

Análisis del sistema

Fco. Javier Bohórquez Ogalla

Índice

1. Modelo conceptual	12
1.1. Intérprete	14
1.2. Nodos ejecutables	15
1.3. Tipos de datos	16
1.4. Sentencias de control de flujo	17
1.4.1. Sentencia	17
1.4.2. Sentencia vacía	17
1.4.3. include	18
1.4.4. exit	18
1.4.5. goto	18
1.4.6. if	18
1.4.7. switch	19
1.4.8. while	19
1.4.9. do...while	19
1.4.10. for	19
1.4.11. foreach	20
1.4.12. iloop	20
1.4.13. continue	20
1.4.14. break	20
1.4.15. try	21
1.4.16. throw	21

1.4.17. with	21
1.5. Definiciones	22
1.5.1. Variables	22
1.5.2. Funciones	23
1.5.3. Clases	23
1.5.4. Objetos	24
1.5.5. Listas	24
1.5.6. Pares clave/valor	24
1.5.7. Etiquetas	25
1.5.8. Definiciones globales	25
1.5.9. Generadores	25
1.6. Asignaciones	26
1.6.1. Asignación	26
1.6.2. Asignación de referencia	26
1.7. Operadores aritméticos	27
1.7.1. Suma	27
1.7.2. Diferencia	27
1.7.3. Producto	28
1.7.4. División	28
1.7.5. Potencia	28
1.7.6. Módulo	29
1.7.7. Tamaño	29
1.7.8. Incremento y asignación	29

1.7.9.	Asignación e incremento	30
1.7.10.	Decremento y asignación	30
1.7.11.	Asignación y decremento	31
1.8.	Operadores lógicos	32
1.8.1.	Or	32
1.8.2.	And	32
1.8.3.	Negación	33
1.8.4.	Igual que	33
1.8.5.	Distinto que	33
1.8.6.	Menor que	34
1.8.7.	Menor o igual que	34
1.8.8.	Mayor que	34
1.8.9.	Mayor o igual que	35
1.8.10.	Idéntico a	35
1.8.11.	No idéntico a	35
1.8.12.	Es nulo	36
1.8.13.	Vacío	36
1.9.	Operadores sobre cadenas	37
1.9.1.	Concatenación	37
1.9.2.	explode	37
1.9.3.	implode	38
1.9.4.	sprintf	38
1.9.5.	Buscar subcadena	38

1.9.6.	Buscar y remplazar	39
1.9.7.	Remplazar subcadena	39
1.9.8.	Convertir a mayúsculas	39
1.9.9.	Convertir a minúsculas	40
1.10.	Operadores sobre array	41
1.10.1.	Dividir en fragmentos	41
1.10.2.	Reducir mediante función	41
1.10.3.	Obtener último	42
1.10.4.	Obtener primero	42
1.10.5.	Insertar en posición	42
1.10.6.	Eliminar posición	42
1.10.7.	Insertar al inicio	43
1.10.8.	Insertar al final	43
1.10.9.	Eliminar del inicio	43
1.10.10.	Eliminar del final	43
1.11.	Operadores sobre expresiones regulares	44
1.11.1.	Crear expresión regular	44
1.11.2.	match	44
1.11.3.	search	45
1.12.	Conversión de tipos	46
1.12.1.	Conversión a lógico	46
1.12.2.	Conversión a entero	46
1.12.3.	Conversión a flotante	46

1.12.4. Conversión a cadena	47
1.13. Operadores de acceso	48
1.13.1. Acceso a clave	48
1.13.2. Acceso a función	48
1.13.3. Acceso a variable de entorno	49
1.14. Operadores condicionales	50
1.14.1. Ternario	50
1.14.2. Fusión de nulos	50
1.15. Operadores de entrada/salida	51
1.15.1. Salida estándar	51
1.15.2. Entrada estándar	51
1.16. Operadores informativos	52
1.16.1. Tipo de	52
1.16.2. Tamaño de	52
1.16.3. Información sobre	52
1.17. Procesos	53
1.17.1. Crear proceso	53
1.17.2. Esperar finalización	53
1.17.3. Obtener identificador	53
1.17.4. Obtener identificador padre	53
1.17.5. Ejecutar como proceso	53
1.17.6. Salir de proceso	54
1.17.7. Señal a proceso	54

1.17.8. Manejador de señales	54
1.17.9. Evaluar cadena	54
1.17.10. Ejecutar comando del sistema	54
1.18. Ficheros	55
1.18.1. Obtener un flujo a fichero	55
1.18.2. Escribir en flujo a fichero	55
1.18.3. Leer de flujo a fichero	55
1.18.4. Cambiar posición en fichero	55
1.18.5. Obtener posición en flujo a fichero	56
1.18.6. Cerrar flujo a fichero	56
1.18.7. Leer fichero	56
1.18.8. Escribir en fichero	56
1.18.9. Escribir al final de fichero	56
1.19. Fechas	57
1.19.1. Tiempo Unix	57
1.19.2. Fecha y hora con formato	57
1.19.3. sleep	57
1.20. Errores	57
1.21. Extensiones	58
1.21.1. Biblioteca GNU de internacionalización (gettext)	58
1.21.1.1. gettext	59
1.21.1.2. setlocale	59
1.21.1.3. dgettext	59

1.21.1.4. bindtextdomain	59
1.21.1.5. textdomain	59
1.22. rTree	60
2. Modelo de casos de uso	60
2.1. Actores	61
2.2. Intérprete	61
2.2.1. Interpretar entrada estándar	61
2.2.2. Interpretar fichero	62
2.2.3. Interpretar línea	63
2.2.4. Interpretar	63
2.2.5. Ver ayuda	64
2.2.6. Cargar extensión	64
2.2.7. Listar extensiones	65
2.2.8. Iniciar interpretación red	65
2.2.9. Obtener pasos interpretación red	66
2.3. runTree	67
2.3.1. Enviar código fuente	67
2.3.2. Siguiente paso	68
2.3.3. Siguiente sentencia	69
2.3.4. Activar ejecución automática	69
2.3.5. Desactivar ejecución automática	70
2.3.6. Limpiar salida	70
2.3.7. Ver información de nodo	71

2.3.8. Ver contenido de la tabla de símbolos	71
2.3.9. Guardar código fuente	72
2.3.10. Abrir código fuente	72
3. Visión general	73
4. Diagramas de secuencia del sistema	73
4.1. Intérprete	73
4.1.1. Interpretar entrada estándar	73
4.1.2. Interpretar línea	74
4.1.3. Interpretar fichero	74
4.1.4. Ver ayuda	75
4.1.5. Cargar extensión	75
4.1.6. Listar extensiones	76
4.1.7. Iniciar interpretación red	76
4.1.8. Obtener pasos interpretación red	77
4.2. runTree	77
4.2.1. Enviar código fuente	77
4.2.2. Siguiente paso	78
4.2.3. Siguiente sentencia	78
4.2.4. Activar ejecución automática	79
4.2.5. Desctivar ejecución automática	79
4.2.6. Limpiar salida	80
4.2.7. Ver infomación de nodo	80

4.2.8.	Ver contenido tabla de símbolo	81
4.2.9.	Guardar código fuente	81
4.2.10.	Abrir código fuente	82
5.	Contratos de operaciones del sistema	82
5.1.	Intérprete	82
5.1.1.	Operación inicio_interprete_stdin	82
5.1.2.	Operación inicio_interprete_linea	83
5.1.3.	Operación interpretar_cadena	84
5.1.4.	Operación interpretar_fichero	86
5.1.5.	Operación iniciar_interpretacion_red	87
5.1.6.	Operación obtener_paso_interpretacion_red	89
5.2.	runTree	90
5.2.1.	Operación enviar_codigo_fuente	90
5.2.2.	Operación siguiente_paso	91
5.2.3.	Operación siguiente_sentencia	91
5.2.4.	Operación activar_ejecucionAutomatica	92
5.2.5.	Operación desactivar_ejecucionAutomatica	93
5.2.6.	Operación limpiar_salida	93
5.2.7.	Operación ver_informacion_nodo	94
5.2.8.	Operación ver_tabla_simbolos	94
6.	Modelo de interfaz de usuario	95
6.1.	Intérprete	95

6.2. runTree	96
6.2.1. Wireframe	96
6.3. Sitio web	96
6.3.1. Wireframe	97

1. Modelo conceptual

El presente documento representa un análisis de los datos que construyen el sistema OMI y cómo estos se relacionan. El documento 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.

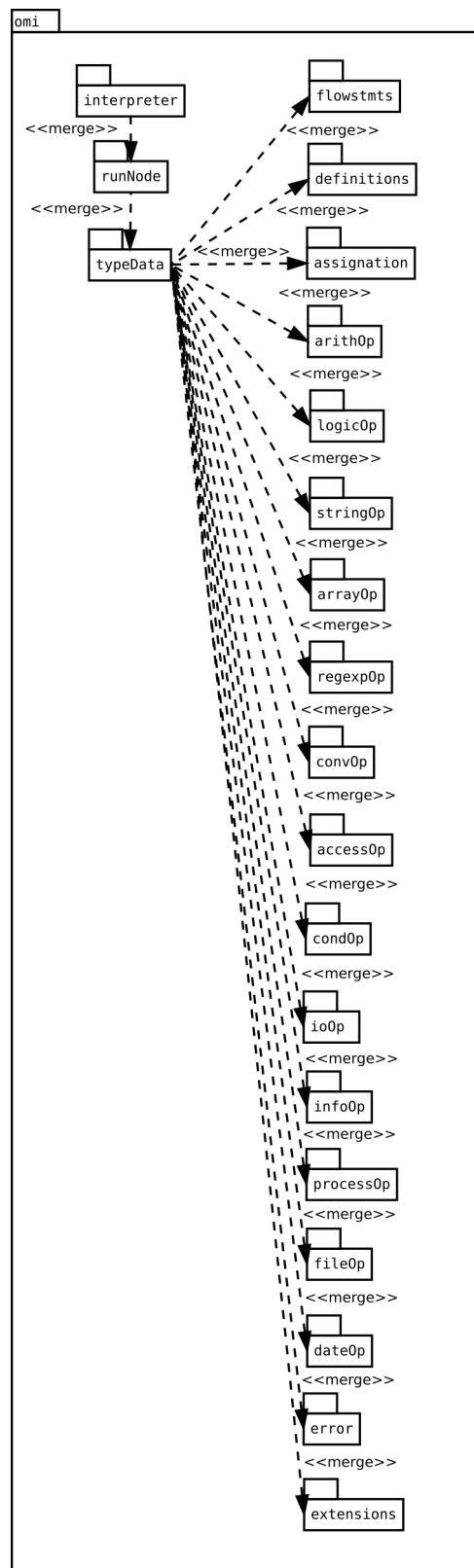
El paquete “typeData” describe los nodos correspondientes a los tipos de datos básicos que pueden ser manipulados por el sistema.

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 interprete, el cual permite extender la funcionalidades del lenguaje de una forma dinámica. Además contiene dos el modelado de dos 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. Una aplicación web que hace uso del interprete de forma online y representa el estado de este.



1.1. Intérprete

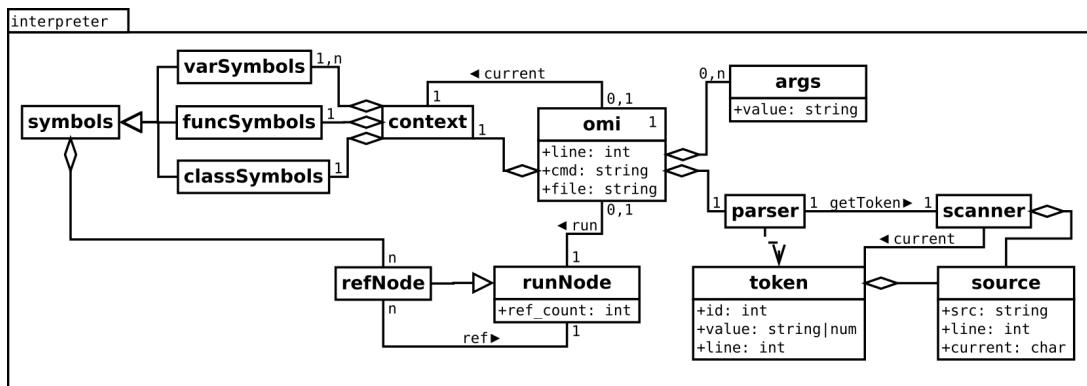
El sistema OMI se corresponde con un interprete que opera sobre un contenido fuente escrito en el lenguaje con el mismo nombre. El interprete se compone de un analizador sintáctico que encierra la gramática del lenguaje, esta es descrita a partir de una serie de tokens.

El analizador sintáctico se vale de un analizador léxico que validará y obtendrá los tokens (bajo petición) desde el código fuente. El analizador léxico debe controlar el fichero que contiene el código fuente, así como la línea y posición que se encuentra procesando en el mismo.

Los tokens obtenidos se definen por un identificador y la línea del código fuente en la que se generó, además pueden tener asociado un valor que puede ser numérico o cadena. Serán utilizados por el analizador sintáctico para determinar las reglas gramaticales que se deben aplicar y construir el árbol sintáctico correspondiente. Este árbol está formado por nodos denominados ejecutables, dado que al ser procesados en profundidad se llevará a cabo la ejecución del programa. Los nodos ejecutables dan significado semántico a cada una de las sentencias que componen el contenido fuente.

El interprete se compone además de un contexto denominado principal, que será sobre el que opere de forma predeterminada. Un contexto está formado por una serie de tablas de símbolos que serán manipuladas por ciertos nodos ejecutables cuando sean procesados. Estas tablas guardan referencias a nodos ejecutables correspondientes a símbolos variables, funciones y clases de objetos que son definidos en el código fuente. Existen determinados nodos que al ser ejecutados pueden cambiar el contexto en uso.

El interprete es ejecutado con una serie de argumentos que alteran su funcionamiento.



1.2. Nodos ejecutables

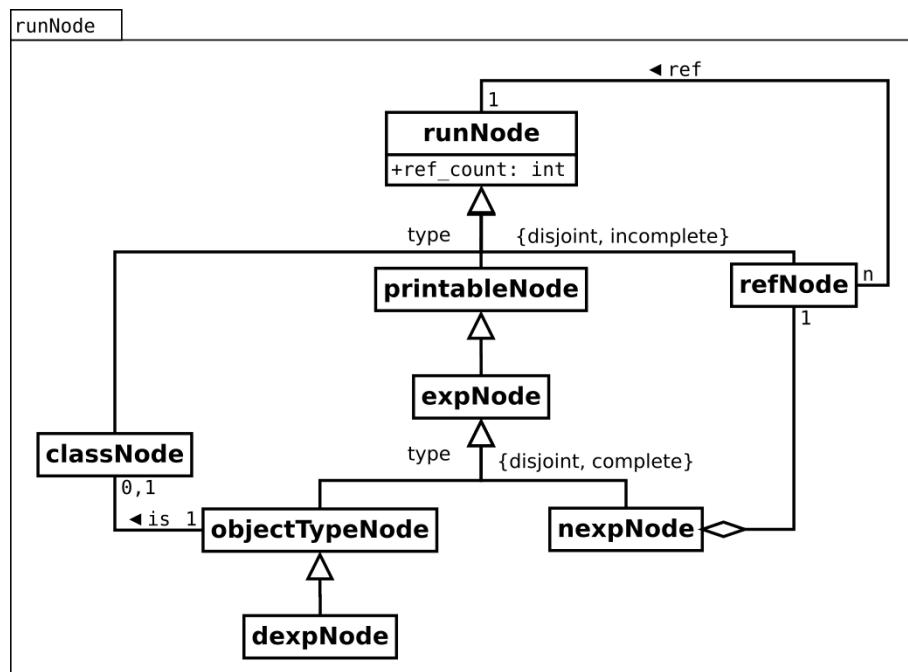
Se definen un nodo ejecutable para cada aspecto o funcionalidad que contemple el lenguaje. Cada sentencia se corresponde con un nodo ejecutable, que a su vez puede estar compuestos de otros nodos. Cada nodo ejecutable guarda el número de nodos que lo referencian para que se pueda hacer un uso óptimo del mismo.

Las expresiones son nodos ejecutables que tomarán un valor tras ser procesados. Generalmente forman parte de otros nodos correspondientes a sentencias u otras expresiones. El valor que toman pueden ser de un tipo determinado y conocido (numérico, lógico, etc), o de tipo indeterminado o no conocido hasta que el nodo es procesado.

Las expresiones de tipo determinado son extendidas por cada tipo de dato soportado por el lenguaje. Además pueden ser consideradas tipos de objetos y estar así asociadas a una clase. De esta forma toda expresión puede disponer de métodos y atributos según el tipo de dato que guarde.

Las expresiones de tipo indeterminado se componen de una referencia al nodo que guarda el valor tras la ejecución, este podrá ser una expresión de tipo determinado.

Las expresiones son nodos imprimibles lo que significa que tienen una representación gráfica asociada que puede ser volcada en la salida estándar.

