**UNIVERSIDAD CAECE**

**Procesadores de Lenguajes**

**Trabajo Práctico:**

**ANALIZADOR SEMÁNTICO**

**Integrantes:**

Barcia, Barsotti, Larrá, Trabalón y Salomone

**Docente:**

Amitrano

Noviembre 2014

# 

# Especificación completa de la gramática de

La implementación de dichos nodos se llevó a cabo gracias al siguiente set de clases:

# atributos

Para la generacion de la gramática de atributos, se utilizó la gramática libre de contexto definida en el TP - Análisis Sintáctico y se agregaron atributos a dicha gramática, para transformarla en una de atributos.

## Atributos

La gramática de atributos se divide en dos grandes partes, una con la definición de atributos por cada NO TERMINAL, y luego aplicar validaciones semánticas en base a los atributos previamente calculados. Es importante aclarar, que el cálculo de estados se deriva de las operaciones realizadas en el uso de los atributos.

Los atributos se dividen en dos grupos:

* **Sintetizados:** Son calculados sobre el padre usando atributos de uno o de varios hijos, ya sean heredados o sintetizados. La forma de identificarlos tienen la forma de NO\_TERMINAL.ATRIBUTO = X; siendo X un valor de los hijos, o puramente definido en dicho atributo. Por ejemplo:

Si *A -> BCd* entonces se puede definir el siguiente atributo:

*A.sint = B.sint + B.heredado + d.valor*

* **Heredados:** Se calculan en los no terminales padres o hermanos. Para el caso de los atributos heredados de un padre, se utilizan los atributos heredados. En cambio, para el caso de los atributos heredados de un hermano, se puede utilizar un atributo sintetizado o heredado.

La particularidad de los atributos heredados es que no son posibles de modificar de un padre a un hijo. Por ejemplo:

Si *A -> BCd* entonces se puede definir el siguiente atributo:

*B.heredado = A.heredado*. En este caso, el atributo heredado pasa de padre a hijo.

*C.heredado = B.heredado*. En cambio, el atributo heredado pasa de hermano a hermano.

Para la especificación de la gramática de atributos, tomamos la siguiente nomenclatura:

* *H\_NOMBREATRIBUTO*: Atributo heredado.
* *S\_NOMBREATRIBUTO*: Atributo sintetizado

## Validaciones

La mayoría de las validaciones semánticas son del tipo:

*If CondicionDeError then ErrorSemantico*

Por ejemplo:

* ***Producción****: WHILE -> IterationWhile EXPRESION IterationDo CUERPO IterationEnd*
* ***Validacion****: If EXPRESSION.S\_Tipo <>'Boolean' then ErrorWhileCondition*

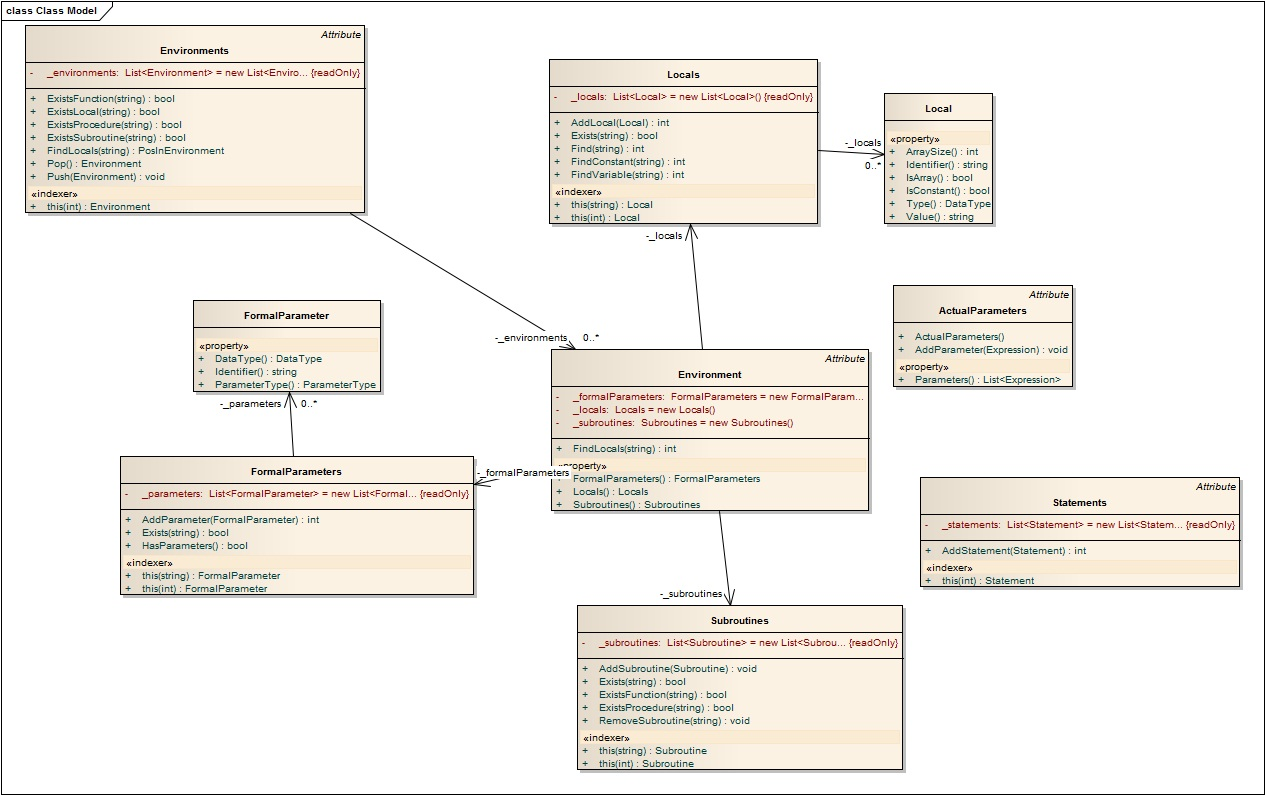
El ejemplo de la validación, significa que el tipo del no terminal Expression sea distinto de Boolean, ya que solamente puede haber una condición booleana en el WHILE.

