UN MODELO GENERAL PARA ASIGNATURAS DE PROCESADORES DE LENGUAJES

Antonio Polo Márquez, Miryam Salas Sánchez

Departamento de Informática
Universidad de Extremadura
e-mail: **polo@unex.es**, **miryam@unex.es**

RESUMEN: Habitualmente el contenido de una asignatura de Procesadores de Lenguajes se centra en el estudio de lenguajes imperativos (tipo C o Pascal) y se utilizan herramientas basadas en reconocedores de estados como lex/yacc. En el presente artículo se propone un marco general que permita introducir, de forma natural e integradora, nuevos conceptos en el programa de esta asignatura. De una parte se propone el estudio de lenguajes de programación no imperativos, así como lenguajes de marcado (dentro de la familia de SGML/XML), tan habituales en el ámbito de Internet. De otra, abordar nuevas técnicas de procesamiento basadas en la manipulación directa de la representación intermedia del documento, como las que suelen emplearse en el procesamiento de los lenguajes de marcado. Esta propuesta se realiza mediante un diagrama de clases de los diferentes conceptos, analizando sus relaciones y aportando sugerencias para la elaboración de un programa real de la asignatura que se adapte a las necesidades de cada Centro.

1.- INTRODUCCIÓN.

En el Plan de Estudios de la Ingeniería de Informática que se imparte en la Escuela Politécnica de la Universidad de Extremadura, la asignatura de "*Procesadores de Lenguajes*" (PL) figura como una asignatura troncal, de carácter anual y con 9 créditos asignados (6 teóricos y 3 prácticos).

Cabe señalar que es la única asignatura obligatoria de carácter anual que deben cursar los alumnos de último curso de la Ingeniería, y que esta situación es similar en la mayoría de los Planes en otras Universidades.
Esta asignatura toma como base los conocimientos adquiridos por el alumno en la de "*Teoría de Autómatas y Lenguajes Formales*", que se imparte en 3º.
El contenido clásico de esta asignatura se ha centrado en el estudio de compiladores de lenguajes imperativos de alto nivel, tipo C ó Pascal, y el estudio de herramientas para su construcción. El núcleo de estas herramientas, denominadas de *generación de compiladores*, suele consistir en los reconocedores de lenguajes lex/yacc, los compiladores del lenguaje en el que está escrito el compilador (denominado *lenguaje de implementación*, y que suele ser C) y todo ello integrado mediante el uso de la herramienta *make* para automatizar el proceso de generación automática del compilador.

La guía central para este tipo de asignaturas es el libro clásico de Aho y Ullman [AHO86], que fue el fruto final de una sucesión de *biblias* de dichos autores sobre procesadores de lenguajes desde 1972.

Sin embargo, en los últimos años, todos los que impartimos esta asignatura, tenemos la sensación de que este esquema debe ser extendido para incluir numerosos conceptos que, de forma a veces vertiginosa, se están desarrollando en el campo de los procesadores de lenguajes.

De una parte, se presenta la necesidad de estudiar otros tipos de lenguajes. Así, en el ámbito de la programación, encontramos distintos paradigmas que proponen tipos de lenguajes con filosofías y necesidades de compilación bien diferentes a las de los lenguajes imperativos; como ocurre con lenguajes funcionales, lógicos y orientados a objetos. Y no sólo los lenguajes de programación, sino que lenguajes de propósito general para representación de información, como los lenguajes de marcado (por ejemplo HTML), se presentan con nuevos fines y necesidades.

De otra parte, las técnicas de procesamiento de estos lenguajes están cambiando profundamente. Por ejemplo, el uso de orientación a objetos para manipular directamente una representación del documento, como propone el Modelo de Objeto de Documento [DOM], o nuevas formas de especificar las reglas de transformación, como ocurre con XSL [XSL].

¿Por qué cuesta tanto introducir estos conceptos en los programas de nuestras asignaturas? Pensamos que no es sólo el vertiginoso cambio del mundo de los lenguajes de programación, o la falta de tiempo material para que los profesores puedan asimilar e integrarlos en la asignatura. Más importante nos parece la falta de un modelo general que aglutine el estudio de todos estos lenguajes y presente una metodología común para afrontar su estudio.