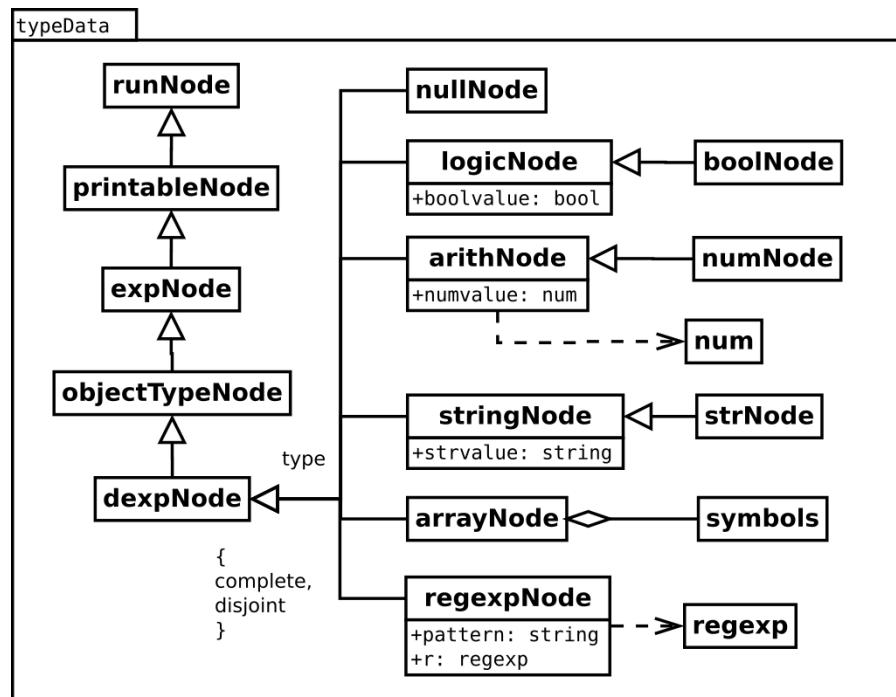


1.3. Tipos de datos

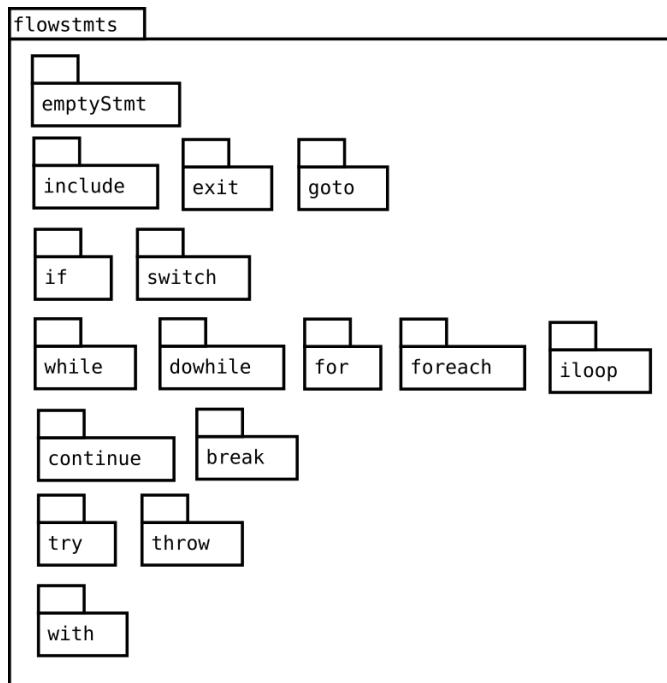
Este paquete contiene los nodos que representan expresiones con tipos de datos definidos. Se describe cada tipo de dato como un nodo con un valor asociado (en algunos casos el tipo puede comprender un único valor).

Muchos nodos son especializaciones de tipos de datos, correspondiéndose con expresiones que guardan un valor del tipo de dato al que extienden. Así por ejemplo los nodos de operaciones aritméticas generalmente extenderán al nodo del mismo tipo de dato.

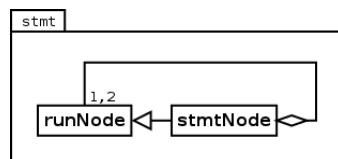
Algunos nodos de tipos datos son concretados por nodos que representan un valor constante de dicho tipo de dato.



