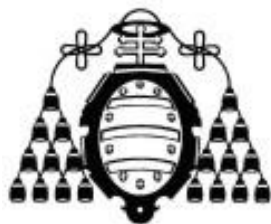


UNIVERSIDAD DE OVIEDO

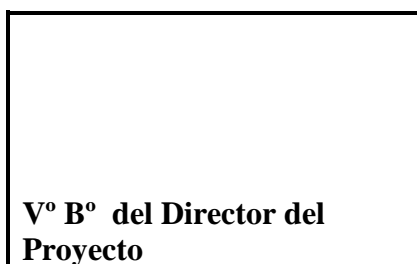


ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

“Plugin de eclipse para el reconocimiento de lenguajes propios”

DIRECTOR: Raúl Izquierdo Castañedo



AUTOR: Sergio Mosquera Dopico

Agradecimientos

Quiero agradecer, en primer lugar, a mis padres por su continuo apoyo y su preocupación continua sobre mis notas, y sobre todo por los tupperes que me han ido trayendo durante estos duros meses de TFG, y a mi hermana por ser la mejor compañera de piso posible.

A mi tutor, Raúl Izquierdo Castañedo por dar los mejores consejos para enfocar el proyecto de la mejor manera. Haciendo, además, que cada reunión fuese entretenida y consiguiendo que sacase muchas cosas en claro de cada una de ellas.

También agradecer a mis amigos de toda la vida por sus intentos de boicotear que aprobase este curso, enfadándose todos y cada uno de los fines de semana que decidía no salir por tener que estudiar. A los compañeros de la universidad, que a lo largo de los años de carrera han dejado de ser compañeros para convertirse en auténticos amigos, en especial a Los Púas, gracias a los cuales seguramente hubiese acabado con menos problemas la carrera, pero también con muchas menos risas.

Finalmente, quiero agradecer a toda la gente de Zapiens Technologies SL, que me dieron la oportunidad de hacer las prácticas con ellos cuando no tenían ninguna obligación, y que desde el primer momento me han permitido sentirme como uno más de la familia (además de dejarme faltar algún que otro día por ir apurado con algún examen o trabajo).

Muchas gracias a todos/as.

Resumen

La intención del proyecto es crear un generador de extensiones para distintos editores de código aprovechando El Protocolo del Servidor de Lenguaje. Las extensiones generadas permiten el reconocimiento de un lenguaje de programación cualquiera definido en una gramática ANTLR.

El objetivo principal es la creación de una herramienta Open Source para programadores que les permita desarrollar código en el editor que ellos deseen, sin necesidad de tener que utilizar un editor concreto en base a un lenguaje determinado.

Muchas de las características que ofrecen los editores de código (autocompletado, información flotante, coloreado de sintaxis, ...) son dependientes del propio lenguaje. Ahí reside la razón de querer desarrollar una herramienta Open Source. El programa proporciona extensiones funcionales y extensibles, para que quién desarrolle código incluya las funcionalidades propias del lenguaje que considere oportunas para trabajar de forma más cómoda.

Palabras Clave

LSP, NodeJS, ANTLR, Visual Studio Code, Cliente, Servidor, Código Abierto.

Abstract

The purpose of this project is to achieve a software that allows extension generation for several code editors which implement the LSP (Language Server Protocol). Those extensions allow to recognize a language defined by an ANTLR grammar.

The main goal is to create an Open Source tool for programmers that allow them choosing which editor use to program in any language, with no need to use an editor based on the language.

Most of the features offered by the code editors like intellisense, hover or syntax highlighting are dependent on the language. That's why I've decided to develop an Open Source tool to let other programmers fulfill this information. The project provides the programmers with functional and easy to extend plugins. This allows to use several language dependent functionalities which will make programming tasks easier for users.

Keywords

LSP, NodeJS, ANTLR, Visual Studio Code, Client, Server, Open Source.

Índice General

CAPÍTULO 1. MEMORIA DEL PROYECTO.....	13
1.1 RESUMEN DE LA MOTIVACIÓN, OBJETIVOS Y ALCANCE DEL PROYECTO	13
CAPÍTULO 2. INTRODUCCIÓN.....	15
2.1 JUSTIFICACIÓN DEL PROYECTO	15
2.2 OBJETIVOS DEL PROYECTO	17
2.3 ESTUDIO DE LA SITUACIÓN ACTUAL	19
2.3.1 Evaluación de Alternativas	19
CAPÍTULO 3. ASPECTOS TEÓRICOS.....	21
3.1 LANGUAGE SERVER PROTOCOL (LSP)	21
3.2 ARQUITECTURA CLIENTE-SERVIDOR.....	24
3.3 LENGUAJES DE PROPÓSITO GENERAL (GPL) Y LENGUAJES DE DOMINIO ESPECÍFICO (DSL)	26
3.4 LENGUAJE COMPILADO Y LENGUAJE INTERPRETADO	28
3.5 PLUGIN/COMPLEMENTO PARA UN EDITOR	30
3.6 OPEN SOURCE.....	31
CAPÍTULO 4. PLANIFICACIÓN DEL PROYECTO Y RESUMEN DE PRESUPUESTOS	33
4.1 PLANIFICACIÓN.....	33
4.2 RESUMEN DEL PRESUPUESTO	36
CAPÍTULO 5. ANÁLISIS	37
5.1 DEFINICIÓN DEL SISTEMA	37
5.2 REQUISITOS DEL SISTEMA	38
5.2.1 Obtención de los Requisitos del Sistema	38
5.2.2 Identificación de Actores del Sistema	40
5.2.3 Especificación de Casos de Uso	40
5.3 IDENTIFICACIÓN DE LOS SUBSISTEMAS EN LA FASE DE ANÁLISIS	43
5.3.1 Descripción de los Subsistemas	43
5.3.2 Descripción de los Interfaces entre Subsistemas	45
5.4 DIAGRAMA DE CLASES PRELIMINAR DEL ANÁLISIS.....	46
5.4.1 Diagrama de Clases	46
5.4.2 Descripción de las Clases	47
5.5 ANÁLISIS DE CASOS DE USO Y ESCENARIOS	52
5.5.1 Generación de extensiones.....	52
5.5.2 Interacción con el editor	54
5.6 ANÁLISIS DE INTERFACES DE USUARIO	57
5.6.1 Descripción de la Interfaz	57
5.6.2 Descripción del Comportamiento de la Interfaz	58
5.7 ESPECIFICACIÓN DEL PLAN DE PRUEBAS.....	60
5.7.1 Niveles de prueba	60
5.7.2 Tipos de prueba	61
5.7.3 Casos de uso	63
CAPÍTULO 6. DISEÑO DEL SISTEMA	65
6.1 ARQUITECTURA DEL SISTEMA	65
6.1.1 Diagramas de Paquetes.....	65

6.1.2	Diagramas de Componentes.....	69
6.1.3	Diagramas de Despliegue	70
6.2	DISEÑO DE CLASES	72
6.3	DISEÑO DE LA INTERFAZ.....	73
6.4	ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS	74
6.4.1	Pruebas de componentes.....	74
6.4.2	Pruebas de sistema	77
6.4.3	Pruebas de aceptación.....	79
CAPÍTULO 7.	IMPLEMENTACIÓN DEL SISTEMA	83
7.1	ESTÁNDARES Y NORMAS SEGUIDOS	83
7.1.1	LSP.....	83
7.1.2	JSON-RPC.....	83
7.2	LENGUAJES DE PROGRAMACIÓN	84
7.2.1	Bash	84
7.2.2	Batch	85
7.2.3	JavaScript	86
7.2.4	TypeScript.....	88
7.3	HERRAMIENTAS Y PROGRAMAS USADOS PARA EL DESARROLLO	89
7.3.1	Node.js	89
7.3.2	ANTLR4.....	91
7.3.3	Visual Studio Code.....	92
7.3.4	Node Package Manager.....	94
7.4	CREACIÓN DEL SISTEMA.....	96
7.4.1	Problemas Encontrados	96
CAPÍTULO 8.	DESARROLLO DE LAS PRUEBAS	101
8.1	PRUEBAS DE COMPONENTES	101
8.2	PRUEBAS DE SISTEMA	103
8.3	PRUEBAS DE ACEPTACIÓN	104
8.3.1	Usuario 1	104
8.3.2	Usuario 2.....	106
8.3.3	Usuario 3.....	108
CAPÍTULO 9.	MANUALES DEL SISTEMA	111
9.1	MANUAL DE INSTALACIÓN	111
9.1.1	Instalación de Node.js.....	111
9.1.2	Instalación de Typescript	113
9.2	MANUAL DE EJECUCIÓN	114
9.3	MANUAL DE USUARIO.....	115
9.4	MANUAL DEL PROGRAMADOR	119
CAPÍTULO 10.	CONCLUSIONES Y AMPLIACIONES	121
10.1	CONCLUSIONES.....	121
10.2	AMPLIACIONES	121
10.2.1	Internacionalización del sistema	121
10.2.2	Soporte para nuevos editores.....	122
10.2.3	Interfaz gráfica para la generación de extensiones.....	122
10.2.4	Soporte para otras herramientas de generación de lenguajes.....	123
10.2.5	Base de datos para reutilizar los procesadores de lenguaje.....	123
CAPÍTULO 11.	REFERENCIAS BIBLIOGRÁFICAS	125

11.1	LIBROS Y ARTÍCULOS.....	125
11.2	REFERENCIAS EN INTERNET	126
CAPÍTULO 12.	APÉNDICES.....	129
12.1	DEFINICIONES Y ACRÓNIMOS.....	129
12.2	ÍNDICE ALFABÉTICO	130
12.3	ENLACE A GITHUB CON EL CÓDIGO DEL SISTEMA	132

Índice de Tablas

Tabla 1. Tareas de la planificación del proyecto.....	35
Tabla 2. Resumen del coste del proyecto por fases.....	36
Tabla 3. Resumen de los recursos del proyecto	36
Tabla 4. Diseño de pruebas para el cliente.....	74
Tabla 5. Diseño de pruebas para el servidor	75
Tabla 6. Diseño de pruebas para el procesador de lenguaje	76
Tabla 7. Diseño de pruebas de sistema	78
Tabla 8. Preguntas de carácter general	80
Tabla 9. Preguntas cortas sobre la aplicación	81
Tabla 10. Cuestionario para el responsable	81
Tabla 11. Resultado de las pruebas para el componente cliente	101
Tabla 12. Resultados de las pruebas para el componente servidor	101
Tabla 13. Resultados de las pruebas para el procesador de lenguaje	102
Tabla 14. Resultados de las pruebas de sistema	103
Tabla 15. Preguntas de carácter general para el usuario 1	104
Tabla 16. Preguntas cortas sobre la aplicación para el usuario 1	105
Tabla 17. Cuestionario para el responsable de las pruebas para el usuario 1.....	105
Tabla 18. Preguntas de carácter general para el usuario 2	106
Tabla 19. Preguntas cortas sobre la aplicación para el usuario 2	107
Tabla 20. Preguntas para el responsable de las pruebas para el usuario 2	107
Tabla 21. Preguntas de carácter general para el usuario 3	108
Tabla 22. Preguntas cortas sobre la aplicación para el usuario 3	109
Tabla 23. Preguntas para el responsable de las pruebas para el usuario 3	109

Índice de ilustraciones

Ilustración 1. Descomposición de matriz de dependencias con LSP	16
Ilustración 2. Comunicación entre cliente y servidor según el protocolo LSP	21
Ilustración 3. Edición simultánea con más de un servidor de lenguaje	22
Ilustración 4. Estructura cliente-servidor del proyecto	24
Ilustración 5. Proceso de compilación de un programa	29
Ilustración 6. Diagrama de casos de uso para generación de extensiones	40
Ilustración 7. Diagrama de casos de uso para la interacción con el editor	41
Ilustración 8. Relación entre los distintos subsistemas	46
Ilustración 9. Pantalla de visualización de procesadores de lenguaje	57
Ilustración 10. Editores con soporte para LSP	58
Ilustración 11. Diagrama de componentes del proyecto	69
Ilustración 12. Diagrama de despliegue de la extensión LSP	70
Ilustración 13. Diagrama de despliegue de la extensión de coloreado	71
Ilustración 14. StackOverflow Survey 2018	92
Ilustración 15. Página de instalación de npm	111
Ilustración 16. Wizard de instalación de Node	112
Ilustración 17. Versiones de Node y npm	112
Ilustración 18. Comando de instalación	113
Ilustración 19. Comando de comprobación de versión	113
Ilustración 20. Situación inicial antes de la solución	115
Ilustración 21. Comando para ejecutar el programa	115
Ilustración 22. Se requiere una entrada por parte del usuario	116
Ilustración 23. Proceso de generación de las extensiones	116
Ilustración 24. Proceso de generación finalizado	117
Ilustración 25. Resultado tras la instalación de las extensiones	117
Ilustración 26. Salida proporcionada por el procesador de lenguaje	118

Capítulo 1. Memoria del Proyecto

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

A raíz de la reciente aparición del LSP (Language Server Protocol), desarrollado por Microsoft desde el 4 de septiembre de 2015, el cual reinventa la forma de desarrollar extensiones o complementos de reconocimiento de lenguajes de programación para los editores de texto y entornos de desarrollo integrados (IDE), surge la idea de crear una herramienta que facilite la labor de desarrollar software para programadores/as.

Llegando a una concepción bastante idealista, se podría pensar en la posibilidad de la aparición de un nuevo lenguaje con gran repercusión en el mundo de la informática (como lo fue C en su momento), y el tiempo que sería necesario emplear para disponer de un soporte adecuado para comenzar a desarrollar programas para este nuevo lenguaje. Gracias al proyecto desarrollado, el propio creador del lenguaje (o cualquiera que desee trabajar con él) tendría a su disposición herramientas suficientes para que cualquier entorno de programación que permita la integración con el LSP ofrezca varias utilidades para el desarrollo de programas.

Continuando con la situación anterior, al tratarse de un proyecto Open Source que se encontraría alojado en un repositorio público, se promueve la discusión en la comunidad para posibles mejoras, tanto del programa, como de los posibles lenguajes de programación desarrollados por cualquier persona.

Pensando en una situación más común, podemos pensar en las personas que, por necesidades profesionales o personales, se ven obligados a utilizar una herramienta de desarrollo con la que no se sienten cómodas ya que son las únicas que proporcionan soporte para el lenguaje utilizado. Gracias a este proyecto, y al creciente número de clientes que proporcionan soporte para el LSP, estas personas podrían desarrollar software de la misma forma en un entorno con el que se sientan más cómodos.

Otra situación más realista sería la utilización de la herramienta para la asignatura de Diseño de Lenguajes de Programación impartida en la escuela. La forma más sencilla de comprobar la validez de la gramática creada sería simplemente escribiendo programas en el editor favorito de cada alumno, permitiéndoles ver en cada momento los errores que aparecen para que puedan llevar a cabo los arreglos correspondientes. Esto se puede llevar a cabo gracias a la herramienta desarrollada.

Como todo proyecto, la herramienta posee una serie de limitaciones a tener en cuenta:

- **Características dependientes del lenguaje o del usuario.** El protocolo mencionado proporciona varias características con las que se encuentran familiarizadas las personas acostumbradas a utilizar IDEs conocidas (Eclipse o IntelliJ entre otras). La gran mayoría de esas características son dependientes del lenguaje proporcionado por

el usuario. Por tanto, no las proporciona la extensión generada y han de ser ampliadas por el propio usuario. Dichas características son las siguientes:

- Autocompletado del código: muy utilizado para las keywords o los nombres de variables. Probablemente la funcionalidad más utilizada por los programadores/as de todo el mundo, permite con un simple atajo de teclado (o de ratón en su defecto) completar el término que el usuario se encuentre escribiendo en ese momento.
- Saltar a la definición: también muy utilizado para encontrar en ciertos lenguajes el lugar donde se declaran las variables y las funciones.
- Coloreado de la sintaxis: llevar a cabo una agrupación de términos que se consideren comunes estableciendo para ellos un color de fuente diferente al resto facilita mucho la lectura del código. Desgraciadamente, esto es una característica que depende puramente del gusto del usuario para elegir los colores de las distintas reglas. Además, actualmente esto solo está permitido a partir de otras extensiones y no a partir del LSP, aunque está planeado introducirlo a medio plazo.
- Información sobre los elementos: posar el ratón sobre un término para obtener más información sobre él es algo que un programador realiza casi de forma instantánea cuando tiene una duda acerca de él. El cuadro (normalmente es un cuadro de texto) que se despliega en esos casos para mostrar la información correspondiente corre a cargo de una característica propia del lenguaje.
- **No todos los entornos lo soportan por el momento**. Desde el año 2016 se convirtió en un estándar, y la gran mayoría de los entornos dan soporte para desarrollar aplicaciones relacionadas con él. Esta es la tónica que se seguirá en los próximos años, ya que se pretende que todos los editores nuevos que aparezcan en los próximos años lo implementen, así como los ya existentes. Aunque por el momento, los más destacados si que permiten todas sus funcionalidades (Eclipse, VSCode, IntelliJ, Sublime, ...) existen otros muy destacados como Atom o Emacs que no soportan todavía todas las características ofrecidas por el LSP.
- **Limitado a análisis léxico y sintáctico**. Esto no resulta un problema para lenguajes interpretados como son JavaScript, Python o Ruby, ya que no requieren de pasos adicionales por parte del editor para poder ser ejecutados, más allá de escribir correctamente sus programas. No es así para los lenguajes compilados como pueden ser Java, C++ o Go, estos lenguajes necesitan ser “traducidos” a lenguaje máquina una vez codificados para poder ejecutarse de forma correcta, lo cual debe hacerse con una herramienta externa al editor si este no proporciona por defecto las herramientas necesarias para llevar a cabo la compilación.

Capítulo 2. Introducción

2.1 Justificación del Proyecto

En un primer momento, el trabajo planteado al tutor del proyecto fue un generador de plugins para el reconocimiento de un lenguaje propio para el entorno de desarrollo de Eclipse, ya que era el que habíamos utilizado a lo largo del tercer año en la asignatura de Desarrollo de Lenguajes de Programación, por tanto se me ocurrió que podría ser interesante que los alumnos pudiesen desarrollar programas escritos en el lenguaje creado en clase y ver en tiempo real los errores y utilizar otras funcionalidades como el autocompletado o similares.

Esta idea tenía como meta, entre otras cosas, permitir al usuario elegir el color de la sintaxis mediante un código de etiquetas en la gramática de forma similar a como se llevó a cabo para unir las reglas definidas con los nodos del árbol AST generado.

Desde el primer momento comenzaron a surgir problemas ya que la documentación, además de escasa, estaba bastante desactualizada. También con respecto al desarrollo existía un problema porque el SDK utilizado para la creación de plugins era muy complejo si nunca se había trabajado en el desarrollo de los mismos. Sin embargo, el principal problema consistía en la interpretación y detección de errores de las gramáticas para la generación de la extensión.

Al tener que detectar las etiquetas especificadas por el usuario para el coloreado de la sintaxis serían necesarios dos análisis de la gramática (la primera por la herramienta encargada de la comprobación de la corrección de la gramática y de la generación de las clases correspondientes), de las cuales yo tendría que encargarme de la segunda mediante expresiones regulares con el fin de detectar los colores para los distintos términos.

A raíz de esta problemática me surgió la necesidad de pensar en otras alternativas, fue entonces cuando aparece la posibilidad del Language Server Protocol, el cual se encontraba en una fase bastante temprana de su implantación. La idea de permitir la misma funcionalidad de reconocimiento de lenguajes para varios IDEs diferentes reutilizando el servidor y la lógica del lenguaje, me hizo ver con mucha perspectiva que aportaba una solución mejor a la que tenía actualmente en mente. Además, el tratarse de un estándar con finalidad Open Source y desarrollado por Microsoft, significaba documentación actualizada, clara y completa, y resolución de dudas en una comunidad muy amplia. Como era de esperar poseía algunos defectos, pero eran totalmente asumibles ya que las posibilidades de esta opción las superaban con creces. La idea fundamental del LSP es muy simple, propone la separación total entre el reconocimiento del lenguaje y las dependencias de un editor de código o IDE, de forma que con un mismo cliente el entorno pueda conectarse a servidores de lenguaje diferentes.



Ilustración 1. Descomposición de matriz de dependencias con LSP

2.2 Objetivos del Proyecto

Continuando con la información incluida en el apartado 1.1, se pueden identificar una serie de objetivos que se buscan alcanzar con la realización del proyecto:

1. **Dar a conocer el Language Server Protocol.** Desde 2016, este protocolo se encuentra en proceso de estandarización, por lo que en unos años llegará a ser una herramienta bastante conocida debido a las posibilidades que presenta. Por esta razón, cuanto antes se de a conocer y se proporcionen facilidades para el desarrollo de servidores de lenguaje, más fácil resultará a los usuarios conocer sus funciones y utilidades.
2. **Crear una comunidad a partir de un proyecto Open Source para conseguir una herramienta más robusta.** El proyecto se encontrará alojado en un repositorio público (Github) bajo una licencia MIT, la cual permite a cualquier usuario que lo desee visualizar el código y modificarlo a su gusto. Además, Github permite la creación de issues referentes al proyecto para discutir distintos fallos, dudas o mejoras. Estas issues tienen especial repercusión cuando el proyecto posee reconocimiento a cierto nivel.
3. **Generar herramientas de reconocimiento de lenguajes cada vez más potentes.** Al tratarse de una herramienta relativamente reciente (el LSP), es común pensar que aún hay muchas características dependientes del lenguaje que no se encuentran implementadas en estas fases tempranas de desarrollo. Sin ir más lejos, el coloreado de la sintaxis es algo que se tiene en mente para futuras versiones, lo cual otorgaría a las extensiones generadas más utilidades y facilidades para los usuarios a la hora de desarrollar código. Este objetivo se aprovecharía del anterior, ya que los usuarios podrían contribuir unos con otros con las extensiones generadas
4. **Facilitar la tarea de desarrollo de una gramática para los alumnos de DLP.** Como se ha mencionado en apartados anteriores, la aplicación más inmediata y realista del programa sería para la asignatura de DLP, en la que los alumnos tendrían la posibilidad de probar la gramática desarrollada a lo largo de todo el curso en otros editores de código pudiendo encontrar posibles errores en la gramática.
5. **Permitir a cualquier persona desarrollar código en un mayor rango de editores.** Normalmente las personas que tienen que programar poseen un editor de código o IDE preferido con el que desarrollar código. Desgraciadamente, no todos los lenguajes de programación se encuentran disponibles para todos los editores, pero eso es precisamente lo que el LSP quiere conseguir. Además, es importante destacar que la gran mayoría de clientes y servidores para lenguajes importantes han sido y están siendo desarrollados por empresas con mucho renombre y la gran mayoría con licencias que permiten la modificación y distribución del código. Esto es una gran noticia para los desarrolladores ya que podrán, entre otras cosas, añadir funcionalidades extra para el reconocimiento de lenguajes o mensajes personalizados para las distintas funcionalidades existentes.
6. **Impulsar la creación de lenguajes de programación y los DSL¹.** En la hipotética situación de que apareciese un lenguaje de propósito general realmente interesante el cual, por las razones que fuesen, no obtuviese el suficiente renombre para llegar a ser conocido por el mundo, la utilización de este proyecto resultaría beneficioso, ya que

¹ https://www.fing.edu.uy/inco/seminarios/papi/web2009/PresentacionesPDF/Pres_Lopez.pdf

ofrece la posibilidad de disponer de un medio que proporciona el suficiente soporte (reconocimiento) para dicho lenguaje en los principales entornos de desarrollo. Esto, sumado a la comunidad mencionada en el punto 2 impulsaría su renombre dando a conocer lenguajes en un principio no muy influyentes a todo el mundo, reduciendo la brecha entre las grandes empresas y las pequeñas empresas o los desarrolladores individuales.

Con respecto a los DSL, pueden resultar muy útiles para ciertos procesos dentro de empresas con objetivos específicos, el problema es como siempre la falta de soporte para el desarrollo de un lenguaje propio. Gracias a la herramienta, el reconocimiento de los DSL resulta realmente sencillo.

2.3 Estudio de la Situación Actual

Durante la etapa de investigación y búsqueda de información, traté de encontrar herramientas que llevaran a cabo la misma tarea que la que pretendo conseguir con mi proyecto. No obstante, al tratarse el LSP de un protocolo relativamente nuevo, no existe gran cantidad de herramientas desarrolladas con este fin. Sin embargo, sí encontré al menos dos herramientas que persiguen fines similares. Además, tuve en cuenta la situación actual para el desarrollo de extensiones de reconocimiento de lenguajes para los distintos editores de código. Antes de la aparición del LSP, el desarrollo de extensiones o plugins para el reconocimiento de lenguajes se realizaba de forma individual para cada editor, con las herramientas proporcionadas por cada uno. Estas herramientas, presumiblemente, no tienen nada que ver unas con otras principalmente por que suelen proporcionar soporte en un lenguaje concreto por cada editor. Poniendo como ejemplo los plugins de Eclipse en contraposición con los de Atom, los del primero se desarrollan en Java, un lenguaje compilado, mientras que los del segundo en JavaScript, un lenguaje interpretado. Respecto a la documentación, los plugins de Eclipse pueden llevarse a cabo siguiendo documentación cuya última revisión fue en 2003 (respetando los temas de dependencias actualizadas), mientras que en Atom, al ser un editor relativamente nuevo, posee una documentación actualizada, interactiva y visualmente interesante para el usuario.

2.3.1 Evaluación de Alternativas

Las herramientas identificadas persiguen alcances diferentes. Esto se debe a que, por una parte, una de ellas es una herramienta destinada al negocio, siendo el producto principal de una empresa; mientras que la otra herramienta es con fines ilustrativos ya que es explicada en un blog personal de un profesional de la informática. Las herramientas son las siguientes:

2.3.1.1 DSLForge

Esta herramienta desarrollada por la empresa del mismo nombre² consiste en un plugin para el entorno de desarrollo Eclipse. Este plugin permite la generación de un editor web a partir de una gramática Xtext. Al tratarse de un plugin para Eclipse, podemos determinar que el lenguaje utilizado para desarrollarlo ha sido Java.

- **Ventajas:** la posibilidad de generar un proyecto web que se ejecuta en un navegador, da más libertad al usuario para usar un navegador de su preferencia fomentando así un software portable. El tiempo empleado en la creación de un editor de texto es relativamente pequeño, ya que, el programa se encarga de generarnos el proyecto necesario para ejecutar el editor en nuestro navegador.
- **Desventajas:** es necesario tener instalado el entorno de desarrollo de Eclipse y estar familiarizado con su uso, así como es necesario poseer conocimientos de instalación y utilización de plugins. Otra desventaja es la limitación a una herramienta de creación de gramáticas

² <https://dslforge.org/>

concreta, actualmente la herramienta más utilizada para la creación de gramáticas para lenguajes de programación es ANTLR, por lo que será necesario tener un conocimiento de desarrollo en Xtext. El soporte se ve limitado a la página web de la empresa, ya que no es una herramienta muy conocida. Además, no existe una comunidad que pueda ayudar ante posibles problemas, lo cual acentúa esta carencia de soporte. Además de lo mencionado anteriormente, el editor no da pie a una customización de la interfaz por parte del usuario. La interfaz por defecto es la correspondiente a un editor de texto de Eclipse.

2.3.1.2 *Kanvas*

Se trata de un proyecto independiente desarrollado por el arquitecto de software Federico Tomassetti, el cual desglosa la información correspondiente en un artículo en su página web³. Este framework desarrollado en Kotlin y que permite la creación de un editor de código que permite el reconocimiento de lenguajes definidos en gramáticas ANTLR.

- **Ventajas:** la posibilidad de reconocer más de un lenguaje distinto en el mismo editor, proporciona más versatilidad a la hora de desarrollar código para proyectos en los que interactúen ficheros escritos en distintos lenguajes de programación. Al tratarse de un proyecto alojado en Github, además del soporte ofrecido por el creador, existe una comunidad muy amplia que notificará mediante mensajes sobre posibles errores. Dichos mensajes podrán ser visualizados por los usuarios de Github. Además, al tratarse de un proyecto abierto, permite la colaboración para ampliar el proyecto o pulir nosotros mismos los fallos encontrados en el código.
- **Desventajas:** posee un SDK propio, el cual al tratarse de un programa personal sin apenas soporte a nivel profesional, podría tener fallos que dificulten el buen desempeño del programa. Además de que será necesario un proceso de aprendizaje de la sintaxis necesaria para realizar programas utilizando el SDK mencionado. Proporciona una interfaz por defecto no customizable (sin tener que realizar cambios en el código) la cual puede llegar a resultar agobiante o aburrida.

³ <https://tomassetti.me/kanvas-generating-simple-ide-antlr-grammar/>

Capítulo 3. Aspectos Teóricos

3.1 Language Server Protocol (LSP)

El protocolo del servidor de lenguaje, o LSP, es un protocolo desarrollado por Microsoft desde el 4 de septiembre de 2015 para facilitar la comunicación entre un editor (cliente) y un proveedor de lógica del lenguaje (servidor) y favorecer el uso de funcionalidades propias del lenguaje. Tradicionalmente, permitir características como el autocompletado, la búsqueda de la definición o la documentación con el hover era un trabajo costoso ya que se tenía que hacer de forma individual para cada herramienta. Cada uno proporcionaba una API diferente para implementar las mismas funcionalidades, con lo cual la dificultad del trabajo era proporcional al número de lenguajes y de editores para los que quisieran los desarrolladores dar soporte.

La idea final de este protocolo consiste en estandarizar las comunicaciones entre los entornos de desarrollo y los servidores de lenguaje. De esta forma, un mismo servidor de lenguaje puede ser reutilizado para distintos clientes sin necesidad de ser modificado. La utilización del LSP es una estrategia beneficiosa tanto para proveedores de lenguajes como para los proveedores de entornos de desarrollo.

