Open Modular Interpreter

Fco. Javier Bohórquez Ogalla

Índice

1. Introducción

El proyecto OMI comprende una serie de herramientas y recursos que facilitan el aprendizaje de la teoría de intérpretes y los lenguajes formales, haciendo uso de la interactividad y la documentación.

1.1. Motivación

Los cursos académicos relacionados en el estudio de los sistemas intérpretes no cuentan con la ayuda de un caso práctico que permita ver una aplicación completa de los conceptos estudiados. Tampoco existe una herramienta interactiva que ayude a ver gráficamente cómo funcionan estos sistemas.

1.2. Alcance

1.2.1. Intérprete OMI

Sistema software capaz de analizar y ejecutar otros programas escritos en un lenguaje específico.

Puede procesar código fuente mediante:

- La entrada estándar
- Fichero externo
- Una consola interactiva (prompt)
- Puerto TCP

Admite una serie de opciones que modifican el modo en el que se ejecuta, entre las que se pueden destacar:

- Ejecución como servidor.
- Ejecución como consola interactiva.
- Ejecución detallada. Produce una salida en un formato de datos estándar (JSON), que detalla el proceso llevado a cabo durante la interpretación paso a paso.

Puede ser extendido mediante módulos y configurado desde el momento que es construido, lo que permite extender el lenguaje que es interpretado y personalizar las funciones y características del propio intérprete.

1.2.2. Lenguaje OMI

Lenguaje multiparadigma de alto nivel y de propósito general, con tipado dinámico. Su sintaxis pretende ser sencilla y similar a los lenguajes de programación modernos. Contempla tipos de datos simples y compuestos, así como un conjunto de operaciones sobre estos. Permite trabajar con variables locales y globales, y procesar estructuras y sentencias que permiten controlar el flujo de ejecución tales como if, while, for, inclusión de ficheros, rotura de bloques, parada de la ejecución, saltos a etiquetas, etc.

Permite definir y realizar llamadas a funciones, que pueden hacer uso de parámetros pasados por referencia o valor, y tomar valores por defecto. Además presenta algunas características de la programación funcional (clausura de funciones, decoradores, pliegues, aplicación parcial...). También permite la definición de clases y la instanciación de estas en objetos, contemplando algunas características de la programación orientadas a objetos (métodos mágicos, duck typing, visibilidad, herencia simple...).

Presenta mecanismos que le dan capacidades reflexivas y que permiten llevar a cabo introspección de tipos. Ofrece un conjunto de funciones que permiten operar sobre fechas, procesos y ficheros.

1.3. Módulos OMI

El intérprete OMI puede extender su funcionalidad y características mediante el uso de módulos. El proyecto incluye el desarrollo de uno de estos módulos y la documentación del proceso. El módulo gettext añade funciones para la internacionalización de programas.

1.4. Biblioteca de desarrollo OMI

Incluye todos los recursos de programación necesarios para construir el intérprete. Está escrita en C++, y se puede incluir en cualquier proyecto software escrito en este lenguaje. Aunque forma parte del intérprete, puede ser instalada de forma independiente y usada para el desarrollo de módulos o cualquier otro sistema software que necesite interpretar código OMI.

1.5. Sitio web del proyecto OMI

El proyecto OMI incluye un sitio web que sirve como presentación del mismo, además de como medio de acceso a la documentación y el software desarrollado. Todas las páginas web pertenecientes al sitio contienen información relativa al proyecto y a las áreas que este ocupa. Presenta un contenido estático mantenido directamente por el desarrollador.

1.6. runTree

Herramienta online que permite escribir código OMI e interpretarlo. La herramienta describe el árbol sintáctico resultado del análisis del código fuente, y lleva a cabo la ejecución semántica del mismo paso a paso. Además muestra información de todo el proceso, incluyendo su estado interno y la entrada/salida de datos. Esta aplicación representa un cliente del intérprete OMI cuando este es ejecutado como un servidor. Se comunica de forma distribuida con el intérprete y procesa los datos devueltos por el mismo para mostrarlos al usuario de una forma gráfica y ordenada.

2. Estado del arte

Se exponen los conceptos teóricos necesarios y previos a la construcción del proyecto. Así mismo se lleva a cabo un estudio de las herramientas disponibles en el mercado y se realiza un comparativa a nivel de funcionalidades y características.