La especificación de la gramática de atributos es la siguiente:

*Ver archivo excel llamado Gramatica\_de\_atributos.xls.*

# Especificación de la implementación de la gramática de atributos

Para la implementación se utilizó otra estructura distinta a la implementada en la gramática de atributos del punto 1. Dada las ventajas arquitectónicas de C#, se decidió implementar el siguiente set de clases para la generacion de un Entorno:



***Environments*** es una lista de ***Environment***, donde por cada Entorno generado en el código fuente, se genera una instancia de ***Environment*** y se agrega a la lista de ***Environments.***

La clase ***Environment*** implementa tres listas: ***Locals***, ***Parameters*** y ***Subroutines***. El objetivo

de contar con dicha clase era el pasaje del Entorno entre las distintas clases *parsers.*

Las clases definidas en ***Environment*** buscan:

* *Locals:* definir las variables, constantes y arrays en dicho entorno, siendo estas visibles para los entornos hijos. La lista de *Locals,* tiene nodos *Local* que cuentan con atributos como *Value, Type y Identifier*. La lista *Parameters* cuenta con todos los metodos para el tratamiento de las variables, como *Exists* que valida la existencia de una variable, constante o array en un Entorno.
* *Parameters:* define los parámetros asociados a los *subroutines*. La lista de *Parameters* tiene nodos *Parameter* que cuentan con atributos como Identifier, *DataType o Position.* La lista *Parameters* implementa métodos como *AddParameters* que agrega parámetros a medida que se va “parseando”.
* *Subroutines*: define todos los procedimientos y funciones en dicho entorno.

Las clases **Parsers** de la interface ***IParseable*** fueron modificadas agregando un parámetro attributes de la clase ***Attributes***. Dicha clase, implementa métodos como ***AddAttribute***, ***RemoveAttribute*** que facilitaron la implementación de los atributos a través de los *parsers*.

A su vez, se modificaron para aplicar las validaciones sematicas descriptas en el punto 1

Por ejemplo:

*var function = exp as FunctionCallExpression;*

*if (!environments.ExistsFunction(function.Identifier))*

*{*

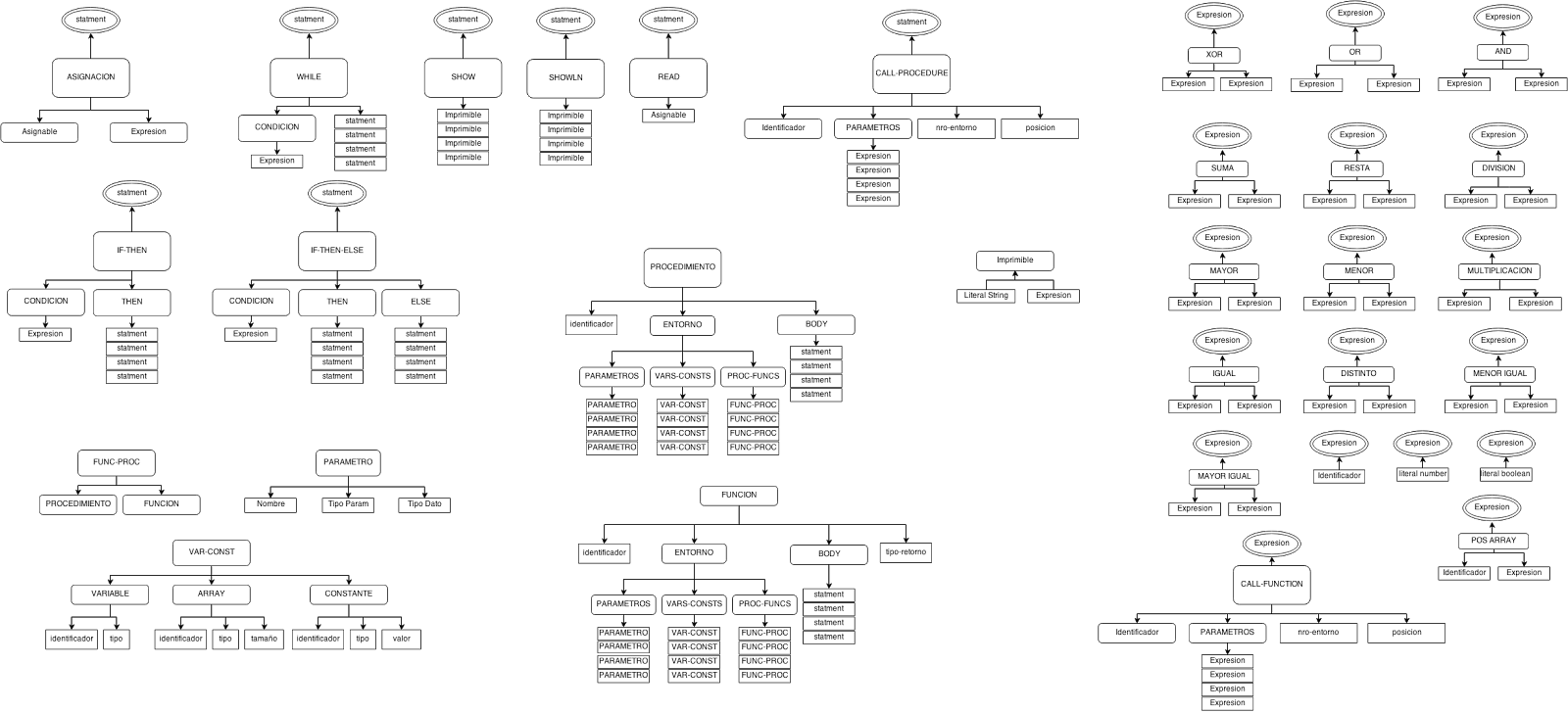
*LogIdentifierNotFound(function.Identifier, token);*