Creemos que, en la mayoría de las ocasiones, el profesor renuncia a realizar estos cambios por la imposibilidad de *añadir* más y más conceptos, en la mayoría de los casos absolutamente dispares. Resulta insuficiente el tiempo asignado y la complejidad de la asignatura se hace inmanejable para el alumno al tratar tantos temas inconexos.

En este artículo se propone, no el programa de la asignatura PL, sino un marco general de presentación de todos los conceptos esbozados anteriormente. Somos conscientes de que los contenidos son excesivamente amplios para abordarse en nuestra asignatura. Nuestro objetivo es aportar una visión unificadora de la asignatura de PL, extendiendo el contenido clásico, que permita al alumno asimilar de forma controlada nuevos conceptos y al profesor construir el programa concreto según las necesidades de cada Centro.

2.- MODELO GENERAL PARA EL ESTUDIO DE PROCESADORES DE LENGUAJES.

Para presentar el curriculum que proponemos para la enseñanza de PL, utilizaremos un modelo basado en objetos y expresaremos el conjunto de conceptos mediante un diagrama de clases, como se representa en la Figura 1.

La clase raíz representa una clase abstracta que indica el conjunto de conocimientos y herramientas necesarias para abordar el estudio de una determinada familia de lenguajes. El concepto de *familia de lenguajes* está determinado por el metalenguaje seleccionado en cada caso.

Esto nos lleva a considerar dos *implementaciones* concretas de esta clase abstracta:

a) La familia de lenguajes cuyas gramáticas se pueden expresar mediante notación EBNF. Y que, en nuestro caso, usaremos para expresar las gramáticas de los lenguajes de programación.
b) La familia de lenguajes cuyas gramáticas se pueden expresar mediante notación SGML/XML. Y que constituye la amplia familia de lenguajes de marcado habituales en el mundo de Internet, con su parafernalia de acrónimos correspondientes que a veces tanto desorientan.

Estas dos familias de lenguajes deben verse como *instancias* de un mismo molde (la clase abstracta raíz), y por tanto, deben soportar el mismo conjunto de propiedades y métodos generales para cualquier lenguaje. Bien es cierto que, en algunos puntos, existan diferencias en la *implementación* de algunas de estas características, como se indican en el resto de este artículo.

Los lenguajes concretos que aparecen como hojas en el árbol de la Figura 1, corresponden a objetos o instancias concretas de cada clase y coinciden con un procesador de un lenguaje fuente a una máquina virtual. Nótese que para los lenguajes de marcado no se especifica la máquina virtual, ya que tanto el lenguaje fuente como las reglas de su interpretación aparecen en la declaración del correspondiente espacio de nombres.