3. Planificación

3.1. Metodología

Se ha seguido una metodología iterativa e incremental. Más concretamente se ha tomado como base el proceso unificado de desarrollo de software, el cual sigue un enfoque dirigido por casos de uso y centrado en la arquitectura. El ciclo de vida sigue un enfoque en espiral, dividido en cuatro etapas: determinar objetivos, análisis de riesgos, desarrollo y planificación. Para la realización de los productos obtenidos en cada paso de la metodología se ha utilizado el lenguaje de modelado UML.

3.2. Planificación

La planificación se divide en una serie de iteraciones. Las etapas y subetapas en las que se divide cada iteración son:

- Objetivos
- Riesgos
- Desarrollo
  - Análisis
  - Diseño
  - Codificación
  - Pruebas
- Planificación

La planificación tiene como punto de partida el día 03/11/2014, día en la que se comenzó el desarrollo del proyecto. Se ha tomado una jornada laboral de 8 horas, y una semana hábil de 5 días. El proyecto recoge los diagramas de Gant correspondiente a la planificación general.

3.3. Organización

Para el desarrollo del proyecto se ha usado un equipo de un solo empleado que lleva a cabo los distintos roles implicados. Se ha utilizado un único equipo de trabajo con unas prestaciones medias. Además se han utilizado una serie de herramientas software que incluyen sistema operativo, generadores de léxicos y gramáticas, compiladores, sistemas de control de versiones, depuradores, servidor web...

3.4. Costes

Para los costes relativos a los recursos humanos se ha tomado como referencia un documento BOE publicado el sábado 30 de noviembre de 2013 por el ministerio de empleo y seguridad social. En este documento se recoge la tabla salarial según el convenio colectivo de la empresa Trevenque Sistemas de Información SL.

En el desarrollo del proyecto ha participado un único empleado con un salario anual bruto de 18120€. El tiempo de desarrollo asciende a 8 meses dando un coste total en

recursos humanos de 12.080,00€. Se ha utilizado un único equipo de prestaciones medias con un coste de 409€. Las herramientas software utilizadas presentan una licencia libre y no suponen costes adicional.

3.5. Riesgos

Se han identificado los riesgos que pueden originar un efecto negativo en el desarrollo del proyecto determinándose la probabilidad de que estos se den y el impacto que tendrían. Una vez identificados los riesgos se han definido los planes seguidos para reducir los efectos derivados estos o disminuir la probabilidad de que ocurran.

El análisis de riesgos se ha llevado a cabo en cada iteración del ciclo de desarrollo. Cada iteración a completado la información mostrada. Muchos de los riesgos indicados son comunes a todas las iteraciones realizadas.

Los riesgos se han cuantificado según el impacto que pueden ocasionar y la probabilidad de que se den. Se han organizado según su tipo, quedando divididos en:

- Riesgos tecnológicos
- Riesgos de requisitos
- Riesgos de soluciones
- Riesgos de costes, tiempos y recurso

3.6. Aseguramiento de calidad

Para asegurar una calidad aceptable del producto y los procesos llevados a cabo en el desarrollo se ha tomado como esquema el conjunto de normas ISO 9000. Se describen los procesos y actividades seguidas para asegurar la calidad, abarcando todo el desarrollo del proyecto. También se describen los criterios de aceptación o rechazo de los productos obtenidos en cada fase del desarrollo.

4. Requisitos del sistema

4.1. Situación actual

Se ha sometido a estudio el proceso de aprendizaje en áreas como la teoría de autómatas, los lenguajes formales y los intérpretes, centrándose en los cursos académicos destinados a ello y las limitaciones que estos presentan. Se llega a la conclusión de que los cursos en los que se estudia estas materias implantan la base para entender cómo funcionan y construyen estos sistemas, pero no contemplan un caso práctico que aplique estos conceptos de una forma total. Tampoco existen herramientas interactivas que ayuden a ver cómo se lleva a cabo el proceso de interpretación.

4.2. Necesidades

Se precisa de una herramienta que ayude a ver cómo se implementa y funciona un intérprete para un lenguaje de programación. Se necesita de un soporte que guarde de forma estructurada toda la documentación relativa a la construcción de un intérprete. Se precisa de una herramienta interactiva que permita al usuario entender el proceso de interpretación de una forma gráfica y didáctica.