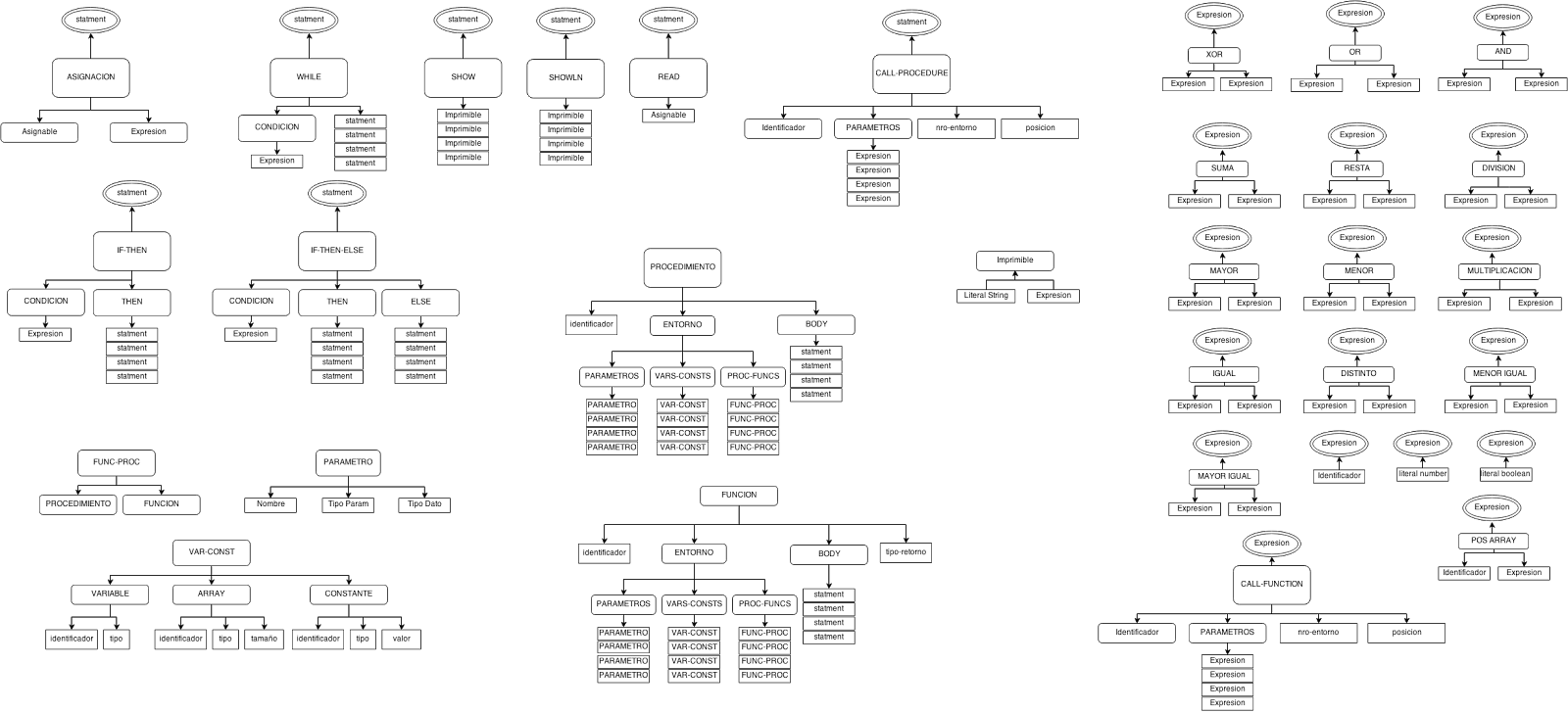
*successfullyParsed = false;*

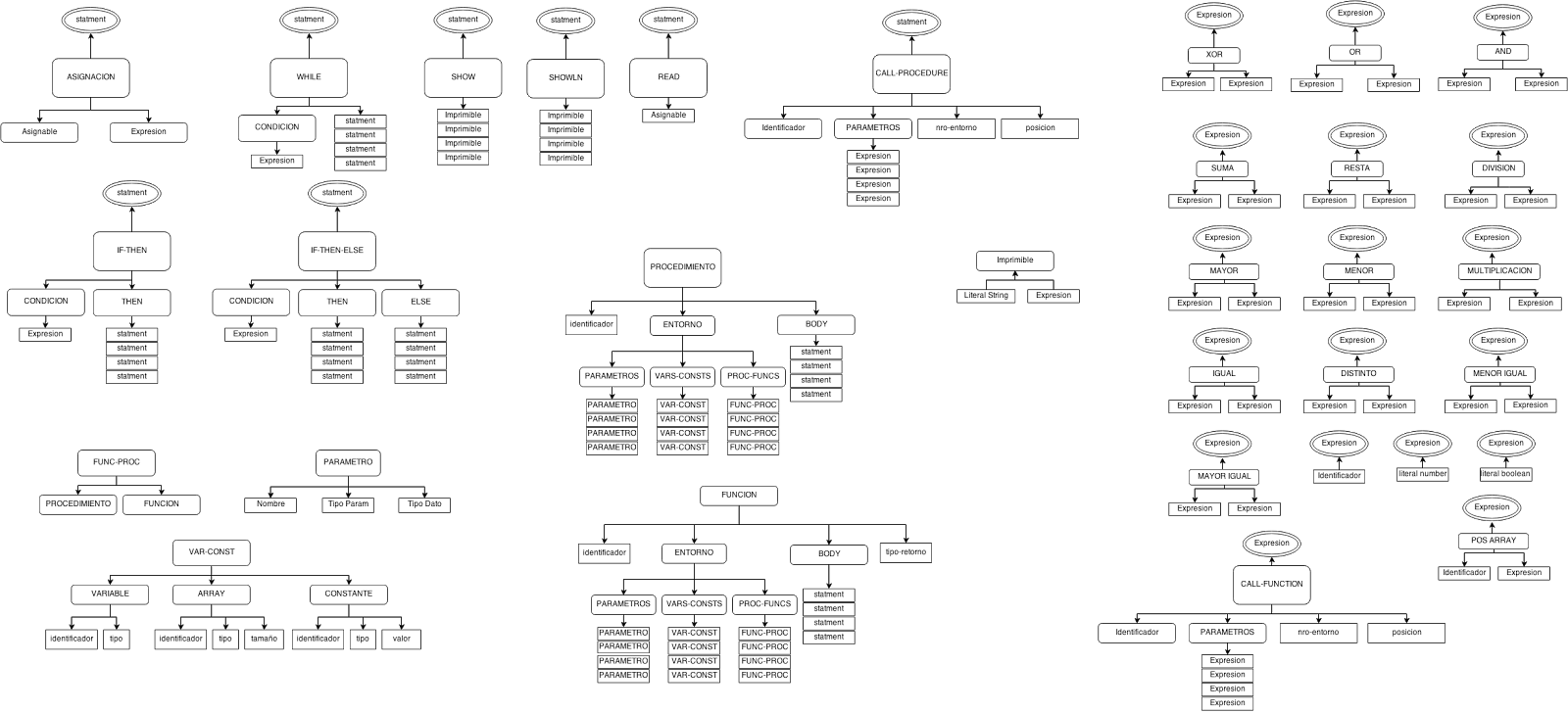
*}*

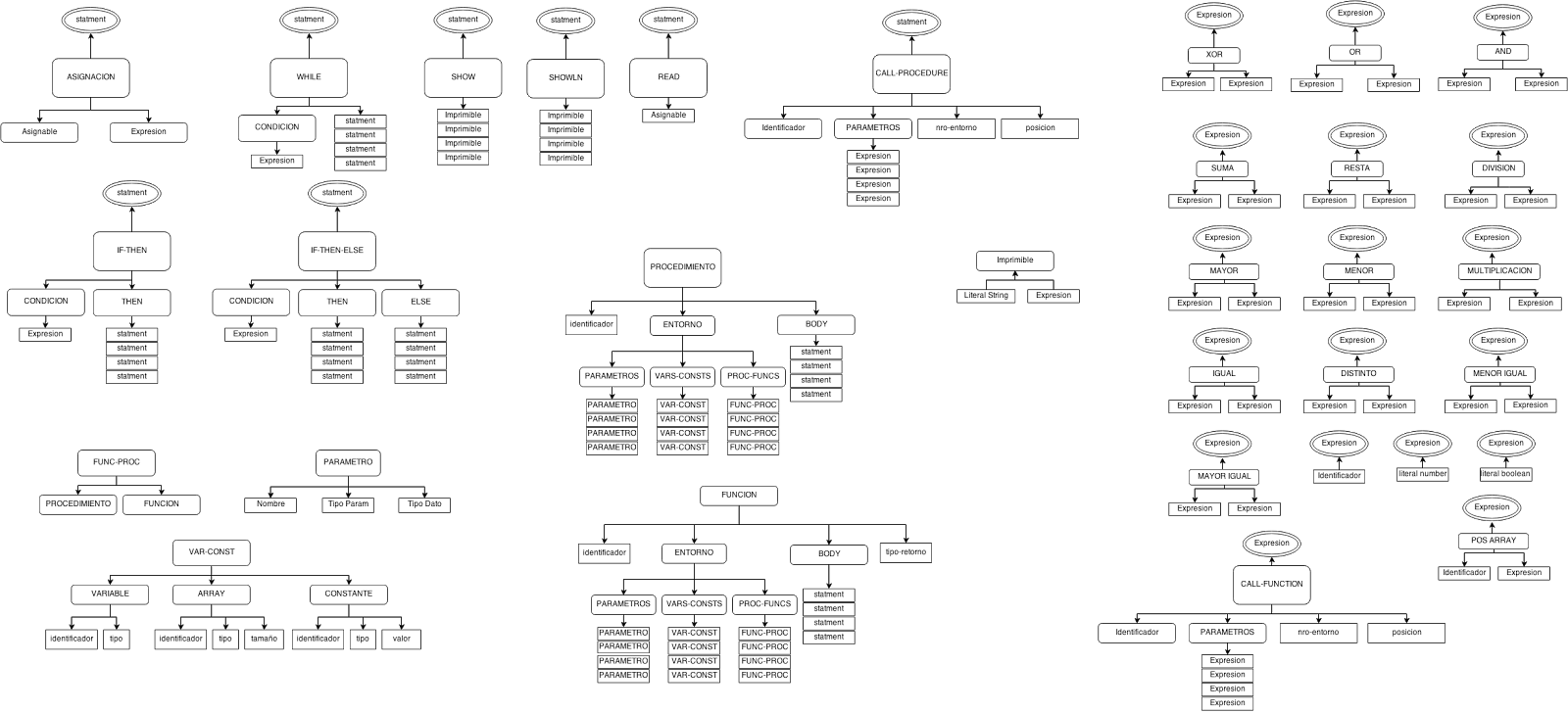
Tal como se ve en el ejemplo, se utilizan las clases generadoras de árboles y luego, se aplican validaciones sobre los mismos.