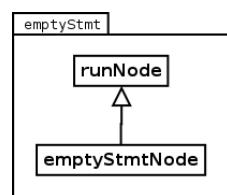
1.4. Sentencias de control de flujo



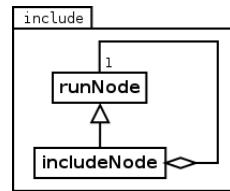
1.4.1. Sentencia



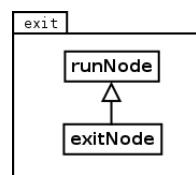
1.4.2. Sentencia vacía



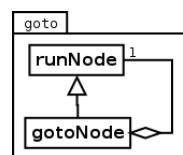
1.4.3. include



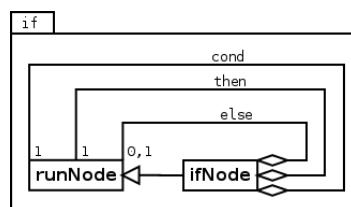
1.4.4. exit



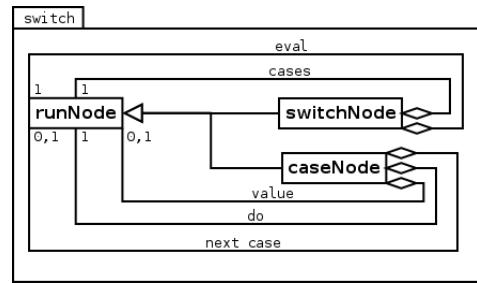
1.4.5. goto



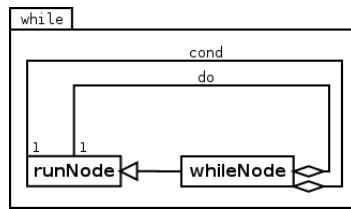
1.4.6. if



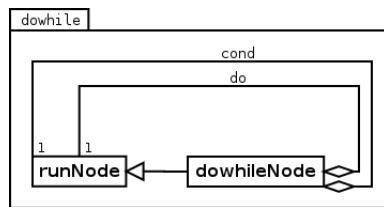
1.4.7. switch



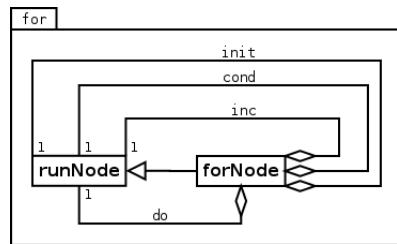
1.4.8. while



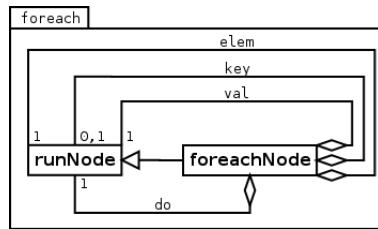
1.4.9. do...while



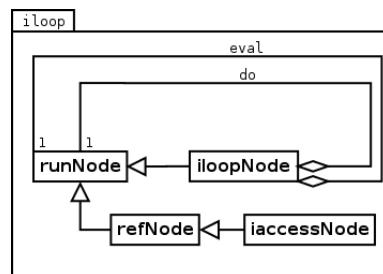
1.4.10. for



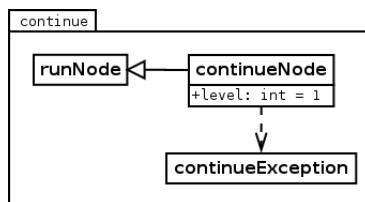
1.4.11. foreach



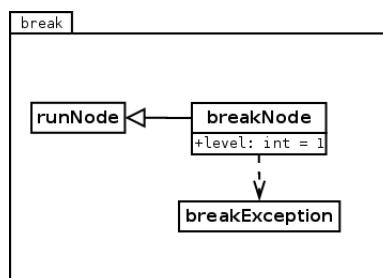
1.4.12. iloop



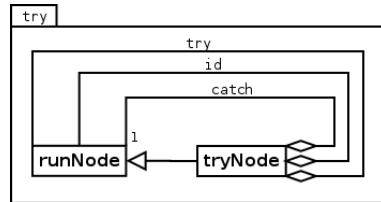
1.4.13. continue



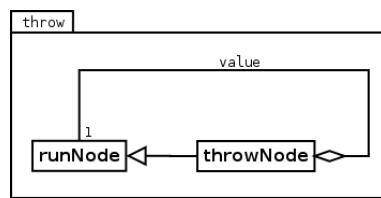
1.4.14. break



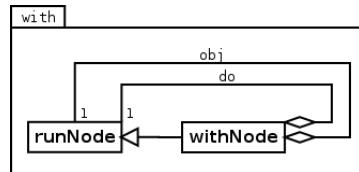
1.4.15. try



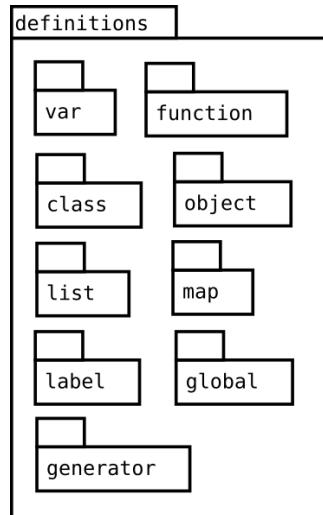
1.4.16. throw



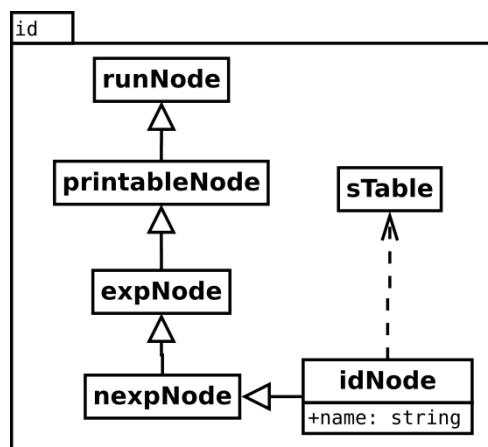
1.4.17. with



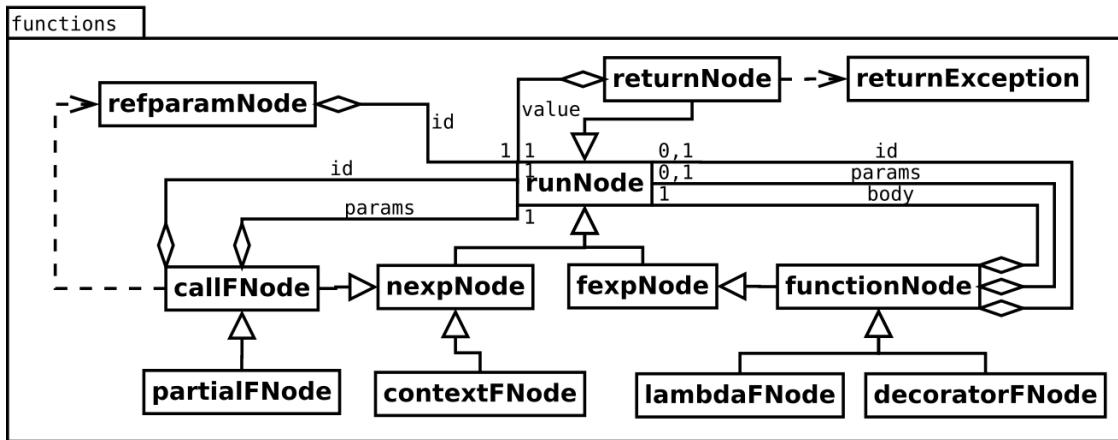
1.5. Definiciones



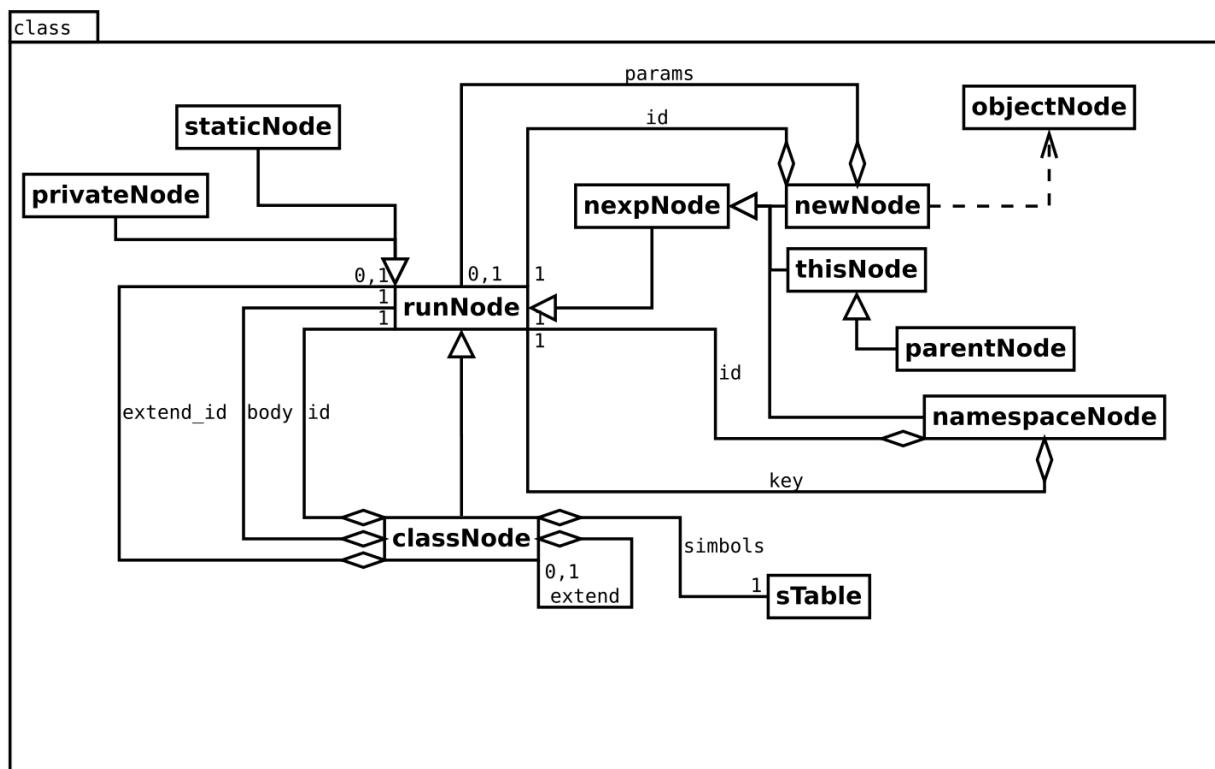
1.5.1. Variables



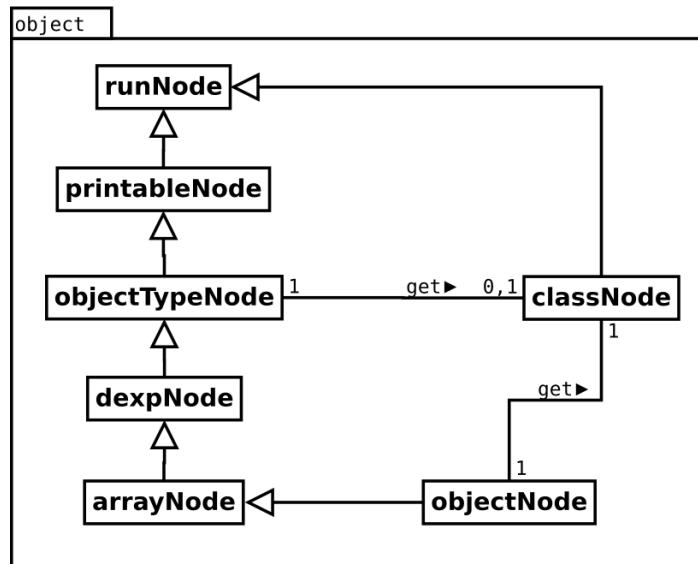
1.5.2. Funciones



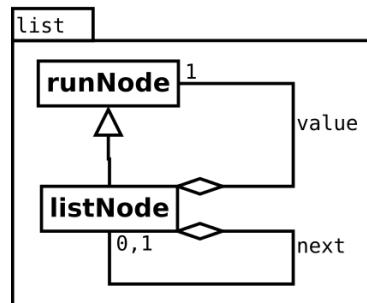
1.5.3. Clases



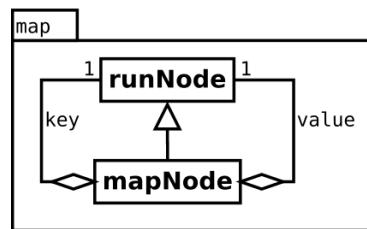
1.5.4. Objetos



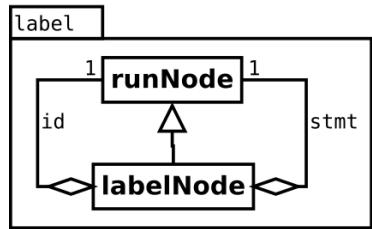
1.5.5. Listas



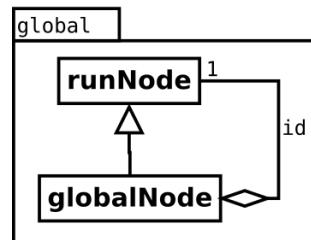
1.5.6. Pares clave/valor



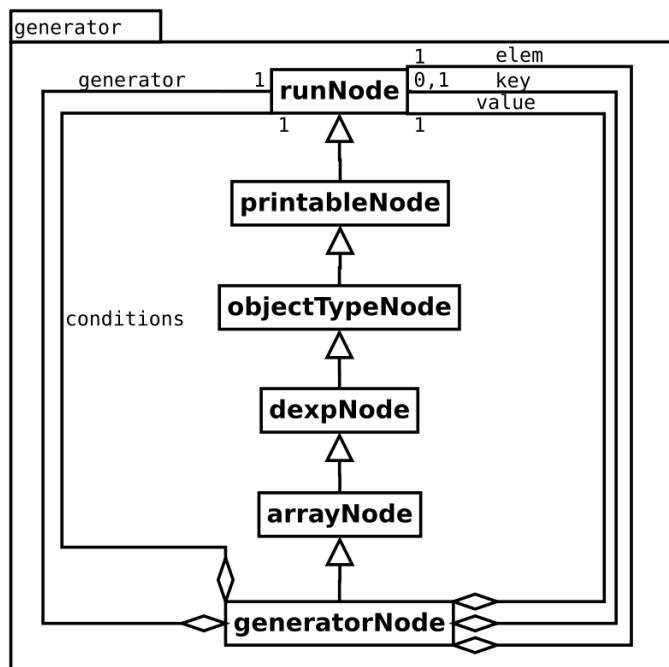
1.5.7. Etiquetas



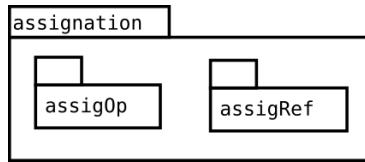
1.5.8. Definiciones globales



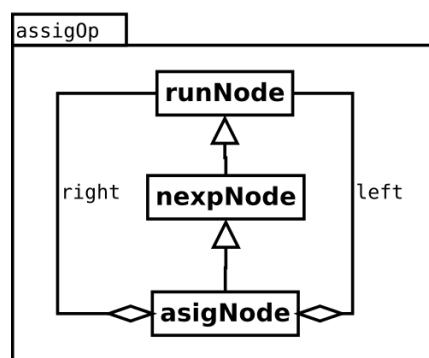
1.5.9. Generadores



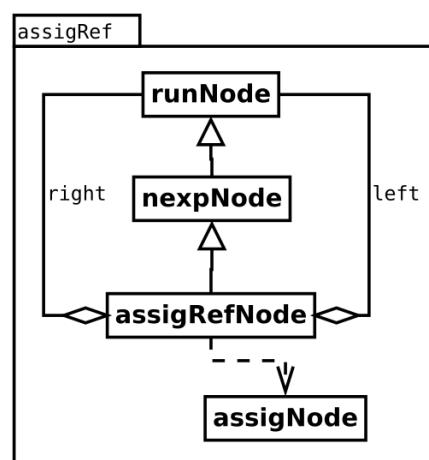
1.6. Asignaciones



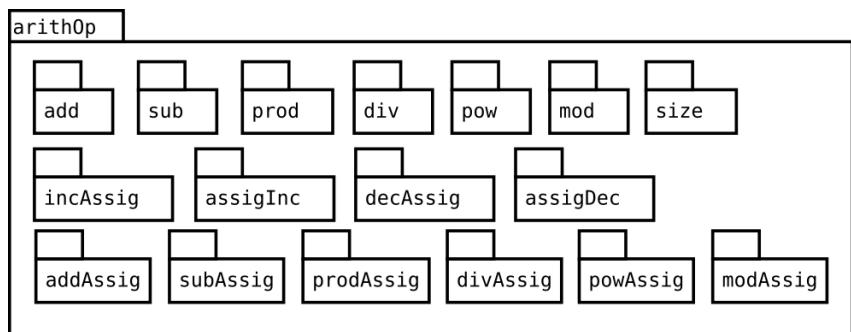
1.6.1. Asignación



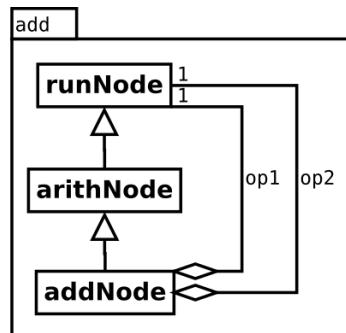
1.6.2. Asignación de referencia



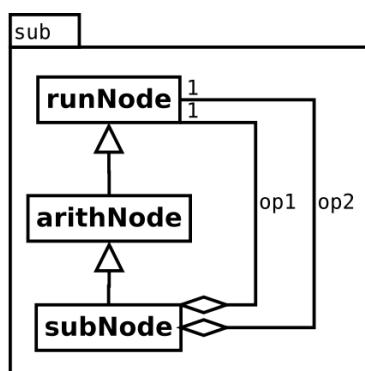
1.7. Operadores aritméticos



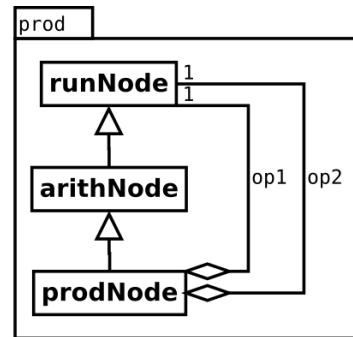
1.7.1. Suma



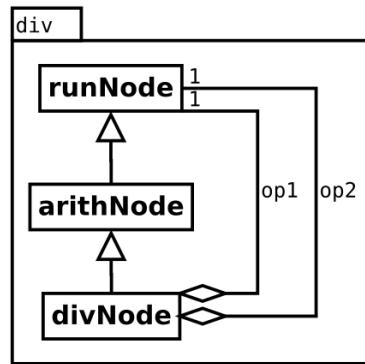
1.7.2. Diferencia



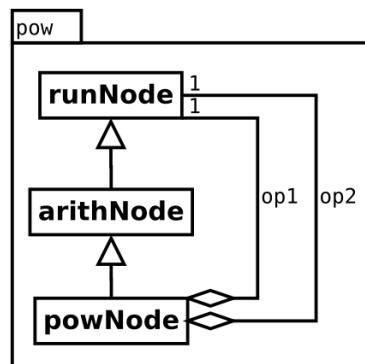
1.7.3. Producto



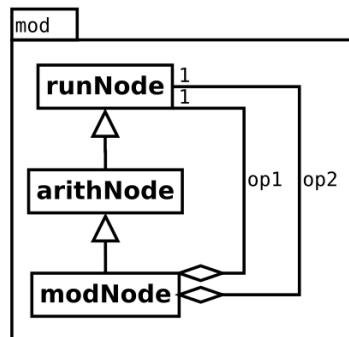
1.7.4. División



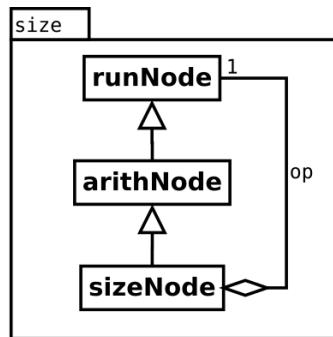
1.7.5. Potencia



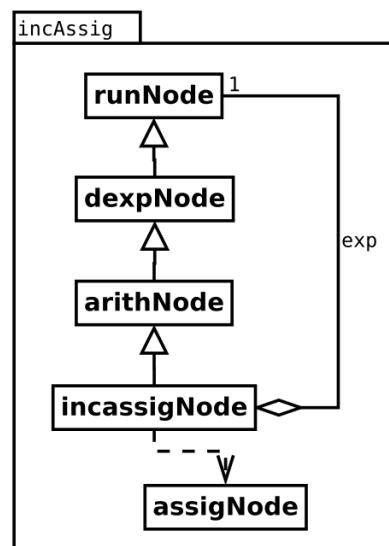
1.7.6. Módulo



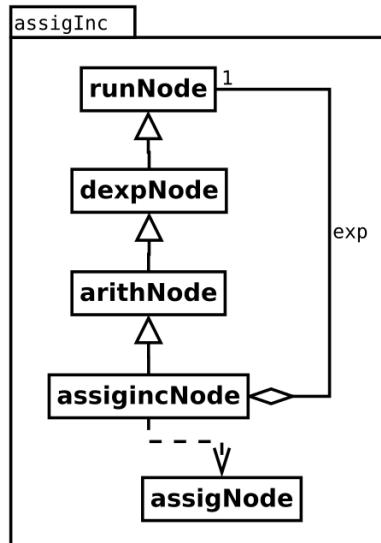
1.7.7. Tamaño



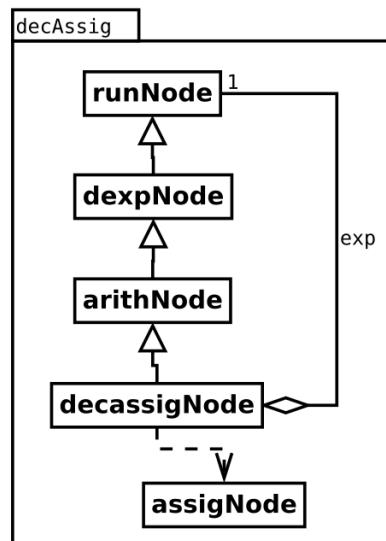
1.7.8. Incremento y asignación



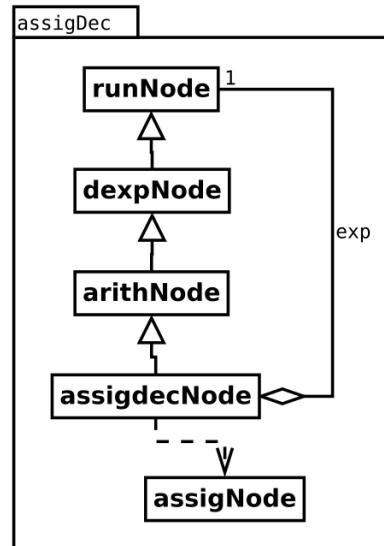
1.7.9. Asignación e incremento



