ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE COMPUTADORES

Lenguaje específico de dominio para el procesamiento de curvas de consumo energético

Domain-specific language for energy consumption curves processing

Autor:

Ignacio Javier Villalobos Salas

Tutorizado por:

Enrique Alba Torres

José Francisco Chicano García

Departamento:

Lenguaje y ciencias de la computación

Universidad de Málaga

Fecha defensa: Julio 2017

El Secretario del Tribunal

Resumen: El cambio climático es un gran problema a nivel global. Esto preocupa tanto a ciudadanos como a científicos que ponen sus esfuerzos en reducir las emisiones contaminantes y el alto consumo energético de nuestras ciudades y edificios. Para ello los científicos necesitan herramientas potentes para realizar sus investigaciones y poder mejorar la eficiencia energética.

Por estas razones, se crea un lenguaje específico de dominio, con su intérprete que dota a los científicos de herramientas potentes y simples para trabajar con curvas de consumo energético y así poder realizar cálculos sobre ellas fácilmente sin necesidad de saber cómo realizar dichos cálculos. El lenguaje estará completamente integrado en el editor Eclipse y se va a utilizar para su desarrollo “Xtext” y “Java” tanto para el lenguaje como para el intérprete. El intérprete utilizará ficheros con extensión “.ecc” de forma que se podrán guardar códigos fácilmente para tareas concretas y reutilizarlos. De esta forma se quiere facilitar el trabajo de los científicos mediante herramientas simples, pero a su vez potentes.

Palabras clave: Java, DSL, lenguaje específico de dominio, Xtext, Xtend, intérprete, eclipse, plugin.

Abstract: Climate change is a big problem at the global level. This concerns both citizens and scientists who put their efforts to reduce pollutant emissions and the high energy consumption of our cities and buildings. To do this, scientists need powerful tools to carry out their research and improve energy efficiency.

For these reasons, a specific domain language is created with its interpreter, that provides scientists with powerful and simple tools to work with energy consumption curves so that. They can easily perform computations on them easily without having to know how to perform those calculations. The language will be fully integrated in the Eclipse editor and will be used for the development "Xtext" and "Java" of both, the language and the interpreter. The interpreter uses files with extension ".ecc" which allows the code to be reused and easily saved for specific tasks. In this way, we want to facilitate the work of scientists using simple but powerful tools.

Keywords: Java, DSL, domain-specific-language, Xtext, Xtend, interpreter, eclipse, plugin.

Índice

Capítulo 1
Introducción

Actualmente, pocas dudas quedan en la comunidad científica del protagonismo que la actividad humana está teniendo en el cambio climático. Por este motivo, muchos países han puesto en marcha programas para reducir la influencia de la humanidad en dicho cambio. Dos ejemplos de especial relevancia son el acuerdo de París [1], donde 195 países acodaron poner en marcha políticas para evitar que la temperatura media global suba 2º C con respecto a épocas preindustriales, y el paquete de medidas sobre clima y energía hasta 2020 [2], en el que la Unión Europea propone como objetivos reducir un 20% la emisión de gases de efecto invernadero (con respecto a 1990), consumir un 20% de energías renovables y mejorar en un 20% la eficiencia energética, todo ello en 2020.

Para poder conseguir una mejora en la eficiencia energética es necesario, en primer lugar, conocer el perfil de consumo de los usuarios. Por esta razón, la normativa actual española (derivada de la europea), obliga a las grandes empresas a realizar auditorías energéticas [3], donde se analiza el perfil de consumo de sus edificios y se señalan posibles acciones que permitan a dichas empresas reducir el consumo energético. Las empresas auditoras, por su parte, instalan un pequeño dispositivo en el cuadro eléctrico principal de los edificios que usa para informar telemáticamente de la potencia consumida en dicho edificio cada cierto tiempo (normalmente 15 minutos). La curva de potencia consumida frente al tiempo es lo que se conoce como *curva de consumo energético* o *curva de carga* (ver Figura 1).

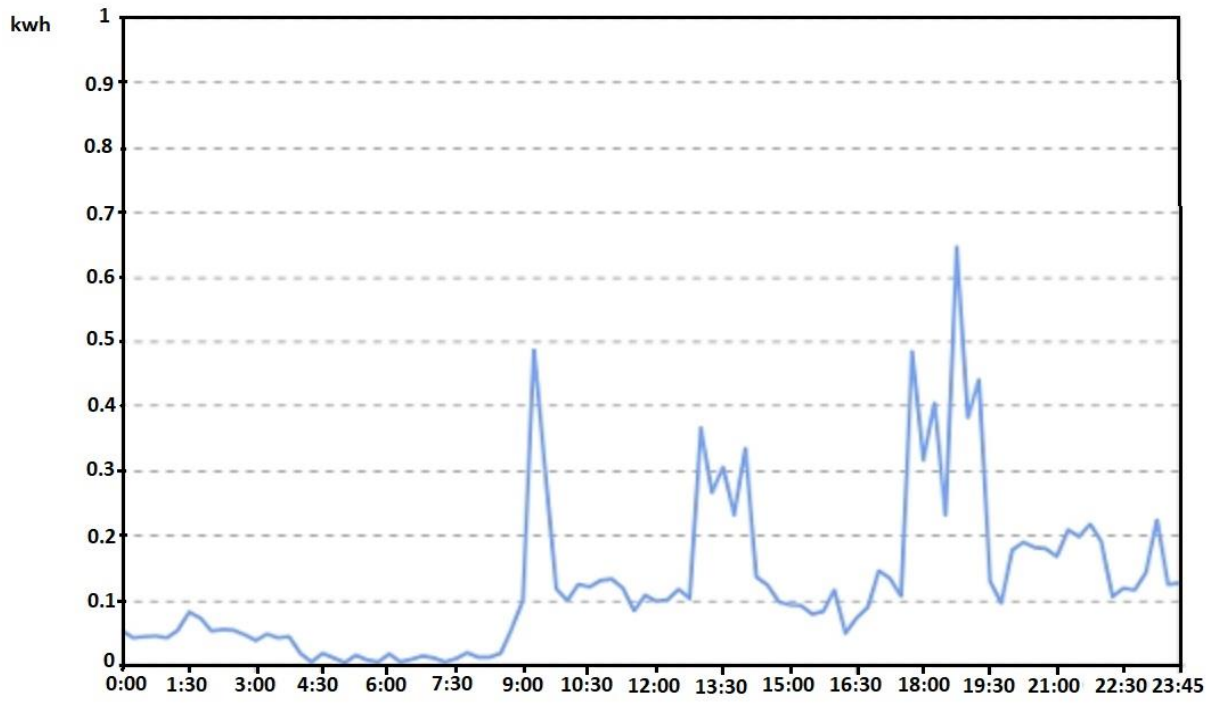

Figura 1: Ejemplo de curva de carga. Se muestra la potencia en Kw/h durante un día.

Las curvas de consumo energético son, con frecuencia, la única fuente de información con la que cuentan las compañías auditoras para realizar el análisis y sugerir acciones de mejora de reducción de consumo. Los cálculos realizados sobre las curvas de consumo incluyen estadísticos básicos como la media, mediana, máximo y mínimo consumo durante un período de tiempo (un día, un mes, un año, etc.), pero también cálculos más sofisticados que requieren considerar la curva como un objeto matemático: promedio de curvas de consumo, cálculo de derivadas de la curva de consumo, etc.

Paquetes software de tratamiento matemático como Matlab, Mathematica o R pueden usarse para crear scripts que permitan hacer los cálculos necesarios, pero debido a su generalidad, presentan una curva de aprendizaje muy lenta para los expertos en energía, que están familiarizados con los conceptos propios de su dominio (curvas de consumo y sus estadísticas) pero no con los detalles de lenguajes de programación funcional (como el que ofrece Mathematica) o imperativos (presentes en Mathematica y Matlab). Además, las curvas de consumo están a veces incompletas, debido a que se producen errores de transmisión de información que pueden provocar que haya datos ausentes o erróneos. Esto impide que se puedan considerar como vectores y requiere establecer una estrategia para tratar los datos ausentes.

1.1 Objetivos

El objetivo del presente Trabajo de Fin de Grado es el desarrollo de un lenguaje especifico de dominio, su intérprete y realizar las pruebas necesarias para asegurar su funcionamiento. Estos deben estar completamente integrados en un entorno de desarrollo moderno para su fácil utilización. El lenguaje incorporará tres tipos de datos básicos: cadenas de caracteres, números reales y un tipo específico para las curvas de carga. El intérprete será capaz de realizar cálculos básicos y estadísticos sobre las curvas que devuelvan un número real como resultado. También tendrá operaciones sobre las curvas que devuelvan otra curva, como, por ejemplo, derivadas numéricas, integrales numéricas, alisado exponencial y la segmentación de una curva en varias. Durante el capítulo cuatro en la sección de diseño tendremos explicadas todas las operaciones de forma detallada.

Por otro lado, se ofrecerá el acceso sencillo a ficheros de datos en formato CSV que contenga los datos de una curva de carga y el tratamiento de esta. Además, se podrán sumar, restar, unir y calcular promedios de varias curvas de consumo eléctrico.

Finalmente, el lenguaje tendrá estructuras de control de un lenguaje de programación como son bucles (while y for) y condicionales (if).

1.2 Estructura de la memoria

La memoria se compone de siete capítulos que son los siguientes:

Capítulo 1: Introducción.
En este capítulo se habla de la motivación para realizar este proyecto, así como de la motivación y de la estructura de la memoria.

Capítulo 2: Especificaciones y análisis.
En este capítulo hablaremos de los requisitos funcionales y no funcionales que requiere para cumplir los requisitos de calidad de nuestro proyecto.

Capítulo 3: Tecnologías utilizadas.
En este capítulo se explicarán las herramientas y tecnologías usadas, así como los motivos por los cuales se han escogido estas como las más adecuadas en el desarrollo del proyecto.

Capítulo 4: Diseño e implementación.
En este capítulo se hablará sobre el diseño de la gramática mostrando y explicando cada elemento de esta, así como su implementación. También se mostrará mediante diagrama de clases y tablas el diseño y la implementación del intérprete y como este se integra en el editor Eclipse.

Capítulo 5: Pruebas de la gramática y del intérprete.
En este capítulo se mostrarán las pruebas realizadas para comprobar el correcto funcionamiento del lenguaje como del intérprete.

Capítulo 6: Aplicaciones reales.
En este capítulo se mostrarán ejemplos reales de uso de nuestro lenguaje específico de dominio para realizar cálculos y obtener resultados útiles.

Capítulo 7: Conclusiones y líneas futuras.
En este capítulo de hablará de las conclusiones sacadas del proyecto, así como de lo aprendido en este, también se habla de posibles futuras líneas para ampliar y mejorar tanto el lenguaje como el intérprete.

También se incluye un apéndice con un manual de usuario donde se explica con detalle cómo utilizar tanto el lenguaje como la gramática.

Capítulo 2
Especificación y análisis

Este capítulo describe los requisitos funcionales y no funcionales del lenguaje especifico de dominio y del intérprete.

2.1Requisitos funcionales

En este apartado se especifican los requisitos funcionales para el lenguaje y para el intérprete, así como una breve explicación de cada uno. Estos requisitos especifican la funcionalidad del sistema, y son nombrados y descritos en la Tabla 1.