El funcionamiento del LSP consiste básicamente en separar las responsabilidades de cada una de las partes, estableciendo una comunicación JSON-RPC entre ellas. El servidor puede recibir varios eventos dependiendo de la actividad del usuario con respecto al lenguaje, estos eventos se explican a continuación:

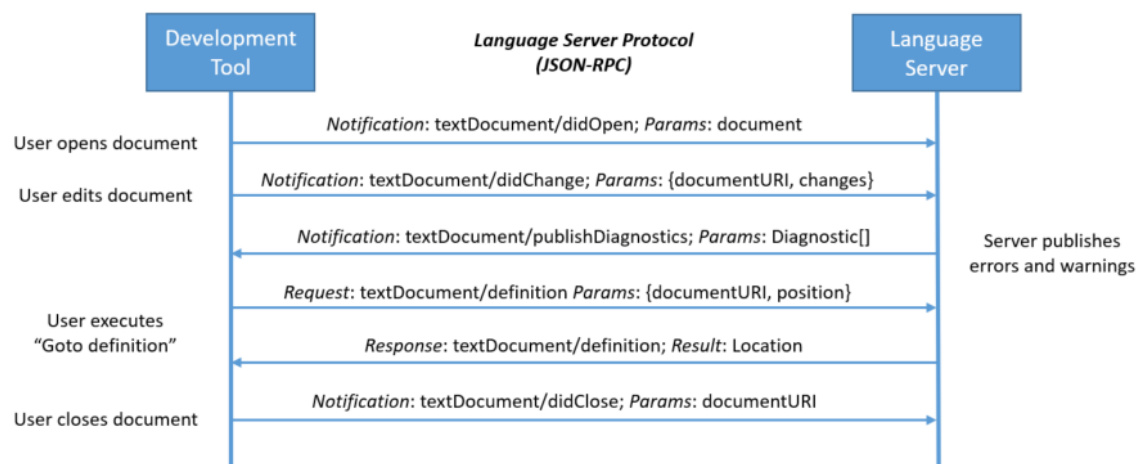


Ilustración 2. Comunicación entre cliente y servidor según el protocolo LSP

- **El usuario abre un fichero en el editor:** cada cliente tiene especificada la extensión de los ficheros para los cuales debe notificar al servidor. Una vez abierto uno de estos ficheros, el cliente envía una notificación al servidor de que un documento con la extensión correspondiente se ha abierto. Además de la notificación, el servidor recibe bien una parte del fichero, o bien el fichero entero (depende de la configuración especificada para ello). También existen otras opciones de configuración como el

número máximo de problemas que el editor notificara al usuario que tendrán que ser especificados durante la creación del cliente.

- **El usuario realiza cambios en el fichero.** Al igual que en el caso anterior, el servidor recibe una notificación junto con el contenido del fichero, además de la posición exacta en el fichero donde se ha producido la modificación. El servidor analiza la información recibida con respecto a la lógica del lenguaje reconocido y envía un mensaje al cliente con un diagnóstico de los errores y los warnings detectados.
- **El usuario utiliza la funcionalidad de “ir a la definición” de un símbolo del fichero.** En este caso, el servidor recibe la URI del documento y la posición del símbolo del que el usuario quiere ir a la definición. La respuesta del usuario consiste en la URI del documento y la posición en el documento de la definición del símbolo.
- **El usuario cierra el fichero.** Se envía una notificación desde el cliente informando de que el documento ha sido cerrado y ya no se encuentra en memoria.

La información enviada utiliza tipos de datos neutros con el fin de poder funcionar con todos los lenguajes de programación. En lugar de buscar construir un árbol sintáctico o una codificación a partir de símbolos de compilación, resulta mucho más simple utilizar este tipo de datos de cara a conseguir una estandarización del protocolo.

Una duda normal en este punto sería “¿Qué ocurriría si deseo editar dos o más ficheros que se encuentren en distintos lenguajes?”. La respuesta es muy sencilla, para cada fichero editado en un lenguaje diferente, el editor ejecuta su servidor de lenguaje correspondiente. Por tanto, un usuario puede encontrarse editando dos ficheros en idiomas diferentes sin ningún problema.

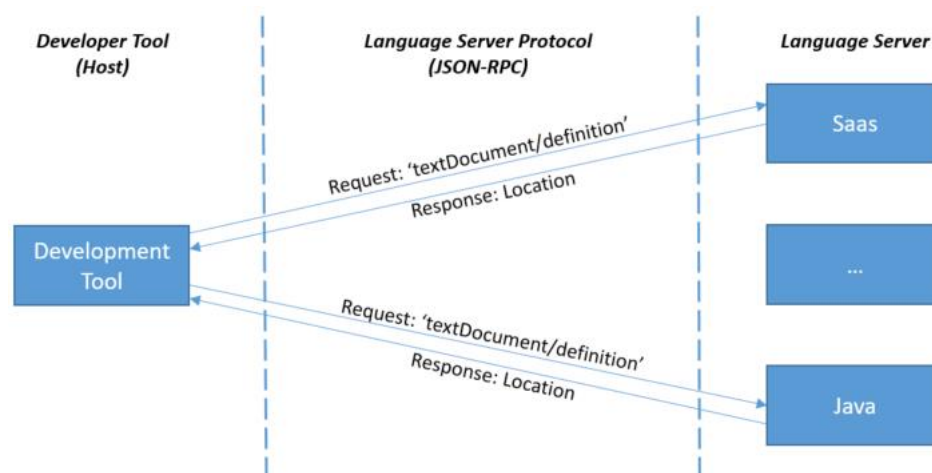


Ilustración 3. Edición simultánea con más de un servidor de lenguaje

Respecto a las funcionalidades que ofrece el LSP, desgraciadamente no todos los servidores ni todos los clientes soportan todas las características. Para solventar esto, se promueve que en la descripción correspondiente de la herramienta desarrollada (ya sea un servidor o un cliente) se especifique las características que es capaz de soportar, para facilitar la integración posterior con la otra parte.

Para desarrollar estas herramientas existen SDKs para varios lenguajes de programación. Los SDK se suelen separar en dos partes, los correspondientes a la parte del servidor, y los correspondientes a la parte del cliente, con lo que la interconexión entre ellos vendrá

determinada por los desarrolladores de las herramientas. Un ejemplo de estos SDKs es el ofrecido para Javascript o Typescript para la parte del servidor, el cual posee un módulo npm propio.

3.2 Arquitectura cliente-servidor

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

Esta metodología se aplica al desarrollo del proyecto ya que las extensiones desarrolladas son una aplicación cliente-servidor en sí misma. Siendo los clientes las extensiones que se instalan en los editores o IDEs, las cuales permiten la comunicación con un servidor interno que se encarga del análisis del código enviado por los clientes. La comunicación entre el cliente y el servidor se lleva a cabo a través de una comunicación JSON-RPC. También se pueden utilizar otras formas de comunicación como es la comunicación por el protocolo HTTP, que es la forma de comunicación establecida entre el servidor y la lógica del lenguaje en este proyecto.



Ilustración 4. Estructura cliente-servidor del proyecto

Esta metodología aporta una serie de ventajas importantes a tener en cuenta a la hora de utilizarlo, a pesar de que no todas apliquen para nuestro proyecto:

- **Centralización del control.** 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.
- **Escalabilidad.** Se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).
- **Facilidad de mantenimiento.** Esta metodología es muy utilizada de forma distribuida (en distintos ordenadores), con lo cual facilita el control individual de los recursos

manejados por cada uno y su posterior mantenimiento. Para este proyecto no aplica esta ventaja, al ejecutarse el servidor de forma local en nuestro ordenador.

- **Encapsulación.** Al realizarse la comunicación entre cliente y servidor por un protocolo independiente de la implementación, es posible realizar cambios en el propio servidor o en los clientes, sin que la otra parte se vea afectada (o si se ve afectada lo hará mínimamente).
- **Seguridad.** La única comunicación que se lleva a cabo entre cliente y servidor es a partir de los protocolos definidos para dicho fin de forma que, asegurando las comunicaciones (simplemente dos puntos del sistema), aseguramos el sistema al completo.

A pesar de estas ventajas, también aparecen una serie de desventajas de las cuales, al igual que en el caso anterior, no todas aplican para el proyecto por las razones explicadas a continuación:

- **Congestión del tráfico.** Esto ocurre cuando una gran cantidad de clientes envían peticiones simultáneas al servidor y este no posee capacidad suficiente para atender a todas de forma correcta. Esta desventaja no aplica para el proyecto debido a que, al ejecutarse el servidor en el propio ordenador donde se encuentra el cliente, el usuario no podrá desarrollar código desde dos editores al mismo tiempo.
- **Single Point of Failure.** Esta es probablemente la principal desventaja de esta metodología: si el servidor no funciona, el sistema entero es inservible. Los clientes no pueden establecer conexión con el servidor ni recibir la información correspondiente si este no se encuentra disponible, al igual que el validador del lenguaje no recibirá la información para evaluarla.
- **Coste del procesamiento.** Cuando el software requiere una gran cantidad de recursos para llevar a cabo su ejecución, se precisa de hardware con alta capacidad de procesamiento. Por lo general, este hardware no suele ser barato, y no disponer de él, puede suponer un rendimiento bastante pobre del ordenador. En el caso del proyecto, no es necesario un gran rendimiento por parte del ordenador ya que, como se ha explicado antes, el usuario no tendrá la posibilidad de programar simultáneamente desde dos editores, por lo que el servidor correspondiente solo recibirá peticiones de un cliente.

3.3 Lenguajes de propósito general (GPL) y lenguajes de dominio específico (DSL)

Existe una distinción en los lenguajes (no solo de programación) en el ámbito de la informática dependiendo de su lenguaje. Si la aplicación del lenguaje se puede realizar a lo largo de varios dominios sin poseer características concretas de un dominio en particular, se considera que el lenguaje es de propósito general. En el caso contrario encontramos los lenguajes de dominio específico, los cuales están limitados a un dominio concreto definido durante la creación del lenguaje.

Respecto a los lenguajes de propósito general podemos realizar una división según su tipo e identificar tres variantes:

- **Lenguajes de marcado de propósito general.** Un lenguaje de marcado es un sistema utilizado para realizar anotaciones en un documento cuya sintaxis se pueda distinguir del propio texto del documento. Normalmente, los lenguajes de marcado de dominio específico se basan en este tipo de lenguajes para su implementación. Algunos lenguajes destacables de esta variante son SGML, XML o YAML.
- **Lenguajes de programación de propósito general.** Esta variante está orientada a la implementación de software para un rango amplio de dominios de aplicación. Los distintos tipos de lenguajes de programación se identifican según el paradigma que siguen; Java, Javascript, C#, Ruby o Python son algunos de los lenguajes de programación más conocidos que siguen el paradigma de programación orientada a objetos.
- **Lenguajes de modelado de propósito general.** Se utilizan para representar una serie de características de un objeto o un sistema. El ejemplo más representativo de este tipo es UML.

Para los lenguajes de dominio específico se puede realizar la misma clasificación que para los lenguajes de propósito general, aunque en este caso los lenguajes de modelado se conocen como lenguajes de especificación, los cuales resultan muy útiles en el proceso de la ingeniería del software, especialmente en la etapa de especificación de los requisitos. También se utilizan durante las etapas de análisis y diseño del sistema, y permiten describir el sistema a un nivel de abstracción mucho más alto que un lenguaje de programación, además de expresar la especificación del sistema de una forma no ambigua. Un ejemplo de este tipo de lenguajes de especificación es el lenguaje Z.

Ante los lenguajes de dominio específico se presentan una serie de situaciones para las cuales es recomendable su uso. Estas situaciones son las siguientes:

- A la hora de realizar procesamiento con herramientas standalone en la línea de comandos principalmente, como se hace con el comando grep para buscar coincidencias de expresiones regulares, o el lex y el yacc a la hora de definir una gramática.
- Se suelen utilizar también como lenguajes embebidos para proporcionar más funcionalidades a un lenguaje de propósito general. Un ejemplo de esto es LINQ

(Language Integrated Query) que permite la creación de consultas siguiendo la sintaxis del propio lenguaje (utilizado en entornos .NET).

- También se utilizan como lenguajes embebidos en situaciones en las que el código es generado de forma dinámica por la aplicación (o gracias a la interacción con el usuario) como ocurre, por ejemplo, con los macros de las hojas de cálculo en Excel.

3.4 Lenguaje compilado y lenguaje interpretado

En primer lugar, para entender lo que significa un lenguaje interpretado o uno compilado, es recomendable definir lo que es un compilador y lo que es un intérprete en informática.

- **Compilador.** Es un programa informático que se encarga de traducir las instrucciones contenidas en el código fuente que recibe a un lenguaje que sea interpretable por la máquina. El proceso de transformación del código se puede desglosar en una serie de fases:
 - **Fase de análisis.** Esta fase a su vez se subdivide en tres tipos de análisis.
 - **Análisis léxico.** En esta primera fase, se lee el programa y se agrupa el contenido en componentes léxicos llamados tokens, los cuales representan secuencias de caracteres que tienen significado. Es también la fase en la que se elimina toda la información innecesaria para el procesamiento del programa (espacios, saltos de línea, comentarios), además de comprobar que los símbolos del lenguaje (keywords y operadores) se han escrito correctamente.
 - **Análisis sintáctico.** En esta fase los tokens identificados en la anterior se agrupan de forma jerárquica en sentencias gramaticales utilizadas para sintetizar la salida. La gramática del lenguaje, la cual se define antes del análisis, es la que determina la validez de los valores introducidos por el código del programa analizado. Normalmente las sentencias obtenidas se representan en forma de árbol sintáctico abstracto.
 - **Análisis semántico.** En esta fase se realizan 2 tareas fundamentales. Por un lado, se comprueban la existencia de posibles errores que no se hayan podido detectar durante la fase de análisis sintáctico. Por otro lado, se realiza la verificación de tipos que consiste en comprobar la validez de los operandos y operadores existentes con respecto a lo definido en la gramática. Esta información es fundamental para las fases posteriores.
 - **Generación de código.** Tras el análisis previo, se genera un código intermedio interpretable por una máquina abstracta que representa el código fuente introducido al principio del proceso. La representación más común es la llamada código de tres direcciones que funciona de una forma similar al lenguaje ensamblador, de forma que cada posición de memoria funciona como un registro.
 - **Optimización de código:** esta es una fase opcional que consiste en la mejora del código generado para obtener una ejecución más rápida por parte de la máquina abstracta.
- **Intérprete.** Es un programa informático capaz de analizar y ejecutar otros programas. Su principal diferencia respecto a los compiladores es que estos programas solo realizan la traducción del código fuente a medida que es necesario (instrucción por instrucción) y por lo general no guardan el resultado de la traducción. Su principal desventaja con respecto a los compiladores es la eficiencia, normalmente es más rápido el proceso de compilación del código que el de interpretación, no

obstante también puede darse el caso en que es más rápido el proceso de interpretación que el de compilación y ejecución.

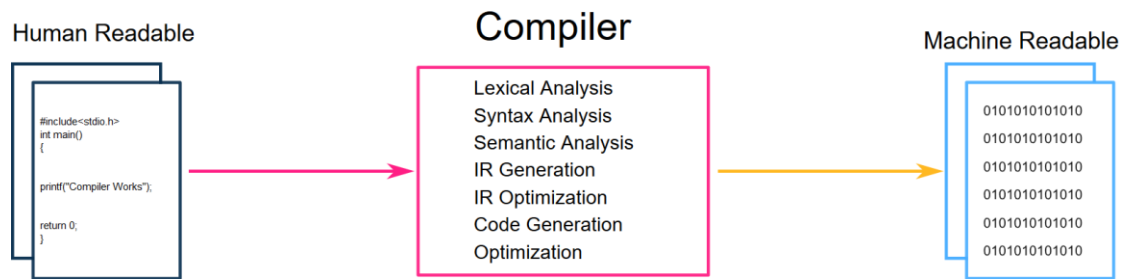


Ilustración 5. Proceso de compilación de un programa

Actualmente, apenas existe distinción entre lenguajes de programación compilados o interpretados ya que las implementaciones más modernas de los lenguajes suelen ofrecer las dos opciones. Algunos ejemplos conocidos de lenguajes de programación compilados son Java o C, mientras que algunos lenguajes de programación interpretados son por ejemplo JavaScript o Ruby.

3.5 Plugin/complemento para un editor

Un plugin o complemento se puede definir como un programa pequeño que se puede añadir a otro mayor para proporcionarle funcionalidades extra. La diferenciación básica entre los términos plugin y complemento se basa en su origen, si el responsable de su creación es una empresa reconocida y el programa cuenta con un certificado de seguridad, se considera un plugin. Esto nos lleva a que los complementos pueden ser creados por un usuario cualquiera. Otras denominaciones que se pueden dar a estos programas es extensiones o add-ons.

A día de hoy, las extensiones están en la gran mayoría de dispositivos electrónicos, desde los add-ons de las Smart TVs hasta los DLCs (downloadable content) de los videojuegos pasando por los complementos de los navegadores como las Chrome Extensions. En particular, los que repercuten en este proyecto son las extensiones para los editores de código e IDEs, en particular los relacionados con el reconocimiento de lenguajes, ya que las extensiones generadas por el programa se utilizarán como complementos para los distintos entornos de desarrollo.

Los plugins otorgan una serie de ventajas que los convierten en elementos realmente útiles para cualquier dispositivo:

- Permiten a desarrolladores externos colaborar con el programa principal extendiendo sus funciones.
- Reducen el tamaño de la aplicación principal que posee las funciones principales, ofreciendo funciones secundarias de forma opcional.
- Permite la separación del código de las extensiones del código del programa principal, haciendo a la aplicación independiente de los fallos de cada extensión.

La aplicación principal o host proporciona servicios, normalmente por medio de un SDK o una API, que el complemento puede utilizar, incluyendo un método para que los complementos se registren a sí mismos y un protocolo para el intercambio de datos. Normalmente

Las interfaces de programación de aplicaciones (API) proporcionan una interfaz estándar, lo que permite a terceros crear complementos que interactúan con la aplicación principal. Una API estable permite que complementos de terceros funcionen como la versión original y amplíen el ciclo de vida de las aplicaciones obsoletas.

3.6 Open Source

El término Open Source se traduce al español por Código Abierto, y comenzó a utilizarse mucho antes de la existencia de los ordenadores y el software. Los primeros datos de los que se tiene constancia en referencia al Open Source data de principios del siglo XX, a raíz de la fabricación de automóviles en Estados Unidos, donde más de 90 patentes de Ford fueron utilizadas por otros fabricantes sin ninguna repercusión legal.

Entre los años 1958 y 1960, la mayor parte de software era desarrollado por académicos e investigadores, los cuales lo compartían íntegramente en una comunidad que se encargaba de buscar errores de programación o añadir nuevas funciones a los programas. La principal novedad de los programas Open Source es que el propio autor revoca todos sus derechos sobre la obra, permitiendo así que cualquiera pueda leer y modificar ese código a su libre albedrío.

Durante los años 70 y principios de los 80, los mayores exponentes de este movimiento fueron las universidades, en particular el MIT y Berkeley con el desarrollo de sus versiones de UNIX, que inicialmente fue desarrollada por AT&T en 1970.

En 1983, Richard Stallman lanzó el proyecto GNU para escribir un sistema operativo completo libre de restricciones para el uso, modificación y distribuirlo con o sin mejoras. Uno de los incidentes particulares que lo motivaron a esto fue el caso de una molesta impresora que no podía ser arreglada porque el código fuente no era revelado. Otro posible evento de inspiración para el proyecto GNU y su manifiesto fue el desacuerdo entre Stallman y Symbolics, Inc. sobre el acceso a las actualizaciones, por parte del MIT, que Symbolics había realizado a su máquina Lisp, la cual estaba basada en código del MIT. Poco tiempo después de su lanzamiento, acuñó el término "software libre" y para promover el concepto fundó la *Free Software Foundation*.

El núcleo Linux iniciado por Linus Torvalds, fue liberado para poder ser modificado libremente en 1991. La licencia inicial, no fue exactamente una licencia de software libre, sin embargo, la versión 0.12 lanzada en febrero de 1992, fue licenciada nuevamente por Torvalds bajo los términos de la Licencia Pública General de GNU. Así como Unix en su tiempo, el núcleo de Torvalds atrajo la atención de programadores voluntarios.

Uno de los principales exponentes del movimiento Open Source actualmente es Microsoft, quién desde el 4 de junio de 2018 es propietario de Github, el mayor repositorio de código abierto del mundo. Con el control de Github, Microsoft tiene el control de la comunidad de desarrollo más grande del mundo. Además, Microsoft es el responsable del LSP, el cual promueve la reutilización de código por su propia definición (un servidor que puede ser utilizado para un sinnúmero de clientes).

El fin del proyecto es convertirlo en Open Source, ya que la plantilla que se proporciona a los usuarios con las extensiones correspondientes para los distintos entornos de desarrollo, no posee todas las funcionalidades que ofrece el LSP, ya que la gran mayoría de ellas son dependientes del lenguaje y requieren ser implementadas a gusto del propio desarrollador.

Capítulo 4. Planificación del Proyecto y Resumen de Presupuestos

4.1 Planificación

La planificación del desarrollo del proyecto se divide en una serie de tareas. Estas tareas representan las principales etapas del ciclo de vida de un proyecto software. Estas tareas son las siguientes:

1. **Análisis.** Esta etapa comprende el estudio de las alternativas, para así poder evaluar la situación actual del sistema y poder determinar las necesidades del mercado con respecto a la idea. Estas alternativas comprenden desde la determinación del ámbito y alcance del proyecto a desarrollar, hasta las tecnologías o arquitecturas que podrán ser aplicadas en el proyecto, pasando por los requisitos del sistema.
2. **Diseño.** Esta fase se beneficiará de los documentos generados en la anterior. A partir de los estudios llevados a cabo y las decisiones tomadas respecto a las alternativas, se comienza a perfilar el sistema y a definir organizaciones importantes como son las estructuras de clases del sistema o la interacción entre ellas.
3. **Formación.** Antes de dar comienzo a la fase de desarrollo es necesario conocer las herramientas que se van a utilizar en el proyecto. En este caso, es necesario aprender a utilizar Node.js y programación de scripts (batch y bash). También es necesario comprender la estructura de la API proporcionada por VSCode para el desarrollo de clientes y servidores que sigan el protocolo LSP, además de unas nociones básicas sobre la instalación de extensiones en este editor.
4. **Desarrollo.** La etapa más larga del ciclo de vida de un proyecto es la de desarrollo / implementación. A lo largo de esta etapa será necesario realizar clases que serán utilizadas como plantillas. Estas clases corresponderán a las tres partes fundamentales del LSP:
 - a. Cliente.
 - b. Servidor.
 - c. Procesador de lenguaje.

También es necesario desarrollar los scripts de generación de extensiones que se encargarán de ejecutar los ficheros desarrollados para la manipulación de las plantillas para adaptarlas a la gramática que se desea reconocer.

5. **Pruebas.** La fase de pruebas se puede llevar a cabo de forma simultánea con la de desarrollo de cada uno de los módulos, en el caso del proyecto no ha sido así. Esto se debe a que los puntos críticos del sistema dependen de las entradas del usuario (gramática introducida y extensión de los ficheros a reconocer), las cuales requieren que el sistema se ejecute al completo para determinar si ha ocurrido algún error o no. Es por esa razón que la fase de pruebas se lleva a cabo de forma posterior.

6. **Documentación.** Una vez finalizado el desarrollo, es necesario comenzar con los manuales del sistema para que los futuros usuarios dispongan de una referencia para utilizar de forma correcta el proyecto.
7. **Alojamiento y distribución del proyecto.** Al tratarse de un proyecto Open Source, lo más importante es que se encuentre disponible para todo tipo de programadores, por eso, una vez terminadas todas las fases anteriores, se procede al alojamiento en un repositorio público de GitHub. La decisión de elegir GitHub es, lo primero, por su tamaño ya que es la herramienta de alojamiento de repositorios de código más utilizada del mundo y, lo segundo, a principios de junio de 2018, pasó a formar parte de Microsoft, la cual es la responsable de la creación del LSP. Este último dato supone un factor de éxito en el proyecto, ya que los proyectos de (y relacionados con) Microsoft en GitHub gozarán de mayor repercusión que otros.

Para ilustrar mejor la distribución del trabajo, se muestra un listado de todas las tareas llevadas a cabo durante el desarrollo del proyecto, y un resumen con las fechas de inicio y fin y horas de trabajo utilizadas para completarlo.

Task Name	Duration	Start	Finish
Análisis	18,5 days	Mon 22/01/18	Tue 27/02/18
Estudio de alternativas	10,5 days	Mon 22/01/18	Sun 11/02/18
Identificación de las alternativas	2 days	Mon 22/01/18	Thu 25/01/18
Clasificación de alternativas	1 day	Fri 26/01/18	Sat 27/01/18
Selección de la mejor alternativa	0,5 days	Sun 28/01/18	Sun 28/01/18
Estudio de tecnologías aplicables al proyecto	2 days	Mon 29/01/18	Thu 01/02/18
Estudio del impacto del proyecto	1 day	Fri 02/02/18	Sat 03/02/18
Determinar arquitectura del proyecto	2 days	Sun 04/02/18	Wed 07/02/18
Determinar lenguajes de programación a utilizar	2 days	Thu 08/02/18	Sun 11/02/18
Determinar alcance del proyecto	1 day	Mon 12/02/18	Tue 13/02/18
Gestión de los requisitos	7 days	Wed 14/02/18	Tue 27/02/18
Identificación de los requisitos	4 days	Wed 14/02/18	Wed 21/02/18
Refinamiento de los requisitos	2 days	Thu 22/02/18	Sun 25/02/18
Elaboración del ERS	1 day	Mon 26/02/18	Tue 27/02/18
Diseño	9 days	Wed 28/02/18	Sat 17/03/18
Determinar las distintas clases	3 days	Wed 28/02/18	Mon 05/03/18
Determinar la interacción entre las distintas clases	2 days	Tue 06/03/18	Fri 09/03/18
Creación de diagramas de flujo	2 days	Sat 10/03/18	Tue 13/03/18
Creación de diagramas de clases	2 days	Wed 14/03/18	Sat 17/03/18
Formación	12 days	Sun 18/03/18	Tue 10/04/18
Estudio de documentación sobre script programming	2 days	Sun 18/03/18	Wed 21/03/18
Estudio de documentación sobre bash	1 day	Sun 18/03/18	Mon 19/03/18
Estudio de documentación sobre batch	1 day	Tue 20/03/18	Wed 21/03/18
Estudio de documentación sobre extensiones de VSCode	2 days	Thu 22/03/18	Sun 25/03/18
Estudio de documentación sobre Node.js	3 days	Mon 26/03/18	Sat 31/03/18
Estudio de documentación sobre Language	3 days	Sun 01/04/18	Fri 06/04/18

Server Protocol			
Estudio de proyectos similares	2 days	Sat 07/04/18	Tue 10/04/18
Desarrollo	20 days	Wed 11/04/18	Sun 20/05/18
Determinación de la estructura de las extensiones generadas	1 day	Wed 11/04/18	Thu 12/04/18
Desarrollo de templates	13 days	Fri 13/04/18	Tue 08/05/18
Desarrollo de los ficheros de dependencias	0,5 days	Fri 13/04/18	Fri 13/04/18
Desarrollo de los ficheros de despliegue	0,5 days	Sat 14/04/18	Sat 14/04/18
Desarrollo del cliente	5 days	Sun 15/04/18	Tue 24/04/18
Desarrollo del servidor	3 days	Wed 25/04/18	Mon 30/04/18
Desarrollo del procesador de lenguaje	4 days	Tue 01/05/18	Tue 08/05/18
Desarrollo de clases de renombrado para las templates	3 days	Wed 09/05/18	Mon 14/05/18
Desarrollo de scripts para la ejecución automática de la generación de extensiones	2 days	Tue 15/05/18	Fri 18/05/18
Documentar métodos de cada clase	1 day	Sat 19/05/18	Sun 20/05/18
Pruebas	11 days	Mon 21/05/18	Mon 11/06/18
Determinar puntos críticos del sistema	2 days	Mon 21/05/18	Thu 24/05/18
Definir pruebas para los puntos críticos	9 days	Fri 25/05/18	Mon 11/06/18
Creación de casos de prueba para cada punto crítico	3 days	Fri 25/05/18	Wed 30/05/18
Testeo de los input del usuario	3 days	Thu 31/05/18	Tue 05/06/18
Testeo de la extensión de los ficheros	1 day	Thu 31/05/18	Fri 01/06/18
Testeo de la gramática introducida	2 days	Sat 02/06/18	Tue 05/06/18
Testeo de los permisos	2 days	Wed 06/06/18	Sat 09/06/18
Testeo de falta de dependencias	1 day	Sun 10/06/18	Mon 11/06/18
Documentación	2 days	Tue 12/06/18	Fri 15/06/18
Crear manuales de usuario para el proyecto	2 days	Tue 12/06/18	Fri 15/06/18
Alojamiento y distribución del proyecto	3 days	Sat 16/06/18	Thu 21/06/18
Subir el proyecto a un repositorio público de Github	1 day	Sat 16/06/18	Sun 17/06/18
Dar difusión al proyecto en foros de programación o discusión de temas relacionados	2 days	Mon 18/06/18	Thu 21/06/18

Tabla 1. Tareas de la planificación del proyecto

Para la realización de las tareas se utiliza un único recurso de tipo trabajo (el autor del actual documento), el cual trabaja como Freelance estableciendo sus propias tarifas como se explica en el siguiente apartado. El horario de trabajo planificado en el calendario para el desempeño de las tareas es de 8 a 12 todos los días.

