# Lenguaje de Manejo de Componentes Educativos Reutilizables Semiestructurados

Francisco Javier Moreno Arboleda

Tesis de grado presentada como
requisito para optar al título de
Magíster en Ingeniería de Sistemas

Director: Fernando Arango Isaza
Codirector: John Freddy Duitama Muñoz

Universidad Nacional de Colombia, Sede Medellín
Facultad Nacional de Minas
Posgrado en Ingeniería de Sistemas
2003

## Agradecimientos

A Freddy Duitama y a Fernando Arango por sus sugerencias.

Dedicado a Kirsty Hawkshaw.

## Resumen

En el dominio educativo pocos educadores reutilizan *recursos educativos* cuando se enfrentan a la tarea de diseñar sus cursos. Dicha reutilización puede lograrse por medio de un entorno de *componentes*. Hasta hoy, sin embargo, el trabajo con componentes ha permanecido en el terreno de los programadores.

Se concibió un modelo para *recursos educativos* bajo un esquema de *componentes reutilizables* y su correspondiente lenguaje de manipulación. El lenguaje permite a los autores crear sus propios *componentes educativos* (**CE**), consultar información sobre los que ya existan (lo que a su vez les facilitará la concepción de nuevos CE's), modificarlos y eliminarlos.

El modelo fue probado con un prototipo mediante el cual se acelera el proceso de creación de *cursos* lo cual es una labor dispendiosa para sus autores.

## Abstract

In the educational field, few teachers reuse educational resources when they are facing the task of designing his courses. Such reuse can be achieved through an environment of components. Nowadays, nevertheless, the work with components has remained in the field of the programmers.

A model for educational resources with a reusable environment of components and its corresponding manipulation language were conceived. The language allows the authors to create his own educational components (CE), to find information on the ones that already exist (at the same time it will facilitate the creation of new CE's), to modify them and to destroy them.

The model was tested with a prototype which accelerates the courses creation process which is a hard work for its authors.

## Tabla de Contenido

**1 Definición del problema a trabajar**

1.1 Introducción 1
1.2 Impacto esperado 3
1.3 Objetivos 3
1.3.1 General 3
1.3.2 Específicos 3
1.4 Resultados Esperados 4

**2 Estado del arte**

2.1 Modelos de Componentes Reutilizables 5
2.1.1 Introducción 5
2.1.2 Presentación de los principales modelos de componentes 9
2.1.2.1 Darwin 9
2.1.2.2 Unicon 10
2.1.2.3 Olan 12
2.1.2.4 COM 13
2.1.2.5 JavaBeans 15
2.1.2.6 CORBA 17
2.1.3 Observaciones y otros modelos 19
2.2 Sistemas relacionados con recursos educativos 20
2.2.1 Introducción 20
2.2.2 Panorama General de los Sistemas Educativos 21
2.2.2.1 El modelo del usuario 22
2.2.2.2 Las técnicas de adaptación 22
2.2.2.3 Propuestas para los metadatos educativos y algunas experiencias en este campo 23
2.2.2.3.1 Dublin 23
2.2.2.3.2 ARIADNE 25
2.2.2.3.3 IEEE LOM 25

2.2.3 Principales sistemas educativos analizados 26
2.2.3.1 ELM-ART: Episodic Learner Model - The Adaptive Remote Tutor 26
2.2.3.1.1 Experimentando con el demo 28
2.2.3.2 Multibook 29
2.2.3.3 AHA: Adaptive Hypermedia Architecture 31
2.2.3.3.1 La creación de un recurso educativo en AHA 33
2.2.3.3.2 Adición de reglas a AHA 34
2.2.3.4 Interbook 35
2.2.3.4.1 La labor de creación de un Interbook 36
2.2.3.4.2 Experimentando con Interbook 39
2.2.3.5 KBS 39
2.2.3.6 Resumen 42
2.3 Estado del arte de las bases de datos semiestructuradas 42
2.3.1 Las Bases de Datos Semiestructuradas y la Web 42
2.3.2 Los Datos Semiestructurados 45
2.3.3 El Papel de XML 46
2.3.4 Lenguajes de consulta para datos semiestructurados 50
2.3.5 El proyecto LORE 52
2.3.6 Lenguajes de Consulta para XML 53
2.3.7 Otras tecnologías relacionadas con XML 56

**3 Modelo de Componentes Educativos Propuesto**

3.1 Introducción 58
3.2 Definición de CE 59
3.3 Modelo del dominio 62
3.3.1 Relación jerárquica 62
3.3.2 Relación retórica 62
3.3.3 Relaciones entre los CE's y los conceptos del dominio 64
3.3.3.1 Relación de $R_{subject}$ 64
3.3.3.2 Relación $R_{prerequisite}$ 65
3.3.3.3 Restricción de $R_{subject}$ y $R_{prerequisite}$ 65
3.3.3.4 Observaciones acerca del modelo 66
3.4 Granularidad de los CE's y relaciones entre los conceptos del dominio 67
3.5 Los metadatos utilizados 68

3.5.1 Elementos de LOM no usados en nuestro modelo 68
3.5.2 Extensión dinámica de LOM 69
3.6 Estructura de un CEC 71
3.6.1 El nodo concepto 72
3.6.2 Arcos 74
3.7 Setting para un CE 74
3.8 Operadores de creación de CE's 75
3.8.1 Operador Sequence (CE1, CE2,..., CEn ) 76
3.8.1.1 Restricción de secuencia 76
3.8.1.2 Elementos del CEC creado 76
3.8.2 Operador Union (CEC1, CEC2, ... , CECn ) 79
3.8.2.1 Caminos en el grafo Gc 80
3.8.2.2 El aspecto de compatibilidad 80
3.8.2.3 Elementos del CEC creado 80
3.9 Construcción de CE típicos 81
3.10 Elementos de lógica descriptiva para la especificación de los CE's 82
3.10.1 Representación de conceptos y algunos constructores 84
3.10.2 Especificación de los CE's en términos de LD 85
3.10.2.1 Parte declarativa de un tópico 86
3.10.2.2 Parte declarativa de un ejercicio y de un workshop 86
3.10.2.3 Parte declarativa de una lección 87
3.10.2.4 Parte declarativa de un curso 87
3.11 Lenguaje de Manipulación de CE's 88
3.11.1 Introducción 88
3.11.2 Operador de Selección 88
3.11.3 Operador de Borrado 93
3.11.4 Operador de Actualización/Modificación 94
3.11.4.1 Add 94
3.11.4.2 Change 96
3.11.4.3 Delete 97
3.11.4.4 Observaciones sobre el operador de Actualización 98
3.12 Operador Show 99
3.13 Operador de Obtención de Caminos: GetPaths 101
3.13.1 Definiciones 101

3.13.2 Propósito y sintaxis del operador 101
3.13.3 Objetivo de uso de este operador 104
3.14 Operador de Replace 104
3.14.1 Sobre el nivel de dificultad 106
3.15 Operador Find Similar 107
3.15.1 Fundamentos: El operador de subsumption de la LD 107
3.15.2 Algoritmo del tabloide para probar subsumption 107
3.15.3 Objetivo y sintaxis del operador 110

**4 Aporte**

4.1 Utilización de la teoría de grafos para la composición de los CEC's 113
4.2 El lenguaje de manipulación 114
4.2.1 El lenguaje propuesto frente a otras aproximaciones 114
4.2.1.1 Comparación con otros Sistemas Educativos 114
4.2.1.2 Una analogía con los Web services 115
4.3 El aspecto de los metadatos extensibles 119
4.4 Otros Aportes 120
4.4.1 Introducción de lógica descriptiva para los recursos educativos 120
4.4.2 Fusión del concepto componente (en el sentido de componente de software) con los recursos educativos y explotación de tal modelo 120
4.4.3 Clasificación automática 122
4.4.4 Contribución de los operadores propuestos para el proceso de adaptación 122

**5 Presentación del prototipo**

5.1 Introducción 123
5.2 Funcionalidad 123
5.2.1 La creación de un CEP 123
5.2.2 La creación de un CEC 124
5.2.3 Los operadores de manipulación 127
5.2.4 Sobre los operadores Find Similar, Replace y GetPaths 128
5.3 Aspectos técnicos 128
5.4 Conclusiones 129

**6 Conclusiones y futuras líneas de trabajo**

6.1 Conclusiones 130

6.2 Futuras líneas de trabajo 131

**Apéndices**

**Apéndice 1:** Los metadatos del LOM 132

**Apéndice 2:** Conexión con el módulo adaptativo 137

**Apéndice 3:** Gramática BNF del lenguaje 141

**Glosario** 146

**Bibliografía** 148

**Listado de Figuras**

**Figura 1.1:** Conformación de un nuevo CE 2

**Figura 2.1:** Representación de un Componente en Olan 13

**Figura 2.2:** El modelo binario de los componentes COM 13

**Figura 2.3:** Representación de un JavaBean 15

**Figura 2.4:** Representación de un objeto CORBA 18

**Figura 2.5:** Sección 1 del curso de LISP de ELM-ART II 28

**Figura 2.6:** Creación y ejemplo de un Interbook 38

**Figura 2.7:** Relación de un hyperbook y el modelo de conocimiento 40

**Figura 3.1:** Representación gráfica de un CE 59

**Figura 3.2:** Estructura de una input (entrada) y de una Output (salida) 60

**Figura 3.3:** Representación gráfica de un CEC 61

**Figura 3.4:** Comportamiento "black box" de un CE 61

**Figure 3.5:** Fragmento del modelo de dominio para Estructuras de Datos 63

**Figura 3.6:** Rsubject de un CE 65

**Figura 3.7:** Rprerequisite de un CE 65

**Figura 3.8:** CE's, árbol de dominio y relaciones Rsubject y Rprerequisite 66

**Figura 3.9:** Estructura de un nodo concepto 72

**Figura 3.10:** Relación entre un nodo concepto y su CE sugerido 73

**Figura 3.11:** Fragmento de modelo de dominio y CE mostrando los labels entre nodos concepto 74
**Figura 3.12:** Visualización de entradas y salidas en un CEC 75
**Figura 3.13:** CEC $c_k$ 77
**Figura 3.14:** Representación de ck en términos de operadores de composición: Sequence y Union 77
**Figure 3.15:** Composición para el operador Sequence 79
**Figura 3.16:** CEC resultante de: Union (Sequence (c1, c2), Sequence (c1, c3)) 80
**Figura 3.17:** CE's típicos 82
**Figura 3.18:** Un ejemplo de red de conocimiento para LD 83
**Figura 3.19:** Especificación de un CE usando LD 86
**Figura 3.20:** CEC $c_5$ 100
**Figura 3.21:** CEC $c_{13}$ 101
**Figura 3.22:** CEC $c_{18}$ 102
**Figura 3.23:** CEC $c_{17}$ 103
**Figura 3.24:** CEC $c_{38}$ 103
**Figura 3.25:** Proceso del operador Replace 105
**Figura 5.1:** Interfaz web para la creación de un CEP 124
**Figura 5.2:** Interfaz web para la creación de un CEC 125
**Figura 5.3:** Interfaz web para el operador Get Components 126
**Figura 5.4:** Interfaz web para el operador Find Similar 128
**Figura AP1:** Arquitectura del Sistema Global 137
**Figura AP2:** Resumen del Proceso de Adaptación: 1 Concepto 139
**Figura AP3:** Resumen del Proceso de Adaptación: Varios Conceptos 140

# 1 Definición del problema a trabajar

## 1.1 Introducción

Se estima que en el dominio educativo menos de un 1% de los educadores [Roschelle 99] han aprovechado o reutilizado recursos educativos cuando se enfrentan a la tarea de diseñar sus cursos, lecciones, evaluaciones, talleres etc. Dicha reutilización puede lograrse por medio de componentes. Hasta hoy, sin embargo, el trabajo con componentes ha permanecido en el terreno de los programadores. Lo anterior es debido a la distancia existente entre los expertos de un dominio y los expertos en programación, ya que la definición y manipulación de componentes se hace pensando más en el trabajo del programador que en facilitar el trabajo del experto de un dominio específico.

Es decir se requiere un entorno de trabajo con CE's, amigable acompañado de mecanismos de definición, consultas expresivas y de reutilización, lo suficientemente simple para que pueda ser utilizado por usuarios no expertos en programación, minimizando la presencia o necesidad de un intermediario, es decir de un programador.

De otro lado el crecimiento acelerado de Internet facilita y promueve el intercambio de información y por ello se hace necesario impulsar el desarrollo por componentes por parte de estos expertos, de tal forma que éstos se *concentren en crear CE's de alta calidad*, para así compartirlos e intercambiarlos con estudiantes y colegas, sin requerirles conocimientos profundos de programación a unos y otros.

Varios de los lenguajes desarrollados hasta la fecha para describir componentes (de cualquier dominio, véase sección 2.1.2) son demasiado formales y poco accesibles para una persona que no sea programadora. Además se concentran y preocupan por garantizar la *compatibilidad de tipos* durante la conexión entre componentes, este aspecto, de hecho, fundamental para los componentes de Software, sin embargo no lo es en el campo de los CE's. La preocupación estará orientada a aspectos de conexión entre ellos desde el punto de vista de topología (secuencias significando tablas de contenido) y aspectos de precedencia (prerrequisitos). Un ejemplo muy sencillo se muestra a continuación, supóngase que va a ser ensamblado un curso de Estructuras de Datos y se tienen CE's

que desarrollan los 3 grandes temas del curso: pilas, colas y árboles los cuales deben ser vistos en ese orden, por lo tanto se tendría algo como:

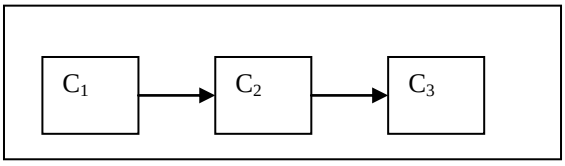



Figura 1.1: Conformación de un nuevo CE

Donde el componente C1 desarrolla el tema de pilas, C2 desarrolla el tema de colas y C3 el de árboles. Se tiene como resultado un nuevo componente *curso*, en el cual las conexiones (flechas) indican en vez del concepto de compatibilidad de conexión, el concepto de precedencia. (Por ejemplo el componente C1 debe ser visto antes que C2). La secuencia C1,C2 y C3 indica asimismo una *tabla de contenido*, en este caso el curso como tal.

Adicionalmente los lenguajes examinados no son ricos en sí mismos en elementos que faciliten la búsqueda de componentes existentes (véase sección 2.1.3) pues se concentran en permitir al diseñador expresar las características técnicas (funcionales) de cada componente y no enfatizan en la posibilidad de agregar elementos de información adicional [Goguen 86] a los componentes, como por ejemplo:

- Palabras claves que identifiquen al componente.
- Información sobre el (los) autor (es), fecha de creación, propósito, etc.
- Descripción de la funcionalidad que poseen.

Al suplir estas deficiencias, se facilitará la labor de localización, creación y reutilización de CE's en un ambiente amigable y basado en una representación flexible (datos semiestructurados) y de intercambio estándar en la web accesible a los expertos del dominio educativo y no sólo manipulado por programadores.

## 1.2 Impacto esperado

- Mejoramiento de la productividad, expresado en la alta calidad en el desarrollo y ensamblaje de CE's, al permitir que los expertos educativos (usuarios finales) se enfoquen en su dominio y no en tareas de programación. Se espera mejorar así el bajo porcentaje de desarrollo de recursos educativos reutilizables por parte de estos expertos, el cual es menor a un 1%.
- Se espera contribuir con este proyecto al fomento de cursos virtuales.
- Las entidades beneficiadas con el proyecto pueden ir desde unidades académicas locales (profesores, departamentos, escuelas) hasta el uso de estos cursos virtuales desarrollados con CE's en programas de regionalización. (Ejemplo: Programa de Suroeste Universidad de Antioquia).
- Se acrecentará el intercambio de CE's vía web entre profesores (colegas) y estudiantes.

## 1.3 Objetivos

### 1.3.1 General

Crear un lenguaje de manejo de componentes educativos reutilizables semiestructurados.

### 1.3.2 Específicos

- Analizar y especificar la configuración del componente, con sus posibles estructuras subyacentes (partes semiestructurada y estructurada) y su naturaleza.
- Diseñar las operaciones semántica y sintácticamente (en notación BNF) de búsqueda, actualización, eliminación más los típicos operadores como son composición, compatibilidad y remplazo de componentes educativos.
- Probar el lenguaje desarrollado mediante un prototipo que implemente las operaciones propuestas.

Los operadores de composición son aquellos que permiten conformar componentes "más grandes" a partir de otros, el aspecto de compatibilidad expresa que conformaciones son válidas y el remplazo significa bajo que circunstancias y reglas un componente puede ser cambiado por otro.

## 1.4 Resultados Esperados

Modelo propuesto de CE's y su correspondiente lenguaje de manejo expresado en notación BNF, desarrollo de un prototipo y se espera publicar un artículo en revista o congreso para dar a conocer los resultados obtenidos.

# 2 Estado del arte

## 2.1 Modelos de componentes reutilizables

### 2.1.1 Introducción

La definición de lo que es un componente reutilizable no está estandarizada [Barroca 00], varias definiciones han surgido y aunque la gran mayoría concuerdan en decir que:

**Definición 1** [Barroca 00]: Un *componente* es una *caja negra* que *encapsula servicios* detrás de *interfaces* bien definidas. Estas interfaces tienden a ser muy restringidas en su naturaleza, reflejando un *modelo de compatibilidad de conexión* soportado por una estructura de componentes. Los componentes no son usados aisladamente, mas bien tienden a ser ensamblados con otros.

Algunos de los conceptos involucrados allí pueden producir confusión debido a las posibles interpretaciones que se pueden derivar. La siguiente definición [Sreedhar 00] traída de ACOEL, el cual es uno de los tantos lenguajes que han sido concebidos para describir e interconectar componentes, expresa que:

**Definición 2:** Un *componente* consiste de un conjunto de *puertos tipados* de entrada y de salida. Los puertos de entrada de un componente consisten de todos los *servicios* que el componente provee, mientras que los puertos de salida son todos los servicios que el componente requiere para su correcto funcionamiento.

Aunque en esta definición se hace una diferencia, la cual se aclarará mas adelante, entre los términos puertos y servicios, es normal encontrarlos como sinónimos en la literatura [Barroca 00]. Otros términos como entradas (y salidas), interfaces y métodos también son frecuentes encontrarlos en la definición de un componente, los cuales pueden producir la idea inicial de referirse a una *misma cosa*.

Volviendo con ACOEL, se continúa diciendo que un *puerto tipado* puede ser de un tipo de interfaz o ser un tipo delegado. Un *tipo de interfaz* consiste de un conjunto de métodos y de constantes con nombre, mientras que un *tipo delegado* es un alias de un método encapsulado.

Estos términos completan el concepto de componente en ACOEL. Un ejemplo hará las cosas más claras. De acuerdo a las definiciones dadas en ACOEL un componente se define así:

```
//Definición de interfaz
interface nom_interface1 {
 //Prototipos de los métodos que componen la interfaz y constantes con nombre.
 void remove(integer position, element z),
 //otros métodos
 ...
}

//Definición de componente
Component nom_comp1
{
 //Declaración de las interfaces que posee el componente, indicando para cada una su
   nombre, tipo (entrada o salida) y a cual tipo de interfaz pertenece:
   In varint11 nom_interface1 // Se declara la interfaz de entrada varint11 de tipo
                                        interface nom_interface1
 //Definición de variables del componente:
   int count = 0;
 //Método constructor:
  nom_comp1(){ ...}
 //Implementación de todos los métodos correspondientes a los prototipos especificados
   en los tipos de interfaces, utilizadas por el componente:
  void remove(integer position, element z) {...}
}
```

Hasta ahora la similitud con el concepto de objeto de la programación orientada a objetos es enorme: Un objeto posee atributos (estado) y métodos (comportamiento).

Los componentes se pueden interconectar por medio de sus interfaces, siempre y cuando sean del *mismo tipo* y sigan el protocolo de conexión de una de salida con una de entrada, ejemplo:

```
Component nom_comp3{
  Out varint31 nom_interface1,
  Out varint32 nom_interface2,
  //Constructor
  Nom_comp3()
 {
```

```
  //Se crea una instancia de tipo nom_comp1:
  Nom_comp1 mycomp1= new nom_comp1();
  //Se crea una instancia de tipo nom_comp2:
  Nom_comp2 mycomp2= new nom_comp2();
  //Se realiza la interconexión:
  connect varint31 to mycomp1.varint11,
  connect varint32 to mycomp2.varint21
 }
}
```

Se asume la declaración de un segundo componente nom_comp2, el cual debe poseer una variable varint21 de entrada de tipo interfaz nom_interface2 (debe declararse también).

Para confrontar las definiciones, considérese el siguiente ejemplo de componentes extraído de Durra [Barbacci 89]. Durra es otro modelo de componentes en el cual éstos se denominan *tareas*. Cada tarea ofrece una interfaz constituida por puertos de entrada y de salida. Los puertos tienen nombre, son unidireccionales y tienen un tipo de datos asociado (definido dentro de Durra) lo cual permite verificación de compatibilidad de conexiones entre componentes.
Como es típico en estos modelos, las tareas se pueden interconectar produciendo nuevos componentes (compuestos). Considérese el siguiente ejemplo:

```
//Se definen algunos tipos de datos:
type byte is size 8
type scalar is size 4*sizeof(byte)
type message is union (byte,scalar)

//Definición de una tarea productor en Durra:
task producer
  ports myoutput: out message;
attributes
  processor = "sun 4"
  procedure_name = "producer"
  library = "/usr/durra/srclib"
end producer;
```

Aquí se tiene un componente de nombre *producer* el cual posee un puerto de salida llamado *myoutput* de tipo de datos *message* (definido arriba).
Los *atributos* aquí se refieren a información técnica sobre el componente, en este caso significa que esta tarea ejecuta sobre un procesador *sun4*, que la implementación se llama *producer* y que está ubicada en */usr/durra/srclib*.

Surgen entonces diferencias y similitudes entre estas las definiciones 1 y 2:
Nótese que, *en general*, la definición 2 está conforme a la definición 1.

Sin embargo se observa una diferencia entre lo que es un puerto:

En ACOEL un puerto pertenece a un tipo de interfaz, la cual es un conjunto de métodos y de constantes.

En Durra, la interfaz de un componente está constituida por puertos de entrada y de salida y cada puerto pertenece a un tipo de datos o dominio (el cual puede haber sido formado por otros tipos de datos primitivos).
Además en ACOEL los atributos proporcionan una descripción del estado del componente mientras que en Durra estos se refieren mas bien a información descriptiva o técnica.

Una diferencia más sutil radica en el concepto de entrada y salida: En ACOEL a través de los puertos de entrada se accede a los servicios que el componente provee, en Durra el concepto de puerto de entrada se refiere mas bien al tipo de dato que el componente necesita para poder realizar su función, o sea es lo que requiere. Igual sucede con el concepto de salida.

Y la diferencia más notable, en Durra existe un solo elemento de "comportamiento", es decir, se hace referencia al archivo ejecutable que es el que implementa la funcionalidad del componente. En ACOEL existen realmente varios elementos de comportamiento asociados al componente: Los métodos, los cuales están contenidos en sus interfaces y que cada componente implementa.

Por ejemplo en ACOEL se puede concebir un componente *matriz* el cual a través de sus interfaces posee métodos de sumar, multiplicar e invertir matrices.

Siguiendo un modelo estilo Durra, se tendrían 3 componentes: Uno dedicado a sumar matrices, otro para multiplicarlas y otro para invertir.

Nótese lo confuso que puede ser la literatura sobre componentes y como los mismos términos se utilizan con significado diferente de acuerdo a cada modelo y hasta ahora sólo se han analizado dos modelos de componentes.

### 2.1.2 Presentación de los principales modelos de componentes

#### 2.1.2.1 Darwin

El concepto fundamental en Darwin [Magee 94], es el componente. Este es descrito por una interfaz que contiene los servicios que ofrece y requiere. Existen 2 tipos de componentes: los primitivos y los compuestos.
Los primitivos encapsulan el código de la función que les es atribuida. Los compuestos, que son interconexiones de componentes, encapsulan las instancias de sus componentes miembros, es decir los compuestos son las unidades de configuración. En efecto una aplicación es llevada a cabo por medio de un compuesto. Los componentes miembros de un compuesto cooperan mediante un modelo de comunicación establecido a través de la interfaz.

Una característica importante en Darwin es la capacidad de crear dinámicamente componentes miembros de un compuesto (aplicación). Esto implica que la arquitectura de una aplicación no es estática y varía de acuerdo a como sea parametrizada. Veamos un ejemplo concreto:

```
Component pipeline (int n) {
//interfaz
provide input;                  //Entrada del compuesto
require output;                 //Salida del compuesto
//implementación
array F[n]: filtro;             //Se define un array para ensamblar componentes primitivos de
                                // tipo filtro
forall k:0 .. n-1 {
 inst F[k];                     //Se crea la instancia de un filtro
 bind F[k].output – output;     //Liga la salida de cada filtro con la salida del compuesto
                                //Nótese que se usa el signo – para enlazar.
 when k<n-1
bind F[k].right – F[k+1].left;  // Liga la salida del filtro F[k] con la entrada del filtro
                                // F[k+1]
 }
bind input F[0].left;           //Liga la entrada del primer filtro con la entrada del compuesto.
bind F[n-1].right – output      //Liga la salida del último filtro con la salida del compuesto.
}
```

Se crea un componente compuesto llamado *pipeline* constituido por el ensamblaje en serie de un grupo de componentes primitivos de tipo filtro, donde cada uno posee una entrada llamada left y un par de salidas llamadas right y output. A una entrada left se le puede conectar una salida right (compatibilidad de conexión). Nótese que la salida del componente compuesto es la unión de todas

las salidas llamadas output de los primitivos (mas la salida right del último filtro) y su entrada está dada por la entrada left del primer filtro. Como puntos negativos se menciona que en Darwin la integración de código ya existente es compleja y que la utilización del ambiente no es trasparente, ya que se requieren conocimientos relativamente altos de programación, de hecho su sintaxis tiene base en el lenguaje C++.

#### 2.1.2.2 Unicon

Unicon (Universal Conector Support) [Shaw 95] propone un lenguaje de definición de una arquitectura donde los principales elementos son los componentes y los conectores. Una aplicación está formada por un ensamblaje de componentes. Un componente es una entidad que encapsula las funciones de un modelo lógico.

Cada componente exhibe a través de una interfaz sus propiedades, es decir, las funciones y datos accesibles y las funciones y datos requeridos para llevar a cabo la instanciación, el tipo de componente, funcionalidad, características de desempeño etc. Las entidades visibles en la interfaz de un componente son denominadas players: Es decir estos son los puntos a través de los cuales un componente se puede comunicar con el exterior. La interfaz incluye las propiedades de interacción de sus players.

Los conectores contienen la información concerniente a las reglas de interconexión de componentes tales como el protocolo, la especificación de intercambio de datos, las transformaciones eventuales de cambios de formatos, tipos de interfaces con las que pueden interactuar etc. Es decir gobiernan la interacción entre los componentes.
Los conectores poseen igualmente propiedades como son su tipo, desempeño etc. Las entidades visibles en un conector son denominadas roles. La interfaz de los componentes incluye reglas que dicen los players con cuales roles pueden conectarse.

Los componentes primitivos son aquellos que se implementan a través de un lenguaje de programación convencional, scripts, etc. Los componentes compuestos *definen una configuración en una notación independiente a la de un lenguaje de programación*.

La notación debe permitir identificar los componentes constituyentes y conectores, y enlaces entre players y roles. Finalmente se debe verificar que la composición resultante si satisfaga las especificaciones de la interfaz de los componentes y los protocolos de los conectores.

Los conectores pueden ser llamados a procedimientos, protocolos de red estandarizados etc. Por ejemplo un conector puede ser un procedimiento para traspaso de datos (mensajes por ejemplo), en el cual existen dos roles quien invoca (emisor) y quien recibe(receptor).
La descripción general de un componente en Unicon y un ejemplo de un primitivo llamado *sort* luce de la siguiente manera:

```
Component nom_comp
Interface is
  Type type_composant
  <lista de propiedades del
   componente>;
  <lista de players del componente>;
 End Interface

Implementation is
 <lista de propiedades>;
  <Implementación Primitiva |
   Implementación Compuesta>
End Implementation

End nom_comp
```

```
Component sort
  Interface is
   Type filter
    Player Input is StreamIn
      Signature ("line")
      Portbinding (stdin)
    End input
    Player Output is StreamOut
      Signature ("line")
      Portbinding (stdout)
    End output
End Interface

  Implementation is
    Variant sort in "sort"
     ImplType (executable)
     InitActuals("-f")
   End sort
  End Implementation

End sort
```

Igualmente **se debe aprender este lenguaje de descripción de componentes para llegar a construir aplicaciones en Unicon** (no se entrará aquí en detalles para ello puede verse [Shaw 95] donde se presenta también un ejemplo de un componente compuesto).

La terminología a veces es confusa (de acuerdo a las fuentes consultadas [Shaw 95, Marvie 00] ) ya que se habla de entidades, elementos, propiedades, protocolos etc., y no se establece con claridad que significa exactamente cada término, sin embargo sigue una filosofía de trabajo similar a los otros modelos examinados.

#### 2.1.2.3 Olan

El ambiente Olan [Marvie 00] permite configurar e instanciar (en Olan se usa el término instalar) aplicaciones. El término configuración designa la especificación de la arquitectura de la aplicación y su modo de ejecución. Los elementos centrales de Olan son los componentes y los conectores. Dos tipos de componentes son definidos:
Los primitivos son las unidades de encapsulamiento de la lógica existente. Estos se conciben mediante un lenguaje de programación convencional.
Los compuestos proveen la estructuración de una aplicación en componentes cooperativos. Los componentes forman una jerarquía totalmente reutilizable en diversas aplicaciones.

Los conectores son utilizados para especificar el protocolo de comunicación entre componentes; permiten conectarlos, adaptar el flujo de datos y establecer la coordinación entre las interfaces interconectadas. Son de tipo SynCall, AsynCall y RandSyncCall.
En Olan una aplicación puede ser distribuida, es decir, se debe especificar cada componente miembro el sitio donde debe ser instalado, esto se logra mediante reglas de administración las cuales establecen los atributos de instalación.

Olan propone un lenguaje llamado OCL(Olan Configuration Language) para describir la estructura de los componentes. Un ejemplo de tal lenguaje extraído de [Marvie 00]:

```
Component <identifier> {
  Interface <interface_identifier>;
  Implementation <implementation_identifier>
  Management <management_identifier>
}
```

En forma gráfica se presenta el siguiente ejemplo:

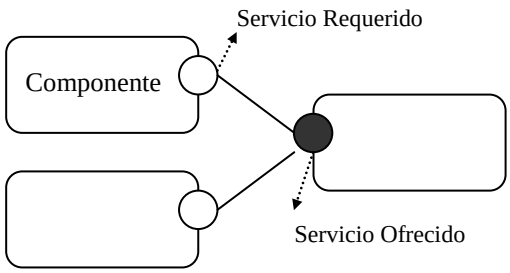



Figura 2.1: Representación de un Componente en Olan

Entre las ventajas de Olan se tiene que su lenguaje permite expresar estructuras dinámicas y el aspecto de la distribución, sin embargo continúa siendo un lenguaje propietario y los atributos de instalación son fijos (no extensibles).

#### 2.1.2.4 COM

COM (Common Object Model) representa la propuesta de Microsoft en términos de componentes lógicos. La propuesta de COM surge como resultado de la evolución progresiva del ambiente MS-Windows.

En COM un componente es esencialmente una entidad *binaria* para la cual es definida una interfaz y el modo de interacción. Estos dos elementos se canalizan a través de un puntero, el cual permite accesar las funciones ofrecidas por las librerías.

COM no impone restricciones sobre la composición de un componente: este puede estar implantado sobre una o más clases, sobre una biblioteca de procedimientos o funciones etc.

Una interfaz Iunknown está disponible para un componente. Esta interfaz ofrece una operación QueryInterface que permite descubrir las interfaces ofrecidas y navegar entre ellas.