Tabla 1 – Requisitos funcionales

ID	Nombre	Descripción
RF-01	Tipos de datos básicos.	La gramática dispondrá de tipos de datos para que el usuario manipule tanto cadenas de texto como números reales.
RF-02	Tipo de dato “curva”.	Estará disponible un tipo de dato específico para almacenar y poder operar sobre curvas de consumo eléctrico.
RF-03	Máximo de una curva.	Encontrará el valor máximo de consumo de un tipo de dato “curva”.
RF-04	Mínimo de una curva.	Encontrará el valor mínimo de consumo de un tipo de dato “curva”.
RF-05	Media de una curva.	Realizará la media aritmética simple de los datos de consumo de un tipo de dato “curva”.
RF-06	Mediana de una curva.	Se encontrará el valor que divide el conjunto de datos de una curva en dos partes de igual número de datos (Mediana estadística).
RF-07	Desviación típica de una curva.	Se calculará la desviación típica de una curva dada a partir de los datos de consumo eléctrico.
RF-08	Rango intercuartilico de una curva.	Se calculará el rango intercuartilico (tercer quantil menos el primero) de los datos de un tipo “curva”.
RF-09	Quantiles de una curva.	Dado un número entero, se calculará el quantil elegido por dicho número dado sobre los datos de un tipo “curva”.
RF-10	Intervalos superior e inferior.	Dado un consumo (número real), se encontrarán a lo largo de un curva intervalos de tiempo que se encuentren por encima o por debajo de dicho número según el caso.
RF-11	Operaciones matemáticas.	Todos los tipos básicos de datos se podrán: sumarse, restarse, multiplicarse o dividirse entre ellos.
RF-12	Derivada de una curva.	Se podrá calcular la derivada numérica y simbólica de los datos de un tipo “curva”.
RF-13	Integral de una curva.	Se podrá calcular la integral numérica y simbólica de los datos de un tipo “curva”.
RF-14	Segmentación de una curva.	Dado un instante de tiempo, se podrá segmentar un tipo “curva” en varios a partir de dicho instante.

RF-15	Unión de curvas.	Dadas varias curvas, estas podrán unirse y formar una curva nueva conteniendo toda la información de las anteriores.
RF-16	Promedio de curvas.	Dadas varias curvas, se calculará el promedio de los datos, creando a partir de este cálculo una nueva curva.
RF-18	Carga de datos.	Se podrá cargar ficheros de datos en formato CSV con el contenido de una curva de carga.
RF-19	Bucles.	En la gramática se podrán usar bucles básicos como el bucle “While” y el bucle “for”.
RF-20	Condicionales.	En la gramática se podrán utilizar elementos de control básicos como por ejemplo el uso de condicionales “if”.
RF-21	Expresiones lógicas.	Se podrán crear y evaluar expresiones lógicas en el lenguaje.

2.2 Requisitos no funcionales

En esta sección se describirán los requisitos no funcionales, los cuales establecen las condiciones y restricciones sobre las que tiene que funcionar el lenguaje y su intérprete. En la Tabla 2 incluimos la descripción de los requisitos no funcionales.

Tabla 2: Requisitos no funcionales

ID	Categoría	Descripción
RNF-01	Usabilidad	El lenguaje debe ser lo más simple posible, el intérprete debe ser lo más sencillo posible de utilizar.
RNF-02	Fiabilidad	Los resultados de los cálculos y de la ejecución deben ser siempre correctos.
RNF-03	Reutilización	Se podrán reutilizar los códigos definidos en el lenguaje independientemente de la maquina donde se intérprete.
RNF-04	Portabilidad	Facilidad para utilizar las herramientas creadas en diferentes maquinas.
RNF-05	Rendimiento	El intérprete debe realizar sus cálculos en un tiempo razonable según la complejidad del código a interpretar.

El cumplimiento de dichos requisitos resulta indispensable para asegurar la calidad del producto final. Es por tanto muy importante que estos requisitos sean correctos, no se contradigan entre si y sean realistas, es decir, se puedan alcanzar adecuadamente dentro de las restricciones del proyecto.

Al cumplir estos cinco requisitos, aseguramos una alta calidad del producto final. Además, estos requisitos son realistas en un desarrollo moderno pues se espera unas herramientas sencillas que funcionen independientemente de la maquina donde se ejecute y se puede guardar el trabajo realizado en una máquina y usarlo en otra, además se querer tener los resultados en un plazo de tiempo razonable.

Capítulo 3
Tecnologías utilizadas

En este capítulo de nombraran todas las tecnologías, así como programas utilizados para la realización del TFG y se dará una breve explicación del cometido de cada uno de ellos y el porqué de su elección.

3.1 Xtext

Xtext es un entorno de código abierto para el desarrollo de lenguajes de programación y lenguajes específicos de dominio (DSL). Mediante Xtext se puede crear un analizador sintáctico y un modelo de clases para el árbol de sintaxis abstracta [4]. Xtext está desarrollado en el proyecto Eclipse como parte del proyecto Eclipse Modeling Framework.

Xtext ofrece una serie de características que facilitan la creación de lenguajes como:

- Coloreado de sintaxis
- Autocompletado
- Validación rápida de la sintaxis
- Integración avanzada con java
- Integración con otras herramientas de Eclipse

3.2 El lenguaje Xtend

Xtend es un lenguaje de programación de alto nivel que está diseñado para su utilización como extensión del lenguaje Java y para el desarrollo de lenguajes específicos de dominio mediante el entorno Xtext. Este lenguaje aporta a Java algunas características típicas de lenguajes de programación funcional [5]. Además, este lenguaje se compila a código Java sin cambios y por lo tanto se integra perfectamente con todas las bibliotecas de Java existentes.

Por otro lado, el uso de este lenguaje se realiza al mismo tiempo que el lenguaje Java pudiendo convivir ambos lenguajes dentro de la misma clase Java habitual.

3.3 Bibliotecas utilizadas

Se ha utilizado la biblioteca matemática de la fundación Apache “Commons Math” [6] para utilizar la resolución simbólica de ciertas operaciones matemáticas como, por ejemplo: derivada e integral simbólica.

Esta biblioteca contiene operaciones matemáticas y estadísticas comunes que no están disponibles en el lenguaje de programación Java.

3.4 Entorno de desarrollo

El entorno de desarrollo utilizado es “Eclipse Neon” para el sistema operativo Windows 10 (versión de 64 bits). Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Esta plataforma típicamente ha sido usada para desarrollar entorno de desarrollo integrados (IDE), como por ejemplo el IDE de Java. Al utilizar el entorno de desarrollo Eclipse se ha decidido utilizar como lenguaje de programación para la implementación del intérprete a Java por la multitud de herramientas y facilidades que este ofrece en este entorno.

3.5 Elaboración de diagramas

Para la elaboración de los diagramas UML se ha utilizado la herramienta “Modelio”, este es un programa de diseño y documentación de proyectos software capaz de realizar una gran variedad de esquemas útiles en este ámbito soportando gran variedad de estándares [8]. Se ha elegido esta herramienta por su facilidad de uso y corta curva de aprendizaje además de ser de código abierto y gratuita.

3.6 Paquete ofimático

Para la elaboración de la memoria se ha utilizado el paquete ofimático “Microsoft Office 365”, del cual se ha utilizado la herramienta “Word” para la elaboración de los documentos. Se ha escogido este conjunto de herramientas debido a su accesibilidad gratuita por ser alumno de la Universidad de Málaga y por la gran cantidad de recursos que esta herramienta nos ofrece para realizar textos de calidad.

Capítulo 4
Diseño e implementación

En este capítulo veremos cómo se ha estructurado el proyecto, así como el diseño e implementación tanto de la gramática del lenguaje como del intérprete y como se integrará el intérprete con el editor Eclipse. En el diseño del lenguaje veremos cómo se define las partes principales de la gramática en “Xtext”. Además, en las secciones dedicadas al intérprete se apreciará la estructura de clases de este así como se explicara con detalle el funcionamiento de cada una de estas.

4.1 Estructura del proyecto

Para crear un lenguaje específico de dominio necesitamos crear un lenguaje y para utilizarlo un intérprete. Un lenguaje está formado por su gramática (conjunto de reglas para escribir correctamente un lenguaje) y por su analizador léxico. La estructura que forma el intérprete y su funcionamiento la veremos más adelante en la sección dedicado a ello. La principal ventaja de haber usado Xtext es que el analizador léxico se genera automáticamente a partir de la definición de la gramática.

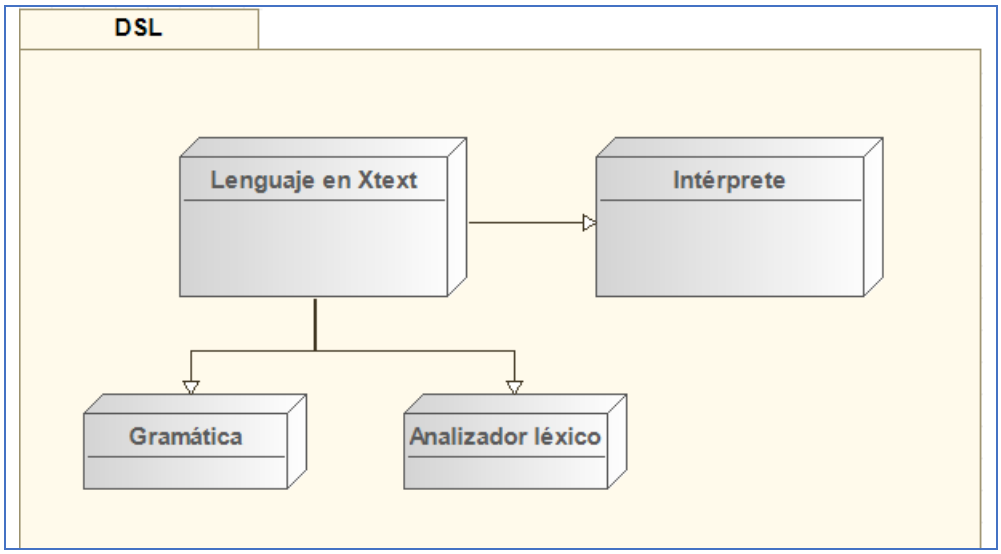

Figura 2: Esquema de implementación del proyecto.

4.2 El proyecto de Xtext en Eclipse

Se ha creado un proyecto de tipo Xtext en Eclipse, esto creará una estructura de directorios para realizar las diferentes tareas que necesitamos y poder crear el lenguaje. Al crear el proyecto se tiene que elegir el nombre de este, el nombre del lenguaje y la extensión que tendrán los ficheros que contengan nuestro lenguaje.

Nombre del proyecto	Org.uma.consumptioncurves
Nombre del lenguaje	Org. uma.consumptioncurves.Energy
Extensión del lenguaje	.ecc

Tabla 3: Nombres del proyecto y lenguajes dentro del proyecto, así como extensión del lenguaje.

Las diferentes carpetas que se generan al crear un proyecto con “Xtext” las podemos apreciar en la Figura 3.

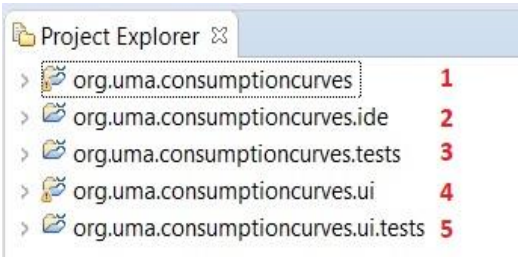

Figura 3: Estructura del proyecto

1- La definición gramatical y todos los componentes específicos del lenguaje (parser, lexer, linker, validación, etc…).

2- Funcionalidades para el IDE.

3- Pruebas unitarias para la gramática del lenguaje.

4- Código relacionado con el editor Eclipse y sus funcionalidades

5- Pruebas unitarias para la interfaz implementada en eclipse.

4.3 La gramática

En esta sección veremos las partes que componen nuestra gramática y cómo han sido definidas en Xtext. En la Figura 4 podremos ver la estructura global de la gramática en un árbol sintáctico abstracto.