4.2 Resumen del Presupuesto

A partir de las tareas obtenidas en la fase anterior, calcularemos el presupuesto, estimando el coste que tendrían cada una de las tareas principales. Estas tareas serán *análisis, diseño, formación, desarrollo, pruebas, documentación y alojamiento y distribución del proyecto*.

Respecto al coste de las fases, la tarea de desarrollo al ser la más larga también será la que mayor coste ocasione. La siguiente que ocasionará un mayor coste será también la de análisis, ya que, al tratarse de una tecnología desconocida hasta el momento para el trabajador, conllevará un estudio amplio y conciso sobre las mejores opciones para llevar a cabo la solución.

Task Name	Fixed Cost	Fixed Cost	Total Cost	Actual	Remaining
Análisis	0,00 €	Prorated	2.368,00 €	0,00 €	2.368,00 €
Diseño	0,00 €	Prorated	1.152,00 €	0,00 €	1.152,00 €
Formación	0,00 €	Prorated	1.536,00 €	0,00 €	1.536,00 €
Desarrollo	0,00 €	Prorated	2.560,00 €	0,00 €	2.560,00 €
Pruebas	0,00 €	Prorated	1.408,00 €	0,00 €	1.408,00 €
Documentación	0,00 €	Prorated	256,00 €	0,00 €	256,00 €
Alojamiento y distribución del proyecto	0,00 €	Prorated	384,00 €	0,00 €	384,00 €

Tabla 2. Resumen del coste del proyecto por fases

Los costes anteriores se calculan a partir de los costes establecidos inicialmente para los distintos recursos. En este proyecto, solo se identifica un recurso que será el desarrollador que llevará a cabo el proyecto. Este desarrollador, no se encontrará en plantilla de la organización que haya encargado el proyecto, actuará como freelance. Por tanto, el coste por hora se obtiene de una tarifa preestablecida por el trabajador.

Cabe destacar, que estas estimaciones de presupuesto son tan solo orientativas a la posibilidad de que este proyecto hubiese sido encargado por una empresa. Como se menciona a lo largo del documento, este proyecto se enmarca en la filosofía Open Source y corre a cargo de los intereses e inquietudes del autor. En ningún momento se baraja la idea de obtener un beneficio económico a raíz del proyecto.

Resource Name	Type	Max. Units	Std. Rate	Ovt. Rate	Accrue At	Base Calendar
Sergio	Work	100%	16,00 €/hr	1,00 €/hr	Prorated	Standard

Tabla 3. Resumen de los recursos del proyecto

Finalmente se obtiene un coste total de 9664.00€ para todo el proyecto. En caso de que se requiriese al trabajador adelantar los plazos de entrega, este se vería obligado a aumentar las horas de trabajo diarias, aplicando un cargo por cada hora extra que llevase a cabo.

Capítulo 5. Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

5.1 Definición del Sistema

Al tratarse de un proyecto Open Source, no se considerará una aplicación finalizada en su entrega. Dependerá de los usuarios de la comunidad de Github continuar con su desarrollo, otorgando funcionalidades nuevas a los distintos servidores de lenguaje que se generen en el proyecto, además de permitir la creación de nuevos clientes (además del que se proporciona ya en el proyecto) como pueden ser Atom, IntelliJ o Eclipse.

Para definir mejor el alcance, las funcionalidades que se proporcionan son:

- El resaltado de la sintaxis en el lugar donde se encuentren errores.
- Una breve aproximación (sin llegar a estar implementada) a la utilidad ‘hover’, la cual permite mostrar una explicación del término sobre el que se sitúa el ratón.
- Una breve aproximación (sin llegar a estar implementada) del autocompletado, resulta sencillo programar de forma automática el autocompletado para las keywords del lenguaje o para los tokens, sin embargo, resulta un poco más complejo para variables definidas en el contexto del programa, por ejemplo.

Queda a juicio de los usuarios la posibilidad de incluir todas las funcionalidades del LSP a los servidores de lenguaje que generen. Además de las siguientes posibilidades de cara a la aplicación principal:

- Reconocimiento alternativo de una gramática. Esto se puede dividir en dos posibilidades.
 - Permitir el reconocimiento en ficheros separados como son el Scanner y el Parser en ANTLR.
 - Permitir el reconocimiento para otros generadores de lenguajes como pueden ser Yacc o XText.
- Añadido de coloreado de la sintaxis. Esto es una característica que actualmente se encuentra en desarrollo por parte de Microsoft, ya que no es algo que consideren propio del servidor de lenguaje, si no de cada uno de los clientes por separado, aunque solo temporalmente.
- Creación de soporte para distintos clientes. Actualmente cada vez más clientes comienzan a dar soporte mediante SDKs a el desarrollo de aplicaciones que utilizan LSP, algunos de estos clientes son por ejemplo IntelliJ, Eclipse, Atom, Vim, Sublime, ... Resultaría muy útil disponer de la posibilidad de generar extensiones también para estos editores, los cuales son muy utilizados en la industria.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

El producto de esta sección se crea para su aprobación formal, es decir, que los potenciales clientes deben ver a partir de él las especificaciones completas del sistema. Además, esta sección construirá una base para solicitar cambios en los requisitos antes de avanzar más en la construcción del sistema.

Los requisitos del sistema se deben mostrar en una tabla como la que se presentará a continuación con ejemplos, ordenados por algún criterio lógico en función de a lo que se refieran. Por ejemplo, si tenemos usuarios tiene sentido agrupar todos los requisitos que tengan que ver con los usuarios.

Tampoco es necesario crear una única tabla para todos los requisitos, pueden crearse varias tablas que agrupen los requisitos que se refieran a una entidad.

Este apartado debe incluir también antes de la tabla de requisitos, si existe como tal, la **especificación textual** que el cliente nos proporcione sobre la aplicación, fruto de las reuniones que hayamos tenido con él o de las entrevistas que podamos haber llevado a cabo. Ha de tenerse en cuenta que esta información es la que usaremos para extraer los requisitos de la aplicación, por lo que no debe faltar sin contamos con ella.

5.2.1.1 Requisitos funcionales

1. El sistema permitirá la generación de extensiones para editores de código y entornos de desarrollo.
 - 1.1. El sistema permitirá crear extensiones para los editores que implementen el LSP.
 - 1.1.1. Inicialmente el sistema permitirá crear extensiones para el entorno Visual Studio Code.
 - 1.2. El sistema generará las extensiones a partir de las entradas del usuario.
 - 1.2.1. El usuario introducirá una gramática desarrollada con ANTLR4.
 - 1.2.1.1. El sistema debe comprobar que la gramática no contiene errores.
 - 1.2.1.1.1. Si el sistema detectase algún error, debe notificar al usuario de forma inmediata e interrumpir la ejecución del programa.
 - 1.2.1.2. La extensión de la gramática introducida debe ser '.g4'.
 - 1.2.1.2.1. Si la extensión del fichero no es correcta, el sistema debe notificar al usuario.
 - 1.2.2. El usuario introducirá la extensión de los ficheros del lenguaje que desea reconocer.
 - 1.2.2.1. La extensión tendrá el formato '.NOMBRE_EXTENSION'.
 - 1.2.2.1.1. El sistema notificará al usuario si la extensión introducida no cumple el formato.
 - 1.3. El sistema generará NUM_EXTENSIONES extensiones tras su ejecución.
 - 1.3.1. NUM_EXTENSIONES será un valor configurable.
 - 1.3.2. El valor inicial de NUM_EXTENSIONES será 2.

- 1.3.3. El sistema generará una extensión que permita el reconocimiento del lenguaje definido en la gramática.
 - 1.3.3.1. La extensión debe ejecutarse cuando el usuario abra un fichero con la extensión definida previamente.
 - 1.3.3.2. La extensión proporcionará características propias del LSP cuando el fichero sea editado.
 - 1.3.3.2.1. La extensión permitirá al editor enviar diagnósticos sobre el fichero editado a partir de la gramática.
 - 1.3.3.2.2. La extensión permitirá buscar referencias respecto de los distintos términos que aparezcan en el fichero.
 - 1.3.3.2.3. El sistema incluirá fragmentos de código para permitir al usuario implementar nuevas funciones del LSP.
 - 1.3.3.2.3.1. Autocompletado de términos.
 - 1.3.3.2.3.2. Dirigir a la definición de un elemento.
 - 1.3.3.2.3.3. Mostrar información flotante de un elemento.
- 1.3.4. El sistema generará una extensión que permita el coloreado de la sintaxis de la gramática.

5.2.1.2 *Requisitos no funcionales*

5.2.1.2.1 **Requisitos de usuario**

- 1. El usuario necesitará poseer conocimiento sobre el sistema si desea editar su código fuente.
 - 1.1. El usuario necesitará poseer conocimiento en Node.js.
 - 1.2. El usuario necesitará poseer conocimiento en el desarrollo de extensiones para Visual Studio Code.
 - 1.3. El usuario necesitará poseer conocimiento en TypeScript.
- 2. El usuario necesitará poseer conocimiento sobre la ejecución de scripts.
 - 2.1. Si el sistema se ejecuta en Windows, necesitará conocimiento sobre ficheros Batch.
 - 2.2. Si el sistema se ejecuta en Linux, necesitará conocimiento sobre ficheros Bash.
- 3. El usuario necesitará disponer de una gramática desarrollada en ANTLR4 para ejecutar la aplicación.

5.2.1.2.2 **Requisitos tecnológicos**

- 1. El sistema necesita que el entorno de utilización disponga de Node.js instalado.
 - 1.1. La versión mínima de Node.js requerida por el programa es 8.6.0.
- 2. El sistema necesita que el entorno de utilización disponga de TypeScript instalado.
 - 2.1. La versión mínima de TypeScript requerida por el programa es 2.9.2.
- 3. El sistema requiere de conexión a internet para instalar las dependencias necesarias.

5.2.1.2.3 **Requisitos de seguridad**

- 1. El sistema transmitirá la información entre el cliente y el servidor por medio del protocolo JSON-RPC.

2. El sistema transmitirá la información entre el servidor y el procesador de lenguaje por medio del protocolo HTTP de forma local.

5.2.2 Identificación de Actores del Sistema

Para el presenta proyecto se detecta un actor, que es el propio usuario del sistema. Esto ocurre debido a que no existe ningún tipo de autenticación de los usuarios para utilizar la aplicación. Por tanto, será el usuario el que realizará las acciones con respecto al sistema.

Respecto al propio sistema, podemos dividirlo en dos partes principales que serán las que llevarán a cabo la comunicación con el usuario:

- El sistema de generación de extensiones. Este sistema requerirá al usuario la introducción de una gramática que defina un lenguaje y una extensión que permita el reconocimiento de los ficheros que se analizarán respecto de dicha gramática.
- El subsistema cliente de la extensión generada. Esta parte de la extensión generada se comunicará con las otras dos partes de la extensión para llevar a cabo las acciones requeridas por el usuario.

5.2.3 Especificación de Casos de Uso

Como se ha explicado en la sección anterior, representamos al usuario respecto a dos subsistemas, el generador de extensiones y el cliente de la extensión generada. Los diagramas de casos de uso correspondientes son los siguientes:

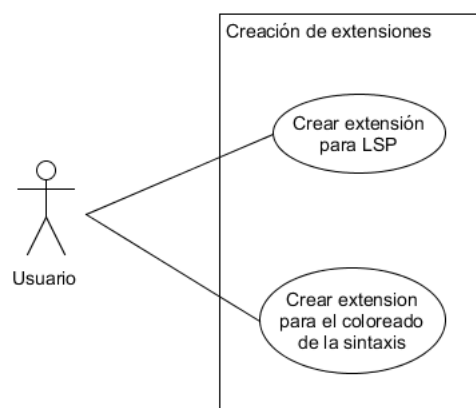


Ilustración 6. Diagrama de casos de uso para generación de extensiones

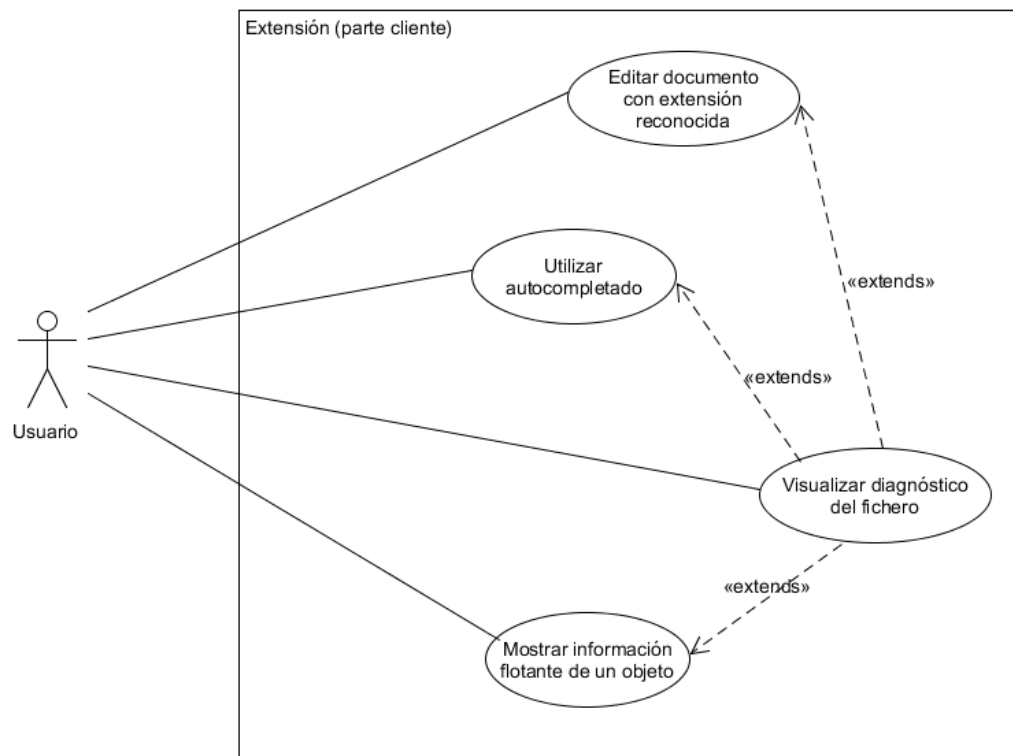


Ilustración 7. Diagrama de casos de uso para la interacción con el editor

A continuación, se incluye una descripción de cada uno de los casos de uso identificados para la comunicación con el cliente de la extensión:

Nombre del Caso de Uso
Editar documento con extensión reconocida
Descripción
La extensión detecta que un fichero con la extensión determinada por el usuario está siendo editado, por lo que envía una notificación de aviso del cliente al servidor junto con el texto editado. Posteriormente el servidor lo enviará al procesador de texto para llevar a cabo una validación del texto recibido y generar diagnósticos.

Nombre del Caso de Uso
Mostrar información flotante de un objeto
Descripción
Una de las funcionalidades que proporciona el protocolo servidor de lenguaje es la de poder situar el cursor encima de un elemento concreto del código que se está editando para obtener información extra sobre ese elemento. Esta información suele definirse de acuerdo a datos incluidos durante el análisis semántico del lenguaje a las distintas construcciones semánticas para ofrecer información más concreta e interesante sobre el objeto, aunque puede realizarse a partir de cualquier tipo de información. Esta información se muestra a partir de los diagnósticos generados por las clases del procesador de lenguaje, y que son enviadas por parte del servidor al cliente.

Nombre del Caso de Uso
Utilizar autocompletado
Descripción
<p>Una de las funcionalidades que proporciona el protocolo servidor de lenguaje es precisamente el autocompletado de términos en concreto del código que se está editando. Esto permite reducir en gran cantidad el tiempo de programación. Esta información suele definirse de acuerdo a datos incluidos durante los análisis léxico y sintáctico del lenguaje (a no ser que se desee permitir el autocompletado de fragmentos de código, lo cual podría realizarse también a lo largo de la fase de análisis semántico).</p> <p>Esta información se muestra a partir de los diagnósticos generados por las clases del procesador de lenguaje, y que son enviadas por parte del servidor al cliente.</p>

Nombre del Caso de Uso
Visualizar diagnóstico del fichero
Descripción
<p>Partiendo de los diagnósticos generados en los otros casos de uso, el servidor se comunica de nuevo con el procesador de lenguaje para recibir estos diagnósticos. Posteriormente, son enviados al cliente el cual procesa el contenido de los diagnósticos y representa la información recibida en el editor. La información se muestra de dos formas distintas, sobre los términos que tengan relación con el diagnóstico (errores, información flotante, autocompletado, ...) y en el panel inferior con un mensaje descriptivo de la información proporcionada.</p>

A continuación, se incluyen cada uno de los casos de uso identificados para la generación de extensiones:

Nombre del Caso de Uso
Crear extensión para LSP
Descripción
<p>El sistema pide al usuario que introduzca una gramática escrita en ANTLR4, además de un nombre que se corresponda con la extensión de los ficheros que desea que el editor reconozca. A partir de las plantillas disponibles para generar las distintas partes de la extensión, se hace una copia de estas y se renombran los aspectos dependientes de la conexión con los distintos subsistemas para el nuevo lenguaje.</p> <p>La extensión generada permite utilizar características propias del LSP con ficheros con extensión reconocida, tales como autocompletado, información flotante, buscar referencias, ...</p>

Nombre del Caso de Uso
Crear extensión para el coloreado de la sintaxis
Descripción
<p>El sistema pide al usuario que introduzca una gramática escrita en ANTLR4, además de un nombre que se corresponda con la extensión de los ficheros que desea que el editor reconozca. A partir de las plantillas disponibles para generar la extensión de coloreado de sintaxis, se hace una copia de estas y se renombran los aspectos dependientes de la conexión con los distintos subsistemas de la extensión para el nuevo lenguaje.</p> <p>Esta extensión permite visualizar en distinto color ciertos elementos del lenguaje, para mejorar así la legibilidad del fichero.</p>

5.3 Identificación de los Subsistemas en la Fase de Análisis

En el presente proyecto detectamos 3 sistemas principales, de los cuales, 2 serán generados por el otro sistema. De esta forma, identificamos un sistema generador de extensiones, una extensión que permite el reconocimiento de lenguajes y proporciona al editor características propias del LSP, y una segunda extensión que permite el reconocimiento de lenguajes y muestra ciertos elementos léxicos de dicho lenguaje en un color diferente, para facilitar la legibilidad del texto.

5.3.1 Descripción de los Subsistemas

Como se ha mencionado previamente, encontramos tres subsistemas que actúan de forma independiente entre sí. Estos subsistemas se explican con más detalle a continuación:

5.3.1.1 *Sistema generador de extensiones*

Será la aplicación con la que el usuario tendrá que interactuar para llevar a cabo la generación de las dos extensiones. Este programa pedirá al usuario como entradas una gramática definida en ANTLR4 y un nombre para la extensión de los ficheros que desea que sean reconocidos por las extensiones generadas.

Este sistema utilizará plantillas predefinidas para la generación de las extensiones. El sistema realizará una copia de estas plantillas y editará los datos dependientes del lenguaje introducido para poder llevar a cabo el procesamiento del lenguaje de forma correcta.

5.3.1.2 *Extensión para el coloreado de la sintaxis*

Esta extensión es generada por el subsistema anterior a partir de la gramática introducida por el usuario. Funciona principalmente a partir de eventos que son activados en distintas situaciones:

- La extensión se activa cuando el usuario abre un fichero con la extensión definida en el fichero de configuración de la propia extensión.
- A partir de su activación, la extensión realizará una revisión del texto con cada pulsación del usuario en el teclado.
- En caso de que alguno de los términos escritos por el usuario en el fichero se corresponda con los definidos en un fichero de configuración de la propia aplicación, este término cambiará de color dotando al texto de una mayor legibilidad.

5.3.1.3 *Extensión LSP*

Al igual que el proyecto completo, esta extensión se puede subdividir en tres partes principales que interactúan entre ellas:

- **El cliente.** Es la única parte dependiente del editor, ya que será el intermediario entre el propio editor y la lógica del lenguaje accedida por el servidor. El lenguaje en el que se programe esta parte dependerá de las herramientas ofrecidas por el editor para dicho fin.
- **El servidor.** Consiste en una aplicación que sirve de intermediario entre las peticiones enviadas por parte del cliente (cuando el usuario abre, cierra o modifica un fichero) y la lógica de lenguaje (generada a partir de la gramática proporcionada por el usuario) que procesa la información enviada por el cliente. Por tanto, recibe los datos por parte del cliente y los envía a la lógica del lenguaje que se encarga de procesarlos y de enviar los resultados de vuelta al servidor. Estos resultados son interpretados en el servidor que los envía como respuesta al cliente para que el editor lleve a cabo las actividades necesarias (resaltar las partes del código erróneas, proporcionar información sobre la palabra seleccionada por el cursor, autocompletar, ...).
- **Procesador del lenguaje.** Se trata de clases generadas de forma automática, a partir de la gramática proporcionada por el usuario, gracias a las herramientas que proporciona ANTLR de forma gratuita. Las herramientas de ANTLR permiten elegir el lenguaje de generación de las clases y, para obtener una mejor comunicación con el servidor, ya que se encuentra desarrollado en JavaScript, la generación del código se realiza también en JavaScript. Estas clases reciben cierta información por parte del servidor, la procesan y devuelven los resultados de la validación al servidor. Este procesador se considera básicamente un servidor que se encuentra desplegado de forma local en un puerto concreto.

5.3.2 Descripción de los Interfaces entre Subsistemas

Como se menciona en el apartado 5.3.1, los sistemas funcionan de forma independiente sin interactuar unos con otros. Es en el caso de la extensión LSP la única situación en la que los módulos interactúan entre sí. En este caso, se distinguen dos protocolos utilizados para la comunicación entre los subsistemas:

- **Protocolo JSON-RPC:** Es el protocolo utilizado para la transmisión de información entre el cliente y el servidor. Es un protocolo sin estado, ligero y que permite la ejecución remota de procedimientos. La posibilidad de permitir la ejecución de procesos de forma remota resulta muy útil para el LSP, ya que el cliente es el que propicia la ejecución de la lógica del servidor. Utiliza JSON como formato de los datos transmitidos, y en un principio fue concebido como una alternativa a SOAP (ya que SOAP usa XML en lugar de JSON).
- **Protocolo HTTP:** Es el protocolo utilizado para la transmisión de información entre el servidor y el procesador de lenguaje. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de la especificación de la versión 1.1. HTTP, la cual define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Respecto a este proyecto, al encontrarse el procesador de lenguaje desplegado de forma local como un servidor web en un puerto concreto, la comunicación con él se produce a partir de peticiones HTTP a las cuales el procesador responde con diagnósticos sobre el texto recibido.

5.4 Diagrama de Clases Preliminar del Análisis

5.4.1 Diagrama de Clases

Previo a realizar el diseño definitivo del diagrama de clases, será necesario realizar primero una representación preliminar de las clases correspondientes a cada subsistema. La representación de los distintos subsistemas se muestra en la siguiente imagen:

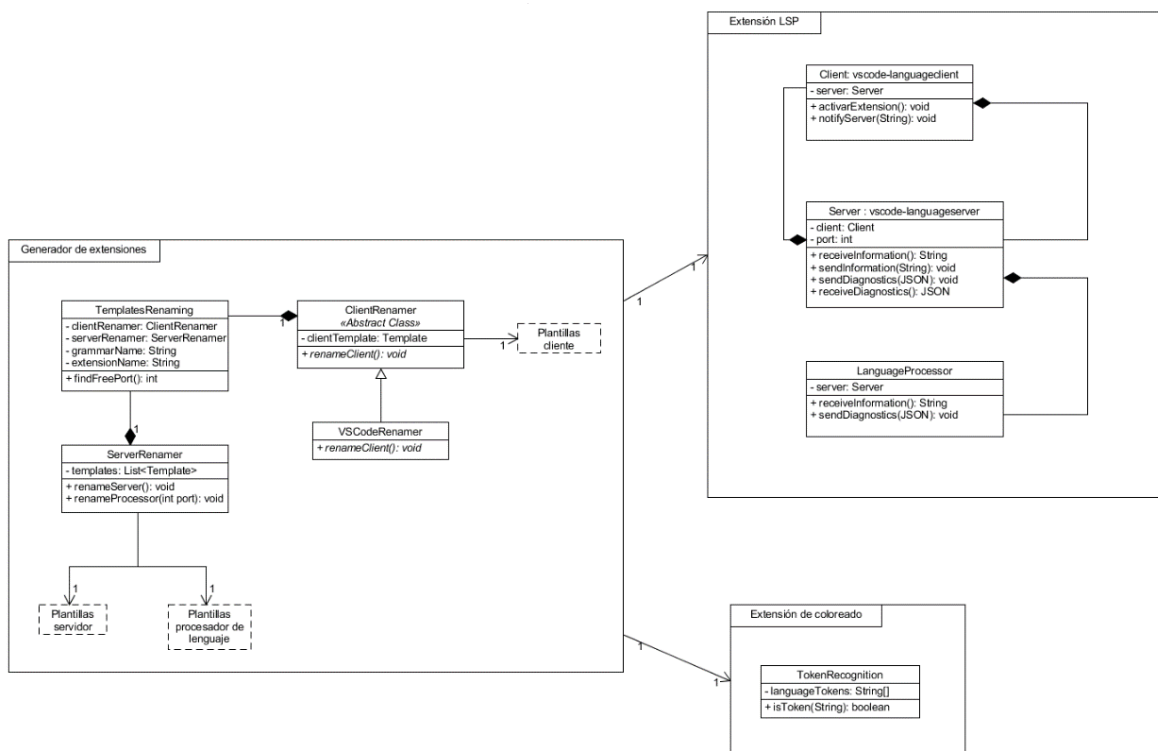


Ilustración 8. Relación entre los distintos subsistemas

5.4.2 Descripción de las Clases

5.4.2.1 Generador de extensiones

Nombre de la Clase
TemplatesRenaming
Descripción
Clase principal sobre la que recae la responsabilidad de llevar a cabo el renombrado de las copias realizadas de las plantillas para cada clase de las extensiones.
Responsabilidades
Ejecutar la lógica de renombrado de las distintas plantillas.
Atributos Propuestos
clientRenamer: objeto cuya clase estática es ClientRenamer aunque de clase dinámica VSCodeRenamer. Se encargará del renombrado de las plantillas del cliente. serverRenamer: objeto cuya clase es ServerRenamer. Se encargará del renombrado de las plantillas del servidor. grammarName: string que contiene el nombre de la gramática introducida por el usuario. extensionName: string que contiene el nombre de la extensión introducida por el usuario.
Métodos Propuestos
findFreePort: este método, como su propio nombre indica, se encargará de buscar un puerto libre en el sistema en el cual se desplegará el servidor que realizará el procesamiento del lenguaje.

Nombre de la Clase
ServerRenamer
Descripción
Clase encargada de llevar a cabo el renombrado de las copias realizadas de las plantillas referentes al servidor. Para esta clase no se contempla ningún tipo de herencia ya que la parte del servidor es compatible con cualquier entorno.
Responsabilidades
Ejecutar la lógica de renombrado de las plantillas del servidor.
Atributos Propuestos
templates: consiste en una lista de plantillas (en concreto contendrá 2) relacionadas con el servidor y el procesador de lenguaje.
Métodos Propuestos
renameServer: método que sobrecargará la implementación del método abstracto de la clase padre. Será el encargado del renombrado de las plantillas del servidor. renameProcessor: método que sobrecargará la implementación del método abstracto de la clase padre. Será el encargado del renombrado de las plantillas del procesador de lenguaje. Este método requiere un parámetro que representa al puerto libre detectado por el método findFreePort de la clase principal del renombrado.

Nombre de la Clase
ClientRenamer
Descripción
Clase padre encargada de llevar a cabo el renombrado de las copias realizadas de las plantillas referentes al servidor. Para esta clase no se contempla ningún tipo de herencia ya que la parte del servidor es compatible con cualquier entorno.
Responsabilidades
Ejecutar la lógica de renombrado de las distintas plantillas.
Atributos Propuestos
Métodos Propuestos
renameClient: este método abstracto, será implementado por la clase hija elegida en su instanciación. Se encargará del renombrado de la plantilla correspondiente al cliente.

Nombre de la Clase
VSCodeRenamer
Descripción
Clase hija que hereda de la clase abstracta ClientRenamer, se instanciará cuando la extensión que se desee crear sea para el editor Visual Studio Code.
Responsabilidades
Ejecutar la lógica de renombrado de las plantillas del cliente para Visual Studio Code.
Atributos Propuestos
clientTemplate: este objeto contiene la plantilla relacionada con el cliente correspondiente.
Métodos Propuestos
renameClient: método que sobrecarga la implementación del definido en la clase padre. Será el método encargado del renombrado de las plantillas del cliente para Visual Studio Code.

5.4.2.2 Extensión LSP

Nombre de la Clase
Client
Descripción
Clase que permite la interacción con el usuario y que se encarga de notificar al servidor de dicha interacción. Enviará el contenido del documento editado al servidor y mostrará al usuario un diagnóstico del fichero en tiempo real.
Responsabilidades
Establecer la comunicación entre el usuario y el servidor a través del editor correspondiente.
Atributos Propuestos
server: objeto que representa al servidor con el cual el cliente se comunicará para la transmisión de información.
Métodos Propuestos
activarExtension: se encargará de ejecutar la lógica de la clase cuando detecte que se abre un fichero con extensión reconocida.
notificarServidor: este método se utilizará para enviar una notificación a la clase servidor cuando el usuario abra, modifique o cierre un fichero con extensión reconocida. Enviará el texto del fichero en el momento correspondiente.