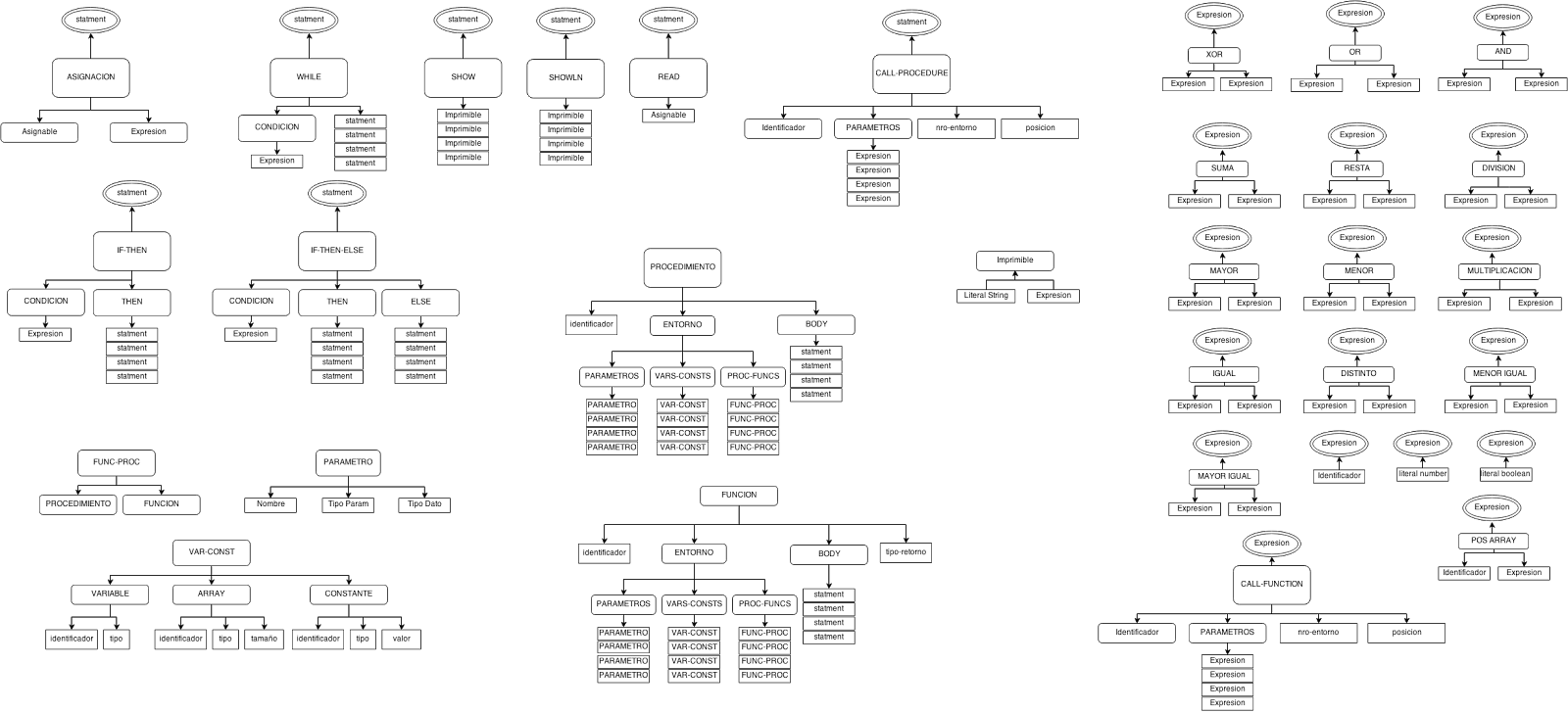
Para la generacion del árbol semántico se utilizó el siguiente esquema de nodos:

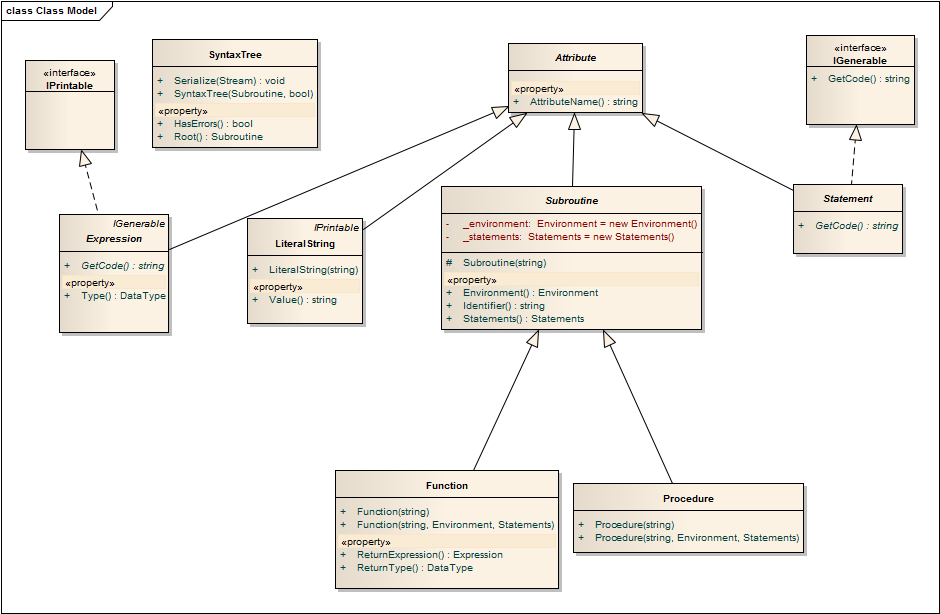






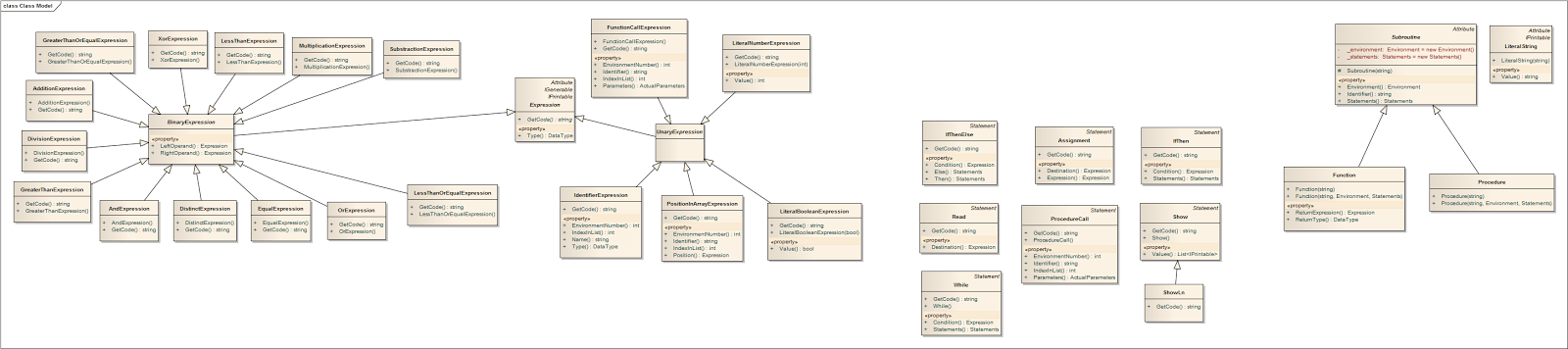


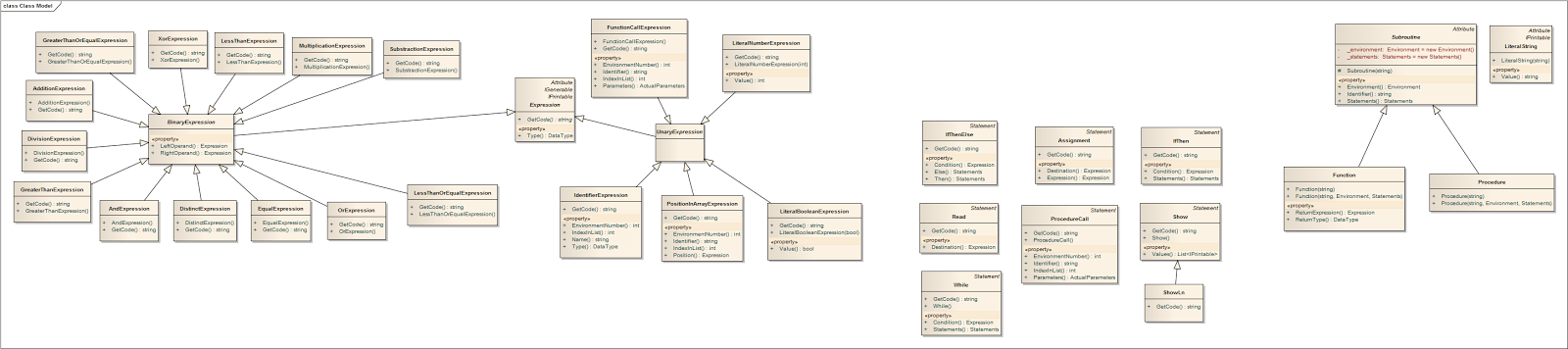




**SyntaxTree** es el árbol N-ario, que solamente contiene el nodo raíz y contiene la implementación del método Serialize, cuyo objetivo es convertir una clase en un tipo XML. La ventaja de esta implementación es que la salida del árbol generado puede armarse en un archivo XML que facilita la lectura y comprension del mismo.

**SubroutineNode** es el nodo correspondiente a procedimientos y funciones y **StatementNode** que implementa la interfaz **IGenerable**.





La implementación de estas clases sirvió para poder relacionar cada parser con un nodo, es decir que el parser ***If***va a tener su generador de árbol ***If***que respetará el esquema planteado anteriormente.

# Explicar el mecanismo de recuperación de errores implementado.

Se mantuvo gran parte el mecanismo de errores del analizador sintáctico, salvo por las siguientes modificaciones:

* El método parse (que fue definido en la entrega 2 para el analizador sintáctico) implementado en cada clase iParseable fue modificado con el fin de devolver en su llamado si el parseo fue sintáctica y/o semánticamente satisfactorio. Este retorno es almacenado en una variable que persiste durante la ejecución y marca al árbol con o sin errores.
* En caso de no pasar las validaciones semánticas, se informan todos los errores. El árbol se genera con todos los nodos del programa pero se informa en la cabecera del árbol que el mismo tiene errores.
* Se agregó un “logger” de errores que informa en que nodo, fila, columna y el token más cercano donde existió un error semántico. Este punto facilitó el bugfixing debido a la complejidad de la lectura del arbol.

**Listado de mensajes:**

|  |  |
| --- | --- |
| **ID del mensaje** | **Descripción del mensaje** |
| ArrayCantBeDeclaredOutsideGlobalEnvironmentMessage | [{0},{1}] SEM: No se puede declarar un array fuera del entorno global |
| ArrayCantBeSubroutineParameterMessage | [{0},{1}] SEM: En la llamada a '{2}' no se puede pasar un array como parámetro |
| ConstantIsNotAssignableMessage | [{0},{1}] SEM: No se puede hacer una asignación a una constante |
| DuplicateIdentifierFoundMessage | [{0},{1}] SEM: Ya existe un identificador con el nombre '{2}' |