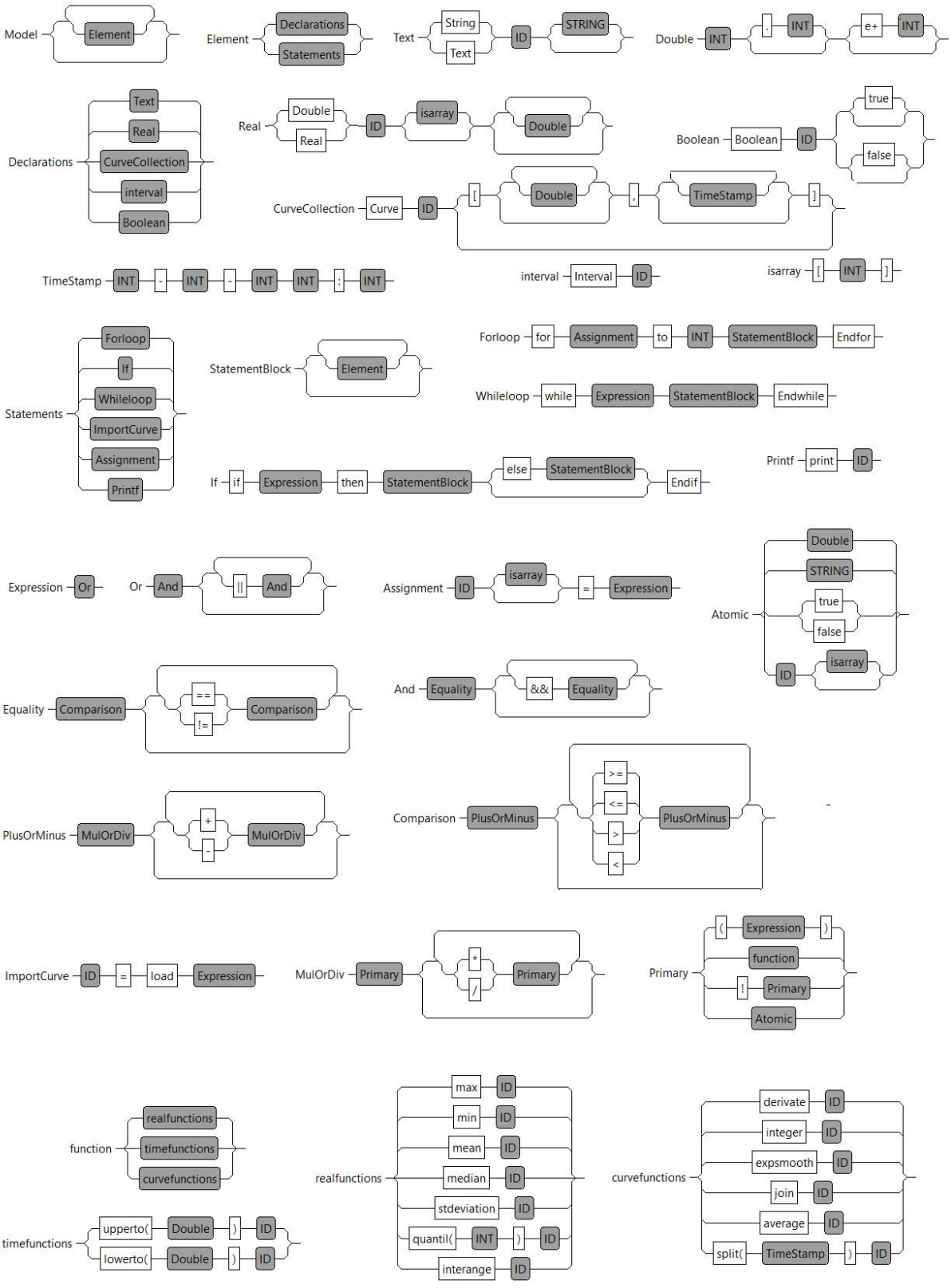

Figura 4: Árbol sintáctico completo de la gramática

El modelo de la gramática está formado por elementos, estos elementos son dos: declaraciones (aquí se definirán las variables para nuestros tipos de datos) y sentencias. También disponemos de funciones que nos darán funcionalidades extras para realizar cálculos concretos.

4.3.1 Las declaraciones de variables

En las declaraciones de variables definimos la gramática referente a los tipos de datos que necesitamos (en los subapartados siguientes se explicará cada tipo de dato individualmente). Podemos ver la definición en “Xtext” de nuestros tipos de datos en la Figura 5.

```
Declarations:
    Text | Real | CurveCollection | interval | Boolean;

Text:
    ("String" | "Text") name=ID (value=STRING)?;

Real:
    ("Double" | "Real") name=ID (array=isarray)? (value+=Double*)?;

Double returns ecore::EDouble:
    INT ('.' INT)? ("e+" INT)?;

CurveCollection:
    "Curve" name=ID ('[' values+=Double* ',' Times+=TimeStamp* ']')?;

Boolean:
    "Boolean" name=ID (value=("true" | "false"))?;

TimeStamp:
    INT '-' INT '-' INT INT ':' INT;

interval:
    "Interval" name=ID;
```

Figura 5: Definición de la gramática de las declaraciones en Xtext.

4.3.1.1 Cadenas de texto

Las cadenas de texto pueden declararse mediante dos cadenas (“Text” o “String”) seguido de un identificador alfanumérico único. También pueden inicializarse escribiendo a continuación una cadena de texto entre comillas dobles.

```
Text nombrevariable
String cadenadetexto
```

Figura 6: Ejemplo declaración de cadenas de texto.

4.3.1.2 Números reales

Los números reales pueden declararse mediante dos cadenas (“Real” o “Double”) seguido de un identificador alfanumérico único. Por otro lado, para inicializar el contenido bastará con escribir a continuación el valor que le queramos dar en cualquiera de los siguientes formatos: número entero con o sin decimales o el formato exponencial (número e+ exponente).

También es posible definir un vector de números reales y para ello solo es necesario inicializarlo con más de un valor.

```
Real nombre
Double numero 5
Real vector 1.2 2 3e+5 4
```

Figura 7: Ejemplo declaración números reales.

4.3.1.3 Curvas de carga

El tipo de datos para las curvas de carga se llama “CurveCollection” como podemos ver en su estructura (ver Figura 4), es definido por la cadena “Curve” seguido de un identificador alfanumérico único. Para inicializar el contenido de este tipo basta con escribir a continuación los datos con la siguiente estructura: los datos vendrán dados en forma de vector de dos elementos donde en el primer lugar se pondrán los valores reales y en el segundo la fecha con el formato “TimeStamp” (Año-Mes-Día Hora-Minuto).

```
Curve curvavacia
Curve curva [5.0 1.0,2017-06-23 17:15 2017-06-23 17:30]
```

Figura 8: Ejemplo declaración de curvas de carga.

4.3.1.4 Variables booleanas

Las variables booleanas se declaran con la cadena “Boolean” seguido de un identificador alfanumérico único. Estas pueden tener dos valores: verdadero o falso y se inicializan escribiendo a continuación de su identificador cualquiera de dichos valores en inglés.

```
Boolean VoF
Boolean v true
Boolean f false
```

Figura 9: Ejemplo declaración variables booleanas.

4.3.1.5 Intervalo temporal

El tipo intervalo temporal es una estructura que almacena intervalos de tiempo, se declaran mediante la cadena “Interval” seguido de un identificador alfanumérico único. La forma en que se almacenan los intervalos es mediante dos “TimeStamps”, el primero para el inicio del intervalo y el segundo para la finalización de este.

4.3.2 Sentencias

Las sentencias de la gramática son los bucles, elementos de control (condiciones), asignaciones, importar curvas y la sentencia “print” para mostrar información a través de la consola. Podemos ver la definición de las sentencias en la gramática en la Figura 4. Además, podemos ver la forma en la que se definen en Xtext en la Figura 10,

```
Statements:
    Forloop | If | Whileloop | ImportCurve | Assignment | Printf;

StatementBlock:
    {StatementBlock} Sentences+=Element*;

Forloop:
    ("for" (forcondition=Assignment) "to" forend=INT
    forBlock=StatementBlock
    "Endfor");

If:
    ("if" ifcondition=Expression "then"
    ThenBlock=StatementBlock
    ("else"
    ElseBlock=StatementBlock)?
    "Endif");

Whileloop:
    ("while" whilecondition=Expression
    whileBlock=StatementBlock
    "Endwhile");
Printf:
    "print" source=[Declarations]
;

Assignment:
    target=[Declarations] (pos=isarray?) '=' expression=Expression;
```

Figura 10: Definición de la gramática para las sentencias en Xtext.

4.3.3 Las asignaciones

Las asignaciones son la forma de atribuir el resultado de una expresión aritmética o lógica a una variable. El formato de las asignaciones es: identificador de la variable a la que se le asignará el valor igualado a la expresión correspondiente.

4.3.4 Las expresiones

Las expresiones están formadas recursivamente de forma que se puede utilizar cualquier operador aritmético (suma, resta, multiplicaciones y división) así como los operadores lógicos habituales y comparadores (mayor que, igual, menor que, distinto, etc.). La estructura de las expresiones es : Expresión-Operador-Expresión.

4.3.5 Los bucles

En los bucles tenemos definidos dos tipos de bucles básicos (“while” y “for”). El bucle “while” o bucle mientras es un ciclo repetitivo basado en los resultados de una expresión lógica. El propósito es repetir un bloque de código mientras una condición se mantenga verdadera. El bucle “for” o ciclo es una estructura de control en la que se puede indicar de antemano el número mínimo de iteraciones que la instrucción o conjunto de instrucciones realizará.

En ambos bucles se utiliza la estructura “StatementBlock”, esto simplemente simboliza a un conjunto indeterminado de elementos de nuestra gramática que serán las instrucciones que se repetirán dentro de nuestro bucle.

4.3.6 La condición “if”

La sentencia “if” nos permite tener varios flujos de control. Si se cumple la expresión evaluada se ejecutarán las instrucciones del primer bloque y si no se cumple se ejecutarán los del bloque que comprendido entre la palabra “else” y el final del condicional (“Endif”). La segunda parte del condicional es opcional pudiendo tener solamente la primera parte de este.

4.3.7 Importar curvas

Para importar curvas de carga desde un fichero en formato CSV bastará con hacer una asignación de una variable de tipo curva seguido de la palabra reservada “load” y la ruta del fichero CSV o el identificador de una variable que contenga la ruta de dicho fichero a cargar. El formato de los ficheros CSV debe ser: “Fecha; valor” (podemos ver un ejemplo en la Figura 11).

```
2015-01-01 00:30;2
```

Figura 11: Formato fichero CSV para curvas de carga.

El formato de las fechas debe ser: Año-Mes-Día Hora-Minuto

4.3.8 Sentencias print

La sentencia “print” puede ser llamada con la palabra reservada “print” seguida del identificador de la variable a mostrar y se encargará de escribir por la consola del intérprete el contenido de dicha variable.

4.3.9 Las funciones

Las funciones definidas en la gramática están organizadas en varias categorías según el tipo de dato que devuelven. En la Figura 12 podremos ver como se definen las funciones en la gramática.

```
function returns Expression:
    realfunctions | timefunctions | curvefunctions;

realfunctions returns Expression:
    {Max} "max" source=[CurveCollection] |
    {Min} "min" source=[CurveCollection] |
    {Mean} "mean" source=[CurveCollection] |
    {Median} "median" source=[CurveCollection] |
    {Stdeviation} "stdeviation" source=[CurveCollection] |
    {Quantil} "quantil(" value=INT ")" source=[CurveCollection] |
    {Interange} "interange" source=[CurveCollection];

timefunctions returns Expression:
    {Upperto} "upperto(" value=Double ")" source=[CurveCollection] |
    {Lowerto} "lowerto(" value=Double ")" source=[CurveCollection];

curvefunctions returns Expression:
    {Derivate} "derivate" source=[CurveCollection]|
    {integrate} "integrate" source=[CurveCollection]|
    {expsmooth} "expsmooth" source=[CurveCollection] |
    {join} "join" source=[CurveCollection] |
    {average} "average" source=[CurveCollection]|
    {split} "split("when=TimeStamp")" source=[CurveCollection]
    ;
```

Figura 12: Definición de las funciones de la gramática.