Nombre de la Clase
Server
Descripción
Clase que contiene la lógica para identificar los distintos tokens y editar el color de los que se reconozcan en el fichero editado en el cliente.
Responsabilidades
Clase que actúa como intermediario entre el cliente y el procesador de lenguaje para transmitir la información correspondiente entre ellos.
Atributos Propuestos
client: objeto instanciado de la clase cliente que permite guardar una referencia para mantener la comunicación entre las clases.
port: consiste en un valor entero que permite identificar el puerto en el cual se encuentra desplegado el servidor que contiene el procesador de lenguaje.
Métodos Propuestos
receiveInformation: este método se utilizará para recibir el contenido del documento desde el cliente.
sendInformation: este método se utilizará para enviar el contenido del documento al procesador de lenguaje. El parámetro consiste en la información contenida en el fichero en formato string.
receiveDiagnostics: este método se utilizará para recibir los diagnósticos del documento desde el procesador del lenguaje.
sendDiagnostics: este método se utilizará para enviar el contenido de los diagnósticos al cliente. El parámetro consiste en el diagnostico generado por el procesador del lenguaje en formato JSON.

Nombre de la Clase
LanguageProcessor
Descripción
Clase que se encarga de evaluar el contenido del documento recibido de acuerdo a la gramática proporcionada por el usuario y generar unos diagnósticos al respecto.
Responsabilidades
Generar diagnósticos sobre el texto contenido en el fichero.
Atributos Propuestos
server: objeto instanciado de la clase servidor que permite guardar una referencia para mantener la comunicación entre las clases.
Métodos Propuestos
receiveInformation: este método se utilizará para recibir el contenido del documento desde el servidor.
sendDiagnostics: este método se utilizará para enviar el contenido de los diagnósticos al servidor. El parámetro consiste en el diagnostico generado por el procesador del lenguaje en formato JSON.

5.4.2.3 Extensión de coloreado

Nombre de la Clase
TokenRecognition
Descripción
Clase que contiene la lógica para identificar los distintos tokens y editar el color de los que se reconozcan en el fichero editado en el cliente.
Responsabilidades
Determinar el color de los distintos términos en los ficheros editados.
Atributos Propuestos
languageTokens: consiste en un vector de strings que contendrá todos los tokens identificados en la gramática que permitirán el coloreado de términos concretos en los ficheros reconocidos.
Métodos Propuestos
isToken: es un método que recibe como parámetro el fragmento de texto que está siendo editado y lo compara con los tokens pertenecientes al vector para determinar si se debe aplicar color sobre ese texto.

5.5 Análisis de Casos de Uso y Escenarios

5.5.1 Generación de extensiones

En esta sección se describirán los casos de uso identificados en el apartado Especificación de Casos de Uso referentes a la generación de extensiones, tanto la extensión LSP como la de coloreado. Los casos de uso identificados son dos:

5.5.1.1 Crear extensión para LSP

Crear Extensión LSP	
Precondiciones	El sistema en el que se ejecuta el programa debe tener Node.js instalado.
Postcondiciones	Debe existir un nuevo directorio con todas las clases de la extensión de reconocimiento de lenguaje para LSP en la carpeta donde se ejecuta el programa.
Actores	Iniciado y terminado por el usuario.
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Introduce la gramática y el sistema la valida. 2. Introduce la extensión requerida por el sistema. <p>El sistema:</p> <ol style="list-style-type: none"> 3. Busca un puerto libre en el que ejecutar el procesador de lenguaje. 4. Lee las plantillas existentes y genera las clases correspondientes al cliente renombrando los elementos necesarios. 5. Lee las plantillas existentes y genera las clases correspondientes al servidor renombrando los elementos necesarios. 6. Lee las plantillas existentes y genera las clases correspondientes al procesador de lenguaje renombrando los elementos necesarios. 7. Organiza la estructura de ficheros para la extensión. 8. Informa del fin de la ejecución.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El usuario introduce una gramática inválida. <ul style="list-style-type: none"> ○ Volver al paso 1 del escenario principal, el usuario debe introducir de nuevo una gramática válida. • Escenario Alternativo 2: El usuario introduce una extensión inválida. <ul style="list-style-type: none"> ○ Volver al paso 2 del escenario principal, el usuario debe introducir de nuevo una extensión válida.
Excepciones	<ul style="list-style-type: none"> • Las clases-plantilla se mueven del directorio original. No se pueden generar extensiones ante la falta de las clases. <ul style="list-style-type: none"> ○ Notificar de un error asociado al problema encontrado.
Notas	-

5.5.1.2 Crear extensión para el coloreado de la sintaxis

Crear extensión para el coloreado de sintaxis	
Precondiciones	El sistema en el que se ejecuta el programa debe tener Node.js instalado.
Postcondiciones	Debe existir un nuevo directorio con todas las clases de la extensión de coloreado de sintaxis en la carpeta donde se ejecuta el programa.
Actores	Iniciado y terminado por el usuario.
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Introduce la gramática y el sistema la valida. 2. Introduce la extensión requerida por el sistema. <p>El sistema:</p> <ol style="list-style-type: none"> 3. Lee las plantillas existentes y genera las clases correspondientes a la extensión de coloreado renombrando los elementos necesarios. 4. Organiza la estructura de ficheros para la extensión. 5. Informa del fin de la ejecución.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El usuario introduce una gramática inválida. <ul style="list-style-type: none"> ○ Volver al paso 1 del escenario principal, el usuario debe introducir de nuevo una gramática válida. • Escenario Alternativo 2: El usuario introduce una extensión inválida. <ul style="list-style-type: none"> ○ Volver al paso 2 del escenario principal, el usuario debe introducir de nuevo una extensión válida.
Excepciones	<ul style="list-style-type: none"> • Las clases-plantilla se mueven del directorio original. No se pueden generar extensiones ante la falta de las clases. <ul style="list-style-type: none"> ○ Notificar de un error asociado al problema encontrado.
Notas	-

5.5.2 Interacción con el editor

En esta sección se describirán los casos de uso identificados en el apartado Especificación de Casos de Uso referentes a la interacción que realiza la extensión LSP con el editor de código correspondiente. Los casos de uso identificados son dos:

5.5.2.1 Editar documento con extensión reconocida

Editar documento con extensión reconocida	
Precondiciones	La extensión del fichero abierto en el editor debe corresponderse con la elegida por el usuario para la generación de la extensión.
Postcondiciones	Deben mostrársele al usuario los errores léxicos y/o sintácticos que se hayan reconocido en el fichero.
Actores	Iniciado y terminado por el usuario.
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Abre un fichero cuya extensión sea reconocida por el plugin LSP instalado. 2. Modifica el fichero para activar el plugin LSP. <p>El sistema:</p> <ol style="list-style-type: none"> 3. Envía el contenido del fichero al procesador del lenguaje. 4. Analiza el contenido recibido y devuelve al servidor diagnósticos con los errores detectados. 5. El servidor envía los errores al cliente, el cual los representa en la consola y en el propio fichero, resaltando las zonas del código que contienen errores.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El usuario modifica el fichero, pero no hay ningún error léxico ni sintáctico. <ul style="list-style-type: none"> ○ No se muestra ningún error en el fichero al usuario.
Excepciones	<ul style="list-style-type: none"> • Existe un proceso ejecutándose en el puerto por defecto del procesador de lenguaje. <ul style="list-style-type: none"> ○ Se informa al usuario sobre el error.
Notas	-

5.5.2.2 Utilizar autocompletado

Utilizar autocompletado	
Precondiciones	La extensión del fichero abierto en el editor debe corresponderse con la elegida por el usuario para la generación de la extensión.
Postcondiciones	Deben mostrársele al usuario los elementos que se sugieren a partir del término que desea autocompletar.
Actores	Iniciado y terminado por el usuario.
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Inicia la escritura de un elemento. 2. Ejecute el atajo de teclado para autocompletar el elemento. <p>El sistema:</p>

	<ol style="list-style-type: none"> Analiza la petición del usuario respecto al término introducido. Sugiere posibles coincidencias al usuario.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> Escenario Alternativo 1: No existe ningún término sugerido con respecto al introducido por el usuario. <ul style="list-style-type: none"> Se informa al usuario que no existen sugerencias.
Excepciones	<ul style="list-style-type: none"> Existe un proceso ejecutándose en el puerto por defecto del procesador de lenguaje. <ul style="list-style-type: none"> Se informa al usuario sobre el error.
Notas	-

5.5.2.3 *Mostrar información flotante de un objeto*

Mostrar información flotante de un objeto	
Precondiciones	La extensión del fichero abierto en el editor debe corresponderse con la elegida por el usuario para la generación de la extensión.
Postcondiciones	Deben mostrársele al usuario la información correspondiente al elemento sobre el que se coloca el cursor.
Actores	Iniciado y terminado por el usuario.
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> Sitúa el cursor sobre el elemento del que desea recibir información <p>El sistema:</p> <ol style="list-style-type: none"> Analiza la petición del usuario respecto al término introducido. Envía la información correspondiente a dicho término al cliente. Se muestra al usuario la información requerida.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> Escenario Alternativo 1: No existe ningún término sugerido con respecto al introducido por el usuario. <ul style="list-style-type: none"> Se informa al usuario que no existe información al respecto o se muestra la información por defecto, en caso de que se haya definido información por defecto.
Excepciones	<ul style="list-style-type: none"> Existe un proceso ejecutándose en el puerto por defecto del procesador de lenguaje. <ul style="list-style-type: none"> Se informa al usuario sobre el error.
Notas	-

5.5.2.4 *Visualizar diagnóstico del fichero*

Visualizar diagnóstico del fichero	
Precondiciones	Nos encontramos ante alguno de los casos de uso anteriores.
Postcondiciones	Se muestran al usuario los diagnósticos obtenidos por el procesador de lenguaje.
Actores	Iniciado y terminado por el usuario.
Descripción	El usuario:

	<ol style="list-style-type: none"> 1. Abre un fichero con una extensión reconocida por el plugin LSP. 2. Lleva a cabo alguno de los desencadenantes de los casos de uso anteriores. <p>El sistema:</p> <ol style="list-style-type: none"> 3. Analiza la petición del usuario y genera diagnósticos correspondientes a su petición. 4. Se envían los diagnósticos al cliente. 5. El editor muestra al usuario la información correspondiente a su petición.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El usuario modifica el fichero y el editor muestra los errores encontrados. <ul style="list-style-type: none"> ○ Referirse al apartado Editar documento con extensión reconocida. • Escenario Alternativo 2: El usuario utiliza el autocompletado. <ul style="list-style-type: none"> ○ Referirse al apartado Utilizar autocompletado. • Escenario Alternativo 3: El usuario sitúa el cursor encima de un elemento para obtener información al respecto. <ul style="list-style-type: none"> ○ Referirse al apartado Mostrar información flotante de un objeto.
Excepciones	<ul style="list-style-type: none"> • Existe un proceso ejecutándose en el puerto por defecto del procesador de lenguaje. <ul style="list-style-type: none"> ○ Se informa al usuario sobre el error.
Notas	-

5.6 Análisis de Interfaces de Usuario

5.6.1 Descripción de la Interfaz

Para seguir permitiendo la portabilidad del sistema al completo entre los distintos sistemas, se proponen dos alternativas de desarrollo:

- Desarrollar la interfaz en Java con un framework como Swing o JavaFX que pueda ser ejecutado en cualquier sistema operativo que permita la ejecución de ficheros .jar
- Desarrollar una interfaz web con HTML, CSS y JavaScript combinada con el framework Electron para convertirlo en una aplicación ejecutable multiplataforma.

La última alternativa resulta la más interesante debido al uso de nuevas tecnologías y a la característica de no requerir la existencia de ninguna herramienta externa para su ejecución. Además de la posibilidad de utilizar hojas de estilo predefinidas que doten a la aplicación de un estilo visual mucho más llamativo que el de una aplicación desarrollado con los frameworks mencionados para la primera alternativa.

Para los estilos se utilizará el kit de desarrollo proporcionado por Photon, que se encuentra especialmente enfocado para llevar a cabo aplicaciones con Electron, y proporciona ejemplos muy interesantes y una documentación muy completa para los distintos componentes que se pueden utilizar en una aplicación que utilice esta herramienta.

Una vez decidida la alternativa a llevar a cabo, se procede a la realización de bocetos que representarán un posible estado final de la interfaz de usuario de la aplicación. Los resultados son los siguientes:

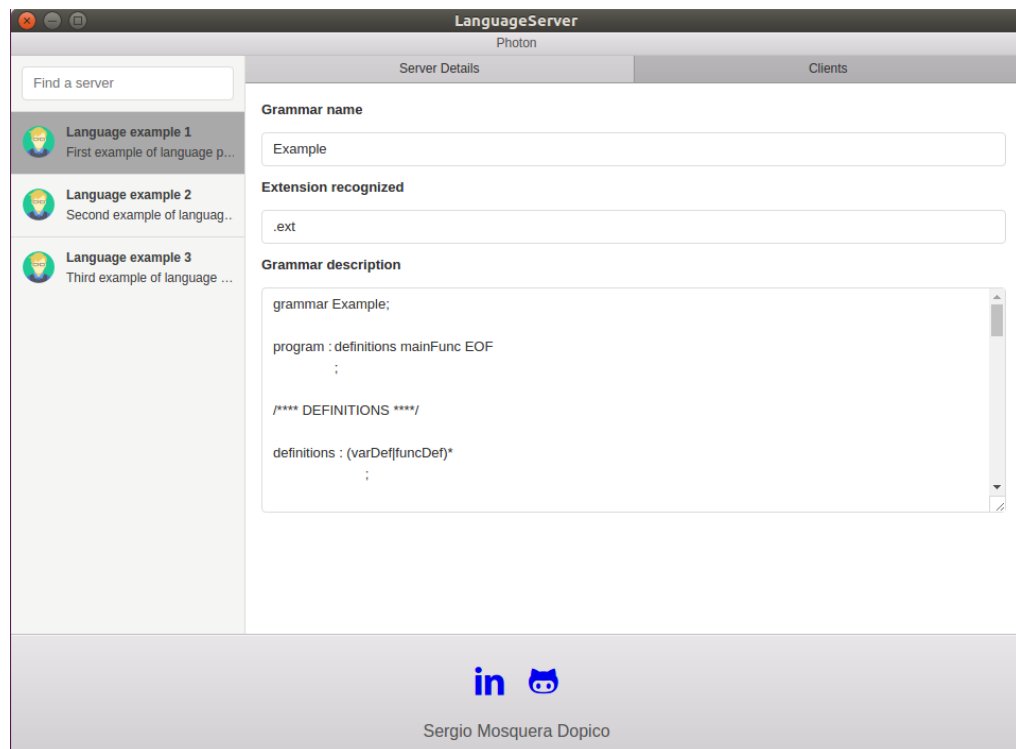


Ilustración 9. Pantalla de visualización de procesadores de lenguaje

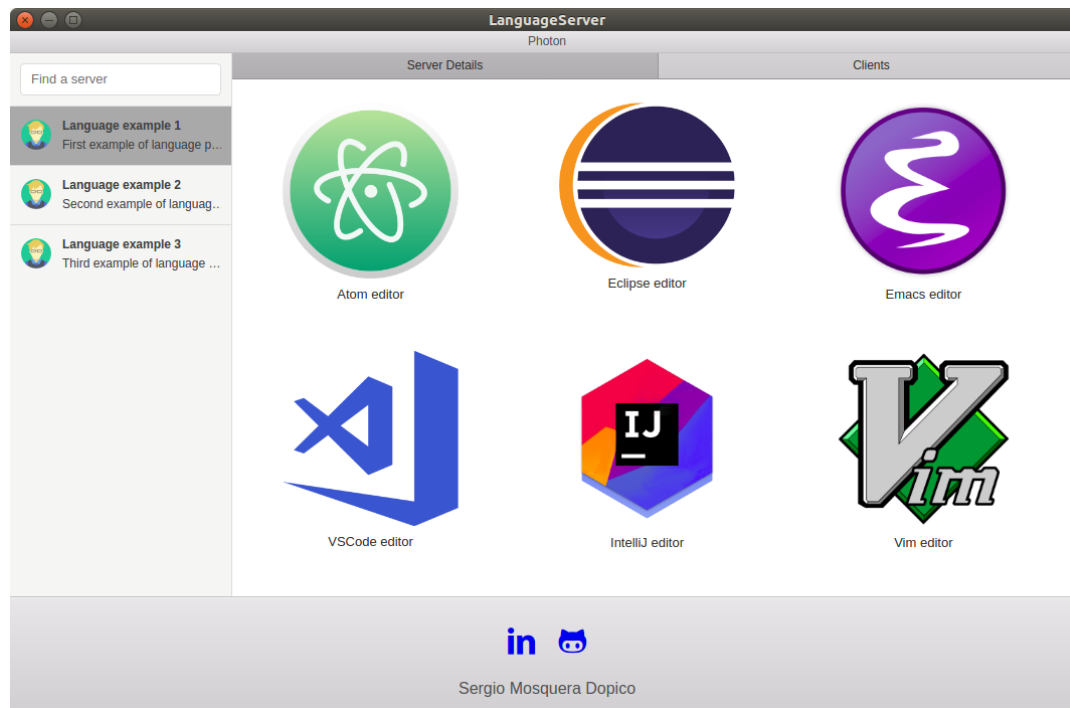


Ilustración 10. Editores con soporte para LSP

5.6.2 Descripción del Comportamiento de la Interfaz

A partir de las pantallas incluidas en el apartado anterior, se pueden definir las funcionalidades principales de la interfaz para cada una de ellas. Estas funcionalidades son susceptibles de cambiar en un diseño futuro:

- **Pantalla de procesadores de lenguaje.**
 - Se muestran en una lista a la izquierda los distintos procesadores de lenguaje creados.
 - Los datos de los elementos mostrados en la columna de la izquierda pueden ser modificados por el usuario.
 - Seleccionando uno de los procesadores de la izquierda, se despliega la información que aparece en el centro de la aplicación (nombre, extensión asociada y gramática al completo).
 - Realizando doble click en un elemento de la columna a la izquierda se procede a su eliminación.
 - Para crear uno nuevo, se hace click en el último elemento que se muestra con un símbolo +. Tras el click, se despliega una ventana pidiendo al usuario el fichero de la gramática correspondiente y la extensión asociada.
 - Se puede llevar a cabo un filtrado por nombre de los distintos procesadores de lenguaje escribiendo en el cuadro de texto que se encuentra en la parte superior de la columna izquierda.
 - En el pie de la aplicación aparecen dos enlaces a la página de LinkedIn y de GitHub del creador del sistema, junto con su nombre. El nombre del creador contiene un enlace a su cuenta de correo personal.

- **Pantalla de editores con soporte para LSP**
 - Una vez seleccionado un elemento de la columna de la izquierda y pulsada la pestaña de “Clients”, se despliega un listado de los distintos editores para los que el programa puede generar extensiones de reconocimiento de lenguajes.
 - El usuario hará click en las imágenes que representan a cada uno de los editores para seleccionar aquel para el que desea generar las extensiones.
 - Una vez seleccionado un editor, se llevará a cabo la tarea de generación de las extensiones, indicando, una vez generadas las extensiones, mediante una ventana emergente la ubicación de las mismas.
 - La cantidad de editores soportados es susceptible de ser alterada en las versiones sucesivas. Los editores que aparezcan en dicha pantalla pueden ser eliminados sustituidos por otros diferentes, además de poder añadirse nuevos editores que den soporte al LSP.

5.7 Especificación del Plan de Pruebas

5.7.1 Niveles de prueba

Los niveles de prueba son grupos de actividades de prueba organizadas y gestionadas en conjunto. Dentro de los niveles encontramos cuatro grupos de pruebas distintos:

- **Componentes:** se prueba la funcionalidad de los componentes identificados por separado. Como ya se ha explicado en apartados anteriores, el sistema principal se compone de los elementos generados (extensiones) y del programa generador de extensiones. Dentro de la extensión LSP, encontramos tres componentes que han de ser analizados como componentes individuales (cliente, servidor y procesador de lenguaje).
- **Integración:** se evalúan las interfaces que se encargan de la conexión entre los distintos componentes y su interacción con otras partes del sistema. Como ya se ha explicado en apartados anteriores, el sistema principal se compone de los elementos generados (extensiones) y del programa generador de extensiones. Con respecto al generador, no aplica realizar pruebas de integración, ya que no se subdivide en sistemas más pequeños que se encuentren conectados por medio de interfaces.
Respecto a la extensión LSP, los elementos se comunican utilizando dos protocolos diferentes; JSON-RPC para comunicar el cliente con el servidor, y HTTP para comunicar el servidor con el procesador de lenguaje. Será necesario por tanto llevar a cabo estas pruebas de integración, para asegurar que los componentes funcionan correctamente de forma conjunta.
- **Sistema:** se comprueba el comportamiento del sistema/producto de forma global. Se aplican pruebas tanto funcionales como no funcionales. Se evaluarán tanto las extensiones generadas como el generador de extensiones para asegurar el funcionamiento global del sistema. Para ello se llevará a cabo una ejecución completa del generador de extensiones y la instalación de las extensiones generadas para evaluar el funcionamiento del producto final.
- **Aceptación:** sirven para determinar cuando el sistema está listo para ser liberado. Estas pruebas suelen llevarse a cabo con el usuario cuyos requerimientos se quieren cumplir con la realización del sistema. En este caso al tratarse de un proyecto realizado por propio interés del autor, las pruebas de aceptación se llevarán a cabo con posibles usuarios finales del proyecto, ya que los resultados serán los más representativos con respecto al mercado del sistema.

5.7.2 Tipos de prueba

Los tipos de prueba son grupos de actividades de prueba para un componente o sistema enfocadas en un objetivo. Dentro de los tipos encontramos tres grupos de pruebas distintos:

- **Funcionales.**
- **No funcionales.**
 - **Interoperabilidad.** Una de las principales ventajas del LSP es que la parte del servidor (servidor y procesador de lenguaje) es reutilizable para los distintos editores, siendo necesario tan solo crear un nuevo módulo para el cliente dependiendo del editor que se utilice en cada caso. En este caso, la interoperabilidad se puede evaluar de forma estática, atendiendo a la definición del LSP. La parte del servidor solo podrá establecer conexión por medio del protocolo JSON-RPC, el cual se encuentra soportado por los SDK que dan soporte al LSP. Esto implica que el servidor se podrá conectar con los editores que den soporte a LSP, siempre que exista un módulo cliente que establezca comunicación con la parte servidor.
 - **Seguridad.** Al desplegarse la aplicación de forma local accediendo al servidor por medio de peticiones HTTP, sin necesitar en ningún momento el acceso a un servidor externo, se considera que las posibles situaciones en las que la seguridad se puede ver comprometida son mínimas. Se ha desestimado la posibilidad de establecer la comunicación por medio de HTTPS, además de por lo anteriormente explicado, por la necesidad de generar un certificado SSL para cada nueva extensión generada. Generar un nuevo certificado de cada vez ocasionaría un descenso del rendimiento en la generación de la extensión bastante notable.
 - **Carga, rendimiento o estrés.** La realización de estas pruebas no aplica al actual proyecto ya que la generación de extensiones se realiza de forma síncrona, no se puede llevar a cabo la generación de más de una extensión de cada vez. El único caso en el que se puede considerar de aplicación una prueba de las mencionadas en este punto, es llevando a cabo la medición del tiempo consumido por la aplicación para llevar a cabo la generación de las extensiones.
 - **Usabilidad y accesibilidad.** Al tratarse de una aplicación sin interfaz de usuario (consiste en ejecutar un script por consola), las pruebas de usabilidad no tienen mucho sentido para este proyecto, ya que su función principal es medir el grado de sencillez de utilización del sistema para un usuario. De todas formas, ya que es un proyecto orientado principalmente para desarrolladores en general y para personas focalizadas en el desarrollo de lenguajes de programación en particular, las pruebas de usabilidad se llevarán a cabo con personas pertenecientes a este campo de estudio.
 - **Fiabilidad, tolerancia a fallos, eficiencia y portabilidad.** De los tipos de pruebas anteriores, para el contexto de este proyecto solo aplican los tipos de portabilidad y fiabilidad. Esto ocurre debido en gran parte a que puede ejecutarse tanto en Windows como en Linux (tanto el sistema generación de extensiones, que consiste principalmente en un script que ejecuta la lógica de

las clases principales, como las extensiones en sí, que dependen de la disponibilidad del editor para los distintos sistemas operativos). Estas pruebas serán realizadas de forma manual, indicando los resultados obtenidos en los distintos sistemas operativos. Respecto a la fiabilidad, se tienen en cuenta los puntos críticos del sistema detectados en la fase de análisis. Las demás pruebas no aplican para este proyecto ya que no es un sistema que requiera estar en ejecución de forma continuada ni depende de infraestructuras complejas que puedan acarrear fallos en el despliegue o mantenimiento del sistema.

5.7.3 Casos de uso

Caso de Uso 1: Crear extensión para LSP

Caso de Uso 2: Crear extensión para el coloreado de la sintaxis

Prueba	Resultado Esperado
Introducir gramática correcta.	El sistema pregunta al usuario por la extensión del sistema
Prueba	Resultado Esperado
Introducir gramática incorrecta.	El sistema notifica al usuario sobre el error detectado en la definición del lenguaje en la gramática
Prueba	Resultado Esperado
Introducir extensión con un formato correcto.	El sistema genera dos carpetas que contienen las clases de la extensión LSP y de coloreado.
Prueba	Resultado Esperado
Introducir extensión con un formato incorrecto.	El sistema informa al usuario sobre el error cometido y le pide que introduzca una extensión de nuevo.

Caso de Uso 3: Editar documento con extensión reconocida

Prueba	Resultado Esperado
Editar documento con la extensión reconocida sin cometer ningún error léxico o sintáctico.	El sistema no mostrará ningún mensaje de error al usuario, ni por consola ni subrayando ningún término. Sin embargo, se mostrará como salida de consola en el procesador de lenguaje el texto del documento editado.
Prueba	Resultado Esperado
Editar documento con la extensión reconocida habiendo cometido algún error léxico o sintáctico.	El sistema mostrará los mensajes de error correspondientes al usuario. Se mostrará por consola y subrayando las partes el código que contengan errores. Sin embargo, se mostrará como salida de consola en el procesador de lenguaje el texto del documento editado.

Caso de Uso 4: Utilizar autocompletado

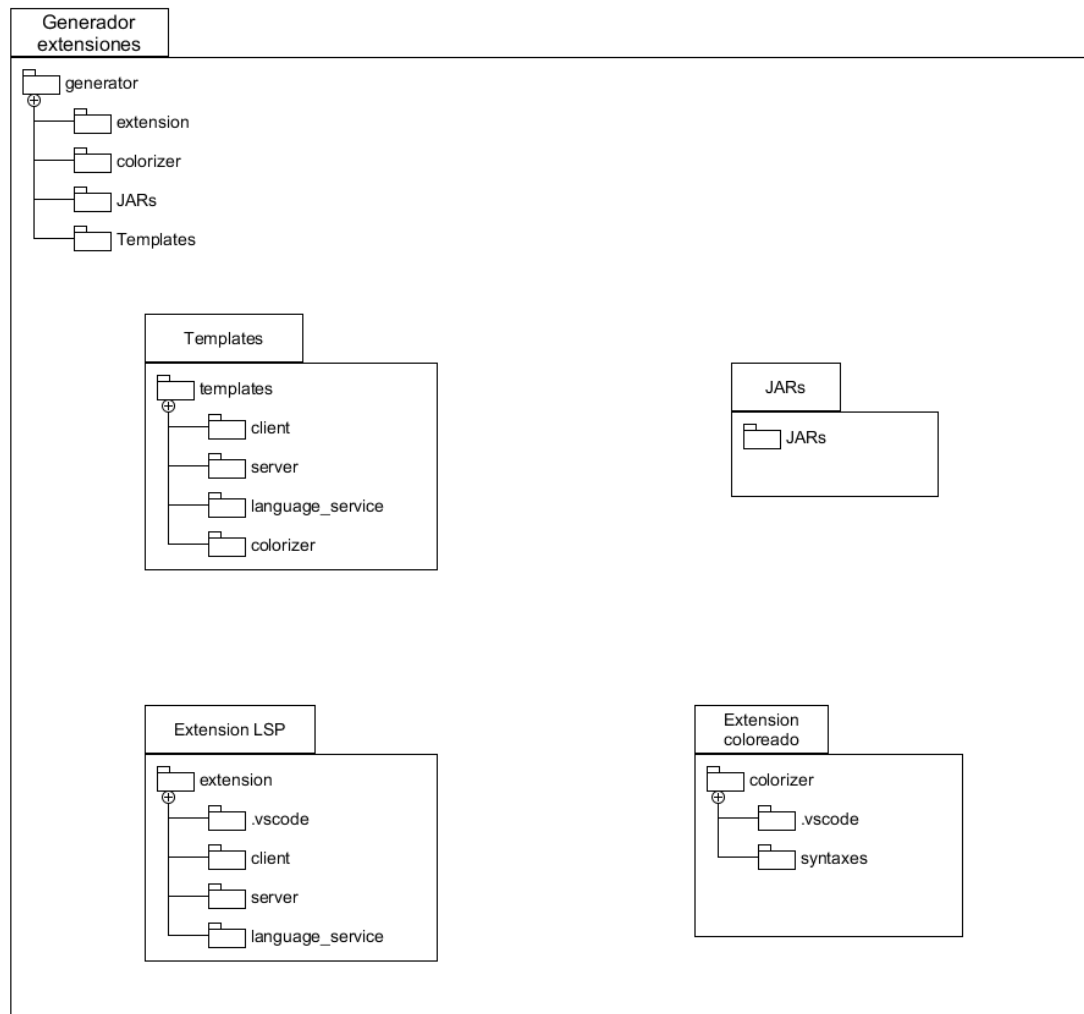
Prueba	Resultado Esperado
Utilizar el atajo de teclado de autocompletado para un término no reconocido por la extensión.	El sistema mostrará al usuario un listado vacío con los términos sugeridos para autocompletar su entrada.
Prueba	Resultado Esperado
Utilizar el atajo de teclado de autocompletado para los primeros caracteres de un término reconocido por la extensión.	El sistema mostrará al usuario un listado de todos los términos sugeridos para autocompletar su entrada.