4.3. Objetivos

Se llegará a construir un intérprete para un lenguaje de programación con características presentes en la tecnología actual. El proceso quedará correctamente documentado y se pondrá a disposición pública. El intérprete podrá ser usado de una forma interactiva y tendrá la capacidad de dar información estructurada del proceso de interpretación. Otros sistemas utilizarán esta información para detallar el proceso de una forma gráfica y flexible. Se construirá una aplicación web que aloje toda la documentación generada y desde la cual se pueda hacer uso de herramientas online que se comuniquen con el intérprete.

4.4. Requisitos

Se han descrito los requisitos funcionales del sistema, centrándose en las características del intérprete y el lenguaje subyacente, así como la web y la herramienta online que hace de cliente. Estos requisitos abarcan los modos de ejecución del intérprete y la entrada de datos, los tipos de datos, los operadores, estructuras de control, los recursos destinados a satisfacer paradigmas como el orientado a objetos o funcional, las extensiones, la distribución de la documentación, etc. En total se detallan 189 requisitos funcionales.

Se detallan los requisitos no funcionales que deben cumplir las distintas herramientas desarrolladas tales como el rendimiento (tiempo y espacio), usabilidad, accesibilidad, estabilidad, mantenibilidad y la capacidad de concurrencia.

5. Análisis del sistema

5.1. Modelo conceptual

Se ha llevado a cabo un análisis de los datos que construyen el sistema OMI y cómo estos se relacionan. Se describe el modelo conceptual de datos del sistema mediante diagramas de clases. Las clases son organizadas en paquetes para facilitar la modularidad del sistema y su entendimiento.

El proceso de interpretar consiste en tomar código fuente, procesarlo y ejecutar su significado semántico. Por tanto el modelo de datos estará constituido por entidades que guardan un significado concreto y preciso dentro del lenguaje. Estos elementos, que representan la unidad semántica mínima, son denominados nodos ejecutables, debido a que cuando son ejecutados producen el resultado semántico asociado. Muchos nodos ejecutables por si solos no presentan un resultado semántico completo, por lo que precisan de otros nodos.

El diagrama general de paquetes describe los paquetes que componen el sistema según el carácter funcional de las entidades que contienen. Un paquete podrá contener clases u otros paquetes.

El paquete "interpreter" describe las entidades que procesan el contenido fuente según el léxico y la gramática del lenguaje OMI. El objetivo es generar el árbol de nodos ejecutables correspondiente al programa. Al procesarse este árbol se aplicará la semántica que encierran las líneas de código del contenido fuente, produciéndose de esta forma la ejecución del programa.

El paquete "runNode" describe el nodo ejecutable y aquellos tipos de nodos derivados de este, que son abstractos y que serán extendidos por tipos más específicos.

El paquete "typeData" describe los nodos correspondientes a los tipos de datos básicos que pueden ser manipulados por el sistema.

El paquete "error" describe el sistema de errores y los nodos ejecutables que permiten su control.

El paquete "extensions" describe el sistema de extensiones del intérprete, el cual permite extender la funcionalidades del lenguaje de una forma dinámica. Además contiene

el modelado de extensiones concretas.

Los paquetes siguientes categorizan y agrupan nodos ejecutables según la funcionalidad que encierran y el tipo de dato sobre el que operan.

El último paquete "rtree" describe el modelo de datos correspondiente al sistema software cliente que hace uso del intérprete de forma online y representa el estado de este.

5.2. Modelo de casos de uso

En esta sección se describen los casos de uso que recogen los sistemas que conforman el proyecto OMI, centrándose en el sistema intérprete y el cliente runTree.

El sistema intérprete presenta un actor humano denominado usuario, este puede llevar a cabo casos de uso relativos a las distintas formas de entrada de código fuente. Estos casos de uso incluyen un mismo caso denominado "interpretar" que describe este proceso. El usuario también puede llevar a cabo casos de usos relativos a las distintas opciones del intérprete como ver la ayuda, cagar una extensión, etc. El sistema intérprete también presenta un actor relativo a un sistema externo que hace uso del mismo cuando este es ejecutado como servidor. El sistema externo, que hace de cliente, lleva a cabo casos de usos como iniciar una interpretación por red u obtener el paso actual en el proceso de interpretación.