4.3.9.1 Funciones reales

Las funciones denominadas reales (ver Tabla 4) son aquellas funciones que tras realizar un cálculo sobre los datos de una curva de carga devuelven un valor real como resultado.

Tabla 4: Conjunto de funciones que devuelven un valor real.

Nombre	Resultado
max	Devuelve el valor máximo.
min	Devuelve el valor mínimo.
mean	Devuelve la media aritmética simple.
media	Devuelve la mediana de los valores.
stdeviation	Devuelve la desviación típica.
interange	Devuelve el rango intercuartílico.
quantil	Devuelve el cuantil dado.

4.3.9.2 Funciones temporales

Las funciones temporales (ver Tabla 5) son las que devuelven intervalos temporales (los intervalos temporales corresponden al tipo de dato "Intervalo temporal"). Las funciones temporales necesitan como entrada el identificador de la variable correspondiente a la curva además del valor correspondiente para comparar y devolverá en la estructura intervalo los intervalos correspondientes.

Tabla 5: Conjunto de funciones que devuelven un intervalo temporal.

Nombre	Resultado
upperto	Devuelve los intervalos temporales superiores al valor dado.
lowerto	Devuelve los intervalos temporales inferiores al valor dado.

4.3.9.3 Funciones de curvas

Las funciones de curvas son aquellas funciones que devuelven como resultado una curva de carga (ver Tabla 6). Estas funciones reciben como entrada al menos el identificador de la variable curva sobre la cual se le realiza la operación.

Tabla 6: Conjunto de funciones que devuelven una curva como resultado.

Nombre	Resultado
Derivate	Devuelve la derivada numérica de la curva.
Integrate	Devuelve la integral numérica de la curva.
IntegrateSym	Realiza una aproximación simbólica a los datos de la curva y realiza la integral a esa aproximación.
Expsmooth	Realiza un alisado exponencial sobre la curva.
Join	Devuelve la unión de varias curvas.
Average	Devuelve la curva promedio de varias curvas.
Split	Divide una curva en varias dado un "TimeStamp"

- La función "IntegrateSym" realiza una aproximación a los datos de la curva mediante una técnica de regresión [7] que nos proporciona la biblioteca matemática de Apache para hallar la función que corresponde a los datos de la curva. Una vez hallada dicha función se integra y se vuelven a calcular los puntos correspondientes a dicha función para ello es necesario ofrecerle los límites de integración que serán por defecto los límites de nuestra curva.

- La función "Expsmooth" realiza un alisado exponencial simple obteniendo los términos alisados de forma recursiva (ver Figura 13).

$$S_1 = \alpha\, X_1 + (1-\alpha)\, S_0,\ \ S_2 = \alpha\, X_2 + (1-\alpha)\, S_1\ ,\ S_3 = \alpha\, X_3 + (1-\alpha)\, S_2\ , \ldots.$$

Figura 13: Ecuación de cálculo de los términos sucesivos para el alisado exponencial.

Los términos S1, S2, etc. son los términos alisados, el termino alfa es la constante de alisamiento que se fija al valor "0.2". En este caso el primer término (S0) tiene el valor de la media aritmética simple de los datos sin alisar.

- La función "join" se encarga de unir dos o más curvas, esta función las une independientemente de sus intervalos temporales pues los datos temporales en la estructura curva están ordenados, haciendo que esta función se encargue de incluir los nuevos datos de forma ordenada.

- La función "average" crea una curva promedio a partir de varias, para ello realiza la media de los valores que estén en el mismo instante temporal, para ello es necesario que en las curvas los intervalos temporales correspondan entre si. Además, si el número de puntos de las curvas es diferente solo se hará el promedio de los puntos hasta llegar al último valor de la curva con menos puntos obviando los puntos restantes de las demás curvas.

- La función “Split” se encarga de crear dos curvas a partir de un instante de tiempo, esto lo realiza pues divide los datos en dos partes (a partir del instante de tiempo dado) y con cada una crea una nueva variable curva.

4.4 El intérprete

El intérprete está compuesto por una estructura de varias clases, unas encargadas de tratar el código, otras clases para almacenar y realizar operaciones los tipos de datos y una para crear y utilizar la salida por consola (a través de la clase “InterpreterConsole” que se encargará de manejar la salida). Podemos ver la estructura de clases en la Figura 14.

En las secciones siguientes veremos en detalle que tarea realiza cada clase en concreto.

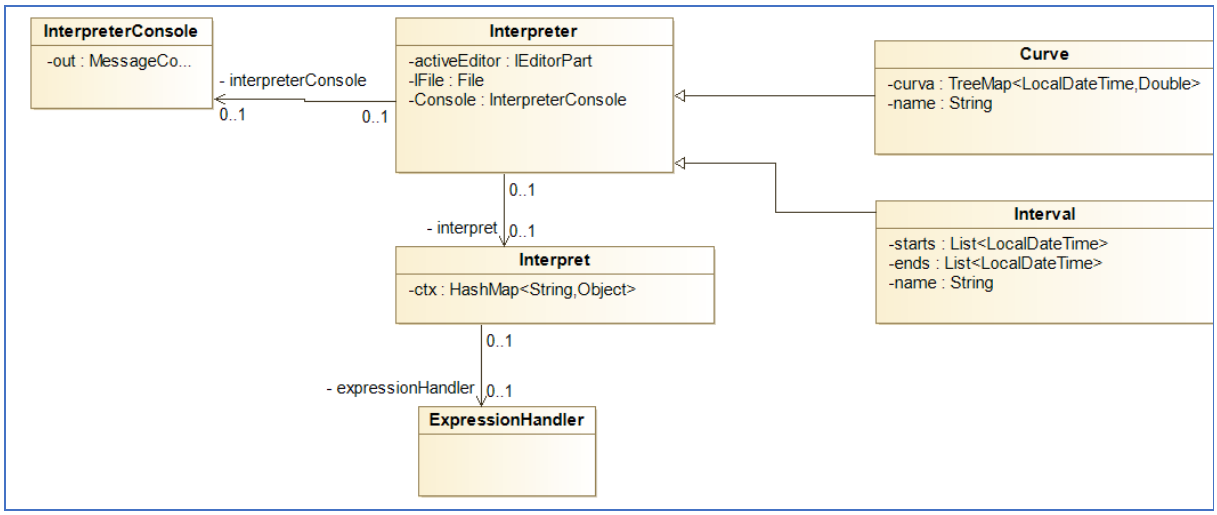

Figura 14: Diagrama de clases del intérprete.

4.4.1 La clase “Interpreter”

La clase “Interpreter” hereda de la clase “AbstractHandler” y a su vez implementa la clase “IHandler”. Esta clase es la encargada de recoger el evento de ejecución (un evento es un suceso que se encuentra previamente programado, en nuestro caso será cuando el usuario pulsa sobre un botón indicado para iniciar el intérprete y entonces empezará el evento de ejecución), tomar el fichero activo en el editor y a continuación crear una instancia de la clase “Interpret” y finalmente llamar al método “load” pasándole un objeto que incluye una lista con todas las instrucciones del fichero a ejecutar.

4.4.2 La clase “InterpreterConsole”

La clase “InterpreterConsole” es la encargada de buscar y activar la consola utilizada para la salida estándar y en el caso de no existir esta, crearla. También proporciona métodos para utilizar la consola pudiendo escribir en ella fácilmente.

4.4.3 La clase “Curve”

La clase “Curve” es la encargada de crear el tipo de dato para almacenar e interactuar con las “curvas de consumo”. La manera de almacenar los datos de una curva es mediante la estructura de datos “TreeMap” donde la clave de cada valor será la fecha y la hora de la toma de la medida y el valor será un número real. También almacenará en formato cadena de texto el nombre de dicha variable.

El constructor de la clase recibe el nombre, los valores y los tiempos de una curva dada y los añade a las variables creadas para almacenar dichos datos. También la clase dispone de los siguientes métodos para hacer operaciones sobre la curva (ver Tabla 7).

Tabla 7: Conjunto de funciones implementadas en la clase “Curve”.

Nombre	Función
setData	Asigna un nuevo valor a los datos de la curva.
getData	Devuelve los valores de los datos actuales de la curva.
Max	Devuelve el valor máximo de los datos reales de la curva.
Min	Devuelve el valor mínimo de los datos reales de la curva.
Mean	Calcula y devuelve la media aritmética simple de los datos reales de la curva.
Stdeviation	Calcula y devuelve la desviación típica de los datos reales de la curva.
Quantile	Calcula y devuelve el percentil dado sobre los datos de la curva.
Interange	Utilizando el método quantil calcula el rango intercuartilico.
Load	Lee un fichero en formato CSV y mediante el método setData introduce los datos nuevos en la estructura de dados de la curva.
Upperto	Encuentra y devuelve los intervalos de los datos que están por encima de una determinada potencia en la curva.
Lowerto	Encuentra y devuelve los intervalos de los datos que están por debajo de una determinada potencia en la curva.
Integrate	Realiza una integración numérica sobre los datos de la curva.
Derivate	Realiza una derivada numérica sobre los datos de la curva, los extremos de integración coinciden con los límites de los datos (inicio y final de los datos).
DerivateSym	Realiza una derivada a partir de la aproximación simbólica de los datos de la curva.

Average	Realiza un promedio de varias curvas generando la curva promedio.
Join	Une los datos de varias curvas.
Split	Dado un momento (fecha y hora) separa los datos en varias curvas.
toString	Devuelve una cadena de texto con los datos y claves de la curva de forma legible por una persona.

4.4.4 La clase “Interval”

La clase “Interval” nos proporciona el objeto para almacenar intervalos temporales que se usarán como resultado de diferentes operaciones para ello se ha decidido usar dos listas, la primera almacenará los comienzos de los intervalos y la segunda los finales, pudiendo así, almacenar un conjunto de intervalos en el mismo objeto. Esta clase dispone de los constructores clásicos para inicializar las variables y los siguientes métodos (ver Tabla 8) para realizar operaciones sobre esta clase.

Tabla 8: Conjunto de método implementados en la clase “Interval”.

Nombre	Función
toString	Devuelve una cadena de texto con los intervalos almacenados en el objeto de forma legible para las personas.
removeAll	Reinicializa el objeto, eliminando todos los intervalos existentes en el objeto.
setInterval	Dado el comienzo y el final de un nuevo intervalo de tipo “LocalDateTime” añade un nuevo intervalo al objeto “Interval”.

4.4.5 La clase “Interpret”

La clase “interpet” es la clase encargada de crear el contexto de ejecución de nuestro intérprete que será una estructura “TreeMap” (clave-valor) donde la clave será una cadena de texto con el nombre de la variable y el valor será de tipo “Object” para almacenar cualquier tipo de objeto.

Esta clase recibe una estructura con todas las instrucciones en el método “load” y también recibe la referencia al objeto consola previamente creado para poder utilizar la salida estándar. Aquí bastará con recorrer de forma secuencial la estructura que contiene las instrucciones e ir llamando al método “interpret” de esta misma clase.

El método “interpret” está definido de forma que según el tipo de instrucción que reciba irá la instrucción a un método u a otro (ver Figura 8), este tipo de métodos son llamados “dispatch”.