<i>Caso de Uso 5: Mostrar información flotante de un objeto</i>	
Prueba	Resultado Esperado
Situar el ratón sobre un término que no disponga de información flotante asociada.	El sistema no mostrara al usuario ningún tipo de información sobre el término seleccionado.
Prueba	Resultado Esperado
Situar el ratón sobre un término que disponga de información flotante asociada.	El sistema mostrará al usuario en una ventana flotante la información asociada al elemento sobre el que se encuentre situado el cursor.

Capítulo 6. Diseño del Sistema

6.1 Arquitectura del Sistema

6.1.1 Diagramas de Paquetes



Dentro del paquete del generador se identifican 4 subpaquetes, de los cuales, la mitad se generan de forma dinámica (las extensiones). Con respecto a los otros dos, son necesarios para llevar a cabo la generación de las extensiones.

- El paquete **JARs** contendrá las clases Java necesarias para poder analizar la gramática recibida por parte del usuario y llevar a cabo la generación de las clases que permitan el reconocimiento de errores.
- El paquete **templates** contendrá las plantillas de los distintos ficheros que permitirán la creación de las extensiones adaptándose al lenguaje correspondiente a cada gramática.

6.1.1.1 Generator

El paquete principal del sistema. Es el que contiene los demás paquetes correspondientes a las extensiones generadas, las plantillas que se utilizan para que estas extensiones puedan ser generadas, y a los ficheros ejecutables de Java que permiten llevar a cabo la generación de clases de utilidades de ANTLR para el procesador de lenguaje. Además de los paquetes explicados a continuación, este paquete posee tres ficheros:

- **Package.json.** Sirve para gestionar todas las dependencias necesarias para la ejecución del generador. El sistema no podrá ejecutarse correctamente a no ser que se instalen estas dependencias.
- **Generar.sh.** Script ejecutable para sistemas Linux cuya ejecución comienza la tarea de generar las extensiones requeridas por el usuario. Recibe como parámetro una gramática escrita en ANTLR4 y, posteriormente, pregunta al usuario por un nombre para la extensión que representarán a los ficheros definidos para ese lenguaje.
- **Generar.bat.** Igual que el fichero anterior, pero para Windows.

6.1.1.1.1 Templates

Este directorio contiene una serie de subcarpetas. En ellas, se almacenan los ficheros de cada uno de los módulos de las extensiones que serán susceptibles de sufrir modificaciones. Estas modificaciones se deben a su dependencia con respecto al lenguaje que traten de identificar. Las carpetas que aparecen son 4 en concreto, una para cada uno de los módulos de las dos extensiones (3 para la extensión LSP y uno para la extensión de coloreado de la sintaxis). Además de los directorios esta carpeta contiene una serie de ficheros que permiten llevar a cabo el renombrado de cada una de las plantillas además de otros ficheros de configuración sueltos. Estos ficheros son los siguientes:

- **Launch.json.** Fichero que recoge las configuraciones necesarias para permitir la ejecución de la extensión una vez instalada.
- **Package.json.** Sirve para gestionar todas las dependencias necesarias para la ejecución del generador.
- **Colorizer_renaming.js.** Fichero que permite el renombrado de las clases para la extensión de coloreado de la sintaxis.
- **Lsp_renaming.js.** La misma lógica que el fichero anterior, pero para la extensión LSP.
- **First_renaming.js.** Este fichero permite llevar a cabo un renombrado de los ficheros JavaScript anteriormente mencionados. Estos ficheros contienen por defecto información que es dependiente del lenguaje que se desea reconocer, por lo que será necesario llevar a cabo un renombrado de estas partes dependientes.

6.1.1.1.2 JARs

Este directorio contiene varios ficheros Java (.jar) que consisten en clases de utilidades de ANTLR4 empaquetados. Estos ficheros permiten la generación de las clases que representan las distintas partes de un compilador obtenido a partir de una gramática ANTLR (scanner, parser, tokens, ...). En concreto los ficheros .jar que se incluyen en este directorio son los siguientes:

- **Antlr3-runtime.jar.**
- **Antlr4-runtime.jar.** Versión actualizada del anterior.
- **Antlr4.jar.**
- **Stringtemplate4.jar.**
- **Treelayout.jar.**

6.1.1.1.3 Extension

Este directorio contiene las distintas clases que ejecutarán la lógica referente a la extensión LSP generada por el programa principal. Como ya se ha explicado en apartados anteriores, la extensión LSP se divide en tres módulos diferentes, cada uno de los cuales se ve representado por una carpeta con sus ficheros correspondientes. Los componentes principales de los módulos cliente y servidor son las siguientes.

- **Carpeta src.** Contiene el código correspondiente al módulo escrito en el lenguaje de programación TypeScript.
- **Carpeta out.** Contiene la transpilación del código definido en la carpeta anterior en TypeScript a JavaScript, ya que, por lo contrario, este código no podría ser ejecutado.
- **Package.json.** Sirve para gestionar todas las dependencias necesarias para la ejecución del generador.

Además de estos dos directorios, encontramos un tercer directorio que es el correspondiente al procesador de lenguaje, el cual contiene tres ficheros y una carpeta:

- **Listeners.js.** Un listener es una estrategia a la hora de llevar a cabo el análisis sintáctico, la otra alternativa sería un visitor. ANTLR proporciona un buen soporte para la utilización de los listener para generar los distintos mensajes de diagnóstico que se van a enviar.
- **Index.js.** Clase correspondiente al servidor que se desplegará de forma local en un puerto determinado. También establece conexión con los listeners definidos en la clase superior para obtener información en un formato correcto que enviar al servidor para su posterior envío al cliente.
- **Carpeta .antlr.** Contiene la lógica generada por los ficheros de la carpeta JARs para la evaluación del contenido del fichero editado por el usuario con respecto a la gramática introducida por el mismo.
- **Package.json.** Sirve para gestionar todas las dependencias necesarias para la ejecución del generador.

6.1.1.1.4 Colorizer

Este directorio tan solo posee un directorio bajo su raíz (dos si se cuenta el directorio .vscode que contiene los ficheros correspondientes a la ejecución de la extensión) y dos ficheros. Estos elementos son los siguientes:

- **Package.json.** Sirve para gestionar todas las dependencias necesarias para la ejecución del generador.
- **Language_configuration.json.** Fichero que determina términos que serán reconocidos para su coloreado como comentarios o introducción de strings.

- **Carpeta `syntaxes`.** Este directorio contendrá el fichero principal de la extensión, `tmlanguage.json`, donde se encontrarán definidos todos los términos que serán reconocidos por la extensión para aplicarles posteriormente un coloreado diferente.

6.1.2 Diagramas de Componentes

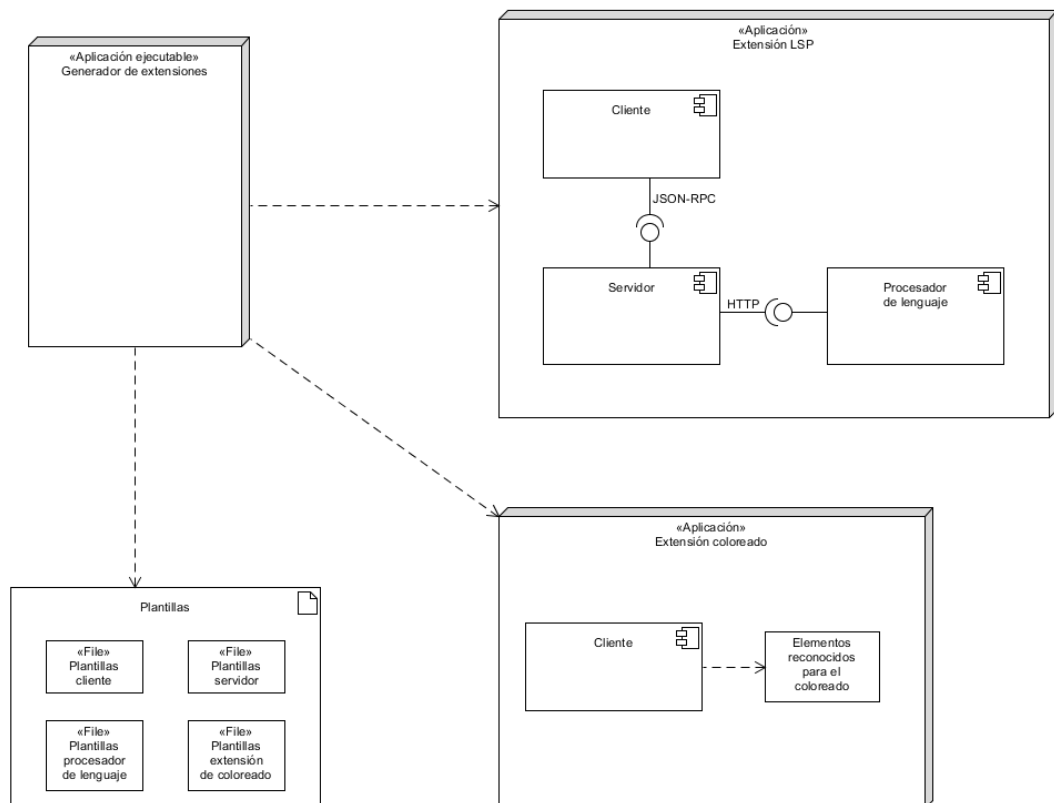


Ilustración 11. Diagrama de componentes del proyecto

La ilustración muestra los distintos componentes identificados en ambas extensiones y como se relacionan entre ellos, y las dependencias que posee la aplicación generadora de las extensiones con los ficheros externos necesarios para su correcta ejecución.

La función de los distintos componentes, junto con el contenido interno de cada uno se explica en apartados posteriores, concretamente en el apartado del Diagramas de Paquetes y en el apartado Descripción de las Clases.

6.1.3 Diagramas de Despliegue

Respecto al elemento principal de este proyecto, la aplicación de generación de extensiones no requiere ningún despliegue como tal (no se encuentra alojada en un servidor, o requiere un contenedor de servlets para estar disponible). Se basa principalmente en un fichero ejecutable que requiere al usuario una serie de inputs para generar una extensión a partir de plantillas.

Sin embargo, estas extensiones generadas sí que requieren un despliegue, el cual se realiza de forma automática una vez son instaladas en el editor correspondiente. A continuación, se incluye la explicación de los elementos que componen los diagramas de despliegue incluidos para cada una de estas extensiones.

6.1.3.1 Extensión LSP

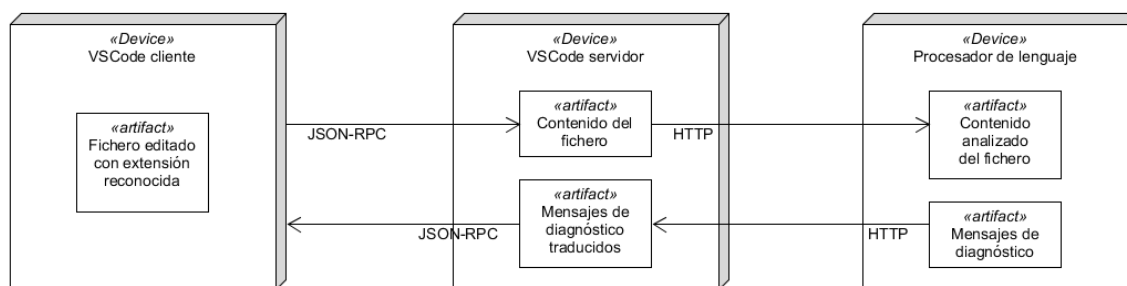


Ilustración 12. Diagrama de despliegue de la extensión LSP

La composición de esta extensión se basa en tres pilares fundamentales, una arquitectura cliente-servidor, una comunicación local de forma segura, y una alta posibilidad de reutilizar sus componentes principales (servidor y procesador de lenguaje). Su funcionamiento se aprecia en el diagrama superior y se puede resumir en cinco partes:

1. El usuario edita un fichero en el editor VSCode, el cual está identificado con la extensión elegida por el usuario en el momento de generar la extensión LSP.
2. El editor VSCode se comunica con el servidor que se encuentra en ejecución por medio del protocolo JSON-RPC y le envía el contenido del fichero editado.
3. El servidor formatea el contenido del fichero recibido y lo envía por medio de una conexión HTTP al procesador de lenguaje, el cual realiza un análisis de la información recibida para detectar errores.
4. Una vez el procesador ha analizado el fichero, envía un diagnóstico en forma de mensajes con los errores existentes al servidor mediante una respuesta HTTP.
5. Posteriormente el servidor envía estos diagnósticos de vuelta al cliente por medio de JSON-RPC para que el cliente se los muestre al cliente de una forma más clara y visual.

6.1.3.2 Extensión de coloreado.

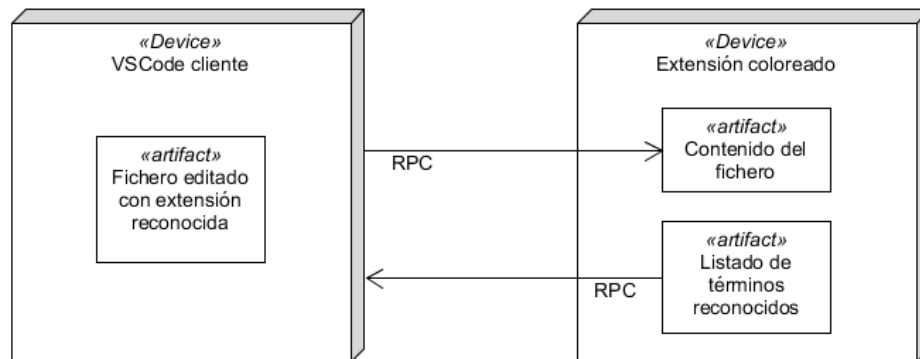


Ilustración 13. Diagrama de despliegue de la extensión de coloreado

La extensión de coloreado otorga al usuario la posibilidad de visualizar el contenido de los ficheros reconocidos por dicha extensión de una forma más amistosa. Para ello identifica una serie de tokens básicos del lenguaje utilizado para la generación de la extensión y les aplica un color distinto, lo cual facilita la legibilidad y agiliza la creación de código. El funcionamiento de dicha extensión se puede resumir en tres etapas.

1. El usuario edita un fichero en el editor VSCode, el cual está identificado con la extensión elegida por el usuario en el momento de generar la extensión de coloreado.
2. El editor se comunica con la extensión y le envía el contenido del fichero editado, el cual es comparado por la extensión contra su listado de términos básicos del lenguaje.
3. Una vez detectadas las coincidencias con los términos de la lista, envía un listado nuevo con los términos que si que han sido reconocidos en el contenido y es el propio editor el que se encarga de cambiar el color a dichos términos.

6.2 Diseño de Clases

Partiendo de la evaluación inicial realizada en la etapa de análisis de un posible diseño de las clases, se ha decidido cambiar la idea propuesta.

La razón de esta decisión viene en gran parte influenciada por las tecnologías utilizadas para el desarrollo (JavaScript + Node.js). Estas tecnologías permiten la creación de clases para estructurar los ficheros, pero también permiten establecer una disposición de los elementos para funcionar como scripts ejecutables simples sin necesidad de utilizar una jerarquía de clases.

Esta decisión también se ha visto influenciada en gran parte por la decisión de no utilizar una interfaz gráfica de usuario para la parte de generación de las extensiones. Ya que la funcionalidad que se busca es la de llevar a cabo la generación de extensiones a través de un fichero ejecutable por la línea de comandos, resulta más sencillo ejecutar comandos para las distintas partes, ya que el proceso de desarrollo de la extensión será transparente para el usuario.

En caso de haber decidido continuar con el desarrollo de la interfaz de usuario, probablemente se hubiese optado por un diseño orientado a objetos, ya que encajaría mucho mejor en dicha situación. Se utilizarían distintas clases para representar los distintos elementos que pudieran aparecer en la interfaz de usuario.

A pesar de esta decisión, en el código desarrollado se utilizan dos clases cuyas instancias actúan como objetos planos (equivalentes a un objeto JSON). Estas clases se utilizan para los diagnósticos generados por el procesador de lenguaje, tanto para los errores (clase Message) como para las distintas características del LSP (clase Name).

6.3 Diseño de la Interfaz

Tras evaluar el coste temporal para llevar a cabo el desarrollo del prototipo de interfaz obtenido en la fase de análisis, se opta por no llevar a cabo la interfaz de usuario. En detrimento, se utilizará un fichero ejecutable que mostrará su avance durante su ejecución por la línea de comandos.

Esta alternativa resulta viable ya que la interacción del usuario se puede resumir en dos acciones:

- Comenzar la ejecución del programa.
- Introducir los datos requeridos por la aplicación.
 - Gramática ANTLR.
 - Extensión asociada.

Las restricciones de las herramientas necesarias para llevar a cabo la ejecución siguen siendo las mismas que para el caso anterior, disponer de conexión a internet y del entorno Node.js en el sistema donde se ejecutará la aplicación.

6.4 Especificación Técnica del Plan de Pruebas

6.4.1 Pruebas de componentes

Como ya se ha mostrado en apartados anteriores como Arquitectura del Sistema, la extensión LSP generada consta de tres partes (componentes): cliente, servidor y procesador de lenguaje. Cada uno de estos componentes posee una única función, por tanto, las pruebas enmarcadas en este apartado se centrarán en los aspectos funcionales de cada uno de los módulos.

Dichas pruebas se mostrarán en forma de tabla para cada uno de los componentes, indicando la prueba llevada a cabo y su resultado. Se indica previamente que la mayor parte de estas pruebas serán llevadas a cabo de forma manual debido a la dificultad para automatizar las pruebas en este contexto.

En caso de que alguna prueba se realizase de forma automatizada, se indicará explícitamente.

El momento de ejecutar estas pruebas será al final de la fase de desarrollo para cada uno de los componentes. Una vez validado el resultado de estas pruebas, se llevarán a cabo pruebas para verificar la calidad del sistema, es decir, las pruebas de integración y sistema. Una vez que la calidad del sistema se ha asegurado lo suficiente por medio de estas pruebas, se presenta el sistema al cliente para que lleve a cabo las pruebas de aceptación, las cuales determinarán la verificación del sistema al completo para poder ser desplegado para los demás usuarios que deseen utilizar el sistema.

6.4.1.1 Pruebas para el cliente

Pruebas para el cliente		
Identificador	Tipo de la prueba	Descripción
PRCCL.1	Funcional.	Se lanza la parte cliente por medio de una configuración de despliegue en modo debug. Una vez desplegado se edita un documento con la extensión asociada a la gramática y se observa que la información que se debería transmitir por JSON-RPC es la deseada.
Identificador	Tipo de la prueba	Descripción
PRCCL.2	No Funcional. Portabilidad.	Llevar a cabo la prueba anterior en varios sistemas operativos diferentes. En particular, en Linux Ubuntu, en OS X y en Windows 10. Evaluar la interoperabilidad comprobando si los resultados obtenidos son iguales (o semejantes).

Tabla 4. Diseño de pruebas para el cliente

6.4.1.2 Pruebas para el servidor

Pruebas para el servidor		
Identificador	Tipo de la prueba	Descripción
PRCPS.1	Funcional.	Se lanza la parte servidor por medio de una configuración de despliegue en modo debug. Una vez desplegado se edita un documento con la extensión asociada a la gramática y se observa que la información que se debería transmitir por HTTP es la deseada.
Identificador	Tipo de la prueba	Descripción
PRCPS.2	Funcional.	Desplegar un servidor de prueba en un puerto concreto que devuelva por consola cualquier contenido recibido. De esa forma se comprueba que el contenido enviado por HTTP se recibe conteniendo la misma información que la que se ha enviado.
Identificador	Tipo de la prueba	Descripción
PRCPS.3	Funcional.	Desplegar un servidor de prueba en un puerto concreto que devuelva una respuesta cada vez que reciba una petición. El contenido de la respuesta será el mismo que el de la petición recibida. De esa forma se comprueba que el servidor puede recibir respuestas además de enviar peticiones.
Identificador	Tipo de la prueba	Descripción
PRCPS.4	No Funcional. Portabilidad.	Llevar a cabo las pruebas anteriores en varios sistemas operativos diferentes. En particular, en Linux Ubuntu, en OS X y en Windows 10. Evaluar la interoperabilidad comprobando si los resultados obtenidos son iguales (o semejantes).

Tabla 5. Diseño de pruebas para el servidor

6.4.1.3 Pruebas para el procesador de lenguaje

Pruebas para el procesador de lenguaje		
Identificador	Tipo de la prueba	Descripción
PRCPL.1	Funcional.	Se lanza el procesador de lenguaje en un puerto concreto. Una vez desplegado, se realiza una petición HTTP a este puerto cuyo contenido pueda ser reconocido por el servidor como un fragmento de texto sin errores gramaticales. Se comprueba que el contenido recibido por el servidor genera un diagnóstico sin errores.
Identificador	Tipo de la prueba	Descripción
PRCPL.2	Funcional.	Se lanza el procesador de lenguaje en un puerto concreto. Una vez desplegado, se realiza una petición HTTP a este puerto cuyo contenido pueda ser reconocido por el servidor como un fragmento de texto con errores gramaticales. Se comprueba que el contenido recibido por el servidor genera un diagnóstico con los errores detectados por el procesador de lenguaje.
Identificador	Tipo de la prueba	Descripción
PRCPL.3	Funcional. No Funcional. Interoperabilidad.	Se lanza el procesador de lenguaje en un puerto concreto. Se despliega un servidor de prueba en otro puerto de forma que envíe peticiones al procesador de lenguaje e imprima por la terminal el contenido de la respuesta recibida. Se comprueba que la respuesta recibida por este segundo servidor se corresponda con la enviada por el primero al recibir una petición. En esta prueba se evalúa que el procesador de lenguaje cumple con su funcionalidad y además se comprueba que es interoperable con el resto de los editores ya que puede recibir peticiones de cualquier origen y enviar respuestas a cualquier destino.
Identificador	Tipo de la prueba	Descripción
PRCPL.4	No Funcional. Portabilidad.	Llevar a cabo las pruebas anteriores en varios sistemas operativos diferentes. En particular, en Linux Ubuntu, en OS X y en Windows 10. Evaluar la interoperabilidad comprobando si los resultados obtenidos son iguales (o semejantes).

Tabla 6. Diseño de pruebas para el procesador de lenguaje

6.4.2 Pruebas de sistema

Se incluirá un símbolo de asterisco para representar que se ha cumplido la siguiente situación inicial *“Una vez generadas las extensiones sin errores, se procede a su instalación en el editor.”*. Este proceso permite evitar redundancias en la descripción de las pruebas que requieran la misma situación inicial.

Pruebas de sistema		
Identificador	Tipo de la prueba	Descripción
PRS.1	Funcional. No Funcional. Rendimiento. No funcional. Fiabilidad.	Se lleva a cabo la generación de las extensiones con varias (tres) gramáticas válidas diferentes. Cada gramática tendrá una extensión diferente a las anteriores (<15 reglas, <30 reglas y >30 reglas). Se comprueba que la generación ocurre sin errores para ninguna de las tres. Además de evaluar la funcionalidad del generador de extensiones, se realizará una medición del tiempo empleado por el programa para la generación de las extensiones. El cual no debería variar en más de 5 segundos para cada gramática.
Identificador	Tipo de la prueba	Descripción
PRS.2	Funcional. No funcional. Fiabilidad.	Se lleva a cabo la generación de las extensiones con una gramática inválida. Se comprueba que el proceso de generación informa de un error con respecto a la gramática.
Identificador	Tipo de la prueba	Descripción
PRS.3	Funcional. No funcional. Fiabilidad.	Se lleva a cabo la generación de las extensiones con una gramática valida, pero utilizando un formato incorrecto para la extensión. Se comprueba que el proceso de generación informa de un error con respecto a la extensión.
Identificador	Tipo de la prueba	Descripción
PRS.4	Funcional. No funcional. Fiabilidad.	Se lleva a cabo la generación de las extensiones con una gramática valida y con una extensión también válida. Se comprueba que el proceso de generación no informa de ningún tipo de error.
Identificador	Tipo de la prueba	Descripción
PRS.5	Funcional.	* Se abre un fichero de la extensión asociada y se edita el fichero cometiendo errores léxicos y/o sintácticos. Se comprueba que el editor devuelve los resultados de error esperados.
Identificador	Tipo de la prueba	Descripción
PRS.6	Funcional.	* Se abre un fichero de la extensión asociada y se edita el fichero sin cometer errores léxicos y/o sintácticos. Se comprueba que el editor no muestra ningún mensaje avisando de ningún error.

Identificador	Tipo de la prueba	Descripción
PRS.7	No Funcional. Portabilidad.	Llevar a cabo las pruebas anteriores en varios sistemas operativos diferentes. En particular, en Linux Ubuntu, en OS X y en Windows 10. Evaluar la interoperabilidad comprobando si los resultados obtenidos son iguales (o semejantes).

Tabla 7. Diseño de pruebas de sistema

6.4.3 Pruebas de aceptación

Este tipo de pruebas determinan la satisfacción del cliente con el producto final y son las que determinan la validez final del proyecto (o de sus distintos entregables) antes de poder dar por finalizada esa fase de desarrollo. En este caso particular, al no existir un cliente con el que realizar las pruebas de aceptación, se determinará la validez del sistema a partir de los resultados obtenidos a partir de las pruebas llevadas a cabo con otros usuarios. Los objetivos de llevar a cabo estas pruebas son:

- Mejorar la calidad de la interacción de los usuarios con nuestra aplicación.
- Reducir el tiempo necesario para hacer las distintas tareas en la aplicación y de esta forma aumentar la productividad.

Debemos tener los siguientes elementos en cuenta, para describirlos completamente en esta sección antes de hacer las pruebas en sí:

- **Usuarios:** Los usuarios seleccionados son personas que llevan más de un año dedicándose a labores de desarrollo de software en distintos ámbitos. Las pruebas se realizarán con tres usuarios distintos.
- **Lugar de realización:** Para que el ambiente sea lo más distendido y familiar posible para los usuarios, la evaluación se hará en la empresa correspondiente a dichos usuarios.
- **Metodología:** Uso de cuestionarios estructurados y con preguntas cerradas. Se opta por este tipo de cuestionarios ya que no contienen muchas preguntas, con lo cual el usuario no se sentirá abrumado o aburrido en el proceso de realización.

6.4.3.1 Diseño de Cuestionarios

6.4.3.1.1 Cuestionario de Evaluación

Debemos elaborar un cuestionario que será realizado a los usuarios para poder determinar así distintos aspectos del mismo y de su interacción con la aplicación. Los puntos a tocar son los siguientes:

1. **Preguntas de carácter general** a través de las cuales se determine el tipo de usuario y su nivel de conocimiento informático.
2. **Actividades guiadas** para llevar a cabo con el propio sistema.
3. **Batería de preguntas cortas** con los distintos aspectos de la aplicación que se pretendan evaluar.
4. **Observaciones**, para que el usuario aporte todo lo que considere oportuno de nuestra aplicación.

6.4.3.2 Actividades de las Pruebas de Usabilidad

6.4.3.2.1 Preguntas de carácter general

¿Con qué rama del desarrollo de software se identifica más?
<ol style="list-style-type: none"> 1. Administración de Sistemas y Redes. 2. Desarrollo frontend. 3. Desarrollo backend. 4. Ciencia de datos.
¿Ha trabajado alguna vez con herramientas de generación de lenguajes (ANTLR, Yacc, ...)?
<ol style="list-style-type: none"> 1. Sí, las uso a menudo. 2. Sí, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca.
¿Ha utilizado alguna vez un sistema de control de versiones (Git, SubVersion, CVS, ...)?
<ol style="list-style-type: none"> 1. Sí, lo uso a menudo. 2. Sí, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca.
¿Ha utilizado alguna vez un manejador de paquetes (npm, yarn, ...)?
<ol style="list-style-type: none"> 1. Sí, lo uso a menudo. 2. Si, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca
¿Qué le llama más la atención de un proyecto software?
<ol style="list-style-type: none"> 1. Que sea fácil de usar. 2. Que sea Open Source. 3. Que tenga una interfaz llamativa y vistosa. 4. Que cubra una necesidad real.

Tabla 8. Preguntas de carácter general

6.4.3.2.2 Actividades a realizar por el usuario (guiadas o sin guía)

A partir de los datos obtenidos en el anterior apartado, podemos inferir el conocimiento del usuario sobre el sistema. Una vez dispongamos de su perfil de conocimiento, decidiremos si la realización de las actividades explicadas a continuación se llevará a cabo de forma guiada o no guiada:

- Llevar a cabo el proceso de descarga del proyecto.
- Llevar a cabo el setup inicial para el proyecto (instalación de dependencias necesarias con npm).
- Ejecutar el programa de generación de las extensiones.

- Instalar las extensiones en el editor Visual Studio Code.

6.4.3.2.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Funciona cada tarea como Vd. Espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
Observaciones				
Estas han de ser redactadas por el usuario sin la intervención (u observación) del conductor de las pruebas.				

Tabla 9. Preguntas cortas sobre la aplicación

6.4.3.2.4 Cuestionario para el responsable de las pruebas

Aspectos a tener en cuenta	Notas
El usuario termina todas las tareas en los 15 minutos estipulados.	Al usuario le sobran alrededor de 6 minutos una vez finalizadas todas las pruebas.
Tareas más conflictivas para el usuario.	Comenzar con la ejecución de la aplicación generadora de extensiones.
El usuario requiere de ayuda del responsable para llevar a cabo las pruebas.	El usuario lleva a cabo las tareas de forma independiente aunque cometiendo algún fallo menor que le ralentiza ligeramente.
Errores cometidos por el usuario.	Inicialmente, el usuario desconoce la necesidad de tener que especificar la gramática a analizar como parámetro.
Errores cometidos por el sistema.	No se registra ningún error del sistema.
El usuario muestra interés en saber más sobre la aplicación.	El usuario pregunta si no sería posible generar extensiones para otros editores como IntelliJ.
El usuario propone posibles mejoras para la aplicación.	El usuario aconseja buscar una forma de guardar los servidores creados para que puedan reutilizarse.