El cliente runTree tiene un único actor denominado usuario, este puede realizar casos de uso relativos al envío de código para su interpretación y la resolución paso a paso del proceso. En la ejecución de estos casos de uso el sistema cliente se comunicará con el servidor para llevar a cabo el proceso de interpretación. En el cliente runTree el usuario también podrá llevar a cabo casos de usos relativos a la interfaz y la información mostrada, como ver información de nodos, limpiar la salida, ver contenido de la tabla de símbolos, etc.

5.3. Modelo de comportamiento

A partir de los casos de uso se presentan los diagramas de secuencias que modelan los eventos que los sistemas pueden recibir del usuario y los valores de retorno de estos. A partir de los diagramas de secuencia se obtiene las operaciones que los sistemas presentan, luego se describen los contratos de cada una de las operaciones.

En general los eventos que reciben el sistema intérprete son los referentes a iniciarlo con una serie de argumentos y de facilitar el contenido fuente a interpretar desde los distintos medios. El sistema producirá un valor de retorno correspondiente a la interpretación del código fuente. También se incluyen los que no se encuentran directamente relacionado con

la interpretación como ver la ayuda o cargar una extensión.

Los eventos que recibe el sistema cliente runTree del usuario son los relativos al envío de código para la interpretación y a la navegación por la descripción paso a paso del proceso. El sistema retornará la información gráfica y textual del estado interno del intérprete en la resolución de cada paso. También se incluyen diagramas de secuencia relativos a los casos para navegar por la interfaz y operar sobre el contenido fuente.

En los contratos de operaciones de los sistemas se describe los cambios en la estructura de datos interna de los mismos. En el caso del intérprete toda operación que implique una interpretación de contenido fuente creará y relacionará estructuras de datos como el analizador léxico, el analizador sintáctico, las tablas de símbolos y los nodos ejecutables correspondientes al código procesado. En el caso del cliente las operaciones crean estructuras similares debido a que este representa gráficamente el estado interno del intérprete cuando se lleva a cabo la el proceso de interpretación.

5.4. Modelo de la interfaz de usuario

Se ha llevado a cabo un análisis de la interfaz de usuario de los distintos sistemas que conforman el proyecto.

El intérprete utilizará una interfaz de consola de comandos. Toda la información de salida la presentará como cadenas de caracteres. Así mismo toda la información de entrada la tomará del teclado en formato orden y opciones.

Por otro lado el cliente runTree presentará una interfaz en la que se disponga de una descripción gráfica de todo el proceso de interpretación. Para una descripción del proceso de interpretación se precisa de la visualización del código fuente, un árbol con los nodos que encierran el significado semántico del código, las distintas tablas de símbolos que referenciarán a variables, funciones y clases, y una consola en la que se mostrará información textual.

También se presenta un diagrama de navegación en el que se expone las distintas ventanas de la web OMI.

6. Diseño del Sistema

6.1. Arquitectura física

El intérprete OMI puede funcionar en una arquitectura de escritorio, utilizándose como un comando del sistema operativo. En este caso el comando recibe el código fuente a interpretar desde distintos medios.

Por otro lado el intérprete puede funcionar en una arquitectura cliente/servidor, jugando el rol de servidor. La forma más sencilla es recibiendo el código fuente mediante un puerto TCP y procediendo a su interpretación. Sin embargo otro modo de ejecución permite además que el intérprete produzca, a partir de una petición, una salida JSON que detalla el paso actual en el proceso de interpretación. De esta forma se puede recorrer paso a paso todo el proceso de interpretación obteniéndose una estructura JSON que defina el estado del intérprete en cada paso.

6.2. Arquitectura lógica

6.2.1. Intérprete

El intérprete presenta dos niveles de organización lógica, la capa frontal que procesa la entrada del usuario y el motor que se encarga de producir el resultado semántico correspondiente.

La capa frontal se compone de un analizador léxico y un analizador sintáctico. El analizador léxico denota un lenguaje regular y se corresponde con un autómata finito. Los estados finales del autómata producen un token que se corresponderá con un elemento léxico del lenguaje como una palabra reservada, un número, un identificador, etc. Los tokens pueden guardar un valor y serán utilizados como entrada por el analizador sintáctico. El analizador sintáctico implementa una gramática libre de contexto de tipo LR(k) , utilizándose así un método de análisis ascendente. Para construir el analizador sintáctico se utiliza un autómata de pila . La capa frontal también incluye componentes para capturar la entrada del usuario y facilitar la salida que produce el sistema.