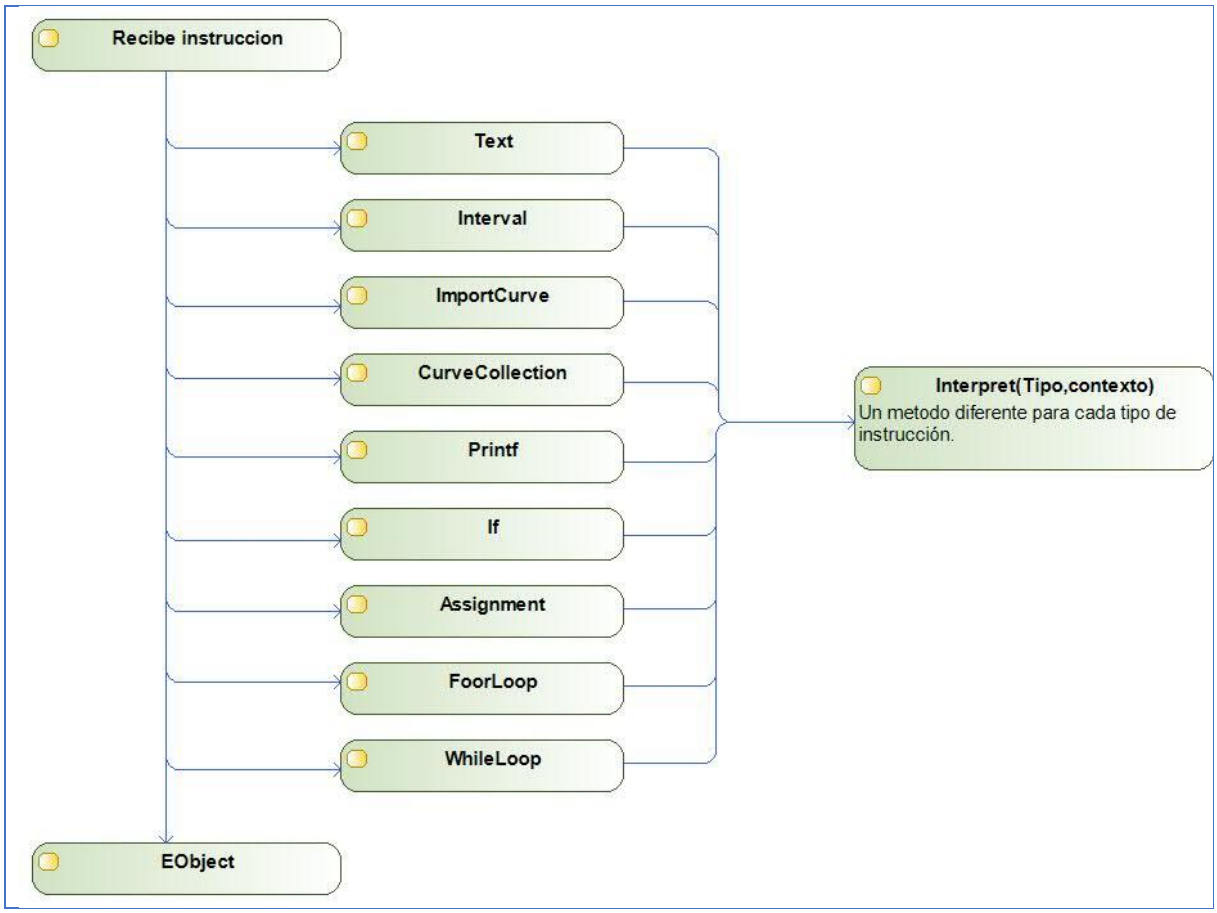

Figura 15: Diagrama de flujo identificación de instrucciones según su tipo.

En estos métodos llamados “Dispatch” se realizan las operaciones (ver Tabla 9) relacionadas con el tipo de instrucción que llega excepto la resolución de las expresiones que se hace en una clase aparte y aquí simplemente se utiliza el resultado de dicha evaluación. En el caso de que llegue otro tipo de instrucción indefinida (tipo “EObject”) no se realiza nada y se descarta, aunque esto no debería ocurrir nunca pues el analizador léxico de la gramática avisará de que lo escrito en el código no es correcto y no se identifica con ningún elemento de la gramática.

Tabla 9: Conjunto de acciones que realiza el intérprete según el tipo de instrucción que recibe.

Tipo de instrucción	Acción que realiza el intérprete
Text	Añade al contexto de ejecución la cadena de texto de tipo “String”.
Interval	Añade al contexto de ejecución una instancia de la variable “Interval”.
ImportCurve	Se encarga de llamar al método “load” del objeto curva que lo llama y además, de actualizar el contexto de ejecución con los nuevos datos de la curva.
Real	Añade al contexto de ejecución los números reales en formato “Double”.

CurveCollection	Crea y añade una instancia de la clase “Curve” al contexto de ejecución del intérprete.
Printf	Busca en el contexto de ejecución la variable que se pretende mostrar por la consola y llama a su método “toString”, después escribe el resultado por la consola del intérprete.
If	Llamará a la clase “ExpressionHandler” para obtener el resultado de la evaluación de la expresión de la condición, una vez evaluada, se procederá a ejecutar las sentencias que estén en el primer bloque (condición verdadera), en el segundo bloque (condición falsa) o en ninguna de estas según el caso. Para ejecutar las instrucciones llamará de nuevo a la clase “interpret” pasándole las instrucciones a ejecutar.
Assignment	La asignación, se compone de dos partes, por una parte, la referencia a una variable y por otra una expresión, en este método se llamará a “ExpressionHandler” que devolvería el resultado de la expresión y se le asignará como nuevo valor a la variable de la primera parte.
ForLoop	Aquí se evaluará la asignación del bucle y se repetirán las instrucciones encontradas en el cuerpo del bucle tantas veces como la asignación del bucle lo determine.
WhileLoop	Aquí se mandará la expresión a la clase “ExpressionHandler” que la evaluará y repetirá las instrucciones contenidas en el cuerpo del bucle mientras el resultado de la expresión sea verdadero.

4.4.6 La clase “ExpressionHandler”

La clase “ExpressionHandler” se encarga de evaluar las expresiones tanto matemáticas como booleanas y devolver el resultado de su evaluación. Para ello la clase se va llamando a si misma de forma recursiva analizando que hay a izquierda y a derecha de los operadores hasta llegar a los valores finales (llamados atómicos). También resuelve las llamadas a las funciones definidas pues el resultado de estas funciones son también valores finales.

En la siguiente imagen (ver Figura 16) podremos ver como se evalúan las expresiones, el diagrama es simétrico, aunque la parte de la izquierda no se expande por simplicidad.

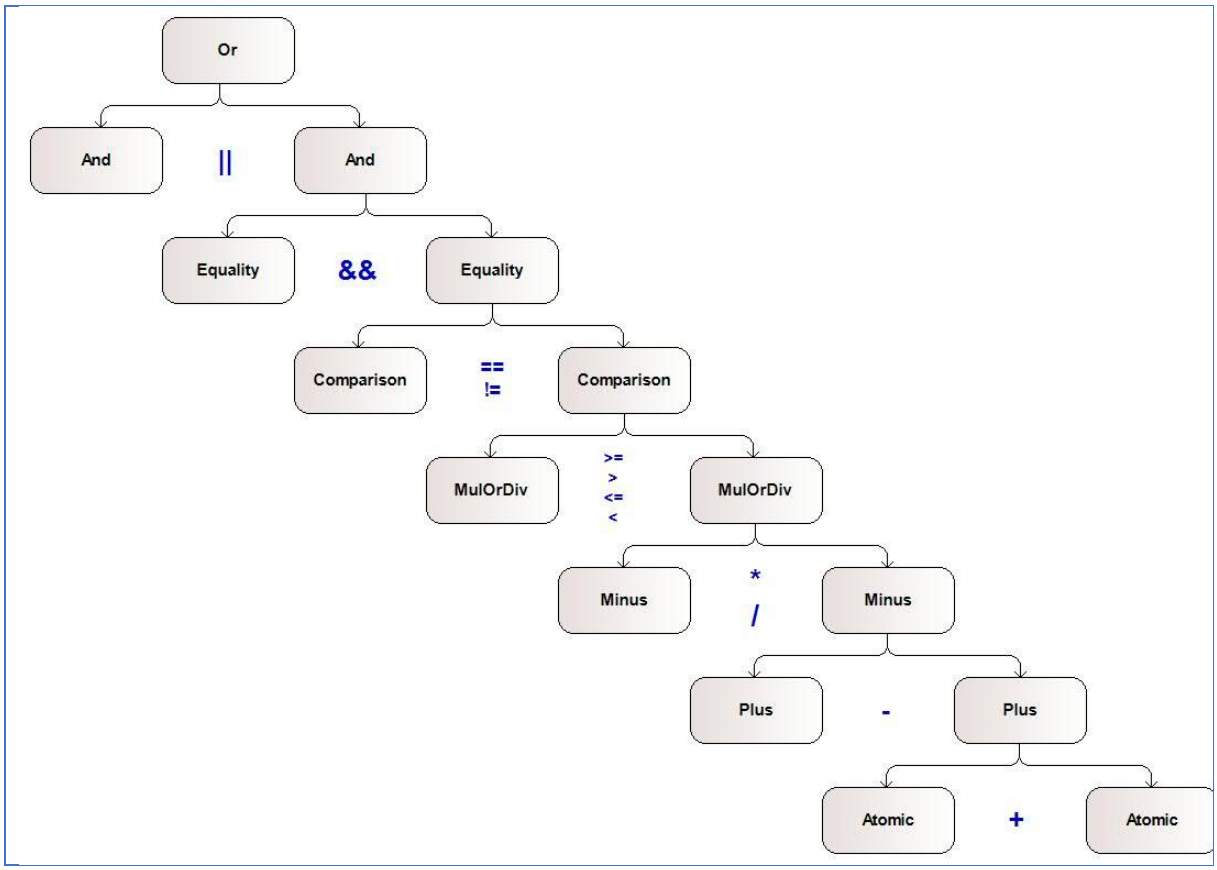

Figura 16: Diagrama de análisis de expresiones del lenguaje.

En la parte “Atomic”, también se evalúan las funciones del lenguaje, por ejemplo, si en una expresión tenemos la llamada “max curva”, el manejador de las expresiones sustituirá esta llamada por el resultado de llamar a dicha función que será el máximo de la curva siendo este un número real.

En la Figura 17 podremos ver un ejemplo de evaluación de una expresión.

