Open Modular Interpreter

Modelo conceptual de datos

Fco. Javier Bohórquez Ogalla

Índice

0.1. 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 conciste en tomar codigo 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.

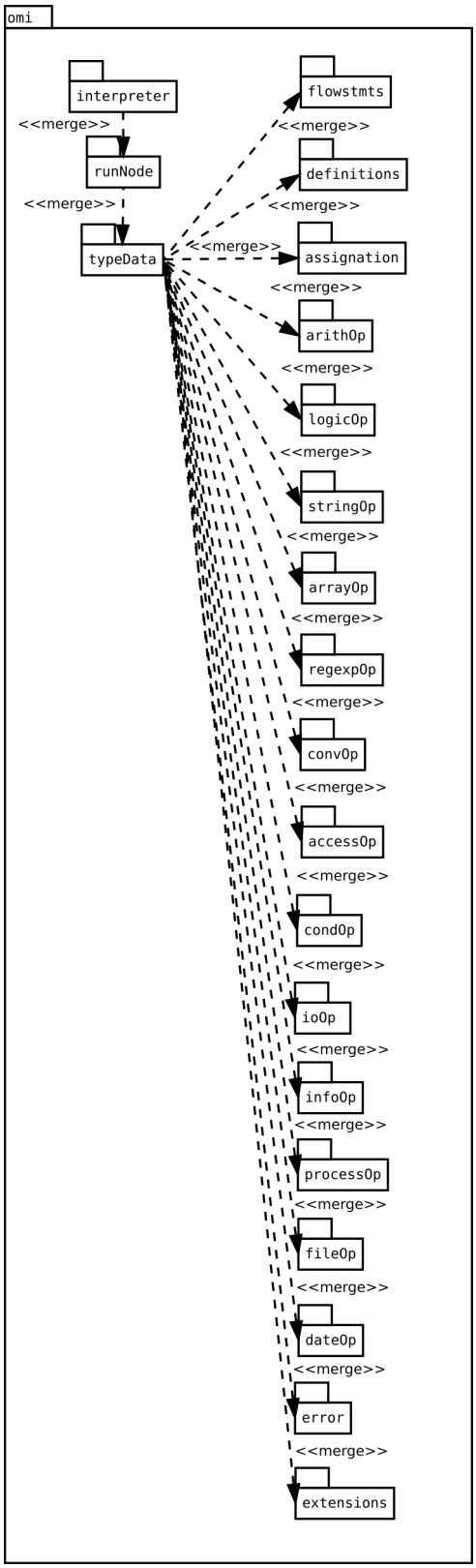
El paquete “typeData” describe los nodos correspondientes a los tipos de datos básicos que pueden ser manipulados por el sistema.

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.


omi
interpreter
<<merge>>
runNode
<<merge>>
typeData
<<merge>>
flowstmts
<<merge>>
definitions
<<merge>>
assignation
<<merge>>
arithOp
<<merge>>
logicOp
<<merge>>
stringOp
<<merge>>
arrayOp
<<merge>>
regexpOp
<<merge>>
convOp
<<merge>>
accessOp
<<merge>>
condOp
<<merge>>
ioOp
<<merge>>
infoOp
<<merge>>
processOp
<<merge>>
fileOp
<<merge>>
dateOp
<<merge>>
error
<<merge>>
extensions

0.1.1. Intérprete

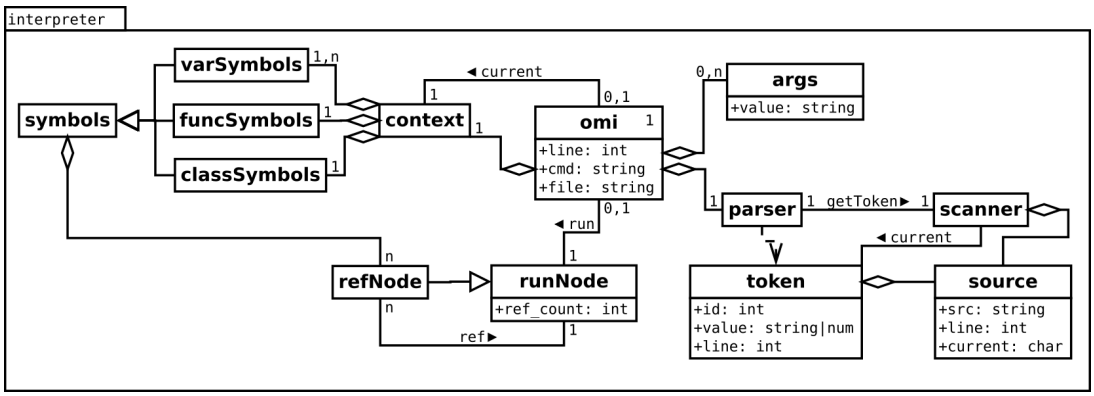
El sistema OMI se corresponde con un interprete que opera sobre un contenido fuente escrito en el lenguaje con el mismo nombre. El interprete se compone de un analizador sintáctico que encierra la gramática del lenguaje, esta es descrita a partir de una serie de tokens.

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 singnificado 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.



0.1.2. Nodos ejecutables

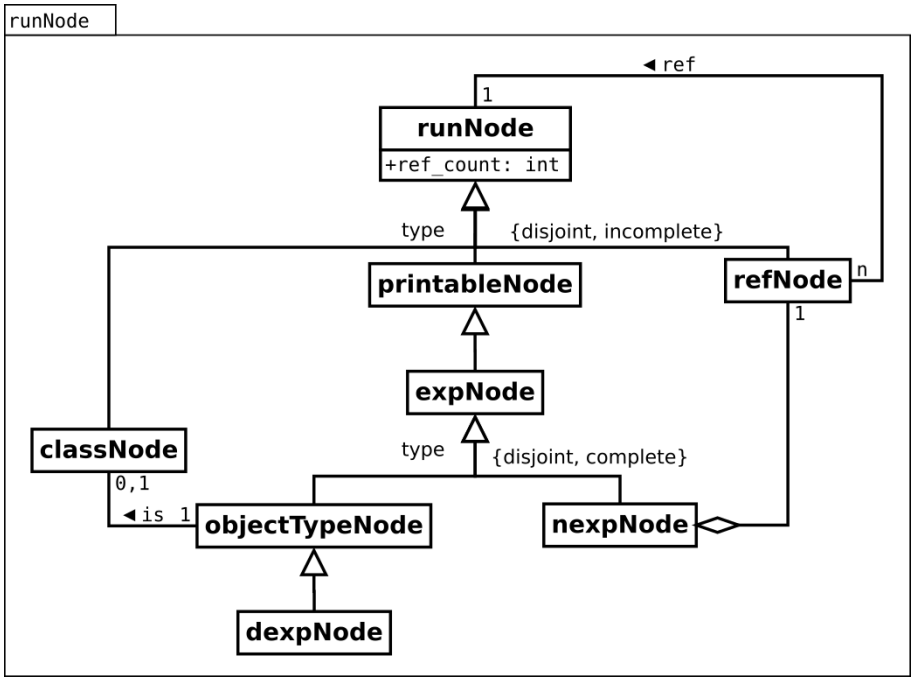
Se definen un nodo ejecutable para cada aspecto o funcionalidad que contemple el lenguaje. Cada sentencia se corresponde con un nodo ejecutable, que a su vez puede estar compuestos de otros nodos. Cada nodo ejecutable guarda el número de nodos que lo referencian para que se pueda hacer un uso óptimo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

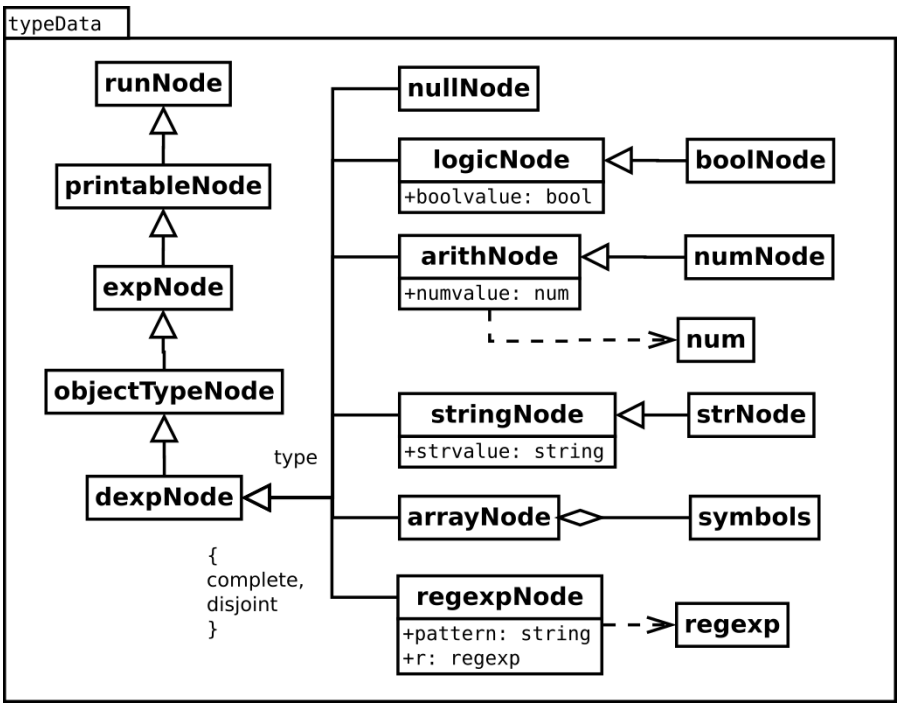


0.1.3. Tipos de datos

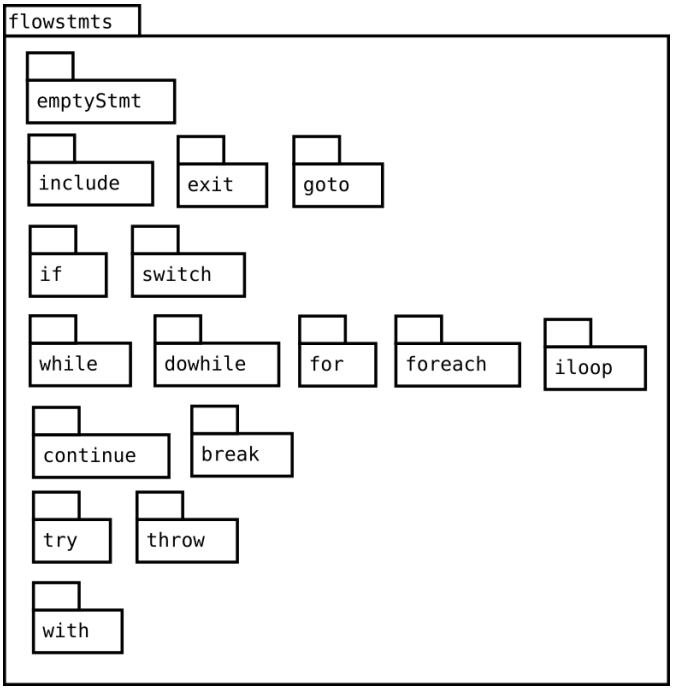
Este paquete contiene los nodos que representan expresiones con tipos de datos definidos. Se describe cada tipo de dato como un nodo con un valor asociado (en algunos casos el tipo puede comprender un único valor).