La capa motor está formada por una serie de componentes llamados nodos ejecutables, los cuales encierran un significado semántico concreto. El analizador sintáctico genera un árbol formado por estos nodos. La ejecución del árbol producirá el resultado. La ejecución de ciertos nodos generarán de forma dinámica otros nodos que serán referenciados mediante una estructura de datos denominada tabla de símbolos.

6.2.2. runTree

El cliente runTree presenta dos niveles de organización lógica, el front-end se encarga de recibir los datos de entrada del usuario, generalmente código fuente, este es pasado al back-end que se encargará de realizar una petición al servidor para que lo procese y devuelva una salida estructurada correspondiente al árbol sintáctico. El back-end procesa esta estructura y el resultado es pasado nuevamente al front-end para que sea representado en la interfaz. El usuario podrá realizar acciones para avanzar en el proceso de interpretación, estas acciones serán capturadas por la capa front-end que nuevamente se comunicará con el back-end para realizar peticiones al servidor y así obtener la descripción estructurada de cada paso del proceso de interpretación

6.3. Gramática

Se define las reglas gramáticales que describen el lenguaje OMI, para ello se hace uso del lenguaje para expresar gramáticas libres de contexto EBNF. También se incluyen diagramas de carril para cada regla.

6.4. Comunicaciones

Se describen las estructuras de datos que el intérprete OMI puede devolver cuando funciona en modo servidor y recibe una petición para iniciar una interpretación de red u obtener el paso actual dentro del proceso de interpretación. Se describen las estructuras JSON devueltas haciendo uso de JSON Schema.

6.5. Diseño de componentes

Se detalla cómo los distintos objetos interaccionan y se comunican entre sí para procesar el código fuente introducido por el usuario e interpretarlo, produciéndose así la ejecución del programa codificado. Para ello se presenta un diagrama de secuencia para la operación "interpretar código fuente". Luego se presentan diagramas de comunicación correspondientes a algunas sentencias escritas en el lenguaje.

Para interpretar código fuente el objeto OMI, correspondiente al sistema, creará un analizador sintáctico que codifica las reglas gramaticales. El analizador sintáctico se valdrá de un analizador léxico para procesar el código fuente mediante una serie de expresiones regulares que hace corresponder con tokens. El analizador sintáctico utiliza los tokens como entrada para comprobar si se cumple alguna regla gramatical. Por cada regla gramatical que se cumpla se creará un determinado nodo ejecutable, produciéndose así un árbol de

nodos ejecutables que representa el código fuente. Este árbol comenzará a recorrerse en profundidad para ejecutar el significado semántico que encierra cada nodo.

6.6. Diseño de la interfaz de usuario

Se describe el uso del comando "omi", correspondiente al intérprete, las opciones y argumentos que admite, y los modos en los que puede ser ejecutado. También se presenta el diseño de la páginas que conforma cliente runTree y la web del proyecto.

7. Construcción del sistema

Se detallan las herramientas utilizadas en el desarrollo y su versión. También se describe la estructura de ficheros en la que se distribuye el código fuente, y cómo estos se relacionan. Por último se presentan fragmentos de código significativos como la asignación de valores, el uso de variables, o las operaciones básicas.

8. Pruebas del sistema

La estrategia seguida ha sido que en cada iteración del desarrollo se realicen pruebas relativas a las funcionalidades comprendidas en la misma. Se ha tomado un enfoque de caja negra centrada en la especificación de las funciones, la entrada y la salida. Se construyen los casos de pruebas a partir de clases de equivalencia.

Se describe el entorno software y hardware utilizado para las pruebas, que abarca las especificaciones del equipo y las distintas versiones de software utilizado. Las pruebas se realizan desde la perspectiva de un único rol usuario que realiza programas con el objetivo de probar cada aspecto funcional del sistema.