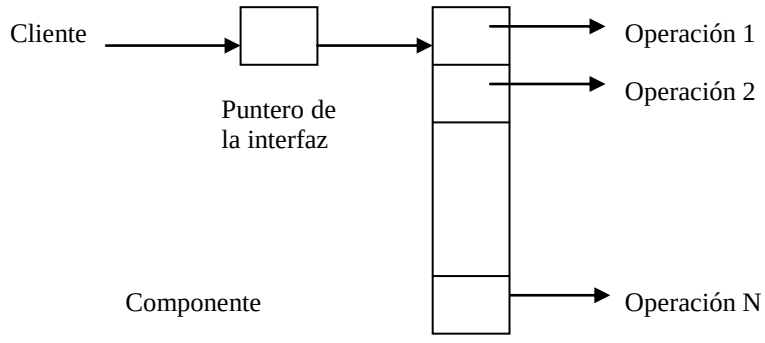



Figura 2.2: El modelo binario de los componentes COM

Un componente puede disponer de la interfaz Idispatch. Esta interfaz agrupa la utilización de todas las operaciones de un componente en una única operación: invoke. Esta permite la utilización sistemática y dinámica de un componente a partir de un lenguaje como Visual Basic.
De manera opcional un componente puede declarar interfaces Outgoing las cuales son medios para que una instancia de un componente pueda proporcionar información, mensajes e invocación directa y son explotadas por el componente si él es conectado a las instancias de componentes que ofrezcan estas interfaces.
La implantación de un componente COM se resume en la definición de una biblioteca que ofrece una o más interfaces y la implantación del comportamiento de estas. Estas bibliotecas se convierten en DLL's (Dynamically Linked Libraries).
La creación de instancias de componentes se logra a través de un patrón de concepción "fabrica". Una fabrica es ofrecida por un servidor de componentes. Los servidores de componentes se matriculan mediante un registro definido en COM y se lleva una lista de los servidores de componentes disponibles.

La reutilización en COM se puede lograr en dos formas: La composición (llamada *containement*) y la agregación. En el caso de la composición, un compone dispone de una referencia exclusiva a otro componente. El componente contenido (interno) es "invisible" para los usuarios que no están conscientes de que están trabajando con un componente compuesto.
En el caso de la agregación no existe la noción de contención entre componentes sino la de colaboración. Un componente puede "escoger" si acepta hacer parte de un agregado o no. Para el cliente todas las interfaces de los componentes son visibles, es decir, un agregado se comporta como un componente multi-interfaz. Los clientes utilizan directamente los componentes que conforman al agregado.
DCOM es una extensión de COM para manejar el aspecto distribuido (confrontar con Olan). También existe COM+ la cual es una extensión de COM la cual maneja objetos "livianos". Para mayores detalles puede verse [Marvie 00].

Se puede concluir que la utilización de COM es compleja, se requieren buenos conocimientos de programación, fundamentación sobre la filosofía MS Windows, conocer especificaciones de ciertas API's etc.

#### 2.1.2.5 JavaBeans

El modelo de JavaBeans es propuesto por Sun Microsystems. Los JavaBeans se presentan más como un modo de utilización del ambiente de desarrollo Java que como un modelo en sí.
La especificación de los JavaBeans define los Beans como los componentes gráficos. Sun define un Bean así [Marvie 00]:

**Definición 3**: *Un JavaBean es un componente de software reutilizable que puede ser manipulado visualmente en un builder tool. (herramienta de construcción).*
Como todo objeto Java, un Bean posee una interfaz de clase. El puede disponer de métodos y de atributos que representan sus propiedades. De hecho de la orientación gráfica de los Beans, la cooperación entre instancias se basa en la utilización de eventos.
Al igual que en Java standard los eventos son manejados de manera síncrona.
Al nivel de la interfaz, nada diferencia a un Bean de un objeto Java. Alguien interesado en las funcionalidad de un Bean podrá sin dificultad utilizarlo según el modelo tradicional de Java.

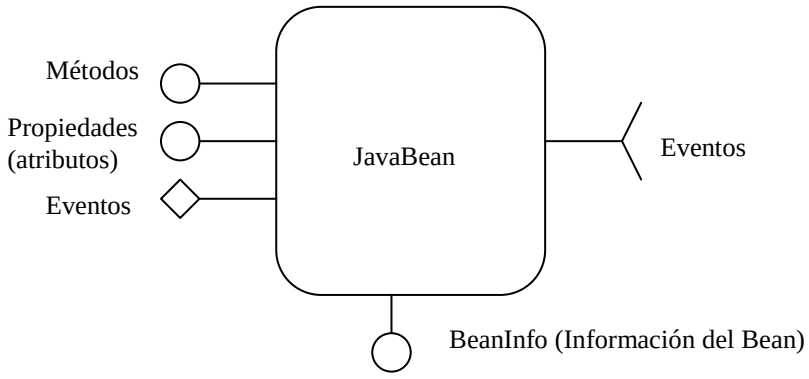


Figura 2.3: Representación de un JavaBean

La definición de un JavaBean se hace de la misma manera que la de un objeto Java. Las instrucciones adicionales son fundamentalmente aspectos de sintaxis. Los atributos son definidos por la presencia de uno de dos métodos, los que leen y los que los modifican. De la misma forma una fuente de eventos es definida por la presencia de 2 operaciones, para capturarlos y dispararlos. Veamos una interfaz para ilustrar la definición de un Bean:

```
Interface mybean {
   //Método para leer el atributo
  String getColor();
  //Manejo de los eventos
  void addmybeanListener (mybeanListener l);
  void removemybeanListener( mybeanListener l);
}
```

El aspecto de la distribución se logra a través de RMI (Remote Method Invocation). RMI ofrece un método bastante simple para hacer cooperar los objetos Java repartidos en diferentes JVM's (Java Virtual Machine). Este permite invocar los métodos sobre un objeto remoto haciendo trasparente la distribución.

Para facilitar la **búsqueda de instancias de componentes en el sistema**, Sun propone la utilización de la tecnología Jini. Jini permite registrar las instancias de las implantaciones de componentes y ofrece una operación de búsqueda. Esta operación permite recuperar una implantación vía su interfaz o vía una caracterización de esta interfaz (utilización de *keywords*).
El ambiente de ejecución de los JavaBeans se simplifica mucho: Un contenedor gráfico. Para que un JavaBean se pueda ejecutar, este hace uso de una ventana gráfica (frame, browser etc.)
Para los JavaBeans se ofrece un ambiente de desarrollo gráfico completo que permite al desarrollador llevar a cabo las fases de concepción, composición, arrastrar y soltar, escritura de código etc. Este ambiente permite leer la meta información de los Beans, contenida en la BeanInfo. Esta meta-información ofrece al desarrollador una lista de los servicios ofrecidos por los beans, guiando así la integración en una aplicación.

La fase de concepción se puede limitar a presentar el "esqueleto" que especifica las propiedades, los métodos y los puertos de producción/consumo de los eventos. La implantación del Bean puede ser hecha de 2 formas: por extensión de uno existente o una implantación nueva.
Lo que ofrece un Bean se empaqueta en un archivo de extensión .jar el cual puede ser distribuido, y allí se expresa la implantación, su interfaz y la información sobre el Bean en una clase de tipo BeanInfo.
Los Beans están entonces destinados a ser ensamblados de manera gráfica para producir Beans más complejos o aplicaciones. Es por ello que es importante explotar la información contenida en la zona de BeanInfo, sobre servicios ofrecidos y gestión de eventos.

Se tiene también los JavaBeans (EJB) los cuales representan un modelo de componentes cliente. Los Beans no son adecuados para la realización de componentes servidores, es por ello que Sun ha propuesto los EJB's. Este modelo representa un escenario de componentes servidores para concebir aplicaciones distribuidas y con manejo transaccional.

Mas que querer formular un modelo, Sun propone una metodología de utilización de su lenguaje y ambiente Java. Los JavaBeans responden bien al problema de la producción de aplicaciones al

ofrecer interfaces gráficas con base en componentes reutilizables. Igualmente explota la meta información (información de la funcionalidad y atributos) para facilitar el desarrollo. Sin embargo los Beans representan una solución propietaria que no facilita la integración de lo ya existente. Así como COM es una guía para desarrollar en una plataforma (MS Windows), JavaBeans es una guía para un lenguaje. Para mayores detalles véase [Marvie 00].

#### 2.1.2.6 CORBA

CORBA (Common Object Request Broker Architecture) es un estándar publicado por la OMG (Object Management Group) – los responsables asimismo del lenguaje OQL – para redes de objetos distribuidas y heterogéneas. La red es distribuida porque los objetos pueden residir en diferentes computadoras. La red es heterogénea porque estas computadoras pueden funcionar con diferentes sistemas operativos y porque *los objetos pueden estar implementados en diferentes lenguajes de programación.*

Las interfaces de los objetos se especifican en un lenguaje de programación de especificación llamado IDL (Interface Definition Language) el cual forma parte del estándar CORBA.

Un cliente CORBA se refiere a cualquier unidad de software que hace uso de objetos CORBA proporcionados en la red (es decir, unidades que hacen uso de objetos servidor CORBA).

El gestor de peticiones de objetos CORBA ORB (Object Request Broker) es una unidad de software que media y redirige las peticiones de los clientes CORBA hacia objetos servidor CORBA. El ORB oculta detalles técnicos de estas peticiones, haciéndolas trasparentes a los clientes. Por ejemplo, las llamadas a los métodos de los objetos servidor remotos en la red son como llamadas a métodos locales. El ORB es la parte fundamental de una implementación CORBA y mantiene toda la comunicación entre un objeto cliente y un objeto servidor. El ORB se debe ejecutar tanto en el lado cliente como en el lado servidor. El ORB implementa un conjunto de servicios específicos que facilitan las operaciones en un entorno CORBA. Por ejemplo los servicios de nombres permiten a los clientes referenciar objetos servidor, los servicios de transacciones permiten que el código cliente y los objetos servidor se dividan en unidades atómicas, denominadas transacciones y los servicios de eventos permiten disponer de datos de eventos sin que los emisores de estos datos conozcan a los receptores.

Uno de los principales objetivos de CORBA es su independencia del lenguaje. Los objetos CORBA escritos en un lenguaje de programación pueden invocar métodos de un objeto escrito en otro lenguaje de programación. Esta independencia del lenguaje es posible mediante el uso del lenguaje

"neutral" IDL. En IDL, los métodos de los objetos no se implementan, sólo se especifican. Es decir se describe el exterior de una caja negra y no se especifica nada sobre la implementación. Véase la figura 2.4.

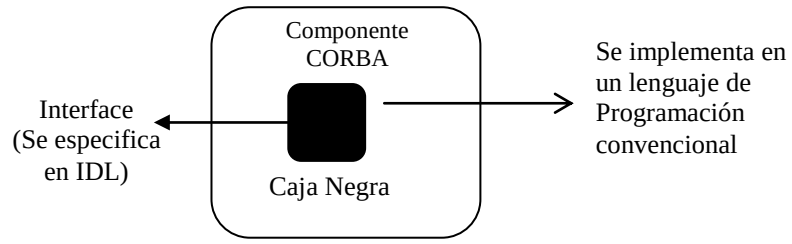



Figura 2.4: Representación de un objeto CORBA

Las interfaces de los objetos se especifican en un lenguaje de programación de especificación llamado IDL (Interface Definition Language) el cual forma parte del estándar CORBA.

Concretamente para cada objeto servidor se debe construir una interfaz IDL que incluya las constantes, los tipos, las excepciones, los atributos y los métodos asociados con el objeto. IDL fue diseñado para trabajar con C++. Luego se compila el archivo IDL que especifica la interfaz y posteriormente se implementa la interfaz en un lenguaje de programación específico.

La unidad básica de IDL es el módulo, que consta de una colección de definiciones de interfaz relacionadas (entre otras cosas). El siguiente ejemplo muestra un módulo que consta de una única interfaz con un solo método sin argumento que devuelve una cadena:

```
/*Nombre del archivo: cool.idl
  Propósito: La implementación  tendrá cool() la cual retornará la cadena " Java is cool"
*/
module cool { interface Cool { wstring isCool(); }; };
```

El tipo wstring (wide string) es un tipo de datos de IDL que corresponde al tipo de datos String de Java.

El archivo de la interfaz como tal es:

*Package cool;*
*Public interface Cool*
*Extends org.omg.CORBA.Object {*
*Public java.lang.String isCool (); }*
La implementación de la interfaz en Java contendrá simplemente la instrucción :
return “Java is cool”.

Concluyendo CORBA, es una propuesta que posibilita y trata de estandarizar la integración de códigos escritos en diferentes lenguajes por medio de un lenguaje que sirve de puente: IDL. Ello implica manejar dicho lenguaje y conocer como se debe establecer la conexión con un lenguaje particular (como se observa en el ejemplo para el caso de JAVA). Una presentación sencilla de lo que es CORBA puede ser vista en [Marvie 00, Basu 01].

### 2.1.3 Observaciones y otros modelos

Los modelos de componentes presentados son los más relevantes hasta donde se pudo establecer, en especial, los modelos COM, JavaBeans y CORBA. Como se explicó en la sección 2.1.1 entre todos ellos se pueden encontrar similitudes y grandes diferencias, de hecho el término componente no está estandarizado. El concepto de componente puede confundirse y en algunos contextos ser considerado como sinónimo de objeto. Un buen documento para establecer las diferencias entre un componente y un objeto es [Szyperski 99].

No es el objetivo de este trabajo hacer un análisis del modelo objetual, tampoco lo es el establecer las diferencias entre el modelo objetual y los componentes. Lo que interesa es recoger los elementos fundamentales de los componentes y sacar provecho de ellos en el mundo de los recursos educativos.
Otros modelos de componentes con características similares a los analizados son:
JavaPods propone un modelo de componentes inspirado en los EJB, ODP (Open Distributed Processing) que tiene aspectos similares a CORBA y propone igualmente una serie de definiciones, protocolos e interfaces que deben poseer los sistemas heterogéneos que se desee que cooperen entre sí. Estas deberán ser lo suficientemente abstractas para permitir que las implementaciones varíen.

LIL (Library Interconnection Language) es un lenguaje propuesto por Goguen [Goguen 86] para interconectar componentes de software. **Como futuros desarrollos el autor plantea que las posibilidades existentes para catalogar y efectuar búsquedas son ricas siempre y cuando dichos componentes sean documentados con información relevante y de acuerdo a estándares establecidos para ello**. Otros lenguajes similares son PI [Cramer 91] y Piccola [Nierstrasz 98].
Otro lenguaje examinado fue CoML (Componente Markup Language), el cual es un lenguaje para componentes basado en XML (ver sección 2.3.3), es decir define una serie de rótulos (*coml*, *meta*, *components*, *component*, *property*, *method-call*, *on-event*, *add* son algunos) que permiten expresar mediante un script una composición dada.

Finalmente se mencionan los COTS (Commercial off the Shelf), los cuales son componentes de software comerciales reutilizables los cuales se ensamblan para crear con ellos aplicaciones de software. Un buen artículo al respecto es [Iribarne 00] .

La introducción de información de documentación en los componentes analizados es bastante incipiente, de hecho la propuesta JavaBeans fue la más interesante en este aspecto al hacer uso de la tecnología Jini para recuperar instancias de componentes existentes.
De hecho este tipo de características se explotará en el modelo que se propondrá para los componentes educativos. Igualmente el modelo concebido deberá poseer un cierto conjunto de características *base* de los componentes de tal forma que se le pueda aplicar el término de componentes (educativos). Ver sección 4.4.2.

## 2.2 Sistemas relacionados con recursos educativos

### 2.2.1 Introducción

El término *recurso educativo* puede ser definido como [El Saddik 01]:
**Definición 4:** "*Recurso educativo: Cualquier entidad digital o no digital, la cual puede ser usada, reutilizada o referenciada durante un proceso de aprendizaje*".[1]

Sin embargo en la literatura referente a sistemas y herramientas educativas es frecuente encontrar términos como: objetos educacionales, componentes educativos, educational bricks (literalmente

[1] En nuestro proyecto la atención se centrará en los recursos educativos en formato *digital*.

"ladrillos educacionales"), entidades educativas etc. Todos ellos se refieren en general a la definición 2. El uso del término componentes educativos usado en la literatura difiere del utilizado en el contexto del desarrollo de este proyecto de maestría (ver sección 3.2) en el cual un componente educativo es *más* que un simple recurso educativo ya que involucra aspectos del mundo de los componentes reutilizables (ver sección 2.1) y de las bases de datos semiestructuradas (ver sección 2.3).

### 2.2.2 Panorama General de los Sistemas Educativos

Con el surgimiento y popularización de Internet, el valor de la información ha cambiado radicalmente. En Internet a través de buscadores especializados es posible encontrar información prácticamente sobre cualquier tema. Sin embargo dicha información no está clasificada u organizada adecuadamente para todos los tipos de usuarios (aunque todos estos usuarios tienen el propósito de aprender; sus objetivos, grado de conocimiento inicial, preferencias etc., son bastante variados). Los *usuarios necesitan una guía* para que la información que yace allí sea eficaz y para que finalmente se logre el objetivo deseado: el aprendizaje.
Lo que diferencia a un sistema adaptativo de un sistema tradicional de enseñanza por computadora es la introducción de técnicas adaptativas (véase sección 2.2.2.2) para facilitar el aprendizaje de cada individuo haciendo uso de contenidos dinámicos.
Es común el problema de la "pérdida en el hiperespacio" el cual consiste en [Web Page 1]:

**Definición 5:** "Problema de la pérdida en el hiperespacio": Experiencia de desorientación cuando se lee hipertexto (ver definición 6). Tendencia a perder el sentido de la localización y dirección en un documento no lineal. Este efecto es más pronunciado en sistemas carentes de suficientes ayudas de navegación y de términos claves del contexto.

Varias de las propuestas que se describen en la siguiente sección tratan de evitar tal problema.

Uno de los elementos comunes en muchas de estas propuestas es el uso de la hypermedia (*hipermedios*) como ayuda para la adaptación y guía de los usuarios, de acuerdo a [Henze 00] la hypermedia se define como (primero se definirá el término *hipertexto*)

**Definición 6:** Hipertexto: "Conjunto de nodos de texto conectados a través de enlaces.
Cada nodo contiene alguna cantidad de información y un número de enlaces hacia otros nodos."

**Definición 7:** Hypermedia: "Extensión del hipertexto la cual hace uso de múltiples formas multimediales como texto, vídeo, audio, imágenes etc."
Estos sistemas usan entonces un modelo del usuario inicial el cual recopila aspectos como su conocimiento, sus objetivos, experiencia, preferencias etc. y a partir de estos elementos tratan de adaptar la estructura navegacional de la manera posible más apropiada para cada usuario.
A medida que el usuario experimenta con el sistema su perfil debe evolucionar, para que el sistema tome las decisiones que se consideran apropiadas, es decir que sea un perfil dinámico.

**Nota:** No es el propósito de esta tesis profundizar en los elementos adaptativos concernientes a un sistema educativo (multimedial o no). Sin embargo se mencionan porque sirven de marco para la gran mayoría de los sistemas analizados.

#### 2.2.2.1 El modelo del usuario

Como se ha mencionado se tienen en cuenta características del usuario como su conocimiento, objetivos, preferencias, experiencia, velocidad de aprendizaje etc. De acuerdo a lo anterior dos tipos de modelos de usuario son comunes:

Modelo "overlay": Teniendo en cuenta los factores anteriores se obtiene el conocimiento inicial del usuario (mediante entrevistas, pruebas etc.) y se compara con el conocimiento de un experto.

Modelo por estereotipos: Los usuarios se clasifican en grupos (igualmente mediante entrevistas, libre elección etc.). Se asume que todos los usuarios que están en una misma clase tienen las mismas características. A medida que el sistema "conoce" más al usuario lo va reclasificando.

#### 2.2.2.2 Las técnicas de adaptación

Se trata de adaptar el contenido de acuerdo a las necesidades de los usuarios. Para ello se emplean técnicas como: insertar explicaciones adicionales, explicación breve de temas que aun no se conocen (temas prerrequisito) , insertar analogías con temas conocidos por el usuario, insertar más de una explicación y en diferentes formatos, orden de presentación de acuerdo a la relevancia para el usuario. (por ejemplo cumplimiento de ciertos prerrequisitos)

Lo anterior se logra mediante estrategias como: texto condicional (el usuario lo puede observar sólo cuando tenga ciertos conocimientos), "alargamiento del texto" (algunas palabras se expanden con su definición), varias páginas almacenadas en el sistema adaptativo con la misma información pero con diferentes niveles de profundidad etc. Para lograr estas estrategias se espera una población de recursos educativos lo suficientemente rica en el sistema adaptativo; de donde se desprende que el éxito potencial de un sistema de este tipo depende en gran medida del esfuerzo inicial y *compromiso de una comunidad educativa.*

También se trata de adaptar la presentación por medio de los enlaces:
Presentando en orden los más adecuados para el usuario de acuerdo a su relevancia, escondiendo enlaces irrelevantes (por ejemplo porque llevan a temas que el usuario ya conoce), sugiriendo mediante un botón *next best* (literalmente próximo mejor o también conocida como *direct guidance*, es decir guía dirigida), presentando pequeñas indicaciones sobre el contenido al cual apunta cada enlace.
Este último puede ser logrado mediante un pequeño texto resumen o mediante tipos de iconos o colores, por ejemplo es frecuente el uso de un **sistema semáforo** en el cual los enlaces son presentados con colores así: (Existen diversas variantes en el colorido).

- Rojo: El usuario aún no cumple los prerrequisitos. (enlaces no recomendado)
- Amarillo: Enlace no recomendado para el usuario (pero menos prohibitivo que el rojo)
- Verde: Enlace recomendado para continuar el aprendizaje
- Gris: Enlaces que contienen información que el usuario ya aprendió.

#### 2.2.2.3 Propuestas para los metadatos educativos y algunas experiencias en este campo

Varias propuestas establecen cuales son los elementos básicos que deben ser definidos para un recurso educativo. Las 3 principales se describen a continuación.

##### 2.2.2.3.1 Dublin

Dublin Core (DC) es un intento para estandarizar la creación de los metadatos que surgió en 1995 [BECTA 01]. El proyecto se concentró en extraer las características comunes a la mayoría de los documentos digitales (no necesariamente educativos) que se emplean en diversas áreas. Esto llevó a

la creación del Dublin Core Element Set (DCES). La organización que controla este proyecto es la Dublin Core Metadata Initiative (DCMI). El DC está compuesto así:

- *Contenido*: Título, Tema, Descripción, Fuente (referencia a otro recurso desde el cual este se deriva), Lenguaje, Relación (con otro recurso digital) y Cubrimiento
- *Propiedad Intelectual*: Autor, Editor, Colaborador, Derechos.
- *Instanciación*: Fecha, Tipo (novela, poema etc.), Formato, Identificador (URL, ISBN etc.)

En 1999 el DCMI comenzó a ampliar el DC con el objetivo de describir exclusivamente recursos educativos, mediante los siguientes grupos [Web Page 14]:

- *Audiencia*: Intermediario (profesor, administrador etc.), Beneficiario el cual incluye a su vez Individuo (estudiante, profesor), nivel de educación, rango de edades etc.
- *Duración*: Tiempo estipulado para asimilar el recurso educativo
- *Pedagogía*: Actividades y (descripción de las actividades a través de las cuales el proceso de aprendizaje ocurre) Métodos (de enseñanza utilizados por los profesores)
- *Estándar*: Identificador y versión del recurso
- *Calidad*: Descripción de la calidad del recurso educativo (determinada por el autor, por ejemplo si un documento es una versión definitiva o si es un borrador).

El DCMI se encuentra activo (su más reciente reunión fue en octubre de 2002) y de acuerdo a la información encontrada en su página oficial [Web Page 14] la intención es unir fuerzas con la propuesta de los metadatos presentada por la IEEE (LOM, véase sección 2.2.2.3.3) durante el año 2003. Esto quedó expresado en [Hodgins 00]:

"El DCMI y el IEEE LTSC expresan su compromiso para el desarrollo de metadatos interoperables para la educación. En particular, ambas organizaciones ven beneficioso el mantener compatibilidad entre LOM y el DC educativo".

#### 2.2.2.3.2 ARIADNE

El proyecto ARIADNE [Web Page 15] ha sido desarrollado en los últimos 7 años (desde 1995) por 20 universidades europeas y 5 corporaciones internacionales. En abril de 1998 las especificaciones de los metadatos desarrolladas en ARIADNE fueron enviadas al comité de la IEEE que se encarga de los metadatos LOM (ver siguiente sección). El estado actual de los metadatos de ARIADNE es una especificación **prácticamente equivalente a la de LOM.**

#### 2.2.2.3.3 IEEE LOM

Como se ha visto en las dos secciones anteriores el LOM (Learning Object Metadata) [Web Page 16] de la IEEE Learning Technology Standards Committee (IEEE LTSC) se está imponiendo como un estándar, para los metadatos educativos.

El LOM de la IEEE se centra en un conjunto mínimo de propiedades que permiten que los recursos educativos sean documentados, ubicados y evaluados.

LOM está compuesto así: (Ver apéndice 1 para la descripción total de LOM.)

**1. General:** Proporciona información como título, una breve descripción y palabras claves.
**2. Ciclo de vida:** Describe el estado actual y de desarrollo del recurso educativo.
**3.Metametadatos:** Describe los metadatos en sí mismos, por ejemplo, quien creó los metadatos.
**4.Técnica:** Formato del recurso, tamaño, requerimientos de software etc.
**5. Pedagógica:** Características pedagógicas del recurso como: grado de interactividad, nivel de dificultad, tiempo típico de aprendizaje etc.
**6. Derechos de Propiedad:** Condiciones de uso del recurso educativo. (costo,copyright etc.)
**7. Relación:** Describe relaciones con otros recursos.
**8.Notas:** Comentarios sobre el recurso.
**9:Clasificación:** Permite ubicar a un recurso en un sistema de clasificación que se establezca por parte de una comunidad.

En este proyecto se acogerá el LOM ya que se ha convertido prácticamente en un estándar. Sin embargo se realizarán ciertas modificaciones, ver sección 3.5 donde se presentan las explicaciones al respecto.

La importancia de los metadatos se menciona en [Suthers 01]: "*Los educadores necesitan ayuda para comprender la utilidad y el potencial disponible de los recursos educativos que pueden ser concebidos y para ello debe facilitarse la localización de los recursos apropiados, para suplir las necesidades de sus estudiantes y para ello los metadatos juegan un papel clave ... los metadatos deben describir los recursos en términos comprensibles para los educadores de tal forma que ellos puedan interoperarlos.*"
Similarmente se expresa en [BECTA 01]: *"Las herramientas para asignar los metadatos a los recursos educativos están surgiendo lentamente"*, esta tarea no puede ser ignorada para el desarrollo de recursos educativos en el presente y futuro próximo.

### 2.2.3 Principales sistemas educativos analizados

#### 2.2.3.1 ELM-ART: Episodic Learner Model - The Adaptive Remote Tutor

ELM-ART [Weber 97,Web Page 2] es una de las herramientas adaptativas pioneras, de hecho es precursora de Interbook, el cual se discute en la sección 2.2.3.4. ELM-ART es un tutorial dedicado exclusivamente a la enseñanza del lenguaje LISP.
Dos de las principales motivaciones para el desarrollo de ELM-ART fueron:
-Propiciar una guía personalizada para los estudiantes
-Evitar que estudiantes con un conocimiento previo estudiasen temas concebidos especialmente para principiantes, esto reduce el tiempo de aprendizaje.

El esquema de ELM-ART es bastante simple, inicialmente se disponía de un curso de LISP en texto y pasó a ser convertido en páginas HTML (Hypertext Markup Language), donde cada página contiene unos conceptos a ser aprendidos y están relacionados a su vez con otros conceptos desde el punto de vista de prerrequisitos, formando una *red de conceptos*. ELM-ART usa el típico método de la metáfora del semáforo ya explicada en la sección 2.2.2.2.
Una característica particular de este sistema es la posibilidad que tiene el estudiante para experimentar "en vivo" los ejemplos y soluciones a ejercicios (ejecutarlos), esto se logra incorporando una "ventana de evaluación" propia de ambientes de trabajo con LISP.

La traducción de texto hacia páginas HTML se realizó usando la misma técnica descrita en Interbook (ver sección 2.2.2.2). Aunque esta técnica según los mismos autores tiene sus limitaciones porque no siempre es fácil convertir un texto "normal" a hipertexto debido a que el

texto “normal” no fue concebido para ser leído de manera no secuencial, es decir, en muchos casos el capítulo n de un libro requiere haber leído los n-1 capítulos anteriores.

El criterio para determinar si una persona conoce o no sobre un tema es bastante simple: se asume que un estudiante ha adquirido los conceptos desarrollados en las páginas que va leyendo (dicha medida es bastante débil a decir verdad, sin embargo en AHA, ver sección 2.2.3.3 se mejora un poco este criterio ya que se exige que el estudiante apruebe algunos cuestionarios sobre el tema para establecer que lo aprendió).

Algunas mejoras en este aspecto motivaron igualmente el desarrollo de ELM-ART II, en el cual se hace lo mismo que en AHA, al final de una página se plantean unos ejercicios (llamados testgroups) y si son desarrollados correctamente por el estudiante, se considera que ha aprendido los conceptos relacionados con la página.
Los testgroups en ELM-ART II son de cuatro tipos: preguntas de si/no, selección única, selección múltiple y respuesta libre. El sistema ofrece estadísticas respecto al desempeño en un testgroup para cada estudiante y explica las soluciones a cada uno de los ejercicios.
Igualmente un proceso de inferencia más sofisticado fue establecido teniendo en cuenta los prerrequisitos de los prerrequisitos y así recursivamente para el perfil de cada estudiante (por ejemplo si un estudiante conoce un concepto X, todos los prerrequisitos de X también se marcan como conocidos) y la metáfora del semáforo fue extendida un poco agregando iconos visuales y el color naranja el cual significa una página ya visitada, pero no todas sus páginas subordinadas.
Para evitar la sensación de pérdida en el hiperespacio (ver sección 2.2.2, definición 5) ELM-ART II usa igualmente la técnica de guía dirigida, la cual provee al usuario de un botón next best, el cual le guiará hasta que se considere que el usuario ha logrado el aprendizaje.
La selección de la próxima página más adecuada para cada usuario utiliza un sencillo algoritmo que tiene en cuenta los prerrequisitos, los conceptos que se supone el estudiante sabe y los ejercicios que ha resuelto exitosamente hasta el momento.

Una característica particular de ELM-ART II es la ayuda que proporciona a los estudiantes para resolver los ejercicios, esta es *gradual*, es decir el sistema comienza dando pistas pequeñas para llegar a la solución, así el estudiante puede completar la solución si estaba “bloqueado” en algún punto. (Por supuesto el estudiante no deberá abusar de esta ayuda gradual, porque eventualmente el sistema terminará presentándole la solución del problema)
Para establecer el conocimiento inicial de un usuario sobre LISP se realiza una entrevista.

Finalmente los autores realizaron algunas pruebas de la efectividad de las técnicas adaptativas presentes en ELM-ART (Next button) concluyendo que al menos para los principiantes el botón Next es de mucha ayuda, (al menos los resultados mostraron que los estudiantes que hacían uso del botón Next aprobaban el curso más rápido que aquellos a los que no se les permitía usarlo o no lo usaban).

#### 2.2.3.1.1 Experimentando con el demo

En [Web Page 2] se puede tener acceso al curso de LISP de ELM-ART II. Luego de ingresar el sistema hace una pregunta respecto a la experiencia con el uso de Navegadores, lenguajes de programación y computadoras en general.
Se llega entonces al curso de LISP (el cual está dividido en 6 secciones). En una página introductoria al curso se sugiere un botón para continuar con la primera sección (aunque en la parte superior existen enlaces de acceso a las otras secciones). Cada lección tiene una breve introducción y una serie de subsecciones. La sección 1 se divide a su vez así:

LISP Course
Lesson 1
 Datatypes
  Atoms ...
  Symbolic Atoms ...
  Numbers ...
  Lists ...
  Nested Lists ...
  Empty List, NIL, and T ...
  Tests on Data Types ...
 Functions
  ...
 Self-defined Functions
  ...
Lesson 2
Lesson 3
Lesson 4
Lesson 5
Lesson 6

Figura 2.5: Sección 1 del curso de LISP de ELM-ART II

El sistema va guiando al usuario a través del anidamiento de las subsecciones de una manera similar a la concepción del sistema de carpetas tipo explorador de Windows, como se observa en la figura 2.5.

Luego de terminar la sección sobre Atoms, se plantean una serie de ejercicios y hasta que no se resuelven exitosamente un buen número de ellos no se logra avanzar hasta la siguiente sección sobre Symbolic Atoms

Una serie de botones ubicados en la parte superior proporcionan acceso a estadísticas, ayudas adicionales, poner ciertas preferencias etc.