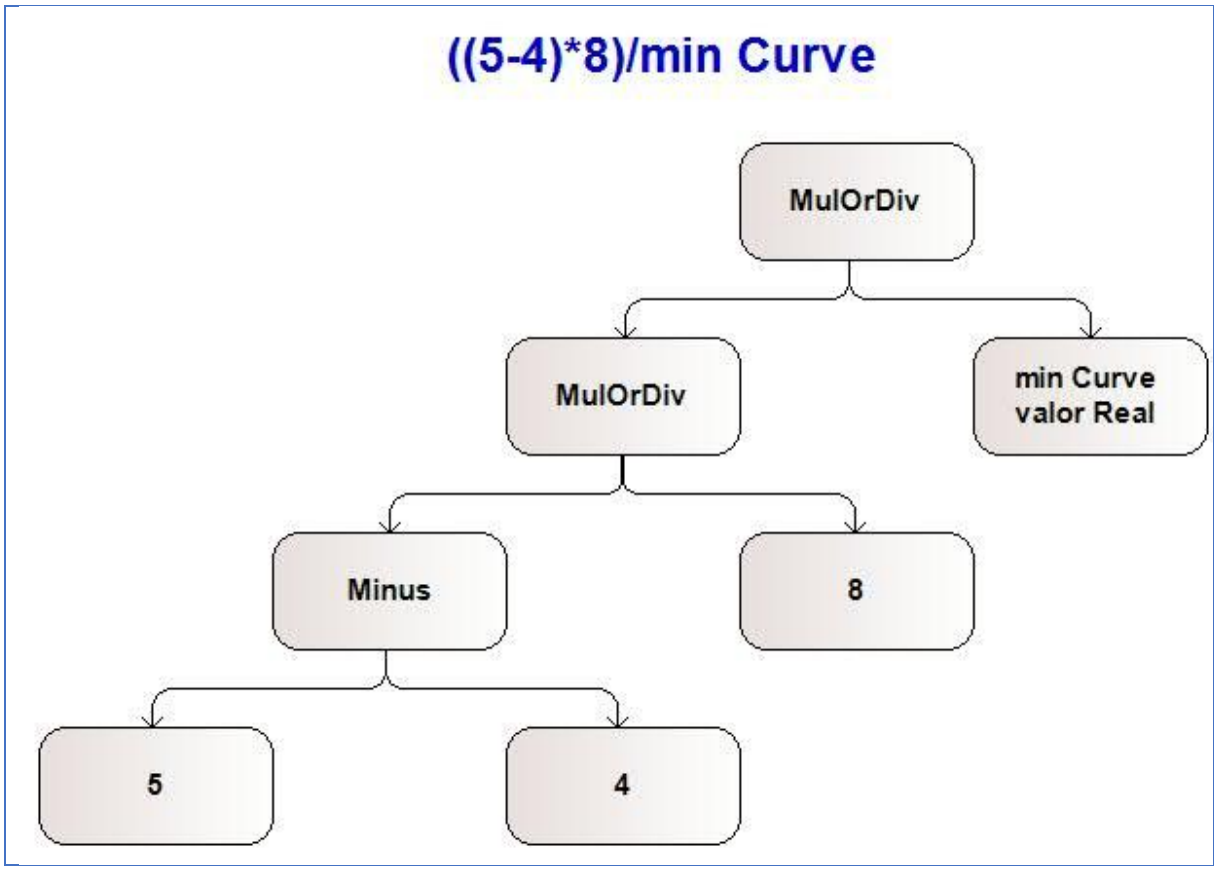

Figura 17: Ejemplo de análisis de una expresión en el intérprete.

4.5 Integración del intérprete en Eclipse

Para poder ejecutar nuestro intérprete desde la interfaz de Eclipse necesitamos poder llamarlo desde la interfaz con ese objetivo modificamos la interfaz y registramos el evento que generaremos al pulsar nuestros botones, así como llamar al método indicado.

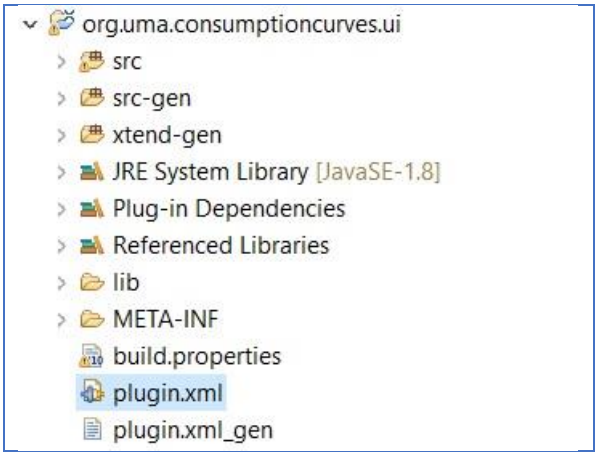

Figura 18: Fichero xml con la interfaz de Eclipse.

Para ello necesitamos modificar dentro de nuestro proyecto, en la carpeta referente a la interfaz, el fichero plugin.xml que contiene la interfaz de la aplicación Eclipse que se generará en formato XML y al final de este, registrar nuestros botones y eventos.

```
<extension point="org.eclipse.ui.menus">
<menuContribution
       locationURI="menu:org.eclipse.ui.main.menu?after=additions">
<menu
          id="org.xtext.uma.consumptioncurves.ui.menus.interpretermenu"
          label="Interpreter"
          mnemonic="M">
       <command
             commandId="org.xtext.uma.consumptioncurves.ui.commands.Interprets"
             id="org.xtext.uma.consumptioncurves.ui.menus.interpretermenu"
             label="Execute interpreter"
             mnemonic="S">
       </command>
    </menu>

</menuContribution>
</extension>
     <extension point="org.eclipse.ui.handlers">
     <handler
            class="org.xtext.uma.consumptioncurves.ui.handler.Interpreter"
            commandId="org.xtext.uma.consumptioncurves.ui.commands.Interprets">
      </handler>
 </extension>
 <extension point="org.eclipse.ui.commands">
       <command name="EJECUTATE"
           id="org.xtext.uma.consumptioncurves.ui.handler.Interpreter">
       </command>
  </extension>
```

Figura 19: Registro de la nueva interfaz y del evento de ejecución.

Capítulo 5
Pruebas de la gramática y del intérprete

En este capítulo veremos algunos ejemplos de las pruebas realizadas a la gramática definida, a través de pruebas unitarias creadas dentro de “Xtext” para probar una gramática previamente definida y también se mostrarán pruebas de ejecución del intérprete con código ya creado en un editor Eclipse.

5.1 Pruebas unitarias de la gramática

Para realizar las pruebas unitarias (mediante JUnit [9]), el proyecto Xtext ofrece una clase predefinida donde se escribirá el código siguiente la gramática previamente definida y se encargará de realizar las pruebas unitarias pudiendo comprobar rápidamente si hay errores en el analizador sintáctico de la gramática o el tiempo que tarda en ejecutarse.

5.1.1 Ejecución de la gramática con JUnit

En esta prueba, vamos a comprobar la gramática de la creación de variables de tipo de cadena de texto y de reales además de inicializarlas. También probaremos asignaciones y cálculo de expresiones, acto seguido se ejecutarán los dos tipos de bucles y finalmente un condicional “if”.

Resultado de la prueba:

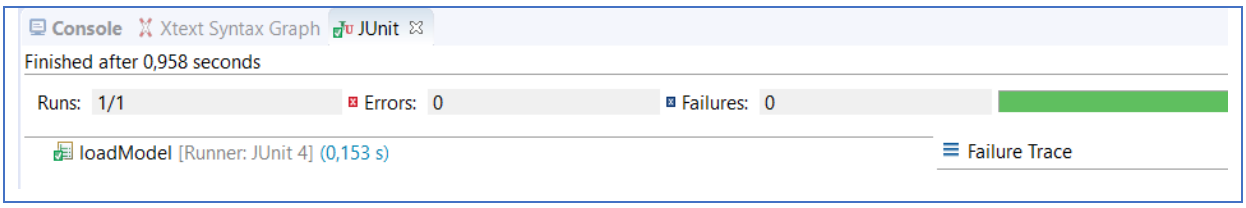

Figura 20: Resultado de la prueba unitaria.

Se puede apreciar que la gramática se analiza con un tiempo de 0,153 segundos y que no contiene ningún fallo gramatical.

En la siguiente imagen (ver Figura 21) podremos ver el código utilizado para hacer la prueba anterior y el código dentro del analizador (parser) que se a ejecutado.

```
@RunWith(XtextRunner)
@InjectWith(EnergyInjectorProvider)
class EnergyParsingTest {
    @Inject
    ParseHelper<Model> parseHelper

    @Test
    def void loadModel() {
        val result = parseHelper.parse('''
            Real nombre
            Double numero 5
            Real vector 1.2 2 3e+5 4

            Text nombrevariable
            String cadenadetexto

            Text cadena "Manual DSL

            nombre=numero
            vector[0]=(vector[1]*numero)/(vector[2]-vector[3])

            while numero!=40
                numero=numero+1
            Endwhile

            for numero=0 to 10
            vector[0]=vector+numero
            Endfor

            if numero<40 then
                numero=numero+1
            else
                numero=10
            Endif"
        ''')
        Assert.assertNotNull(result)
        Assert.assertTrue(result.eResource.errors.isEmpty)
```

Figura 21: Ejemplo de prueba unitaria en Xtext de la gramática.

5.2 Pruebas de ejecución del intérprete

En esta sección haremos pruebas de ejecución de todas las partes principales que componen nuestra gramática y comprobaremos el correcto funcionamiento de nuestro intérprete a través de visualizar si los datos tras realizar las operaciones son los correctos.

5.2.1 Definiendo variables

Vamos a definir cuatro tipos de variables e inicializarlas, primero se crearán dos variables de tipo “Real”, la primera con valor cinco y la segunda con un vector con valores comprendidos entre uno y cinco. Después se declarará una cadena de texto con el contenido “helloWorld” y finalmente una variable de tipo curva con dos fechas y dos valores de consumo.

```
Real a 5
Real b 1 2 3 4 5
Text hello "helloWorld"
Curve curve [5.0 3.0,2017-06-16 20:15 2017-06-16 20:30]

print a
print b
print hello
print curve
```

Figura 22: Prueba declaración de variables.

Y probamos si realmente las variables tienen los valores inicializados escribiendo sus valores por consola con el comando “print”.

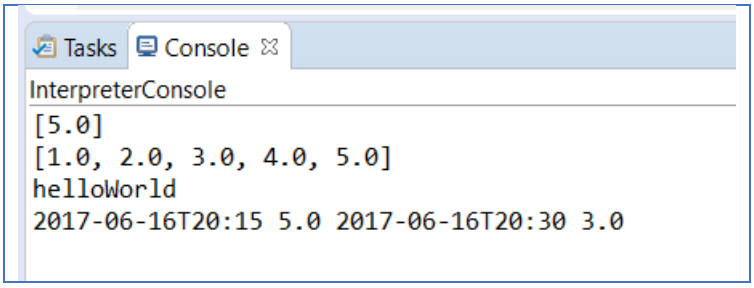

Figura 23: Mostrando el contenido de las variables por consola.

5.2.2 Asignaciones y expresiones

En esta sección probaremos a asignar a varias variables el resultado de varias expresiones matemáticas y a asignar una cadena de texto a otra variable. Los valores que contienen inicialmente las variables son los mismos que los de la sección anterior (ver Figura 23).

```
a=(b[0]+b[1])/b[2]
hello="Bye"
b[4]=(a+1)*2
```

Figura 24: Ejemplo de asignaciones y expresiones

Y comprobamos si se han evaluado correctamente las expresiones y se han asignado los valores mostrando los nuevos valores de las variables por la consola.

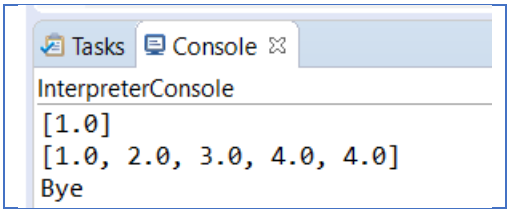

Figura 25: Mostrando el contenido de las variables por consola

5.2.3 Condicionales

Los condicionales (if) pueden tener dos formas principales, con un bloque que se ejecuta si se cumple la condición y con dos bloques uno se ejecuta si se cumple la condición y el otro si no se cumple dicha condición.

```
if a<10 then
    a=a+1
else
    a=12
Endif
```

```
if b[1]!=9 then
 b[2]=b[2]+1
Endif
```

Figura 26: Ejemplos de condicionales con y sin bloque "else".

5.2.4 Bucles while y for

En esta sección probaremos la construcción y ejecución de los bucles "while" y "for", comprobando que se realicen el número preciso de instrucciones en cada uno de los bucles. El contenido es la variable "a" es 0. Aquí nuevamente el contenido de las variables es el mismo que en la sección anterior (ver Figura 23).

```
while a<10
    a=a+1
Endwhile
```

```
for b[0]=1 to 5
    a=a-1
Endfor
```

Figura 27: Ejemplos de bucles while y for.