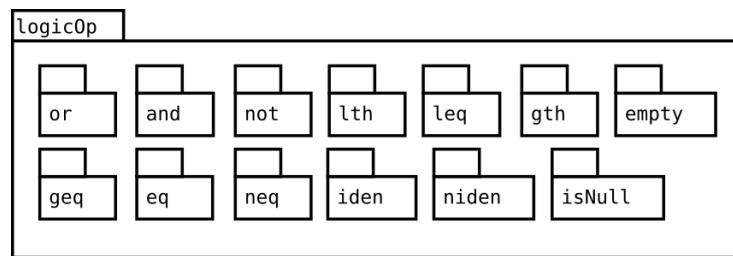
1.7.10. Decremento y asignación



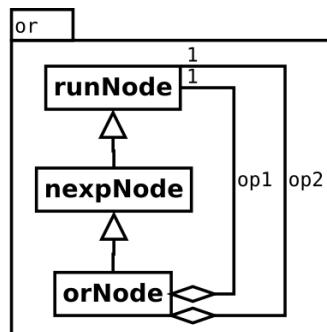
1.7.11. Asignación y decremento



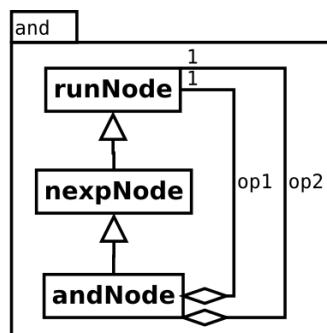
1.8. Operadores lógicos



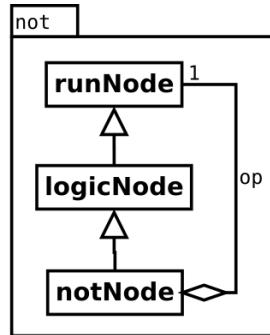
1.8.1. Or



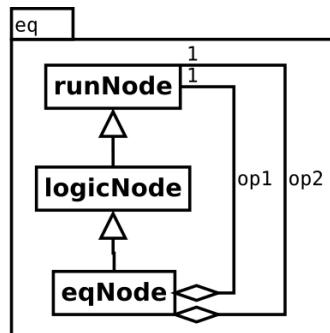
1.8.2. And



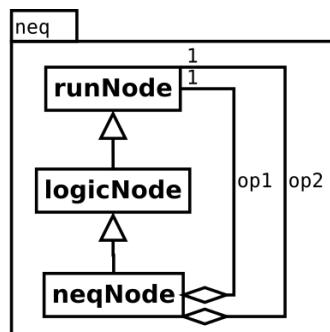
1.8.3. Negación



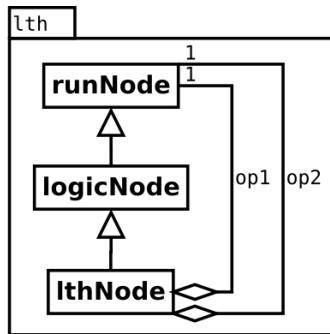
1.8.4. Igual que



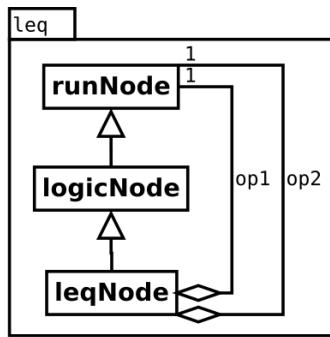
1.8.5. Distinto que



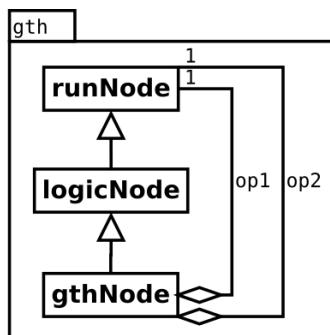
1.8.6. Menor que



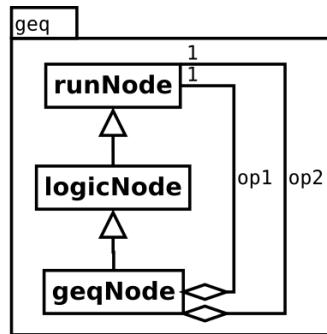
1.8.7. Menor o igual que



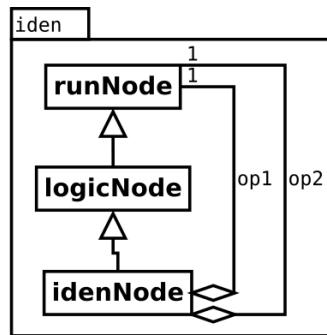
1.8.8. Mayor que



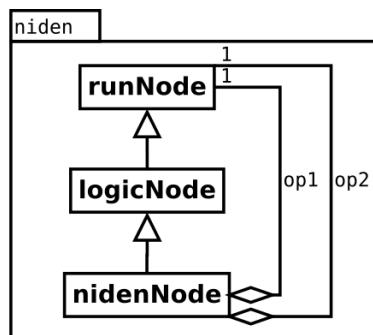
1.8.9. Mayor o igual que



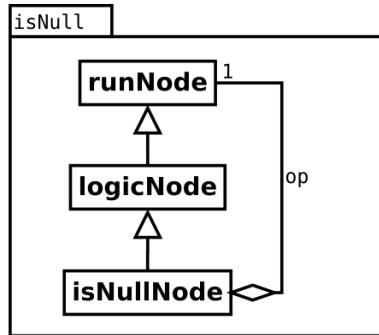
1.8.10. Idéntico a



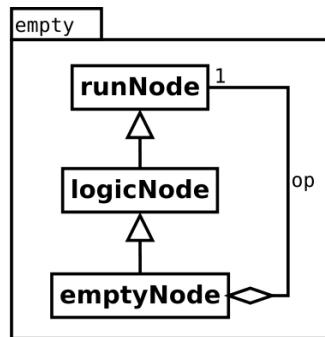
1.8.11. No idéntico a



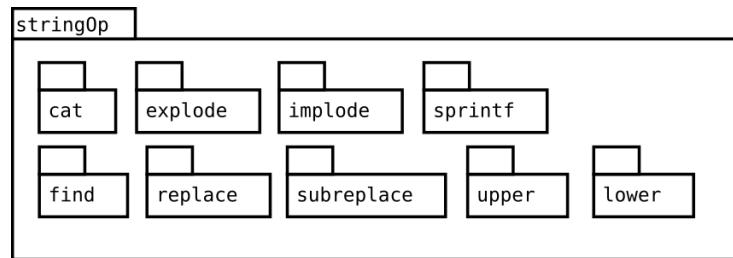
1.8.12. Es nulo



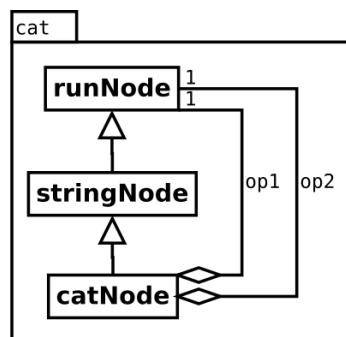
1.8.13. Vacío



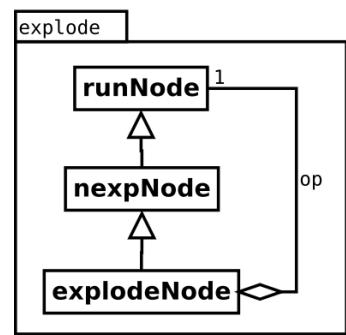
1.9. Operadores sobre cadenas



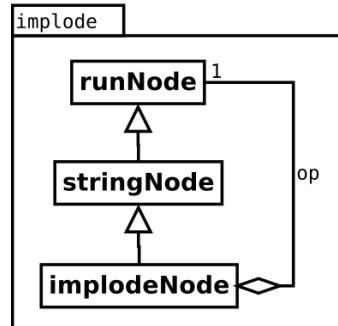
1.9.1. Concatenación



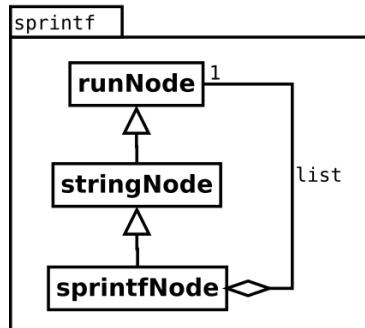
1.9.2. explode



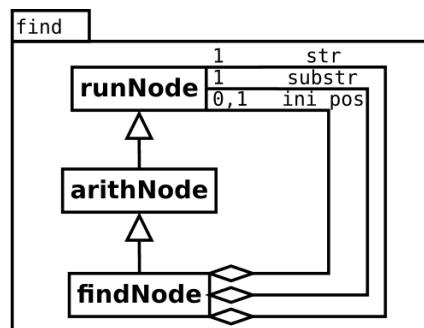
1.9.3. implode



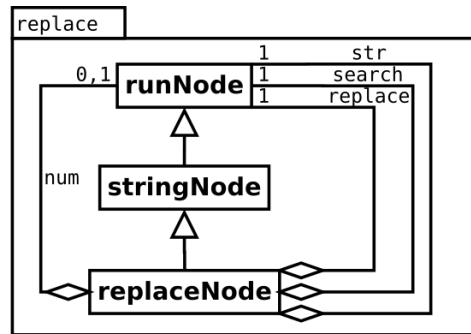
1.9.4. sprintf



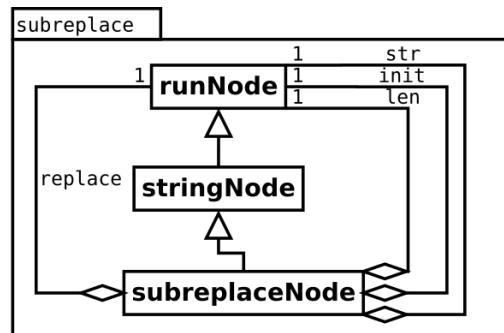
1.9.5. Buscar subcadena



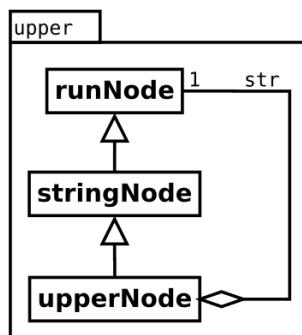
1.9.6. Buscar y remplazar



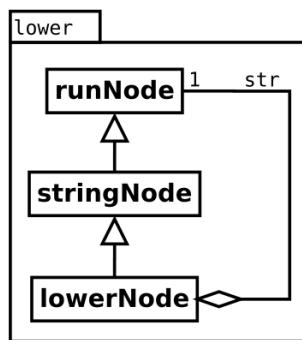
1.9.7. Reemplazar subcadena



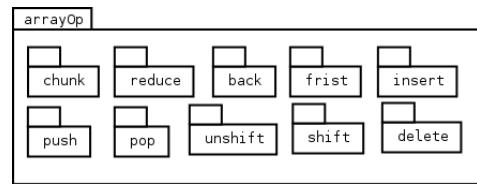
1.9.8. Convertir a mayúsculas



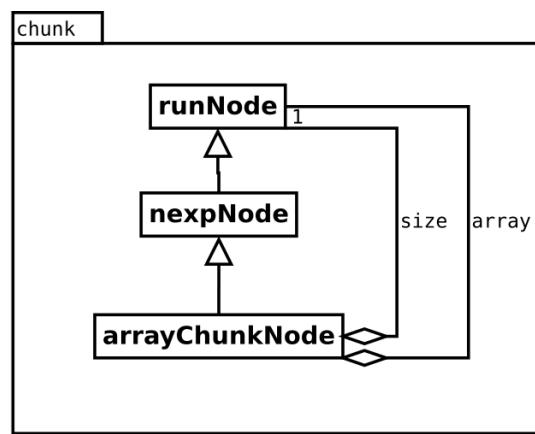
1.9.9. Convertir a minúsculas



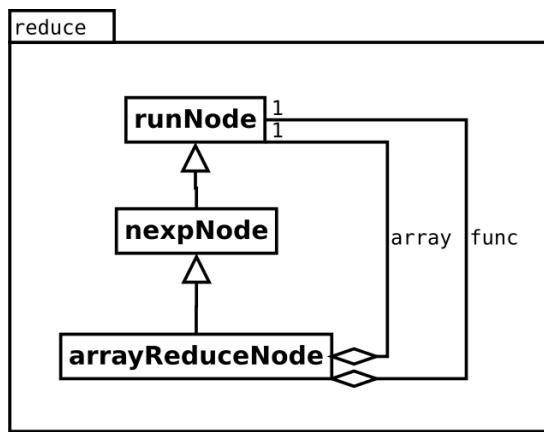
1.10. Operadores sobre array



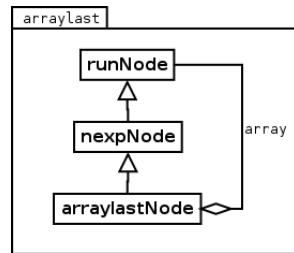
1.10.1. Dividir en fragmentos



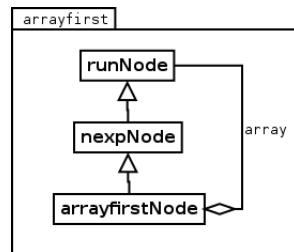
1.10.2. Reducir mediante función



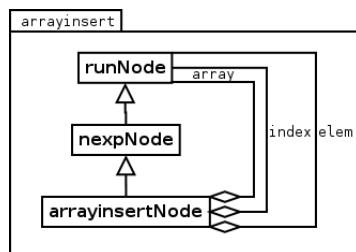
1.10.3. Obtener último



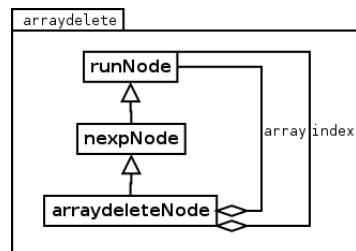
1.10.4. Obtener primero



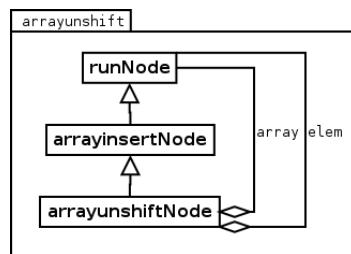
1.10.5. Insertar en posición



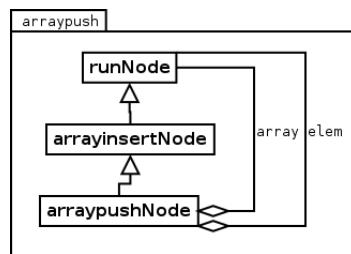
1.10.6. Eliminar posición



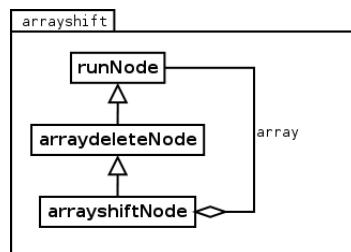
1.10.7. Insertar al inicio



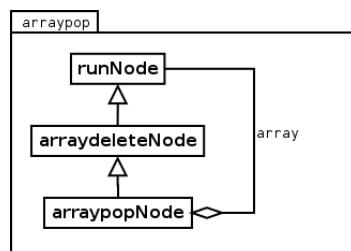
1.10.8. Insertar al final



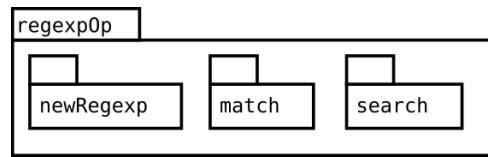
1.10.9. Eliminar del inicio



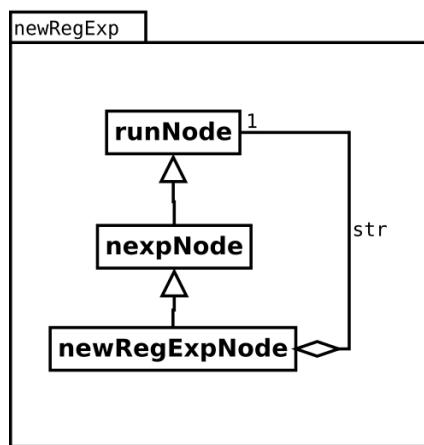
1.10.10. Eliminar del final



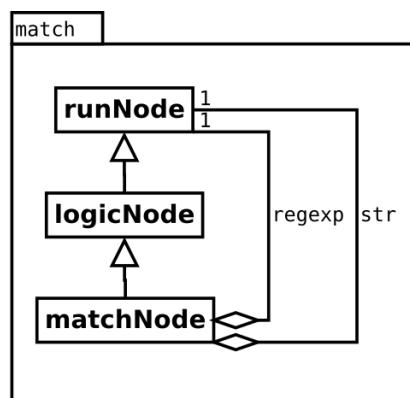
1.11. Operadores sobre expresiones regulares