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.

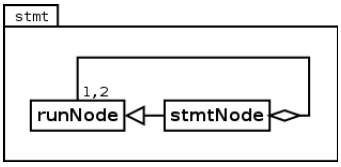
Algunos nodos de tipos datos son concretados por nodos que representan un valor constante de dicho tipo de dato.



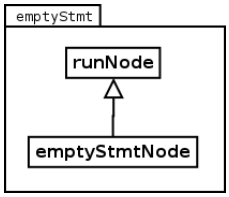
0.1.4. Sentencias de control de flujo



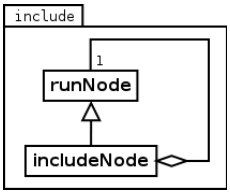
0.1.4.1 Sentencia



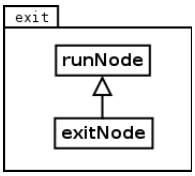
0.1.4.2 Sentencia vacía



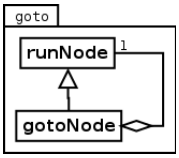
0.1.4.3 include



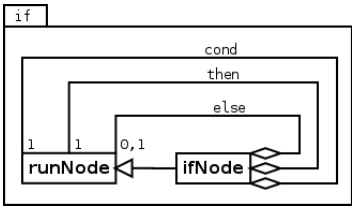
0.1.4.4 exit



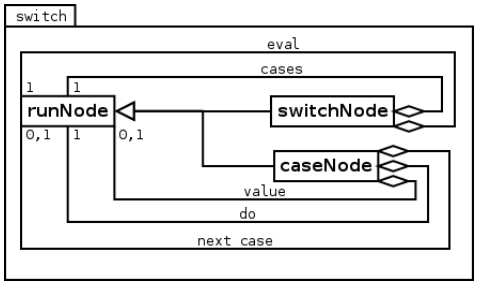
0.1.4.5 goto



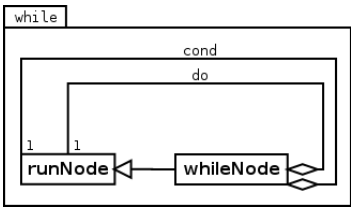
0.1.4.6 if



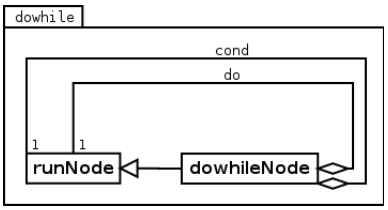
0.1.4.7 switch



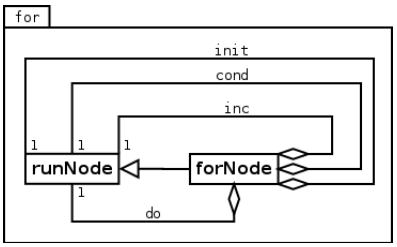
0.1.4.8 while



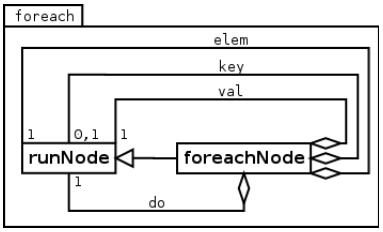
0.1.4.9 do...while



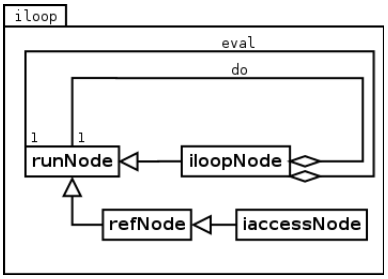
0.1.4.10 for



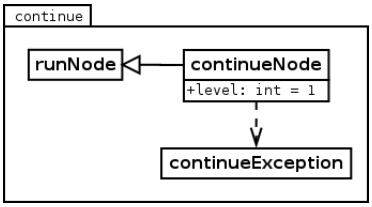
0.1.4.11 foreach



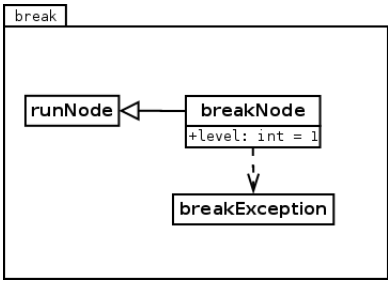
0.1.4.12 iloop



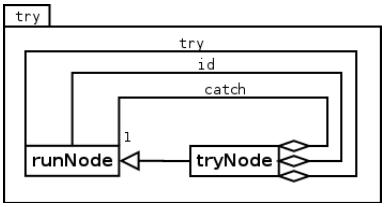
0.1.4.13 continue



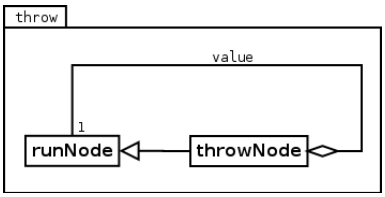
0.1.4.14 break



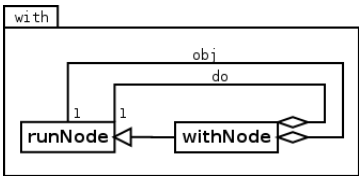
0.1.4.15 try



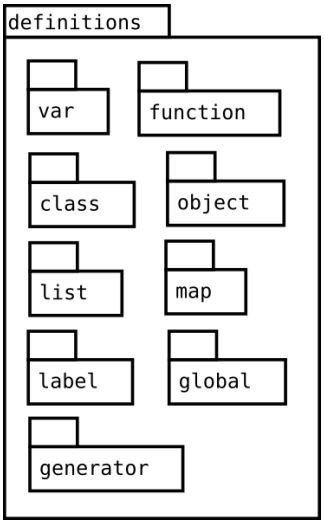
0.1.4.16 throw



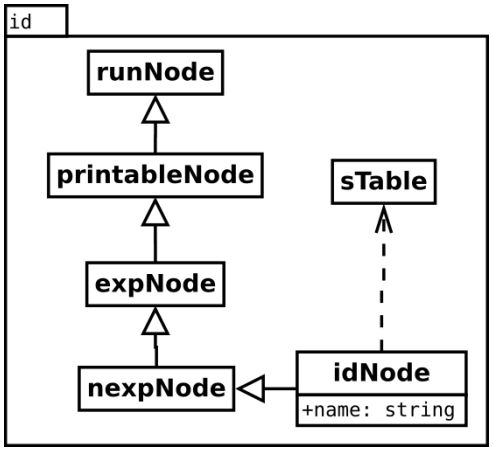
0.1.4.17 with



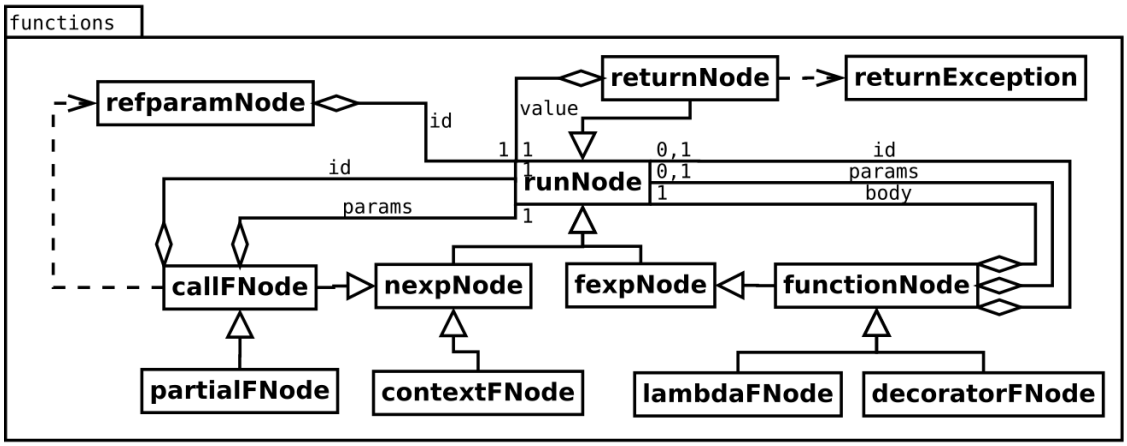