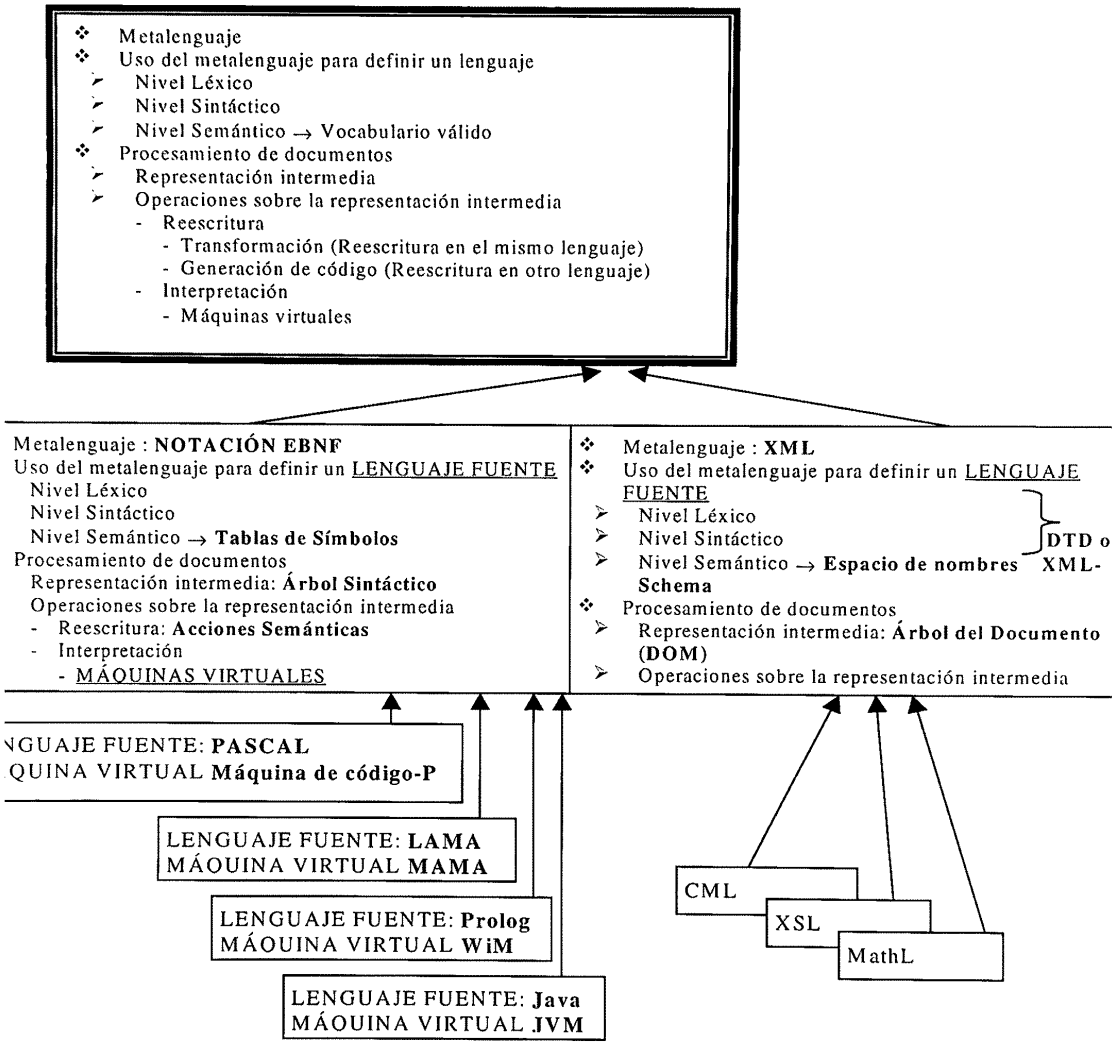

Figura 1. Diagrama de clases para el estudio de lenguajes

3.- PROCESADORES PARA LENGUAJES DE PROGRAMACIÓN.

Para abordar el estudio de los lenguajes de programación, se ha tomado como base el libro de Reinhard Wilhelm y Dieter Maurer [WIL95], que presenta un enfoque innovador en la línea de lo que proponemos en este artículo. La idea es definir siempre la misma estructura de presentación para cada tipo de lenguaje. Esto permite al alumno realizar un estudio comparativo, en los diferentes aspectos relevantes al procesamiento de lenguajes.
Para cada tipo de lenguaje de programación, se realiza el siguiente estudio:

a) Definición de un lenguaje fuente sencillo, pero representativo del paradigma de programación correspondiente. Se estudiarán sus niveles léxico, sintáctico y semántico.
b) Definición de una máquina virtual capaz de interpretar un código intermedio al que se traducirá el lenguaje fuente anterior.
c) Especificación de las funciones de compilación abstractas, que indican la forma de traducción de cada construcción del lenguaje fuente a código para la máquina virtual.
d) Finalmente, se realizarán los ejercicios o se darán las indicaciones necesarias para implementar las funciones de compilación anteriores utilizando herramientas del tipo lex/yacc.

Este esquema de estudio deberá adaptarse en cada caso, tanto al tipo de lenguaje concreto que se esté tratando, como también a la formación previa de cada alumno.

a) Lenguajes imperativos.