Y obtenemos en la consola el valor de a tras el bucle "while" y tras el bucle "for".

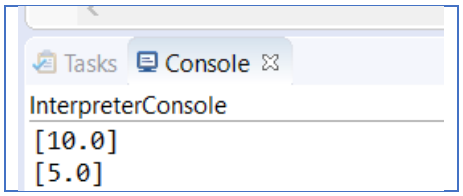

Figura 28: Resultado de la ejecución de los bucles.

5.2.5 Utilización de funciones

En esta sección probaremos como utilizar las funciones, tanto en simples asignaciones como en expresiones. Las llamadas a las funciones se pueden incluir dentro de cualquier expresión, independientemente del uso que se le dé a la expresión. En esta prueba como anteriormente los valores de las variables son los mismos que en la sección anterior de declaración (ver Figura 23).

```
a = max curve
b[0]= min curve
b[1] = mean curve + min curve
```

Figura 29: Ejemplo de uso de funciones.

Y finalmente comprobamos si los valores son los correctos.

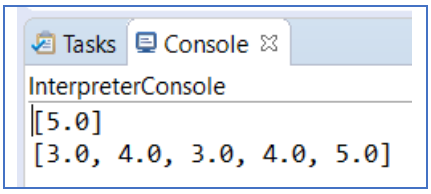

Figura 30: Resultado de la ejecución de la prueba del uso de funciones.

Capítulo 6
Aplicaciones reales

En este capítulo veremos algunos ejemplos de cómo podrían utilizarse las herramientas diseñadas en este TFG en casos prácticos para obtener algunos resultados interesantes o como estos resultados podrían obtener a través de unas operaciones en nuestro DSL.

Mediante el tratamiento de las curvas de carga a través de nuestro lenguaje especifico de dominio se puede realizar diversas tareas sobre cualquier tipo de instalación que recoja dicha información.

1. Podemos llevar un control preciso del consumo en un determinado periodo de tiempo: diario, semanal, mensual o incluso según las horas.

   Al tener los datos de consumo en las curvas de carga, podemos trocear dicha curva de forma que nos quedemos solo con los datos del intervalo que nos interese estudiar (un día, horas concretas, una semana, etc.) y así no tener que tratar con el volumen completo de datos siendo más sencillo trabajar únicamente con los datos de los cuales estamos interesados.

2. Podemos registrar excesos de potencia y por tanto de consumo y encontrar momentos en donde se consume más de lo debido.

   Utilizando las funciones “upperto” podemos determinar rápidamente si hay periodos durante el cual se está consumiendo por encima de una determinada potencia.

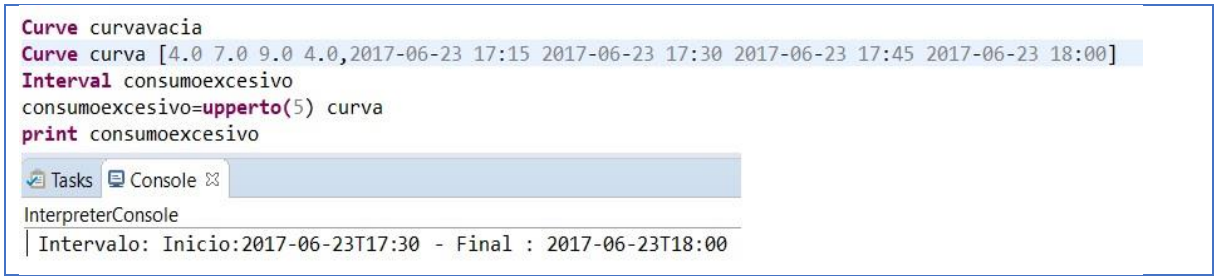

Figura 31: Código calculando un intervalo de consumo por encima de un valor (5).

3. Podemos ver la tendencia creciente o decreciente del consumo mediante las derivadas numéricas sucesivas.

Al poder derivar sucesivamente el mismo conjunto de datos, haciendo que se pueda obtener la segunda derivada de forma rápida se puede estudiar si dicho punto es un máximo o mínimo local, pudiendo predecir que la curva crece hacia el máximo o decrece hacia un mínimo.

```
Curve derivada
Curve curva [4.0 7.0 9.0 4.0,2017-06-23 17:15 2017-06-23 17:30 2017-06-23 17:45 2017-06-23 18:00]
//primera derivada
derivada=derivate curva
//segunda derivada
derivada=derivate curva
```

Figura 32: Ejemplo de cálculo de derivadas sucesivas.

4. Podemos calcular rápidamente el consumo total en intervalos de tiempo mediante integrales numéricas definidas.

   Se puede calcular el consumo total de una determinada franja de tiempo calculando la integral definida, pues el área resultante de la operación sería el consumo total.

5. Comparar consumos de diferentes edificios o instalaciones.

   Teniendo varias curvas con los datos de consumo de diferentes edificios o instalaciones se pueden calcular el consumo total de cada uno (área de los datos) mediante una integral y compararlos mediante condicionales “if” sabiendo que edificio o instalación tiene un mayor consumo.

```
if(integrate edificio1 > integrate edificio2) then
    //consumo total edificio 1 es mayor
else
    //consumo total edificio 1 es menor
Endif
```

Figura 33: Ejemplo comparando el consumo total de dos edificios.

6. Unir el consumo de diferentes edificios y hacer operaciones con los conjuntos para determinar consumos.

   Teniendo el consumo de varias casas o edificios en diferentes ficheros, podemos sumar estos o restar, con los operadores matemáticos de forma que se trabaja con el consumo total de diferentes instalaciones, pudiendo rápidamente tener en una curva el consumo total de un barrio o una calle a partir de los datos de las diferentes casas.

```
Curve casa1
Curve casa2
Curve casa3
Curve barrio

barrio = casa1+casa2+casa3
```

Figura 34: Ejemplo de suma de curvas de consumo.

Capítulo 7
Conclusiones y líneas futuras

En este capítulo haremos una recapitulación de todo lo aprendido y puesto en práctica, así como lo logrado con este trabajo fin de grado en las conclusiones. También se hablará de las posibles mejoras que se le podrán aplicar para una posible ampliación de este lenguaje especifico de dominio.

7.1 Conclusiones

Hoy día, las nuevas tecnologías están empezando a hacer su incursión en muchos sectores de la economía y la sociedad que habitualmente habían mantenido su forma de trabajar durante muchos años intacta, como por ejemplo: la ganadería, agricultura, medicina, etc. También en el campo de las ciudades inteligentes tenemos grandes avances en movilidad inteligente, donde se presente tener ciudades más urbanas con una mejor gestión del tráfico y de la contaminación. En este proyecto se crea un lenguaje especifico de dominio para ofrecer a los expertos en el sector de la energía de herramientas potentes para que puedan estudiar las ciudades inteligentes desde un punto de vista energético, tratando así que las ciudades sean más eficientes energéticamente y así reducir los costes energéticos, como la dependencia energética a los países productores y además luchas contra la contaminación.

El lenguaje específico de dominio desarrollado y las herramientas para este, ofrece a las personas expertas en energía de herramientas sencillas pero potentes que les permite estudiar el consumo de cualquier instalación (casas, edificios, calles, barrios, etc.) y así poder planificar medidas para minimizar el consumo y el coste energético mejorando la calidad de vida de nuestras ciudades.

Finalmente, hemos podido comprobar también el gran potencial de los lenguajes específicos de dominio que cada día están en mayor uso, surgiendo nuevas empresas que se dedican al diseño de estos para ámbitos como la automoción, aeronáutica, biología o medicina y proporcionar a todos los profesionales de herramientas potentes para realizar su tarea de forma más efectiva.

7.2 Líneas futuras

Tras la realización del trabajo fin de grado se han recopilado algunas líneas futuras:

- **Paralelismo en el intérprete.** Se podría previa a la ejecución del código, hacer un análisis de las dependencias entre las diferentes instrucciones y ejecutar en diferentes hilos las ejecuciones que sean completamente independiente entre si, no importando el orden en el que se ejecuten. Esto mejoraría el tiempo de ejecución para scripts altamente grandes y complejos. Aunque el tiempo de

ejecución es prácticamente nulo al ser la complejidad de la lectura de los ficheros lineal, siempre será bueno optimizar todo lo posible pues con conjuntos gigantescos de datos el tiempo se ejecución empezará a incrementarse.

- **Implementación de gráficas.** Al utilizar las curvas de carga, manejamos y mostramos datos de forma alfanumérica, pero también podríamos dibujar gráficas para mostrar el consumo de forma más visual y quizás de mejor comprensión para el usuario.

- **Agregar más operaciones.** En este trabajo fin de grado se ha intentado dar todas las operaciones posibles, pero siempre hay cálculos más específicos que puedan interesar a los posibles usuarios que pueden ser añadidos.

- **Portar el lenguaje y el intérprete a la web.** Para una mayor portabilidad y facilidad de utilización se podría portar las herramientas fuera del editor Eclipse y llevarlo a la web mediante el uso de Applets de Java.

Referencias

[1]. https://ec.europa.eu/clima/policies/international/negotiations/paris_es

[2]. https://ec.europa.eu/clima/policies/strategies/2020_es

[3]. Ministerio de Industria, Energía y Turismo, “Plan Nacional de Acción de Eficiencia Energética 2014-2020”, https://ec.europa.eu/energy/sites/ener/files/documents/NEEAP_2014_ESes.Pdf

[4]. http://www.lsi.us.es/~troyano/documentos/arboles.pdf

[5]. https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional

[6]. http://commons.apache.org/proper/commons-math/

[7]. https://es.wikipedia.org/wiki/An%C3%A1lisis_de_la_regresi%C3%B3n

[8]. https://www.modelio.org/about-modelio/features.html

[9]. https://github.com/junit-team/junit4/wiki/Getting-started

[10]. Marjan Mernik, Jan Heering y Anthony M.Slone , “When and how to develop domain specific languages”.

[11]. Lorenzo Bettini, “Implementing Domain-Specific Languages with Xtext and Xtend: Second Edition”.

[12]. Dave Steinberg,Frank Budinsky,Marcelo Paternostro y Ed Merks, “EMF: Eclipse Modeling Framework”, 2ª edición, Addison-Wesley Professional, 2008.

[13]. Documentación oficial de Xtext: “https://www.eclipse.org/Xtext/documentation/index.html”.

[14].“https://sites.google.com/site/programacionhm/conceptos/dsls/domainspecificlanguage/dsl---xtext”.

Apéndices

A. Ejemplo de uso

A.1. Creación del proyecto

Lo primero que hay que hacer para utilizar nuestro DSL en el editor Eclipse es crear un proyecto de tipo general (ver Figura 35).