0.1.5. Definiciones



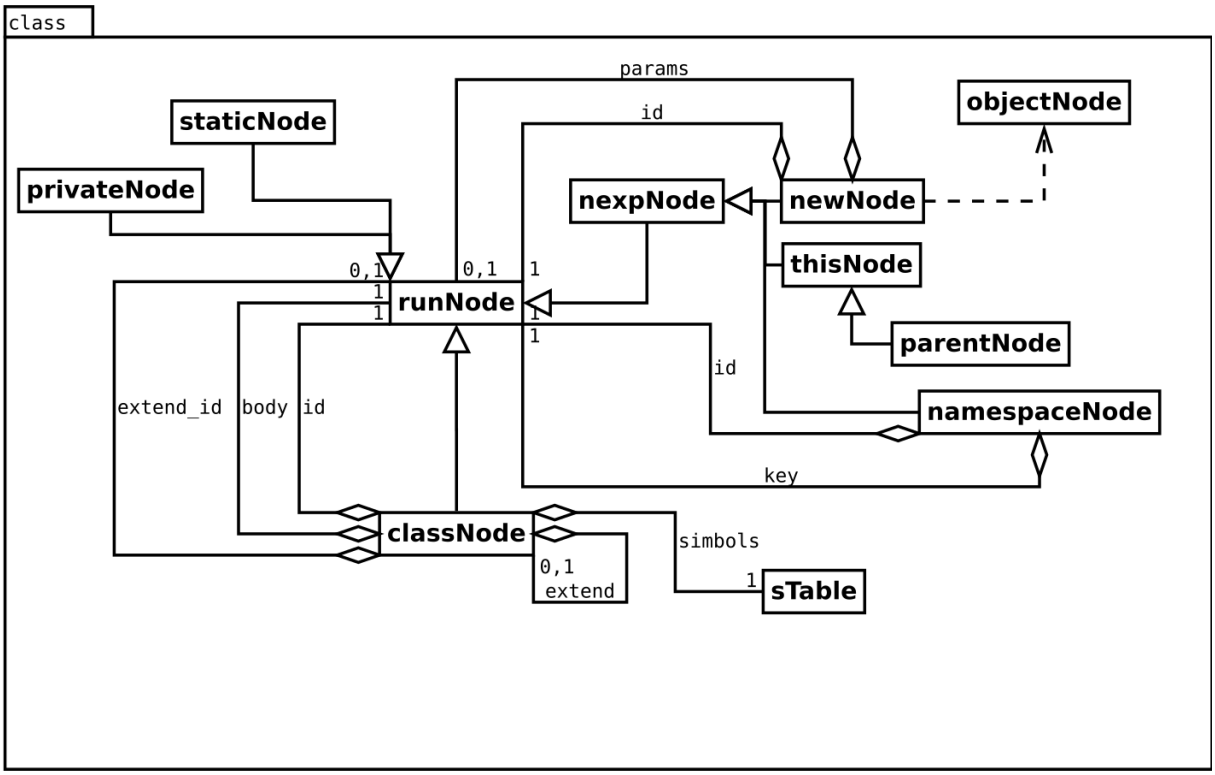
0.1.5.1 Variables



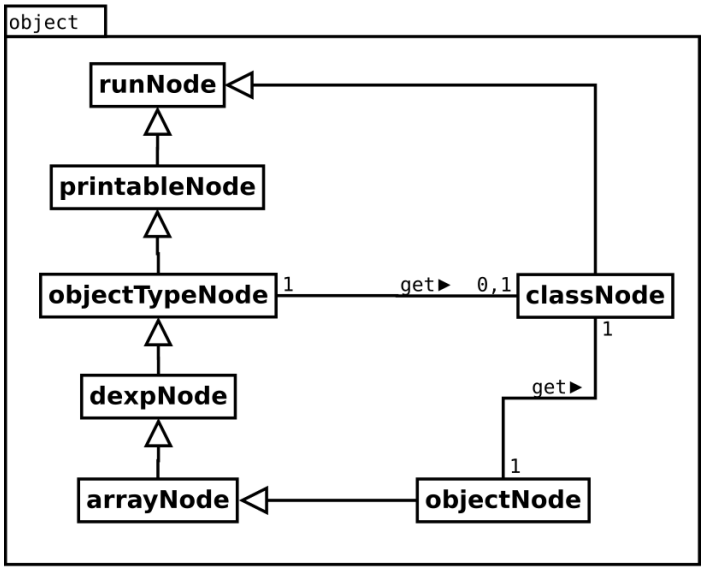
0.1.5.2 Funciones



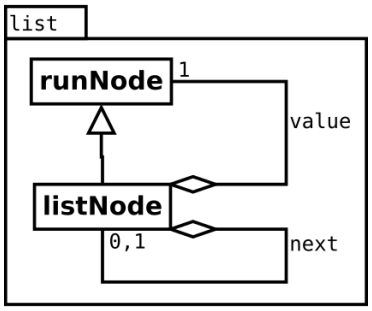
0.1.5.3 Clases



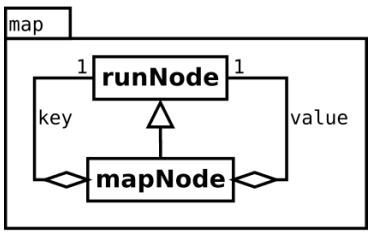
0.1.5.4 Objetos



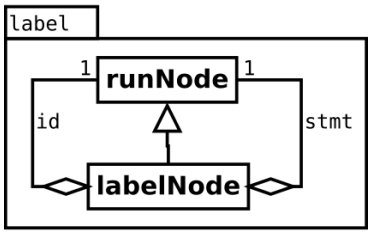
0.1.5.5 Listas



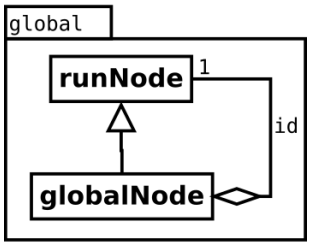
0.1.5.6 Pares clave/valor



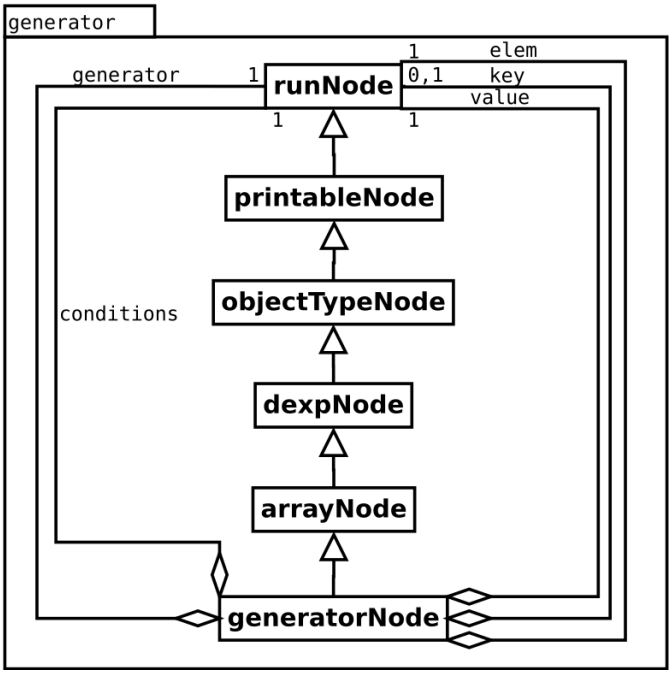
0.1.5.7 Etiquetas



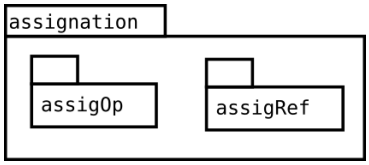
0.1.5.8 Definiciones globales



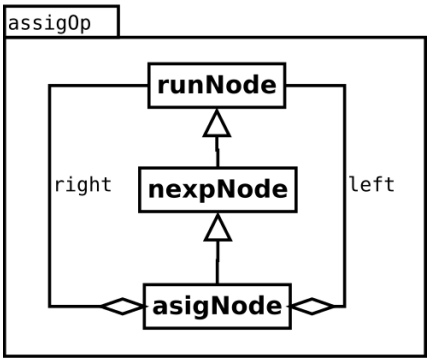
0.1.5.9 Generadores



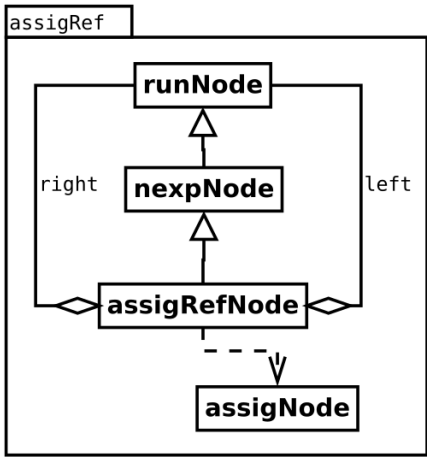
0.1.6. Asignaciones



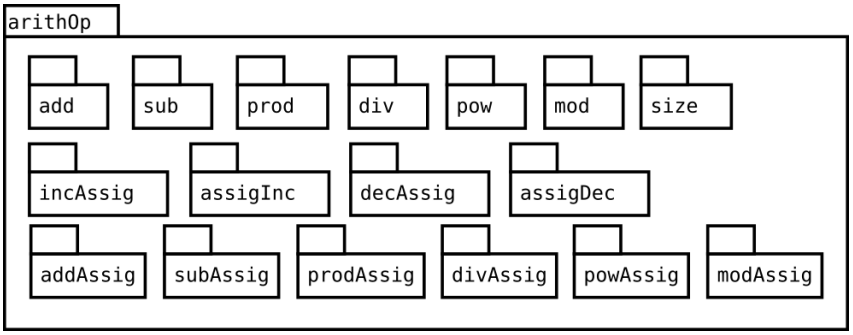
0.1.6.1 Asignación



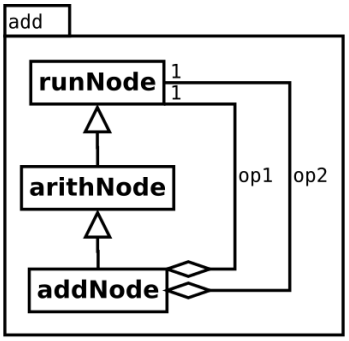
0.1.6.2 Asignación de referencia



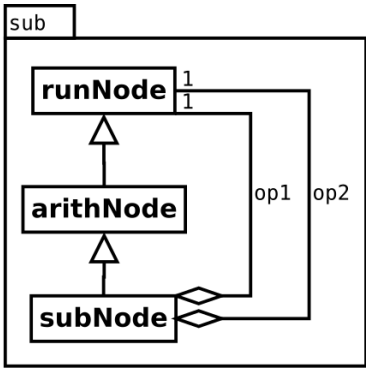
0.1.7. Operadores aritméticos



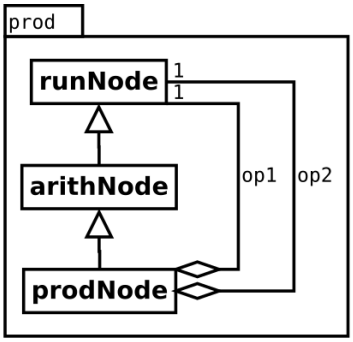
0.1.7.1 Suma



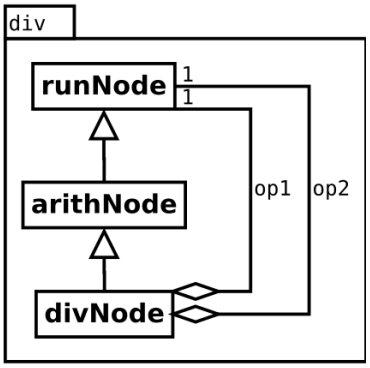
0.1.7.2 Diferencia



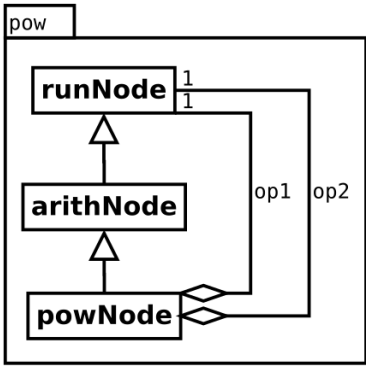
0.1.7.3 Producto



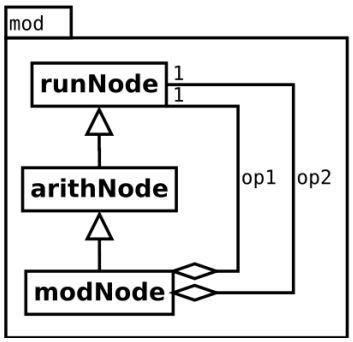
0.1.7.4 División



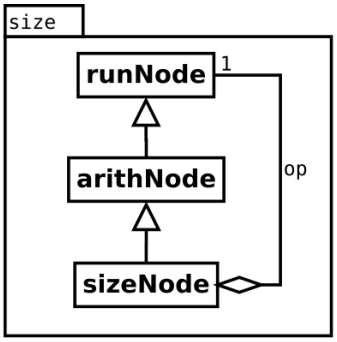
0.1.7.5 Potencia



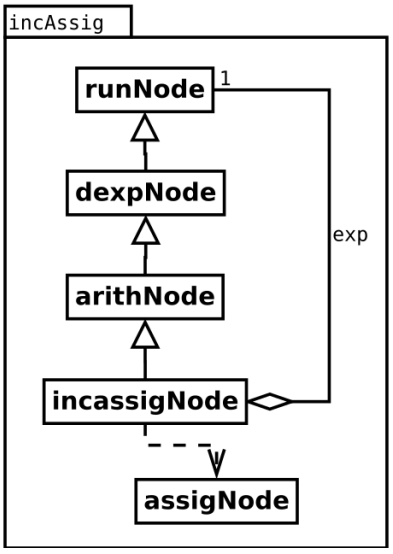
0.1.7.6 Módulo



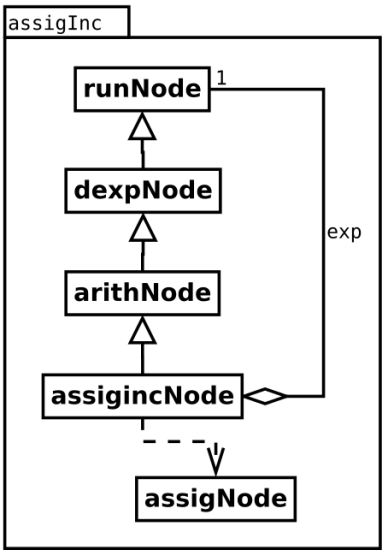
0.1.7.7 Tamaño



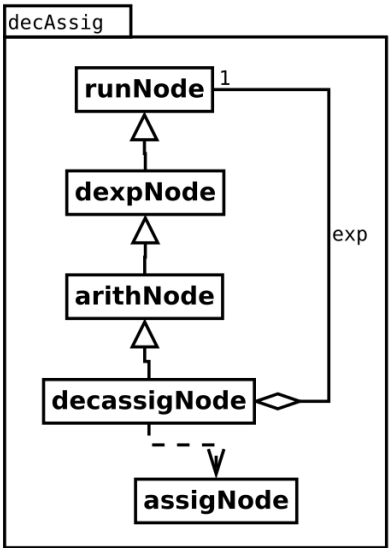
0.1.7.8 Incremento y asignación