Tabla 10. Cuestionario para el responsable

Los resultados de las pruebas se deben analizar conjuntamente para así establecer una conclusión respecto a cuatro aspectos de usabilidad:

1. **Facilidad de aprendizaje:** Capacidad para aprender la funcionalidad de la aplicación desarrollada y desarrollar las tareas de manera adecuada, midiendo cuanto se tarda en hacer las distintas tareas.
2. **Eficiencia:** Cuanto mejora la labor de los usuarios por usar la aplicación respecto a lo que se hacía anteriormente.
3. **Errores:** Cuantos errores cometen los usuarios en las distintas tareas, lo que decrementa la usabilidad del mismo.

4. **Satisfacción del usuario:** Impresión general de los usuarios al usar la aplicación.

Capítulo 7. Implementación del Sistema

7.1 Estándares y Normas Seguidos

7.1.1 LSP

El 27 de junio de 2016, Microsoft anuncia un acuerdo con las compañías RedHat y Codenvy para estandarizar el protocolo servidor de lenguaje (LSP). A día de hoy, se considera un estándar abierto que ha sido adoptado y posee soporte por parte de las tres compañías anteriores. La especificación de este protocolo se aloja y desarrolla en la plataforma de alojamiento de código GitHub.

Con la reciente compra de la plataforma GitHub por parte de Microsoft, el protocolo ha recibido un gran empuje y dispone de un mayor soporte por parte de las compañías colaboradoras para completar la especificación correctamente.

Una descripción más extensa del LSP se encuentra en el apartado Language Server Protocol (LSP).

7.1.2 JSON-RPC

El protocolo JSON-RPC es uno de los medios de transmisión de información que se utilizan dentro del protocolo LSP, en concreto es el que se utiliza para el intercambio de información entre el cliente y el servidor.

Es un protocolo cuya primera versión, la 1.0, surge en 2005 y ha ido recibiendo actualizaciones hasta marzo de 2010, cuando surge la revisión de la versión 2.0. La página que contiene esta versión de la especificación, sin embargo, se ha seguido actualizando hasta enero de 2013.

Las soluciones más comunes para las que se utiliza este protocolo es para aplicaciones que siguen una arquitectura cliente-servidor (igual que ocurre en la extensión generada), la cual se explica en el apartado Arquitectura cliente-servidor.

Este protocolo utiliza el formato JSON para el intercambio de información, el cual pertenece a los estándares RFC 7159 y ECMA-404.

7.2 Lenguajes de Programación

7.2.1 Bash

7.2.1.1 Descripción

El lenguaje de consola Bash (**B**ourne-**a**gain **s**hell) se encuentra presente en los sistemas operativos GNU/Linux, así como en Minix, Solaris, BSD, Mac e incluso Windows 10. Que se encuentre presente en estos sistemas se debe a que es compatible con el estándar POSIX, el cual cumplen los sistemas operativos anteriores.

En el año 1987, Stephan Bourne desarrolla Bash para el proyecto GNU, convirtiéndolo en la alternativa libre al lenguaje utilizado por el intérprete de comandos Bourne.

7.2.1.2 Razón de uso en el proyecto

Al tratarse de un programa sin interfaz gráfica, se intenta reducir al máximo la responsabilidad del usuario para el resultado final de la ejecución. Por tanto, la mejor opción es la automatización de la generación de las extensiones por medio de scripts de sencilla ejecución. La interacción con el usuario se lleva a cabo en dos ocasiones, en las que el usuario tendrá que proporcionar al programa una gramática válida escrita en ANTLR4 por un lado, y por otro lado una extensión para detectar los ficheros sobre los que se desea llevar a cabo la validación.

7.2.1.3 Versión

Se utiliza para el desarrollo una versión estable del programa (4.4.18).

7.2.1.4 Distribución

Este software se encuentra firmado bajo la licencia GPL (GNU General Purpose License), la cual garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software.

7.2.1.5 Complementos

No se ha utilizado ningún complemento para este lenguaje.

7.2.2 Batch

7.2.2.1 Descripción

Un tipo *batch* es un archivo de texto que contiene órdenes a ejecutar en un intérprete de órdenes propio de DOS o OS/2. Cuando es iniciado, un programa shell lo lee y ejecuta, generalmente línea a línea. De este modo, se emplea para ejecutar series de comandos automáticamente. Su extensión es .bat o .cmd.

El hecho de que funcione solo para MS-DOS lo hace muy limitado. Se puede considerar como la alternativa a Bash para sistemas operativos no mencionados en el apartado anterior. A pesar de que Windows permite la instalación de Bash, proporciona de forma nativa la programación Batch.

7.2.2.2 Razón de uso en el proyecto

La explicación es similar a la indicada en el apartado anterior Bash, pero con el fin de conseguir una aplicación que se pueda utilizar en varias plataformas, es necesario desarrollar los scripts de generación de las extensiones en dos formatos distintos (uno para sistemas operativos basados como Windows, y otro para sistemas operativos como Linux o Mac).

7.2.2.3 Versión

La versión utilizada es una versión estable del programa (6.2).

7.2.2.4 Distribución

Este software se encuentra firmado bajo una licencia EULA (End-user License Agreement) de Microsoft. Este tipo de licencias, por lo general, no permiten que el software sea modificado, desensamblado, copiado o distribuido de formas no especificadas en la propia licencia, regula el número de copias que pueden ser instaladas e incluso los fines concretos para los cuales puede ser utilizado.

7.2.2.5 Complementos

No se ha utilizado ningún complemento para este lenguaje.

7.2.3 JavaScript

7.2.3.1 Descripción

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,³ basado en prototipos, imperativo, débilmente tipado y dinámico.

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

7.2.3.2 Razón de uso en el proyecto

En la actualidad es uno de los lenguajes de programación más utilizados en todo el mundo. A pesar de que, normalmente se utiliza para programación orientada a la web, JavaScript ofrece muchas más posibilidades como la integración con diversos frameworks que permiten llevar a cabo proyectos muy interesantes. A lo largo del grado, la formación que reciben los alumnos en JavaScript es prácticamente nula, por tanto, se ha considerado que su utilización para el proyecto es una buena oportunidad para aprender a utilizar un lenguaje tan utilizado y demandado a día de hoy.

Otra razón que se ha tenido en cuenta (aunque es de menos peso que la anterior) es la intención de unificar los distintos ficheros de código en un lenguaje común. Y la elección de JavaScript se justifica por su aparición en varias partes del proyecto:

- Tanto el cliente como el servidor que se encuentran escritos en TypeScript, se transpilan a JavaScript para poder ser ejecutados.
- Las clases generadas por ANTLR para el reconocimiento del lenguaje se puede generar en distintos idiomas entre los cuales se encuentra JavaScript.

7.2.3.3 Versión

Como JavaScript se considera un dialecto de ECMAScript, la versión que se tiene en cuenta es la de ECMAScript. La versión que utilizamos es ES6 (ECMAScript 6) lanzada en 2015, a pesar de que la más reciente es la versión 8. Esta decisión se debe a que no necesitamos funcionalidades más allá de las definidas en la versión 6 y anteriores.

7.2.3.4 Distribución

JavaScript es una marca registrada de Oracle Corporation y es usada bajo licencia con los productos creados por Netscape Communications y entidades actuales como la Fundación Mozilla.

7.2.3.5 Complementos

Se utiliza el entorno de ejecución Node.js como complemento para este lenguaje de programación, el cual se explicará con más detalle en el apartado Node.js.

7.2.4 TypeScript

7.2.4.1 Descripción

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases.

TypeScript extiende la sintaxis de JavaScript, por tanto, cualquier código JavaScript existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

7.2.4.2 Razón de uso en el proyecto

La principal razón de su uso es que el SDK proporcionado por Visual Studio Code para el desarrollo de extensiones requiere el uso de TypeScript. Este lenguaje, puede ser transformado en código Javascript por un proceso de transpilación, el cual nos permite tener disponibles las extensiones en código JavaScript para su despliegue en el editor VSCode de forma inmediata.

Otra razón importante a destacar es que es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Al tratarse de un proyecto Open Source el sistema desarrollado, este lenguaje de programación encaja en la filosofía llevada a cabo. Además, es un lenguaje diseñado por Microsoft, al igual que el protocolo LSP, de esta forma centralizamos la tarea de soporte en el menor número de responsables.

7.2.4.3 Versión

La versión utilizada es una versión estable del programa (2.9.2).

7.2.4.4 Distribución

Como se ha mencionado con anterioridad, es un lenguaje de programación libre y de código abierto. En su página de Wikipedia se incluye información que indica que dispone de una licencia Apache. Este tipo de licencia permite al usuario del software de la libertad de usar el software para cualquier propósito, para distribuirlo, modificarlo y distribuir versiones modificadas del software, bajo los términos de la licencia.

7.2.4.5 Complementos

No se ha utilizado ningún complemento para este lenguaje.

7.3 Herramientas y Programas Usados para el Desarrollo

Descripción de todas las herramientas de desarrollo (Eclipse, Visual Studio, etc.), sistemas adicionales existentes, complementos y otros productos software que necesitemos para la implementación de nuestro sistema. Debemos dejar claro que versión usamos, para qué y cómo interactuará con nuestro sistema.

7.3.1 Node.js

7.3.1.1 Descripción

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema más grande de librerías de código abierto en el mundo. Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.

Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent, empresa en la que trabaja además su creador. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables como, por ejemplo, servidores web.

Node.js posee una larga lista de ventajas, pero sin duda la más destacable de ellas es el uso de módulos. Estos módulos dotan al programa desarrollado de nuevas funcionalidades y son muy versátiles a la hora de gestionar qué módulos se desean incluir y cuales se desean eliminar. Estos módulos se gestionan con el programa npm (Node Package Manager), el cual se explica más adelante.

7.3.1.2 Razón de uso en el proyecto e interacción con los demás elementos

La principal razón de optar por esta herramienta para el desarrollo es similar a la razón de elegir JavaScript, es una tecnología muy demandada hoy en día que, además, proporciona muchas utilidades. Entre estas utilidades destaca el uso de los módulos, los cuales permiten la inclusión de funcionalidad extra contenidas en paquetes externos para conseguir un código más limpio y potente.

Esta herramienta se utiliza a lo largo del proyecto para desarrollar dos elementos principales:

- Ficheros utilizados para el renombrado de las distintas clases. Se aprovechan los módulos fs y find-free-port principalmente para esta tarea.
- Clase utilizada para el procesador de lenguaje. Representa un servidor web que utiliza express y body-parser para recibir y enviar peticiones por medio de HTTP.

7.3.1.3 *Versión*

La versión utilizada es una versión estable del programa (8.11.3).

7.3.2 ANTLR4

7.3.2.1 Descripción

ANTLR (ANOther Tool for Language Recognition) es una herramienta creada principalmente por Terence Parr, que opera sobre lenguajes, proporcionando un marco para construir reconocedores (parsers), intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos (conteniendo acciones semánticas a realizarse en varios lenguajes de programación).

ANTLR cae dentro de la categoría de meta-programas, por ser un programa que escribe otros programas. Partiendo de la descripción formal de la gramática de un lenguaje, ANTLR genera un programa que determina si una sentencia o palabra pertenece a dicho lenguaje (reconocedor), utilizando algoritmos LL(*) de parsing.

7.3.2.2 Razón de uso en el proyecto e interacción con los demás elementos

La principal función de las extensiones generadas es permitir el reconocimiento de una gramática ANTLR en un editor de código concreto. Para poder llevar a cabo esta función es necesario disponer de una herramienta que permite la generación de clases que lleven a cabo el análisis de un fragmento de texto para determinar que cumplen con las reglas gramaticales definidas.

Las clases generadas son referenciadas desde la clase principal del procesador de lenguaje. En esta clase se instancian tanto un parser como un scanner para poder encontrar errores léxicos y sintácticos en el fragmento de código introducido por el usuario. Los errores detectados se devuelven al cliente y son los que posteriormente se representan en el editor para que el usuario pueda visualizarlos.

7.3.2.3 Versión

La versión utilizada es una versión estable del programa (4.7.1).

7.3.3 Visual Studio Code

7.3.3.1 Descripción

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software propietario.

A día de hoy, Visual Studio Code es uno de los editores de código más utilizados en todo el mundo, debido principalmente a la gran cantidad de extensiones que existen para él y su comunidad muy activa.

Most Popular Development Environments

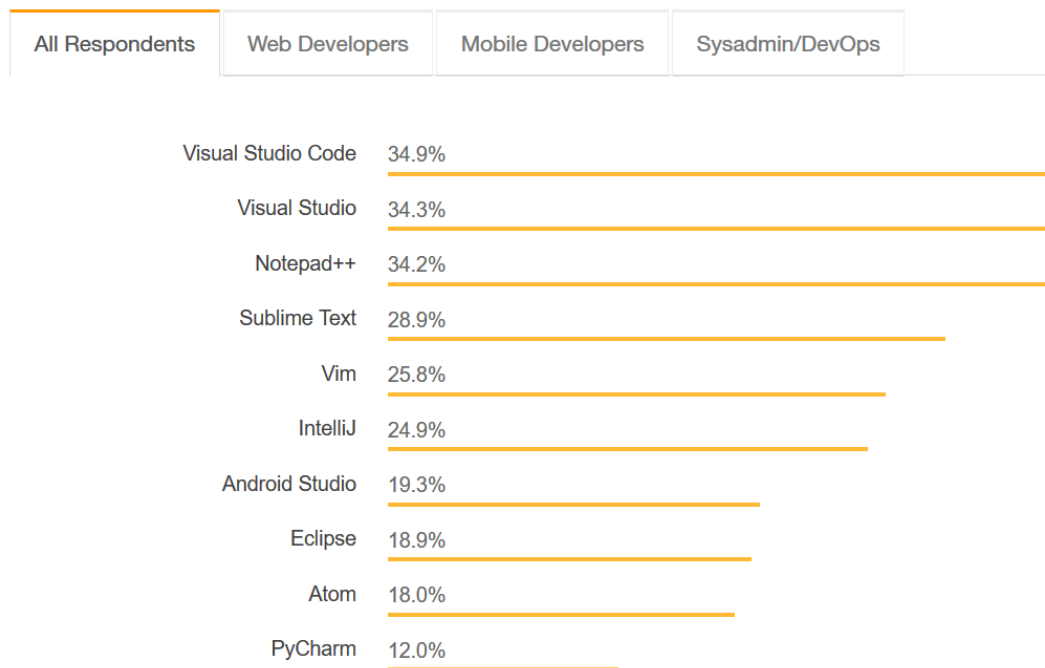


Ilustración 14. StackOverflow Survey 2018

7.3.3.2 Razón de uso en el proyecto e interacción con los demás elementos

Ambos, Language Server Protocol y Visual Studio Code, han sido creados por Microsoft. Por tanto, no es de extrañar que el primer editor que proporcionó soporte para el LSP fuese VSCode. Esta situación convierte a VSCode en el editor más consolidado para la realización de extensiones que utilicen LSP.

Resulta mucho más sencillo encontrar ejemplos de extensiones que utilizan LSP para VSCode que para cualquier otro editor, a pesar de que el crecimiento en su uso se encuentra en auge. Es por esta razón que se ha decidido que las extensiones desarrolladas se puedan utilizar para

VSCoDe en primera instancia. Como posibles funcionalidades futuras del proyecto se permitirá crear soporte para varios editores más.

7.3.3.3 Versión

La versión utilizada es una versión estable del programa (1.23.0).

7.3.4 Node Package Manager

7.3.4.1 Descripción

Node Package Manager o simplemente npm es un gestor de paquetes, que facilita mucho la tarea de trabajar con Node.js, ya que permite tener cualquier librería disponible con solo una línea de código. Npm además proporciona una ayuda a la hora de administrar los módulos, distribuir paquetes y agregar dependencias de una manera sencilla. Desde la versión 0.6.3 de Node.js, npm es instalado automáticamente con el entorno de Node.js.

Npm se ejecuta desde la línea de comandos y maneja las dependencias para una aplicación. Dispone de un gran repositorio con más de 700.000 módulos para utilizar en un programa Node.js. Estos módulos pueden desarrollados tanto por empresas altamente reconocidas como por cualquier usuario medio.

7.3.4.2 Razón de uso en el proyecto e interacción con los demás elementos

Al utilizarse Node.js como herramienta para el desarrollo del proyecto, como se explica en el apartado Node.js, será necesario disponer de un gestor de paquetes para las distintas dependencias que se utilicen en el proyecto.

Para completar el desarrollo del proyecto se utilizan una serie de paquetes de npm que se detallan a continuación:

- **antlr4.** Generación de clases ANTLR para el reconocimiento de la gramática.
 - Versión: ^4.7.1
- **find-free-port.** Utilizado para buscar un puerto libre en el dispositivo en el que se esté ejecutando el programa.
 - Versión: ^1.2.0
- **fs.** Proporciona una API para interactuar con el sistema de ficheros del dispositivo en el que se ejecute.
 - Versión: 0.0.1-security
- **node.** Determina la versión de Node.js que se utilizará para la ejecución del proyecto.
 - Versión: ^8.6.0
- **typescript.** Determina la versión de TypeScript que se utilizará en el desarrollo del proyecto.
 - Versión: 2.9.2
- **vscode.** Determina la versión de VSCode que se utilizará en el desarrollo del proyecto.
 - Versión: ^1.23.0.
- **vscode-languageclient.** Determina la versión del cliente de VSCode que se utilizará en el desarrollo del proyecto.
 - Versión: ^4.1.4
- **vscode-languageserver.** Determina la versión del servidor de VSCode que se utilizará en el desarrollo del proyecto.
 - Versión: ^4.1.3.

- **request.** Simplifica la lógica necesaria para crear una petición HTTP o HTTPS.
 - Versión: ^2.81.0.

7.3.4.3 Versión

La versión utilizada es una versión estable del programa (5.6.0) que data de mayo de 2018.

7.4 Creación del Sistema

7.4.1 Problemas Encontrados

A lo largo del desarrollo del proyecto me he topado con una serie de problemas que he necesitado subsanar. Entre estos problemas hago dos distinciones principales según el tipo de problema encontrado. Son los problemas de concepto y los problemas de desarrollo.

7.4.1.1 Problemas de concepto

7.4.1.1.1 Planteamiento inicial de la solución no válido

En un principio el proyecto se concibió de forma similar a la solución final aportada, como un generador de plugins de reconocimiento de lenguaje, aunque tan solo para el entorno de desarrollo Eclipse. Esta solución presentaba, a priori, una buena oportunidad por el amplio número de usuarios de este entorno de desarrollo y la cantidad de plugins desarrollados que existen para instalar en el entorno.

El principal problema que detecté fueron la falta de documentación actualizada, ya que el documento más reciente referente al desarrollo de plugins databa de 2008. Además de encontrarse la documentación desactualizada, también lo hacían los SDKs creados para tal fin, resultando en programas que no aprovechaban de ninguna forma las funcionalidades de las nuevas versiones de Java.

Determiné entonces que no tenía intención de llevar a cabo un proyecto con herramientas desactualizadas, además de tener que desarrollar las extensiones en el lenguaje de programación Java, algo nada nuevo para nosotros ya que lo utilizamos prácticamente cada año en la carrera.

A raíz de informarme sobre posibles alternativas, encontré el Language Server Protocol desarrollado por Microsoft, que me permitía subsanar todos los anteriores problemas:

- Las extensiones se desarrollan en TypeScript.
- Se estandarizó en 2016 y a día de hoy se sigue actualizando su documentación y se siguen lanzando nuevas versiones al mercado.
- Permite el desarrollo de extensiones de lenguaje portables entre los distintos entornos de desarrollo que proporcionen soporte para este protocolo.

7.4.1.1.2 Falta de soporte visual para la extensión

Una de las principales desventajas de la aplicación es que, si bien muestra los errores del código en la pantalla del editor al usuario, el color del texto es monocromático (mismo color para todos los términos). En la actualidad hay una gran demanda por parte de los usuarios hacia Microsoft para que esta característica se incluya como una de las funcionalidades opcionales para las extensiones que utilizan el LSP.

La solución a este problema no resultaba sencilla ya que, aunque se desarrollase otra extensión cuya función fuese únicamente la de llevar a cabo el coloreado de la sintaxis, la responsabilidad de determinar los términos importantes que aparecerán en la gramática recae principalmente en el usuario desarrollador de la gramática.

Como primera solución a este problema, el programa lleva a cabo la generación de una segunda extensión cuya responsabilidad es la de realizar el coloreado de la sintaxis especificada. Los términos especificados en la configuración de la extensión son los que el scanner generado de ANTLR ha determinado como elementos literales.

7.4.1.2 Problemas de desarrollo

7.4.1.2.1 Integración entre los componentes de la extensión

La estructura de la extensión LSP consta de tres módulos independientes que han de comunicarse entre sí y lo hacen utilizando dos protocolos distintos:

- JSON-RPC para comunicar al cliente y al servidor.
- HTTP para comunicar al servidor con el procesador de lenguaje.

El flujo de funcionamiento de la extensión una vez que detecta que el fichero modificado es de la extensión reconocida es el siguiente:

1. Se realiza una llamada a la función *“activate()”* del cliente.
2. La clase del cliente transpilada a JavaScript se sincroniza con el servidor (también transpilado) para el envío de información.
3. El cliente envía al servidor el contenido del fichero editado.
4. El servidor envía al procesador de lenguaje una petición HTTP con el contenido del fichero.
5. El procesador de lenguaje recibe la información, la analiza y genera un diagnóstico.
6. El procesador de lenguaje envía el diagnóstico al servidor como una respuesta HTTP y este último, se la reenvía al cliente por JSON-RPC.
7. El cliente interpreta el diagnóstico y lo representa en el editor para el usuario.

El error de integración surge en el punto 2, ya que la conexión entre los ficheros transpilados depende de la carpeta en la que se hayan generado. Inicialmente se establecía todo dentro de la carpeta cliente de la siguiente forma.

- *“extension/client/out/client.js”*. Localización de la clase cliente transpilada a JavaScript.
- *“extension/client/server/server.js”*. Localización de la clase servidor transpilada a JavaScript.

Esto producía un error ya que para establecer la conexión ambas clases debían encontrarse en paquetes separados, es decir en dos directorios de forma que cada directorio tuviese un fichero *package.json* distinto. La solución al error pasó por mover los ficheros transpilados del servidor a una carpeta similar a la de los ficheros del cliente, es decir, a *“extension/server/out/server.js”*.

Los errores de la conexión HTTP se explican en el siguiente apartado.

7.4.1.2.2 Formato del contenido de la petición

Este error se corresponde con el apartado anterior, en el que se explican los problemas de integración entre los módulos de la extensión LSP. En particular, este se enfoca en los problemas que han aparecido relacionados con la comunicación entre el servidor y el procesador de lenguaje que se lleva a cabo por HTTP.

El problema identificado es que el procesador de lenguaje detectaba más errores de los que existían, además de errores inexistentes. Para poder identificar la causa del problema llevé a cabo pruebas enviando peticiones de forma directa al procesador de lenguaje utilizando la herramienta *curl* con el formato especificado por los ejemplos que había tenido en cuenta para el desarrollo del sistema.

El problema ya aparecía en el momento de recepción de información por parte del procesador de código, el cual recibía la información con un formato extraño y tan solo la mitad del contenido. Buscando en Internet sobre este problema, encontré que los servidores web desarrollados con Express, requieren de otra dependencia que se denomina *body-parser*. Este paquete determina la codificación del contenido enviado y recibido por parte del servidor. Una vez incluida esta extensión, se hacía mucho más sencillo controlar el tipo de datos recibidos y enviados, de forma que configuré el servidor para recibir y enviar contenido en formato JSON (como especificaban los ejemplos).

A pesar de este cambio, el problema persistía, por tanto, la solución pasaba por enviar y recibir la información de la forma que necesitase el menor tipo de tratamiento necesario para funcionar. La opción aceptada fue la de enviar el contenido del fichero al procesador de lenguaje en texto plano sin ningún tipo de codificación para no requerir ningún tipo de traducción previa ni posterior, y el envío de diagnóstico en formato JSON, ya que es el formato nativo de JavaScript, haciendo así su tratamiento mucho más sencillo.

7.4.1.2.3 Reconocimiento de una extensión cualquiera por parte del editor

En los distintos editores de código se proporcionan asociaciones con ficheros en función de su extensión por defecto (.java, .js, .py, ...), y cada editor dispone de una forma diferente de establecer nuevas asociaciones. En los ejemplos que encontré, esta asociación de los ficheros para con las extensiones LSP se lleva a cabo con lenguajes cuyas extensiones ya se encontraban definidas por defecto en el propio editor.

El problema entonces residía en encontrar la forma de llevar a cabo la asociación de forma dinámica en el momento de generación (o instalación) de la extensión. A base de revisar la documentación de VSCode en su página oficial, encontré que la asociación de ficheros se hace de dos formas distintas, restringido al workspace (a la carpeta abierta por el usuario) o de uso general en el programa. La opción que realmente resultaba interesante era la segunda, ya que se puede asociar el reconocimiento a ese tipo de ficheros para todo el programa, ¿para qué íbamos a restringirlo al workspace del usuario?

La solución pasaba por editar un fichero de configuración en el directorio del usuario que estuviese utilizando el sistema, añadiendo dos líneas que indicasen el nombre que se le da a la asociación, junto con la extensión que será reconocida. Esto se incluye en el proceso de

generación de las extensiones, y se lleva a cabo de la misma forma que el renombrado de las características de los ficheros que son dependientes del lenguaje introducido por el usuario.

Capítulo 8. Desarrollo de las Pruebas

En este apartado se incluye el resultado de las pruebas diseñadas en el apartado Especificación Técnica del Plan de Pruebas. Se llevarán a cabo y se anotarán los resultados (positivo o negativo) en la tabla junto con el identificador de cada prueba (cuando la prueba disponga de un identificador) junto con una columna de observaciones sobre el desarrollo de la prueba, por ejemplo, cambios en la situación inicial, problemas encontrados u otros datos relevantes respecto a la ejecución y/o resultados de la prueba.

Salvo que la prueba lo requiera de forma específica, el entorno para llevar a cabo la ejecución de las pruebas será el preparado en el ordenador del alumno, el cual contiene las dependencias necesarias para que el sistema funcione correctamente. El sistema operativo utilizado por defecto será Ubuntu 16.04, aunque como segunda alternativa se utilizaría Windows 10 si la prueba así lo requiriese.

8.1 Pruebas de componentes

<i>Pruebas para el cliente</i>		
Identificador	Resultado	Observaciones
PRCCL.1	Positivo.	Se ha tenido que cambiar la implementación del componente respecto al resultado final para añadir la posibilidad de mostrar por consola el contenido que se transmitiría por JSON-RPC al servidor (para esta prueba se supone inexistente el servidor).
Identificador	Resultado	Observaciones
PRCCL.2	Positivo.	Se ejecuta el sistema en Windows de la misma forma que se ejecutó la prueba anterior en Linux y se obtuvieron los mismos resultados.

Tabla 11. Resultado de las pruebas para el componente cliente

<i>Pruebas para el servidor</i>		
Identificador	Resultado	Observaciones
PRCPS.1	Positivo.	Esta prueba necesita que exista un cliente funcional que le transmita la información, por tanto, esta prueba se realiza de forma posterior a PRCCL.1.
Identificador	Resultado	Observaciones
PRCPS.2	Positivo.	El contenido se recibe de forma correcta y se muestra por la terminal desde la que se despliega el procesador de lenguaje.
Identificador	Resultado	Observaciones
PRCPS.3	Positivo.	El servidor recibe correctamente la respuesta y muestra por consola el contenido esperado.
Identificador	Resultado	Observaciones
PRCPS.4	Positivo.	Se ejecutan las pruebas en Windows siguiendo el mismo procedimiento que en Linux y se obtienen los mismos resultados.

Tabla 12. Resultados de las pruebas para el componente servidor

<i>Pruebas para el procesador de lenguaje</i>		
Identificador	Resultado	Observaciones
PRCPL.1	Positivo.	Se lleva a cabo el mismo procedimiento que para PRCPS.2.
Identificador	Resultado	Observaciones
PRCPL.2	Positivo.	Se lleva a cabo el mismo procedimiento que para PRCPS.3.
Identificador	Resultado	Observaciones
PRCPL.3	Positivo.	El servidor que envía peticiones se programa para que envíe periódicamente peticiones (cada 20 segundos) e imprima tanto la petición enviada (un texto simple de dos líneas), como la respuesta recibida por parte del procesador de lenguaje.
Identificador	Resultado	Observaciones
PRCPL.4	Positivo.	Se ejecutan las pruebas en Windows siguiendo el mismo procedimiento que en Linux y se obtienen los mismos resultados.

Tabla 13. Resultados de las pruebas para el procesador de lenguaje

8.2 Pruebas de sistema

Pruebas de sistema		
Identificador	Resultado	Observaciones
PRS.1	Positivo.	Se lleva a cabo la prueba con tres gramáticas de 5, 20 y 35 reglas respectivamente. La generación de las extensiones se lleva a cabo 3 veces con cada una y se realiza la media del tiempo de generación. La diferencia de tiempo entre la gramática de 5 y la de 20 es de apenas 2 segundos y entre la de 20 y la de 35 de menos de 1 segundo.
Identificador	Resultado	Observaciones
PRS.2	Positivo.	El script de generación informa de que la gramática introducida ha de ser válida y detiene la ejecución.
Identificador	Resultado	Observaciones
PRS.3	Positivo.	El script de generación informa de que el formato de la extensión introducida ha de ser válida y detiene la ejecución.
Identificador	Resultado	Observaciones
PRS.4	Positivo.	Los ficheros se generan correctamente.
Identificador	Resultado	Observaciones
PRS.5	Positivo.	El editor subraya los errores en el código y además los muestra por consola.
Identificador	Resultado	Observaciones
PRS.6	Positivo.	El editor no muestra ningún informe de error.
Identificador	Resultado	Observaciones
PRS.7	Positivo.	Se ejecutan las pruebas en Windows siguiendo el mismo procedimiento que en Linux y se obtienen los mismos resultados.