En conclusión es un demo sencillo, agradable y que ofrece aspectos precursores interesantes para otros sistemas educativos.

De acuerdo [Seeberg 97] ELM-ART II es actualmente uno de los más avanzados ambientes de aprendizaje de programación.

#### 2.2.3.2 Multibook

El proyecto Multibook [Seeberg 97] es un robusto sistema hypermedial adaptativo de aprendizaje concebido en la Universidad de Darmstadt, Alemania.

Los autores explican que: "...*Es ambicioso pretender que una máquina pueda hacer un trabajo tan inteligente como un profesor humano...pero sí se pueden implementar algoritmos que permitan extender las posibilidades de un libro tradicional...*"

Multibook es una fusión de hypermedia y tutoriales inteligentes. Ambos por sí solos tienen desventajas, por ejemplo los sistemas tutoriales inteligentes son bastante rígidos (tienden a anticipar o predefinir todos los caminos). La hypermedia explotan los diversos formatos multimediales, permitiendo a los usuarios elegir diversos formatos, pero con el inconveniente de que se carece de una guía bien estructurada.

Los recursos educativos en Multibook se denominan Media Bricks (literalmente ladrillos multimediales) y con ellos se construyen lecciones.

En Multibook los usuarios pueden especificar un *rol* (estudiante, programador, manager), las lecciones vienen en 6 diferentes niveles de dificultad, se tienen dos métodos de enseñanza (orientación jerárquica y a la solución de problemas), los usuarios pueden elegir sus formatos preferidos de presentación.

Estos elementos hacen parte del modelo o perfil del usuario (el cual es ingresado por el usuario), por supuesto existen restricciones porque no siempre se dispondrá de todas las variedades posibles

de bricks; cuando ello sucede Multibook trata de realizar combinaciones entre los bricks existentes para tratar de satisfacer al modelo del usuario con una lección apropiada, Por ejemplo suponga que el usuario posee un nivel de dificultad 3 (medio) y no existen bricks correspondiente a tal nivel, pero sí de nivel 4 (más avanzado) y 2; el sistema puede tomar la decisión de presentarle una lección con bricks de nivel 4 y 2.

Multibook utiliza un modelo para representar los conceptos llamado ConceptSpace, en el cual se definen una serie de relaciones entre conceptos como son: Superconcepto (un concepto incluye a otro), Continuidad (luego de estudiar un concepto A se debe estudiar el concepto B), Problema_y_Solución (Un concepto provee ideas para la solución de problemas que surgen a raíz de otro concepto) y muchas otras etc.

También utiliza un modelo para los Media Bricks llamado el MediaBrickSpace en cual se establecen también relaciones entre los Media Bricks como por ejemplo: Ilustración( Brick A presenta algunas cifras, Brick B grafica las cifras presentadas en A) Contradicción (Brick A explica una teoría, Brick B presenta casos donde la teoría falla) etc.
De los sistemas analizados, Multibook es sin duda el más rico en la representación de un modelo del dominio, aun así sus autores concuerdan en que se pueden extender mucho más los tipos de relaciones entre conceptos y bricks.

Los bricks tienen asociada información adicional que el diseñador define, como son el concepto fundamental que desarrolla, indicaciones sobre el nivel de dificultad, datos técnicos etc. Multibook posee metadatos básicos, los cuales son la base para lograr una selección “inteligente” y conectar los bricks con el ConceptSpace. Estos metadatos son denominados *atributos* de los bricks. En nuevas versiones de Multibook [El Saddik 00] los media bricks han sido documentados con LOM (ver sección 2.2.2.3.3) para ello usan una base de datos relacional y construyeron un editor llamado xLOM (extended LOM editor, ver también 3.5.2).

Multibook provee guías para el usuario en forma de **tablas de contenido**, estas **son justificadas**, según sus autores porque:

- Los usuarios en un sistema de aprendizaje deben aprender ciertos temas lo deseen o no.

- Los usuarios pueden no poseer el suficiente bagaje del dominio que intentan aprender como para seleccionar los temas relevantes.
- Los usuarios pueden no estar capacitados para construir una lección coherente de acuerdo a un método de enseñanza apropiado.
- La base de conocimientos es demasiado extensa como para ser manejada por sí solo. Ver problema de la pérdida en el hiperespacio sección 2.2.2, definición 5.
  Una tabla de contenido ofrece una orientación al usuario, le dice donde está y hacia donde puede ir (manteniendo cierta coherencia).

En este proyecto de maestría se adoptan algunas de estas observaciones.
Después de que estas guías son establecidas, son pobladas con los bricks más adecuados para cada usuario, teniendo en cuenta las relaciones entre los bricks y los conceptos explicadas antes.

En este momento el perfil de usuario de Multibook es poco dinámico (el sistema sólo hace un seguimiento básico del usuario, que conceptos ha observado, desempeño en los tests etc.) se pretende extenderlo de tal forma que tenga mayor evolución,
Igualmente se espera proveer al sistema de más métodos de enseñanza y ayudar a los usuarios a comprender dichos métodos para que seleccionen el más adecuado para cada caso.
Se plantea también la exportación e importación de los metadatos en formato XML [El Saddik 00].
Un demo en alemán bastante restrictivo --varias opciones no son funcionales-- de Multibook puede ser bajado de [Web Page 3]. Se intentó obtener una versión "profesional" pero no hubo respuesta por parte del equipo de trabajo de Multibook. De todas formas a partir de la documentación existente se infieren bastante bien los fundamentos de dicho sistema.

#### 2.2.3.3 AHA: Adaptive Hypermedia Architecture

El núcleo del proyecto AHA [De Bra 98] consiste en una base de datos que guarda un modelo del usuario basado en el conocimiento de conceptos.
AHA soporta contenidos multimediales y usa varias de las características mencionadas en la sección 2.2.1, esquema de colores para los enlaces (sistema semáforo, en AHA el usuario puede cambiar dicho esquema) y para el borde de las imágenes. Alternativamente maneja "ocultamiento" (enlaces condicionales), eliminación y comentarios sobre enlaces pero no soporta ningún tipo de ordenamiento de estos.

AHA no fue pensado solamente para la educación, también sirve en áreas como sistemas de información en línea, ayudas en línea, etc.

En AHA cada página está asociada a un concepto y se asume que un usuario tiene conocimiento sobre un concepto si ha leído una página que trata sobre este o si ha tomado un test (los tests son de selección múltiple) al respecto y lo ha aprobado. (Para indicar si un usuario conoce o no sobre un concepto se usan valores booleanos, no se manejan estados intermedios). Dentro del modelo de cada usuario se guarda un histórico que indica cuales páginas ha visitado el usuario y para cada sesión se guarda adicionalmente el tiempo de entrada y salida de cada página. (aunque no se menciona en la documentación de AHA, es de suponer que si un usuario entra y sale "muy rápido" de una página no debe haberse apropiado del concepto al cual está asociado dicha página).

Para lograr la adaptación se usa el siguiente estilo de código mediante un preprocesador que filtra el contenido de fragmentos a ser presentados, por medio de una estructura condicional inmersa en HTML: (ocultamiento de enlaces, eliminación de enlaces)

```
<!-- if definition and history -->
        Esta parte se muestra si los dos conceptos,
        definición e historia, son conocidos por el
        Usuario de acuerdo a su modelo (perfil)
<!-- else -->
        Si no se cumple lo anterior entonces se
        Muestra esta parte.
<!-- endif -->
```

Esto facilita las labores de creación para el autor, ya que el uso de otras alternativas para lograr el mismo efecto podrían ser un poco complicadas (HTML dinámico, JavaScript , Vbscript etc.) Esta es **una aproximación que trata de minimizar los aspectos de programación a los autores de cursos o lecciones.**

Por medio de condicionales se pueden lograr también efectos que permiten insertar o no información adicional acerca de un elemento.

**El autor es el responsable de abrir y cerrar todos los tags en forma correspondiente.**

También por medio condicionales el autor puede lograr los métodos mencionados para la adaptación en la sección 2.2.2.2, incluso [De Bra 98]: "*se puede lograr direct guidance (guía dirigida) pero con mayor esfuerzo*" como por ejemplo:

```
<!-- if definition -->
   <a href="theorem.html">
 <!-- else -->
   <a href="definition.html">
 <!-- endif -->
  next
 </a>
```

Sin embargo el lograr guía dirigida no es tan sencillo porque el autor debe estar "pendiente" de elegir verdaderamente la "mejor" página o la más adecuada de acuerdo a cada usuario; **este es un aspecto al cual se espera ayudar en nuestro trabajo**, véase sección 4.4.3.

#### 2.2.3.3.1 La creación de un recurso educativo en AHA

El trabajo de *authoring* en AHA consiste en lo siguiente:
El autor llena ciertos datos como son el título del curso, el directorio en el Web Server, su email, el nombre del Web Server etc.
Cada página que el autor ingresa debe llevar los dos siguientes comentarios:
En el primer comentario se deben colocar los conceptos (prerrequisitos) que deben haberse logrado para llegar hasta esta página, ejemplo:

```
<!-- requires readme and history but not (intro or
  definition)
-->
```

En el segundo comentario se colocan los conceptos que se supone llegan a ser conocidos luego de leer la página en cuestión, ejemplo:

```
<!-- generates history -->
```

A partir de estas líneas el sistema crea un archivo de dependencia el cual contienen pares de páginas y conceptos prerrequisitos y una lista de todos los conceptos existentes.
Desde el punto de vista de los usuarios, el proceso es simple, estos se registran a través de un formulario inicial y adquieren un código identificador de usuario y un password.
A partir de allí las páginas que se muestran a cada usuario tienen un URL así:
*http://servername.domain/2L670/cgi/get.cgi/user-id/page.html*

#### 2.2.3.3.2 Adición de reglas a AHA

En [Wu 01] se plantean algunos elementos para mejorar la adaptación de AHA:
Dado que en AHA el conocimiento de un concepto por parte de un usuario se indica por medio de un valor booleano, se extiende esta calificación a un rango de *not known*, *known* y *well known* y se especifica que not known < known < well known.
La intención principal es hacer que la actualización del perfil del usuario sea un proceso más "exacto" y automático, el cual se realiza de acuerdo a la interacción con el usuario. Para ello se define un modelo del dominio el cual incluye conceptos y relaciones entre ellos. Las relaciones entre conceptos son prerequisite (un concepto es prerrequisito de otro) link (un concepto conduce hacia otro) e inhibit (si c1 inhibe a c2, significa que si se conoce c1 no se justifica visitar a c2). También se expresa que "*En teoría se pueden definir muchos tipos de relaciones*".
Se plantea entonces un lenguaje, similar al SQL, mediante el cual los autores pueden definir reglas (las cuales se comportan como triggers) que establecen la conexión entre el modelo de cada usuario y el modelo del dominio, es decir, *operadores dirigidos a que los autores creen sus propias reglas para propiciar el proceso de actualización del modelo de cada usuario.*

Existen 2 tipos de reglas:

- Generales: Que se aplican entre todos los conceptos
- Específicas: Que se aplican a conceptos particulares.Tienen precedencia sobre las generales.

Veamos un par de ejemplos

Ejemplo 1:

C: ***Select*** *P.access*
***Where*** *P.access =* ***true and*** *P.ready_to_read =* ***true***
A: ***Update*** *P.knowledge= "well known"*

Cada regla consta de dos partes, primero se tiene una condición C que es la que dispara la ejecución de la acción A (segunda parte).
En el ejemplo se tiene que cuando el concepto P se considere listo para ser leído (ready_to_read en true) y es accesado (propiedad access del concepto en true) entonces se actualizará la propiedad knowledge del concepto como "well known".

Ejemplo 2:

*C:* ***Select*** *C1.knowledge* ***where*** *C1.knowledge >= "known"*
*A:* ***Update*** *C2.ready_to_read :=* ***true***
***where*** *prerequisite (C1,C2)* ***and***
***not exists*** *(****select*** *c3* ***where*** *prerequisite (C3,C2)* ***and***
*C3.knowledge < "known")*

Explicación: Cuando un concepto C1 llega a ser conocido (a su propiedad knowledge se le asigna known o well known) se toma cada concepto C2 (del cual C1 es un prerrequisito) y se le coloca su propiedad ready_to_read en true, pero siempre y cuando el concepto C2 no posea algún concepto C3 prerrequisito cuyo valor sea not known. Es decir un concepto C2 llega a ser considerado ready_to_read cuando todos sus prerrequisitos son al menos conocidos (known).

Nótese la ausencia de la cláusula **from** en este tipo de lenguaje similar a SQL.
Finalmente se expresa que hacia el futuro se espera contar con herramientas **mucho más amigables de tal forma que se libere un poco a los autores de la necesidad de tener que aprender este lenguaje.**

Una anotación final traída de [De Bra 98] "***AHA no minimiza la necesidad de autores expertos***". Es decir los autores **deben aprender** ciertas herramientas que les permitan explotar tal sistema. (Por ejemplo el lenguaje de preprocesamiento para lograr los diferentes métodos de adaptación mencionados y el de creación de reglas de adaptación incorporado recientemente).

En [Web Page 4] es posible probar un curso sobre hypermedia diseñado con AHA.
Desde el punto de vista del creador la entrada de la información se hace a través de formularios y haciendo uso del lenguaje de preprocesamiento descrito.

#### 2.2.3.4 Interbook

Interbook [Schwarz 97] es una herramienta orientada a la web para crear libros electrónicos adaptables. Como la gran mayoría de estos sistemas, se tienen en cuenta las diferentes características de los usuarios para lograr la adaptación (conocimiento, objetivos, experiencia con la computadora, velocidad de aprendizaje etc.)

De los métodos de adaptación mencionados antes, Interbook soporta el ordenamiento de enlaces de acuerdo al perfil de usuario y de acuerdo a algún criterio como puede ser la relación de conceptos con los objetivos del usuario.
El conocimiento acerca de un dominio está representado por un *modelo del dominio* basado en conceptos, el modelo más sencillo puede ser una lista simple de conceptos (como en AHA) o una red de conceptos interconectada con relaciones especiales como en Interbook. (siendo más complejas en Multibook, véase sección 2.2.3.2).
Respecto al usuario se manejan 3 niveles de conocimiento sobre un concepto, estos son:
not-known, in-work y clear-nothing new. Se ha planteado otra categoría, well-learned (bien aprendido), esta debería ser evaluada mediante ejercicios pero la versión actual de Interbook no soporta la presencia de tests y por lo tanto no la considera actualmente.

Interbook se concentra en la clasificación de los enlaces como: ready to be learned, not ready to be learned y nothing new. (Usa un sistema semáforo sencillo con colores verde, rojo y blanco respectivamente). Para realizar la clasificación anterior se lleva, un perfil para cada usuario y se maneja un sistema de prerrequisitos y conceptos que desarrolla cada nodo.
Ofrece guía dirigida por medio de una característica que se denomina "teach me".

Los libros electrónicos generados poseen una tabla de contenido, un glosario y una interfaz de búsqueda. Cada entrada del glosario corresponde a un concepto del dominio.
Los libros están divididos en secciones y cada sección está asociada a uno o varios conceptos (el "espectro de conceptos") que desarrolla (*outcome concepts*) y a otros que tiene como prerrequisito (*background concepts*). En forma correspondiente cada concepto presente en el glosario esta asociado a las secciones donde puede ser aprendido.
Con los conceptos prerrequisitos se proporciona ayuda al estudiante así: cuando entra a una sección donde existen conceptos que no domina (según su modelo) se le sugieren enlaces que se dirigen hacia secciones que cubren tales conceptos.
Se utiliza un modelo de estereotipos para clasificar al usuario y a medida que este evoluciona se produce su reclasificación.

#### 2.2.3.4.1 La labor de creación de un Interbook

Los pasos para crear un libro electrónico en Interbook [Brusilovsky 97] son los siguientes:

Se requiere el “Interbook toolkit”, el cual consta de tres herramientas: el Interbook system, Microsoft Word y el RTFtoHTML converter. El producto (versión 1.47) está disponible para trabajar en ambiente Apple Macintosh (ver página oficial de Interbook [Web Page 5]).

Primero se debe preparar un documento en Word de una manera muy especial, siguiendo algunas indicaciones y un formato en el cual se hace uso de headings (secciones): sección 1, sección 2, título etc., de Microsoft Word. El resultado de este paso es un archivo de MS Word “propiamente estructurado”. El documento en Word puede contener URL’s y enlaces internos propios, en el resultado final, estos serán respetados por el programa transformador (RTFtoHTML).
El segundo paso consiste en la anotación basada en conceptos de cada sección, esto le permite a Interbook saber que conceptos (el espectro) existen “detrás” de cada sección.
El formato para colocar los conceptos que se desarrollan es:
*out: concepto1, concepto2 etc.*
y para los prerrequisitos (background concepts):
*pre: Concepto 1, concepto 2 etc.*

El resultado de este paso es un archivo de MS Word “propiamente anotado y estructurado”.
Estas anotaciones siguen un formato especial; allí se detallan los conceptos que desarrolla cada sección y sus prerrequisitos y se deben colocar al inicio de cada sección (entre el header y el primer párrafo).

Una vez las anotaciones han sido hechas el archivo es salvado en formato RTF, se usa un programa especial (un transformador de RTF a HTML) llamado RTFtoHTML el cual realiza algunos cambios internos para convertir a formato HTML. La extensión html es manualmente cambiada a .inter para que sea reconocida por el sistema Interbook y se coloca el archivo en el directorio *texts*.

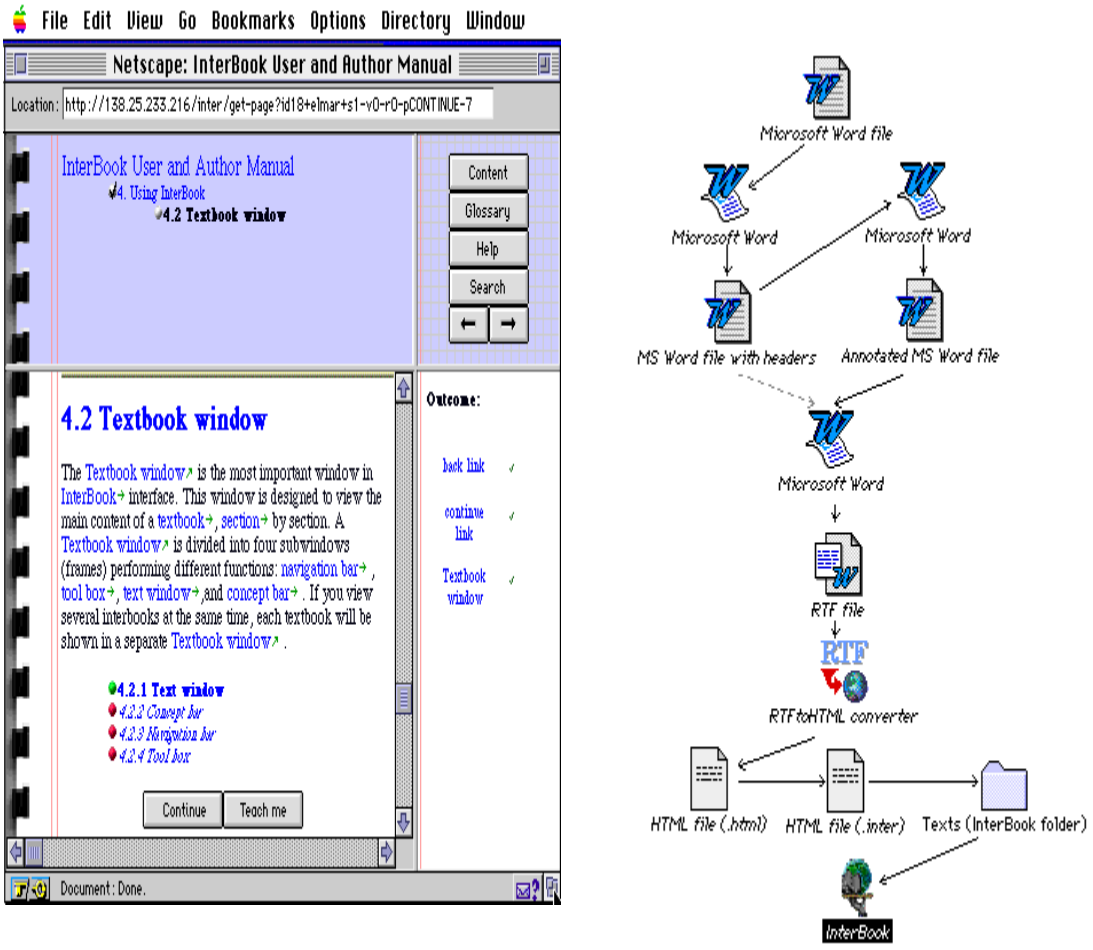



Figura 2.6: Creación y ejemplo de un Interbook

Finalmente cuando el servidor de Interbook se inicia, reconoce los archivos con extensión .inter (ubicados en el directorio texts) y genera el libro electrónico en el que aparecen el título, las secciones, el glosario construido etc. El proceso anterior puede verse en la figura 2.6.

#### 2.2.3.4.2 Experimentando con Interbook

Como en la mayoría de los sistemas cuando se llega al mundo de Interbook [Web Page 5], el usuario se matricula mediante un username y un password, y accede a varios Interbooks (*content window*) y al último Interbook (sección específica) que estaba leyendo (*text window).*
Similarmente a ELM-ART en cada text window se muestra un Interbook desglosado por secciones (con enlaces hacia ellas), también se muestran los conceptos asociados, acceso a los enlaces de ayuda para los background concepts, acceso al glosario etc.

El glosario se muestra en una ventana y es una sencilla interfaz que permite buscar conceptos alfabéticamente y para cada uno de ellos se proporcionan enlaces que los desarrollan y otros que lo tienen como prerrequisito, como ya se mencionó. Se trata de un glosario **general**, es decir, se reúnen allí todos los conceptos de todos los Interbooks existentes.

Existe igualmente un **operador de búsqueda similar al de Word**, el cual simplemente recibe una cadena de texto (similar a un buscador Google,Lycos etc) y busca todas las secciones de Interbooks que contienen dicha cadena. En este proyecto de maestría tal tipo de operador no es de interés porque el contenido como tal de un recurso educativo es caja negra, pero recuérdese que toda la documentación (los metadatos) asociada a un recurso educativo sí se puede consultar y se supone que allí debe encontrarse información valiosa sobre el recurso.

La content window funciona como un catálogo de los Interbooks disponibles. Si se da click en alguno de ellos se desplegará entonces su correspondiente text window.

Los libros electrónicos generados en Interbook sirven para crear cursos sobre diferentes tópicos a diferencia de ELM-ART (ver sección 2.2.3.1) el cual estaba enfocado específicamente a un curso de LISP. De hecho ha habido otros sistemas cuyo enfoque ha sido el de cursos muy específicos [Henze 00]: PT para aprender lenguaje C, HYDRIVE para enseñar hidráulica sobre aviones, AVANTI sistema de información de ayuda en el área metropolitana y otros.

### 2.2.3.5 KBS

El proyecto KBS Hyperbook System [Henze 98, Henze 00, Web Page 6] permite la creación de sistemas de hypermedia adaptativa e implementa el aprendizaje basado en proyectos.

En KBS los conceptos (llamados *knowledge items*: KI) están relacionados unos con otros por medio de un modelo conceptual de cada hyperbook y pueden ser elementales (Ej: “if” o “while” en el dominio de la programación) o compuestos (Ej: “Estructuras de flujo de control”).

Las unidades de un hyperbook y los proyectos son indexados con algunos conceptos del dominio del conocimiento (existe en cada unidad un main (principal) KI). Sin embargo las unidades no son secciones o capítulos de un libro de texto, son unidades conectadas a otras por medio de relaciones semánticas (por ejemplo una unidad que trata sobre “Java Objects” estará seguramente conectada a otras que hablen sobre “Objects” y “Object instantiation”). Lo anterior genera una navegación dinámica entre tales unidades, y da lugar a un sistema de semáforo el cual usa un algoritmo que genera secuencias de páginas que el usuario debería leer de acuerdo a sus objetivos y conocimientos. (las convenciones utilizadas son "already known" (ya conocida), "suggested" (recomendada), "too difficult" (demasiado difícil).
Los proyectos se conectan o relacionan entonces con las unidades correspondientes que desarrollan los conceptos que el proyecto requiere.

Un modelo pedagógico de conocimiento separado es construido y contiene todos los conceptos y algunas dependencias entre ellos. Así cada hyperbook no contiene en sí mismo prerrequisitos.
Un glosario es generado (similarmente como en Interbook) a partir del modelo del conocimiento, para cada término allí presente se establecen enlaces hacia ejemplos y unidades de los hyperbooks en forma correspondiente.

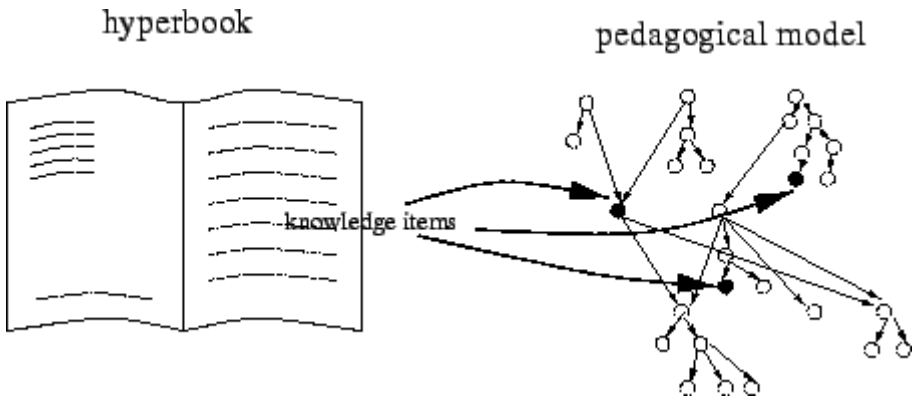



Figura 2.7: Relación de un hyperbook y el modelo de conocimiento

También se maneja guía dirigida por medio de un "botón" next learning step teniendo en cuenta el conocimiento actual y el conocimiento que se debe adquirir al terminar de leer el hyperbook. Es decir el mecanismo de adaptación considera:

-Recursos de adaptación (dar al estudiante información apropiada mientras desarrolla sus proyectos)
-Adaptar la estructura navegacional para darle al estudiante la información apropiada de cual material explorar o próximo a aprender.
-Ofrecer una guía al estudiante por medio de una secuencia a través del hyperbook (confrontar con Multibook) dependiendo de los objetivos del estudiante *(adaptive trail generation).*
-Ofrecer los proyectos más adecuados de acuerdo de acuerdo a los objetivos del estudiante y sus conocimientos previos (*adaptive project selection*).
Igualmente KBS soporta aprendizaje basado en objetivos; los usuarios pueden definir sus objetivos o solicitar el próximo objetivo de aprendizaje ( *adaptive goal selection).* Para cada uno de estos objetivos una secuencia de lectura es generada, la cual contiene el conocimiento necesario para lograr el objetivo y teniendo en cuenta los prerrequisitos del usuario.

Para el modelo del usuario, se maneja una calificación para cada KI, para el estudiante esta puede ser "expert's knowledge" (experto), "advanced knowledge" (avanzado), "beginner's knowledge" (principiante) o "newcomer's knowledge" (nuevo). Para obtener estas calificaciones se usa un modelo bayesiano.
Los usuarios son monitoreados a medida que van realizando proyectos existentes en el sistema. De acuerdo a su desempeño se actualiza su perfil. El estudiante es sometido a una encuesta al final de un proyecto en la cual se le pregunta cuan fácil le pareció el proyecto. A diferencia de otros sistemas (AHA y ELM-ART secciones 2.2.3.3 y 2.2.3.1) en KBS no se considera que se ha apropiado un conocimiento por el simple hecho de leer una unidad al respecto.

KBS a diferencia de los otros sistemas aquí analizados soporta "la velocidad de aprendizaje". Para ello los usuarios definen "cuanto" y que aprender. Si el sistema observa que un usuario está desarrollando un proyecto muy bien, entonces actualiza las estimaciones que tenía sobre el conocimiento del usuario y este puede seguir con un proyecto más avanzado. Si el sistema, por el contrario, observa que el usuario no es muy familiar con los tópicos de un proyecto, puede sugerir otros que lo familiaricen con dichos tópicos.
Tanto documentos "nativos" del sistema KBS como páginas en Internet pueden ser incorporadas a KBS; por supuesto que esto implica el riesgo de enlaces perdidos.

Finalmente se expresa que los sistemas más comparables a KBS son AHA, ELM-ART II e Interbook y que entre los aportes de KBS se destaca el empleo de redes bayesianas para modelar y propagar el dominio de todo el conocimiento en el sistema.

#### 2.2.3.6 Resumen

En los sistemas analizados la forma de creación de un recurso educativo (llámese curso, lección, proyecto, libro electrónico etc.) se puede resumir así:

- Interbook: Transformación de un Documento en Word a un Interbook mediante unas reglas que se deben seguir y uso de algunas herramientas.
- ELM-ART: Proceso similar a Interbook pero más primitivo y orientado específicamente a un lenguaje de programación.
- Multibook: Interfaz sencilla y amigable para que los autores creen sus recursos. Carece de características deseables como consultas expresivas, facilidades de actualización, etc.
- AHA: Lenguaje de marcado similar al lenguaje utilizado para crear páginas web, HTML.
- KBS: Interfaz sencilla para la creación de los hyperbooks.

Véase además la sección 4.2.1.1 donde se analizan más aspectos referentes a estos sistemas y se establece una comparación con el modelo aquí propuesto.

## 2.3 Estado del arte de las bases de datos semiestructuradas

### 2.3.1 Las Bases de Datos Semiestructuradas y la Web

Las fuentes de información accesibles electrónicamente están creciendo rápidamente. Muchas de estas fuentes almacenan y exportan datos sin una estructura claramente definida. Sin embargo en la mayoría de los casos los datos no están totalmente desprovistos de estructura, éstos reciben entonces el nombre de *datos semiestructurados*.

Las bases de datos semiestructuradas, a diferencia de las tradicionales, no están ajustadas a un esquema o éste no se conoce de antemano. La definición más común de dato semiestructurado es la de *un dato que se describe a sí mismo* (*self-describing*), esto permite modelar datos tanto con estructura homogénea como heterogénea.

Los datos semiestructurados consideran aspectos como:

- Estructura *irregular*: Dado que los elementos son heterogéneos, algunos pueden carecer de cierta información y otros poseer información extra. De igual modo, el mismo tipo de información puede estar almacenado de diferentes formas.

- Estructura *implícita*: Aunque en muchas ocasiones existe una estructura precisa, ésta está dada implícitamente. Una estructura puede ser vista como implícita cuando algún tipo de procesamiento adicional es requerido para obtenerla, este es el caso de la web en donde los documentos pueden venir en forma de texto plano y es necesario analizar el documento para hallar su estructura.

- Estructura *parcial*: Una parte de los datos puede carecer de estructura y otra puede contener una estructura muy vaga, por esta razón estructurar completamente los datos sigue siendo una meta difícil de lograr.

De otro lado está la web donde la unidad de información es el archivo, el cual se accede a través de una URL (Uniform Resource Locator).
La información tiene una estructura formada por las marcas que se crean con HTML se estructura texto para presentación visual.

Dada la convergencia de la web y las bases de datos se tiene la necesidad de comunicarlas de una manera eficiente: Se requiere un puente.

Supóngase dos organizaciones (compañías) 1 y 2.
La organización 2 requiere accesar la información que la otra publica en páginas HTML:

*Organización 1 → Base de Datos → Publicación en HTML ← Acceso ← Organización 2*

Si la organización 2 pretende ,por ejemplo, calcular un simple promedio de una columna extrayéndolo de la página HTML, puede ser una labor pesada. Este cálculo lo podría realizar directamente la base de datos. Además un cambio de formato o estructura de la página HTML

aunque fuese muy simple, implica cambiar el software que revisa dichas páginas. (Dicho software se acostumbra llamar *wrapper*)

XML (eXtensible Markup Language) soluciona el problema en el nivel lógico al proveer un lenguaje autodescriptivo de los datos.