Se plantea la construcción de un compilador de un subconjunto de Pascal a código-P [WIL95] o de un subconjunto de C a código de una máquina de pila del tipo código-P [SCH85]. Como en nuestro caso el alumno tiene un conocimiento del lenguaje, apenas se tratan los aspectos léxicos, sintácticos y semánticos del lenguaje fuente. Pero se hace hincapié en el uso de las funciones de compilación y la máquina virtual. A nivel práctico se suele estudiar un compilador ya desarrollado para este propósito y se plantean diferentes cambios y mejoras en el mismo. Se destacan los mecanismos de implementación usando lex/yacc para construir el compilador.
En el futuro del curso, para el resto de lenguajes, ya se parte de estos conocimientos de las herramientas lex/yacc y se debe profundizar en otros aspectos como la definición y semántica del lenguaje fuente.

b) Lenguajes funcionales.

El estudio de un lenguaje funcional resulta, para la mayoría de nuestros alumnos, el primer contacto con este paradigma de programación, pues nuestro plan de estudios presenta escasas asignaturas que lo traten y siempre son de carácter optativo.
Siguiendo [WIL95], se puede proponer un lenguaje funcional muy sencillo, pero con las características principales como definición y aplicación de funciones, así como definiciones de expresiones recursivas mediante la construcción *letrec*. En este caso, los niveles léxico y sintáctico son muy simples, pero cuesta introducir al alumno en la filosofía funcional y que sean, no ya capaces de *interpretar mentalmente* cada programa para poder *leerlo*, sino que sepan capaces de *escribir* situaciones elementales en el lenguaje propuesto.
Otro concepto absolutamente novedoso para el alumno es la máquina virtual que se presenta, fundamentalmente en el uso intensivo de la memoria del *montón* (*heap*) para representar las clausuras y los mecanismos de evaluación de las mismas.
El estudio de este tipo de lenguajes se termina con una sencilla extensión del lenguaje para que admita el tratamiento de listas.

c) Lenguajes lógicos.

[WIL95] propone un subconjunto de Prolog y una máquina virtual, la máquina WiM, que es una simplificación de la máquina Warren [KOG91].
De nuevo, uno de los principales problemas es el caso de que el alumno desconozca la programación lógica. Esta situación requiere de nuevo, no sólo el estudio de las técnicas de procesamiento de estos lenguajes, sino todo un curso de programación en este paradigma.
Esto nos ha llevado en nuestro caso particular a la imposibilidad de incluirlo en el temario general y sólo se ha propuesto como extensión opcional en las prácticas, bien para aquellos alumnos que querían ampliar sus conocimientos, o bien para aquellos alumnos que ya habían tenido un contacto previo con este tipo de lenguajes al haber cursado las asignaturas optativas correspondientes.

El estudio de las máquinas virtuales para lenguajes no imperativos puede extenderse en [KOG91].

d) Lenguajes orientados a objetos.

Para los lenguajes orientados a objetos, proponemos el contenido indicado en [WIL95], pero apostamos reajustarlo para Java, en lugar de C++. La principal razón es presentar la máquina virtual de Java (JVM), e introducir conceptos como interpretación abstracta, validación de código, seguridad, portabilidad frente a eficiencia, etc.
En este caso, más que definir un nuevo lenguaje, se trata de extender los conceptos tratados para los lenguajes imperativos (Pascal o C) y analizar las soluciones para compilar características de orientación a objetos como clases y objetos, herencia, generalización, encapsulación, etc.
De nuevo, es necesaria una *cultura* del alumno en programación orientada a objetos para que pueda afrontar con éxito el estudio de las técnicas de compilación de estos lenguajes.
Un ejercicio interesante es la posibilidad de que el alumno acceda a la lectura del código fuente de los compiladores de estos lenguajes, fácilmente accesibles en la red. En ellos encontrarán *nuevas* formas de procesamiento de lenguajes, distintas de las herramientas lex/yacc.
Otra posibilidad es utilizar como lenguaje de implementación un lenguaje orientado a objetos, como Java en [APP98].

4.- PROCESADORES PARA LENGUAJES DE MARCADO.

La principal fuente para implementar los conceptos de este módulo se hayan en la página **http://www.w3.org** del World Wide Web Consortium [W3C], que debe ser consultada periódicamente para estar al tanto de las direcciones de desarrollo en XML.

a) Herramientas para el procesamiento de lenguajes de marcado.

El punto de partida del modelo de estudio que proponemos consiste en partir de XML[XML][1] como metalenguaje.