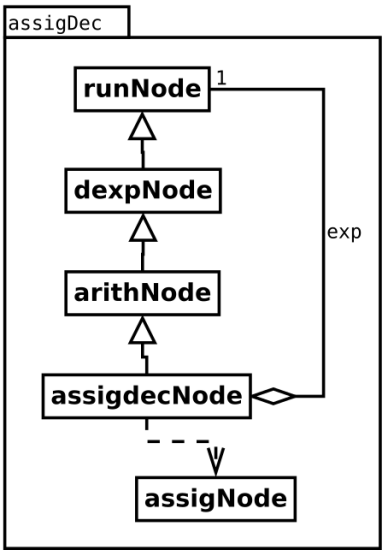
0.1.7.9 Asignación e incremento



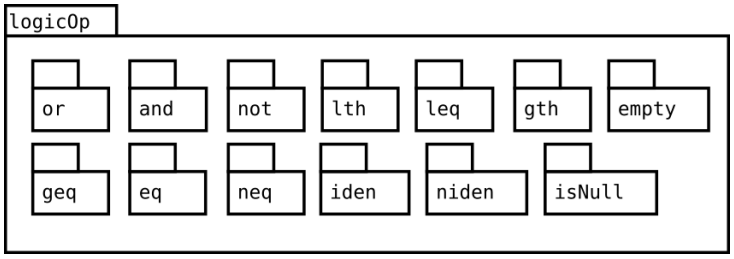
0.1.7.10 Decremento y asignación



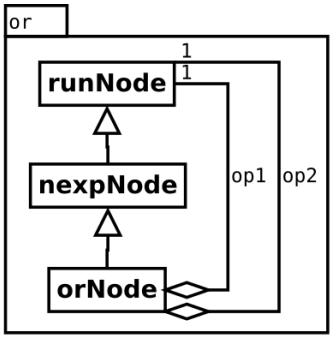
0.1.7.11 Asignación y decremento



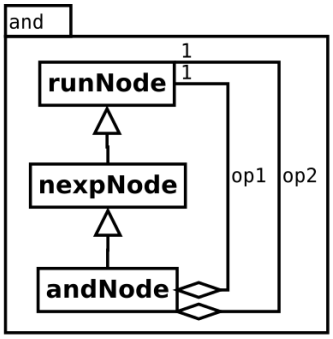
0.1.8. Operadores lógicos



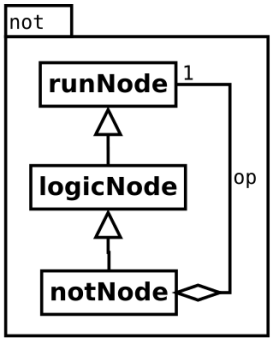
0.1.8.1 Or



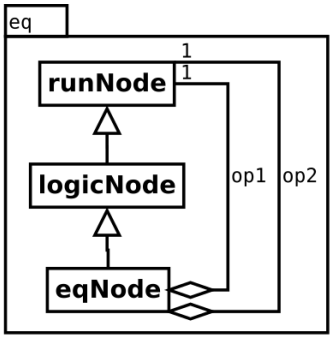
0.1.8.2 And



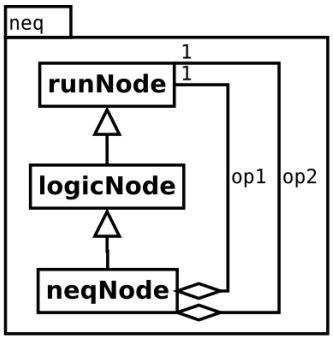
0.1.8.3 Negación



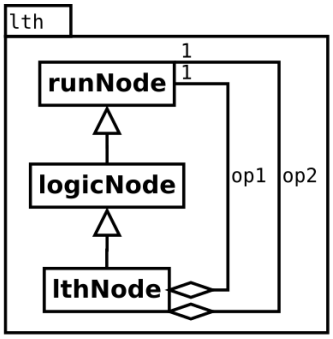
0.1.8.4 Igual que



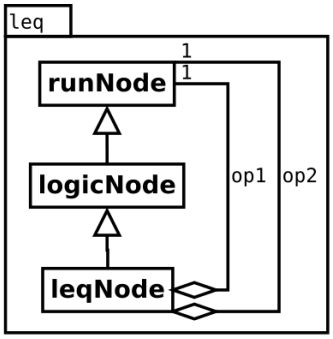
0.1.8.5 Distinto que



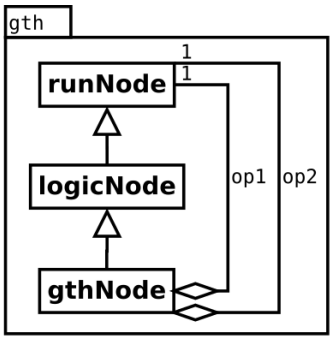
0.1.8.6 Menor que



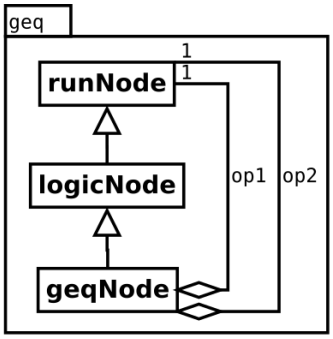
0.1.8.7 Menor o igual que



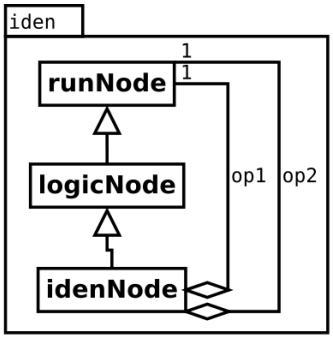
0.1.8.8 Mayor que



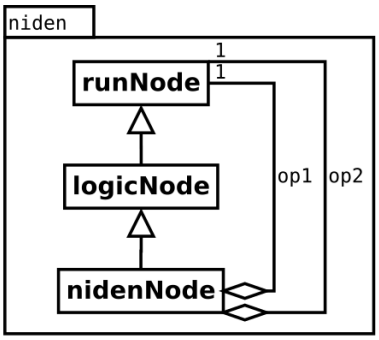
0.1.8.9 Mayor o igual que



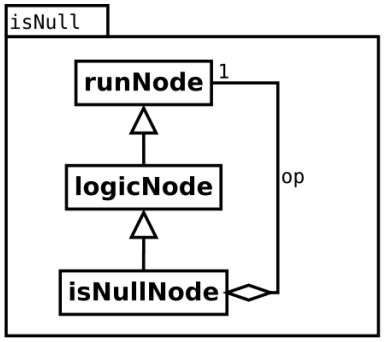
0.1.8.10 Idéntico a



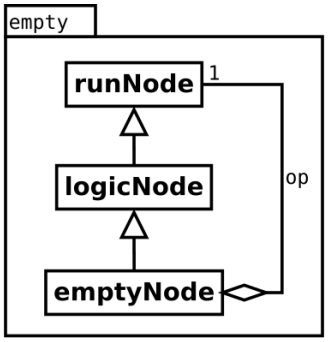
0.1.8.11 No idéntico a



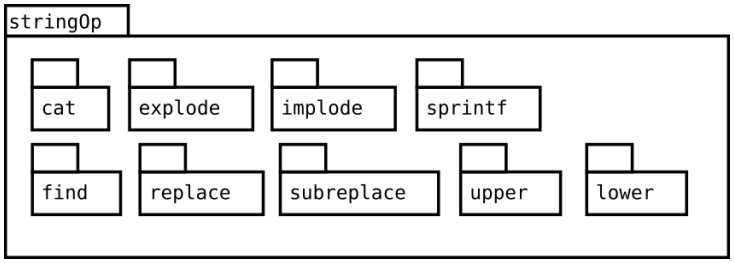
0.1.8.12 Es nulo



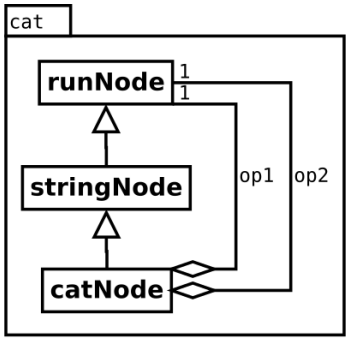
0.1.8.13 Vacío



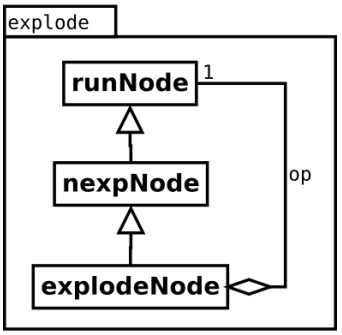
0.1.9. Operadores sobre cadenas



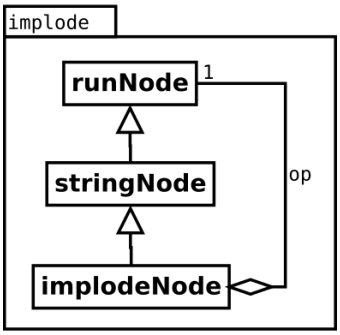
0.1.9.1 Concatenación



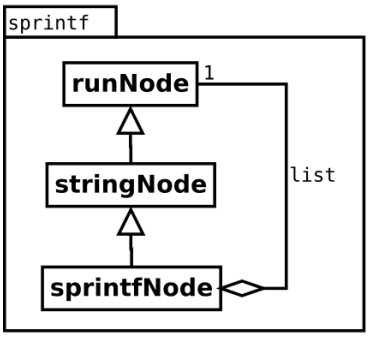
0.1.9.2 explode



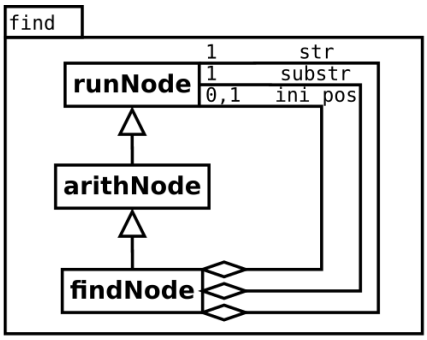