En este punto es cuando entran en juego los datos semiestructurados. Difieren de las bases de datos tradicionales en las cuales primero se define el esquema y luego se "entran" los datos, y todos ellos tienen que cumplir con tal estructura (homogeneidad).
Los datos semiestructurados son bastante irregulares, por ejemplo un grupo de datos brindando información sobre hoteles en una página de Internet puede diferir de la forma como son descritos en otra página o incluso en la misma.
Puede ser que algunos hoteles posean por ejemplo email, 3 teléfonos etc., mientras que otros no posean email y sólo tengan un teléfono y muchas otras posibilidades. Sin embargo estos hoteles deben poseer un grupo de características comunes entre sí que sean suficientes para considerarlos *hoteles.* Por medio de XML, o de otro lenguaje intermediario, se puede describir toda esta información, y si se pretende ejecutar consultas contra la integración de la información de todos estos hoteles, se requiere una plataforma que soporte consultas y manipulación para tal situación.

*No significa que lo anterior no pueda ser representado en una base de datos tradicional (relacional por ejemplo)*, es posible, manejar por ejemplo una estructura hotel con muchos campos de tal forma que den cabida a todos los posibles atributos que se puedan presentar y realizando un manejo de nulos.
Este mismo aspecto es planteado en [Wu 01] donde se explica que aunque para los propósitos de su proyecto manejaron una tabla con muchos nulos --para manejar el modelo del usuario--, es deseable obtener una forma de almacenamiento más eficiente y económica.

Pero existen otras situaciones más complejas y que pueden ser más complicadas de representar, supóngase que dos hoteles, , tienen información sobre su teléfono, el hotel 1 lo presenta así:
*Teléfono: 2238998*

Y el hotel dos así:
*Teléfono: Indicativo: 095 Número: 2287788*

Esta *composición* de teléfono en el segundo hotel (y otras que se podrían presentar en otros hoteles) hace un poco más difícil la labor de representar estas instancias de una manera tradicional, aunque no imposible.

XML se presenta (ver sección 2.3.3) como un lenguaje muy apropiado para representar datos semiestructurados. Esta facilidad ha hecho que precisamente se desarrollen lenguajes de consulta para documentos XML, los cuales también se describen (ver sección 2.3.6) al igual que otra aproximación: el proyecto LORE el cual es una base de datos semiestructurados.

Es clave la fusión entre **XML** y los **datos semiestructurados,** además XML es un lenguaje que se está imponiendo como un mecanismo estándar para el intercambio de datos en la **web.**
Sobre el intercambio, por ejemplo vía XML se puede migrar "fácilmente" una base de datos de SQL Server a Oracle, se extraen los datos en XML (ya existen sentencias nativas en muchos productos de bases de datos que producen la salida en este formato) y se leen igualmente en el otro producto.

### 2.3.2 Los Datos Semiestructurados

Como se ha dicho el significado de semiestructurado es "sin esquema" y "autodescriptivo", véase la sección anterior. La forma de representarlos es por medio de parejas: Rótulo : Valor.
Una sintaxis para un registro podría ser:
*{ rótulo: valor , rótulo: valor, ... }*
Donde valor a su vez puede ser un registro (anidamiento).
Veamos un ejemplo de un registro en tal notación:
*{ nombre: "Pedro", email: "peter@mailbox.com" }*
y con anidamiento (los cuales se pueden realizar hasta cualquier nivel):
*{ nombre: { pila: "Juan", apellido: "Otero" } , edad: "22" }*

Es posible repetir rótulos:
*{ nombre: "Franky" , tel: 2139870 , tel: 3458976 }*

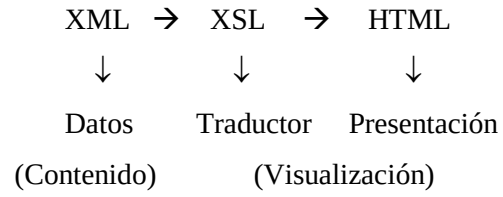
Hay otras formas sintácticas para representar datos como los anteriores, una de ellas es AceDB, concebido especialmente para el campo de la genética, pero muy genérico, y el más prometedor que

se describe a continuación: XML. En [Web Page 7] puede verse información de un congreso sobre datos semiestructurados, celebrado en el 2001.

### 2.3.3 El Papel de XML

XML(eXtensible Markup Language) [Web Page 10] a diferencia de HTML permite describir contenido *no* presentación.

Se puede transformar XML hacia HTML por medio de XSL (eXtensible Stylesheet Language). Se puede representar el proceso mediante el cual se obtiene contenido + presentación así:




Leer datos desde una página HTML es complicado, se puede crear un programa *wrapper* (ver sección 2.3.1) que lo haga, pero ante cualquier cambio de la página fracasará.

XML permite definir sus propios rótulos, permite anidamientos y adicionalmente expresar una gramática (pero no es obligatoria) o DTD (Document Type Definition) que debe ser satisfecha. Estos elementos se describen a continuación.

Para comenzar veamos la siguiente tabla relacional:

**Persona**

| *Nombre* | *Telefono* |
|---|---|
| Juan | 2158756 |
| Pedro | 5651209 |

En XML esto se puede escribir así:

```
<bd>
<persona>
        <nombre> Juan </nombre>
        <telefono> 2158756 </telefono>
</persona>
<persona>
        <nombre> Pedro </nombre>
        <telefono> 5651209 </telefono>
</persona>
</bd>
```

Hay mucha similitud con HTML, la diferencia radica en que los rótulos pueden ser creados por el usuario y que no existen rótulos para dar presentación, pero se conserva por ejemplo la apertura y cierre de rótulos en forma correspondiente.
Una de las primeras abreviaturas que se puede usar es: *<email> </email>*

Lo cual indica que no hay email (algo muy similar al concepto de nulo) lo cual se puede abreviar así: *<email/>*

Otra característica es que dentro de los rótulos de apertura pueden colocarse propiedades o atributos, pero hay que tener cuidado no se trata aquí de los mismos conceptos traídos del mundo de los objetos y del modelo relacional. Veamos un ejemplo:
*<telefono area="Boston"> 5651209 </telefono>*

La intención es usar estas propiedades para extender o dar mayor información acerca de un elemento, sin embargo *la distinción entre qué debe ser colocado como rótulo y qué como propiedad no es clara*, por ejemplo es posible escribir algo como:
*<persona nombre="Juan" telefono="2158756"/>*

Como se observa en las propiedades se usan comillas para los valores.

Las propiedades son atómicas, es decir no permiten anidamientos y además no se pueden repetir a diferencia de los rótulos en los cuales se permite ambas cosas, es decir:

*<persona hobbie="música" hobbie="lectura"/>* --Inválido
*<persona> <hobbie> música </hobbie> <hobbie> lectura </hobbie> </persona>* -- Válido.

Ahora bien existen una serie de reglas que permiten decir si un documento XML está bien formado (válido):

- Que los rótulos guarden el orden de apertura y de cierre en forma correspondiente.
- Que no se repitan propiedades.(en un elemento)
- Que cumpla la DTD (si la hay, véase más abajo en esta misma sección)

Ya se ha visto como se puede representar una tabla sencilla en XML. Ahora veamos como se pueden representar datos semiestructurados, es fácil realizar la transformación:

```
{ persona: { nombre: { pila: "Juan", apellido: "Otero" } , edad: "22" }
  persona: { nombre: "Pedro", email: "peter@mailbox.com" }
 }
```

```
<bd>
        <persona>
                <nombre>
                        <pila> Juan </pila>
                        <apellido> Otero </apellido>
                </nombre>
                <edad> 22 </edad>
        </persona>
        <persona>
                <nombre> Pedro </nombre>
                <email> peter@mailbox.com </email>
        </persona>
</bd>
```

En XML los elementos (un elemento es un rótulo de apertura, su contenido y su rótulo de cierre) tienen orden pero ¡las propiedades no! Veamos:

*a). <nombre a="5" b="8"/>*
*b). <nombre b="8" a="5"/>*
*c). <emp> <nom> Ana </nom> <edad> 5 </edad> </emp>*
*d). <emp> <edad> 5 </edad> <nom> Ana </nom> </emp>*

a) y b) se consideran iguales pero c) y d) son diferentes; dependiendo de su uso lo anterior puede llegar a ser ventajoso o desventajoso.

Se puede definir una gramática (DTD) la cual debe cumplir el contenido de un documento XML. Veamos un ejemplo:

```
<!DOCTYPE bd [
        <!ELEMENT bd(emp*) >
        <!ELEMENT emp(ced,nom)>
        <!ELEMENT ced(#PCDATA)>
        <!ELEMENT nom(#PCDATA>
        ]
>
<bd>
<emp>
        <ced> 707 </ced>
        <nom> Pedro </nom>
</emp>
<emp>
        <ced> 808 </ced>
        <nom> Juan </nom>
</emp>
</bd>
```

En esta gramática, bd es el rótulo raíz y los elementos son:
bd el cual se compone de emp's (el * significa cualquier cantidad de ocurrencias), donde cada emp consta de ced y nom, los cuales son datos atómicos (#PCDATA).
Cada elemento emp debe poseer ced y nom y deben ser colocados en ese orden.
De acuerdo a lo anterior la muestra de datos (que "siguen" a la gramática), en el ejemplo anterior, son válidos.

Es posible también lo siguiente: *<!ELEMENT emp(ced, email?, tel+)>*
Lo que significa que en un elemento emp, ced es única, el email es opcional y que debe tener al menos un teléfono (pudiendo tener muchos).

Hay muchas opciones adicionales entre ellas cabe destacar que se puede especificar algo similar a las claves foráneas y primarias dentro de la gramática para las propiedades:
*<!ATTLIS libro autor IDREF #REQUIRED >*
*Ejemplo: <libro autor="A1">...</book>*

Aquí se tiene un intento "débil" para manejar claves foráneas que indica que cada elemento libro debe tener una propiedad *autor* obligatoria.
Igualmente se puede especificar una restricción de clave primaria por medio de:
*<!ATTLIS dep iddep ID #REQUIRED >*

Aquí se está indicando que todo *dep* (departamento) debe tener una propiedad *iddep* la cual funciona como clave primaria.

También existe la posibilidad *idrefs*, para indicar que se espera un conjunto de identificadores de elementos a los cuales se "apunta".
Resumiendo se tiene:

- *idref*: Es similar a una clave foránea.
- *id*: Es similar a una clave primaria.
- *idrefs*: Es similar a un conjunto de claves foráneas.

La gramática puede almacenarse en un sitio web y desde el documento XML invocarla, mediante esta forma se pueden compartir gramáticas entre varios documentos evitando repetirla logrando cierta estandarización. La forma de escribirlo es:

*<!DOCTYPE db SYSTEM "http://www.aaa.com/esquema.dtd">*

Similarmente se pueden hacer referencias externas así:

*<!ENTITY %resumen SYSTEM "http://www.aaa.com/curriculum.txt">*
*....*
*<persona> Hoja de vida </persona>*
*%resumen;*

Dicho lenguaje consta de muchos más elementos pero los básicos han sido ya descritos.
XML es débil en algunos aspectos:

- No hay forma de especificar restricciones de tipo check ni between
- Sólo hay un tipo de datos atómico (PCDATA)
- La integridad referencial requiere un mejor tratamiento

En la sección 2.3.6 se describen lenguajes para consultar documentos XML. El estado de estos es bastante maduro y continúan evolucionando. En el momento de escribirse este trabajo, ya se disponía del Internet Explorer en su versión 6.0, en el cual ya se puede cargar un documento XML y se puede observar en forma similar a un árbol de navegación, tipo explorador de Windows, pudiéndose expandir y contraer.
Hay muchos libros sobre XML uno que dedica un buen capítulo es [Buyens 01] y un excelente *white paper*, sobre tendencias en XML puede verse en [Web Page 8].

### 2.3.4 Lenguajes de consulta para datos semiestructurados

Ya sea que una base de datos se represente por medio de XML o por medio de otro formalismo, se han diseñado lenguajes de consulta para datos semiestructurados, las respuestas de las consultas son a su vez datos semiestructurados.
Para los siguientes ejemplos se utilizará la siguiente base de datos llamada *biblio*, la cual se pasa a representar a mediante XML, en la sección siguiente se verán lenguajes de consulta específicos para documentos XML.

```
<biblio>
        <book>
                <author> Roux </author>
                <author> Combal </author>
                <date> 1976 </date>
                <title> Database Systems </title>
        </book>
        <book>
                <author> Smith </author>
                <date> 1999 </date>
                <title> database systems </title>
        </book>
        <paper>
                <author> Smith </author>
                <date> 1998 </date>
                <title> Inf. 1 </title>
        </paper>
</biblio>
```

En el segundo libro se ha escrito a propósito todo el título en minúsculas.

El lenguaje que se describe a continuación se denomina LOREL y proviene del proyecto LORE del cual se habla en la sección 2.3.5. Los siguientes ejemplos son tomados de [Abiteboul 00]

Ejemplo 1.

```
select author: X
from biblio.book.author X
```

Explicación

X es una variable que se va "moviendo" por cada camino que cumpla la expresión del from, o sea busca caminos los cuales tengan como raíz biblio que luego sigan con book y que terminen con author.

Aquellos que cumplen son seleccionados, teniendo en cuenta que la X devuelve en el select su valor correspondiente (apellidos de autores para el ejemplo, el contenido dentro de <author> y </author>) y adicionalmente se le coloca a cada uno de ellos su rótulo correspondiente en este caso author, para que igualmente la salida sea semiestructurada.

Por lo tanto la salida en este caso tiene la forma:

{ author: "Roux", author: "Combal", author: "Smith" }

O representados en formato XML se tendría:

```
<author> Roux </author>
<author> Combal </author>
<author> Smith </author>
```

Nótese que el último autor seleccionado Smith corresponde al Smith que figura en la base de datos como autor de un book no de un paper.

Ejemplo 2.

*select row: ( select author: Y*
*from X.author Y )*
*from biblio.book X*

Aquí se observa una característica que existe también en SQL, las subconsultas escalares. En este ejemplo *por cada book*, el cual va siendo recorrido por la variable X, se ejecuta la subconsulta escalar. El primer libro posee dos autores los cuales son recorridos con la variable Y. Estos harán parte de la salida, la cual escrita en XML es:

*<row>*
*<author> Roux </author>*
*<author> Combal </author>*
*</row>*
*<row>*
*<author> Smith </author>*
*</row>*

Los anteriores ejemplos muestran los elementos básicos de cómo consultar datos semiestructurados. Existen otras posibilidades como se verá a continuación con los lenguajes de consulta para XML. Otro lenguaje para operar con datos semiestructurados, que no se describirá aquí, es UnQL (Unstructured Query Language), y está basado en principios similares a LOREL pero es más estricto que éste ya que no realiza ciertos *castings* o coerciones. Veamos un ejemplo, en LOREL es válido emitir una condición del tipo:

*where X.year > 1998*

Estrictamente hablando X.year es un conjunto de años (de acuerdo a la semántica de datos que esté representando puede ser un conjunto unitario o de muchos elementos). LOREL admite dicha comparación realizando coerción. UnQL exige una comparación con un operador orientado a conjuntos tipo *exists* o *in*, según el caso para llegar a realizarla.

### 2.3.5 El proyecto LORE

El proyecto LORE (Lightweight Object Repository) se desarrolló en la universidad de Stanford para manejar datos semiestructurados y se ha enfocado más recientemente en la definición de un

lenguaje de consultas declarativo para XML, desarrollando nueva tecnología para las búsquedas interactivas sobre datos en XML y construir eficientemente un procesador de consultas XML.

Construir un DBMS para XML presenta muchos problemas interesantes de investigación. Muchos de los componentes estándares de un DBMS, desde el lenguaje de consulta hasta la optimización necesitan ser revisados (por ejemplo mientras se comprenden bien las estructuras de indexación tradicionales --árboles B y las tablas Hash persistentes--, determinar cual es la mejor técnica a aplicar sobre datos en XML es un desafío en un DBMS). Además el proyecto es una plataforma eficaz para dar lugar a áreas de investigación tales como sencillas consultas de fuentes de datos externas y la búsqueda de elementos basados en la proximidad a otros.
El lenguaje LOREL descrito en sus aspectos básicos en la sección anterior, se ha migrado para consultar documentos XML.

Finalmente se ha desarrollado un sistema llamado Ozone [Lahiri 99], que permite que datos estructurados y semiestructurados coexistan en la misma base de datos, con referencias de un lado a otro.

Como se esboza en los anteriores lineamientos existen muchas áreas de investigación y muchas de ellas han sido desarrolladas en el proyecto LORE.
Mucha información sobre LORE incluyendo documentos técnicos y el producto está en [Web Page 9]. Un documento excelente para describir su lenguaje de consulta LOREL, del cual se habló en la sección 2.3.4, es [Abiteboul 97]. El prototipo fue bajado e instalado en un máquina Solaris y los diversos ensayos mostraron un prototipo bastante sólido en el cual se hicieron ensayos similares a los ejemplos presentados en las secciones anteriores.

### 2.3.6 Lenguajes de Consulta para XML

Como se mencionó en la sección 2.3.3, existen lenguajes para consultar documentos XML directamente. Se describirá brevemente uno de ellos, llamado XML-QL (XML Query Language) [Web Page 17]. Al final se mencionan otras alternativas.

Comencemos con un ejemplo. Supóngase que se tiene un documento XML llamado a.xml y que está ubicado en el sitio www.biblio.com.

El contenido de dicho documento es:

```
<biblio>
        <book>
                <editor> Hill </editor>
                <title> Isla </title>
                <author> Steve </author>
        </book>
        <book>
                <editor> Morgan </editor>
                <title> El león </title>
                <author> Johnny B </author>
                <price>5000 </price>
        </book>
        <book>
                <editor> Hill </editor>
                <title> Monicongo </title>
                <author> Don Duende </author>
                <price> 60000 </price>
        </book>
</biblio>
```

Consulta ejemplo 1.

```
where <book>
        <editor> Hill </editor>
        <title> $T </title>
        <author> $A </author>
        </book> in "www.biblio.com/a.com",
construct
<author> $A </author>
```

Significado de la consulta: (aunque es bastante intuitivo) se busca en todo el documento XML ubicado en el sitio www.biblio.com/a.com , todos aquellos libros, que tengan el patrón <editor> Hill </editor>, o sea libros publicados por la editorial Hill, que tengan un título (no importa cual título, lo que importa es que tengan un título contenido entre los rótulos <title> ... </title> ) y que tengan un author ( <author> ... </author> ) no importa cual.

Observaciones:

- La salida serán elementos de tipo $A, o sea lo contenido entre los rótulos <author> ...</author> para aquellos books que cumplieron el patrón y la salida queda precisamente cobijada por los rótulos <author> ... </author> para cada $A seleccionado.
- $A y $T son variables que se utilizan para referirse al contenido de los elementos <author> ... </author> y <title> ... </title> respectivamente; y se pueden igualmente utilizar para construir el resultado, en este caso haciendo uso de $A.

Nótese que el contenido de $A y/o $T podría estar conformado de estructuras anidadas, no sólo de elementos atómicos.

- Si un libro (book) tiene otros subelementos aparte de los que establece la consulta, no importan, es decir si un libro tiene por ejemplo <price> ... </price>, siempre y cuando cumpla el patrón, sin importar si tiene o no otros subelementos, será seleccionado.
- No importa si el book o alguno de sus subelementos tienen propiedades, si cumplen el patrón especificado es suficiente para ser seleccionado.(Aunque las propiedades también pueden consultarse).

Para la muestra de datos anterior la salida generada es:

*<author> Steve </author>*
*<author> Don Duende </author>*

La salida son elementos XML, aunque les falta un elemento raíz, el cual puede ser añadido como se muestra en el siguiente ejemplo.

Consulta ejemplo 2.

```
<rta>
where <book>
        <editor> Hill </>
        <title> $T </>
        <author> $A </>
        </> in "www.biblio.com/a.xml",
construct <result>
                <author> $A </>
                <title> $T </>
</rta>
```

La condición de búsqueda es la misma del primer ejemplo pero se tienen varias adiciones:

- Se usa la notación </> como abreviatura para evitar escribir el nombre del rótulo de cierre correspondiente.
- Se añaden los rótulos <rta> </rta> como elementos raíz de la salida. Estos sólo aparecerán una vez en la salida que se genere.
- Para cada libro (book) se generan elementos XML de la forma:

```
<result>
        <author> ... </author>
        <title> ... </title>
</result>
```

Donde los puntos suspensivos (...) son remplazados en cada libro por el contenido de las variables $A y $T respectivamente.

Para este ejemplo la salida generada será entonces:

*<rta>*
*<result>*
*<author> Steve </author>*
*<title> Isla </title>*
*</result>*
*<result>*
*<author> Don Duende </author>*
*<title> Monicongo </title>*
*</result>*
*</rta>*

Existen otros lenguajes de consulta para XML, dos de los más interesantes son XPath y XML Query, los cuales pueden consultarse en [Web Page 10]. Tomando a XML-QL como lenguaje de aproximación de consulta para XML se observa que su concepción es bastante aceptable en especial para aquellos que han programado con SQL la analogía entre ambos es fuerte.

Cualquier lector puede crear un documento XML, colocarlo en cualquier servidor, luego dirigirse a [Web Page 11] y entrar allí el código de la consulta correspondiente. Desde tal sitio se ejecuta la consulta y se devuelve el resultado allí mismo. Por lo menos hasta la fecha de entrega de este proyecto el sitio estaba activo. Por lo tanto, para probar lo mostrado aquí se requiere simplemente acceso a Internet, el resto es crear un documento XML (con cualquier editor) y escribir alguna consulta, guiándose por la sintaxis que allí se presenta igualmente.

### 2.3.7 Otras tecnologías relacionadas con XML

DOM (Document Object Model) y SAX (Simple API for XML) son dos formas de consultar documentos XML.
Un excelente curso sobre DOM está en [Web Page 12] y en [Buyens 01].
Oracle a partir de la versión 8i soporta DOM [véase Chang 01]. JAVA también ofrece librerías que soportan toda la funcionalidad para manipular documentos XML usando DOM y SAX.

dbXML [Web Page 18] es una base de datos nativa de XML de código abierto. Almacena e indexa documentos de XML comprimidos con el fin de proveer datos a una aplicación cliente con muy bajo consumo de procesamiento del servidor.
eXist es otra base de datos nativa para documentos XML la cual soporta complejas búsquedas de texto.
El motor de búsqueda ha sido diseñado para crear consultas usando el lenguaje de consulta XPath para XML, usando índices para todos los elementos y las propiedades. Puede ser bajado del sitio [Web Page 13] y allí mismo existe un demo, similar al de XML-QL (ver sección 2.3.6), el cual provee una sencilla forma para enviar consultas XPath a una base de datos eXist.

eXist está en evolución (por lo menos al momento de realizarse este trabajo); algunos de entre los muchos desarrollos posteriores incluyen: Métodos de manipulación DOM, Xupdate [Web Page 19] --lenguaje propuesto para modificar documentos XML-- y soporte de más funciones de XPath.

# 3 Modelo de Componentes Educativos Propuesto

## 3.1 Introducción

Revisada así la literatura del estado del arte sobre: Componentes reutilizables, sistemas de recursos educativos y datos semiestructurados, no se encontró hasta donde se pudo establecer, un modelo o producto que trate a estos recursos como componentes reutilizables, ni que explote de una manera adecuada el uso de los metadatos en estos. Se propone un modelo para recursos educativos concebidos como componentes reutilizables para un sistema adaptable orientado a la web, en los cuales se explotarán los metadatos mediante un lenguaje de manipulación que será concebido bajo un esquema de datos semiestructurados.

El lenguaje ofrece operadores para construir, agregar y buscar CE's. Los operadores de construcción y agregación permiten definir *patrones* de curso en vez de cursos fijos, ya que en tiempo de ejecución los usuarios (estudiantes) podrán personalizarlo de acuerdo a sus necesidades. Cada CE posee una especificación basada en lógica lo que hace posible definir un operador de búsqueda por proximidad (subsumption, ver sección 3.15).

Se utiliza un modelo de dominio donde cada CE juega uno o varios roles (papeles) frente a un concepto del dominio. Un rol es la forma como un concepto del dominio es desarrollado, por ejemplo ofreciendo una definición, presentando un ejemplo, planteando ejercicios etc. Por otro lado existen varias relaciones entre los conceptos del dominio: relaciones retóricas y jerárquicas, lo cual provee de información respecto a la coherencia global entre ellos. A diferencia por ejemplo de la propuesta del LOM (ver 2.2.2.3.3), no hay relaciones directas entre los CE's.

Los objetivos que se persiguen con el modelo son:

a) Proveer flexibilidad de tal forma que se pueda extender el repositorio con nuevos CE's. Esta flexibilidad está basada en la *definición de los CE's* -- es decir, un CE debe ser una unidad independiente de contenido -- y en el *sistema de clasificación usado* -- el cual no requiere una relación directa entre los CE's almacenados en el repositorio.

b) Facilitar la reutilización de CE's por parte de no expertos en tecnología. Todavía hoy el proceso de reutilización es llevado a cabo por programadores o expertos en ciencias de la computación [Roschelle 99]. Un mecanismo sencillo de reutilización requiere un nivel adecuado de granularidad para definir los CE's, tal como el nivel de abstracción usado por los profesores durante el plan de un curso, y operadores de composición los cuales faciliten la creación de otros CE's.

c) Soportar un ambiente de aprendizaje interactivo basado en CE's para proveer adaptación a los requerimientos de los usuarios. Los operadores de composición permiten definir patrones de curso los cuales son personalizados en tiempo de ejecución, además el sistema provee un operador de búsqueda por proximidad (véase sección 3.15), de tal forma que los usuarios (estudiantes en este caso) puedan hallar el CE más cercano de acuerdo a sus objetivos.

## 3.2 Definición de CE

Un CE es una unidad de composición con una interfaz que provee información acerca de los requerimientos para su uso, ensamble y remplazo durante el proceso de aprendizaje. Esta unidad puede ser usada independientemente o para composición de otros CE's.

Sea C={ $c_i$ } un conjunto de CE's almacenados en una base de datos, donde $c_i$ es descrito por *[Coid, Input, Output, Metadata, Composition ]* con la siguiente definición: Ver figura 3.1

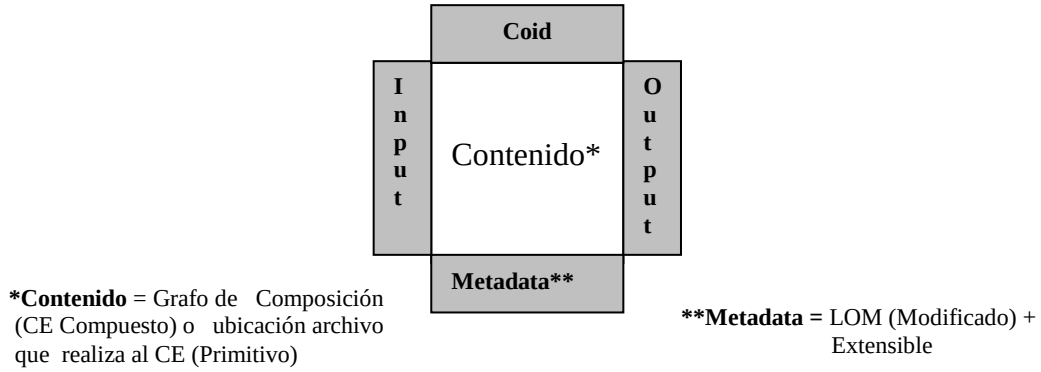



Figura 3.1: Representación gráfica de un CE

a) *Coid* = Identificador único del CE en la base de datos.

b) *Input* = { $I_i$ }, i = 0…n. Cada $I_i$ es una tupla *[label, value, documentation]*, donde *label* es el nombre de la entrada, *valor* es asignado durante la ejecución (aunque posee un valor por defecto) y *documentation* da información sobre valores que puede adoptar Ii.

La estructura de la documentación podría incluir elementos, como por ejemplo: Posibles valores para $I_i$, comentarios, valor por default etc.

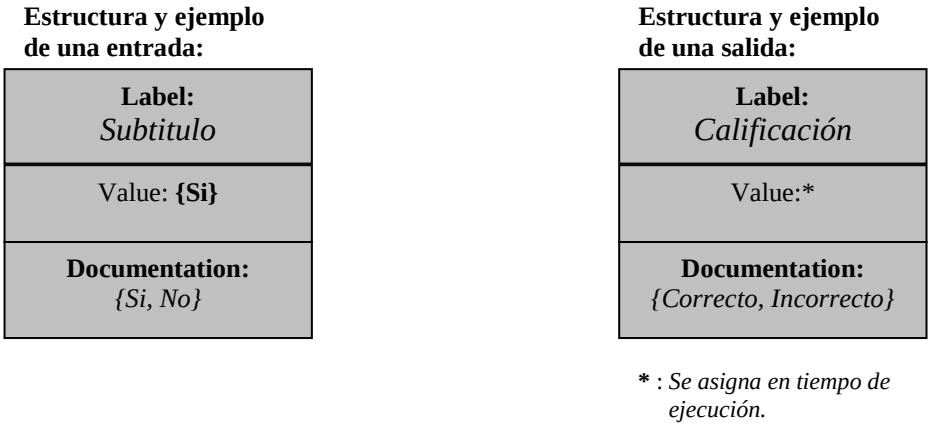



Figura 3.2: Estructura de una input (entrada) y de una Output (salida)

c) *Output* = { $O_i$ }, i = 0…m. Conjunto de servicios que provee el CE. Cada $O_i$ puede retornar información acerca del uso de la instancia del CE. Por ejemplo, puede retornar el desempeño específico de un estudiante durante un taller. Cada $O_i$ es una tupla *[label, value, documentation]*, *label* y *documentation* tienen el mismo significado que en Input, mientras que *value* será retornado por la instancia del CE luego de su ejecución.

La *interfaz* de un CE está constituida por sus inputs y outputs. Ver figura 3.2

d) *Metadata* = { $M_i$ } i = 1...k. Este conjunto es usado para describir propiedades no funcionales para el CE y ayuda en la búsqueda de CE's existentes en la base de datos. Se usa como base los metadatos de LOM (ver apéndice 1) pero se puede extender dinámicamente (ver sección 3.5). Cada $M_i$ puede ser

- (Categoría). Ejemplo: (General, Technical).
- (Categoría_Padre, Categoría, Valor(es) ). Ejemplo: (General, Title, 'Energía Solar').

e) *Composition*: Si el CE es compuesto (véase sección 3.6), este elemento describe la topología de la composición, ver figura 3.3. (véanse los operadores de composición secciones 3.8.1 y 3.8.2). Si el CE es primitivo la composición es nula.

Un CE se dice es *compuesto* (CEC) si está basado en la agregación de otros CE's (véase operadores de composición, secciones 3.8.1 y 3.8.2), será primitivo (CEP) de lo contrario, es decir, está basado en un archivo que representa el contenido educativo del CE (Diapositivas, archivo en Word, un vídeo etc.)