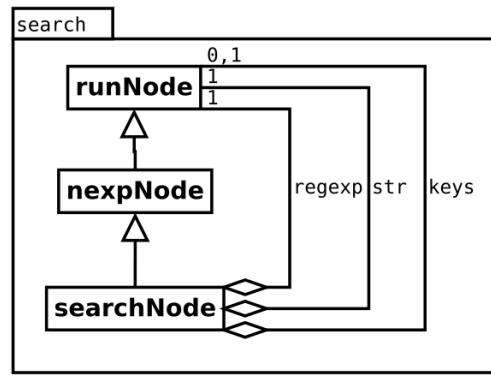
1.11.1. Crear expresión regular



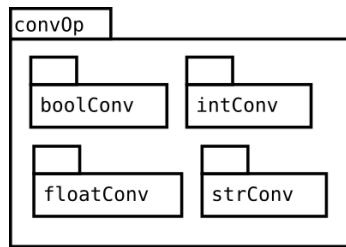
1.11.2. match



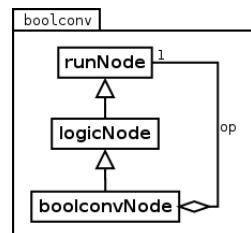
1.11.3. search



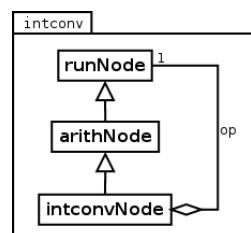
1.12. Conversión de tipos



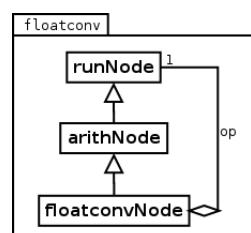
1.12.1. Conversión a lógico



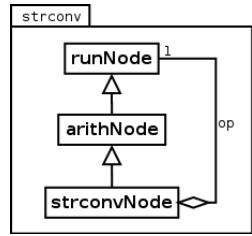
1.12.2. Conversión a entero



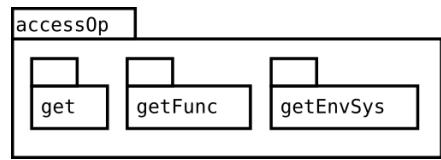
1.12.3. Conversión a flotante



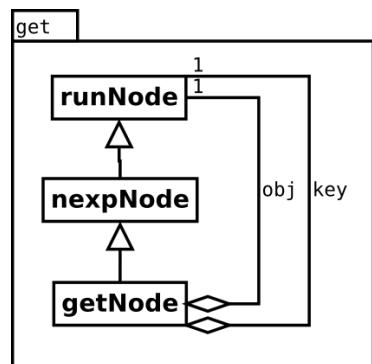
1.12.4. Conversión a cadena



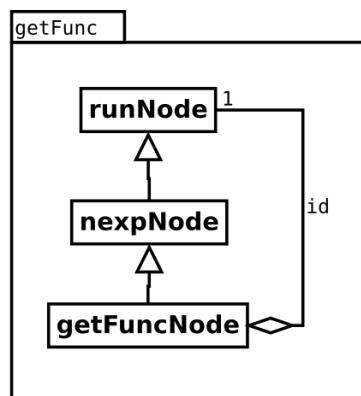
1.13. Operadores de acceso



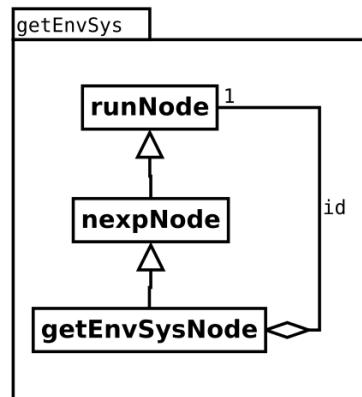
1.13.1. Acceso a clave



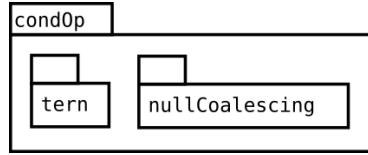
1.13.2. Acceso a función



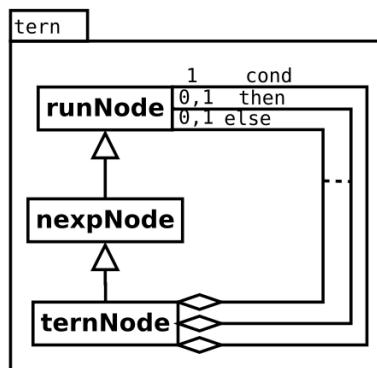
1.13.3. Acceso a variable de entorno



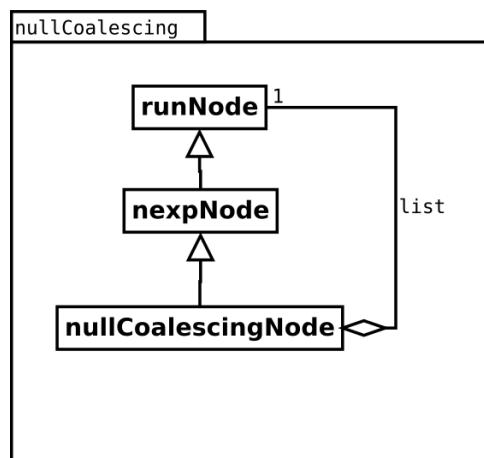
1.14. Operadores condicionales



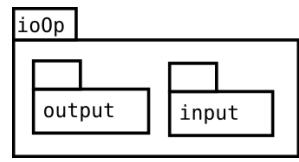
1.14.1. Ternario



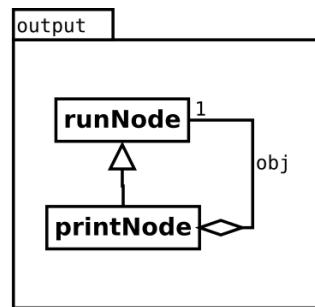
1.14.2. Fusión de nulos



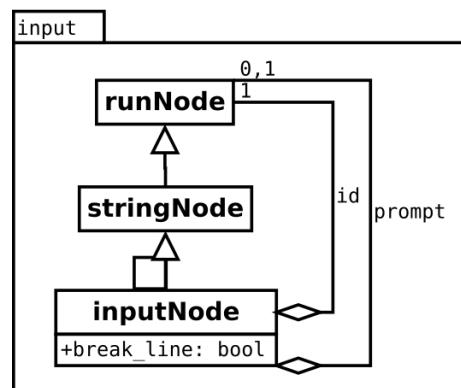
1.15. Operadores de entrada/salida



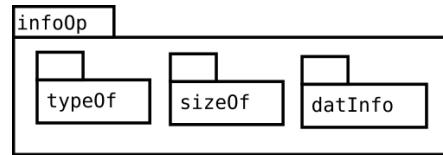
1.15.1. Salida estándar



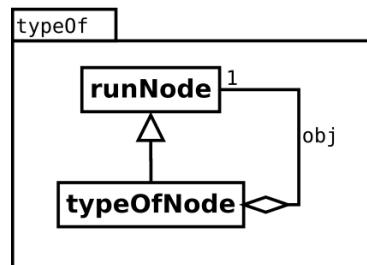
1.15.2. Entrada estándar



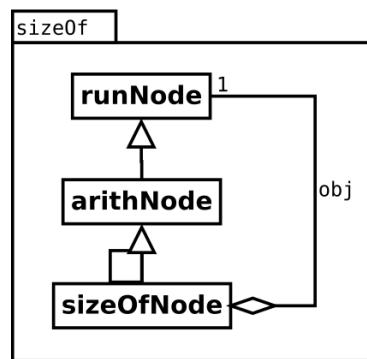
1.16. Operadores informativos



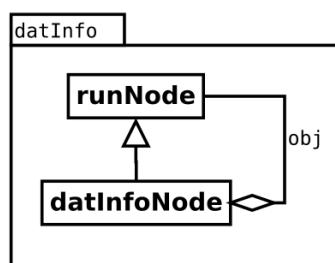
1.16.1. Tipo de



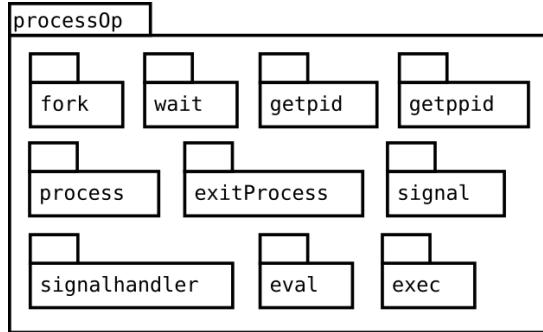
1.16.2. Tamaño de



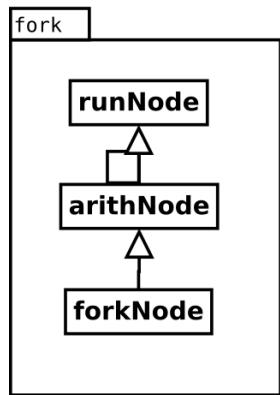
1.16.3. Información sobre



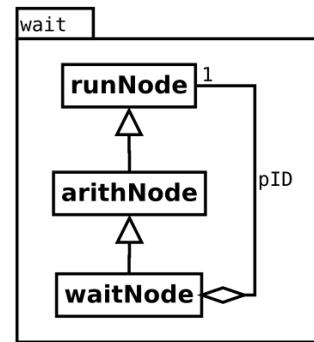
1.17. Procesos



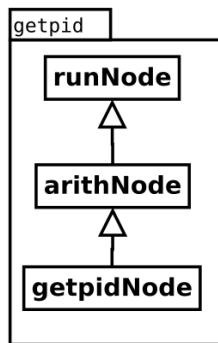
1.17.1. Crear proceso



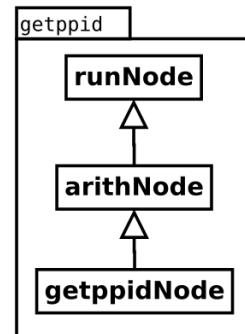
1.17.2. Esperar finalización



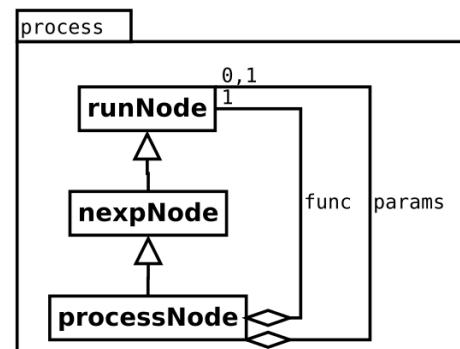
1.17.3. Obtener identificador



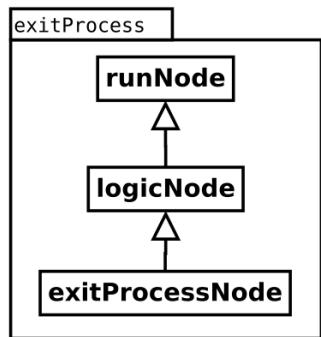
1.17.4. Obtener identificador padre



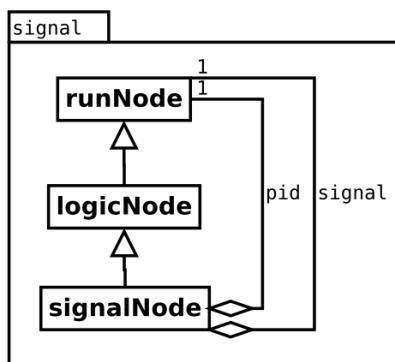
1.17.5. Ejecutar como proceso



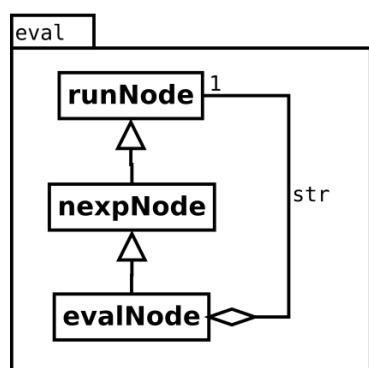
1.17.6. Salir de proceso



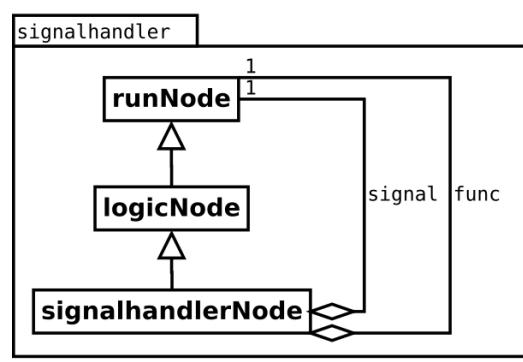
1.17.7. Señal a proceso



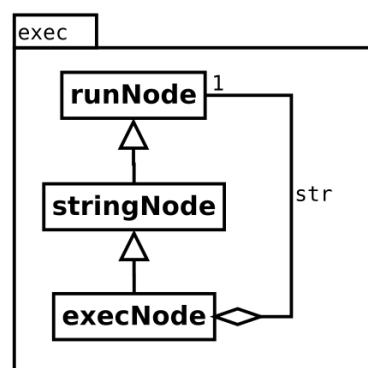
1.17.9. Evaluar cadena



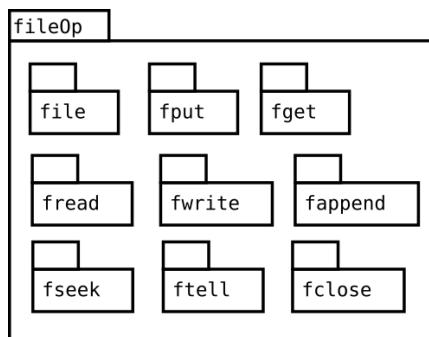
1.17.8. Manejador de señales



1.17.10. Ejecutar comando del sistema

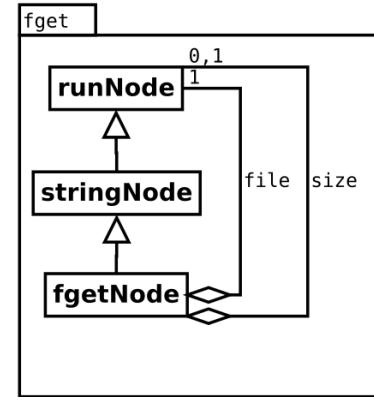
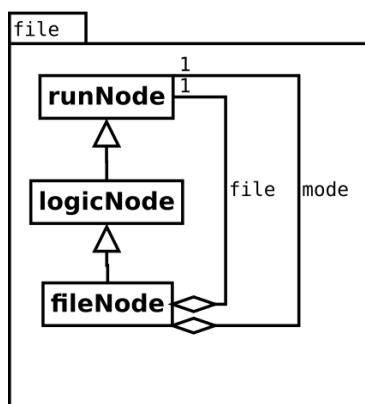


1.18. Ficheros



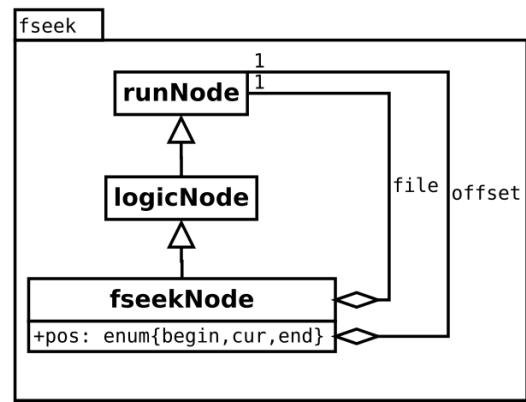
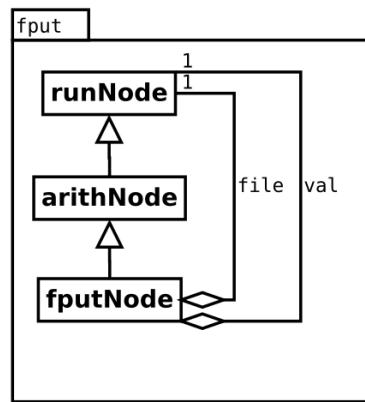
1.18.1. Obtener un flujo a fichero

1.18.3. Leer de flujo a fichero

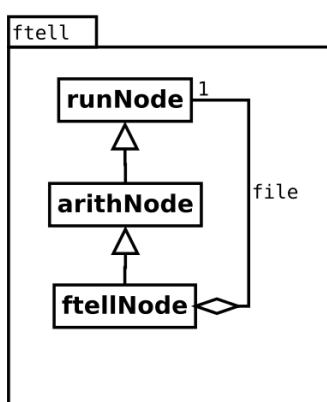


1.18.2. Escribir en flujo a fichero

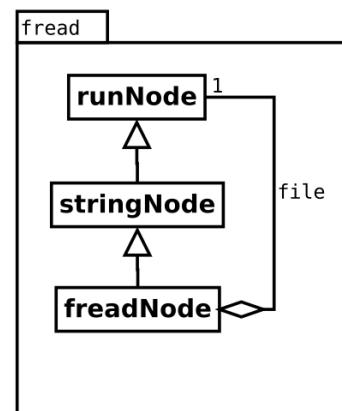
1.18.4. Cambiar posición en fichero



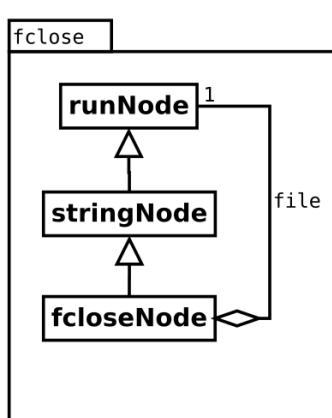
1.18.5. Obtener posición en flujo a fi-



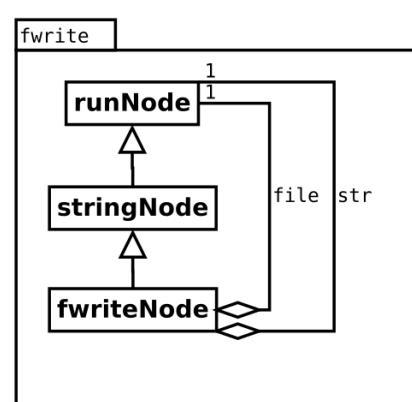
1.18.7. Leer fichero



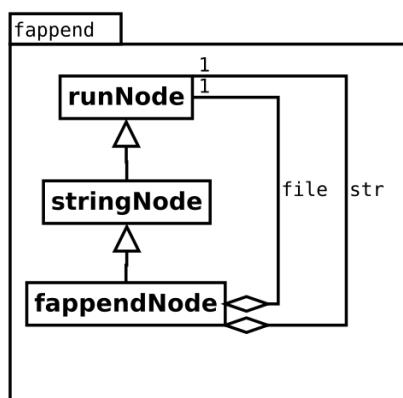
1.18.6. Cerrar flujo a fichero



1.18.8. Escribir en fichero

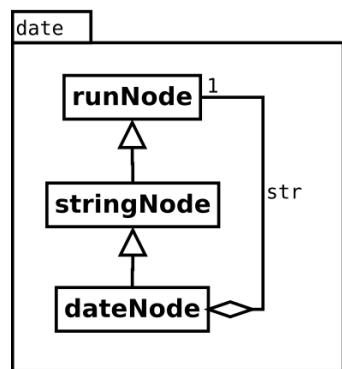
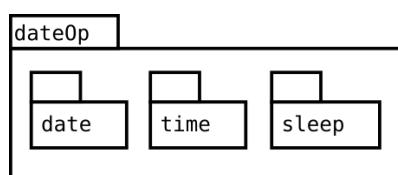


1.18.9. Escribir al final de fichero

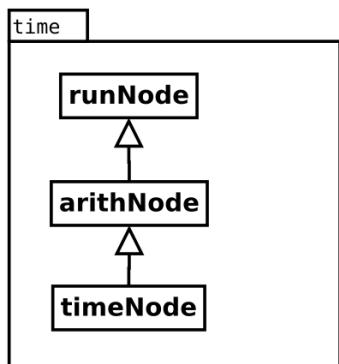


1.19. Fechas

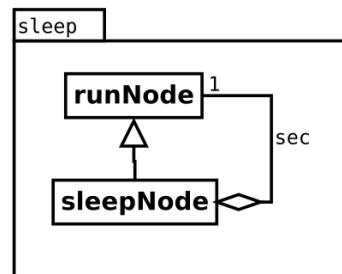
1.19.2. Fecha y hora con formato



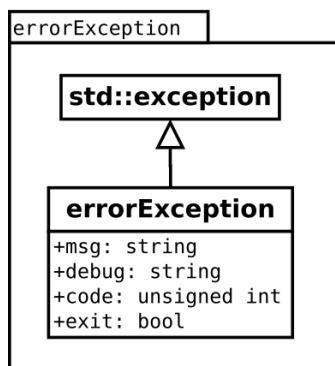
1.19.1. Tiempo Unix



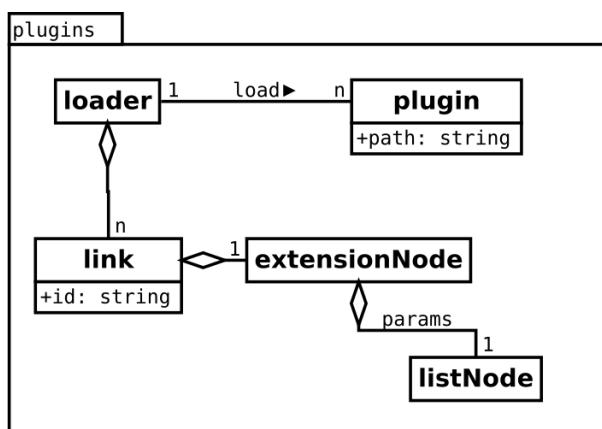
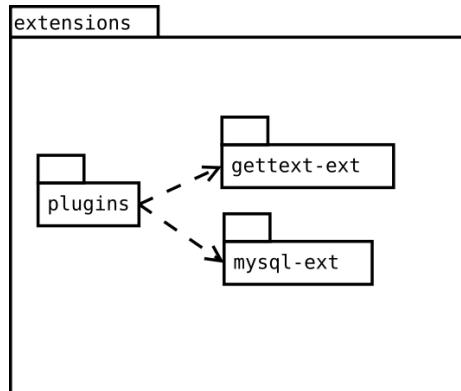
1.19.3. sleep



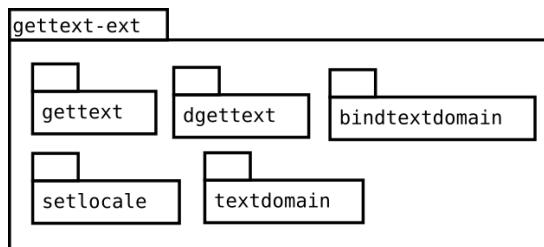
1.20. Errores



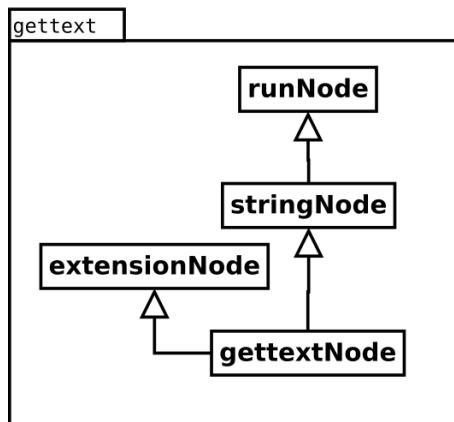
1.21. Extensiones



1.21.1. Biblioteca GNU de internacionalización (gettext)

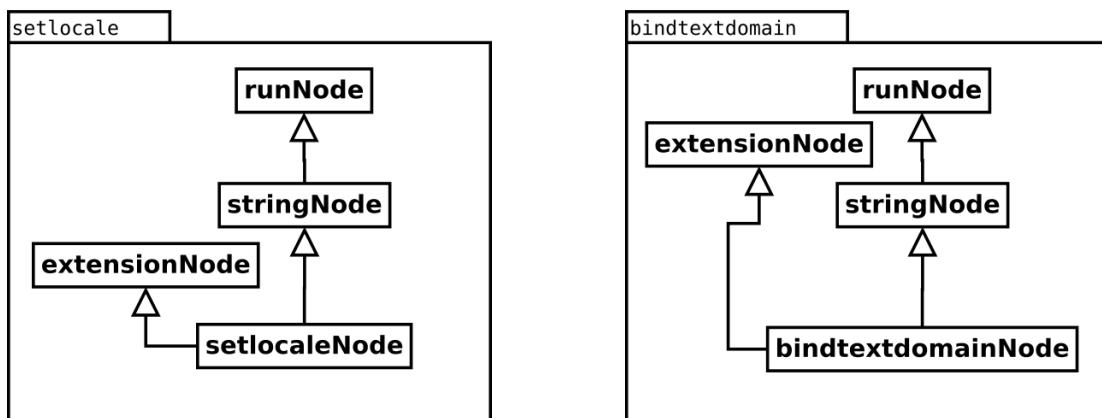


1.21.1.1 gettext



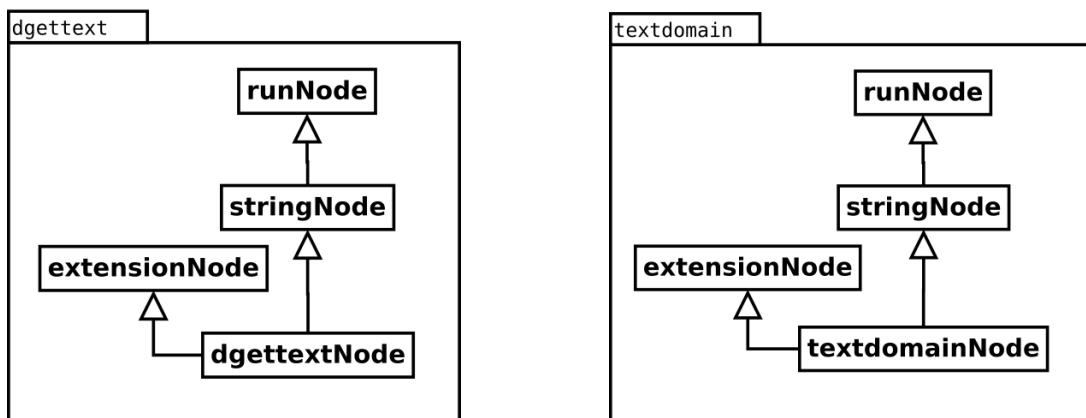
1.21.1.2 setlocale

1.21.1.4 bindtextdomain



1.21.1.3 dgettext

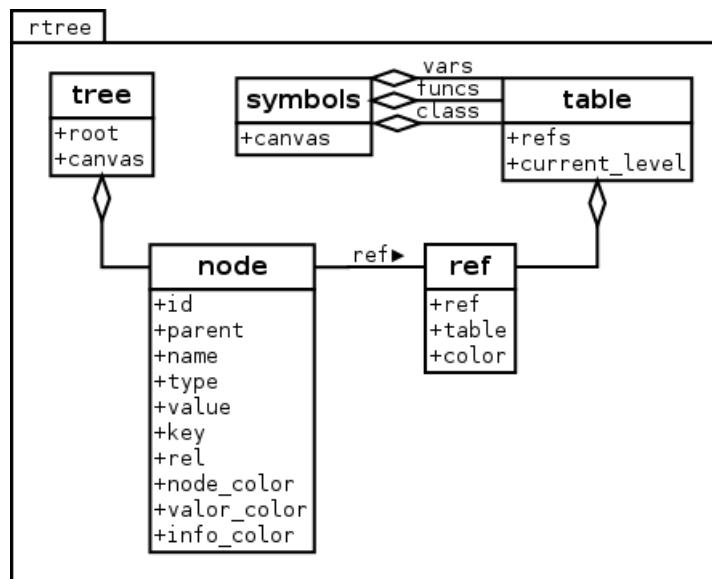
1.21.1.5 textdomain



1.22. rTree

El intérprete OMI tiene la capacidad de generar una salida relativa a su estado y funcionamiento. Para completar el proyecto se precisa de una herramienta capaz de interpretar y representar este estado interno de forma gráfica y textual.

El modelo de datos del cliente OMI se define de forma similar al intérprete. La principal diferencia es que en el intérprete este modelo de datos se usa para procesar y ejecutar el código fuente, mientras que en el cliente se usa para representar gráficamente el proceso llevado a cabo. Es por ello que el modelo de datos del cliente es más abstracto.



2. Modelo de casos de uso

En esta sección se detallan los casos de usos que recogen los sistemas que conforman el proyecto OMI. Un caso de uso describe la secuencia de interacciones entre el sistema y sus actores como consecuencia de un evento.