[1] Consideramos XML como una simplificación y mejora de SGML, pues aunque XML es una restricción de SGML, podemos decir que prácticamente todos los lenguajes de marcado utilizados se expresan mediante XML. Por ello nos limitaremos a la clase de lenguajes cuya gramática está definida mediante XML.

Se resalta la forma de expresar la gramática del lenguaje, y para ello, preferimos utilizar especificaciones más avanzadas al uso del DTD propuesto inicialmente en la especificación de XML, como son los XML-Schemas [XML-Sch].
En los niveles de definición de un lenguaje de marcado basado en XML, destacamos la simplicidad de los niveles léxico y sintáctico, así como el mecanismo de uso de espacio de nombres para extender la riqueza semántica del lenguaje.
Un punto importante son las operaciones de procesamiento de documentos. En primer lugar se resalta la simplicidad del modelo de objeto de documento [DOM], y el conjunto de métodos que permiten manipular el documento. Esta técnica de procesamiento es radicalmente distinta a la utilizada mediante lex/yacc en el procesamiento de lenguajes de programación.
A continuación, se aborda el estudio de XSL como un lenguaje que permite especificar las operaciones de reescritura. La potencia de XSL es suficiente para realizar las operaciones de transformación que puedan requerirse en el curso. Pensamos que los avances en las mejoras de XSL le convertirán en un lenguaje de manipulación de documentos lo suficientemente potente como para realizar todas las labores necesarias en el procesamiento de lenguajes de marcado.
Una característica a destacar es que el lenguaje XSL es declarativo, pero más próximo al estilo de los lenguajes lógicos (con reglas del tipo si ... entonces ...) que a las reglas de reescritura en las gramáticas definidas mediante notación EBNF.
La realización de prácticas en esta parte [MARC99] está fuertemente condicionada por los constantes cambios en las especificaciones.
Finalmente, se resalta las dos técnicas esenciales de interpretación de documentos XML:

I. Definiendo la semántica de un espacio de nombres. Que será equivalente a definir una máquina virtual.
II. Construyendo programas capaces de interpretar las instrucciones de procesamiento del documento.

Para la implementación de herramientas que puedan interpretar estos documentos suele ser necesario desarrollar programas en Java y librerías que implementen las especificaciones del modelo DOM [MARU99].

b) Ejemplos de lenguajes de marcado.

En el caso de los lenguajes de marcado la variedad de lenguajes disponibles es enorme. No sólo en cantidad, sino en la funcionalidad y objetivos de cada lenguaje propuesto.
Para afrontar este tema, nos parece que lo importante es que el alumno aprenda dos habilidades esenciales:

- ¿Cómo localizar lenguajes ya existentes en un dominio de aplicación concreto? Pues no debemos caer en la trampa de querer resolver cualquier problema que se nos presente definiendo nuestro propio lenguaje específico y empezar a construir las aplicaciones necesarias. Debemos de inculcar al alumno la necesidad de reflexionar previamente, y buscar soluciones existentes para no *reinventar la rueda* constantemente. Para ello se estudiarán lugares de encuentro en la red donde estén disponibles las definiciones de los espacios de nombres y herramientas disponibles para procesar los documentos del correspondiente lenguaje.
- ¿Cómo aplicar esos lenguajes para resolver un problema concreto? Pues la existencia de espacios de nombres y herramientas que resuelvan un dominio del problema no es útil si no se sabe integrar en el sistema concreto que estemos desarrollando. Se trata, por consiguiente, de saber definir nuestro lenguaje y las herramientas adecuadas para

procesar sus documentos, así como la interfaz de publicación/lectura de documentos con el exterior.

Para llevar a cabo estas acciones se proponen dos prácticas:

1. Acceso a lugares de registro de especificaciones XML [XMLorg], y examinar algunas de las propuestas que el profesor considere más representativas de la metodología a seguir para definir estos *dominios de conocimiento.*
2. Definir un lenguaje ML1, que sirva para realizar un marcado de alguno de los lenguajes desarrollados en el bloque de lenguajes de programación.