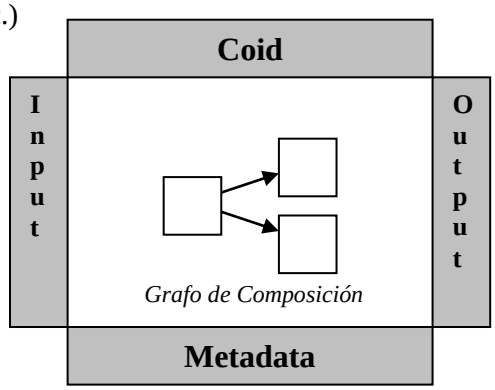



Figura 3.3: Representación gráfica de un CEC

El archivo (localización por ejemplo un URI) que implementa la "realización" de cada CEP que exista en el sistema se considera como una caja negra para nuestro sistema, es decir, el creador de dicho recurso, lo concibe y adapta de acuerdo a sus necesidades. Por ejemplo supóngase que se crea un vídeo para enseñanza de idiomas, su autor le provee de una entrada (llamada por ejemplo subtítulo) que lo parametriza de tal forma que se pueda observar en inglés o en español. De acuerdo al sistema de caja negra el comportamiento de este recurso se muestra en la figura 3.4:

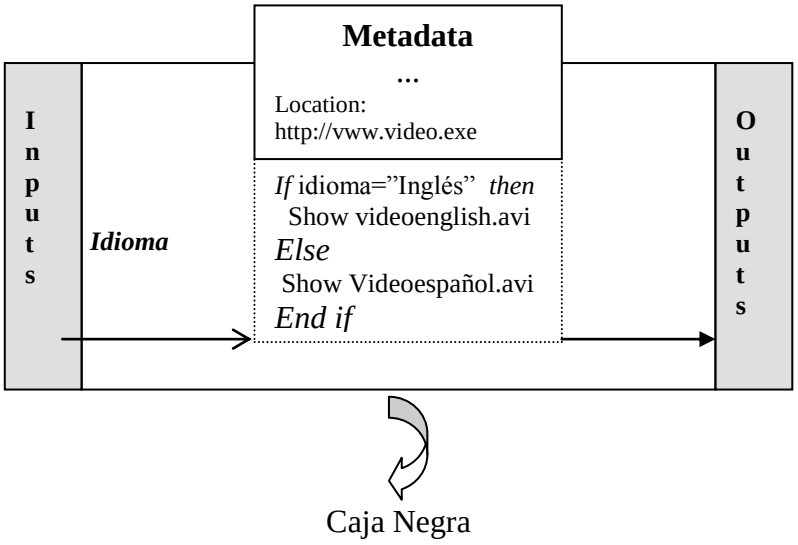



Figura 3.4: Comportamiento "*caja negra*" de un CE

Los archivos (recursos educativos) con los cuales se trata aquí son todos aquellos elementos digitales que es posible sean desplegados por un browser (Netscape, Internet Explorer, Neoplanet etc.). La evolución y adición de complementos de software (*plugins*) en los navegadores contribuirá de manera significativa a que cada vez sean más los tipos de archivos susceptibles de ser instanciados por los navegadores. Esta misma idea aparece en [Weber 97]

## 3.3 Modelo del dominio

Sea el grafo G = < C, E >, donde C son nodos que representan el nombre de un concepto del dominio y E son arcos (*edges*), es decir, relaciones entre dos nodos.
Cada concepto puede tener un conjunto de sinónimos asociado (Ejemplo: Grafos Dirigidos = Dígrafos) y de traducciones (Ejemplo: Bases de Datos = Databases).
Se definirán dos tipos de relaciones posibles entre los conceptos del dominio en nuestro modelo aunque [Wu 01]: "*En teoría se pueden definir muchos tipos de relaciones entre los conceptos*", (ver además sección 2.2.3.3).

### 3.3.1 Relación jerárquica

"*Is broader than"/"Is narrower than"*: (es más amplio que / es más preciso que). Se dice que el concepto A es más amplio que el concepto B si siempre se cumple que cualquier búsqueda de ítems que incluyan a A implica que los ítems que incluyan a B deban ser encontrados. Similarmente se dice que B es más preciso que A. [Fischer 98]

Se define una jerarquía (árbol) $T_i$ para cada dominio específico (biología, electrónica, física etc.). Esta es definida usando la relación *broader/narrower* . Todos los nodos dentro de la jerarquía $T_i$, excepto el concepto raíz, deben poseer una relación "*is narrower than*" con uno y sólo un concepto. Todo concepto dentro de la jerarquía Ti puede tener cero o muchas relaciones "*is broader than*". La raíz en cada jerarquía $T_i$ es el concepto más general y los nodos finales corresponden a los conceptos más específicos del dominio.

### 3.3.2 Relación retórica

Dos conceptos tienen una relación retórica si ésta existe independientemente de cómo ambos conceptos son desarrollados.

Existe un conjunto de relaciones retóricas que se han tomado de [Mann 87]; sin embargo, el administrador del sistema podrá extenderlas:

a) *Restatement*. El concepto A es una re-expresión del concepto B. En forma correspondiente el concepto B es re-expresado por el concepto A.
b) *Contrast:* El concept A es una teoría alternativa para el concepto B. En forma correspondiente el concepto B es una teoría alternativa para el concepto A.
c) *Extend:* El concepto A desarrolla y añade nuevos elementos al concepto B. En forma correspondiente el concepto B es extendido por el concepto A.
d) *Antitheses*. El concepto A es una antítesis del concepto B. En forma correspondiente el concepto B es una a*ntitheses* para el concepto A.

Al momento de componer CE's, este tipo de relaciones proveen ayudas pedagógicas, ya que ofrecen información acerca de coherencia global entre los diferentes conceptos. De esta forma, es fácil definir una secuencia lógica para los conceptos de un curso.

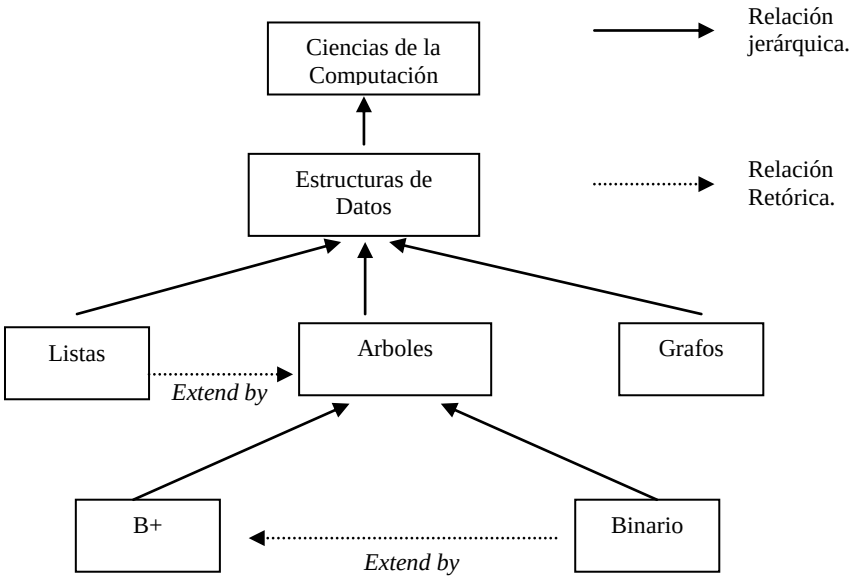



Figure 3.5: Fragmento del modelo de dominio para Estructuras de Datos

La figura 3.5 muestra un fragmento de la jerarquía del dominio para ciencias de la computación. Este usa relaciones *broader-narrower* (Data Structures *is narrower than* Computer Sciences) y relaciones retóricas (Trees *extends* Lists).

Existirá un administrador del sistema que se encargará de crear los árboles de dominio correspondientes a cada disciplina de acuerdo a estándares que se establezcan en una comunidad e igualmente para establecer que relaciones y de que tipo existen entre los diferentes conceptos.

### 3.3.3 Relaciones entre los CE's y los conceptos del dominio

#### 3.3.3.1 Relación de $R_{subject}$

Una relación de $R_{subject}$ se establece entre un CE y un concepto del dominio. Esto significa que el CE desarrolla una o varios roles del concepto. Por ejemplo, el CE $c_k$ *motivate*, *define* y *evaluate* el concepto A. Un CE $c_i$ define sus relaciones de $R_{subject}$ a través de un conjunto:
$R_{subject} = \{ rel_j \} \neq \phi$. Donde cada relación $rel_j$ es representada por la tupla $[\{rol_k\}, c_n]$ donde:

a) $\{rol_k\}$: Representa el conjunto de roles en la cuales un CE desarrolla un concepto. Los posibles roles $rol_k$ son: ( Aunque el administrador del sistema puede incluir otros roles)

- *Motivate*: El CE $c_i$ ofrece una introducción acerca de $c_n$.
- *Define*: El CE $c_i$ presenta el significado de $c_n$.
- *Evaluate*: El CE $c_i$ contiene ejercicio(s) de $c_n$.
- *Demonstrate*: El CE $c_i$ desarrolla una demostración de $c_n$.
- *Apply*: El CE $c_i$ presenta una aplicación de $c_n$.
- *Experiment*: El CE $c_i$ describe un experimento para explicar a $c_n$.
- *Example*: El CE $c_i$ presenta ejemplos o casos de estudio sobre $c_n$.

b) $c_n$ : Es un concepto tomado del modelo del dominio.

El $R_{subject}$ para el CE $c_k$ descrito arriba, sería entonces: *[{motivate, define, evaluate}, Concept-A ]*
La figura 3.6 muestra el $R_{subject}$ de un CE.
Se espera que los CEP's sólo posean una relación de Rsubject (es decir que desarrollen un concepto), se dice que este tipo de CE's son de granularidad fina y fomentan la reutilización. Aunque no se prohiben, los CEP's que posean varias relaciones en su conjunto no son deseables

(granularidad alta) ya que constituyen una "mezcla" de conceptos. Un CEC bien podría tener varias relaciones en su conjunto aunque igualmente se espera que posea sólo una.

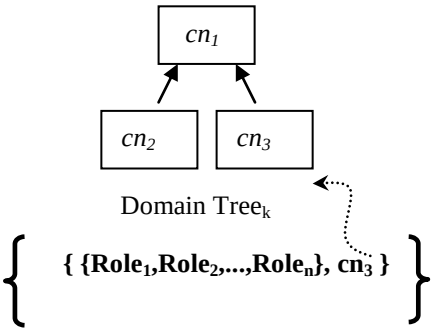



Figura 3.6: $R_{subject}$ de un CE

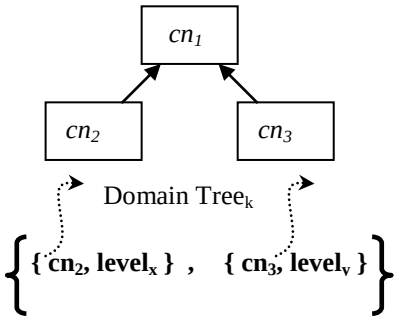



Figura 3.7: $R_{prerequisite}$ de un CE

#### 3.3.3.2 Relación $R_{prerequisite}$

Una relación de prerrequisito se establece entre un CE y un concepto del dominio (ver figura 3.7). Esto significa que durante el proceso de aprendizaje, un usuario debería conocer el concepto antes de usar ese CE. Cada CE $c_i$ define un conjunto (el cual puede ser vacío) de relaciones $R_{prerequisite} = \{ rel_j \}$. Cada relación $rel_j$ es representada por la tupla *[$c_n$ , level]* donde:

a) $c_n$ : Es un concepto tomado del modelo del dominio.

*b)* *level*: Puede tomar uno de los siguientes valores: *{"very low", "low", "medium", "high", "very high"}*. Ejemplo: *["Grammar", low];* el CE que posea esta relación, requiere preparación de nivel *"low"* en *"Grammar"*

**Nota:** Se han empleado 5 posibles valores para el nivel (similar a como se hace en algunos elementos de LOM, ver apéndice 1); por supuesto podrían extenderse a muchos más o establecerse de otra forma.

#### 3.3.3.3 Restricción de $R_{subject}$ y $R_{prerequisite}$

Si el CE $c_i$ tiene los conjuntos de relaciones $R_{subject}$ y $R_{prerequisite}$, donde $\{s_i\}$ es el conjunto de conceptos de dominio usados por $R_{subject}$ y $\{s_k\}$ es el conjunto de conceptos del dominio usados por $R_{prerequisite}$, entonces $\{s_i\} \cap \{s_k\} = \phi$. Es decir un CE no puede desarrollar un concepto dado y al mismo tiempo tenerlo como prerrequisito.

La figura 3.8 muestra las relaciones $R_{subject}$ y $R_{prerequisite}$ de algunos CE's con un fragmento del modelo de dominio de ciencias de la computación para estructuras de datos.

Algunas observaciones sobre la figura 3.8:

- El CE c1 tiene como prerrequisito saber de pilas y desarrolla el tema de expresiones.
- El CE c2, tiene como prerrequisito expresiones y desarrolla el tema de pilas.
- El CE c3 es inválido porque Rsubject y Rprerequisite no se pueden interceptar.
- El CE c4 muestra que un CE puede tener varios prerrequisitos, incluso de diferentes "ramas" y adicionalmente que puede desarrollar varios conceptos, (y con cada uno estableciendo un rol), incluso de diferentes niveles.(aunque es una situación que podría ser redundante, pero se permite por flexibilidad).

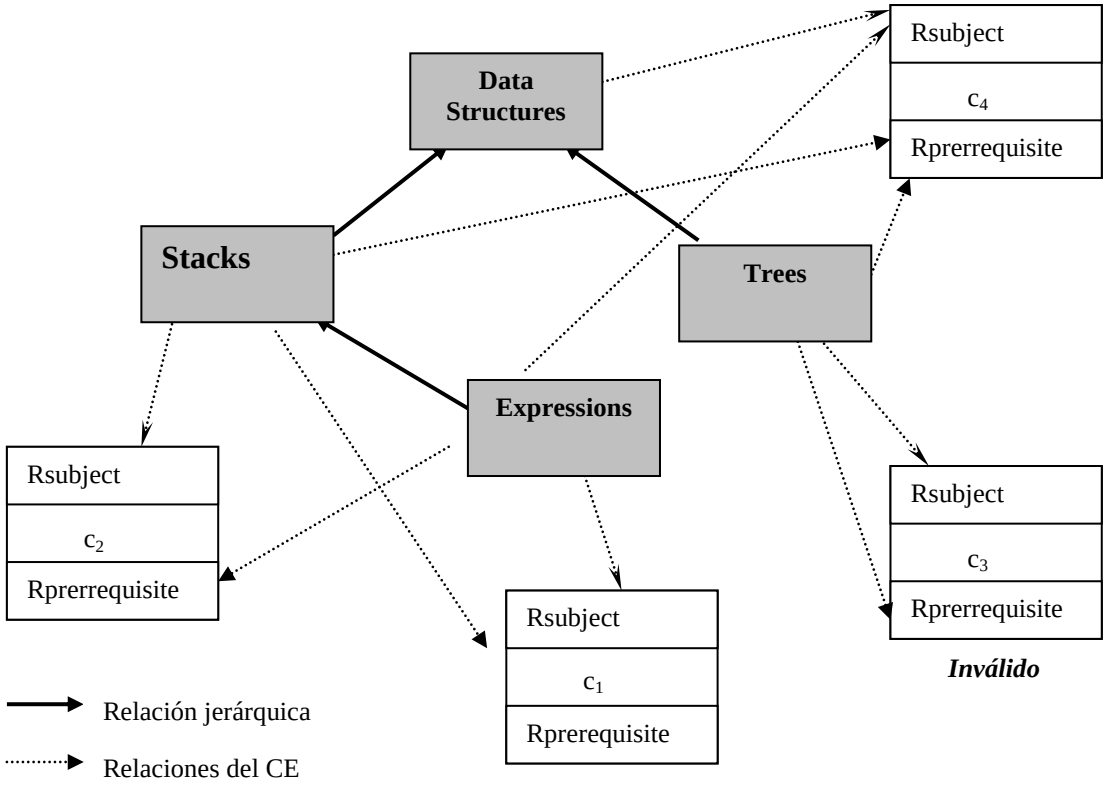



Figura 3.8: CE's, árbol de dominio y relaciones $R_{subject}$ y $R_{prerequisite}$

#### 3.3.3.4 Observaciones acerca del modelo

Varios de los sistemas de aprendizaje analizados usan un modelo del dominio para representar todas las posibles relaciones entre conceptos del dominio y entre los recursos educativos. Sin embargo,

para un sistema de gran escala, el cual posee varios dominios del conocimiento y profesores, esto sería demasiado complejo porque es muy difícil definirle a un nuevo CE todas las posibles relaciones con los demás CE's.

Como resultado se ha definido entonces un modelo de dominio como una ontología de términos que soporta una clasificación general (relaciones *broader than* / *narrower than*) por concepto, que provee ayudas pedagógicas (relaciones retóricas) y que ayuda a la búsqueda de CE's. Adicionalmente agregar un nuevo CE al sistema sólo demandará definir las relaciones $R_{subject}$ y $R_{prerequisite}$ con los conceptos del modelo del dominio.

### 3.4 Granularidad de los CE's y relaciones entre los conceptos del dominio

El objetivo en nuestro trabajo no es el de estructurar el modelo del dominio propuesto exhaustivamente, es decir, representando todos los posibles hechos y relaciones. Se trata mas bien de una aproximación sencilla con las siguientes características:

a) Cada dominio del conocimiento usa una jerarquía de conceptos diferente. El nivel de granularidad de los conceptos varía ampliamente entre los dominios, por ello no todos los caminos desde la raíz hasta los nodos hojas tendrán la misma longitud. Estas jerarquías facilitan la búsqueda de CE's en la base de datos.
b) Se espera que los conceptos más específicos (nodos hoja) sirvan para crear *tópicos* o *ejercicios* para un curso (véase sección 3.9). Si un concepto es una combinación de conceptos más pequeños, este no será un concepto hoja dentro de la jerarquía correspondiente, ya que puede ser desarrollado usando la composición de CE's. (ver operadores de composición secciones 3.8.1 y 3.8.2)
c) Las relaciones retóricas ofrecen ayudas pedagógicas a los autores, además los estudiantes pueden usarlas para navegar en el dominio. Sólo el administrador del sistema puede definir nuevas relaciones retóricas.
d) Un CEP desarrolla uno o varios aspectos de un concepto. Los roles dentro de su $R_{subject}$ representan esta información.
e) Durante el momento de creación de cursos, los autores definen *patrones* (ver 3.6.1) de cursos en vez de generar un curso específico. En tiempo de ejecución del CE (patrón del curso), este llegará a ser un curso instanciado para cada individuo, teniendo en cuenta sus preferencias, background y sus objetivos (ver operador *Replace* 3.14).

### 3.5 Los metadatos utilizados

Concretamente en el campo de la educación, los metadatos permiten a los educadores y estudiantes: Buscar, evaluar y usar recursos de aprendizaje, facilita su intercambio y su reorientación (adaptaciones) luego de pasar por un proceso de evaluación.
El uso de los metadatos permite a los "agentes" de búsqueda "orientarse" mejor y recuperar los objetos más relevantes, así como realizar clasificaciones.

#### 3.5.1 Elementos de LOM no usados en nuestro modelo

En nuestro proyecto se han adoptado los metadatos de LOM (ver sección 2.2.2.3.3) debido a que se han convertido prácticamente en un estándar para los metadatos educativos. Sin embargo algunos de sus elementos no serán usados y será a su vez ampliado (ver sección 3.5.2).
De hecho LOM es una guía, y puede ser alterado de acuerdo a las necesidades, de hecho en [Suthers 01], se explica un proyecto en el cual se usó LOM y se modificó igualmente suprimiéndole y agregándole elementos.

Por ejemplo la categoría "relation", no será usada, porque nuestro objetivo es que cada CE tenga al menos una relación en su $R_{subject}$ y no es necesario definirle relaciones con los otros CE's. Hay una razón para ello: Sería muy costoso el añadir un nuevo CE al sistema, ya que los autores tendrían que hallar todos los posibles CE's existentes en la base de datos que tendrían relación con el nuevo.

Otras propiedades que el estándar LOM define para los elementos educativos son, {*requires/isRequiredBy, isBasedOn/isBasisFor, isPartOf / hasPart, references / isReferencedBy*}. Estas son igualmente contrarias al concepto de componente y son cubiertas por las relaciones de $R_{subject}$ y $R_{prerequisite}$.
La propiedades *Requires/IsRequiredBy* significa que el elemento educativo A es un prerrequisito para el elemento educativo B, esto puede ser inferido a través de $R_{prerequisite.}$
Con respecto a las propiedades *IsBasedOn/IsBasisFor* y *Requires/IsRequiredBy* y de acuerdo a [Seeberg 00] "*La relación que establece* Requires/isRequiredBy *es inapropiada. Si un elemento depende completamente de la existencia y accesibilidad de otro, la idea de un modelo de aprendizaje independiente y reutilizable se pierde. Los elementos conectados por medio de la relación* isBasedOn/IsBasisFor *tienen el mismo problema... si se desea expresar el hecho de que un*

*elemento trata de un concepto, el cual es explicado en otro elemento, entonces se debe expresar este hecho conectando esos dos elementos entre sí...”*. Un problema similar se presenta con la propiedad *isPartOf / hasPart.*

Respecto a la propiedad *References/isReferencedBy* que establece una relación que dice que dos elementos tienen información acerca de tópicos similares, tampoco se utilizará, ya que las relaciones jerárquicas y retóricas entre conceptos permiten por transitividad hallar todas las posibles referencias entre CE’s.

Con respecto a los elementos { *isFormat / hasFormat* e *isVersionOf / hasVersion*} el sistema de aprendizaje posee las propiedades “version” y “format”.

Se obviará también el elemento "difficulty" dado que tiene un alto grado de subjetividad, no aplicable en ambientes en donde diversos usuarios van a determinar el grado de dificultad para diferentes CE’s. En su lugar se va a evaluar que prerrequisitos tiene un CE y cuales de ellos han sido cubiertos por el estudiante.

Un listado completo de los elementos de LOM e indicando cuales se usan en nuestro proyecto se puede observar en el apéndice 1.

### 3.5.2 Extensión dinámica de LOM

Un creador de un CE podría estar interesado en documentar una serie de elementos adicionales sobre dicho objeto y para los cuales no existe un cubrimiento en LOM. Ha habido aproximaciones que pretenden extender al LOM básico para ciertos tipos de recursos [El Saddik 01] y para ciertos contextos educativos [Suthers 01]. En concreto la contribución del proyecto de [El Saddik 00] fue la de extender LOM con elementos adicionales que son aplicables y necesarios para la documentación de recursos educativos *multimediales* y se sugiere que otros recursos como los generados con *VRML* (Virtual Reality Markup Language) y *películas* también tienen características que requieren la extensión de LOM; el proyecto de [Suthers 01] adaptó a LOM para suplir necesidades en el contexto de recursos para la educación en las escuelas públicas, desde Kindergarten hasta el grado 12 (conocido como K-12).

En nuestro proyecto no se pretende cubrir o extender LOM de acuerdo a todos los posibles tipos de recursos educativos específicos imaginables. La intención es trabajar con el LOM básico (quitándole algunos elementos ver 3.5.1) el cual estandariza los metadatos de los recursos

educativos + unos metadatos *extensibles (**ME**)* o de *dominio* (domain metadata), en la cual los autores podrán documentar libremente las propiedades que necesiten para sus recursos.

Este ME es similar a un esquema de directorios (categorías) y subdirectorios (subcategorías) de esquema arbóreo y su categoría principal (raíz) se denominará *Extensible*.

Los elementos finales serán atributos.Veamos un ejemplo, supóngase que el autor de un recurso va a matricularlo en nuestro sistema y aparte de llenar el LOM básico desea documentar cierta información que tiene que ver con un proyecto de una universidad en el cual está enmarcado ese recurso. Los metadatos para este recurso podrían tener la siguiente estructura:

*Zona de metadatos obligatorios LOM*
***General***
***Title:*** *"Energía solar"*
***Language:*** *"Español"*
***...***
***Technical***
*Format: "PDF"*
*...*
*Fin zona de metadatos obligatorios LOM*

*Zona de metadatos extensible (Opcional):*
***Extensible***
***Proyecto***
***Nombre:*** *"Omega"*
***Director:*** *"Dr. Guy"*
***Fase:*** *12*
***...***
*Fin Zona de metadatos extensible* (Opcional).

En este ejemplo el autor habrá llenado toda la información que solicita el LOM básico y ha adicionado libremente la siguiente información: el CE tiene que ver con un proyecto llamado *Omega* cuyo director es el *Dr. Guy* y está asociado a la fase *12*.

Para lograr esta flexibilidad de documentación de cada CE se utilizará un esquema semiestructurado, igualmente se pudo haber empleado un esquema relacional, pero de una u otra forma, el objetivo es dotar a cada CE de un esquema de documentación flexible. Por supuesto también conlleva desventajas (optimización compleja, almacenamiento pesado), como es natural en las bases de datos semiestructuradas, ver sección 2.3.

Otra desventaja sería una posible "desestandarización" de los metadatos de los recursos educativos. Sin embargo esto puede verse disminuido por dos factores:

-Seguir una política de estandarización de los metadatos extensibles por parte de un grupo de profesores. Por ejemplo si se van a documentar recursos multimediales, perfectamente se podría adoptar la propuesta de [El Saddik 00] (u otra) y ser llevada a dicha zona de ME.
-Se cuenta siempre con el LOM base en todo CE (Véase sección 3.5)

El ME puede ser *consultado*, *actualizado* y *eliminado*, ver operadores secciones 3.11 y 3.12; lo cual le provee de la funcionalidad necesaria que permita explotarlo.
Incluso aunque no se siguiese una política estándar del ME, sería posible aún sacar cierto provecho de dichos metadatos, considérese el ejemplo:
Un CE $c_1$ ha sido documentado en su ME así:

***Extensible***
***Proyecto***
***Fase**: 12*
*...*

Otro CE c2 ha sido documentado así:

***Extensible***
***Fase**: 12*
*...*

Como se puede observar el atributo fase está en diferentes niveles, sin embargo y como se verá en la sección 3.11.2 aun así es posible emitir una consulta del tipo: "*Seleccionar los CE's que pertenezcan a la fase 12*" (la condición en el lenguaje que aquí se propone en concreto se escribirá así: **.fase=12*) y tanto $\mathbf{c_1}$ como $\mathbf{c_2}$ serán encontrados (aprovechamiento de las características de los lenguajes semiestructurados).

### 3.6 Estructura de un CEC

Un CEC $c_c$ es un grafo dirigido acíclico conectado G = <V, E>, donde V es un conjunto de vértices llamados nodos concepto y E es un conjunto de arcos (edges) nombrados (con rótulo) entre dos nodos.
Para cada $c_i$ i = 1,…, n se define un nodo concepto (ver sección 3.6.1).
Cada posible camino (trayectoria) en G se crea mediante el operador *Sequence*. (véase 3.8.1)
Un camino o trayectoria ($nc_1$, $nc_2$, ..., $nc_n$) es un conjunto de nodos que posee un nodo *origen (al cual no llegan otros arcos*) y un nodo final (*del cual no salen arcos*), y $\forall$ ($nc_i$, $nc_{i+1}$); i=1, ..., n-1,

existe el arco en G. Se define Lenght (camino) como el número de nodos concepto presentes en la secuencia y debe ser >1.

El orden de cada camino significa la secuencia (*tabla de contenido*) propuesta por el creador del CE para los estudiantes por razones pedagógicas (para las razones y ventajas de una tabla de contenido ver 2.2.3.2

### 3.6.1 El nodo concepto

Un nodo concepto es una tupla *T = [Noid, Suggested CE, ({Roles}, Domain Concept ), prerequisites, mandatory, weight, interchangeable]* (ver figura 3.9) donde:

a) *Noid*: Identificador único del nodo en la secuencia

*b) Suggested CE $c_i$:* Es el *CE sugerido* para el nodo concepto *i*. $c_i \neq c_c$ (es decir no se permite recursividad). $c_i$ se denomina CE partícipe o miembro (CEM) de $c_c$

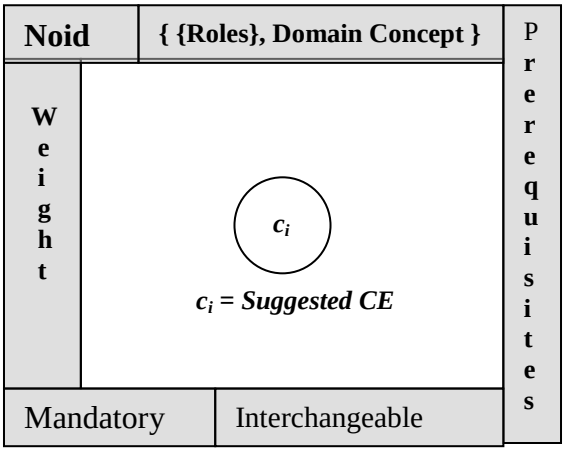



Figura 3.9: Estructura de un nodo concepto

*c) Domain-concept y roles*: Está constituido por una relación existente en el conjunto $R_{subject}$ de $c_i$ . Es decir el *domain concept* del nodo concepto y sus correspondientes roles, se extrae del $R_{subject}$ del CE sugerido. Este es entonces el *concepto desarrollado* por el nodo concepto. Gráficamente se puede ver en la figura 3.10:

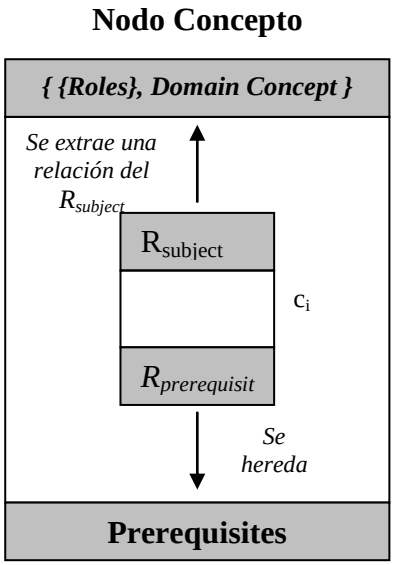



Figura 3.10: Relación entre un Nodo concepto y su CE sugerido

*d) Prerequisites*: Es igual al conjunto $R_{prerequisite}$ de $c_i$ , es decir el conjunto de prerrequisitos del nodo concepto está dado por los prerrequisitos de su CE sugerido.

*e) Mandatory:* Esta es una propiedad del nodo concepto que puede tener uno de dos valores: *True* o *False*. *True* significa que el estudiante debe estudiar el nodo concepto, mientras que *False* significa que ese nodo concepto es opcional.

*f) Weight:* Es el peso relativo del nodo concepto en el $c_c$. Este será usado para medir el desempeño del estudiante. Este tiene un valor entre 0 y 100. La suma de los pesos de los nodos conceptos de un CEC deben sumar 100. Los nodos concepto que son opcionales (mandatory = False) tienen un peso de 0.

*g) Interchangeable:* Esta propiedad tiene dos posibles valores *True* or *False*. *True* significa que el CE sugerido puede ser remplazado por otro en tiempo de ejecución (ver operador *Replace* sección 3.14), mientras que *False* significa que el CE sugerido no puede ser remplazado.

**Nota:** Un CEC se dice que es **patrón** si posee *al menos* un nodo concepto con la propiedad de *interchangeable* en true. Se dice que es *específico* de lo contrario.