0.1.9.3 implode



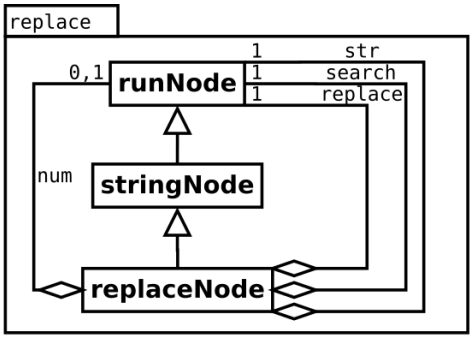
0.1.9.4 sprintf



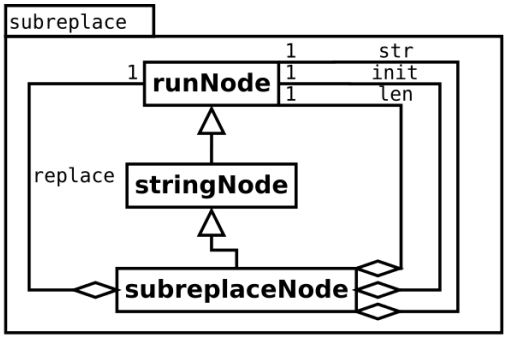
0.1.9.5 Buscar subcadena



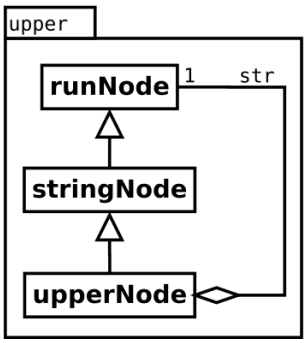
0.1.9.6 Buscar y remplazar



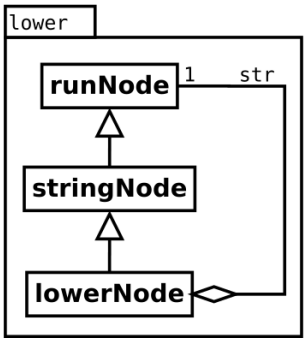
0.1.9.7 Remplazar subcadena



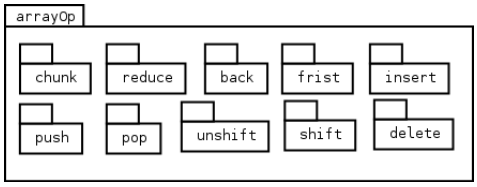
0.1.9.8 Convertir a mayúsculas



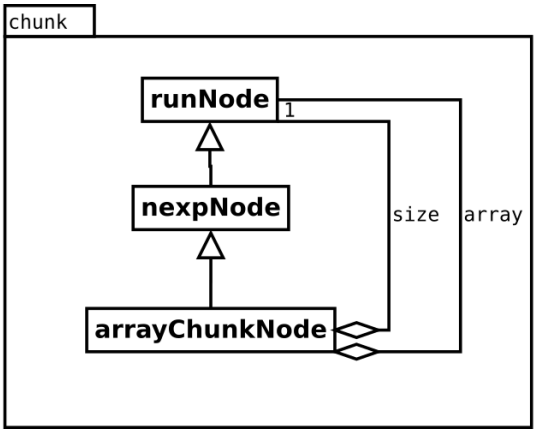
0.1.9.9 Convertir a minúsculas



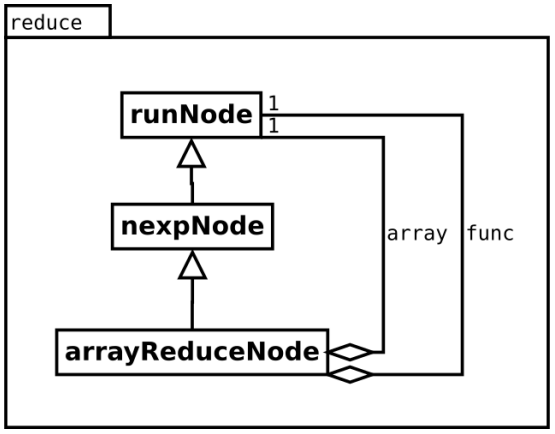
0.1.10. Operadores sobre array



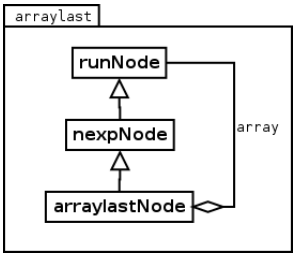
0.1.10.1 Dividir en fragmentos



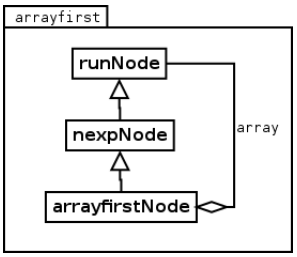
0.1.10.2 Reducir mediante función



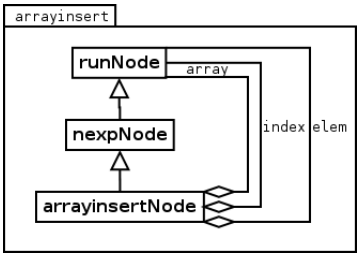
0.1.10.3 Obtener último



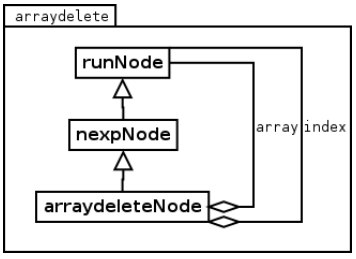
0.1.10.4 Obtener primero



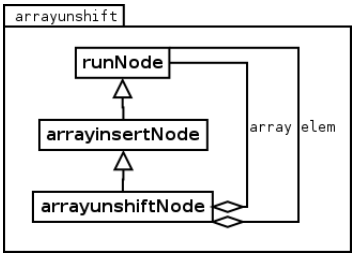
0.1.10.5 Insertar en posición



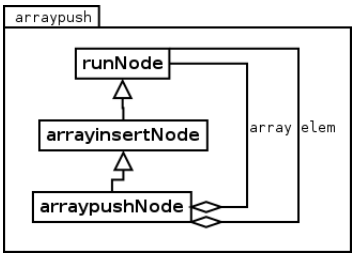
0.1.10.6 Eliminar posición



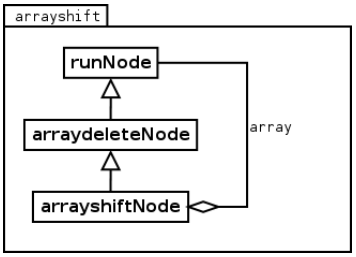
0.1.10.7 Insertar al inicio



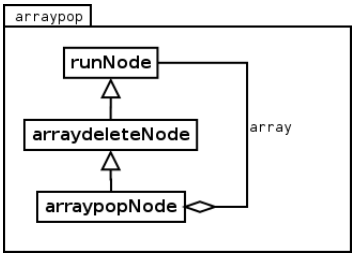
0.1.10.8 Insertar al final



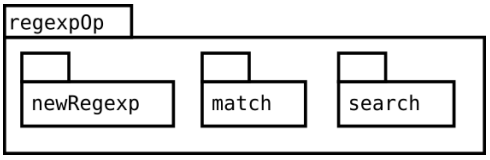
0.1.10.9 Eliminar del inicio



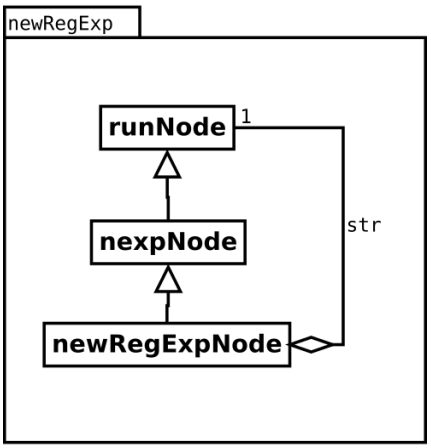
0.1.10.10 Eliminar del final



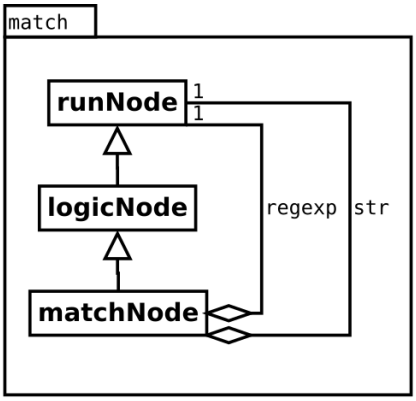
0.1.11. Operadores sobre expresiones regulares