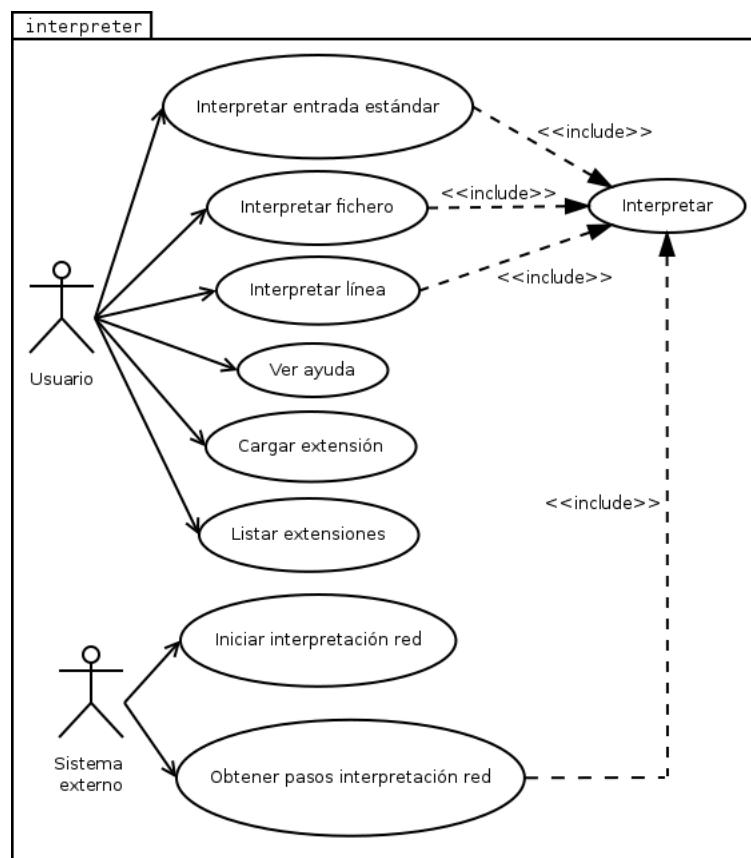
En primer lugar se describen los actores que hacen uso del sistema, llegándose a distinguir dos que serán nominados usuario y sistema externo. Luego se describen los casos de uso del sistema intérprete y del cliente runTree.

2.1. Actores

Los actores humanos que interactúan con los sistemas OMI presentan todos un mismo rol. Así el único actor definido será llamado usuario. Los usuarios que hacen uso del intérprete pueden ser desarrolladores, estudiantes u otros perfiles técnicos, pero todos usarán el sistema de la misma forma.

Por otro lado el intérprete OMI puede ser usado por otros sistemas, viéndose estos actores de determinados casos de uso. Algunos sistemas del proyecto OMI hacen uso del intérprete para llevar a cabo su propósito.

2.2. Intérprete



2.2.1. Interpretar entrada estándar

Caso de Uso: Interpretar entrada estándar.

Tipo: General.

Descripción: El usuario introduce un bloque de código en forma de cadena de caracteres mediante la entrada estándar. El sistema lo interpreta y ejecuta.

Actores: Usuario.

Precondiciones: El sistema debe estar esperando un bloque de código mediante la entrada estándar.

Postcondiciones: El código introducido es interpretado y ejecutado.

Escenario principal:

1. El usuario inicia el sistema facilitando un listado de argumentos.
2. El sistema asigna como variables cada argumento y solicita contenido fuente al usuario.
3. El usuario introduce un bloque de código en la entrada estándar.
4. El sistema obtiene el bloque de código a interpretar mediante la entrada estándar.
5. Incluir (Interpretar).

2.2.2. Interpretar fichero

Caso de Uso: Interpretar fichero.

Tipo: General.

Descripción: El usuario indica un fichero que contiene código y que será interpretado y ejecutado por el sistema.

Actores: Usuario.

Precondiciones: El sistema espera que se le indique un fichero.

Postcondiciones: El fichero es leído y el código en el mismo es interpretado y ejecutado.

Escenario principal:

1. El usuario indica la ruta a un fichero y una serie de argumentos
2. El sistema lee el fichero y obtiene el código en el mismo, además asigna cada argumento a variables.
3. Incluir (Interpretar).

Flujo alternativo:

- 2a. El fichero indicado no se encuentra.
 1. El sistema informa del error y finaliza.

2.2.3. Interpretar línea

Caso de Uso: Interpretar línea.

Tipo: General.

Descripción: El usuario introduce bloques de códigos denominados líneas de una forma interactiva. El sistema solicita por la entrada estándar las líneas de código, que serán interpretadas y ejecutadas.

Actores: Usuario.

Precondiciones: El sistema se encuentra en modo interactivo.

Postcondiciones: Se interpreta cada línea introducida por el usuario

Escenario principal:

1. El usuario inicia el sistema facilitando un listado de argumentos y la opción de interprete de línea.
2. El sistema asigna como variables cada argumento y muestra un prompt que indica que espera una línea de código.
3. El usuario introduce una línea de código.
4. El sistema lee de la entrada estándar la línea introducida.
5. Include (Interpretar).

El sistema repite el caso de uso hasta que se interpreta una sentencia que produzca la salida.

2.2.4. Interpretar

Caso de Uso: Interpretar.

Tipo: General.

Nivel: Subfunción.

Descripción: El sistema analiza, interpreta y ejecuta un bloque de código facilitado por el usuario. Para ello comprueba que este cumple con el léxico y la gramática del lenguaje que define, dividiéndolo a su vez en sentencias que serán interpretadas.

Precondiciones: Se dispone de un bloque de código.

Postcondiciones: El bloque de código es interpretado.

Escenario principal:

1. El sistema procesa y comprueba el bloque de código, aplicando la gramática y léxico que define.
2. El sistema obtiene e interpreta cada sentencia en el código, produciéndose el significado semántico que estas encierran.

Flujo alternativo:

- 1a. El código no respeta el léxico del lenguaje.
 1. El sistema informa del error y finaliza.
- 1b. El código no respeta la gramática del lenguaje.
 1. El sistema informa del error y finaliza.
- 2a. El código no contiene ninguna sentencia.
 1. El sistema finaliza.

2.2.5. Ver ayuda

Caso de Uso: Ver ayuda.

Tipo: General.

Descripción: Se muestra una ayuda que detalla cada opción disponible en el sistema.

Actores: Usuario.

Precondiciones: Sin precondiciones.

Postcondiciones: El sistema muestra un listado que presenta las distintas opciones.

Escenario principal:

1. El usuario indica que quiere visualizar la ayuda.
2. El sistema muestra un listado completo de las opciones que presenta.

2.2.6. Cargar extensión

Caso de Uso: Cargar extensión.

Tipo: General.

Descripción: El usuario indica que una extensión que será cargada por el sistema.

Actores: Usuario.

Precondiciones: Sin precondiciones.

Postcondiciones: El sistema cargar la extensión facilitada.

Escenario principal:

1. El usuario indica la ruta a la extensión que desea cargar.
2. El sistema carga la extensión para disponer de las distintas opciones que ofrece.

Flujo alternativo:

- 2a. La extensión indicada no se encuentra.
 1. El sistema informa del error.
- 2b. La extensión indicada no es una extensión válida.
 1. El sistema informa del error.

2.2.7. Listar extensiones

Caso de Uso: Listar extensiones.

Tipo: General.

Descripción: Lista las extensiones que serán cargadas en cada ejecución del sistema.

Actores: Usuario.

Precondiciones: Sin precondiciones.

Postcondiciones: El sistema lista las extensiones que serán cargadas.

Escenario principal:

1. El usuario indica que desea listar las extensiones cargadas.
2. El sistema lista las extensiones cargadas por defecto.

2.2.8. Iniciar interpretación red

Caso de Uso: Iniciar interpretación red

Tipo: Red

Descripción: Un sistema externo inicia una interpretación por red, estableciendo el contenido fuente que será interpretado. El sistema procesa la petición y abre un nuevo proceso de interpretación por pasos.

Precondiciones: No tiene.

Postcondiciones: Se establece el código fuente a interpretar y se inicia una interpretación por pasos.

Escenario principal:

1. El sistema espera una petición de interpretación por red.
2. El sistema externo inicia la comunicación y facilita el código fuente.
3. El sistema establece el código fuente y abre un nuevo proceso de interpretación por pasos.

Flujo alternativo:

- 1a. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.

2.2.9. Obtener pasos interpretación red

Caso de Uso: Obtener pasos interpretación red

Tipo: Red

Descripción: Un sistema externo obtiene un nuevo estado dentro de los pasos en el proceso de interpretación del código fuente establecido.

Precondiciones: Se ha establecido código fuente para una interpretación por pasos.

Postcondiciones: Se lleva a cabo un nuevo paso dentro del proceso de interpretación. Obteniéndose el siguiente estado.

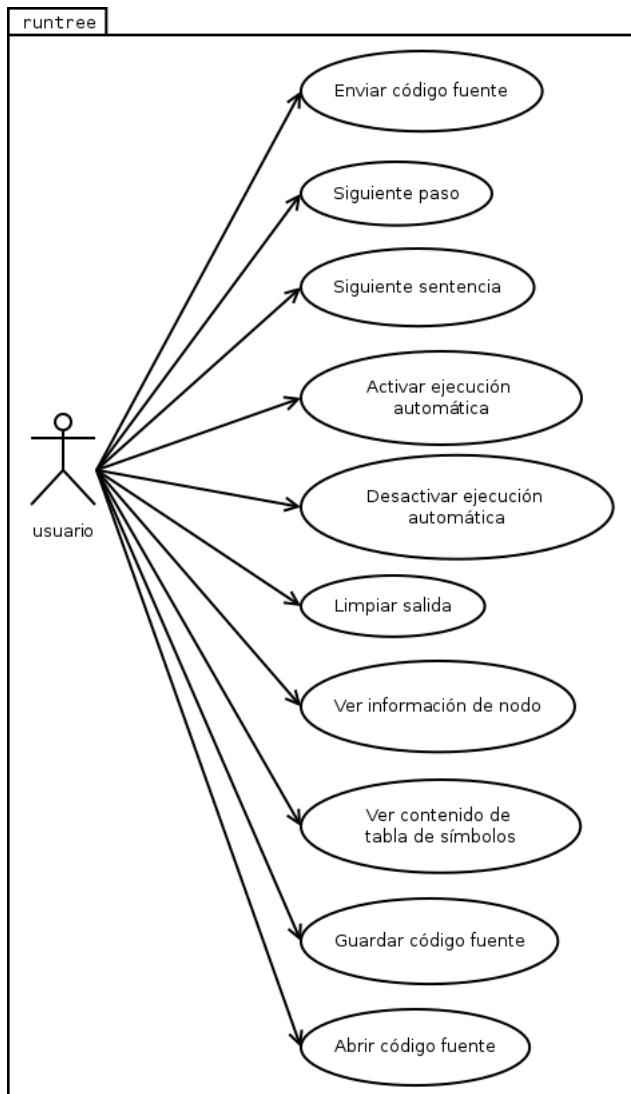
Escenario principal:

1. El sistema espera la petición de un nuevo paso en una iterpretación por red.
2. El sistema externo realiza la petición de un nuevo paso.
3. Incluir (Interpretar).
4. El sistema devuelve una estructura de datos que representa el estado actual del proceso de interpretación.

Flujo alternativo:

- 1a. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.
- 3a. El código fuente presenta errores sintáctico o léxicos.
 1. Se devuelve un estado de error.

2.3. runTree



2.3.1. Enviar código fuente

Caso de Uso: Enviar código fuente

Tipo: runTree

Descripción: El usuario envía texto correspondiente a código fuente para su interpretación y análisis. El sistema cliente envía el código al servidor que lo establecerá como fuente a interpretar y enviará datos relativos al proceso. El cliente imprime el árbol sintáctico y espera acciones del usuario.

Precondiciones: No tiene.

Postcondiciones: Se ha establecido el código fuente para el análisis del proceso de interpretación y se ha imprimido el árbol sintáctico.

Escenario principal:

1. El sistema solicita código fuente escrito en OMI.
2. El usuario introduce código fuente escrito en el lenguaje OMI.
3. El sistema cliente runtree envía el código fuente al servidor para su interpretación.
4. El sistema servidor devuelve una representación de los datos que describen el árbol sintáctico relativo al código fuente enviado.
5. El sistema cliente imprime el árbol sintáctico.

Flujo alternativo:

- 4a. El código fuente no presenta una sintaxis OMI correcta.
 1. Se devuelve un estado de error.
- 4b. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.

2.3.2. Siguiente paso

Caso de Uso: Siguiente paso

Tipo: runTree

Descripción: El usuario usuario solicita un nuevo paso en el proceso de interpretación.
El sistema cliente obtiene el nuevo paso del servidor y lo representa en pantalla.

Precondiciones:

- Existe un código fuente establecido para estudio.
- El estado actual del proceso no es final.
- La ejecución automática se encuentra desactivada.

Postcondiciones: Se representa en nuevo paso.

Escenario principal:

1. El usuario solicita un nuevo paso en el proceso de interpretación
2. El sistema cliente solicita la resolución de un nuevo paso y representa el nuevo estado en pantalla.

Flujo alternativo:

- 2a. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.

2.3.3. Siguiente sentencia

Caso de Uso: Siguiente sentencia

Tipo: runTree

Descripción: El usuario solicita la resolución de la siguiente sentencia. El sistema cliente obtiene el estado correspondiente a la interpretación de la siguiente sentencia dentro del proceso de interpretación y lo representa en pantalla.

Precondiciones:

- Existe un código fuente establecido para estudio.
- El estado actual del proceso no es final.
- La ejecución automática se encuentra desactivada.

Postcondiciones: Se representa en nuevo estado correspondiente a la interpretación de una sentencia completa.

Escenario principal:

1. El usuario solicita la resolución de una nueva sentencia en el proceso de interpretación.
2. El sistema cliente solicita la resolución de la sentencia completa y representa el nuevo estado en pantalla.

Flujo alternativo:

- 2a. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.

2.3.4. Activar ejecución automática

Caso de Uso: Activar ejecución automática

Tipo: runTree

Descripción: El usuario solicita la ejecución automática. El sistema cliente obtiene y representa cada paso dentro del proceso de interpretación hasta obtenerse un estado final.

Precondiciones:

- Existe un código fuente establecido para estudio.
- El estado actual del proceso no es final.
- La ejecución automática se encuentra desactivada.

Postcondiciones: Se obtiene un estado final.

Escenario principal:

1. El usuario activa la ejecución automática.
2. El sistema obtiene y representa cada paso en el proceso de interpretación hasta que se obtiene un estado final.

Flujo alternativo:

- 2a. El servicio no se encuentra disponible.
 1. Se devuelve un estado de error.

2.3.5. Desactivar ejecución automática

Caso de Uso: Desactivar ejecución automática

Tipo: runTree

Descripción: El usuario solicita la detención de la ejecución automática. El sistema cliente detiene la ejecución automática y representa el estado actual.

Precondiciones:

- Existe un código fuente establecido para estudio.
- El estado actual del proceso no es final.
- La ejecución automática se encuentra activada.

Postcondiciones: Se obtiene el estado actual del proceso.

Escenario principal:

1. El usuario desactiva la ejecución automática.
2. El sistema cliente para la ejecución automática y representa el estado actual

2.3.6. Limpiar salida

Caso de Uso: Limpiar salida

Tipo: runTree

Descripción: El usuario solicita la limpieza de los datos de salida. El sistema cliente elimina la información disponible en la consola de salida.

Precondiciones: No tiene.

Postcondiciones: La consola de salida queda vacía.

Escenario principal:

1. El usuario solicita la limpieza de la salida.
2. El sistema cliente limpia la información disponible en la consola de salida.

2.3.7. Ver información de nodo

Caso de Uso: Ver información de nodo.

Tipo: runTree

Descripción: El usuario marca un nodo para ver su información. El sistema cliente muestra la información relativa al nodo.

Precondiciones: Existe un código fuente establecido para estudio.

Postcondiciones: Se muestra información relativa al nodo.

Escenario principal:

1. El usuario marca un nodo para ver su información.
2. El sistema muestra información relativa al nodo tal como su posición de memoria interna, su tamaño, su tipo y su nombre.

2.3.8. Ver contenido de la tabla de símbolos

Caso de Uso: Ver contenido de la tabla de símbolos.

Tipo: runTree

Descripción: El usuario marca una tabla de símbolos para ver su contenido. El sistema cliente muestra la información relativa a la tabla de símbolos.

Precondiciones: Existe un código fuente establecido para estudio.

Postcondiciones: Se muestra información relativa a la tabla de símbolos.

Escenario principal:

1. El usuario indica una tabla de símbolos para ver su contenido.
2. El sistema cliente muestra los nodos referenciados desde la tabla de símbolos dada.

2.3.9. Guardar código fuente

Caso de Uso: Guardar código fuente.

Tipo: runTree

Descripción: El usuario solicita guardar el código fuente escrito en un fichero local y el cliente abre el cuadro de diálogo correspondiente.

Precondiciones: Existe un código escrito en el cliente

Postcondiciones: Se guarda el código fuente en un fichero local.

Escenario principal:

1. El usuario solicita guardar el código fuente en un fichero local.
2. El sistema abre el cuadro de diálogo correspondiente.

2.3.10. Abrir código fuente

Caso de Uso: Abrir código fuente.

Tipo: runTree

Descripción: El usuario solicita cargar código fuente desde un fichero local.

Precondiciones: No tiene.

Postcondiciones: Se carga el código fuente contenido en el fichero.

Escenario principal:

1. El usuario solicita abrir un fichero de código fuente.
2. El sistema abre el cuadro de diálogo correspondiente.
3. El sistema carga el contenido del código fuente.

Flujo alternativo:

- 2c. El fichero no contiene código en texto plano.
 1. Se devuelve un estado de error.

3. Visión general

El presente documento constituye el modelo de comportamiento del sistema.

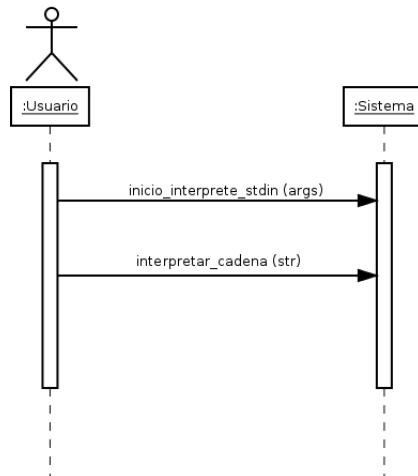
Partiendo de los casos de usos se presentan los diagramas de secuencias que modelan los eventos que el sistema puede recibir del usuario y los valores de retorno que produce como consecuencia de estos. Esta sección considera tanto al sistema intérprete como el cliente.

A partir de los diagramas de secuencia del sistema se obtienen las operaciones que este presenta. Luego se describen los contratos de cada una de las operaciones.