### 3.6.2 Arcos

El modelo del dominio o el creador de un CEC provee *rótulos* a los arcos del grafo de composición. Si existe una relación retórica entre los conceptos del dominio usados por dos nodos conceptos que están conectados, el sistema puede automáticamente definir esta relación. El autor también tiene la opción de colocar el rótulo que él desee ya sea que exista o no relación retórica entre los dos conceptos. La figura 3.11 ilustra este aspecto:

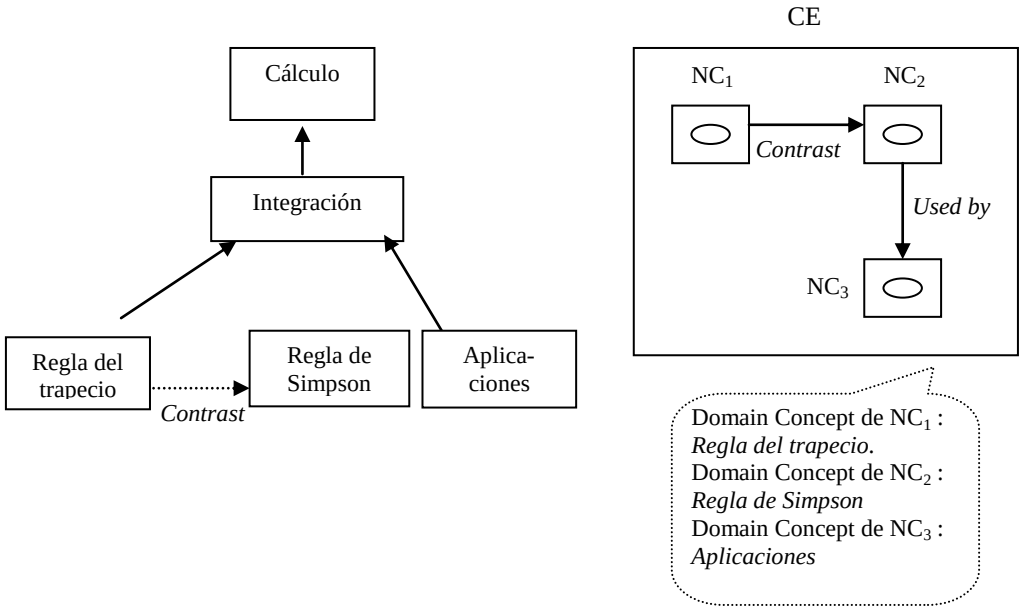



Figura 3.11: Fragmento de modelo de dominio y CE mostrando los *rótulos* entre nodos concepto

## 3.7 Setting para un CE

Con setting se quiere significar la asignación de un valor a un elemento. En nuestro modelo consiste en que el creador de un CEC puede asignar (predeterminar) valores a las entradas de los CEM's $c_j$. Es decir, el conjunto de *Setting* = { $I_k$ / $I_k \in$ *Input*-of-$c_j$ y a $I_k$ se le asigna valor}, o sea, es el conjunto de entradas de un CE a las cuales el autor les asigna valores predeterminados dentro de un CEC. El usuario de este CE sólo podrá entonces asignarle valores al resto de las entradas (o sea a las que no se les asignó valor). La figura 3.12 muestra esta característica:

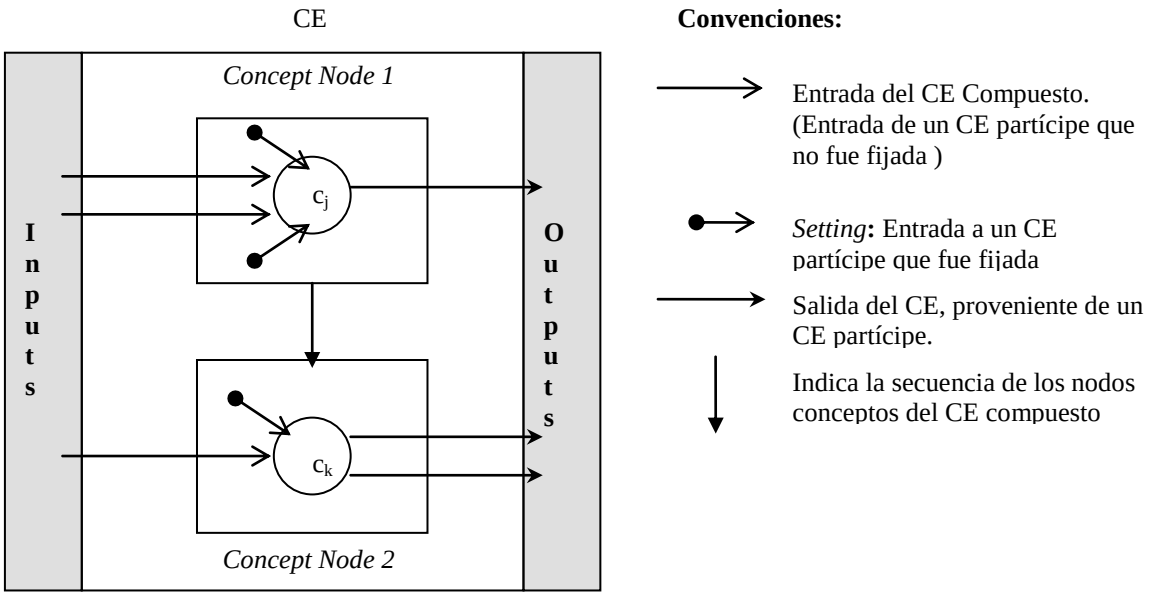



Figura 3.12: Visualización de entradas y salidas en un CEC

## 3.8 Operadores de creación de CE's

*Create Component (Metadata, $R_{subject}$ ,[Input], [Outputs],[$R_{prerequisite}$], [compostion] )*

Los CEP's deben cumplir con todas las características expresadas en la sección 3.2. Input, Output y $R_{prerequisite}$ son opcionales. Metadata y $R_{subject}$ son obligatorios.

Los operadores de composición permiten construir un CEC a partir de otros CE's (primitivos o compuestos) existentes en la base de datos. Estos operadores permiten definir un CE patrón, el cual es un grafo acíclico conectado cuyos nodos contienen conceptos del dominio y requerimientos en el momento de su ejecución (prerrequisitos, CE sugerido etc.; véase sección 3.6.1).
Este grafo de composición permite que cada autor defina su propio modelo de dominio específico (el curso que crea). Este modelo específico provee facilidades a los estudiantes para visualizar la estructura del curso (coherencia global), estrategias de navegación personalizadas y conocimiento acerca de las relaciones entre los nodos concepto. Los roles dentro de los CE's proveen facilidades para visualizar la estructura de las lecciones (coherencia local).

En la fase de ejecución los operadores de instanciación de CE's [2] se encargarán de tomar un CE patrón y permitir el remplazo (por parte de los estudiantes) de los CE's sugeridos intercambiables, así como satisfacer sus preferencias y objetivos pedagógicos.

**Nota:** Los conceptos de curso y lección dentro del contexto de nuestro trabajo se definen en la sección 3.9, el aspecto de remplazo en la sección 3.14 y un CE patrón en la sección 3.6.1.

### 3.8.1 Operador Sequence ($CE_1$, $CE_2$, … , $CE_n$)

El operador Sequence crea un CEC cuyo grafo de composición está dado por la trayectoria (ver sección 3.6) de nodos concepto $NC_1$, $NC_2$,..., $NC_n$ donde el arco $NC_i$ hacia $NC_{i+1}$ existe para $1 <= i <= (n-1)$.

Debe aclararse que cada $CE_i$ conlleva toda la información del nodo concepto del cual hace parte, es decir, *estrictamente* hablando los argumentos del operador *Sequence* realmente son nodos conceptos, sin embargo en el operador se quiere expresar más bien que los elementos relevantes son los CEM's $CE_1$, $CE_2$, … , $CE_n$ ; donde el $CE_i$ no es necesariamente diferente del $CE_j$, $1<= i <= n$, $1 <= i <=n$. Si $CE_i = CE_j$ muy posiblemente conllevarán diferente información para el nodo concepto del cual harán parte. Véase figura 3.15.

#### 3.8.1.1 Restricción de secuencia

Si A es el concepto de dominio desarrollado por el nodo concepto j, entonces A $\notin$ prerrequisitos del nodo concepto k, si j>k, es decir, en una secuencia de la composición, un nodo concepto no puede tener como prerrequisito un concepto del dominio que aparece desarrollado en un nodo concepto posterior.

#### 3.8.1.2 Elementos del CEC creado

a) Input (entradas) de $c_c$ = { p / $\exists$*concept-node*$_i \in$ *concept-nodes* de $c_c$ y $\exists$component-$c_k$ dentro de *concept-node*$_i$ ( p $\in$ {Input de $c_k$ - setting de $c_k$} )}. Esto significa que los estudiantes sólo podrán asignar valores a las entradas que no se les hizo setting. Ver figura 3.12.

[2] Los cuales no hacen parte de este trabajo

b) *Output* (salidas) de $c_c$ = {p / ∃*concept-node*$_i$ ∈ *concept-nodes* of $c_c$ y ∃component-$c_k$ dentro de *concept-node*$_i$ (p ∈ Output of $c_k$ ) }. Ver figura 3.12.

a) Metadata de $c_c$ = Los metadatos conservan la definición dada en la sección 3.2. Sin embargo sólo el CE final resultante de una composición posee metadatos en sí, es decir, todos los CE's "intermedios" que aparecen en una composición no requieren metadatos. Los CE's intermedios son aquellos CE's que se forman para dar lugar a la creación del CE final, por ejemplo supóngase que se crea el siguiente CEC figura 3.13:

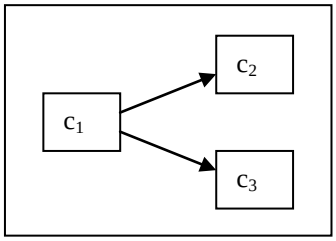



Figura 3.13: Un CEC $c_k$

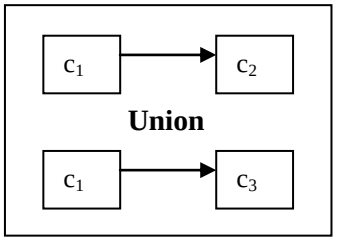



Figura 3.14: Representación de $c_k$ en términos de operadores de composición: *Sequence* y *Union*

Esto da lugar a la creación de un CE interno como se muestra en la figura 3.14.
En términos de operadores de composición se llega a la siguiente expresión:

*Union( Sequence($c_1$, $c_2$), Sequence($c_1$,$c_3$)).*

**Nota:** El operador *Union* y la representación del grafo de un CE en términos de los operadores de composición se explica en la sección 3.8.2.

Nótese que se crean 2 CE's intermedios, producto de los dos operadores *Sequence*. Estos dos CE no requieren metadatos, el único que lo requiere es el CE resultante de la suma de estos dos.

b) $R_{subject}$ de $c_c$ = El conjunto $R_{subject}$ fue definido en la sección 3.3.3.1. Para una secuencia de composición existen 4 casos:

- Si todos los nodos concepto de $c_c$ usan el mismo concepto del dominio**, el sistema lo calcula automáticamente**. Ejemplo, sea el $c_4$ = *Sequence ($c_1$, $c_2$, $c_3$)* donde los tres nodos conceptos

correspondientes usan el concepto A, el nodo concepto de $c_1$ tiene rol *motivate* el de $c_2$ tiene rol *define* y el de $c_3$ *evaluate* por lo tanto el sistema construirá el siguiente conjunto $R_{subject}$ = [{motivate, define, evaluate}, concept-A].

- Si todos los nodos concepto sólo tienen el rol *evaluate*, el rol del nuevo será *evaluate*, pero el concepto lo deberá colocar el autor (a menos que todos fueran iguales, ver caso anterior)

- Si el CE que se forma a raíz del operador *Sequence* hace parte (es decir es el argumento) de otro operador, como por ejemplo en *Union( Sequence($c_1$, $c_2$), Sequence($c_1$,$c_3$))* el conjunto $R_{subject}$ no se define. (Sólo se define para el CEC final, no para los CEC's intermedios)

- Si los nodos conceptos usan diferentes conceptos del dominio, entonces el autor define el conjunto $R_{subject}$

e) $R_{prerequisite}$ *de* $c_c$ = El conjunto $R_{prerequisite}$ fue definido en la sección 3.3.3.2. Recuérdese que cada nodo concepto "hereda" el conjunto de prerrequisitos $R_{prerequisite}$ de su CE sugerido. Sea $s_p$ el conjunto de todos los conceptos prerrequisitos de todos los nodos concepto, sea $s_j$ el conjunto de todos los conceptos de dominio desarrollados por todos los nodos concepto y $s_k = s_p - s_j$ (**ecuación de prerrequisitos).** Es decir los prerrequisitos de $c_c$ **no** son la "suma" de los prerrequisitos de todos sus nodos concepto, los prerrequisitos **son la suma de los prerrequisitos de los todos los nodos concepto *menos* los conceptos *desarrollados* en todos los nodos concepto**.
Para cada elemento $c_n \in s_k$ se define una relación de prerrequisito *[$c_n$, level]*, donde level es tomado a partir del nodo concepto que posee a $c_n$ como prerrequisito. En caso de que varios nodos concepto posean a $c_n$ como prerrequisito se escoge el *level* de mayor valor.

Ejemplo: Sea $c_4$ = *Sequence ($c_1$, $c_2$, $c_3$)* donde los conceptos prerrequisitos para el nodo concepto de $c_1$ son = *{[a, medium], [b, medium]}*, los del nodo concepto de $c_2$ son = *{[b, high], [d, medium]}* y los del nodo concepto de $c_3$ son = *{*[*e*, high], [f, medium]}; además, $c_1$, $c_2$ y $c_3$ desarrollan los conceptos d, e y m respectivamente. En consecuencia:

- $s_p$ = { a, b, d, e, f}; $s_j$ = { d, e, m} y $s_k$ = {a, b, f}.
- $R_{prerequisite}$ de $c_4$ = { *[a,medium], [b,high], [f,medium]* } .

- Nótese que hay 2 nodos concepto que tienen al concepto *b* en su conjunto de prerrequisitos y con diferente level: *[b, medium]* y *[b, high]*; por lo tanto se escoge el mayor de los dos.

*f) Composition* = Es la secuencia de nodos concepto: *concept-node-1 AND concept-node-2 AND … AND concept-node-n.* Donde el orden indica la secuencia a seguir (tabla de contenido).
La figura 3.15 muestra una configuración lograda por medio del operador *Sequence*:

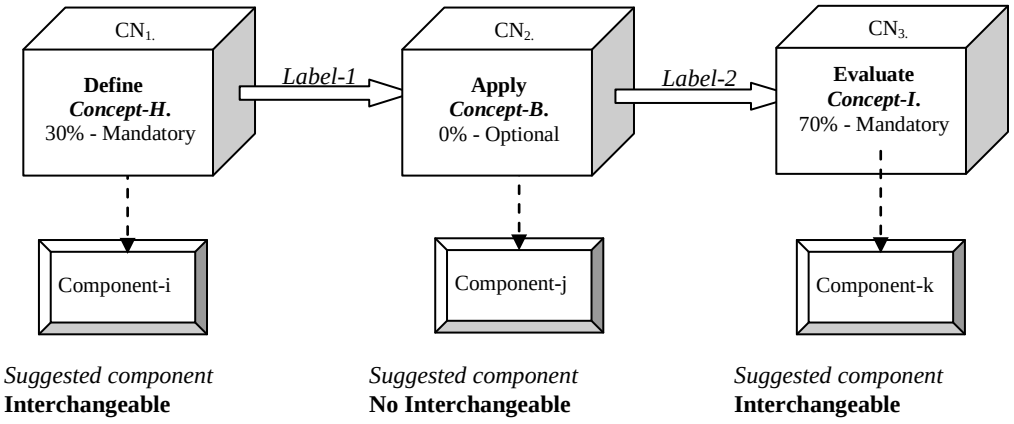



Figure 3.15: Composición para el operador *Sequence*

### 3.8.2 Operador Union ($CEC_1$, $CEC_2$, … , $CEC_n$)

El operador *Union* construye un nuevo CEC. Este es un grafo dirigido acíclico conectado que representa un conjunto de caminos (secuencias). Los usuarios pueden escoger el orden para atravesar los diferentes caminos durante el proceso de aprendizaje. Cada camino contiene una secuencia de temas sugerida de acuerdo a razones pedagógicas.

Sean los CEC's $c_i$ y $c_j$ cuya composición está representada por los grafos $G_i = < V_i, E_i >$ y $G_j = < V_j , E_j >$ , donde $V_i$,$V_j$ son conjuntos de vértices y Ei, Ej son conjuntos de arcos.
La operación $c_c = c_i$ *Union* $c_j$ ; consiste en $G_i \cup G_j$, es decir, el grafo $G_c = < V_i \cup V_j , E_i \cup E_j >$ donde el símbolo U representa la unión de conjuntos. Nótese que esta definición puede ser generalizada para la unión de *n* CEC's.

En general el operador Union unirá un conjunto de CEC's donde cada uno es creado mediante el operador *Sequence* como en el siguiente ejemplo. Supóngase los CEP's $c_1$, $c_2$ y $c_3$. Es posible crear entonces un CEC así: *Union (Sequence ($c_1$, $c_2$), Sequence ($c_1$, $c_3$)).*

El CEC resultante se muestra en la figura 3.16:

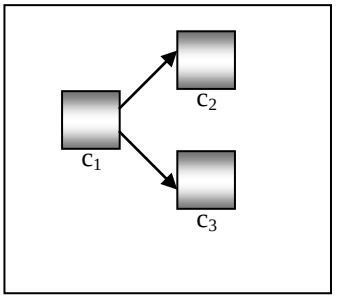



Figura 3.16: CEC resultante de: *Union (Sequence (c1, c2), Sequence (c1, c3))*

#### 3.8.2.1 Caminos en el grafo $G_c$

La secuencia de nodos concepto < $NC_0$, $NC_1$, ..., $NC_n$ > es un camino (*path*) $p_i$ de longitud *n* en el grafo $G_c$ si: Existe el arco $NC_i \rightarrow NC_{i+1}$ 1<i<=n; al $NC_0$ no llegan arcos y del $NC_n$ no salen.

#### 3.8.2.2 El aspecto de compatibilidad

Para cada secuencia presente en el grafo se debe cumplir la restricción de la sección 3.8.1.1 y no se permiten grafos cíclicos ni inconexos.

#### 3.8.2.3 Elementos del CEC creado

a) Input (entradas) de cc $\subseteq$ { p / $\exists$concept-nodei $\in$ concept-nodes de cc y $\exists$componente ck en el concept-nodei ( p $\in$ {Input de ck - setting de ck} )}.
Esto significa que los usuarios sólo podrán asignar valores a las entradas que no se les hizo *setting*. Además, en tiempo de ejecución, el usuario podría no usar todos las entradas en caso de que no explore todos los posibles caminos.

b) Metadata de cc = Los metadatos conservan la definición dada en la sección 3.2 y se presentan las observaciones vistas en la sección 3.8.1.2.

c) Rsubject y Rprerequisite de cc = Se conserva la definición dada en la sección 3.3.3.1. Se presentan los mismos casos explicados en la sección 3.8.1.2.
e) Composition = Gc < Vi U Vj , Ei U Ej>= <Vc ,Ec>

### 3.9 Construcción de CE típicos

Mediante los operadores de creación y composición vistos se pueden crear CE's típicos( recomendables), de diferente nivel de granularidad así (ver figura 3.17):

*a) Tópico*: Un tópico es un CEP que juega un solo rol (excepto el rol *evaluate*) con respecto a un único concepto del dominio asociado. Por ejemplo el CE $c_1$ que define el concepto "Data Model": *[{define}, DATA-MODEL]*.

b) *Ejercicio*: Un ejercicio es un CEP que juega solamente el rol evaluate con respecto a un único concepto del dominio asociado. Por ejemplo: *[{evaluate}, DATA-MODEL]*.

*c) Workshop* (Taller): Un workshop es un conjunto de ejercicios. Por ejemplo el CE $c_5$ = Sequence ($c_2$,$c_3$,$c_4$), donde cada CE parámetro del operador *Sequence* es un ejercicio, es decir juega el rol *evaluate* respecto a un concepto del dominio asociado. Pueden ser ejercicios de diferentes conceptos (workshop *heterogéneo*) o todos del mismo concepto (workshop homogéneo). El nuevo CE juega el rol evaluate para su conjunto de relaciones Rsubject.

d) *Lección:* Un conjunto de tópicos y que probablemente incluirá un workshop, pero todos ellos asociados a un *mismo* concepto del dominio, pero con diferentes roles. Sea CE $c_7$ = *Sequence* ($c_6$, $c_1$, $c_5$); donde el concepto del dominio para todos los nodos conceptos de $c_7$ es "Data Model" y sus CE's $c_6$, $c_1$ y $c_5$ juegan los roles *motivate*, *define* y *evaluate* respectivamente. Por lo tanto el conjunto $R_{subject}$ de $c_7$ está dado por *[{motivate, define, evaluate}, DATA-MODEL]*.

e) Curso: Un curso es un conjunto de lecciones. El autor define su Rsubject.

Los anteriores "tipos" de CE's no son los únicos que se pueden concebir pero si se espera que sean una guía para los autores, es decir, es posible configurar un CEC tan heterogéneo como se desee e incluso es posible tener un CEP que juegue diferentes roles frente a diferentes conceptos, sin

embargo, este tipo de CE's, no son deseables ya que pueden impedir la reutilización, se prefieren CE's de granularidad fina como los que se han planteado aquí.

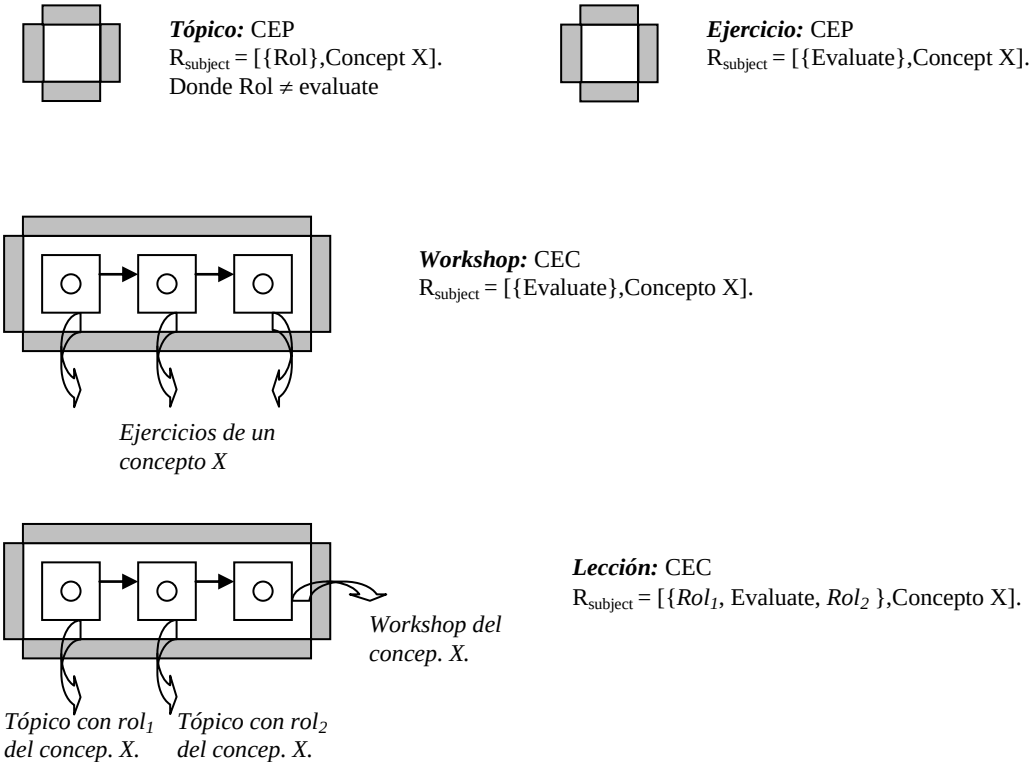



Figura 3.17: CE's típicos

## 3.10 Elementos de lógica descriptiva para la especificación de los CE's

La lógica descriptiva (LD) [Baader 03] es un formalismo para representar conocimiento. Para hacerlo más intuitivo se acostumbra una representación en forma de red (o grafo). Los elementos de la red son nodos y arcos. Típicamente los nodos son usados para caracterizar *conceptos*, es decir clases de individuos, y los arcos sirven para caracterizar relaciones entre conceptos. En algunos casos algunas relaciones más complejas son representadas ellas mismas como nodos, estos son cuidadosamente diferenciados de aquellos que representan conceptos.

Adicionalmente, los conceptos pueden tener *propiedades* o *atributos*, los cuales son normalmente "pegados" a los correspondientes nodos. Consideremos el ejemplo que se muestra en la figura 3.18, la cual representa conocimiento sobre personas, padres, hijos, etc. La estructura de esta figura se conoce también como una *terminología* y de hecho va a representar *generalización / especialización*

de los conceptos involucrados. Por ejemplo la relación entre madre y progenitor significa que "las madres son progenitores", esto es llamado una relación IS-A (ES UN).

Las relaciones IS-A definen una jerarquía de conceptos y proporcionan la base para la "herencia de propiedades": cuando un concepto es más especializado que otro, entonces hereda las propiedades del más general.

Una característica de la LD es su capacidad para representar otros tipos de relaciones que se pueden establecer entre conceptos, por ejemplo en la figura 3.18, el concepto *padre* posee una propiedad que es usualmente llamada "rol", expresada por una relación desde el concepto hasta un nodo que representa el rol y que se ha llamado *TieneHijo*. El rol tiene asociado una restricción de valor (value restriction), denotada por el rótulo *v/r*, la cual expresa que tipo de objetos (conceptos) pueden "llenar" (conocidos como los *fillers*) ese rol. Adicionalmente el nodo tiene una *restricción de número* (number restriction) expresada como *(li,ls)*, donde el primer número *li* indica el límite inferior, en el caso del ejemplo, el número de hijos 1 (es lo mínimo para ser considerado un progenitor) y el segundo número *ls* es el límite superior, en el ejemplo NIL el cual significa infinito. Por lo tanto la representación del concepto progenitor puede ser leída como:

"Un progenitor ES UNA persona que tiene al menos un hijo y todos sus hijos(as) son personas". Nótese que el concepto *persona* es el *filler* del rol *TieneHijo*. Las relaciones de este tipo son a su vez heredadas por conceptos más específicos, por ejemplo el concepto madre hereda el rol TieneHijo.

Ente los objetivos de la LD está el de proporcionar operadores (constructores) que permitan construir otros conceptos y extraer relaciones implícitas a partir de una red (inferencia).

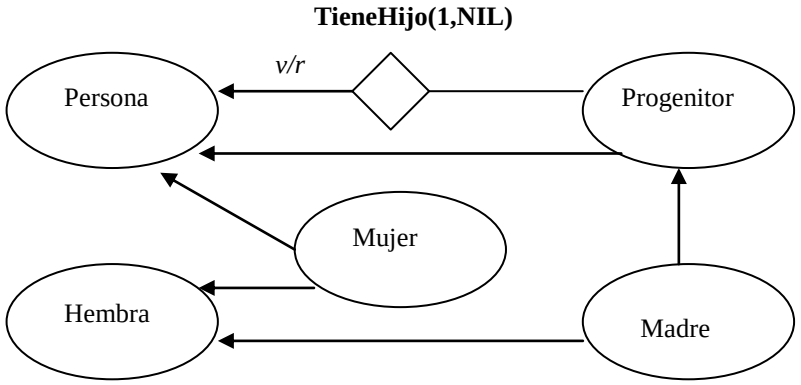



Figura 3.18: Un ejemplo de red de conocimiento para LD

La aplicación para CE's se muestra en la sección 3.10.2.

### 3.10.1 Representación de conceptos y algunos constructores

Los conceptos son conjuntos de individuos y los roles son parejas de individuos [Baader 03]. Los conceptos son interpretados como subconjuntos de un dominio (universo).
Sean los conceptos C y D. Se definen operadores entre los conceptos, por ejemplo:

- La conjunción: C ⊓ D; representa el conjunto de individuos obtenidos de intersectar los conjuntos de individuos C y D respectivamente.
- La disyunción C⊔ D, representa el conjunto de individuos que se obtienen de unir los conjunto de individuos C y D.
- La negación ¬C representa a todos los individuos que no son el concepto C.

Finalmente se presentan dos constructores que involucran roles, los cuantificadores ∀ universal y ∃ existencial:

- ∀R.C, donde R es un rol, representa el conjunto de individuos cuyos roles R son todas llenas por el concepto C, por ejemplo, *∀TieneHijo.hembra*, representa a los individuos que sólo tienen hijas (ya que todas sus roles TieneHijo son llenas por el concepto hembra).
- ∃R.C representa el conjunto de individuos que poseen al menos un rol R que es lleno por un concepto C, por ejemplo, *∃TieneHijo.macho*, representa el conjunto de individuos que tienen al menos un hijo macho.

Existen otros constructores que hacen **más expresiva** la LD, como los de restricción numérica para roles, pero no se utilizarán en el contexto de este trabajo (ver sección 6)
El uso de estos operadores da lugar a nuevos conceptos, la conformación de estos se denomina la expansión del concepto. Los conceptos que no son formados por otros se denominan *atómicos*.
Finalmente, se utilizan los símbolos ⊥ (*botton concept*) para indicar el concepto nulo (o inconsistente) y ⊤ (*top concept*) para indicar el concepto universal.

Veamos otros ejemplos: sean los conceptos macho, hembra, persona y mujer. Por medio de los constructores presentados se puede expresar:
- "Personas que no son hembras": *persona* ⊓ ¬*hembra*
- "Individuos que son hembras o machos": *hembra* ⊔ *macho*
- "Individuos que carecen de hijos": *∀TieneHijo.* ⊥

Hasta este punto la LD puede verse como la típica lógica de primer orden, sin embargo y como se expresa en [Baader 03], los operadores de restricción numérica sobre roles y algunas operaciones como la inferencia, lograda a través del operador de subsumption el cual será utilizado (ver sección 3.15), dan su razón de ser a la LD.

### 3.10.2 Especificación de los CE's en términos de LD

Cada CE tiene una especificación basada en LD. Esto permitirá desarrollar un operador de búsqueda para facilitar el proceso de adaptación. (ver sección 3.15)

La figura 3.19 muestra una representación usada para describir mediante LD el contenido de un CE. Esta representa las relaciones del CE con los conceptos del dominio. Como se ha visto los CE's se relacionan con los conceptos del dominio a través de roles.
Por ejemplo un CE define 0 ó muchos conceptos del dominio. La figura 3.19 muestra que cada CE puede tener desde 0 hasta muchas relaciones (roles) con los conceptos del dominio, los cuales están organizados en una jerarquía. Esta representación no es más que un modelo entidad relación con la notación que es usada en el contexto de la lógica descriptiva. [Baader 03]
Debe quedar claro que, aunque en algunos casos el conjunto $R_{subject}$ de un CE y la especificación de un CE mediante LD coinciden, esta última tiende más bien especificar los conceptos y roles asociados a un CE, implicados a través de sus CEM's directos.
Nota: La forma como se modeló un CE desde el punto de vista de LD hace muy adecuado su manejo para trabajar ciertos operadores y algoritmos de este campo. Desde el punto de vista del modelo Entidad Relación el modelo es un poco incómodo, ya que por cada rol existente se debe crear una relación especifica. Más elegante y compacta hubiese sido un diseño en el cual se involucre una relación llamada *has_relation* y en la cual se especifique que tipo de relación y con cual concepto; pero para el trabajo con LD no era adecuado este esquema debido a los constructores con los que se trabajó (ver sección 3.10).

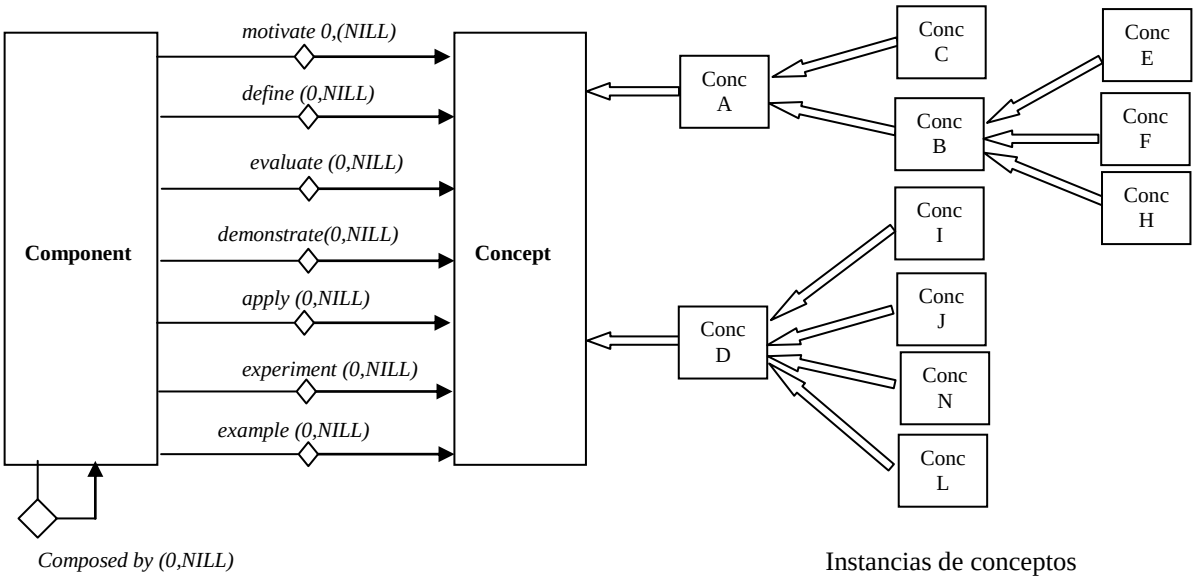



Figura 3.19: Especificación de un CE usando LD

### 3.10.2.1 Parte declarativa de un tópico

Supóngase un CEP $c_1$ que desarrolla el concepto C mediante el rol define. Su conjunto $R_{subject}$ es *[{define}, concept-C]*. Mediante lógica descriptiva se especificará como:

$$c_1 = \exists \text{define.concept-C} \sqcap \forall \text{composed_by.}\bot$$

Esto significa que posee una relación *define* cuyo *filler* es concept-C y que no posee relaciones de tipo *composed_by*, es decir, se trata de un CEP.