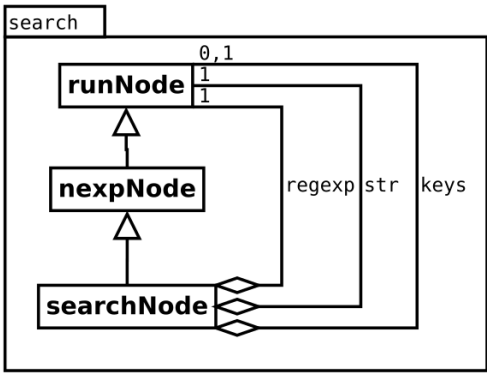
0.1.11.1 Crear expresión regular



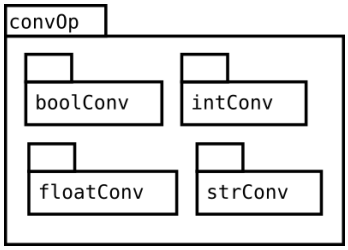
0.1.11.2 match



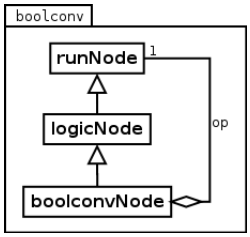
0.1.11.3 search



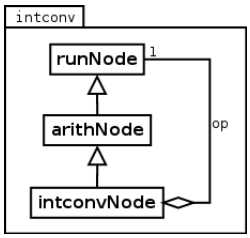
0.1.12. Conversión de tipos



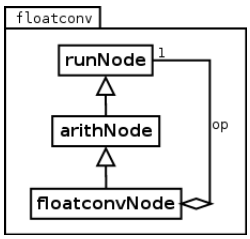
0.1.12.1 Conversión a lógico



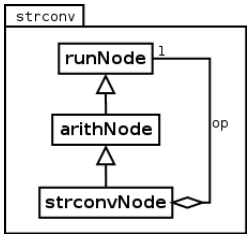
0.1.12.2 Conversión a entero



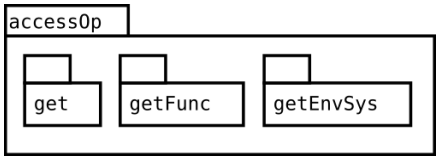
0.1.12.3 Conversión a flotante



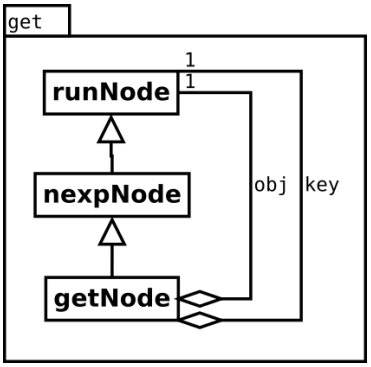
0.1.12.4 Conversión a cadena



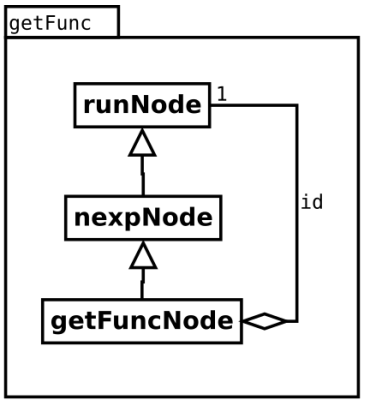
0.1.13. Operadores de acceso



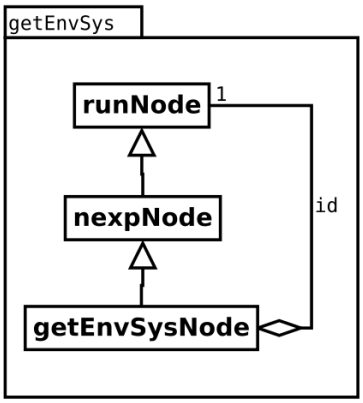
0.1.13.1 Acceso a clave



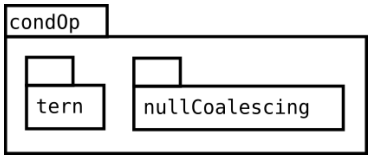
0.1.13.2 Acceso a función



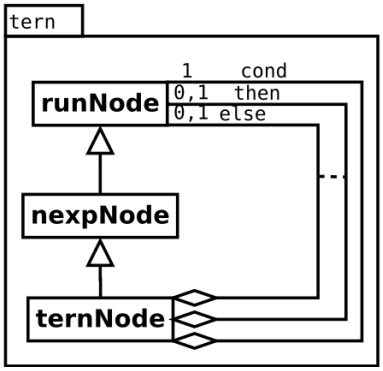
0.1.13.3 Acceso a variable de entorno



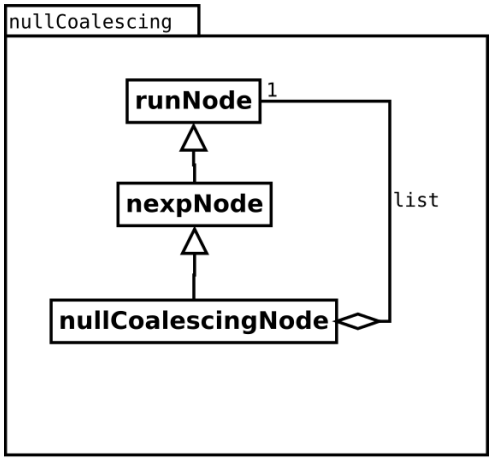
0.1.14. Operadores condicionales



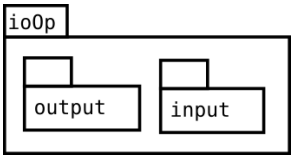
0.1.14.1 Ternario



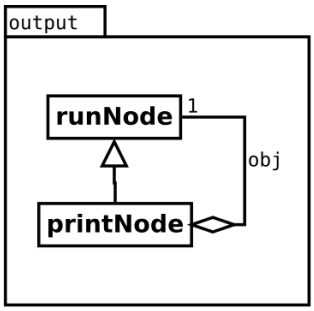
0.1.14.2 Fusión de nulos



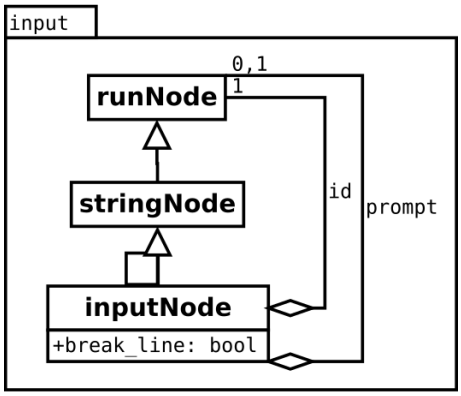
0.1.15. Operadores de entrada/salida



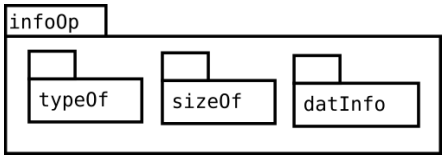
0.1.15.1 Salida estándar



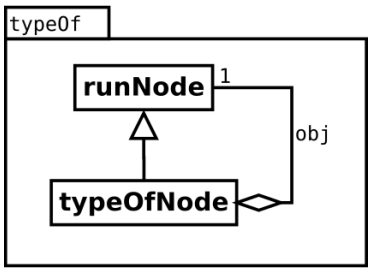
0.1.15.2 Entrada estándar



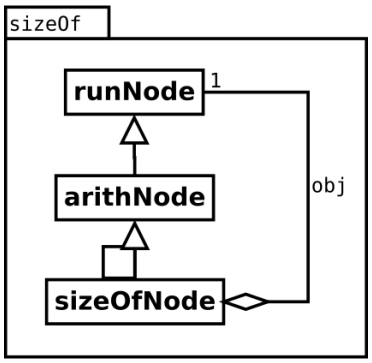
0.1.16. Operadores informativos



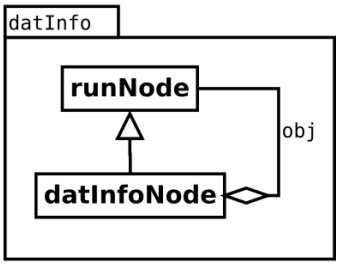