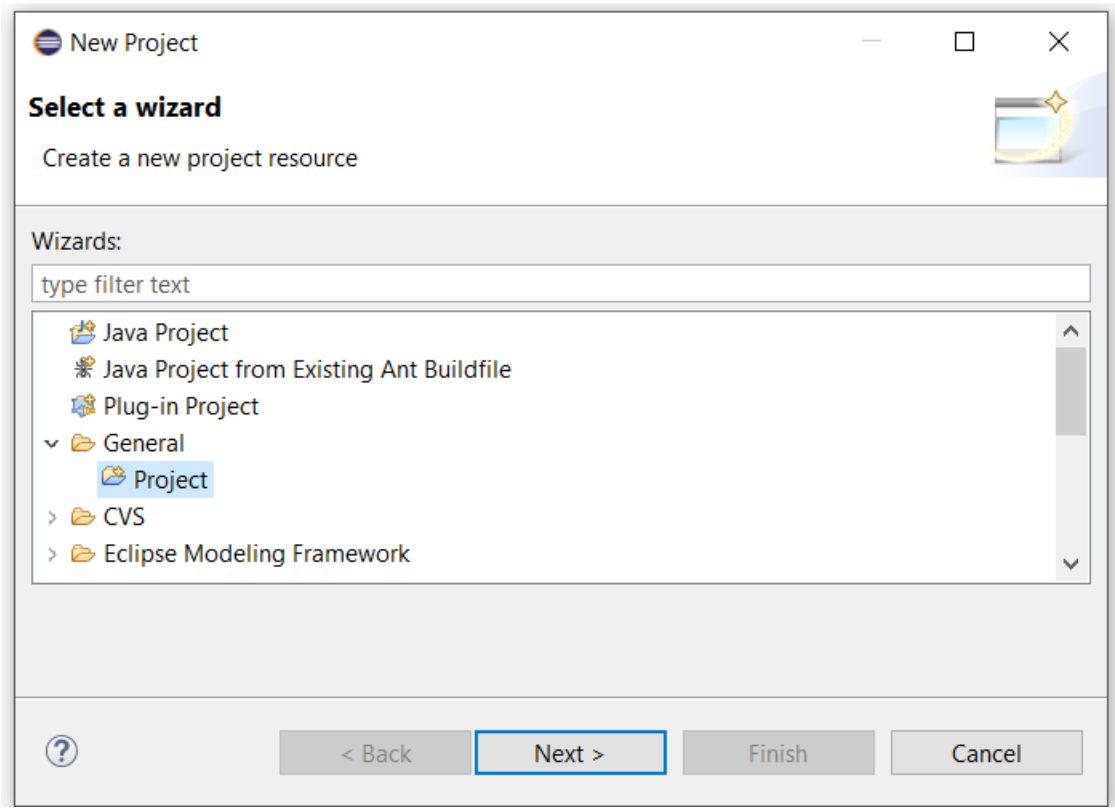

Figura 35: Creando el proyecto

Tras la ventana mostrada en la Figura 35, nos pedirá un nombre de proyecto que será a libre elección y finalmente darle a finalizar. Hecho esto ya tendremos nuestro proyecto creado en la ventana del explorador de eclipse.

A.2. Creación de un script para nuestro DSL

Tras crear el proyecto, es necesario crear ficheros que contengan nuestro código, para ello, bastará con pulsar el botón derecho del ratón sobre nuestro proyecto y crear un nuevo fichero.

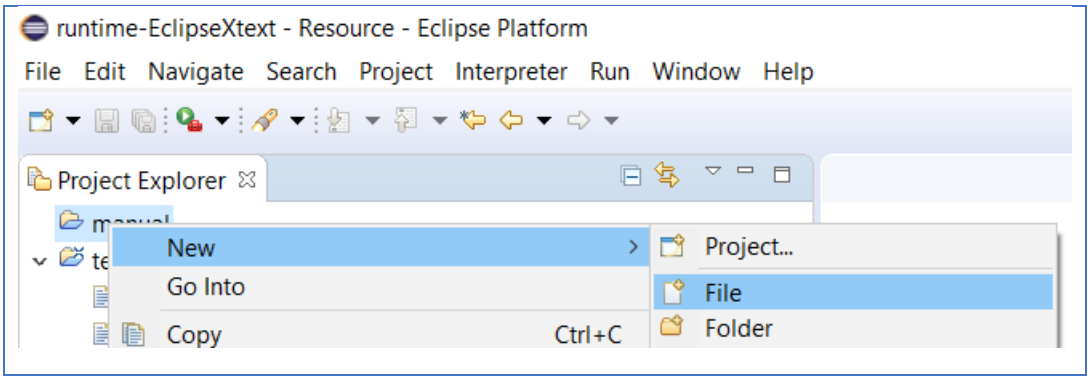

Figura 36: Creación de un nuevo fichero de la gramática.

El fichero debe tener extensión “. ecc” para ser detectado por nuestro intérprete como fichero de nuestro lenguaje. Una vez creado nos preguntará en una ventana emergente si deseamos crear un fichero para una gramática “Xtext”, elegiremos la elección de que si estamos seguros. En el caso de crear el fichero sin la extensión adecuado no será identificado como parte de nuestro DSL y por lo tanto será simplemente un fichero de texto plano y no podrá utilizarse junto a nuestro intérprete.

Figura 37: Extensión del fichero.

A.3.Escribiendo scripts para nuestro intérprete

Una vez creado el fichero, pulsando dos veces con el botón izquierdo del ratón se nos abrirá en la ventana del editor para empezar a escribir nuestro código.

A.3.1. Declarando variables

Vamos a declarar tres variables reales, una sin inicializar, una inicializada a cinco y la siguiente en forma de vector de cuatro elementos. También se declarará dos variables de tipo cadena de caracteres, el primero

sin inicializar y el segundo inicializado y lo mismo para dos variables de tipo curva.

```
Real nombre
Real numero 5
Real vector 1.2 2 3e+5 4

Text cadenavacia
Text cadena "Manual DSL"

Curve curvavacia
Curve curva [5.0 1.0,2017-06-23 17:15 2017-06-23 17:30]
```

Figura 38: Declaración de variables.

A.3.2. Asignaciones y expresiones matemáticas

Se pueden asignar valores de una variable a otra simplemente realizando una igualación entre ellas, quedando el valor de la variable a la derecha de la igualación contenida en la variable de la izquierda, esto en ningún momento modifica el valor de la variable o variables a la derecha de la igualación. También se pueden poner cualquier tipo de expresión matemática como parte derecha de la igualación y el resultado de dicha expresión será el valor finalmente asignado a la variable a la izquierda de la igualación.

```
nombre=numero
vector[0]=(vector[1]*numero)/(vector[2]-vector[3])
```

Figura 38: Ejemplo de asignación directa o mediante expresión matemática.

A.3.2. Utilizando condiciones if

En esta sección mostraremos las diferentes formas de utilizar los condicionales “if”, en el primer caso tendremos un “if” con una condición simple y dos bloques de ejecución el primero se realizará si la condición se cumple y el segundo si la condición no se cumple. En el segundo ejemplo mostramos una condición booleana más compleja para mostrar la potencia del lenguaje. Si la condición es verdadera se ejecutará el primer bloque de instrucción (seguido de la palabra reservada “then”) y si es falsa se ejecutará el bloque seguido de la palabra reservada “else”. Como tercer y último ejemplo tenemos una condición sin el segundo bloque (bloque tras la palabra reservada “else”) pues este bloque en todo momento es opcional pudiendo ponerse o no según el usuario lo requiera.

```
if numero<40 then
    numero=numero+1
else
    numero=10
Endif

if(cadena!=true)||(max curva>50)then
curva=integer curva
else
curva=derivate curva
Endif

if(numero<=40)&&(vector[0]!=4)then
    numero=numero+1
Endif
```

Figura 39: Ejemplo de utilización de las condiciones "if".

A.3.3. Utilización de los bucles while y for

En esta sección veremos un ejemplo de bucle "while" y otro de un bucle "for". En el bucle "while" las instrucciones dentro del bloque del bucle se ejecutarán mientras la condición se está cumpliendo, en este caso mientras la variable número sea distinta de cuarenta se ejecutara la instrucción del interior.

En el caso del bucle "for", se inicializa una variable contadora a un valor definido y se elige un numero de iteraciones final. En este caso se ejecutará la instrucción del interior un numero de once veces, pues lo hará desde el valor cero hasta el valor diez del contador. Al ejecutar el bloque completo de instrucciones el contador se incrementa en uno automáticamente.

```
while numero!=40
    numero=numero+1
Endwhile

for numero=0 to 10
vector[0]=vector+numero
Endfor
```

Figura 40: Ejemplo bucle "while" y bucle "for".

A.3.4. Utilización de las funciones

Para utilizar cualquier de las funciones que tenemos disponibles bastara con escribirlas dentro de una expresión, pueden ser utilizadas dentro de la condición de los bucles, de los condicionales y de las asignaciones.

A.3.5. Cargando ficheros

Para cargar ficheros externos en formato CSV con los datos de nuestra curva de carga, se hará utilizando una asignación a una variable de tipo curva y usando la palabra reservada “load”, esta tiene que recibir la ruta del fichero a cargar. Esta ruta puede ser escrita directamente o puede estar contenida en una variable de tipo cadena de texto. Al usar este método el intérprete se encargará de leer el fichero e ir creando los datos “LocalDateTime” y “Double” contenidos en el fichero, obviando los datos erróneos en la toma de medidas y generando un nuevo conjunto de datos para la variable de tipo curva a la que se le asigna.

```
curva=load "C:\\archivos\\ficherocurva.csv"
```

Figura 41: Ejemplo carga de ficheros.

Hay que tener en cuenta que, a la hora de escribir la ruta hay que poner doble barra invertida pues si solo ponemos una se entenderá como un carácter de escape como ocurre en otros lenguajes de programación modernos.

A.4. Ejecutar el script en el intérprete

Para ejecutar el script creado, bastará con pulsar con el botón izquierdo del ratón en la interfaz la orden de ejecutar y el intérprete tomará el script activo en el editor y lo ejecutará.

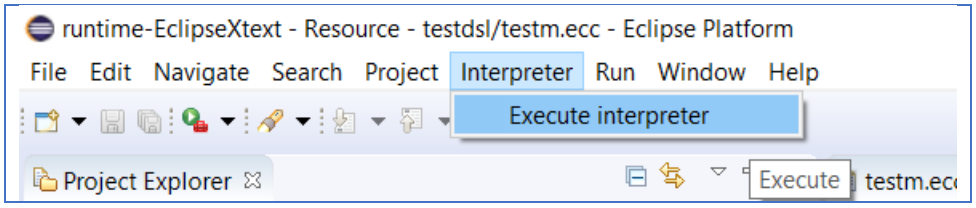

Figura 42: Interfaz de eclipse para ejecutar el intérprete.

En el caso de que el código no sea correcto, saldrá subrayado en rojo y te saldrá una ventana avisándote antes de ejecutar de que dicho código es incorrecto y que el resultado puede ser impredecible en el caso de que se prosiga con la ejecución. En el caso de mantener el ratón encima del error nos proporcionará más información acerca de este.

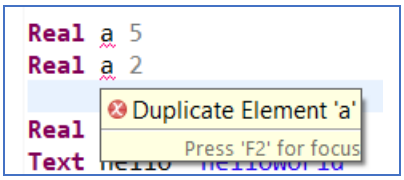

Figura 43: Ejemplo de código erróneo y ventana de error.

A.5.Salida de datos por la consola

Una vez ejecutado el intérprete los resultados de nuestro script se mostrarán por la consola creada por el intérprete para ello mostrando en ella todos lo que quisimos mostrar previamente a través del código del script y utilizando siempre la función “print”.

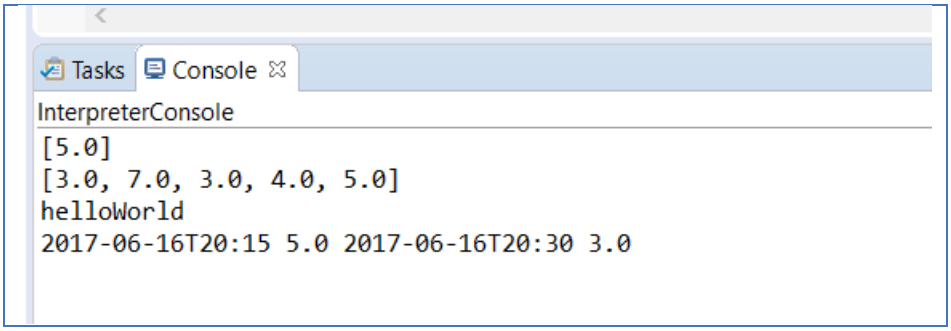

Figura 44: Ejemplo salida de datos por consola.