Esta última práctica es la culminación del curso y servirá para integrar y conectar todos los conocimientos adquiridos. La representación de un programa como un documento sugiere de forma inmediata varias ideas de fácil aplicación que el alumno puede realizar de forma práctica como:

- Dotar al programa de información adicional que pueda ser eliminada/recuperada para su documentación. Por ejemplo un mecanismo de generación automática de documentación como la que realiza *javadoc* en Java, incluyendo guardas, pseudocódigo, etc.

- Escritura de programas XSL que realicen la generación de código a la máquina virtual correspondiente o reescritura hacia otro lenguaje de alto nivel.

- Generar el código destino con estructura de marcado y facilitar operaciones de transformación para optimización del código o decompilación

5.- CONCLUSIONES Y TRABAJOS FUTUROS.

La propuesta de introducir nuevos conceptos en PL, readaptando los ya tradicionales de este tipo de asignaturas, se ha presentado mediante un diagrama de clases. En este diagrama, los conceptos para el estudio de diferentes lenguajes aparecen de forma jerárquica heredando de una misma clase raíz, que representa el modelo de estudio. Se han propuesto dos *implementaciones* de esa clase raíz: una para los lenguajes de programación y otra para los lenguajes de marcado.
La clase de los lenguajes de programación puede presentar diferentes instancias de objetos, que corresponden a lenguajes modelo de diferentes paradigmas de programación. Cada uno de ellos presenta una máquina virtual y las funciones de compilación hacia dicha máquina.

Para la clase de los lenguajes de marcado se resaltan los mecanismos de definición y las técnicas de manipulación de documentos a través del Modelo de Objeto de Documento [DOM] y el lenguaje XSL [XSL].
Creemos que esta visión de alto nivel del diseño de la asignatura PL puede ser de gran utilidad a los profesores implicados en estos temas.

Una de las ventajas de esta forma de modelado de contenidos es la facilidad para realizar adaptaciones a cada caso particular y sobre todo, proporciona una metodología para futuros diseños.
Actualmente, estamos trabajando un rediseño de la propuesta en el que figura como clase raíz la clase de documentos de marcado. Los lenguajes de programación se contemplarían en este

diseño como un caso particular de documento XML. Esta propuesta nos parece más lógica y supone un cambio más radical y atrevido en el planteamiento de esta asignatura. Sin embargo, nos parece que actualmente sería complicado llevarla a la práctica pues la mayoría de las herramientas para XML disponibles en el mercado (tanto a nivel conceptual como de software), se encuentran en proceso de febril desarrollo y por ello preferimos dar un poco de tiempo para que madure este campo.

ANEXO: Esquema De Programa Para Una Asignatura De Procesadores De Lenguajes

Objetivo General:

Partiendo de distintos modelos de comunicación, se analizarán los principales tipos de lenguajes que el hombre utiliza para expresar conceptos y cuya representación material es un documento. El curso se centra en el estudio de las técnicas de procesamiento de dichos documentos y cómo se pueden desarrollar herramientas informáticas que realicen o sirvan de ayuda en dicha tarea. Se distinguirán dos tipos especiales de documentos:

- Documentos de propósito específico, cuyo objetivo es representar conocimiento interpretable por dispositivos informáticos (máquinas virtuales), y que se corresponden con la categoría de los lenguajes de programación.
- Documentos genéricos, cuyo objetivo es simplemente representar información, combinando lenguajes multimedia (texto, imagen y sonido); y que se corresponden con la categoría de los denominados lenguajes de marcado.

Objetivos Específicos:

Teoría:

- Obtener un marco general de estudio de técnicas y herramientas para el procesamiento de lenguajes, que abarque tanto a los lenguajes de programación como a los lenguajes de marcado.
- Conocer y saber utilizar herramientas prácticas para el diseño léxico, sintáctico y semántico de los lenguajes de programación, así como para el proceso de reconocimiento, generación y optimización de código en la construcción de compiladores.
- Extender los conceptos anteriores a tipos de lenguajes distintos del imperativo, como son lenguajes funcionales, lógicos y orientados a objetos.
- Entender la importancia de la arquitectura de la máquina destino y la necesidad de definición del entorno de ejecución al que se va a traducir.
- Conocer y saber aplicar los principios de diseño e implementación de intérpretes como base para la definición de máquinas virtuales.
- Obtener la base suficiente para aplicar los principios básicos de la asignatura al procesamiento de información mediante su representación en documentos.