0.1.16.1 Tipo de



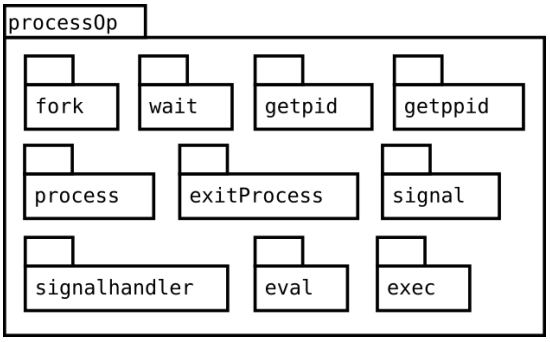
0.1.16.2 Tamaño de



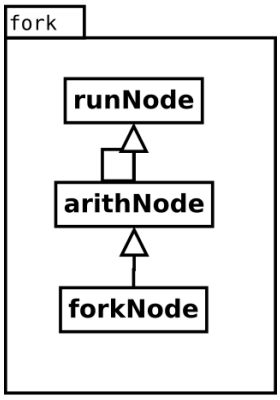
0.1.16.3 Información sobre



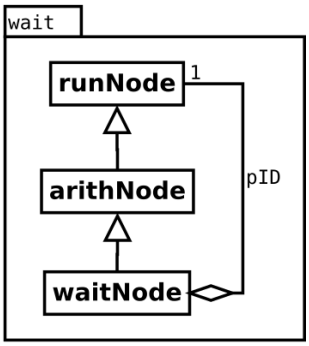
0.1.17. Procesos



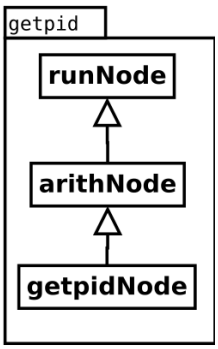
0.1.17.1 Crear proceso



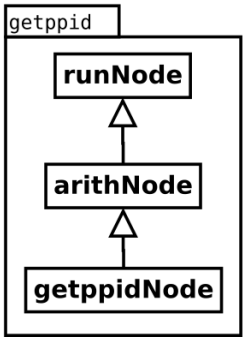
0.1.17.2 Esperar finalización



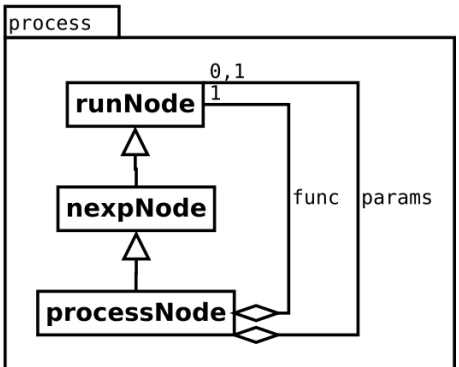
0.1.17.3 Obtener identificador



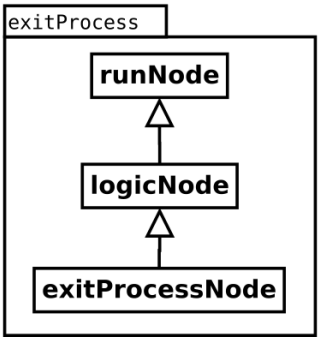
0.1.17.4 Obtener identificador padre



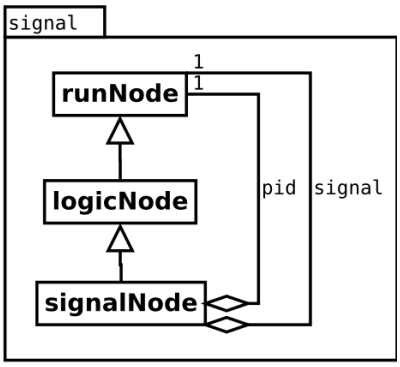
0.1.17.5 Ejecutar como proceso



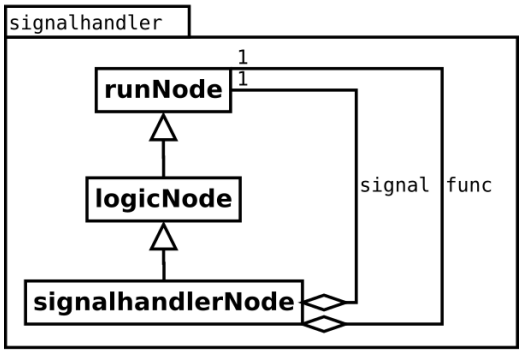
0.1.17.6 Salir de proceso



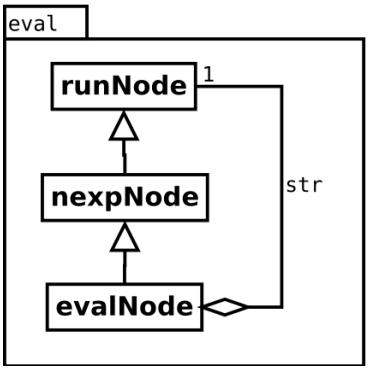
0.1.17.7 Señal a proceso



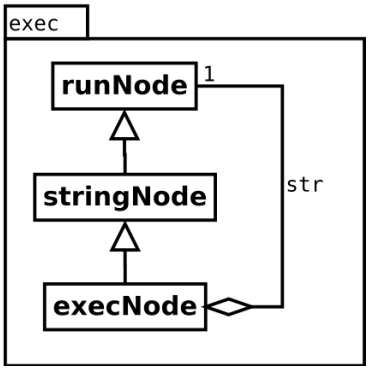
0.1.17.8 Manejador de señales



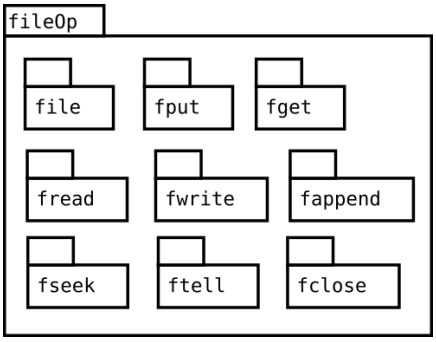
0.1.17.9 Evaluar cadena



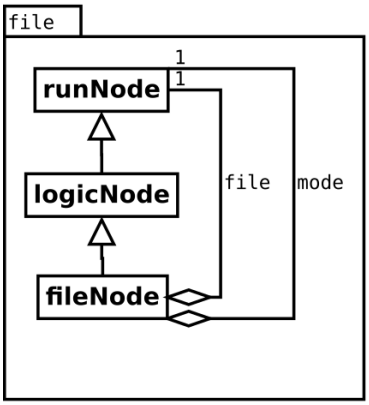
0.1.17.10 Ejecutar comando del sistema



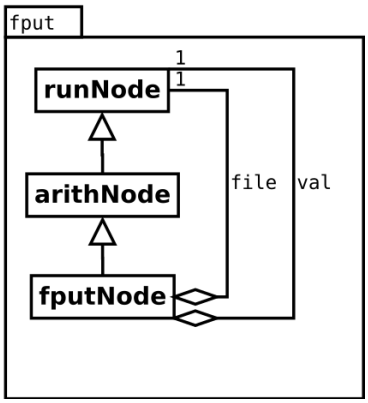
0.1.18. Ficheros



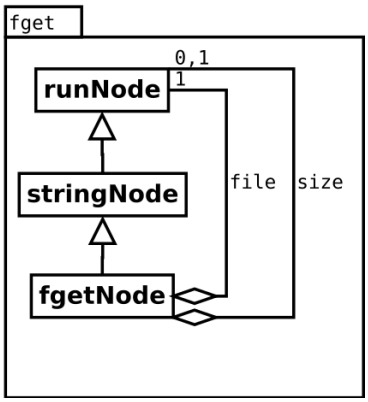
0.1.18.1 Obtener un flujo a fichero



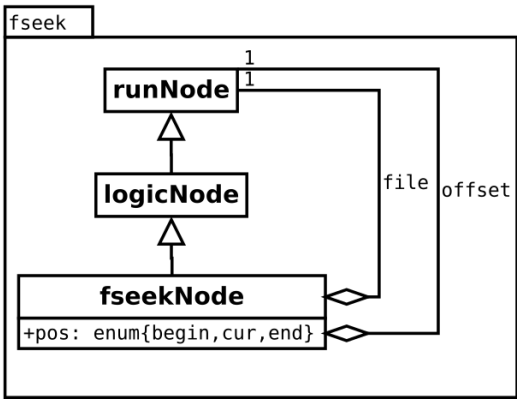
0.1.18.2 Escribir en flujo a fichero



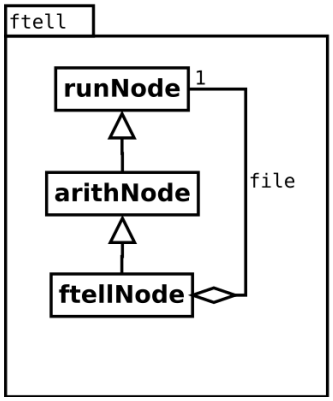
0.1.18.3 Leer de flujo a fichero



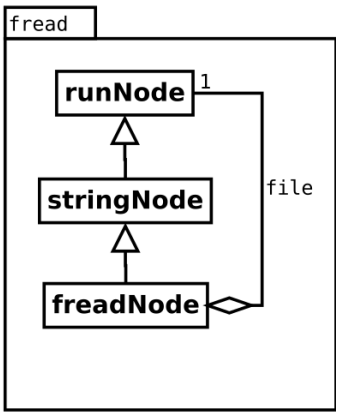
0.1.18.4 Cambiar posición en fichero



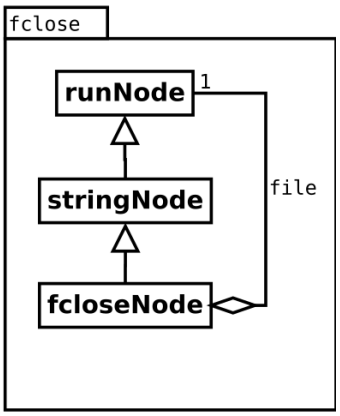
0.1.18.5 Obtener posición en flujo a fichero