Tabla 14. Resultados de las pruebas de sistema

8.3 Pruebas de aceptación

Este tipo de pruebas se llevan a cabo con tres usuarios con perfiles distintos. Para llevar a cabo las pruebas se les proporciona un entorno con todas las herramientas necesarias instaladas y una gramática simple de prueba (15 reglas).

8.3.1 Usuario 1

8.3.1.1 Preguntas de carácter general

¿Con qué rama del desarrollo de software se identifica más?
<ol style="list-style-type: none"> 1. Administración de Sistemas y Redes. 2. Desarrollo frontend. 3. <u>Desarrollo backend.</u> 4. Ciencia de datos.
¿Ha trabajado alguna vez con herramientas de generación de lenguajes (ANTLR, Yacc, ...)?
<ol style="list-style-type: none"> 1. Sí, las uso a menudo. 2. <u>Sí, alguna vez.</u> 3. No, aunque conozco su funcionamiento. 4. No, nunca.
¿Ha utilizado alguna vez un sistema de control de versiones (Git, SubVersion, CVS, ...)?
<ol style="list-style-type: none"> 1. <u>Sí, lo uso a menudo.</u> 2. Sí, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca.
¿Ha utilizado alguna vez un manejador de paquetes (npm, yarn, ...)?
<ol style="list-style-type: none"> 1. <u>Sí, lo uso a menudo.</u> 2. Si, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca
¿Qué le llama más la atención de un proyecto software?
<ol style="list-style-type: none"> 1. Que sea fácil de usar. 2. Que sea Open Source. 3. Que tenga una interfaz llamativa y vistosa. 4. <u>Que cubra una necesidad real.</u>

Tabla 15. Preguntas de carácter general para el usuario 1

8.3.1.2 Preguntas cortas sobre la aplicación y observaciones

A partir de los resultados obtenidos en el anterior cuestionario, se decide que la realización de las pruebas se llevará a cabo de forma independiente por parte del usuario, solo guiándole en caso de que el propio usuario requiera ayuda.

Facilidad de Uso	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Sabe dónde está dentro de la aplicación?	X			
¿Existe ayuda para las funciones en caso de que tenga dudas?	X			
¿Le resulta sencillo el uso de la aplicación?	X			
Funcionalidad	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Funciona cada tarea como Vd. Espera?	X			
¿El tiempo de respuesta de la aplicación es muy grande?			X	
Observaciones				
La aplicación de generación de extensiones apenas tiene complicación, ya que está todo muy bien explicado en el fichero README. A pesar de que utilizando un script para ejecutar toda la lógica necesaria se logra una generación más ágil, resultaría más interesante para el usuario utilizar una interfaz gráfica.				

Tabla 16. Preguntas cortas sobre la aplicación para el usuario 1

8.3.1.3 Cuestionario para el responsable de las pruebas

Aspectos a tener en cuenta	Notas
¿El usuario termina todas las tareas en los 15 minutos estipulados?	Al usuario le sobran alrededor de 6 minutos una vez finalizadas todas las pruebas.
Tareas más conflictivas para el usuario.	El usuario completó todas las tareas correctamente tras leer el fichero README.
¿El usuario requiere de ayuda del responsable para llevar a cabo las pruebas?	El usuario lleva a cabo las tareas de forma independiente, sin requerir en ningún momento la ayuda del conductor de las pruebas.
Errores cometidos por el usuario.	El usuario no ha cometido ningún error.
Errores cometidos por el sistema.	No se registra ningún error por parte del sistema.
El usuario muestra interés en saber más sobre la aplicación.	El usuario pregunta por la estructura del sistema, literalmente “¿Cómo hace la aplicación para encontrar los errores en el código?”.
El usuario propone posibles mejoras para la aplicación.	El usuario pregunta si se podría desarrollar algo similar para IntelliJ ya que es el editor que utiliza normalmente.

Tabla 17. Cuestionario para el responsable de las pruebas para el usuario 1

8.3.2 Usuario 2

8.3.2.1 Preguntas de carácter general

¿Con qué rama del desarrollo de software se identifica más?
<ol style="list-style-type: none"> 1. Administración de Sistemas y Redes. 2. Desarrollo frontend. 3. Desarrollo backend. 4. <u>Ciencia de datos.</u>
¿Ha trabajado alguna vez con herramientas de generación de lenguajes (ANTLR, Yacc, ...)?
<ol style="list-style-type: none"> 1. Sí, las uso a menudo. 2. Sí, alguna vez. 3. <u>No, aunque conozco su funcionamiento.</u> 4. No, nunca.
¿Ha utilizado alguna vez un sistema de control de versiones (Git, SubVersion, CVS, ...)?
<ol style="list-style-type: none"> 1. <u>Sí, lo uso a menudo.</u> 2. Sí, alguna vez. 3. No, aunque conozco su funcionamiento. 4. No, nunca.
¿Ha utilizado alguna vez un manejador de paquetes (npm, yarn, ...)?
<ol style="list-style-type: none"> 1. Sí, lo uso a menudo. 2. Si, alguna vez. 3. <u>No, aunque conozco su funcionamiento.</u> 4. No, nunca
¿Qué le llama más la atención de un proyecto software?
<ol style="list-style-type: none"> 1. <u>Que sea fácil de usar.</u> 2. Que sea Open Source. 3. Que tenga una interfaz llamativa y vistosa. 4. Que cubra una necesidad real.

Tabla 18. Preguntas de carácter general para el usuario 2

8.3.2.2 Preguntas cortas sobre la aplicación y observaciones

A partir de los resultados obtenidos en el anterior cuestionario, se decide que la realización de las pruebas se llevará a cabo de forma independiente por parte del usuario, solo guiándole en caso de que el propio usuario requiera ayuda.

Facilidad de Uso	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Sabe dónde está dentro de la aplicación?		X		
¿Existe ayuda para las funciones en caso de que tenga dudas?		X		
¿Le resulta sencillo el uso de la aplicación?	X			
Funcionalidad	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Funciona cada tarea como Vd. Espera?		X		
¿El tiempo de respuesta de la aplicación es muy grande?			X	
Observaciones				
El uso de la herramienta no resulta muy intuitivo para el usuario, se echa de menos una interfaz gráfica de usuario.				

Tabla 19. Preguntas cortas sobre la aplicación para el usuario 2

8.3.2.3 Cuestionario para el responsable de las pruebas

Aspectos a tener en cuenta	Notas
¿El usuario termina todas las tareas en los 15 minutos estipulados?	El usuario lleva a cabo todas las pruebas sobrándole entre 1 y 2 minutos
Tareas más conflictivas para el usuario.	Llevar a cabo el setup inicial para el proyecto (instalación de las dependencias necesarias con npm). Ejecutar el programa de generación de las extensiones.
¿El usuario requiere de ayuda del responsable para llevar a cabo las pruebas?	El usuario lleva a cabo las tareas de forma independiente, aunque cometiendo algún fallo menor en las tareas mencionadas anteriormente que requieren la ayuda del conductor de la prueba. Esto ralentiza ligeramente la finalización de las pruebas.
Errores cometidos por el usuario.	El usuario no conocía la necesidad de instalar las dependencias necesarias con npm install para poder ejecutar el sistema. Inicialmente, el usuario desconoce la necesidad de tener que especificar la gramática a analizar como parámetro.
Errores cometidos por el sistema.	No se registra ningún error del sistema.
¿El usuario muestra interés en saber más sobre la aplicación?	El usuario no realiza ninguna pregunta.
¿El usuario propone posibles mejoras para la aplicación?	El usuario aconseja utilizar una interfaz gráfica para mejorar la facilidad de uso de la aplicación.

Tabla 20. Preguntas para el responsable de las pruebas para el usuario 2

8.3.3 Usuario 3

8.3.3.1 Preguntas de carácter general

¿Con qué rama del desarrollo de software se identifica más?
<ol style="list-style-type: none"> 1. <u>Administración de Sistemas y Redes.</u> 2. Desarrollo frontend. 3. Desarrollo backend. 4. Ciencia de datos.
¿Ha trabajado alguna vez con herramientas de generación de lenguajes (ANTLR, Yacc, ...)?
<ol style="list-style-type: none"> 1. Sí, las uso a menudo. 2. Sí, alguna vez. 3. No, aunque conozco su funcionamiento. 4. <u>No, nunca.</u>
¿Ha utilizado alguna vez un sistema de control de versiones (Git, SubVersion, CVS, ...)?
<ol style="list-style-type: none"> 1. Sí, lo uso a menudo. 2. Sí, alguna vez. 3. <u>No, aunque conozco su funcionamiento.</u> 4. No, nunca.
¿Ha utilizado alguna vez un manejador de paquetes (npm, yarn, ...)?
<ol style="list-style-type: none"> 1. Sí, lo uso a menudo. 2. Si, alguna vez. 3. <u>No, aunque conozco su funcionamiento.</u> 4. No, nunca
¿Qué le llama más la atención de un proyecto software?
<ol style="list-style-type: none"> 1. Que sea fácil de usar. 2. Que sea Open Source. 3. Que tenga una interfaz llamativa y vistosa. 4. <u>Que cubra una necesidad real.</u>

Tabla 21. Preguntas de carácter general para el usuario 3

8.3.3.2 Preguntas cortas sobre la aplicación y observaciones

A partir de los resultados obtenidos en el anterior cuestionario, se decide que la realización de las pruebas se llevará a cabo de forma guiada, ya que se considera que podría encontrarse con muchas dificultades en la realización de las mismas.

Facilidad de Uso	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Sabe dónde está dentro de la aplicación?		X		
¿Existe ayuda para las funciones en caso de que tenga dudas?			X	
¿Le resulta sencillo el uso de la aplicación?			X	
Funcionalidad	Siempre	Frecuentemente	Ocasionalmente	Nunca
¿Funciona cada tarea como Vd. Espera?		X		
¿El tiempo de respuesta de la aplicación es muy grande?		X		
Observaciones				
Resulta muy complicado trabajar con ficheros bash si no estás acostumbrado a ello. La mejor solución pasaría por incluir una interfaz gráfica para aquellos usuarios que no controlen este tipo de ejecución.				

Tabla 22. Preguntas cortas sobre la aplicación para el usuario 3

8.3.3.3 Cuestionario para el responsable de las pruebas

Aspectos a tener en cuenta	Notas
¿El usuario termina todas las tareas en los 15 minutos estipulados?	El usuario no acaba a tiempo la tarea de instalar las extensiones en el editor. Necesita 5 minutos extra para terminar todas las pruebas.
Tareas más conflictivas para el usuario.	La mayor parte de las tareas, aunque en particular la tarea inicial (instalar dependencias necesarias con npm) y la final (instalar las extensiones).
¿El usuario requiere de ayuda del responsable para llevar a cabo las pruebas?	El proceso de realización de las tareas se realiza de forma guiada. Aunque no se le dice punto por punto lo que hacer, si que se le dan unas directrices básicas para poder terminar las tareas.
Errores cometidos por el usuario.	El usuario no sabía cómo descargar código de un repositorio de GitHub. El usuario no conocía la necesidad de instalar las dependencias, ni tampoco la forma de instalarlas. El usuario no conocía la forma de instalar las extensiones y de probarlas a continuación.
Errores cometidos por el sistema.	No se registra ningún error del sistema.
El usuario muestra interés en saber más sobre la aplicación.	El usuario no realiza preguntas.
El usuario propone posibles mejoras para la aplicación.	El usuario propone que la aplicación muestre un tutorial al usuario para facilitar la ejecución de la aplicación.

Tabla 23. Preguntas para el responsable de las pruebas para el usuario 3

Capítulo 9. Manuales del Sistema

9.1 Manual de Instalación

Este apartado contendrá una explicación de la instalación de todas las herramientas necesarias para poder ejecutar el programa correctamente.

9.1.1 Instalación de Node.js

Independientemente del sistema operativo en el que queramos ejecutar el programa, necesitaremos tener instalado Node.js y npm, las correspondientes versiones se encuentran en el apartado Herramientas y Programas Usados para el Desarrollo.

Ambos programas se pueden instalar con un gestor de paquetes como es apt-get o yum de la siguiente manera:

```
apt - get install nodejs == yum install nodejs  
apt - get install npm == yum install npm
```

Otra opción es descargándolos directamente de la página oficial tanto de [Node](https://nodejs.org/) como de [npm](https://www.npmjs.com/) e instalándolos a mano.

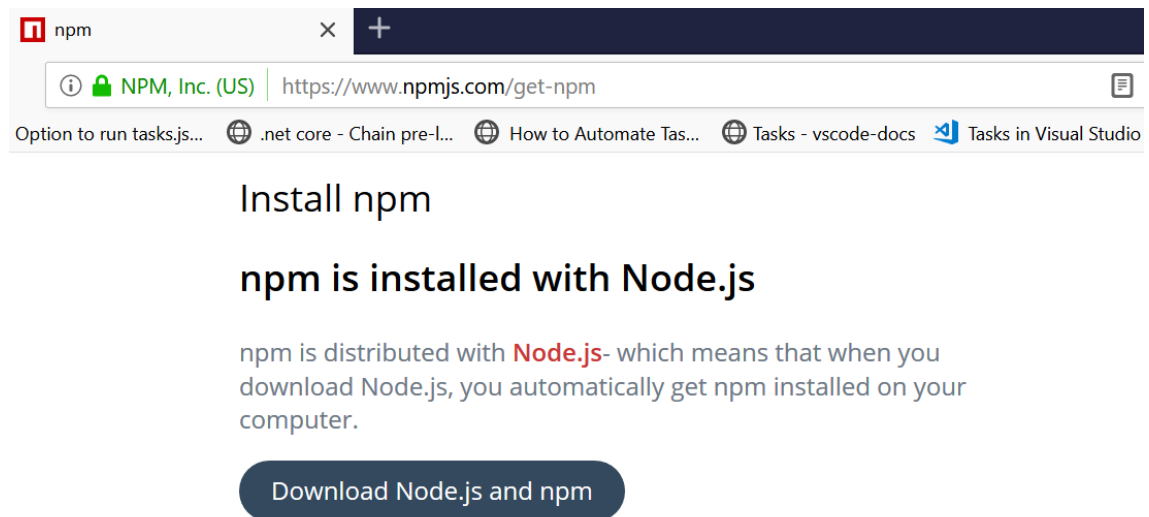


Ilustración 15. Página de instalación de npm

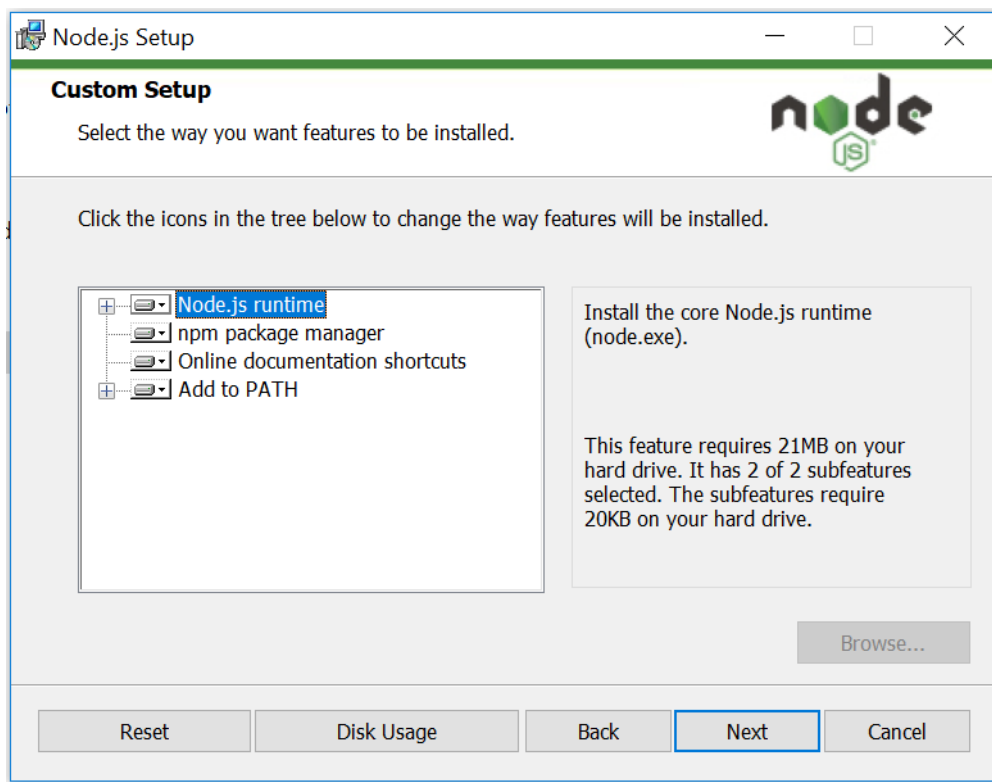


Ilustración 16. Wizard de instalación de Node

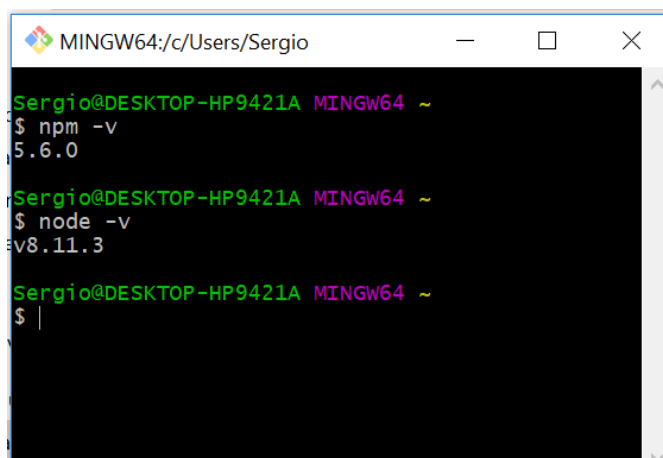


Ilustración 17. Versiones de Node y npm

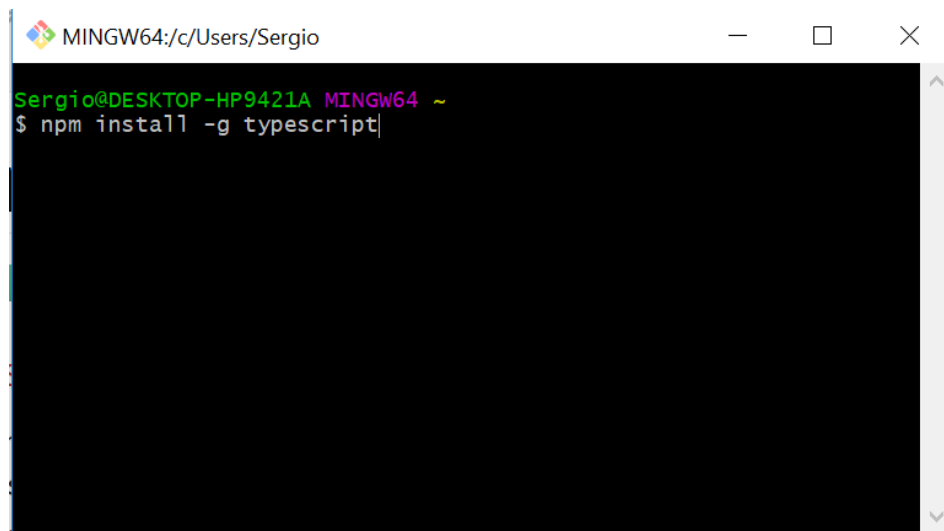
9.1.2 Instalación de Typescript

Para llevar a cabo la compilación de las clases, es necesario tener instalado de forma global el paquete Typescript. Para esto, es necesario tener instalado Node.js y npm, lo cual se encuentra explicado en el apartado anterior.

El proceso para instalar Typescript es igual al seguido para instalar cualquier paquete de npm, a diferencia de que la instalación se ha de hacer de forma global. Esto se puede llevar a cabo añadiendo una flag al comando, lo cual indica que el paquete se instalará de forma global.

Llevar a cabo una instalación de un paquete de forma global, permite la ejecución de dicho paquete en cualquier lugar del sistema en el que se instale. Su alcance no se ve reducido al directorio donde se encuentre la carpeta *node_modules* que contiene todos los módulos instalados.

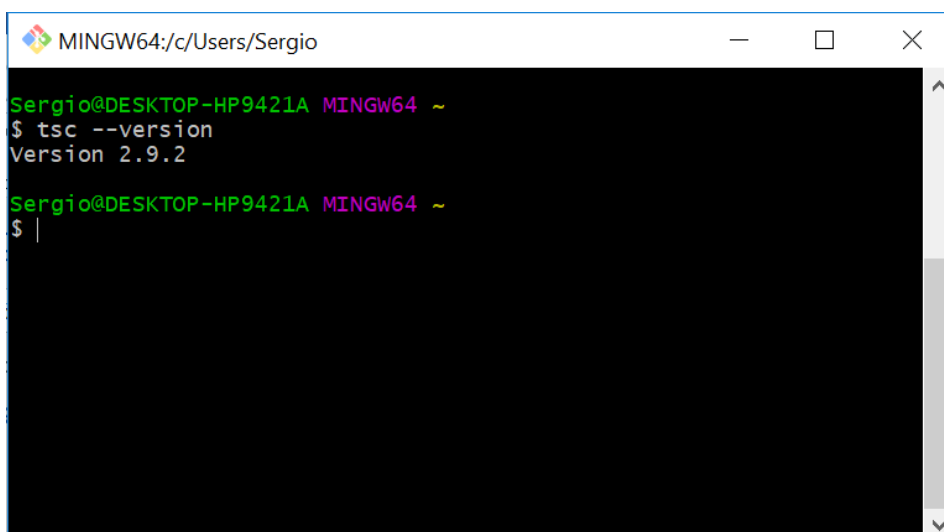
El proceso para llevar a cabo la instalación es el siguiente:



```
MINGW64:/c/Users/Sergio

Sergio@DESKTOP-HP9421A MINGW64 ~
$ npm install -g typescript
```

Ilustración 18. Comando de instalación



```
MINGW64:/c/Users/Sergio

Sergio@DESKTOP-HP9421A MINGW64 ~
$ tsc --version
Version 2.9.2

Sergio@DESKTOP-HP9421A MINGW64 ~
$ |
```

Ilustración 19. Comando de comprobación de versión

9.2 Manual de Ejecución

Una vez cumplidas las necesidades de instalación del apartado Manual de Instalación se procede a la ejecución del programa. Para llevar a cabo la ejecución se siguen los siguientes pasos:

1. Descargar el programa de su repositorio de GitHub correspondiente.
2. Localizar la carpeta que contiene el programa en el sistema utilizado.
3. Abrir la terminal (o línea de comandos) en la carpeta contenedora del proyecto.
4. Ejecutar el fichero `generar.sh` (o `.bat`) de la siguiente forma:

```
./generar.sh < param >
```

Donde *param* representa el fichero `.g4` que contiene la gramática del lenguaje que se quiere reconocer.
5. El sistema comprobará que la gramática no contiene fallos.
6. Una vez determinado que no existen fallos, el sistema pide al usuario que introduzca una extensión para los ficheros que van a ser reconocidos como ficheros de ese lenguaje.
7. El programa comprobará que la extensión introducida por el usuario no contiene errores.
8. El programa lleva a cabo la generación de todas las clases necesarias para las extensiones.
9. El sistema crea dos carpetas *extension* y *colorizer* que contienen cada una de las extensiones.
10. El usuario instalará las extensiones moviéndolas a la carpeta principal de Visual Studio Code.

9.3 Manual de Usuario

Antes de disponer del proyecto, el desarrollo de código para un lenguaje no reconocido por un lenguaje era un proceso tedioso, ya que el color del texto era monocromático, lo cual dificultaba mucho la lectura. Además de carecer de una sintaxis con color, los editores no mostraban los errores en el código, los cuales eran detectados a la hora de ejecutar dicho código. La imagen siguiente ilustra la situación actual de los editores.

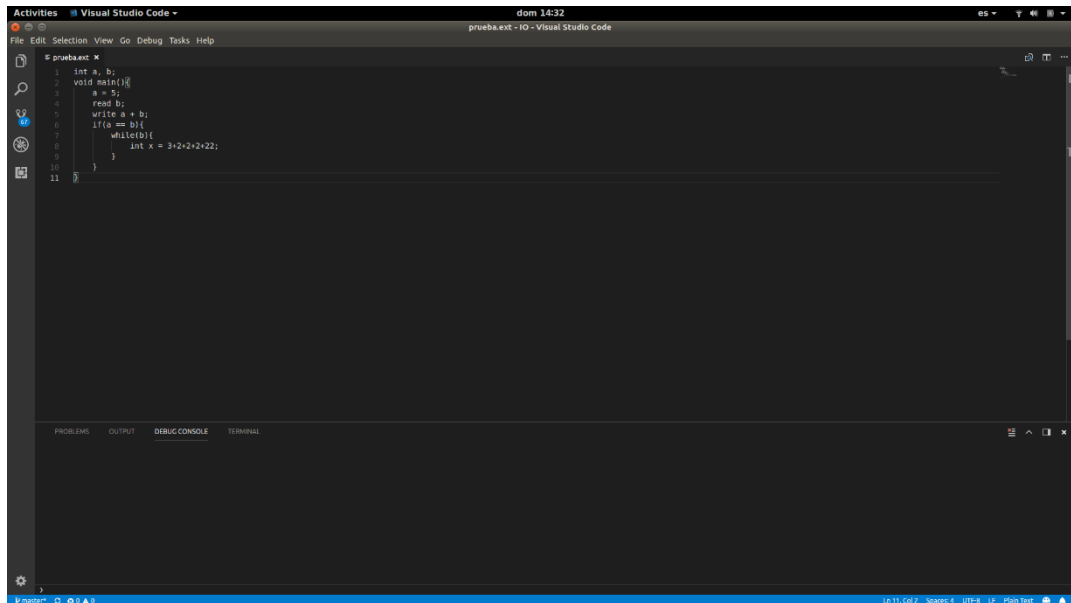


Ilustración 20. Situación inicial antes de la solución

Para poder permitir el reconocimiento de un lenguaje cualquiera, tenemos a nuestra disposición el programa que se ha desarrollado a lo largo de este proyecto. Dicho programa requiere de una gramática ANTLR4 como parámetro para poder llevar a cabo su funcionalidad de forma correcta. La siguiente imagen muestra como ejecutar el programa.

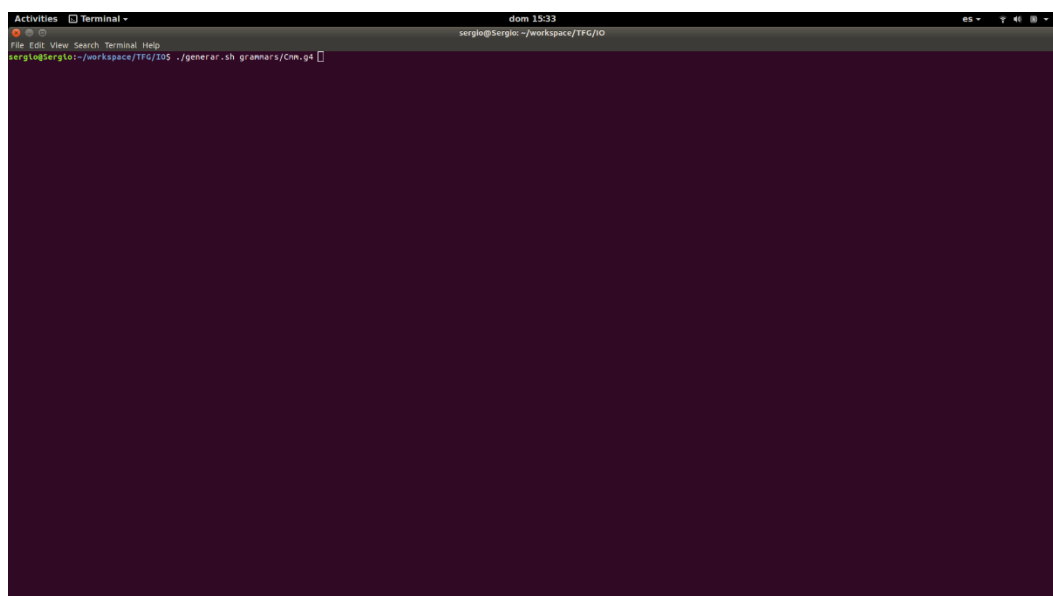


Ilustración 21. Comando para ejecutar el programa

Tras comenzar la ejecución del programa, este programa pedirá al usuario que introduzca la extensión de los ficheros que serán analizados con respecto a esta gramática.