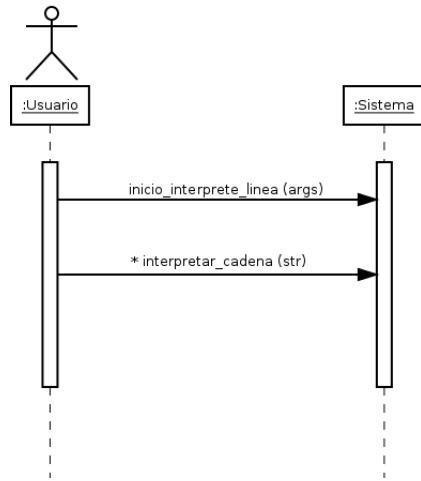
4. Diagramas de secuencia del sistema

4.1. Intérprete

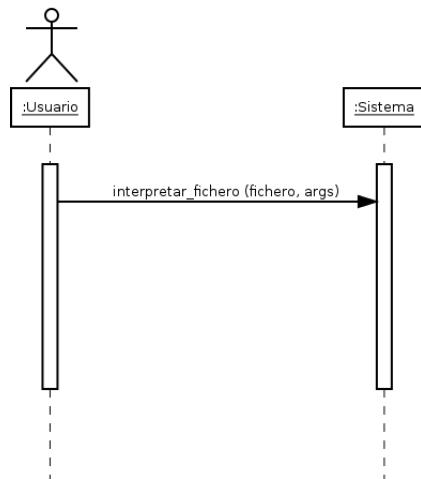
4.1.1. Interpretar entrada estándar



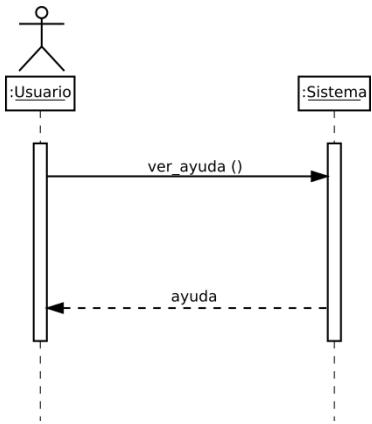
4.1.2. Interpretar línea



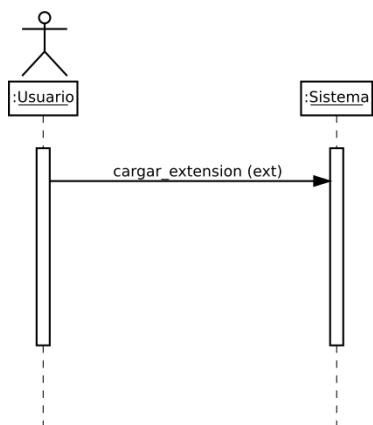
4.1.3. Interpretar fichero



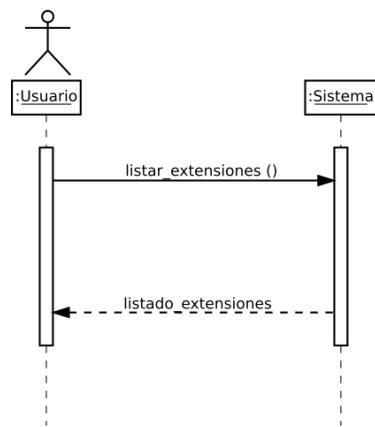
4.1.4. Ver ayuda



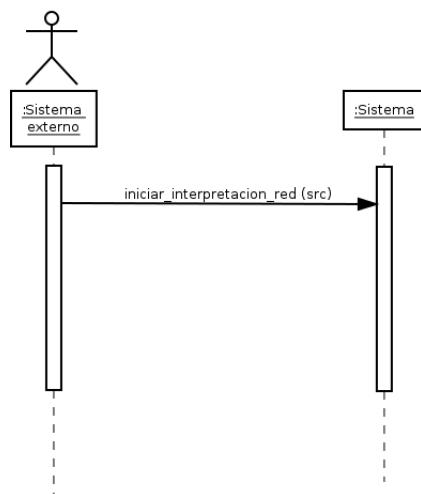
4.1.5. Cargar extensión



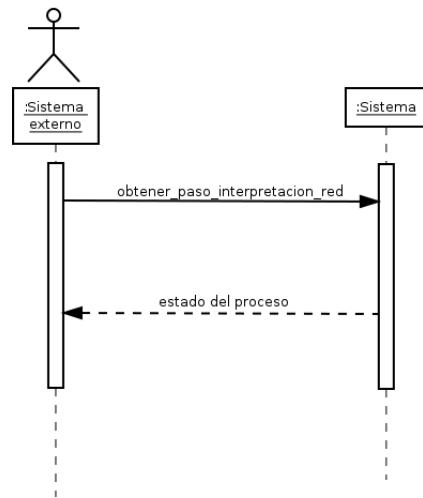
4.1.6. Listar extensiones



4.1.7. Iniciar interpretación red

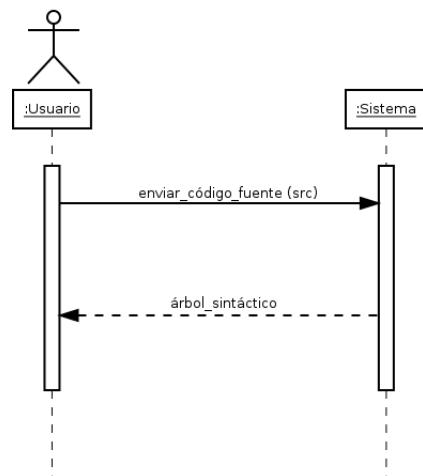


4.1.8. Obtener pasos interpretación red

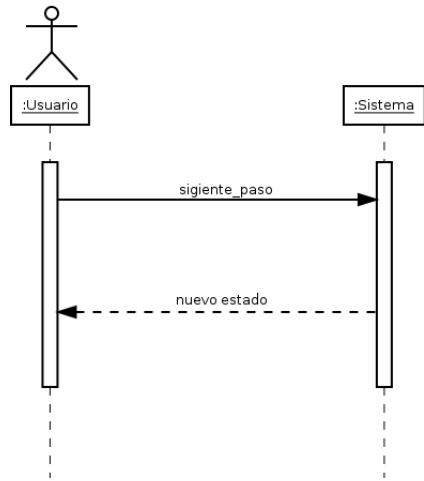


4.2. runTree

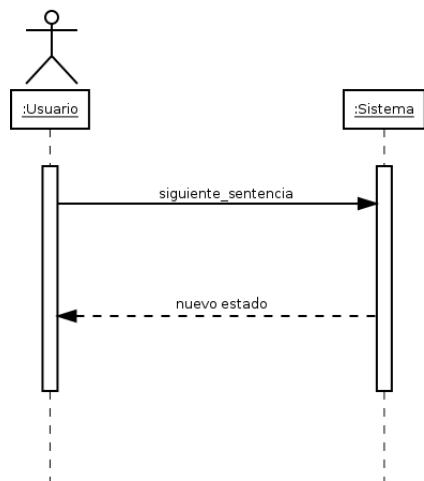
4.2.1. Enviar código fuente



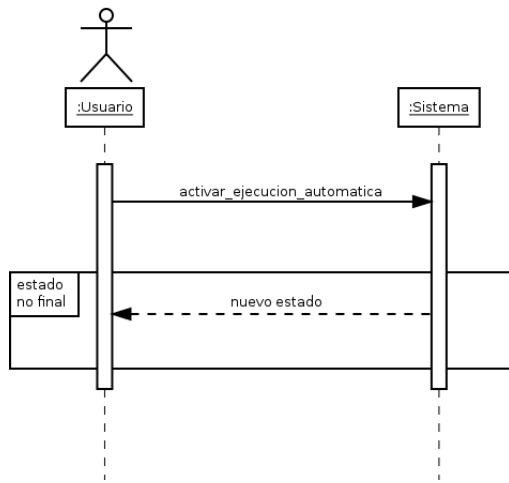
4.2.2. Siguiente paso



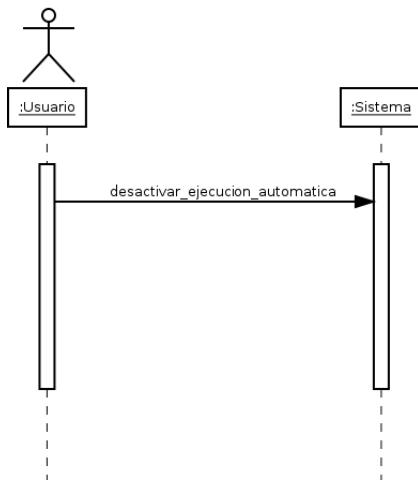
4.2.3. Siguiente sentencia



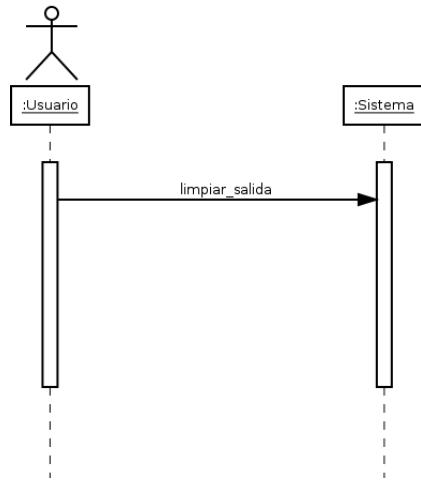
4.2.4. Activar ejecución automática



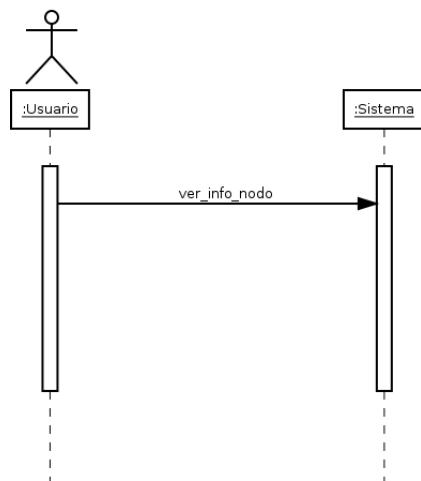
4.2.5. Desactivar ejecución automática



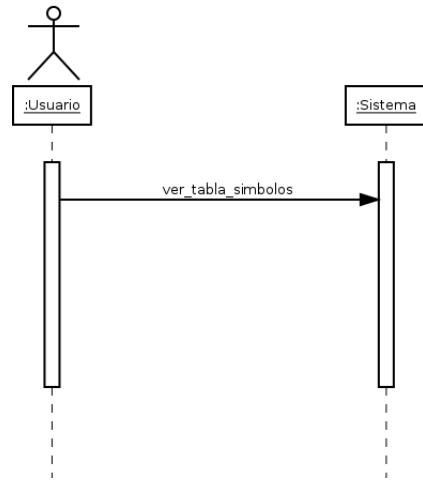
4.2.6. Limpiar salida



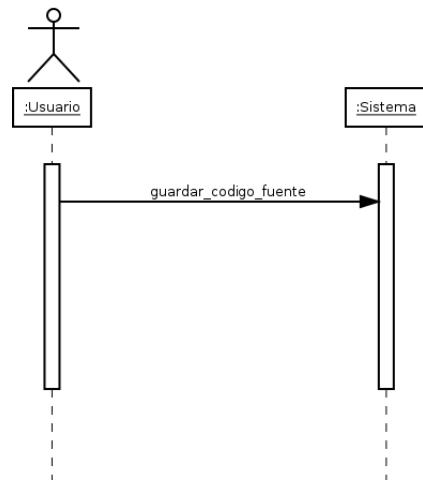
4.2.7. Ver infomación de nodo



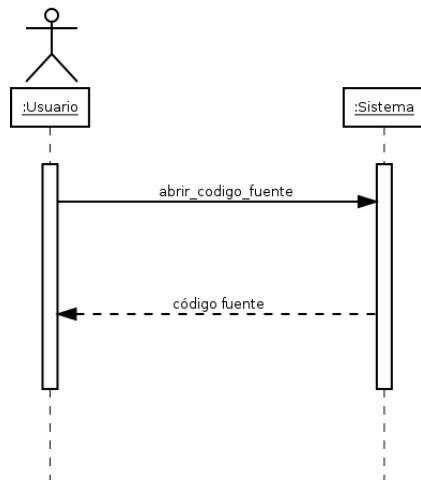
4.2.8. Ver contenido tabla de símbolo



4.2.9. Guardar código fuente



4.2.10. Abrir código fuente



5. Contratos de operaciones del sistema

En esta sección se listan las operaciones del sistema y se detallan los contratos de aquellas que implican un cambio en la estructura de datos interna del programa.

5.1. Intérprete

Sistema
+inicio_interprete_stdin(args) +inicio_interprete_linea(args) +interpretar_cadena(str) +interpretar_fichero(fichero,args) +ver_ayuda() +cargar_extension(ext) +listar_extensiones() +iniciar_interpretacion_red(src) +obtener_paso_interpretacion_red()

5.1.1. Operación inicio_interprete_stdin

Nombre: inicio_interprete_stdin(args)

Responsabilidades: Iniciar la interpretación de la entrada estándar.

Referencias Cruzadas: Caso de Uso: Interpretar entrada estándar

Precondiciones: No tiene.

Postcondiciones:

- Se creó una instancia “i” de “interpreter” (creación de objeto).
- Se inicializaron los atributos de “i”.
- Se crearon las instancias “ $a_0 \dots a_n$ ” de “arg” por cada argumento en “args” (creación de objeto).
- Se asignó “ args_i ” a “ $a_i.value$ ” ($a_i.value = \text{args}_i$) (modificación de atributos).
- Se asoció los “arg” “ $a_0 \dots a_n$ ” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “p” de “parser” (creación de objeto).
- Se inicializaron los atributos de “p”.
- Se creó una instancia “s” de “scanner” (creación de objeto).
- Se inicializaron los atributos de “s”.
- Se asoció el “scanner” “s” al objeto “p” de “parser” (creación de enlace).
- Se asoció el “parser” “p” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “c” de “context” (creación de objeto).
- Se inicializaron los atributos de “c”.
- Se creó una instancia “var” de “varSymbols” (creación de objeto).
- Se inicializaron los atributos de “var”.
- Se asoció el “varSymbols” “var” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “func” de “funcSymbols” (creación de objeto).
- Se inicializaron los atributos de “func”.
- Se asoció el “funcSymbols” “func” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “class” de “classSymbols” (creación de objeto).
- Se inicializaron los atributos de “class”.
- Se asoció el “classSymbols” “class” al objeto “c” de “context” (creación de enlace).
- Se asoció el “context” “c” al objeto “i” de “interpreter” (creación de enlace).

5.1.2. Operación inicio_interprete_linea

Nombre: inicio_interprete_linea(args)

Responsabilidades: Iniciar la interpretación interactiva línea a línea.

Referencias Cruzadas: Caso de Uso: Interpretar línea

Precondiciones: No tiene.

Postcondiciones:

- Se creó una instancia “i” de “interpreter” (creación de objeto).
- Se inicializaron los atributos de “i”.
- Se crearon las instancias “ $a_0 \dots a_n$ ” de “arg” por cada argumento en “args” (creación de objeto).
- Se asignó “ args_i ” a “ $a_i.value$ ” ($a_i.value = \text{args}_i$) (modificación de atributos).
- Se asoció los “arg” “ $a_0 \dots a_n$ ” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “p” de “parser” (creación de objeto).
- Se inicializaron los atributos de “p”.
- Se creó una instancia “s” de “scanner” (creación de objeto).
- Se inicializaron los atributos de “s”.
- Se asoció el “scanner” “s” al objeto “p” de “parser” (creación de enlace).
- Se asoció el “parser” “p” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “c” de “context” (creación de objeto).
- Se inicializaron los atributos de “c”.
- Se creó una instancia “var” de “varSymbols” (creación de objeto).
- Se inicializaron los atributos de “var”.
- Se asoció el “varSymbols” “var” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “func” de “funcSymbols” (creación de objeto).
- Se inicializaron los atributos de “func”.
- Se asoció el “funcSymbols” “func” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “class” de “classSymbols” (creación de objeto).
- Se inicializaron los atributos de “class”.
- Se asoció el “classSymbols” “class” al objeto “c” de “context” (creación de enlace).
- Se asoció el “context” “c” al objeto “i” de “interpreter” (creación de enlace).

5.1.3. Operación interpretar_cadena

Nombre: interpretar_cadena(str)

Responsabilidades: Interpreta el contenido fuente almacenado en la cadena “str”

Referencias Cruzadas:

- Caso de Uso: Interpretar entrada estándar
- Caso de Uso: Interpretar línea

Precondiciones:

- Se creó un “interpreter” “i”.
- Se creó y asoció una instancia de “parser” y “scanner” a “i”.
- Se creó y asoció una instancia de “varSymbols”, “funcSymbols” y “classSymbols” a “i”.

Postcondiciones:

- Se creó una instancia “s” de “source” (creación de objeto).
- Se asignó “str” a “s.src” ($s.\text{src} = \text{str}$) (modificación de atributos)
- Se asoció “s” al “scanner” componente del “interpreter” “i” (creación de enlace).
- Se creó un conjunto “ $t_0 \dots t_n$ ” de “token” a partir del análisis léxico (creación de objeto).
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “runNode” a partir del análisis sintáctico (creación de objetos).
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico (creación de enlace).
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “interprter” “i” (creación de enlace).
- Se creó un conjunto “ $var_0 \dots var_n$ ” de “refNode” correspondientes a las variables definidas en el contenido fuente (creación de objetos).
- Se creó un conjunto “ $val_0 \dots val_n$ ” de “runNode” correspondientes a los valores asignados a las variables definidas en el contenido fuente (creación de objetos).
- Se asoció “ $val_i \in val_0 \dots val_n$ ” a “ $var_i \in var_0 \dots var_n$ ” donde val_i es el valor de la variable var_i (creación de enlace).
- Se asoció todo “refNode” “ $var_0 \dots var_n$ ” al componente “varSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $func_0 \dots func_n$ ” de “refNode” correspondientes a las funciones con identificador definidas en el contenido fuente (creación de objetos).
- Se asoció “ $func_i \in func_0 \dots func_n$ ” a “ $n_i \in n_0 \dots n_n$ ” donde n_i es un “funcNode” correspondiente a la definición de la función $func_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $func_0 \dots func_n$ ” al componente “funcSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $class_0 \dots class_n$ ” de “refNode” correspondientes a las clases definidas en el contenido fuente (creación de objetos).
- Se asoció “ $class_i \in class_0 \dots class_n$ ” a “ $n_i \in n_0 \dots n_n$ ” donde n_i es un “classNode” correspondiente a la definición de la clase $class_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $class_0 \dots class_n$ ” al componente “classSymbols” de “i” (creación de enlace).