Prácticas:

- En la primera parte del curso se estudiará el diseño y realización de un compilador clásico sencillo para/con lenguajes imperativos. Para ello se seleccionará un subconjunto de un lenguaje imperativo conocido como Pascal ó C, proponiéndose distintas extensiones y modificaciones al mismo. El alumno se familiarizará con el uso de herramientas de generación de compiladores del tipo Lex/Yacc. Este lenguaje se denominará L1.
- En la segunda parte se aplicarán las técnicas anteriores a la construcción de compiladores para/con nuevos tipos de lenguajes: un lenguaje funcional, un lenguaje lógico y un lenguaje orientado a objetos.
- Finalmente deberá realizar la definición de un lenguaje de marcado (que denominaremos ML1) que almacene representaciones equivalentes a las del lenguaje L1 y deberá construir distintas herramientas de transformación de estos documentos. Una de ellas traducirá de ML1 a L1.

PROGRAMA DE CONTENIDOS

BLOQUE 1: MARCO GENERAL PARA EL PROCESAMIENTO DE LENGUAJES

TEMA 1: CONCEPTOS BÁSICOS

- Metalenguaje
- Uso del metalenguaje para definir un lenguaje
  - Nivel Léxico / Nivel Sintáctico
  - Nivel Semántico → Vocabulario válido
    - Ámbitos de validez / Homonimia, sinonimia y polisemia
- Procesamiento de documentos
  - Representación intermedia
  - Operaciones sobre la representación intermedia
    - Reescritura
      - Transformación (Reescritura en el mismo lenguaje)
      - Generación de código (Reescritura en otro lenguaje)
    - Interpretación
      - Máquinas virtuales
- Técnicas de construcción de procesadores de lenguajes

BLOQUE 2: LENGUAJES DE PROGRAMACIÓN

TEMA 2: HERRAMIENTAS PARA EL PROCESAMIENTO DE LENGUAJES DE PROGRAMACIÓN

- Metalenguaje: Notación EBNF
  - Nivel Léxico / Nivel Sintáctico
- Técnicas y herramientas para la construcción de procesadores de lenguajes de programación
- Nivel Semántico
  - Tablas de símbolos: Tipos / Ámbito de validez de una palabra
- Interpretación
  - Máquinas virtuales
- Generación de código
  - Organización de la memoria
- Optimización de código

TEMA 3: LENGUAJES IMPERATIVOS

- Lenguaje fuente: Subconjunto de Pascal
  - Nivel Léxico / Nivel Sintáctico
  - Nivel Semántico → Tabla de símbolos
- Máquina virtual: Máquina de código-P
- Funciones de compilación
  - Asignaciones y expresiones
  - Flujo de control
  - Asignación de memoria
  - Procedimientos
  - Programa principal
- Optimización

TEMA 4: LENGUAJES FUNCIONALES

- Introducción a los lenguajes de programación funcional
- Lenguaje fuente: LAMA
  - Nivel Léxico / Nivel Sintáctico
  - Nivel Semántico → Entornos y ligaduras
- Máquina virtual: MAMA
- Funciones de compilación
  - Expresiones programas
  - Expresiones simples
  - Compilación de ocurrencias de variables
  - Definiciones de funciones
  - Aplicación de funciones
  - Construcción y evaluación de clausuras
  - Expresiones *letrec* y variables locales
- Implementación de listas

TEMA 5: LENGUAJES DE PROGRAMACIÓN LÓGICA

- Introducción a los lenguajes de programación lógica
  - Fundamentos de lógica
  - Unificación
  - Ejecución de programas lógicos
- Lenguaje fuente: Prolog
  - Árbol de prueba
  - Entornos
- Máquina abstracta de Prolog: WiM
- Funciones de compilación
  - Objetivos
  - Términos cabeceras
  - Compilación de cláusulas
  - Retroceso (*backtracking*)
  - Procedimientos, programas y consultas
- Optimización