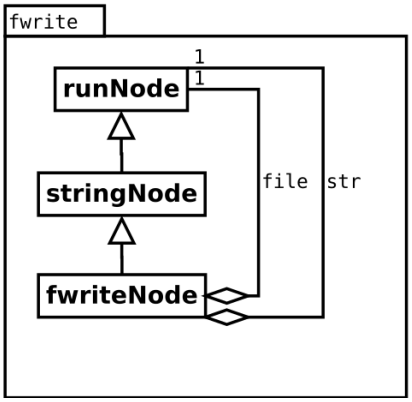
0.1.18.7 Leer fichero



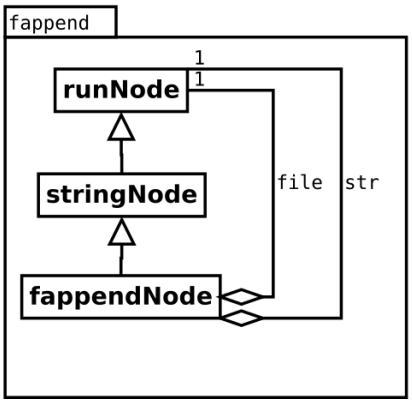
0.1.18.6 Cerrar flujo a fichero



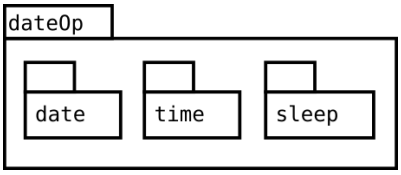
0.1.18.8 Escribir en fichero



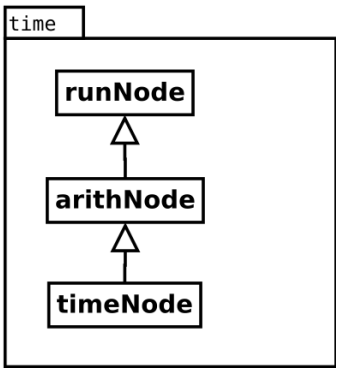
0.1.18.9 Escribir al final de fichero



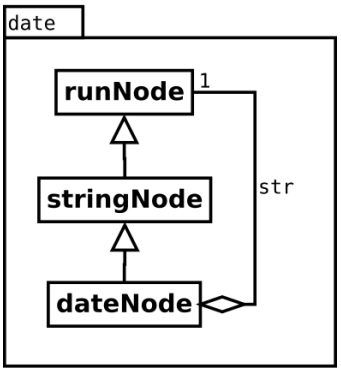
0.1.19. Fechas



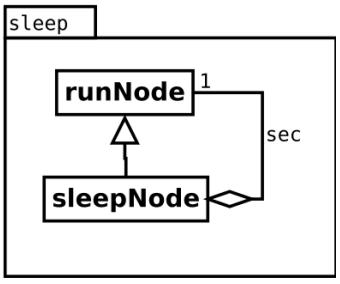
0.1.19.1 Tiempo Unix



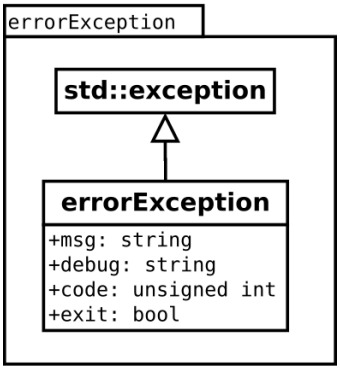
0.1.19.2 Fecha y hora con formato



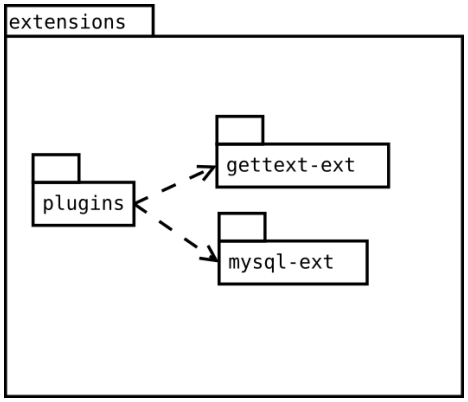
0.1.19.3 sleep

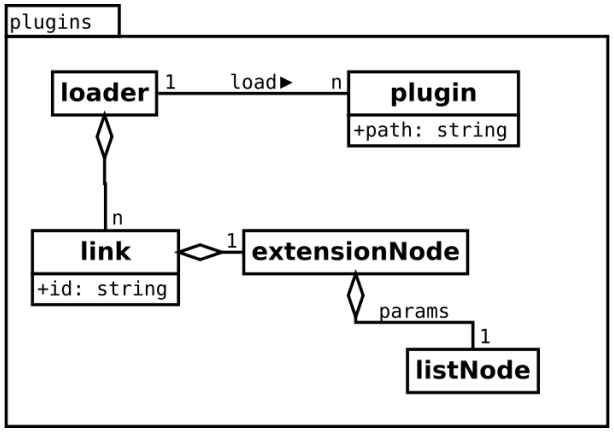


0.1.20. Errores

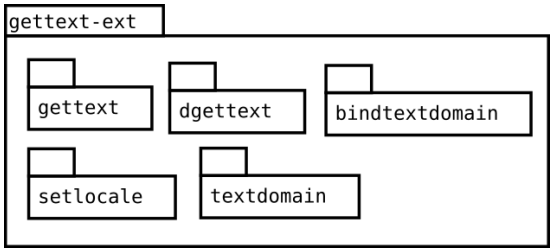


0.1.21. Extensiones

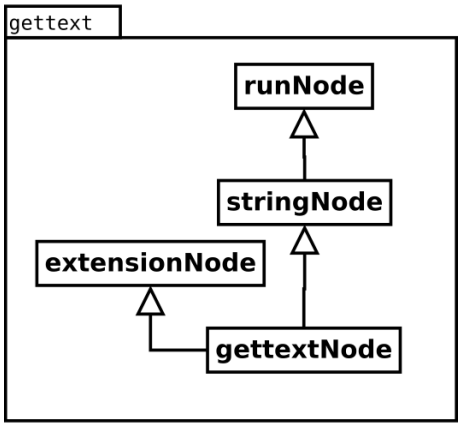




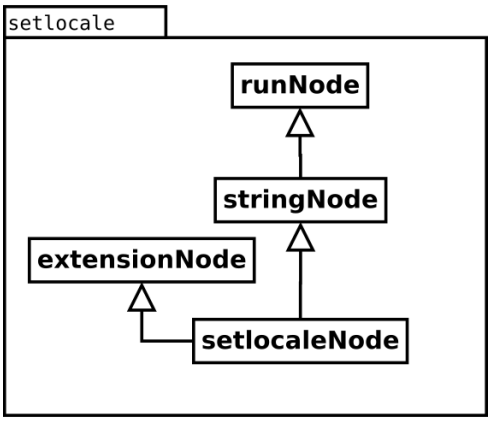
0.1.21.1 Biblioteca GNU de internacionalización (gettext)



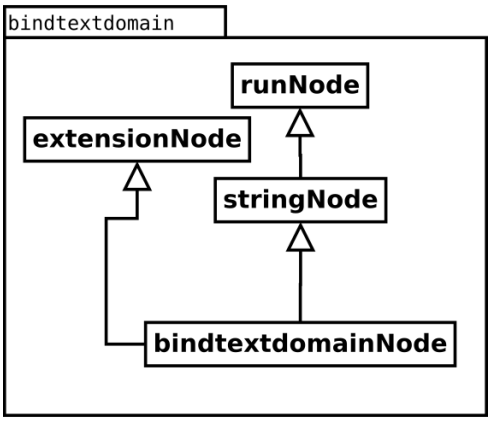
gettext



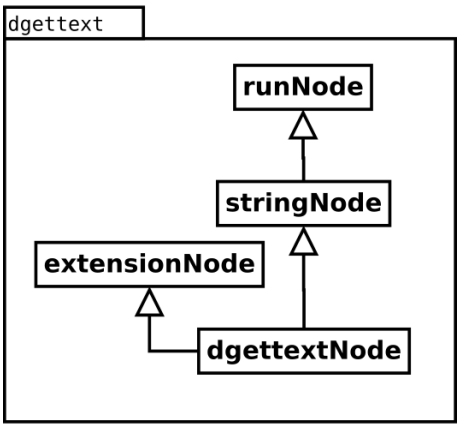
setlocale



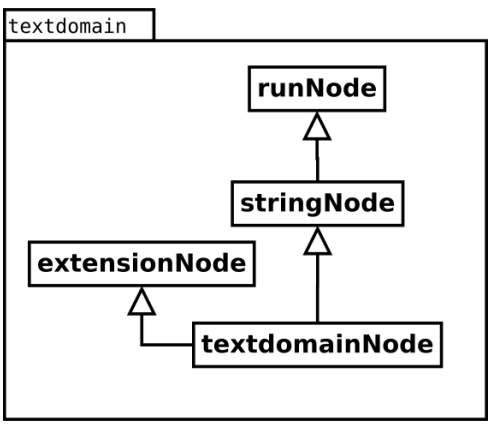
bindtextdomain



dgettext



textdomain



0.1.22. 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.