# Sintaxis Concreta y Abstracta

La sintaxis concreta (CS), de un lenguaje, es con lo que interactuan los usuarios para crear programas.

La sintaxis abstracta (AS) de un lenguaje es una estructura de datos que contiene la información principal del programa, pero sin ninguno de los detalles conteidos en la sintaxis concreta. Las instancias que representan la sintaxis abstracta, son por lo general llamadas árboles de sintaxis o AST. Muchos formalismos también soportan *referencias cruzadas* a través del árbol.

Si para el usuario, la sintaxis concreta es la interfaz del lenguaje, la sintaxis abstracta actua como una API para acceder a los programaas por medio de herramientas de procesamiento (validadores, transformaciones y generación de código).

Hay dos formaas para desarrollar un lenguaje por medio de la CS y la AS:

**Primero la CS**, la sintaxis abstracta se deriva, de la sintaxis concreta, ya sea de forma automática o por medio del uso de "*hints*" en la especificación de la sintaxis concreta.

**Primero la AS**, primero se define la AS, y luego se define la sintaxis concreta, en base a las especificaciones de la sintaxis abstracta. El meta-modelo de RAM, es basado en Ecore, por tanto TextRam utilizará el enfoque primero la AS.

Una vez que el lenguaje has sido definido, hay dos estrategias para crear programas relacionados a la sintaxis concreta y abstracta:

"**Parsing**", en este enfoque, el árbol de la sintaxis abstracta es construído a partir de la sintaxis concreta. Un "*parser*" instancia y popula el AS, basado en la información en el texto del programa. En este caso, una definición format del CS es llamada *gramática*.

"**Projection**", en este enfoque, el árbol de la sintaxis concreta, es contruído en forma directa por las acciones del editor, y la sintaxis concreta se representa por medio de las reglas de proyección de AST.

## Tecnología de la generación del "Parser"

En compiladores tradiconales e IDESs, los *parsers* son generalmente escritos en forma manual, por medio de grandes programas monolíticos que leen un flujo de caracteres y utilizan la recursión para crear la estructura de árbol.

En contraste, "*language workbenches*", y muchos de los compiladores actuales, *generan* un "*parser*" por medio de una gramática.

Una **gramática**, es una especificación de sintaxis escrita en un DSL, por medio de la definición textual de una sintaxis concreta.

El uso de los "parsers" generados viene con "trade-off", por una parte los "parser" generados no ofrecen el mismo rendimiento o reporte de errores en comparación con un "parser" hecho a la medida por un experto; sin embargo los "parser" generados disminuye en gran medida el esfuerzo de construir un "parser", además el desarrollador que define el lenguaje no tene que ser un experto en la tecnología de "parsers".

## Gramática

La gramática, es la definición formal de una sintaxis concreta textual. Consiste en la producción de reglas que definen como debe verse una entrada válida de texto. La gramática es la base para la definición de sintaxis basada en texto.

Resumen de la definición de la gramática según xtext documentation o el libro de xtend

## Características de los formalismos del AST

Los formalismos de AST (también llamados meta-modelos), son utilizdos para representar árboles o grafos. usualmente, dicho formalismo es "meta circular" en el sentido que puede describirse asi mismo.

### EMF Ecore

"Eclipse Modeling Framework" (EMF), es el núcleo de las herramientas de modelado de Eclipse. Expone una amplia varidad de servicios y herramientas para persistir, editar y procesar modelos y definiciones de la sintaxis abstracta.

Su componente principal es *Ecore*, que actua como meta-modelo de los modelos EMF. Xtext usa Ecore, como base para el AS: a partir de la definición de la gramática. Xtext deriva el AS como una instancia de Ecore.

Los conceptos centrales de Ecore son: 1. *EClass*, representación de los elementos del AS o conceptos del lenguaje. 2. *EAtrribute*, representación de las propiedades primitivas de EClasses 3. *EReference*, representación de asociaciones entre EClases. Los EReferences opcionalmente pueden tener semántica de contenedores. 4. *EObject*, representación de instancias de EClasses (por ejemplo nodos AST). Cada EObject puede ser contenido de al menos una instancia de EReference.

Voelter, resume el modelo de Ecore en la Figura 1.

"Figura 1. Meta-modelo de Ecore visualizado como un UML"

"Figura 1. Meta-modelo de Ecore visualizado como un UML"

## "Scoping and Linking"

Muchos formalismos del AST (también llamados meta-modelos), son