### 3.10.2.2 Parte declarativa de un ejercicio y de un workshop

Supóngase el CEP $c_6$ que contiene un ejercicio del *concepto-C.* Por lo tanto su $R_{subject}$ es *[ {evaluate}, concept-C]*. Mediante lógica descriptiva se especificará como:

$$c_6 = \exists \text{evaluate.concept-C} \sqcap \forall \text{composed_by.}\bot$$

Ahora supóngase un CEC $c_{12}$ (un taller) = *Sequence* ($c_6$, $c_7$, $c_8$) donde $c_6$, $c_7$, $c_8$ son ejercicios del concepto C. Su $R_{subject}$ es *[ {evaluate}, concept-C]*. En lógica descriptiva será:

$$C_{12} = \exists \text{evaluate.concept-C} \sqcap \forall \text{composed_by.}(\exists \text{evaluate.concept-C})$$

Esto significa que $c_{12}$ es un CEC que posee una relación *evaluate* cuyo *filler* es el concepto C y que cada una de sus relaciones composed_by está constituida por un *filler* que posee una relación *evaluate* del concepto C. (es decir el *filler* es un CE).

#### 3.10.2.3 Parte declarativa de una lección

Supóngase que se tiene un CEC mediante *Sequence* ($c_2$, $c_1$, $c_4$, $c_{12}$) donde $c_2$, $c_1$, $c_4$ son tópicos del concepto C y $c_{12}$ es un workshop también de dicho concepto. Su Rsubject está dado por: *[ {motivate, define, example, evaluate}, concept-C]*. Mediante lógica descriptiva se tiene:

$$c_{14} \subseteq \exists \text{motivation.concept-C} \sqcap \exists \text{define.concept-C} \sqcap \exists \text{example.concept-C} \sqcap$$

Nótese que esta definición podría traer CEP's de granularidad alta (indeseables), pero en general se espera que se obtengan CEC's.

#### 3.10.2.4 Parte declarativa de un curso

Sea el CE $c_{16}$, el cual consiste en la composición *Sequence* ($c_{14}$, $c_{15}$) y $R_{subject}$ es *[{ define, evaluate },concept-A]*. Supóngase que $c_{15}$ es un CE cuya declaración es:

$$\text{c15} \subseteq \exists \text{define.concept-B} \sqcap \exists \text{evaluate.concept-B}$$

El $R_{subject}$ de $c_{16}$ no incluye al concepto B (Recuérdese que para los cursos el creador define el $R_{subject}$). Por lo tanto $c_{16}$, puede ser expresado como:

$$\text{c16} \subseteq \exists \text{motivation.concept-C} \sqcap \exists \text{define.concept-C} \sqcap \exists \text{example.concept-C} \sqcap \exists \text{evaluate.concept-C} \sqcap \exists \text{example.concept-B} \sqcap \exists \text{evaluate.concept-B}$$

ó:

$$\text{c16} \subseteq \exists \text{define.concept-A} \sqcap \exists \text{evaluate.concept-A}$$

## 3.11 Lenguaje de Manipulación de CE's

### 3.11.1 Introducción

Así como la unidad de trabajo en el lenguaje SQL es la relación y dicho lenguaje procura ser cerrado [3] (es decir todas las operaciones trabajan sobre y retornan relaciones), la unidad de trabajo en nuestro sistema es el CE. Se procurará mantener un esquema de cerradura para la mayoría de los operadores (esto facilita por ejemplo que la salida de un operador sea la entrada de otro para lograr anidamientos). Sin embargo algunos operadores dada su naturaleza no serán cerrados, por ejemplo Show y GetPaths (ver secciones 3.12 y 3.13). El lenguaje se espera sea lo suficientemente expresivo y a la vez sencillo de tal forma que pueda ser aprendido por inexpertos en programación. Para lograr este último propósito se facilitará el trabajo por medio de un entorno visual, de hecho, tal y como se expresa en [Goguen 86] un usuario final no necesariamente tiene que ver las formas sintácticas de un lenguaje propuesto para interactuar con él, ya que esto se puede lograr a través de interfaces.

Se presenta para cada operador la sintaxis básica, **la sintaxis completa puede verse en el apéndice 3 escrita en BNF.**

### 3.11.2 Operador de Selección

*Get Components [option] where Condition [With Recursive (n) ]*

El operador de Selección de CE's consiste de las palabras claves *Get Components* + un *predicado* (el cual es una condición + algunas opciones), el cual retorna siempre identificadores de CE's, en este sentido el operador se comporta análogamente a los motores de búsqueda (Google, Alltheweb, etc.), ya que estos reciben palabras clave y retornan URL's, en nuestro caso se reciben predicados y se retornan Coid's. Las características del predicado se describen a continuación. Algunos de sus elementos son similares a lenguajes semiestructurados como LOREL (véase sección 2.3.4) y se ha limitado a sus características más esenciales y útiles.
**Nota:** Para acceder y consultar las partes de un CE véase el operador Show, sección 3.12.

Observaciones

- El operador retorna un conjunto de Coids en un documento XML.
- El operador puede confrontarse con el lenguaje similar a SQL creado para AHA (véase sección 2.2.3.3.2 y [Wu 01]) y con PQL (véase sección 4.2.1.2 y [Klein 02]).
- Las opciones (option) que pueden aparecer luego de la condición (o solas) sirven para discriminar el tipo de CE's que cumplen la condición:

  -*All*: Retorna los Coids de todos los CE's. Es la opción por default.

  -*Specific*: Retorna sólo Coids de CE's específicos.

  -*Pattern*: Retorna sólo Coids de CE's abstractos (patrón)

  -*Primitive*: Retorna sólo Coids de CEP's

  -*Composite*: Retorna sólo Coids de CEC's
- La condición es muy simple y se compone de cualquier *expresión de camino* válida existente en la estructura de cada CE, o sea, permite consultar las secciones de metadatos, inputs, outputs etc.
- Si no hay condición se devuelven todos los CE's solicitados.
- La condición puede contener operadores booleanos (And, Or, Not)
- Por defecto se busca en todo el repositorio de CE's, aquellos que cumplan la condición estipulada. Si se desea consultar si un CE particular cumple una condición se debe agregar : *...and Coid='cxx',* donde cxx es el Coid del CE en cuestión.
- En la condición se permite el uso de 2 comodines que actúan similarmente como en las expresiones regulares, estos son el * y el _ , donde el _ significa un elemento (categorías o subcategorías) y el * significa cero o muchos elementos.

  Por ejemplo, sea la consulta, obtener los Coid's de los CE's cuyo título es Energía Solar:

  *a) Get Components where general.title='Energía Solar'*

  Su condición podría ser remplazada por las siguientes:

  *b) _.title='Energía Solar'* *c) *.title='Energía Solar'*

  Sin embargo estrictamente hablando las 3 expresiones no son equivalentes, debido a lo siguiente: (véase la estructura de metadatos sección 3.5)

  -La consulta a) indaga en los metadatos de LOM en su categoría *general* en su atributo *title*.

[3] SQL *no* es, estrictamente hablando, cerrado relacionalmente, ya que algunas sentencias producen tuplas repetidas por lo tanto el resultado no es una relación.

-La consulta b) busca caminos que comiencen por cualquier categoría y que posean un atributo llamado *title*. Por ejemplo suponga que en el ME se ha escrito lo siguiente:

--*Zona de metadatos de LOM:*
**General**
    **Title='Manchas Solares'**
    **...**
--*Fin de los metadatos de LOM*
--*Zona de metadatos extensible:*
**Extensible**
    **Title='Energía Solar'**
    **...**
--*Fin de los metadatos extensibles.*

La consulta b) encontrará este CE a diferencia de la consulta a).
La consulta c) es "más potente" porque el significado del asterisco es buscar cualquier camino (no importa que tan anidado esté) que conduzca a un atributo llamado title cuyo valor sea 'Energía Solar'.
Por lo tanto si se desea consultar sin ambigüedad en los metadatos de LOM se debe cualificar la consulta con la categoría apropiada (*general* por ejemplo). Si se desea consultar en el ME se debe comenzar usando el nombre de categoría *extensible.* Si es indiferente se puede consultar con _ o con * dependiendo de la necesidad.

En el ejemplo anterior, es claro que un atributo *title* ubicado directamente en la raíz del ME no tiene mucho sentido que aparezca allí, sólo se ha presentado para ejemplificar. Lo que sí es viable sería que existiese por ejemplo dentro de dicha zona una categoría llamada *proyecto* y dentro de ésta podría existir un atributo llamado *title*, que probablemente represente el título de algún proyecto en el que estaría enmarcado un CE por ejemplo:

--*Zona de metadatos de LOM:*
**General**
    **Title='Manchas Solares'**
    **...**
--*Fin de los metadatos de LOM*
--*Zona de metadatos extensible:*
**Extensible**
    **Proyecto**
        **Title='Energía Solar'**
        **...**
--*Fin de los metadatos extensibles.*

En este caso la consulta c) encontraría este CE pero no la a) ni la b).

El siguiente ejemplo muestra que la condición puede ser un poco compleja:

*extensible.proyecto._.fase._.*. title ='Multimedia I'*

La condición busca caminos que comiencen en el ME, que continúen con una subcategoría llamada proyecto, luego debe seguir una subcategoría cualquiera, luego una llamada fase, luego cualquier otra y luego cualquier cantidad de subcategorías para finalmente terminar en un atributo llamado *title* cuyo valor sea '*Multimedia I'*. Un camino válido sería:

*extensible.proyecto.ppal.fase.desarrollo.prueba.title*

- Se podría pensar en otras posibilidades para comodines, los cuales podrían adicionarse en un futuro desarrollo, pero en principio el lenguaje se conservará simple. De hecho el ejemplo anterior muestra que aunque sea poco probable que se llegue a un nivel de documentación tan anidado, el lenguaje y el sistema están preparados para soportarlo. Igualmente el lenguaje se podría hacer más rico en futuras versiones adicionando subconsultas, agrupamientos, algunos tipos de joins entre CE's etc.
- Se dispone de un operador tipo LIKE como en SQL el cual permite buscar subcadenas dentro de un atributo. Por ejemplo la consulta:

  *Get Components where general.title LIKE '%Energía%'*

  Busca CE's que en su título posean la palabra Energía. El símbolo % sustituye cualquier cantidad de caracteres.
- La opción With Recursive indica el nivel de agregación que se está consultando, es decir permite consultar por la información de los CEM's de un CEC.

  Por ejemplo la consulta:

  *Get Components where (general.title='Energía Solar') With Recursive (2)*

  Consulta cuales CE's (compuestos por supuesto) poseen un CEM (al menos uno) cuyo título en LOM sea 'Energía Solar'.

  Sin la opción With Recursive sólo se consulta el nivel propio de cada CE (nivel 1).

  Es posible consultar varios niveles de composición al mismo tiempo, por ejemplo:

  *Get Components where (extensible.proyecto='Codi') and (general.title='Energía')*
  *With Recursive (2)*

Consulta cuales CE's tienen en su ME un atributo llamado *proyecto* llamado *Codi* y que estén compuestos por al menos otro CE cuyo título sea *'Energía'*.
Existe una opción especial:
*(general.title='Derecho Penal') With Recursive (~)*

Significa: Obtener los CE's que estén compuestos (en cualquier nivel) por cualquier CE cuyo título LOM sea *'Derecho Penal'*.
Una utilidad de esta opción se puede ver en el operador Delete.(ver sección 3.11.3)

- Existe *coerción* análogamente al lenguaje LOREL con los operadores de comparación (=, >,< etc.), es decir, una condición del tipo:
  *Educational.Context ='Primary Education'*

  Busca si un CE tiene ***al menos*** un contexto educativo *'Primary Education'.*
- Es posible consultar las partes **semiestructuradas** en dos "estilos":

  *a) inputs.language='English'* → Consulta los CE's que posean una entrada llamada *Language* cuyo valor sea *'English'*.

  *b) inputs.label='Language'* → Consulta los CE's que tengan una entrada llamada *'Language'*.

  **Nota:** Véase una comparación de esta consulta con el lenguaje PQL (sección 4.2.1.2).

  Igualmente el ME, es susceptible de ser consultado de estas dos formas. Por ejemplo:
  *Get Components where extensible.proyecto.fase=12 and extensible.proyecto.name='P3'*

  Consulta los CE's que tienen en su ME, un atributo llamado *fase* (dentro de una categoría *Proyecto*) cuyo valor sea *12* y que el proyecto se llame 'P3', en cambio:
  *Get Components where extensible.proyecto.attribute='fase' and extensible.proyecto.attribute='name'*

  Consulta los CE's que tienen en su ME una categoría llamada proyecto y dentro de esta que existan atributos llamados *Fase* y *Name*.

**Nota:** La palabra *attribute* es reservada, es decir, no pueden existir atributos en el ME cuyo nombre sea attribute.

### 3.11.3 Operador de Borrado

*Delete {Coid} | (Consulta)*

- El operador recibe el o los Coids de los CE's que van a ser eliminados del sistema. Ejemplos:
  -Eliminar el CE cuyo Coid es c8: *Delete c8*
  -Eliminar los CE's cuyo Coid es $c_1$,$c_3$ ó $c_4$: *Delete c1,c3,c4*
  -Eliminar los CE's cuyo título sea "Energía Solar":
  *Delete (Get Components where general.title = 'Energía Solar')*
- Si uno de los CE's que va a ser eliminado es partícipe de un CEC, la operación no se permite, (se restringe). Para saber si un CE que va a ser eliminado es usado por otros el sistema usa la cláusula *With Recursive* del operador de Selección. Ejemplo: Supóngase que se desea saber si el CE con Coid='xxx' es partícipe de otro CE:
  *Get Components where (Coid='xxx') With Recursive (~)*

  Esta consulta devolverá los Coid's de todos los CEC's que usen al CE 'xxx' en cualquier nivel de composición. Si el resultado es vacío el CE 'xxx' podrá ser eliminado.
  Similarmente una instrucción del tipo:
  *Delete ( (Get Components where Coid='xxx') With Recursive (~) )*

  Intentará eliminar todos los CEC que usen al CE 'xxx', pero si alguno de estos es usado por otros CE's la operación fracasará por completo. (en este caso la operación se comporta atómicamente, se eliminan todos si es posible o no se elimina ninguno)

- No se ha considerado una opción de eliminación "en cascada" debido a que puede desencadenar un borrado masivo de CE's no deseable. Por ejemplo la instrucción:
  *Delete cx* --Supóngase en cascada

  Eliminaría del sistema al CE $c_x$ y adicionalmente a todos los CE's que usen a $c_x$ y a su vez a todos los que usen a estos etc. Similarmente una instrucción del tipo:
  *Delete ( Get Components )* --Supóngase en cascada

Eliminaría todos los CE's *del usuario*; si esta es en realidad la necesidad (**aunque en este sistema el borrado de CE's no se espera que sea una operación frecuente, contrario a la adición**) el usuario tendrá que proceder a eliminar primero a los CE's que no son partícipes de otros y así sucesivamente.

### 3.11.4 Operador de Actualización/Modificación

Existen dos opciones:
*Update Coid add | change | delete Elemento [: Value]*
*Update Coid change Elemento (New Value, Old Value)*

Elemento puede ser: *metadata_path, subject* o *prerequisite.*
En algunos casos las operaciones add y delete requieren *Value*.(ver ejemplos en esta misma sección).
La estructura de *Value* depende de elemento, véase el siguiente cuadro:

| **Elemento** | **Estructura de Value** |
|---|---|
| Metadata_Path | Un valor escalar string o numérico* |
| Subject | (rol, id de nodo de árbol de dominio) |
| Prerequisite | Id de nodo de árbol de dominio y nivel de dificultad |

* En algunos casos el Value es compuesto, es decir:
value: ($label_1$: $value_1$, $label_2$: $value_2$, ..., $label_n$: $value_n$)
** Ver además sección 3.11.4.2 para el caso de inputs y outputs.

#### 3.11.4.1 Add

Permite adicionar un nuevo elemento de información a un CE.
Para los elementos LOM, esta opción sólo es válida para aquellos multivaluados, es decir, que permiten varios valores, por ejemplo *general.language*, *educational.context* etc. (véase apéndice 1 para una lista completa). Ejemplo:
*Update c128 add Educational.Context: 'Secondary Education'*

Se agrega un contexto educativo al CE $c_{128}$.
Se debe tener presente que no pueden haber contextos educativos repetidos en un CE.

En algunos casos se debe añadir un elemento multivaluado que se compone de varios ítems, por ejemplo, cuando se va a añadir una contribución (*contribute*), se debe especificar su *role*, *entity* y *date*, en este caso *value* se compone de parejas *label: value* así:
*Update c128 add life_cycle.contribute: ( role: 'Author', entity:'Pedro Pérez', date:'17-Jul-98')*

Restricción: Los elementos multivaluados compuestos deben ingresarse con todos sus elementos.La instrucción puede ser un poco tediosa para un usuario no programador, sin embargo mediante una interfaz se espera se facilite su uso.

-Agregando un elemento al ME
Supóngase que se desea agregar la siguiente información: un CE se encuentra enmarcado en un proyecto denominado "Omega" y pertenece a la fase 12 de dicho proyecto. Se deben realizar tres sentencias así:
*Update c125 add extensible.proyecto*
*Update c125 add extensible.proyecto.nombre : 'Omega'*
*Update c125 add extensible.proyecto.fase :12*

Si se emitiese primero la sentencia 2 ó la 3 estas fracasarían porque la categoría proyecto no existe. (suponiendo que en verdad no existiese). Por ello se debe agregar primero el padre de los atributos Nombre y Fase, o sea, Proyecto.
Recuérdese que no se permite la existencia de 2 caminos iguales en el ME, a diferencia de LOM donde si se permiten elementos multivaluados.

Aunque desde el punto de vista de documentación sería válido se tomó la decisión de no permitir la adición ni la eliminación de una entrada ni de una salida. Por ejemplo la adición de una entrada a un CE es un trabajo que debe sincronizarse con la correspondiente actualización del recurso educativo que contiene la entrada en cuestión para no crear la falsa impresión de que con la simple documentación se ha agregado *realmente* una entrada al CE. Lo mismo aplica para el caso de una salida.
-Agregando un subject
*Update c125 add subject : (define, d123)*

En este caso se agrega al conjunto $R_{subject}$ del CE $c_{125}$ un elemento con rol de definición del concepto $d_{123}$ (el cual pertenece a algún árbol de dominio del sistema).
Si el CE ya posee en una de sus relaciones el concepto $d_{123}$, simplemente se agrega el rol (siempre y cuando no quede repetido).

-Agregando un prerrequisito:
*Update c125 add prerequisite : (d126, 'Low')*

### 3.11.4.2 Change

Permite modificar el valor de un elemento existente en un CE.
Esta opción es sólo válida para elementos de LOM no multivaluados y para elementos del ME.
(El cambio de un elemento en un grupo multivaluado se logra a través de borrados y adiciones)
Ejemplos:
- Cambiando el título de un CE:
*Update c125 change general.title : 'Física: Las Ondas'*

- Actualizando el valor de un elemento del ME:
*Update c125 change extensible.proyecto.fase : 4*

En este ejemplo se cambia el valor del atributo fase dentro de la categoría proyecto del CE $c_{125}$ y se le coloca un valor de 4.

- Actualizando valores existentes en una entrada dada:
Este es un caso especial para el cual la sintaxis es la siguiente:
*Update Coid change input : (nombre_entrada, nombre_elemento_cambio, valor)*
Donde *nombre_elemento_cambio* sólo puede ser la palabra *value* o *label*.
Ejemplos:
Cambiar el valor de la entrada subtítulo del CE $c_{125}$:
*Update c125 change input : (subtitulo, value, 'No')*
Cambiar el nombre de la entrada *subtitulo* por *subtitle*:
*Update c125 change input : (subtitulo, label, :'subtitle')*
No se permite cambiar el campo documentation por las mismas razones expuestas en 3.11.4.1.
Cambios a los conjuntos $R_{subject}$ y $R_{prerequisite}$ se logran a través de adiciones y borrados.

#### 3.11.4.3 Delete

Permite eliminar un elemento de un CE. Algunas restricciones aplican: Los elementos LOM que un CE posee como indispensables o básicos no pueden ser eliminados. En el ME no se puede eliminar un atributo padre si tiene hijos, se deben eliminar primero estos. No se puede eliminar ni una entrada ni una salida (ver 3.11.4.1).
Ejemplos:
- Eliminando un contexto educativo de un CE:
*Update c125 delete Educational.Context: 'Primary Education'*

La operación fracasará si el CE tiene un solo contexto educativo.
Los elementos multivaluados compuestos se eliminan completos, es decir, no se permite eliminar partes de un elemento multivaluado Ejemplo: Eliminando una contribución:
*Update c125 delete life_cycle.contribute:( entity:'Pedro Pérez')*

En este caso se eliminan todos los datos de la contribución. Si era la única contribución la eliminación fracasa. Si *Pedro Pérez* tenía varias contribuciones todas ellas se eliminarán (Se podría eliminar sólo 1 de sus contribuciones pero añadiendo más información, como la fecha).

- Eliminando un atributo del ME:
*Update c125 delete extensible.proyecto.fase*

En este caso se elimina la existencia del atributo fase el cual está ubicado dentro de la categoría proyecto, esta sigue existiendo y podría tener mas hijos. Si se hubiera intentado primero:
*Update c125 delete extensible.proyecto*

La operación fracasaría porque proyecto tiene hijos.
Recuérdese que existe la restricción de que en el ME no se permiten valores múltiples para un mismo camino.

-Eliminando un elemento de $R_{subject}$:
*Update c125 delete Rsubject: (define, d126)*
Se está eliminando el elemento con identificador de concepto igual a $d_{126}$ y rol *define* de $R_{subject}$. Si el CE posee otros roles con ese concepto, se elimina sólo ese rol de ese concepto.

Si a raíz de este borrado $R_{subject}$ quedase vacío la operación fracasa.

-Eliminando un elemento de $R_{prerequisite}$:

*Update c125 delete Rprerequisite: (d126, 'Low')*

El conjunto $R_{prerequisite}$ si puede quedar vacío, a diferencia del $R_{subject}$. En este caso sobraría el especificar el nivel de dificultad, sin embargo se ha conservado por coherencia.

#### 3.11.4.4 Observaciones sobre el operador de Actualización

- Sobre alteraciones a los conjuntos $R_{subject}$ y $R_{prerequisite}$ de un CE

Si un CE $c_1$ sufre una alteración en sus conjuntos $R_{subject}$ y/o $R_{prerequisite}$ esto puede dar lugar a conflictos que se presenten si dicho CE es miembro de otros, ya que puede dar lugar a violaciones tanto a la restricción de secuencia (sección 3.8.1.1) como a la ecuación de prerrequisitos (sección 3.8.1.2).

Algunas de las alternativas para dar solución a este problema son:

a) Impedir la actualización de dichos conjuntos si el CE es partícipe de al menos otro CE. La desventaja que conlleva esta alternativa es la falta de independencia desde el punto de "propiedad" de un CE por parte de su creador de poderlo alterar en este aspecto. Significa esto que el creador de un CE sólo podrá actualizar tales conjuntos pero sólo en sus CE's que no son usados por otros CE's.
b) Propagar los cambios y hacerlos efectivos siempre y cuando no se produzca ningún conflicto. Esta alternativa implica un gran trabajo porque puede generar una serie de alteraciones en cascada significativa.
c) Realizar la actualización y manejar un esquema de CE's válidos e inválidos. Los CE's inválidos serían aquellos que violan la restricción de secuencia o la ecuación de prerrequisitos, esto llevaría a un manejo algo complejo, no sólo por las implicaciones de verificación en cascada que se puedan generar sino también por el manejo adicional de objetos válidos e inválidos, pero tendría como ventaja el que se respeta la independencia de alteración de un CE por parte de su creador.

Otras alternativas podrían ser contempladas. En este trabajo se ha decidido optar por la alternativa a) ya que se espera que la alteración de los conjuntos $R_{subject}$ y $R_{prerequisite}$ no sea una tarea muy frecuente, de hecho dichos conjuntos deben *tender* a permanecer bastante estables.

- Sobre Alteraciones del aspecto composicional de un CE

El cambio de la composición de un CEC se puede realizar a través de la interfaz y no "a modo comando" ya que no es práctico realizar esta labor de manera manual, aunque se podrían proponer operadores para tal propósito en desarrollos futuros.
Otra posibilidad sería que el usuario altere directamente el documento XML que representa al grafo, lo cual es prácticamente impensable para un usuario no experto, por ello la interfaz es fundamental en este caso, así como también lo es para los operadores de creación.

### 3.12 Operador Show

*Show Section of components = Coid*

Donde section puede ser una de las siguientes: LOM_Metadata, Domain_Metadata, Interface (Inputs y Outputs), $R_{subject}$, $R_{prerequisite}$,Composition y All.
El operador sirve para observar cada una de las secciones que conforman a un CE.
El operador retornará, en todos los casos, un documento en formato XML (el cual es extraído directamente del documento XML que representa al CE) correspondiente a la sección solicitada, **sin embargo para el usuario final esto será trasparente, ya que a través de la interfaz, la salida se le presentará en un formato más "agradable".** Por ejemplo si el usuario escribe:
*Show Composition Of components = c123*
El resultado es el grafo que constituye la composición del $c_{123}$, que por supuesto debe ser un CEC de lo contrario el operador retorna un error.Veamos los otros casos:
*Show LOM_Metadata Of components = c125*
Retorna un documento XML con toda la información de los metadatos de LOM del CE $c_{125}$. El resultado para el usuario final es una forma (similar a la forma de entrada de los metadatos).
La opción *Domain_Metadata*, sólo muestra el ME, esta opción es fundamental porque a través de ella un usuario puede descubrir cual ME ha sido agregado a un CE en particular. Por ejemplo él podría descubrir que el CE $c_{125}$ posee una categoría *extensible.multimedia* y con base en esto emitir posteriormente una consulta en busca de otros CE's que hayan sido documentados así usando el operador de selección (*Get Components* ).
También se muestran las secciones de $R_{subject}$ e Interface (Inputs más Outputs). Igualmente para:
*Show Rprerequisite Of components = c123*

Retorna el conjunto de prerrequisitos del CE $c_{123}$. Visualmente se le presentará al usuario por medio de una tabla o forma.

Para este caso, existe una opción especial que permite extraer un subconjunto de prerrequisitos. Recuérdese que un CEC está formado por un grafo el cual ha sido construido con los operadores *Sequence* y *Union*. Por ejemplo:

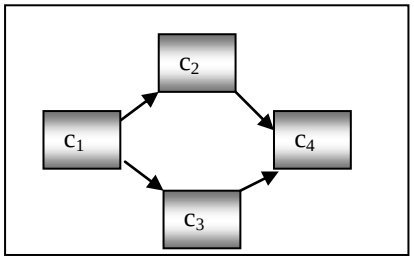



Figura 3.20: CEC $c_5$

Los prerrequisitos de $c_5$ se derivan de sus CEM's . (ver sección 3.8 para todas las especificaciones). Sin embargo un estudiante podría estar interesado en estudiar sólo el camino que lleva de $c_1$ a $c_4$, debido a que allí están los conceptos que requiere (ver adaptación, apéndice 2). Por lo tanto los prerrequisitos de $c_3$ no deben ser tenidos en cuenta. El operador entonces permite obtener el subconjunto de prerrequisitos de un CEC especificando una subcomposición así:

Supóngase que el concepto asociado al nodo concepto que contiene a $c_2$ es **p** y que el concepto asociado al nodo concepto que contiene a $c_4$ es **q**, entonces:
*Show prerequisites of GetPaths from c5 with { p,q }*

Nota: Ver la descripción del operador *GetPaths* en la siguiente sección.
El conjunto de prerrequisitos devuelto será la unión de los prerrequisitos de $c_1$,$c_2$ y $c_4$.
Ahora supóngase que el concepto asociado al nodo que contiene a $c_1$ es **r**. Si se ejecuta:
*Show prerequisites of GetPaths from c5 with { r,q }*
Devolverá el conjunto de prerrequisitos de $c_1$, $c_2$ y $c_4$ ó de $c_1$, $c_3$ y $c_4$ dependiendo de cual de las dos opciones devuelva el operador GetPaths.
Finalmente la opción *All* muestra todas las secciones anteriores de un CE.

Nota: Al mostrar la sección de los metadatos, se podría pensar en que el usuario sólo esté interesado en observar partes de él (al igual que de las otras secciones), por ejemplo, mostrar sólo la sección *general* de LOM y más aun por ejemplo: mostrar sólo los idiomas (*languages*). Esto perfectamente

viable, sin embargo en este trabajo se ha restringido a sólo poder observar las secciones completas, lo cual no es mayor inconveniente a la hora de buscar un dato concreto. En desarrollos futuros esto se puede extender a este punto si se observa una marcada necesidad.

## 3.13 Operador de Obtención de Caminos: GetPaths

### 3.13.1 Definiciones

**N0E**: Nodo concepto con grado de entrada cero, es decir, son aquellos nodos que inician el desarrollo del CEC. De estos nodos "nacen" flechas pero ninguna llega a ellos.
**N0S**: Nodo concepto con grado de salida cero, es decir, son aquellos nodos en los cuales termina el desarrollo de un CEC. A estos nodos les "llegan" flechas pero no les "nacen".
**Nodos Intermedios**: Nodos concepto con grado de salida y grado de entrada diferentes de cero, es decir, nodos a los que llegan y de los que nacen flechas.
**Camino**: Ver definición en la sección 3.6.
Se define **LS** como la longitud de un conjunto de caminos, la cual es igual al número total de *nodos conceptos distintos* presentes entre todos los caminos.

### 3.13.2 Propósito y sintaxis del operador

El operador permite obtener un conjunto de caminos de mínima LS, a partir de un CEC el cual contenga una serie de conceptos dado. Por ejemplo, supóngase que se desea obtener del CEC $c_{13}$, figura 3.21, el conjunto de caminos de mínimo LS que contenga a los conceptos B y D.
Para este caso son posibles los siguientes conjuntos de caminos: {(A→B→D→F)} y {(A→B→D→F),(A→C→E→F)}. En este caso se selecciona el primer conjunto ya que posee una LS de 4 frente a 6 del segundo conjunto (el cual de hecho es todo el CEC).

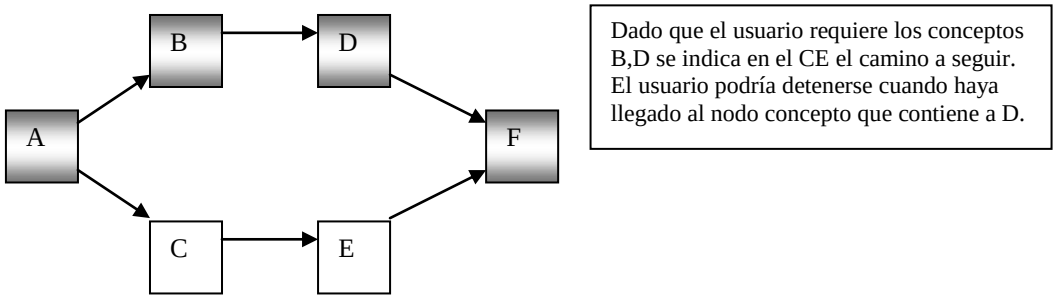



Figura 3.21: CEC $c_{13}$

**Nota**: En este ejemplo se ha usado el nombre del concepto para referirse al nodo concepto. Se seguirá esta notación siempre y cuando no se presente confusión. (Por ejemplo cuando no hayan varios nodos concepto con el mismo concepto)

En este mismo grafo, supóngase que se desean obtener los caminos que contengan los conceptos B y E. El operador devolverá un conjunto con 2 caminos los cuales se requieren para abarcar los conceptos B y E. El conjunto es: {(A→B→D→F), (A→C→E→F)}; esto significa que si un usuario desea estudiar los conceptos B y E usando este CEC deberá estudiar todo el CEC. (Exceptuando quizás los nodos posteriores a B y a E, pero sí debe iniciar a partir de A Esto implica que el estudiante puede no cubrir el 100% del curso). Veamos otros ejemplos.

Sea el grafo de la figura 3.22 que representa un CEC $c_{18}$. Supóngase que se desean obtener un conjunto de caminos que comprenda los conceptos A y C. En este caso existen varios conjuntos de caminos mínimos. Un primer conjunto de caminos es:
{(A→B→C)}. Otro posible conjunto de caminos que también consta de un solo camino es: {(A→E→C)}. Otro es {(A→D→I→H→F→C)} y finalmente
{(A→D→E→C)}. El operador *no* devuelve todos estos posibles conjuntos de caminos, sólo devuelve uno de estos conjuntos, aquel que tenga menor LS. Por ejemplo el conjunto {(A→B→C)} tal y como aparecen sombreados en la figura 3.22.

Nota: En caso de empates la decisión es aleatoria, aunque un segundo criterio podría ser seleccionar aquel conjunto de caminos con menos prerrequisitos.