Las pruebas se han realizado por niveles:

- Pruebas unitarias: Se llevan a cabo sobre cada artefacto software producido en cada iteración. Estas se centran en cada característica del lenguaje por separado.
- Pruebas de integración: Incluyen casos de prueba correspondiente a la interacción de varios artefactos. Se centran en el uso conjunto de las características del lenguaje.
- Prueba funcionales: Tienen como objetivo comprobar la funcionalidad integra del sistema. Se centran en la funcionalidad principal que es interpretar programas com-

pletos. Se incluyen varios programas desarrollados en OMI: una calculadora, un sistema de cuestionarios y un programa de IA con el que jugar al tic-tac-toe.

- Pruebas no funcionales: Se describen pruebas para medir el rendimiento en velocidad y espacio, y la seguridad.

9. Manual de implantación

Se describen la configuración del entorno hardware y software sobre el que funciona el proyecto OMI, lo que incluye el servidor web, el servidor de dominio, el propio intérprete y la aplicación web. Se detallan los requisitos previos para la implantación del sistema, el inventario de componentes y el proceso de instalación llevado. Por último se describen las pruebas de implantación llevadas a cabo.

10. Manual de usuario

Se listan las características de los sistemas software desarrollados. Luego se detallan los requisitos previos que se han de cumplir antes la instalación de los mismos. También se describe el uso de los sistemas, lo que incluye:

- Cómo obtener las herramientas del proyecto.
- Distintas formas de instalación:
  - Configuración, compilación e instalación desde código fuente
  - Instalación de binario precompilado y empaquetado.
- Intérprete OMI. Opciones y usos del comando.
- Referencias del lenguaje OMI. Recursos y usos del lenguaje.
- Extensiones del lenguaje. Carga y construcción de módulos.
- Funcionamiento cliente/servidor.
- Modo que detalla el proceso de interpretación.
- runTree. Interfaz y usos del cliente.

11. Conclusiones

Como se introdujo se ha construido un intérprete modular y completo de un lenguaje de programación de alto nivel, capaz de funcionar de forma distribuida y describir mediante estructuras de datos el proceso de interpretación paso a paso. Se ha construido un cliente online que se comunica con el intérprete para representar gráficamente el proceso de interpretación. La documentación y las herramientas generadas han sido dispuestas en una web del proyecto.

Durante el desarrollo del proyecto se han aprendido lecciones y profundizado en el estudio de materias, tales como los intérpretes y lenguajes formales, características de los lenguajes de programación actuales, la configuración de sistemas distribuidos, el desarrollo web, etc.

Como trabajo futuro se podría añadir características al lenguaje pertenecientes a los paradigmas contemplados u otros, añadir características al intérprete como la compilación justo a tiempo, un mayor detalle en la descripción del proceso de interpretación, etc.

12. Bibliografía

Referencias bibliográficas:

- Compiladores y procesadores del lenguaje (José Antonio Jiménez Millám) [Servicios publicaciones universidad de Cádiz].
- Compiladores. Principios, técnicas y herramientas 2ªed. (Alfred Aho, Ravi Sethi, Jeffrey Ullman y Monica S. Lam) [Pearson Addison Wesley].
- El lenguaje unificado de modelado (Booch, Rumbaugh y Jacobson) [Pearson Addison Wesley].

Portales webs sobre lenguajes de programación:

- isocpp.org: Estándar C++.
- cplusplus.com: C++ referencias y tutoriales.
- docs.oracle.com: Java Platform, Standard Edition.
- php.net: PHP documentación y referencias.
- python.org: Python documentación y referencias.

- ruby-lang.org: Ruby documentación y referencias.
- w3schools.com: JavaScript tutoriales y referencias W3C.
- developer.mozilla.org: JavaScript tutoriales y referencias Mozilla.
- nodejs.org: Node.js documentación y referencias.
- uam.es: Manual básico LISP.
- haskell.org: Haskell documentación y referencias.
- swi-prolog.org: Prolog documentación y referencias.
- perl.org: Perl documentación y referencias.
- scala-lang.org: Scala documentación y referencias.

Otros recursos generales:

- wikipedia.org
- stackoverflow.com

13. Licencia

Todo el software desarrollado se encuentra bajo licencia GPLv3. La documentación generada durante el desarrollo del proyecto se encuentra bajo licencia Creative Commons con libertar para compartir y adaptar, y prohibiéndose su uso comercial.