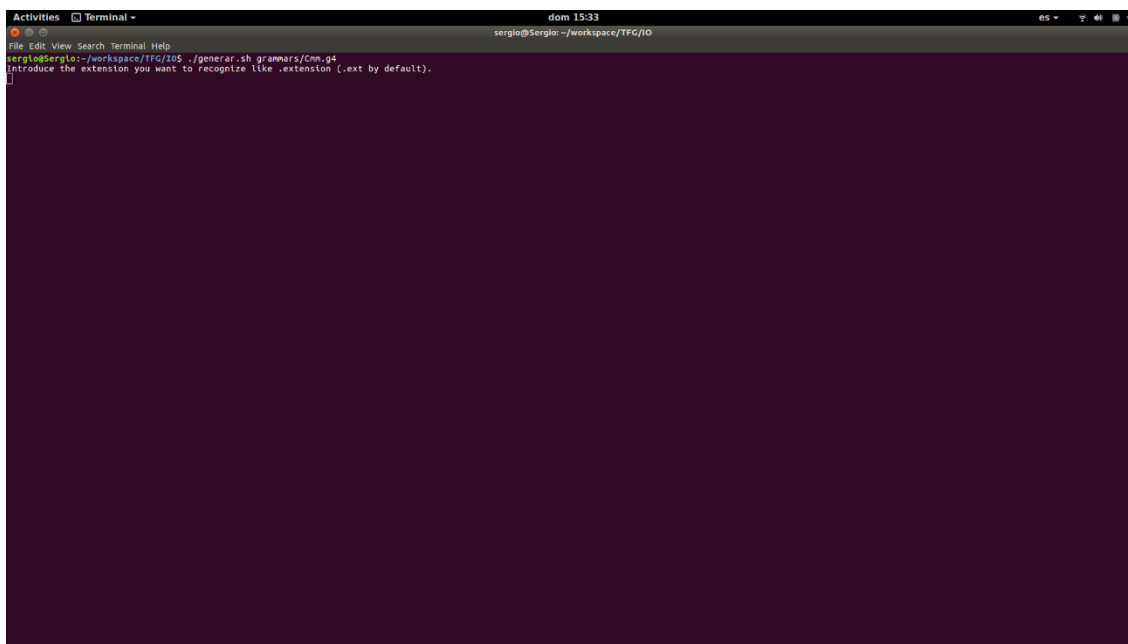


Ilustración 22. Se requiere una entrada por parte del usuario

Tras esta interacción con el usuario, el proceso de generación se llevará a cabo de forma automática y sin requerir para nada más la ayuda del usuario.

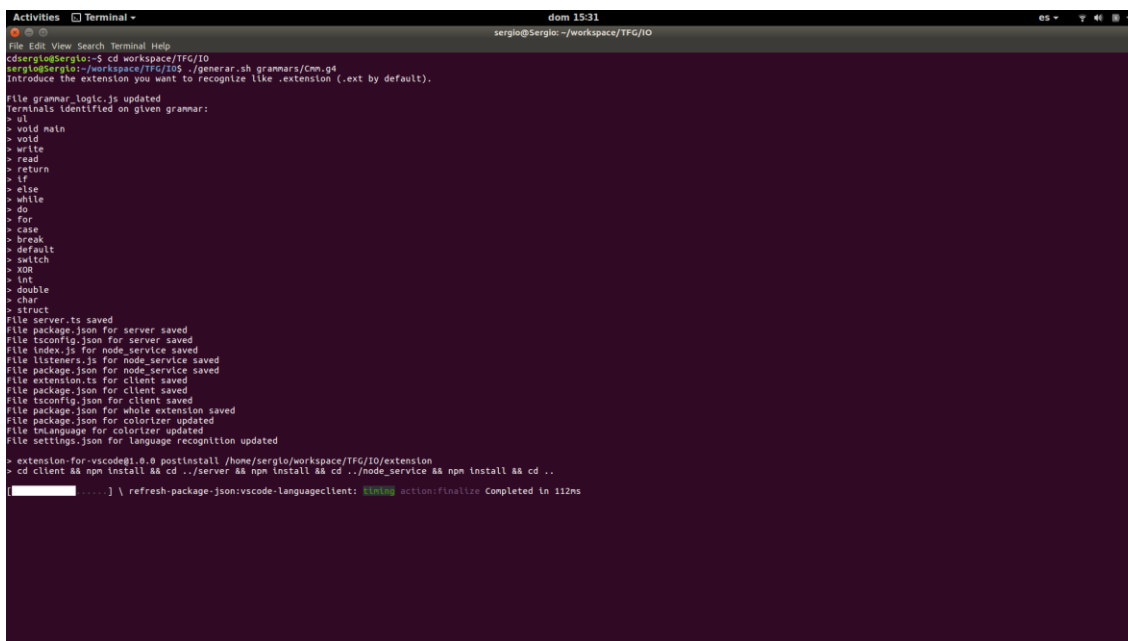
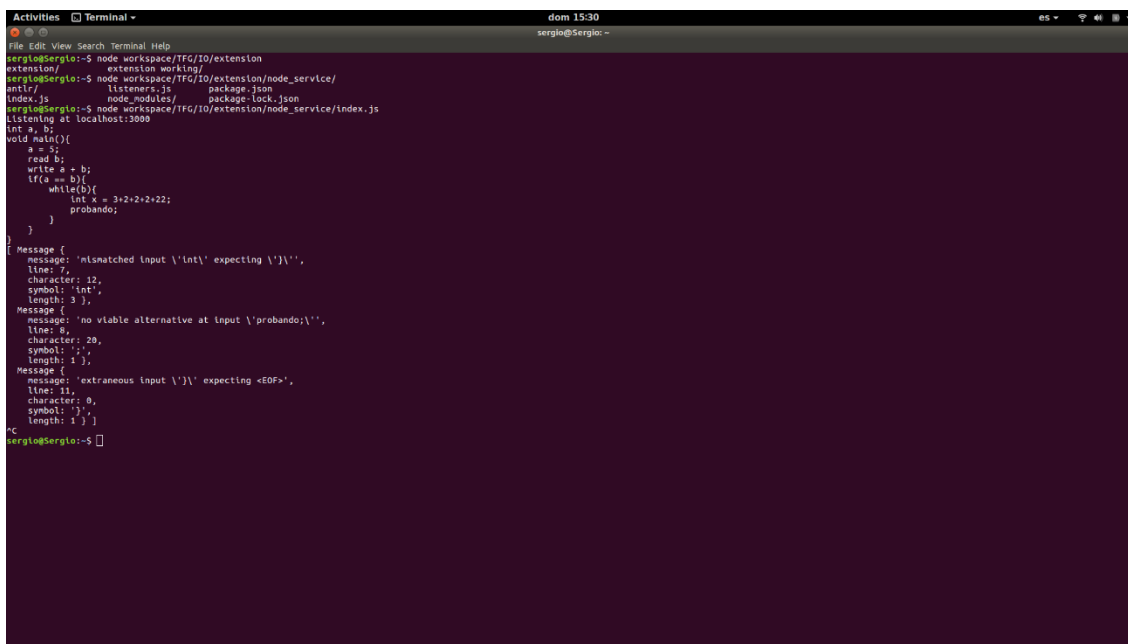


Ilustración 23. Proceso de generación de las extensiones

También podemos conectarnos al puerto en el que se encuentra desplegado el procesador de lenguaje para visualizar el contenido que recibe y los diagnósticos generados por este.



```

sergio@sergio:~$ node workspace/TFG/10/extension
extension/
sergio@sergio:~$ node workspace/TFG/10/extension/node_service/
node_modules/
sergio@sergio:~$ node workspace/TFG/10/extension/node_service/index.js
Listening at localhost:3000
int a, b;
void main(){
    a = 5;
    read b;
    write a - b;
    if(a == b){
        while(b){
            int x = 3*2+2*2+22;
            probando;
        }
    }
}
[
  Message {
    message: 'mismatched input \'int\' expecting \'\'\'',
    line: 7,
    character: 12,
    symbol: 'int',
    length: 3 },
  Message {
    message: 'no viable alternative at input \'probando;\'',
    line: 8,
    character: 20,
    symbol: ';',
    length: 1 },
  Message {
    message: 'extraneous input \'\'\' expecting <EOF>',
    line: 11,
    character: 0,
    symbol: '}',
    length: 1 } ]
]
sergio@sergio:~$
  
```

Ilustración 26. Salida proporcionada por el procesador de lenguaje

9.4 Manual del Programador

El proyecto se posee una licencia MIT, lo que permite la distribución del código por parte de cualquiera, así como su manipulación total y ampliación. De hecho, se promueve su modificación para contribuir a crear un proyecto más completo y atractivo para la gente.

Se anima a cualquier programador a incluir cualquier tipo de nueva función y a mejorar las actuales, sin embargo, se hace especial hincapié en llevar a cabo los siguientes puntos:

- **Generador de extensiones.**
 - *Optimizar la generación.* El sistema actual permite la generación de las extensiones en un rango de tiempo entre 15 y 45 segundos (sin tener en cuenta el tiempo de pregunta al usuario), dependiendo de la velocidad de internet y de la capacidad de procesamiento del sistema donde se ejecute. A pesar de que es un tiempo muy pequeño para la generación de toda una extensión que pesa alrededor de 80 MB, se debería intentar reducir el rango de tiempo para que la diferencia entre el menor y el mayor tiempo utilizado fuese el menor posible (por ejemplo ± 5 segundos).
- **Extensión LSP.**
 - *Mejorar el diseño de las clases.* Actualmente el sistema solo da soporte para la creación de una extensión para un único editor, si fuese necesaria la creación de una extensión para otro nuevo, se podría reutilizar gran parte del código. A pesar de que la estructura está adaptada a ello, se podría mejorar esta arquitectura para favorecer la jerarquía con herencias y el uso de algún patrón de diseño.
 - *Soporte para más editores.* Si se permitiese a los usuarios de la aplicación generar extensiones para más editores que VSCode, el proyecto resultaría en una aplicación realmente útil para todos, ya que no forzaría a ningún usuario a utilizar de forma obligatoria una herramienta de edición concreta, si no que podría utilizar la herramienta que considerase oportuna para desarrollar en cualquier lenguaje.
 - *Soporte para más características de LSP.* Actualmente se proporciona soporte para el resaltado de la sintaxis y una breve introducción al autocompletado y a la información flotante. Si bien es cierto que la mayor parte de las características que ofrece LSP son dependientes del propio lenguaje para el que se desarrolla, se podría ofrecer una base con contenido sencillo para las principales características del LSP.
 - *Mantener y refinar la documentación.* Una parte fundamental del desarrollo de software es la documentación del código desarrollado, lo cual facilita mucho su comprensión y posterior utilización, además de facilitar la comprensión de la estructura general del programa y la identificación de posibles fallos o patrones de diseño. Es por esto que es fundamental mantener el código del proyecto debidamente documentado.
- **Extensión de coloreado.**
 - *Mejorar la detección de los términos que serán coloreados.* Actualmente, la extensión determina los términos a los que se cambiará el color a partir de los

“literales” identificados por el Scanner generado por ANTLR a partir de la gramática introducida por el usuario. Esto es una buena aproximación para la funcionalidad, pero no es la mejor. Habría que buscar una forma de que el usuario pudiese elegir los términos o reglas a los cuales desea que se aplique un estilo de color diferente.

- *Unificar esta funcionalidad en la extensión LSP.* El equipo de Microsoft que trabaja en el protocolo LSP ha recibido gran cantidad de peticiones para que incluyan el coloreado de la sintaxis como una de las funcionalidades del LSP, aunque por el momento no tienen planteado incluirlo, lo tienen en cuenta para futuras versiones. Mientras que esto no sea posible, se podría buscar una alternativa para que en lugar de tener que instalar dos extensiones independientes, se tuviese que instalar tan solo una.

Capítulo 10. Conclusiones y Ampliaciones

y

10.1 Conclusiones

Una vez obtenida la versión final a entregar del proyecto y validadas las pruebas diseñadas, se puede decir que la solución conseguida finalmente supera con creces a la concepción inicial del proyecto. A pesar de las grandes posibilidades que ofrece el proyecto, ha sido necesario reservar una serie de futuras ampliaciones para versiones posteriores debido a falta de tiempo y soporte.

Los principales valores que me han aportado el proyecto final son los siguientes:

- El proyecto en sí, el cual satisface una necesidad importante en el campo del diseño de lenguajes de programación.
- La satisfacción personal de haber podido llevar a cabo un proyecto de tal envergadura con tecnologías hasta ahora desconocidas.
- Todos los conocimientos adquiridos durante el desarrollo del mencionado proyecto.

Las futuras ampliaciones mencionadas anteriormente se desglosan en el apartado siguiente.

10.2 Ampliaciones

Bien por falta de tiempo o de medios, ha habido funcionalidades que no se incluyen en esta primera versión del proyecto. Estas se recogen en los posteriores apartados indicando para cada una en qué consiste, cómo ampliará el sistema, qué ventajas nos aporta y por qué no se ha incluido en el sistema diseñado, entre otros aspectos.

10.2.1 Internacionalización del sistema

A pesar de que no es una ampliación que se considere urgente para mejorar la experiencia del usuario, ya que la interacción con el usuario se reduce al mínimo y es un sistema orientado a desarrolladores acostumbrados a trabajar con scripts los cuales por lo general se encuentran solo en inglés.

A pesar de lo mencionado anteriormente, añadir soporte para varios idiomas permite ampliar el público objetivo. Esta ampliación tendría mucho más sentido si se combinase con el desarrollo de una interfaz gráfica para el sistema.

10.2.2 Soporte para nuevos editores

La idea original del proyecto era que las extensiones generadas se instalasen en el entorno de desarrollo de Eclipse, finalmente se desechó esta idea por lo explicado en el apartado Planteamiento inicial de la solución no válido. A pesar de que la solución actual es mucho mejor que la original, solamente da soporte para un editor VSCode, que si bien es uno de los más utilizados por los programadores a día de hoy, restringe al usuario a utilizar una herramienta que puede no ser de su agrado.

Los principales editores para los que se proporcionará soporte, debido a su amplio rango de usuarios son Eclipse, IntelliJ, Atom, Vim y Sublime. Estos editores junto con VSCode engloban a un alto porcentaje de los desarrolladores actuales, y proporciona un gran rango de herramientas en las que desarrollar código.

La razón de no haber incluido soporte para estos editores en la primera versión es debido a la falta de tiempo principalmente, y también debido a que, a diferencia de lo que ocurre con VSCode, el soporte para estos editores se encuentra en una fase temprana de desarrollo, por tanto, no se puede asegurar que las extensiones proporcionadas carezcan de fallos importantes.

Con el apoyo de una comunidad activa como es la de GitHub, este tipo de funcionalidades sería fácilmente alcanzable en un periodo de un mes para cada editor, ya que la parte correspondiente al servidor y al procesador de lenguaje es totalmente reutilizable.

10.2.3 Interfaz gráfica para la generación de extensiones

La concepción inicial del sistema pasaba por mostrar al usuario una interfaz gráfica que le permitiese generar las extensiones de una forma mucho más visual y sencilla que directamente por el tablero de comandos.

A pesar de ser una herramienta destinada a desarrolladores principalmente, pensando a gran escala, ayuda a captar la atención de un usuario cualquiera si el programa posee una interfaz con un diseño sencillo a la vez que llamativo.

El desarrollo se llevaría a cabo como se explica en el apartado de Análisis de Interfaces de Usuario, utilizando la herramienta Electron para proporcionar un programa ejecutable con diseño web. El desarrollo de esta ampliación con sus correspondientes pruebas, se estimaría en un total de unos dos meses, ya que ya se posee la concepción inicial indicada en la etapa de análisis.

10.2.4 Soporte para otras herramientas de generación de lenguajes

No todo el mundo desarrolla lenguajes de programación utilizando ANTLR. Dependiendo del tipo de parser que desee generar el usuario, top-down (ANTLR) o bottom-up (Yacc), elegirá una herramienta u otra.

Es por esta razón que sería una buena ampliación permitir al usuario utilizar la herramienta de generación de lenguajes que considerase oportuna para poder utilizar el generador de extensiones.

Como existen varias herramientas, inicialmente la propuesta de ampliación se centraría en las principales como son Yacc, Bison o JavaCC. Más adelante, se buscaría también la posibilidad de dar soporte a otras menos conocidas.

10.2.5 Base de datos para reutilizar los procesadores de lenguaje

Una de las principales características de la extensión LSP es la posibilidad de reutilizar el componente que representa la parte del procesador de lenguaje, el cual es común a todas las implementaciones. Este componente, sin embargo, es generado para cualquier extensión independientemente del editor para el que esta sea concebida. Este comportamiento tiene un impacto negativo en el rendimiento del proceso de la generación, que podría ser reducido considerablemente si se eliminase esta etapa de desarrollo cuando exista ya el procesador de lenguaje correspondiente.

Esto se puede solucionar de una forma sencilla si se dispusiese de una base de datos que almacenase el contenido de este procesador de lenguaje evitando así la necesidad de que sea generado de cada vez. Esto puede permitir al usuario generar los clientes a partir del servidor de lenguaje elegido, lo cual aplica mucho más con la filosofía del LSP.

El procedimiento necesario para llevar a cabo esta ampliación no resultaría muy costoso, aunque por falta de tiempo y pensando en hacer el sistema lo más liviano posible, se desestimó esta opción. Sin embargo, se presenta como una posibilidad muy atractiva y factible para mejorar el sistema en un futuro. Su desarrollo no debería llevar más de 15 días de trabajo, ya que la estructura de las tablas de la base de datos no debería ser apenas compleja, al igual que tampoco lo sería la conexión con esta base de datos desde dentro del sistema.

Capítulo 11. Referencias Bibliográficas

11.1 Libros y Artículos

[Parr07] Parr, Terence. “The Definitive ANTLR Reference”. The Pragmatic Bookshelf. 2018. ISBN-10: 0-9787392-5-6; ISBN-13: 978-09787392-4-9.

[Mätzel04] Mätzel, Kai-Uwe. “Text Editors and How to Implement Your Own”. IBM OTI Labs Zurich. 2018.

[Ho03] Ho, Elwin. “Creating a Text-based Editor for Eclipse”. Hewlett Packard Company. 2018.

[Glozic01] Glozic, Dejan; McAffer, Jeff. “Mark My Words. Using markers to tell users about problems”. Sun Microsystems Inc. 2018.

[VanLiew17] Van Liew, Greg; Arnott, Andrew; Hogenson Gordon; Plubell, Brandon; Jones, Mike. “Language Server Protocol”. Microsoft Inc. 2018.

[Martínez04] Martínez Rubín Celis, Héctor. “Lenguajes de propósito general, lenguajes de Simulación y simuladores”. Instituto Tecnológico de TEPIC. 2018.

[Tomassetti17] Tomassetti, Federico. “Kanvas: Generating a Simple IDE from your ANTLR Grammar”. Federico Tomassetti. 2018.

[Tomassetti17] Tomassetti, Gabriele. “Language Server Protocol: A Language Server for DOT with Visual Studio Code”. Federico Tomassetti. 2018.

11.2 Referencias en Internet

[Wikipedia18] Wikipedia. “Cliente-Servidor”. <https://es.wikipedia.org/wiki/Cliente-servidor>.

[Wikipedia18] Wikipedia. “Domain Specific Language”. https://en.wikipedia.org/wiki/Domain-specific_language.

[PlugBee18] PlugBee. “DSLForge”. <https://dslforge.org/>.

[Wikipedia18] Wikipedia. “General purpose language”. https://en.wikipedia.org/wiki/General-purpose_language.

[Sourcegraph17] Sourcegraph. “Langserver.org”. <https://langserver.org>.

[Wikipedia18] Wikipedia. “Programación Orientada a Objetos”. https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos.

[Wikipedia18] Wikipedia. “Specification Language”. https://en.wikipedia.org/wiki/Specification_language.

[Wikipedia18] Wikipedia. “LINQ”. https://en.wikipedia.org/wiki/Language_Integrated_Query.

[Wikipedia18] Wikipedia. “Compilador”. <https://es.wikipedia.org/wiki/Compilador>.

[Wikipedia14] Wikipedia. “Código de tres direcciones”. https://es.wikipedia.org/wiki/C%C3%B3digo_de_tres_direcciones.

[Wikipedia17] Wikipedia. “Complemento (informática)”. [https://es.wikipedia.org/wiki/Complemento_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)).

[Open4U18] Open4U, BBVA. “Infografía, la historia del Open Source”. <https://bbvaopen4u.com/es/actualidad/infografia-la-historia-del-open-source>.

[Wikipedia17] Wikipedia. “Software de código abierto”. https://es.wikipedia.org/wiki/Software_de_c%C3%B3digo_abierto.

[MPCM13] MPCM Technologies LLC; JSON-RPC Google Group. “JSON-RPC Specification”. <https://www.jsonrpc.org/specification>.

[Wikipedia18] Wikipedia. “Protocolo de transferencia de hipertexto”. https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto.

[Wikipedia18] Wikipedia. “List of system quality attributes”. https://en.wikipedia.org/wiki/List_of_system_quality_attributes.

[Wikipedia18] Wikipedia. “Protocolo de transferencia de hipertexto”. https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto.

[Gutiérrez09] Gutiérrez, Demián. “UML Diagrama de Paquetes”. http://www.codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf.

[Fakhroutdinov12] Fakhroutdinov, Kirill. "UML Package Diagrams Reference". <https://www.uml-diagrams.org/package-diagrams-reference.html>.

[Wikipedia18] Wikipedia. "Language Server Protocol". https://en.wikipedia.org/wiki/Language_Server_Protocol.

[Milan17] Milan, Ish. "Aprender a programar en BASH 00: ¿por qué aprender BASH?". <https://colaboratorio.net/ishmilan/colaboratorio/2017/aprender-a-programar-en-bash-00-por-que-aprender-bash/>.

[Wikipedia18] Wikipedia. "GNU General Public Licenses". https://es.wikipedia.org/wiki/GNU_General_Public_License.

[Wikipedia18] Wikipedia. "Archivo Batch". https://es.wikipedia.org/wiki/Archivo_batch.

[Handy16] Handy, Alex. "Codenvy, Microsoft and Red Hat collaborate on Language Server Protocol". <https://sdtimes.com/che/codenvy-microsoft-red-hat-collaborate-language-server-protocol/>.

[W3Schools18] Wikipedia. "JavaScript Versions". https://www.w3schools.com/js/js_versions.asp.

[Cipsa18] Cipsa. "Lenguajes de programación más populares 2018 (Ranking Tiobe)". <https://cipsa.net/lenguajes-programacion-mas-populares-2018-ranking-tiobe/>.

[Wikipedia18] Wikipedia. "JavaScript". <https://es.wikipedia.org/wiki/JavaScript>.

[Wikipedia18] Wikipedia. "TypeScript". <https://en.wikipedia.org/wiki/TypeScript>.

[Wikipedia17] Wikipedia. "Apache License". https://es.wikipedia.org/wiki/Apache_License.

[Wikipedia18] Wikipedia. "JSON". <https://en.wikipedia.org/wiki/JSON>.

[Wikipedia18] Wikipedia. "Visual Studio Code". https://es.wikipedia.org/wiki/Visual_Studio_Code.

[Wikipedia18] Wikipedia. "Node.js". <https://es.wikipedia.org/wiki/Node.js>.

[Node.js18] Node.js Foundation. "Node.js". <https://nodejs.org/es/>.

[TutorialsPoint18] TutorialsPoint. "Node.js – Express framework". https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm.

[Microsoft18] Microsoft. "Documentation for Visual Studio Code". <https://code.visualstudio.com/docs>.

[StackOverflow2018] Stack Overflow. "Stack Overflow Developer Survey 2018". <https://insights.stackoverflow.com/survey/2018/#development-environments-and-tools>.

[Github18] Wikipedia. "Releases - npm/npm". <https://github.com/npm/npm/releases>.

[npm18] npm. "npm". <https://www.npmjs.com/get-npm>.

[Guevara18] Guevara Benites, Alexander. “¿Qué es npm?”. <https://devcode.la/blog/que-es-npm/>.

[Maruyama18] Maruyama, Rio. “ESDoc – A Good Documentation Generator for JavaScript”. <https://esdoc.org/>.

[Wikipedia18] Wikipedia. “APT (Debian)”. [https://en.wikipedia.org/wiki/APT_\(Debian\)](https://en.wikipedia.org/wiki/APT_(Debian)).

[Wikipedia18] Wikipedia. “Swing (Biblioteca gráfica)”. [https://es.wikipedia.org/wiki/Swing_\(biblioteca_gr%C3%A1fica\)](https://es.wikipedia.org/wiki/Swing_(biblioteca_gr%C3%A1fica)).

[Wikipedia18] Wikipedia. “JavaFX”. <https://es.wikipedia.org/wiki/JavaFX>.

[Lermen16] Lermen, Gustavo. “Parsers: What is the difference between YACC and ANTLR?”. <https://www.quora.com/Parsers-What-is-the-difference-between-YACC-and-ANTLR>.

Capítulo 12. Apéndices

12.1 Definiciones y acrónimos

Por orden alfabético, todos los términos que se consideren importantes en la aplicación con una descripción breve de su significado dentro de la aplicación.

- **LSP:** Acrónimo de Language Server Protocol. Protocolo creado por Microsoft y estandarizado desde 2016 para mejorar la portabilidad de las herramientas de reconocimiento de lenguajes.
- **npm:** Acrónimos de Node Package Manager. Programa utilizado para gestionar las distintas dependencias y paquetes de Node en un proyecto.
- **Framework:** es una estructura conceptual y tecnológica que contiene artefactos o módulos concretos de software que puede servir de base para la organización y desarrollo de software.
- **Parser.** Un analizador sintáctico (en inglés parser) es un programa informático que analiza una cadena de símbolos de acuerdo a las reglas de una gramática formal.
- **Scanner.** Un analizador léxico (en inglés scanner) es un programa que lleva a cabo la primera fase de un compilador. Su función consiste en recibir como entrada el código fuente de un programa (secuencia de caracteres) y producir una salida compuesta de tokens (componentes léxicos) o símbolos.
- **Token.** Un token o también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.
- **Bottom-up parsing.** Es una estrategia utilizada por el parser para analizar las reglas. Esta estrategia consiste en comenzar el análisis en los elementos con mayor nivel de detalle (hojas) e ir avanzando hasta llegar a la raíz del árbol.
- **Top-down parsing.** Es una estrategia utilizada por el parser para analizar las reglas. Es lo contrario a la estrategia anterior.

12.2 Índice Alfabético

A

ANTLR, 3, 5, 7, 9, 24, 43, 50, 72, 73, 79, 92, 97, 100, 103, 126, 131
 ANTLR4, 44, 45, 48, 49, 72, 90, 97, 121
 API, 25, 34, 37, 100
 Atom, 18, 23, 43, 128
 Autocompletado, 18, 45

B

Bash, 45, 90, 91

C

cliente, 19, 25, 26, 28, 29, 39, 44, 45, 46, 47, 48, 50, 51, 53, 54, 55, 57, 58, 60, 61, 62, 66, 67, 73, 76, 80, 85, 89, 92, 97, 100, 103, 107, 108
 Coloreado de la sintaxis, 18

D

DSL, 21, 30
 DSLForge, 23, 132

E

Eclipse, 17, 18, 19, 23, 43, 95, 102, 128, 131
 Emacs, 18
 estándar, 18, 19, 34, 89, 90, 92

F

framework, 24, 63, 133

G

generador de extensiones, 3, 46, 49, 66, 83
 Github, 21, 24, 35, 39, 43

H

HTTP, 28, 45, 51, 66, 67, 76, 81, 82, 95, 101, 103, 104

I

IDE, 17, 19, 21, 131
 IntelliJ, 17, 18, 43, 128

J

JavaScript, 18, 23, 33, 50, 63, 72, 73, 78, 92, 94, 95, 103, 104, 133, 134
 JSON, 25, 28, 45, 51, 55, 56, 66, 67, 76, 78, 80, 89, 103, 104, 132, 133
 JSON-RPC, 25, 28, 45, 51, 66, 67, 76, 80, 89, 103, 132

K

Kanvas, 24, 131

L

Language Server Protocol
 LSP, 7, 17, 19, 21, 25, 38, 89, 98, 102, 131, 133
lenguaje, 1, 3, 17, 18, 19, 21, 23, 24, 25, 26, 28, 29, 30, 32, 35, 37, 39, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 55, 56, 58, 60, 61, 62, 63, 64, 66, 67, 69, 71, 72, 73, 76, 77, 78, 80, 82, 89, 90, 91, 92, 93, 94, 95, 97, 102, 103, 104, 105, 120, 121, 124, 125, 128, 129
léxico, 18, 32, 48, 60, 69
 LSP, 5, 7, 9, 17, 18, 19, 20, 21, 23, 25, 26, 35, 37, 38, 43, 44, 45, 48, 49, 51, 55, 58, 60, 62, 64, 65, 66, 67, 69, 72, 73, 76, 78, 80, 89, 94, 98, 102, 103, 104, 123, 125, 126, 129

M

Microsoft, 17, 19, 25, 35, 38, 43, 89, 91, 94, 98, 102, 126, 131, 133

N

Node.js, 37, 38, 45, 58, 59, 78, 79, 93, 95, 100, 117, 119, 133
 npm, 27, 95, 100, 117, 119, 133, 134

O

Open Source, 3, 7, 9, 17, 19, 21, 35, 38, 43, 94, 132

P

Plugin, 34
 procesador de lenguaje, 51, 82, 104, 129

S

SDK, 19, 24, 26, 34, 67, 94
 servidor, 19, 25, 26, 28, 29, 35, 39, 43, 45, 47, 48, 50,
 51, 53, 54, 55, 56, 58, 60, 66, 67, 73, 76, 80, 81,
 82, 89, 92, 95, 100, 103, 104, 128, 129, 132
sintáctico, 18, 26, 32, 48, 60, 69, 73
 sintaxis, 3, 19, 21, 24, 30, 31, 43, 45, 48, 49, 51, 59,
 69, 72, 94, 98, 103, 121, 123, 125, 126
 Sublime, 18, 43, 128

T

tokens, 32, 43, 55, 57, 72, 77

V

Visual Studio Code, 5, 9, 44, 45, 54, 94, 98, 120, 131,
 133
 VSCode, 18, 37, 38, 76, 77, 94, 98, 100, 104, 125, 128

12.3 Enlace a GitHub con el código del sistema

<https://github.com/SergioMD15/lsp-extension-generator>