5.1.4. Operación interpretar_fichero

Nombre: interpretar_fichero(fichero, args)

Responsabilidades: Interpreta el contenido fuente almacenado en “fichero”

Referencias Cruzadas: Caso de Uso: Interpretar fichero

Precondiciones: No tiene.

Postcondiciones:

- Se creó una instancia “i” de “interpreter” (creación de objeto).
- Se inicializaron los atributos de “i”.
- Se creó el conjunto “ $a_0 \dots a_n$ ” de “arg” según el número de argumentos en “args” (creación de objeto).
- Se asignó “ $args_i$ ” a “ $a_i.value$ ” ($a_i.value = args_i$) (modificación de atributos).
- Se asoció los “arg” “ $a_0 \dots a_n$ ” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “p” de “parser” (creación de objeto).
- Se inicializaron los atributos de “p”.
- Se creó una instancia “s” de “scanner” (creación de objeto).
- Se inicializaron los atributos de “s”.
- Se asoció el “scanner” “s” al objeto “p” de “parser” (creación de enlace).
- Se asoció el “parser” “p” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “c” de “context” (creación de objeto).
- Se inicializaron los atributos de “c”.
- Se creó una instancia “var” de “varSymbols” (creación de objeto).
- Se inicializaron los atributos de “var”.
- Se asoció el “varSymbols” “var” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “func” de “funcSymbols” (creación de objeto).
- Se inicializaron los atributos de “func”.
- Se asoció el “funcSymbols” “func” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “class” de “classSymbols” (creación de objeto).
- Se inicializaron los atributos de “class”.
- Se asoció el “classSymbols” “class” al objeto “c” de “context” (creación de enlace).
- Se asoció el “context” “c” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “src” de “source” (creación de objeto).

- Se asignó el contenido de “fichero” a “src.src” ($\text{src}.\text{src} = \text{fichero}$) (modificación de atributos)
- Se asoció “src” al “scanner” “s” componente del “interpreter” “i” (creación de enlace).
- Se creó un conjunto “ $t_0...t_n$ ” de “token” a partir del análisis léxico (creación de objeto).
- Se creó un conjunto “ $n_0...n_n$ ” de “runNode” a partir del análisis sintáctico (creación de objetos).
- Se asoció “ $n_i \in n_0...n_n$ ” a “ $n_k \in n_0...n_n$ ” para construir el árbol sintáctico (creación de enlace).
- Se asoció “ $n_r \in n_0...n_n$ ”, raíz del árbol sintáctico, al “interpret” “i” (creación de enlace).
- Se creó un conjunto “ $var_0...var_n$ ” de “refNode” correspondientes a las variables definidas en el contenido fuente (creación de objetos).
- Se creó un conjunto “ $val_0...val_n$ ” de “runNode” correspondientes a los valores asignados a las variables definidas en el contenido fuente (creación de objetos).
- Se asoció “ $val_i \in val_0...val_n$ ” a “ $var_i \in var_0...var_n$ ” donde val_i es el valor de la variable var_i (creación de enlace).
- Se asoció todo “refNode” “ $var_0...var_n$ ” al componente “varSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $func_0...func_n$ ” de “refNode” correspondientes a las funciones con identificador definidas en el contenido fuente (creación de objetos).
- Se asoció “ $func_i \in func_0...func_n$ ” a “ $n_i \in n_0...n_n$ ” donde n_i es un “funcNode” correspondiente a la definición de la función $func_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $func_0...func_n$ ” al componente “funcSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $class_0...class_n$ ” de “refNode” correspondientes a las clases definidas en el contenido fuente (creación de objetos).
- Se asoció “ $class_i \in class_0...class_n$ ” a “ $n_i \in n_0...n_n$ ” donde n_i es un “classNode” correspondiente a la definición de la clase $class_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $class_0...class_n$ ” al componente “classSymbols” de “i” (creación de enlace).

5.1.5. Operación iniciar_interpretacion_red

Nombre: iniciar_interpretacion_red (src)

Responsabilidades: Iniciar la interpretación de una petición por red.

Referencias Cruzadas: Caso de Uso: Iniciar interpretación red

Precondiciones: No tiene.

Postcondiciones:

- Se creó una instancia “i” de “interpreter” (creación de objeto).
- Se inicializaron los atributos de “i”.
- Se creó una instancia “p” de “parser” (creación de objeto).
- Se inicializaron los atributos de “p”.
- Se creó una instancia “s” de “scanner” (creación de objeto).
- Se inicializaron los atributos de “s”.
- Se asoció el “scanner” “s” al objeto “p” de “parser” (creación de enlace).
- Se asoció el “parser” “p” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “c” de “context” (creación de objeto).
- Se inicializaron los atributos de “c”.
- Se creó una instancia “var” de “varSymbols” (creación de objeto).
- Se inicializaron los atributos de “var”.
- Se asoció el “varSymbols” “var” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “func” de “funcSymbols” (creación de objeto).
- Se inicializaron los atributos de “func”.
- Se asoció el “funcSymbols” “func” al objeto “c” de “context” (creación de enlace).
- Se creó una instancia “class” de “classSymbols” (creación de objeto).
- Se inicializaron los atributos de “class”.
- Se asoció el “classSymbols” “class” al objeto “c” de “context” (creación de enlace).
- Se asoció el “context” “c” al objeto “i” de “interpreter” (creación de enlace).
- Se creó una instancia “src” de “source” (creación de objeto).
- Se asignó el contenido de “fichero” a “src.src” ($\text{src}.src = \text{fichero}$) (modificación de atributos)
- Se asoció “src” al “scanner” “s” componente del “interpreter” “i” (creación de enlace).
- Se creó un conjunto $t_0 \dots t_n$ de “token” a partir del análisis léxico (creación de objeto).
- Se creó un conjunto $n_0 \dots n_n$ de “runNode” a partir del análisis sintáctico (creación de objetos).
- Se asoció $n_i \in n_0 \dots n_n$ a $n_k \in n_0 \dots n_n$ para construir el árbol sintáctico (creación de enlace).
- Se asoció $n_r \in n_0 \dots n_n$, raíz del árbol sintáctico, al “interprter” “i” (creación de enlace).

5.1.6. Operación obtener_paso_interpretacion_red

Nombre: obtener_paso_interpretacion_red

Responsabilidades: Obtiene el siguiente paso de la interpretación de una petición por red abierta.

Referencias Cruzadas: Caso de Uso: Obtener pasos interpretación red

Precondiciones:

- Se creó un “interpreter” “i”.
- Se creó y asoció una instancia de “parser” y “scanner” a “i”.
- Se creó y asoció una instancia de “varSymbols”, “funcSymbols” y “classSymbols” a “i”.
- Se creó una instancia “src” de “source”.
- Se asoció “src” al “scanner” “s” componente del “interpreter” “i”.
- Se creó un conjunto “ $t_0 \dots t_n$ ” de “token” a partir del análisis léxico.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “runNode” a partir del análisis sintáctico.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “interprter” “i”.

Postcondiciones:

- Se creó un conjunto “ $var_0 \dots var_n$ ” de “refNode” correspondientes a las variables definidas en el paso correspondiente (creación de objetos).
- Se creó un conjunto “ $val_0 \dots val_n$ ” de “runNode” correspondientes a los valores asignados a las variables definidas en el paso correspondiente (creación de objetos).
- Se asoció “ $val_i \in val_0 \dots val_n$ ” a “ $var_i \in var_0 \dots var_n$ ” donde val_i es el valor de la variable var_i (creación de enlace).
- Se asoció todo “refNode” “ $var_0 \dots var_n$ ” al componente “varSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $func_0 \dots func_n$ ” de “refNode” correspondientes a las funciones con identificador definidas en el paso correspondiente (creación de objetos).
- Se asoció “ $func_i \in func_0 \dots func_n$ ” a “ $n_i \in n_0 \dots n_n$ ” donde n_i es un “funcNode” correspondiente a la definición de la función $func_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $func_0 \dots func_n$ ” al componente “funcSymbols” de “i” (creación de enlace).
- Se creó un conjunto “ $class_0 \dots class_n$ ” de “refNode” correspondientes a las clases definidas en el paso correspondiente (creación de objetos).

- Se asoció “ $class_i \in class_0...class_n$ ” a “ $n_i \in n_0...n_n$ ” donde n_i es un “classNode” correspondiente a la definición de la clase $class_i$ (creación de enlaces).
- Se asoció todo “refNode” “ $class_0...class_n$ ” al componente “classSymbols” de “i” (creación de enlace).

5.2. runTree

Sistema
+enviar_codigo_fuente(src)
+siguiente_paso()
+siguiente_sentencia()
+activar_ejecucion_automática()
+desactivar_ejecucionAutomatica()
+limpiar_salida()
+ver_informacion_nodo()
+ver_tabla_simbolos()
+guardar_codigo_fuente()
+abrir_codigo_fuente()

5.2.1. Operación enviar_codigo_fuente

Nombre: enviar_codigo_fuente (src)

Responsabilidades: Envía código fuente para una interpretación por red.

Referencias Cruzadas: Caso de Uso: Enviar código fuente

Precondiciones: No tiene.

Postcondiciones:

- Se creó un “tree” “t” a partir del código enviado a interpretación (creación de objeto).
- Se creó un conjunto “ $n_0...n_n$ ” de “node” a partir de los datos recibidos de la petición (creación de objeto).
- Se asoció “ $n_i \in n_0...n_n$ ” a “ $n_k \in n_0...n_n$ ” para construir el árbol sintáctico (creación de enlace).
- Se asoció “ $n_r \in n_0...n_n$ ”, raíz del árbol sintáctico, al “tree” “t” (creación de enlace).
- Se inicializó el atributo “canvas” del “tree” ‘t’ para dibujar el árbol correspondiente a los nodos “ $n_0...n_n$ ”.
- Se creó un “symbols” “s” (creación de objeto).
- Se creó tres instancias de “table”: “vars”, “funcs” y “class” (creación de objeto).
- Se asociaron “vars”, “funcs” y “class” a “s” (creación de enlace).

5.2.2. Operación siguiente_paso

Nombre: siguiente_paso

Responsabilidades: Obtiene el siguiente paso del proceso de interpretación abierto.

Referencias Cruzadas: Caso de Uso: Siguiente paso

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociaron “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se creó el conjunto “ $r_0 \dots r_n$ ” de ‘refs’ según el nuevo paso del proceso de interpretación (creación de objeto).
- Se creó el conjunto “ $v_0 \dots v_m$ ” de ‘node’ correspondiente a los valores generados en el paso del proceso de interpretación (creación de objeto).
- Se asoció “ v_i ” a “ r_i ” como valor de la referencia (creación de enlace).
- Se asoció “ $r_i \in r_0 \dots r_n$ ” a “vars”, “funcs” o “class” según el paso del proceso de interpretación (creación de enlace).
- Se actualizó el valor del atributo “canvas” de “s” para reflejar el nuevo estado tras la ejecución del paso.
- Se actualizó el valor del atributo “canvas” de “t” para reflejar el nuevo estado tras la ejecución del paso.

5.2.3. Operación siguiente_sentencia

Nombre: siguiente_sentencia

Responsabilidades: Obtiene la siguiente sentencia dentro del proceso de interpretación abierto.

Referencias Cruzadas: Caso de Uso: Siguiente sentencia

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0...n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0...n_n$ ” a “ $n_k \in n_0...n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0...n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociaron “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se creó el conjunto “ $r_0...r_n$ ” de ‘refs’ según la nueva sentencia interpretada (creación de objeto).
- Se creó el conjunto “ $v_0...v_m$ ” de ‘node’ correspondiente a los valores generados en la sentencia interpretada (creación de objeto).
- Se asoció “ v_i ” a “ r_i ” como valor de la referencia (creación de enlace).
- Se asoció “ $r_i \in r_0...r_n$ ” a “vars”, “funcs” o “class” según la sentencia interpretada (creación de enlace).
- Se actualizó el valor del atributo “canvas” de “s” para reflejar el nuevo estado tras la ejecución de la sentencia.
- Se actualizó el valor del atributo “canvas” de “t” para reflejar el nuevo estado tras la ejecución de la sentencia.

5.2.4. Operación activar_ejecucionAutomatica

Nombre: activar_ejecucionAutomatica

Responsabilidades: Activa la ejecución automática del proceso de interpretación.

Referencias Cruzadas: Caso de Uso: Activar ejecución automática

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0...n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0...n_n$ ” a “ $n_k \in n_0...n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0...n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociaron “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se establece a “1” el atributo “auto” de “t”.

5.2.5. Operación desactivar_ejecucionAutomatica

Nombre: desactivar_ejecucionAutomatica

Responsabilidades: Desactiva la ejecución automática del proceso de interpretación.

Referencias Cruzadas: Caso de Uso: Desactivar ejecución automática

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociaron “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se establece a “0” el atributo “auto” de “t”.

5.2.6. Operación limpiar_salida

Nombre: limpiar_salida

Responsabilidades: Limpia la salida producida por el proceso de interpretación

Referencias Cruzadas: Caso de Uso: Limpiar salida

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociaron “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se restablece los valores del atributo “canvas” de “t”.

5.2.7. Operación ver_informacion_nodo

Nombre: ver_informacion_nodo

Responsabilidades: Muestra la información de un nodo

Referencias Cruzadas: Caso de Uso: Ver información nodo

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociarón “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se establece los valores del atributo “canvas” de “t” para representar la información del nodo.

5.2.8. Operación ver_tabla_simbolos

Nombre: ver_tabla_simbolos

Responsabilidades: Muestra la información contenida en una tabla de símbolos

Referencias Cruzadas: Caso de Uso: Ver información nodo

Precondiciones:

- Se creó un “tree” “t”.
- Se creó un conjunto “ $n_0 \dots n_n$ ” de “node”.
- Se asoció “ $n_i \in n_0 \dots n_n$ ” a “ $n_k \in n_0 \dots n_n$ ” para construir el árbol sintáctico.
- Se asoció “ $n_r \in n_0 \dots n_n$ ”, raíz del árbol sintáctico, al “tree” “t”.
- Se creó un “symbols” “s”.
- Se creó tres instancias de “table”: “vars”, “funcs” y “class”.
- Se asociarón “vars”, “funcs” y “class” a “s”.

Postcondiciones:

- Se establece los valores del atributo “canvas” de “s” para representar la información de la tabla de símbolos.

6. Modelo de interfaz de usuario

En esta sección se procede al análisis de la interfaz de usuario. Para ello se analizará la interfaz precisa para el intérprete y para el cliente runTree.

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

6.1. Intérprete

El interprete será accesible como cualquier otro comando de la consola del sistema. Este recibirá una serie de opciones y parámetros. Las opciones del intérprete tendrán los siguientes propósitos:

- Determinar cómo se toma el código fuente, pudiéndo ser desde la entrada estándar, un fichero o la propia línea de comandos
- Ejecutar el intérprete de forma interactiva, mostrando un prompt en el que se introduzca directamente las sentencias
- Listar y cargar los módulos del intérprete.
- Ver la ayuda.
- Abrir un puerto de escucha para peticiones de red.
- Configurar el formato de la salida que describe el proceso de interpretación.

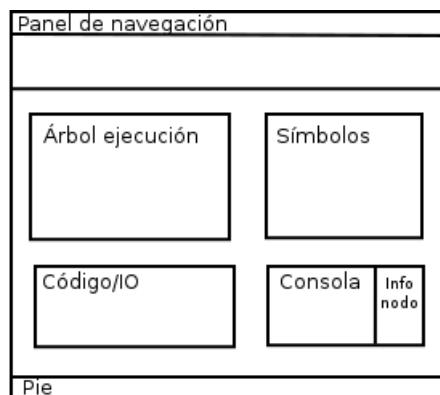
Como cualquier interprete que tome su entrada de la estrada estándar, el interprete OMI puede ser ejecutado por el sistema operativo si se indica al comienzo de un script como el shebang del mismo.

6.2. runTree

El cliente runTree será accesible desde un navegador web, y presentará una interfaz gráfica compatible con las versiones actuales de estos. La interfaz gráfica del cliente deberá contener la siguiente información:

- El código fuente introducido por el usuario y que será enviado a interpretar.
- El árbol de nodos resultado del análisis léxico y sintáctico.
- Las distintas tablas de símbolos que guardarán información sobre las variables, las funciones y las clases que serán creadas.
- La explicación del proceso semántico llevado a cabo.
- La salida producida como fruto de la ejecución del código fuente.
- La entrada introducida por el usuario y que ha sido solicitada por el código fuente.
- Información relativa a los nodos y tablas de símbolos que el usuario señale.

6.2.1. Wireframe



6.3. Sitio web

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.

El sitio web OMI se compone de:

Página de inicio: Describe e introduce brevemente el proyecto. Contiene enlaces a las demás secciones del sitio web. Además presenta un listado de noticias y enlaces de descargas a la última versión del intérprete.

Índice de documentación: Página que representa un índice de los documentos que conforman el proyecto.

Documentos: Páginas relativas a la documentación del proyecto en sí.

Navegador de clases: Páginas relativas a la documentación de las clases incluidas en la biblioteca.

Navegador de ficheros: Páginas relativas a la documentación de los ficheros que conforman el código fuente de la biblioteca y el intérprete.

Navegador de gramática: Páginas relativas a la documentación gráfica de la gramática del lenguaje.

Descargas: Página que enlaza la descarga de las distintas versiones del software que conforma el proyecto, disponibles en varios formatos de instalación.

Sobre OMI: Página con información relativa a la motivación y circunstancias en las que se ha dado el proyecto. Además da detalles sobre los autores y los organismos implicados en el desarrollo del mismo.

Contacto: Página con información de contacto.

6.3.1. Wireframe

