los requerimientos impuestos al proyecto.

En el apartado personal, ha sido una gran oportunidad poder realizar este proyecto en una empresa como es CIC, ya que ayuda a tener una constancia y hábito de trabajo y nos prepara para el ámbito empresarial. Además ha sido muy satisfactorio poder trabajar con mas de una tecnología ya que aporta muchos conocimientos.

Bibliografía

[1] Apache. hc.apache.org/httpcomponents-client-ga/.

[2] Ari Handler Gamboa. adictosaltrabajo.com/tutoriales/introduccion-a-vaadin/.

[3] Eduardo Barrio. mhp-net.es/spring-vaadin-hibernate-tutorial-8-el-patron-mvp/.

[4] Erich Gamma And Kent Beck. junit.org/junit5/.

[5] Northwoods Software. gojs.net/latest/index.html.

[6] Oracle. www.oracle.com/technetwork/java/javase/jdbc/index.html.

[7] Pivotal Software. docs.spring.io/spring-data/jpa/docs/current/reference/html.

[8] Pivotal Software. docs.spring.io/spring/docs/current/spring-framework-reference/

[9] SeleniumHQ. www.seleniumhq.org.