TEMA 6: LENGUAJES ORIENTADOS A OBJETOS

- Conceptos de lenguajes orientados a objetos
  - Objetos
  - Clases
  - Herencia
  - Generalización
  - Encapsulación
- Máquina virtual Java: JVM
  - Verificación de código e Interpretación abstracta
- Lenguaje fuente: JAVA
- Funciones de compilación
  - Compilación de métodos
  - Esquemas para una compilación de herencia
  - Generalización

BLOQUE 3: LENGUAJES DE MARCADO

TEMA 7: HERRAMIENTAS PARA EL PROCESAMIENTO DE LENGUAJES DE MARCADO

- Metalenguaje: SGML-XML
- Uso del metalenguaje para definir un lenguaje: DTDs y XML-Schemas
  - Nivel Léxico / Nivel Sintáctico
  - Nivel Semántico → Vocabulario válido
    - Ámbitos de validez: Espacios de nombres
    - Homonimia, sinonimia y polisemia
- Procesamiento de documentos
  - Representación intermedia: Modelo DOM
  - Operaciones sobre la representación intermedia
    - Reescritura: XSLT
      - Transformación (Reescritura en el mismo lenguaje)
      - Generación de código (Reescritura en otro lenguaje)
    - Interpretación
      - Técnicas de construcción de programas que implementen instrucciones de procesamiento.
      - Máquinas virtuales
- Técnicas de definición e implementación de Dominios de Conocimiento.

TEMA 8: EJEMPLOS DE LENGUAJES DE MARCADO

- Lugares comunes en la WEB para las definiciones de lenguajes de marcado.
- Ejemplos de aplicación

REFERENCIAS.

[AHO86] Aho, A.V.; Sethi, R.; y Ullman, J.D. *Compilers: Principles, techniques and tools*. Addison Wesley, 1986. (Existe traducción al castellano).

[APP98] Appel, Andrew W. *Modern Compiler Implementation in Java/C*. Cambridge University Press, 1.998.

[DOM] http://www.w3.org/DOM/. Ver:

- Document Object Model (DOM) Requirements W3C Working Draft *12 April, 2000* (http://www.w3.org/TR/2000/WD-DOM-Requirements-20000412/)
- Document Object Model (DOM) Level 2 Specification Version 1.0 W3C Candidate Recommendation *10 May, 2000* (http://www.w3.org/TR/DOM-Level-2/)

[KOG91] Kogge, Peter M.. *The architecture of symbolic computers*, Mc Graw-Hill, 1.991.

[MARC99] Marchal, Benoit. *XML by example* Edt. QUE, 1999.

[MARU99] Maruyama, Hiroshi; Uramoto, Naohiko; Tamura, Kent. *XML and Java : developing Web applications*. Addison Wesley, 1999.

[SCH85] Schreiner, Axel T.; Friedman, George H. Jr. *Introduction to Compiler Construction with UNIX*. Prentice Hall, 1.985

[WIL95] Wilhelm, Reinhard; Maurer, Dieter. *Compiler Design*. Addison-Wesley, 1995.

[W3C] World Wide Web Consortium. http://www.w3.org

[XML] eXtensible Markup Language (XML) 1.0 W3C Recommendation 10-Feb-1998 (http://www.w3.org/TR/1998/REC-xml-19980210/). Ver http://www.w3.org/XML/.

[XMLorg] http://www.xml.org/xmlorg_registry/index.shtml

[XML-Sch] http://www.w3.org/XML/Schema.htmlVer:

- XML Schema Requirements W3C Note 15 February 1999 (http://www.w3.org/TR/NOTE-xml-schema-req/)
- XML Schema Part 0: Primer W3C Working Draft, 7-04-2000 (http://www.w3.org/TR/xmlschema-0/)
- XML Schema Part 2: Datatypes W3C Working Draft 7-4-2000 (http://www.w3.org/TR/xmlschema-2/)
- XML Schema Part 1: Structures W3C Working Draft 7-4-2000 (http://www.w3.org/TR/xmlschema-1)

[XSL] http://www.w3.org/Style/XSL/. Ver:

- Extensible Stylesheet Language (XSL) Version 1.0 W3C Working Draft 27 March 2000 (http://www.w3.org/TR/xsl/)