| ExpressionIsNullMessage | [{0},{1}] SEM: La expresión no puede ser nula |
| FunctionNotFoundMessage | [{0},{1}] SEM: No se encontró una función con nombre '{2}' |
| IdentifierNotFoundMessage | [{0},{1}] SEM: No se encontró el identificador '{2}' |
| InvalidArrayIndexMessage | [{0},{1}] SEM: El indice de un vector debe ser una expresión entera |
| InvalidAssignableTypeMessage | [{0},{1}] SEM: La expresión no corresponde a un tipo asignable |
| InvalidExpressionTypeMessage | [{0},{1}] SEM: Se esperaba una expresión '{2}' y se encontró una del tipo '{3}' |
| InvalidParametersCountOrTypeMessage | [{0},{1}] SEM: La cantidad de parametros proporcionados o el tipo de los mismos no coinciden con la definición de '{2}' |
| InvalidPrintableTypeMessage | [{0},{1}] SEM: Se esperaba un imprimible y se encontró un '{2}' |
| MainHasLocalEnvironmentMessage | SEM: El procedimiento MAIN no puede tener un entorno local |
| MainHasParametersMessage | SEM: El procedimiento MAIN no puede tener parámetros |
| MainNotFoundMessage | SEM: No se encontró el procedimiento MAIN |
| ProcedureNotFoundMessage | [{0},{1}] SEM: No se encontró un procedimiento con nombre '{2}' |
| TypeMismatchMessage | [{0},{1}] SEM: No se pueden comparar una expresión de tipo '{2}' con una de tipo '{3}' |
| UnexpectedTokenMessage | [{0},{1}] SIN: Se esperaba {2} y se encontró '{3}' |

Donde {0} es el número de linea y el {1} es el número de columna. Los demás índices son parámetros de los mensajes.

**Ejemplo del errores logueados:**

[13,9] SEM: No se encontró el identificador 'X'.

[13,8] SEM: La expresión no puede ser nula.

[11,17] SEM: Se esperaba una expresión 'Boolean' y se encontró una del tipo 'Integer'

SEM: El procedimiento MAIN no puede tener parámetros.

SEM: El procedimiento MAIN no puede tener un entorno local.

# 

# 

# Otros comentarios que crean necesario realizar acerca de la implementación.

Para el armado del archivo XML, que contiene el árbol generado según el input del programa con las validaciones tanto léxicas, sintácticas como semánticas, se utilizó la biblioteca Serialization que ya cuenta con métodos para el armado de un archivo XML en base a los atributos y métodos de una clase. Esto facilitó mucho la generacion del árbol y su validación para el bug fixing, en base a los nodos generados.

Veamos un ejemplo de dicha implementación.

Con el siguiente código:

const M : integer = 7;

var N : integer;

function FUN1(T:integer) : integer;

begin

end-func T\*5;

procedure MAIN();

begin

N := FUN1(M);

if(N>20) then

show 'N es mayor a 20 y vale ', N;

else

show 'N es menor a 20 o igual y vale ', N;

end-if;

end-proc;

Nuestro código se estructura de un enviroment global con la definición de 1 constante y 1 variable. Además, tiene la definición de una función que retorna un valor entero.

Dicha función recibe como parámetro un entero, que luego lo usa para su retorno multiplicándolo por 5.

Luego, nuestro procedimiento Main utiliza la variable definida anteriormente para asignarle una llamada a la función pasándole como parámetro la constante definida.

Con el valor asignado, se verifica que dicho valor sea mayor a 20 en un if, imprimiendo por pantalla un valor si es verdadero y otro valor si es falso.

Para ver el árbol generado, deberá abrir el adjunto example.xml

# 

# Código fuente y ejecutable del analizador semántico.

Ver adjuntos:

* Entrega 3.zip

# Casos de prueba

Se crearon varios archivos para ejecutar los tests, estos son:

* ArrayIndiceExpresionNoEntera.txt
* AsignacionDistintoTipoDatoAsignable.txt
* AsignacionExpresionAConstante.txt
* AsignacionIdentificadorFueraDeScope.txt
* AsignacionIdentificadorInexistente.txt
* DeclaracionDeArrayFueraDeEntornoGlobal.txt
* DeclaracionParametrosDuplicados.txt
* FuncionInexistente.txt
* FuncionRetornoNoCoincideConTipoDatoRetorno.txt
* FuncionUsaParametros.txt
* IdentificadorDuplicadoDiferenteEntorno.txt
* IdentificadorDuplicadoMismoEntorno.txt
* IfCondicionFuncionBoolean.txt
* IfCondicionFuncionInteger.txt
* IfCondicionNoBooleana.txt
* IfCondicionProcedimiento.txt
* LlamadasRecursivas.txt
* LlamarProcEnEntornoLocal.txt
* MainConEntornoLocal.txt
* MainConParametros.txt
* OperacionBinariaDistintosTipoDeDatos.txt
* OperacionDivisionNoEnterosEnCondicion.txt
* OperacionMayorIgualNoEnterosEnCondicion.txt
* OperacionMayorNoEnterosEnCondicion.txt
* OperacionMenorIgualNoEnterosEnCondicion.txt
* OperacionMenorNoEnterosEnCondicion.txt
* OperacionMultiplicacionNoEnterosEnCondicion.txt
* OperacionRestaNoEnterosEnCondicion.txt
* OperacionSumaNoEnterosEnCondicion.txt
* OperadorAndNoBooleanoEnCondicion.txt
* OperadorOrNoBooleanoEnCondicion.txt
* OperadorXorNoBooleanoEnCondicion.txt
* PasajeParametrosDistintoTiposSegunVariable.txt
* PasajeParametrosFueraDeScope.txt
* PasajeParametrosIdentificadorInexistente.txt
* PasajeParametrosMenosParametros.txt
* PasajeParametrosTipoArray.txt
* ProcedimientoInexistente.txt
* ReadIdentificadorInexistente.txt
* ReadIdentificadorTipoDatoErroneo.txt
* RefVarEnEntornoGlobal.txt
* RefVarEnEntornoLocal.txt
* RefVarEnEntornoPadre.txt
* ShowFuncionInexistente.txt
* ShowIdentificadorInexistente.txt
* WhileCondicionFuncionBoolean.txt
* WhileCondicionFuncionInteger.txt
* WhileCondicionNoBooleana.txt
* WhileCondicionProcedimiento.txt

# Conclusiones de la implementación de la etapa actual.

Continuando con el suite de compilación, se agregó la tercera etapa de análisis semántico que son necesarias luego del análisis léxico. En esta etapa, en analizador sintáctico planteado en la Entrega 2 fue modificado para convertirse en un analizador sintactico - semantico, es decir que la implementación del primero debió ser modificada para soportar todas las validaciones semánticas nuevas que fueron siendo agregadas.

En primer lugar, la gramática libre de contexto planteada en dicha entrega no pudo mantenerse debido a que, por sus características no cuenta con lo necesario para soportar validaciones del tipo semánticas, como podria ser, chequeo de tipos de variables. Con dicha modificación, la gramática se convirtió en una gramática de atributos, donde se buscó generar atributos para los no terminales que posibiliten dichas validaciones, siendo estos atributos heredados o sintetizados.

En segunda instancia, luego de haber generado la gramática de atributos, era necesario contar con un árbol que dé soporte a dichas validaciones y a la generacion de codigo futuro, planteado para la proxima entrega.

Luego de haber planteado de manera teórica la gramática de atributos y el esquema de nodos, se diseñaron varias clases nuevas cuyas implementaciones fueron descritas en este documento, y la modificación de las clases *parsers*.

Con respecto a la implementación del analizador sintáctico, pudo mantenerse toda la implementación de los parsers planteados en la Entrega 2, y solamente modificandolos para agregar las validaciones semánticas de este Entrega. Se mantuvo el análisis Left - Right de las gramáticas y el input de datos desde el analizador léxico.