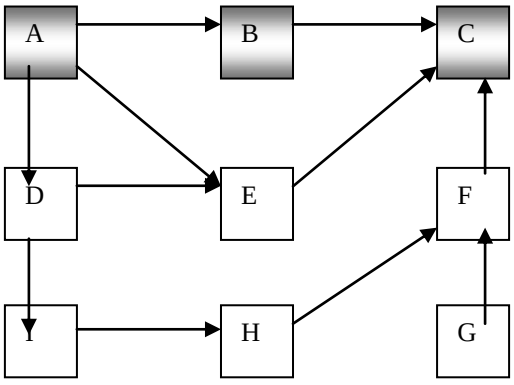



Figura 3.22: CEC $c_{18}$

Ahora supóngase que se tiene un CEC $c_{17}$ cuyo grafo es el de la figura 3.23. Se desea obtener el conjunto de caminos que contienen a los conceptos B y C. La respuesta está dada por un conjunto de 2 caminos que se requieren para abarcar ambos conceptos, estos son: {(J→E→B→D),(J→E→C→D)}. En este caso se requieren 2 caminos pero no todo el curso, por ejemplo el estudiante no tiene que estudiar los nodos con conceptos K y L. (Incluso el estudiante podría decidir no estudiar el concepto D, pero sí debe abordar los conceptos J y E).

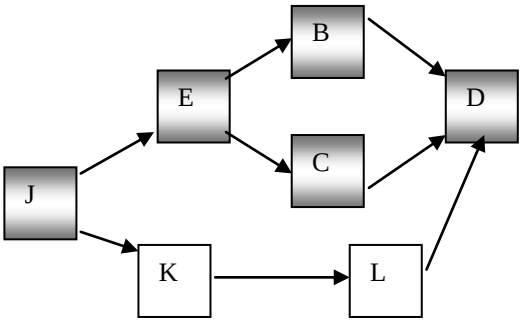



Figura 3.23: CEC $c_{17}$

Finalmente supóngase que se tiene el siguiente CEC $c_{38}$, figura 3.24.

Ya que es posible que varios nodos conceptos de un CEC tengan el mismo concepto, para evitar ambigüedad, se han colocado identificadores explícitos a los nodos concepto.

Si se desea obtener el conjunto de caminos que contengan a los caminos C y D, en este caso se tienen dos posibles respuestas, la primera consta de dos caminos y es:

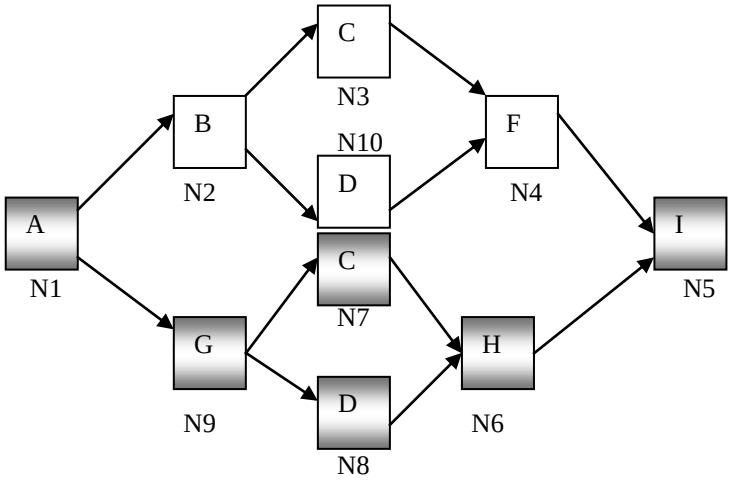



Figura 3.24: CEC $c_{38}$

{(N1→N2→N3→N4→N5), (N1→N2→N10→N4→N5)}. La segunda respuesta también se compone de dos caminos: {(N1→N9→N7→N6→N5), (N1→N9→N8→N6→N5)}.
El operador devolverá en este caso cualesquiera de las dos posibles respuestas, ya que la LS de los conjuntos es igual.

La sintaxis del operador es la siguiente:
*GetPaths from Coid with { Concept List }*

Ejemplo: *GetPaths from c28 with { B,C }*

Se retorna un conjunto de caminos mínimos si el CE $c_{28}$ contiene ambos conceptos, se devolverá un conjunto vacío en caso contrario.

### 3.13.3 Objetivo de uso de este operador

Como se habrá advertido la intención con este operador es ofrecer al estudiante un subconjunto de un CEC. Este subconjunto incluye una serie de conceptos en los cuales el estudiante está interesado. La idea es guiarlo a través de los caminos dentro del CEC de tal forma que el estudiante alcance su objetivo (estudiar los conceptos deseados).
Esta guía es importante porque permite enfocar directamente al estudiante hacia el estudio de una línea o camino (**que se espera guarde cierta coherencia y aspectos didácticos, ya que se supone ha sido concebida por un profesor con criterios pedagógicos**) inmersa en un CE.
Este operador es usado en el sistema de adaptación para presentar de una manera gráfica los caminos que deben ser seguidos por un estudiante. (Ver apéndice 2.)

Igualmente sirve para seleccionar entre el conjunto de CEC's que contienen tales conceptos, **aquel cuyo conjunto de caminos posea menos prerrequisitos**, apoyándose a su vez en el operador *Show Prerequisites* (véase sección 3.12).

## 3.14 Operador de Replace

*Replace in Coid ( Noid ) where Preferences_Condition*
*and Comparison_Operator Difficulty*

Donde Comparison_Operator puede ser =, <,>. La condición sobre preferencias se construye igual que las condiciones para el operador de consulta, la única diferencia es que aquí sólo se admiten condiciones que tengan que ver con preferencias técnicas.

El objetivo del operador *Replace* es permitir que en tiempo de ejecución (instanciación) en un CEC el usuario pueda cambiar CE's sugeridos de los nodos concepto de acuerdo a sus necesidades. Recuérdese que dentro de la estructura de cada nodo concepto existe una propiedad que indica si el CE sugerido es intercambiable o no.

Sea un CEC $C_{68}$ y sea un sujeto $U_1$ (learner) que va a instanciar dicho CEC; el siguiente diagrama explica el proceso de *Replace*:

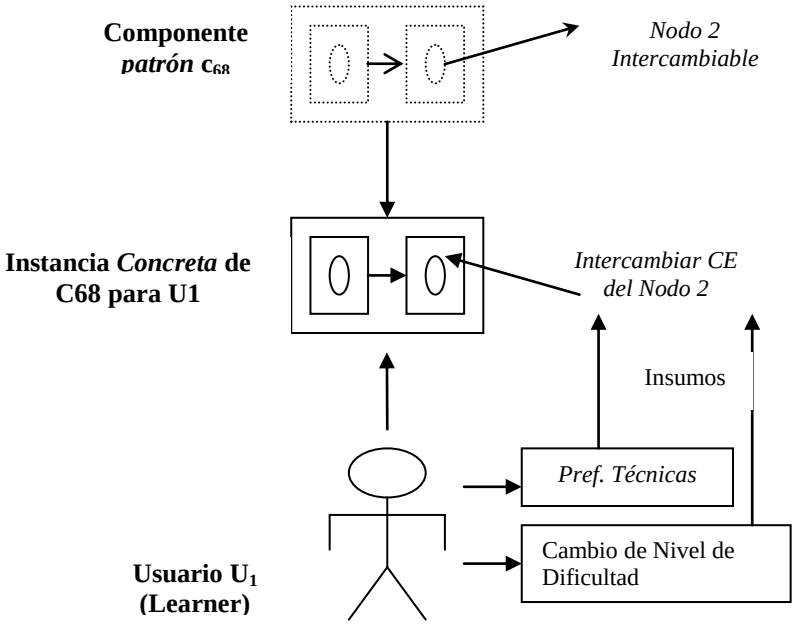



Figura 3.25: Proceso del operador *Replace*

Para realizar una operación de *Replace* se debe tener presente lo siguiente:

- El nodo concepto debe ser intercambiable (interchangeable = true)
- El CE remplazante debe *desarrollar el concepto* del nodo bajo *el mismo rol* que el CE remplazado.
- El criterio para realizar un remplazo consiste en que el usuario puede variar el nivel de dificultad (véase sección 3.14.1) de un CE o las preferencias técnicas (formato, duración etc.).
- El operador afecta instancias de CEC's.

- El peso del nodo concepto no se afecta por el remplazo, ya que dicho peso está asociado al concepto del nodo no al CE que lo desarrolle. (Independientemente del nivel de dificultad o naturaleza del CE)
- *Los prerrequisitos del nodo no cambian ante un remplazo.* Estos prerrequisitos se conservan y sirven de base para otros remplazos, es decir, no se adoptan como prerrequisitos los provenientes de un CE remplazante.

Ejemplo: *Replace in C68 (N2) where technical.format='Video' and > Difficulty*

La condición de preferencias o la de nivel de dificultad pueden estar ausentes, pero al menos una de las dos debe estar presente.
Si el usuario sólo especifica un cambio en el nivel de dificultad, se toman por defecto las preferencias extraídas de su perfil. Si el usuario sólo especifica cambio en preferencias se buscarán CE's que tengan el mismo nivel dificultad que el CE actual.
Para el cálculo del nivel de dificultad se debe implementar una rutina que de acuerdo a un criterio dado devuelva un valor que represente la dificultad de un CE (véase sección 3.14.1)

### 3.14.1 Sobre el nivel de dificultad

El cáculo del nivel de dificultad de un CE es una labor compleja. Una consideración sencilla sería trabajar como se propone en LOM, con el elemento *difficulty*, usando valores *Very Easy, Easy, Medium, Difficult y Very Difficult* para cada CE, sin embargo este esquema es discutible y subjetivo. Otra alternativa sería la de manejar un nivel de dificultad de un CE dependiendo del sujeto y teniendo en cuenta lo que éste ya haya aprendido.
Un trabajo futuro más profundo debe ser realizado al aspecto, *incluso la sintaxis del operador Replace podría permanecer intacta ya que lo que se debe cambiar es el procedimiento para determinar el nivel de dificultad de un CE*.

Para el prototipo (Ver sección 5) se tomarán algunas decisiones de implementación de este operador, en concreto el nivel de dificultad se calculará de acuerdo al número de prerrequisitos (a mayor número, más difícil).

## 3.15 Operador Find Similar

### 3.15.1 Fundamentos: El operador de subsumption de la LD

La inferencia básica en LD (ver sección 3.10) se logra por medio de la subsumption la cual se denota por C ⊑ D [4], donde C y D son conceptos (que pueden estar formados por los constructores vistos). La subsumption es el problema de determinar si el concepto D (el subsumer) es considerado más general que el denotado por el concepto C (el *subsumee*), es decir, la subsumption prueba si el primer concepto representa un subconjunto del segundo.

Para verificar si la subsumption se cumple se deben tener en cuenta por supuesto las relaciones definidas en la terminología. Por ejemplo de la figura 3.18 se aprecia que progenitor ⊆ persona. Este es un caso muy simple, pero en general para conceptos complejos y en especial si se usan constructores como los cuantificadores y otros no vistos aquí, se emplea un algoritmo formal, denominado *tabloide*, el cual se expone en la sección 3.15.2.

El objetivo es incorporar elementos de la LD para describir el sistema de los CE's y construir un operador de consulta de similitud basado en el operador de subsumption. En todo el estado del arte no se observó la aplicación de la LD en el mundo de los recursos educativos. Lo más próximo fue la reciente aplicación de LD en el mundo de los Web Services, tal y como puede observarse en [Hacid 02]

### 3.15.2 Algoritmo del tabloide para probar subsumption

Para probar si la expresión C ⊆ D se cumple, el algoritmo del tabloide lo que hace es probar si: C ⊓ ¬D es *insatisfiable* ( o sea si denota el concepto vacío), si es así se cumple la subsumption de lo contrario no.

Para ello se procede así: Sea c0 = C ⊓ ¬D

**1.** La expresión C ⊓ ¬D debe estar normalizada (NFF=Negation Formal Form) es decir expresada en la forma:

$A_1$ *constructor ... constructor* $A_m$ *constructor (cuantificador*$R_1$*.*$C_1$*) constructor ...*

*constructor (cuantificador*$R_n$*.*$C_n$*)*

Condiciones:

[4] Se utilizará el símbolo ⊆ siempre que no de lugar a confusión con la operación subconjunto.

Cuantificador puede ser ∀, ∃

Constructor puede ser conjunción ( ⊓ ) o disyunción ( ⊔ ).

La negación (¬) sólo puede ocurrir al frente de conceptos.

$A_1$ ... $A_m$ son conceptos distintos (y expandidos con base en los conceptos atómicos), $R_1$, ..., $R_n$ son roles diferentes y $C_1$, ..., $C_n$ son a su vez expresiones en NFF.

Un ejemplo de una expresión en NFF: ( ∃ R.A) ⊓ ( ∃ R.B) ⊓ (∀R. ¬A) ⊔ Z

Donde A, B y Z son conceptos atómicos.

**2.** Se inicia un conjunto A={ c0(x0) } donde x0 es un individuo cualquiera que cumple c0.

**3.** Se aplican las siguientes reglas de acuerdo al constructor empleado hasta que ya no se pueda aplicar ninguna de ellas:

∃: Condición: Si A contiene (∃ R.C)(x), y no hay un individuo z tal que C(z) ni R(x,z) en A

Acción: A'=A U {C(y), R(x,y)} donde *y* es un individuo que no aparece en A.

∀: Condición: Si A contiene (∀R.C)(x) y R(x,y) pero no contiene C(y).

Acción: A'= A {C(y)}

⊔ : Condición: Si A contiene(C1⊔ C2)(x) pero no contiene ni C1(x) ni C2(X).

Acción: A'= A U {C1(x)}, A''= A U {C2(x)}

⊓: Condición: Si A contiene (C1 ⊓ C2)(x) pero no contiene ni C1(x) ni C2(x).

Acción: A'= A U {C1(x), C2(x)}

**4.** Se examina el conjunto A resultante en busca de conflictos, si se encuentran entonces la expresión es *insatisfiable* y se cumple la subsumption. De lo contrario la expresión es *satisfiable* y por lo tanto no se cumple la subsumption. Un conflicto se produce si:

a) ⊥(x) está presente en A

b) (P) (x), (¬P)(x) están presentes en A.

Es de anotar que la regla de la unión genera varios conjuntos A's, en tal caso se debe examinar cada uno de ellos y si se encuentra al menos uno que sea *satisfiable* entonces la expresión lo es y por tanto no se cumple la subsumption. Veamos 3 ejemplos. En la sección 3.15.3 se presentan ejemplos para CE's.

**Ejemplo 1:**

Suponga que : woman = person ⊓ female. Se desea probar si: woman ⊆ person.

Se debe entonces probar la satisfabilidad de: woman $\sqcap$ ¬person

1. Se normaliza la expresión: person $\sqcap$ female $\sqcap$ ¬person

(en este ejemplo desde ya se observa que la subsumption se cumple pero se realizará el proceso para ejemplificar como trabaja el algoritmo)

2. Se inicia el conjunto A= { (person $\sqcap$ female $\sqcap$ ¬person) (x0) }

3 Aplicando la regla de la $\sqcap$ nos queda :

A= { (person $\sqcap$ female $\sqcap$ ¬person) (x0), (person) (x0), (female)(x0) (¬person)(x0) }

Al conjunto A ya no se le pueden aplicar mas reglas por lo tanto se continúa con el paso 4

4. Se busca si existen conflictos en el conjunto A:

En este caso se encuentra que (person) (x0) y (¬person)(x0) están presentes en A, por lo tanto el conjunto A es inconsistente y se cumple la subsumption.

**Ejemplo 2:**

Se desea probar si: ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\subseteq$ ( ∃ R.A)

Se debe entonces probar la satisfabilidad de : ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\sqcap$ ¬ ( ∃ R.A)

1. Normalizando: ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\sqcap$ (∀R. ¬A)
2. Se inicia el conjunto A= { ( ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\sqcap$ (∀R. ¬A )) (x0) }
3. Se aplica la regla de la conjunción:

A= { ( ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\sqcap$ (∀R. ¬A )) (x0), ( ∃ R.A) (x0), ( ∃ R.B) (x0) , (∀R. ¬A )(x0) }

Al aplicar la regla de a ∃ a ( ∃ R.A) (x0) se generan los siguientes miembros de A:

(A)(x1) , R(x0,x1)

Y al aplicar la regla de a ∃ a ( ∃ R.B) (x0) se generan:

(B)(x2), R(x0,x2)

Por lo tanto hasta este punto el conjunto A está así:

A= { ( ( ∃ R.A) $\sqcap$ ( ∃ R.B) $\sqcap$ (∀R. ¬A )) (x0), ( ∃ R.A) (x0), ( ∃ R.B) (x0) , (∀R. ¬A )(x0),
(A)(x1) , R(x0,x1), (B)(x2), R(x0,x2) }

Ahora se aplica la regla ∀ a (∀R. ¬A )(x0) y se generan los siguientes miembros:

(¬A)(x1), (¬A) (x2)

Ya no se pueden aplicar mas reglas por lo tanto se continúa con el paso 5

4. Como en el conjunto A están presentes los elementos: (A)(x1), (¬A)(x1) existe un conflicto y por lo tanto A es inconsistentes, entonces se cumple la subsumption.

**Ejemplo 3:**

Se desea probar si: $( \exists R.A) \subseteq ( \exists R.A) \sqcap ( \exists R.B)$

Se debe probar la satisfabilidad de: $( \exists R.A) \sqcap \neg ( ( \exists R.A) \sqcap ( \exists R.B) )$

1. Normalizando se obtiene: $( \exists R.A) \sqcap ( (\forall R. \neg A) \sqcup (\forall R. \neg B) )$
2. Se inicia el conjunto A= { $(( \exists R.A) \sqcap ( (\forall R. \neg A) \sqcup (\forall R. \neg B) )) (x0)$ }
3. Aplicando la regla de la conjunción nos queda:

A= { $(( \exists R.A) \sqcap ( (\forall R. \neg A) \sqcup (\forall R. \neg B) )) (x0)$,

$( \exists R.A)(x0), ( (\forall R. \neg A) \sqcup (\forall R. \neg B))(x0)$ }

Aplicando la regla de ∃ a $( \exists R.A)(x0)$ genera los siguientes elementos para A:

$(A)(x1), R(x0, x1)$

Hasta este punto A está así:

A= { $(( \exists R.A) \sqcap ( (\forall R. \neg A) \sqcup (\forall R. \neg B) )) (x0)$,

$( \exists R.A)(x0), ( (\forall R. \neg A) \sqcup (\forall R. \neg B))(x0), (A)(x1), R(x0, x1)$ }

Ahora se aplica la regla de la unión la cual genera dos conjuntos A' y A'' los cuales quedan así:

A' = A U $(\forall R. \neg A)$

A''= A U $(\forall R. \neg B)$

Al aplicar a A' la regla de ∀ a $(\forall R. \neg A)$ se genera el miembro: $(\neg A) (x1)$

Y al aplicar la misma regla al elemento $(\forall R. \neg B)$ de A'' , se genera el miembro: $(\neg B)(x1)$.

4. En A' se encuentra el conflicto: $(A)(x1)$ , $(\neg A)(x1)$ pero en A'' no hay conflicto por lo tanto el resultado total es consistente, por lo tanto la subsumption no se cumple.

### 3.15.3 Objetivo y sintaxis del operador

*Find Similar [all | primitive | composed] Components*

*Where Contents [("roles")] "domain-concept"*

El operador permite al sistema efectuar búsquedas por proximidad de contenidos más roles que se adapten a los requerimientos de los usuarios (aprendizaje por objetivos, véase apéndice 2). Lo anterior contribuye en el proceso de adaptación simplificando el proceso de búsqueda y estableciendo una guía por contenidos a los usuarios.

Si no se especifican roles, el sistema automáticamente las trae desde el perfil del usuario (learner group, ver apéndice 2). *all* es la opción por defecto.

La opción composed es muy útil cuando se desean evitar CEP's que posiblemente tengan una granularidad alta (véanse secciones 3.3.3.1 y 3.4). Veamos algunos ejemplos:

**1.** *Find Similar Components Where Contents ("define") "concept-C"*

En términos de L.D: ∃define.concept-C

Traerá los CE's (CEC's o CEP's) que definan el concepto-C. Muy probablemente traerá CEP's tópicos o CEC's (cursos o lecciones por ejemplo).

**2.** *Find Similar Primitive Components Where Contents ("evaluate") "concept-c"*

En términos de L.D: ∃evaluate.concept-C ⊓ ∀composed_by.⊥

Traerá *posiblemente* CEP's ejercicios tal y como se definieron en la sección 3.10.2. Posiblemente porque también pueden ser retornados CEP's de granularidad, es decir, que aparte de evaluar el concepto C, tengan otras relaciones o desarrollen el concepto C con otros roles.

**3.** *Find Similar Components Where Contents ("define", "apply") "concept-c"*

En términos de L.D: ∃define.concept-C ⊓ ∃apply.concept-C

Traerá posiblemente CEC's que sean lecciones o cursos. Incluso puede traer CEP's de granularidad alta.

**4.** *Find Similar Components Where Contents ("define", "experiment", "evaluate") "concept-c"*

En términos de L.D: ∃define.concept-C ⊓ ∃experiment.concept-C ⊓ ∃evaluate.concept-C

Traerá posiblemente CEC's que sean lecciones o cursos. Igualmente puede traer CEP's de granularidad alta.

**5.** *Find Similar Composed Components Where Contents ("evaluate") "concept-c"*

En términos de L.D: ∃evaluate.concept-C ⊓ ∀composed_by.(∃evaluate.concept-C)

Traerá CEC's que *muy posiblemente* son workshops (homogéneos).

**6.** *Find Similar Component Where contents ("define", "apply", "evaluate") "concept-c" and ("motivate") "concept-b"*

En términos de L.D:

∃define.concept-C ⊓ ∃apply.concept-C ⊓ ∃evaluate.concept-C ⊓ ∃motivate.concept-B

Traerá muy posiblemente CEC's que sean cursos. Igualmente puede traer CEP's de granularidad muy alta que tengan todo lo anterior.

7. *Find Similar Composed Component*

*Where Contents ("evaluate") "concept-c" and ("evaluate") "concept-d"*

En términos de L.D: ∃evaluate.concept-C ⊓ ∃evaluate.concept-D ⊓ ∀composed_by.(∃evaluate.concept-C)

Traerá CEC's que muy posiblemente son workshops (heterogéneos).

# 4 Aporte

## 4.1 Utilización de la teoría de grafos para la composición de los CEC's

La teoría de grafos ha sido aplicada en el marco de los recursos educativos de dos formas:

a) Para ayudar en el proceso de búsqueda y adaptación.

Se ha concebido un operador GetPaths (véase sección 3.13) el cual permite obtener un conjunto de caminos dentro de un CEC que contiene un grupo de conceptos dado por un usuario (por ejemplo un grupo de conceptos que un estudiante desea aprender); el operador se encarga de devolver el conjunto de caminos más corto que contiene a los conceptos dados, de tal forma que sirve de guía al señalar una ruta pedagógica coherente a seguir con el objetivo de aprender los conceptos solicitados.

b) Para la construcción de la composición de un CEC.

La composición de un CEC está basada en un grafo acíclico conectado. Se ha utilizado además el operador de unión de grafos para formalizar la construcción del grafo. La representación de un recurso educativo como un grafo (CEC), no se observó en la literatura revisada. Las ventajas de esta propuesta incluyen entre otras:

- Visualización del contenido de un curso (CEC) de manera amigable e intuitiva para los usuarios.
- Se facilita el proceso de guiar al estudiante por un subgrafo de conceptos (ver operador GetPaths, sección 3.13).
- Plantea alternativas al esquema simple de tabla de contenido (secuencialidad) de un curso (diferentes estudiantes pueden seguir diferentes caminos en un mismo curso) confrontar con tabla de contenido del proyecto Multibook , sección 2.2.3.2.
- Se facilita el proceso de construcción de un CEC (se ha construido un prototipo que permite construir el grafo de composición de manera "visual", ver sección 5.2.2).
- Cada nodo del grafo puede ser personalizado por parte del estudiante, es decir, se pueden crear cursos patrones (ver 3.6.1) en lugar de cursos específicos.

## 4.2 El lenguaje de manipulación

### 4.2.1 El lenguaje propuesto frente a otras aproximaciones

Se pretende establecer cuales son las diferencias y similitudes que existen entre el lenguaje aquí propuesto y las otras aproximaciones para establecer así la contribución.
Primero se realiza una comparación frente a los sistemas educativos analizados en la sección 2.2.3, y luego se establece una analogía con un lenguaje de consulta traído del mundo de los Web Services, una de las áreas de mayor interés en la actualidad y con la cual se ha encontrado una interesante analogía.

#### 4.2.1.1 Comparación con otros Sistemas Educativos

Hecha la revisión de publicaciones e internet, se examinaron los Sistemas educativos (ver sección 2.2.3) desde el siguiente punto de vista: ¿Qué tipo de facilidades de construcción y en especial de consulta sobre los recursos (llámense componentes educativos, media bricks, Interbooks etc.) son proporcionados?

Lo que se logró concluir es que la gran mayoría de estos sistemas, no se centran en aspectos como la elaboración de consultas, no porque sea algo irrelevante, sino porque tenían otras preocupaciones, por ejemplo algunos sistemas se preocupan mucho por el perfil del usuario, caso de KBS [Henze 00], otros eran bastante sencillos y orientados a un único recurso, como ELM-ART [Weber 97] (en este caso el problema de búsqueda de otros recursos educativos no existía, sencillamente porque *sólo existía un* recurso educativo).

En el caso de AHA este posee un lenguaje de preprocesamiento (véase sección 2.2.3.3) de tipo scripting (similar a HTML) para la creación de sus objetos educativos y su último desarrollo fue la introducción de un lenguaje tipo SQL para adicionar triggers los cuales desencadenan automáticamente la actualización del perfil del usuario. Respecto a los metadatos posee unos elementos mínimos (título del curso, conceptos asociados, el directorio en el Web Server, email de su autor etc.)

En el caso de Interbook, (véase 2.2.3.4) la creación de un recurso educativo es más una transformación de un documento escrito en Word hacia una página HTML que se obtiene por medio

de un programa "traductor". Posee un glosario general indexado donde se reúnen todos los conceptos de todos los Interbooks existentes y posee un operador de búsqueda similar al de Word, el cual recibe una cadena de texto y busca en el contenido de todos los Interbooks.

De hecho exceptuando a Multibook --que usa LOM y plantea la introducción de *templates* que facilitan la entrada de metadatos y funciones de exportación e importación de los metadatos en formato XML-- la presencia de metadatos es pobre en estos sistemas, por consiguiente no se emplea ni se incluye un lenguaje de consulta que opere sobre ellos.
Las posibilidades de explotar los metadatos son claras (ver sección 2.1.3, [Goguen 86] y [Klein 02]) y el lenguaje propuesto en nuestro proyecto pretende suplir esta necesidad.

El lenguaje aquí propuesto contiene algunos elementos que van mas allá que los que proporciona un lenguaje de consulta típico. En particular se han desarrollado operadores como el de *GetPaths* (sección 3.13), *find similar* (sección 3.15) y la opción *with recursive* (para CEC's) del operador *Get Components*. Además se proponen dos operadores de *creación de CEC's basados en grafos* (secciones 3.8.1 y 3.8.2).

Finalmente es importante mencionar que aunque se trató de experimentar desde el punto de vista de *authoring* (crear un curso) con estos sistemas, en algunos casos no fue posible debido a que no estaban disponibles, otros si lo estaban pero para sistemas operativos (Apple Macintosh concretamente, caso de Interbook) que no se poseían. Se podría pensar entonces que las conclusiones que se derivan podrían no ser muy exactas. Sin embargo sí se dispuso en general de documentación (manuales de authoring, sitios oficiales, demos --algunos en línea-- y artículos correspondientes) lo suficientemente clara y completa, en la mayoría de los casos, como para poder establecer como se concebían cursos en estos sistemas desde el punto de vista de *authoring*.
Véase asimismo la sección 4.4.3 donde se explica como los operadores propuestos contribuyen en el proceso de adaptación.

#### 4.2.1.2 Una analogía con los Web Services

No es el objetivo realizar una comparación exhaustiva de este lenguaje frente al enorme panorama de los Web Services, sólo se presentará un caso particular con el cual se lograron establecer algunas similitudes interesantes.

Un análisis exhaustivo del estado del arte de los Web Services está más allá del alcance de este proyecto. De hecho no es uno de los temas principales de este trabajo. El estado del arte se examinó (ver seccion 2) para los temas relevantes.

De acuerdo a [Klein 02], la precisión en la recuperación de la información es actualmente una labor de investigación ya que la gran mayoría de las aproximaciones se han enfocado a la búsqueda de palabras claves (keywords) en documentos y no han tenido en cuenta otras posibilidades debido a la facilidad que conlleva una técnica basada en keywords, por ejemplo, *los autores no tienen que preocuparse por documentar de una manera especial*.
Si se desean obtener resultados más precisos, entonces para cada **documento deberá añadirse más información sobre él** (metainformación) es decir una descripción sobre su contenido, y de esta manera los "agentes" de búsqueda podrán explotar esa información. Por supuesto que ello implica un **esfuerzo adicional por parte de los autores** y se debe tratar en lo posible de **minimizar dicho esfuerzo mediante el uso de herramientas amigables y automatizando el proceso hasta donde sea posible.** Se debe establecer un balance entonces: *Precisión de la recuperación vs. Cantidad de Metainformación.*

Dicho artículo se centra en la recuperación de Web Services, para ello definen un lenguaje especial el cual se describe brevemente más abajo en esta misma sección.

En nuestro proyecto el CE es el elemento sobre el cual se construye un lenguaje de consulta simple que permite explotar su estructura. El uso de los **Metadatos de LOM** (modificados como se ha explicado en la sección 3.5), la documentación de las **entradas**, **salidas** y las relaciones $\mathbf{R_{subject}}$ **y** $\mathbf{R_{prerequisite}}$ con el **árbol de dominio**, el cual incluye a su vez para cada concepto un manejo de sinónimos y traducciones y jerarquía de conceptos; son elementos suficientes y adecuados para desarrollar algunos operadores de consulta.
Por supuesto que el éxito de la recuperación estará basado en que se mantenga por parte de los autores dicha **política de documentación** y que los CE's sean relacionados con el lugar "correcto" del árbol de dominio. Igualmente (aunque este es un aspecto de implementación) deberán existir mecanismos de indexación que permitan obtener buenos tiempos de desempeño.

Con el objetivo de llevar a cabo una sencilla comparación entre los CE's aquí propuestos y los Web Services, se tiene que en el caso de estos últimos se documenta lo siguiente:

- **Atributos**: (descripción del servicio, tiempo de desempeño etc.)
- **Puertos**: (Entradas, lo que requiere para trabajar y salidas, o sea, lo que produce)
- **Descomposición**: (Documentación de subactividades que llevan a cabo el Web Service y como se comunican entre ellas, esto se denomina dependencias)
- **Excepciones**: Condiciones de falla del Web Service y acciones que se siguen
- **Especializaciones**: El servicio se ubica dentro de una ontología de servicios ya establecida (dichos autores siguen una propuesta de ontología de servicios desarrollada en el MIT la cual se llama MIT Process Handbook), por ejemplo un Web Service que ofrece el servicio de Venta de vehículos se ubica debajo del servicio más general llamado simplemente venta.

Nótese que la analogía es alta entre estas dos estructuras: Atributos vs. Metadatos, Puertos vs. Entradas y Salidas, Descomposición vs. CEC's.
Con el aspecto de excepciones no se tiene un elemento análogo y con el de Especializaciones la analogía es menos clara pero se pueden encontrar elementos comunes en la relación de los CE's y el árbol de dominio (ver sección 3.3.3).

Es por lo tanto de gran interés comparar lenguaje propuesto en nuestro proyecto frente al lenguaje propuesto en [Klein 02], dicho lenguaje es denominado PQL (Process Query Language). Veamos un ejemplo en PQL el cual va a encontrar los Web Services que sean servicios **Préstamos** y que tengan un puerto de entrada llamado **Colateral** cuyo valor sea **Propiedad Raíz**. Dicho requerimiento se escribe así:

1. *Entity ?proc ISa Prestamo*
2. *Relationship ?proc has-port ?port1*
3. *Entity ?port1 ISa input-port*
4. *Entity ?port1 ISa Colateral*
5. *Entity ?port1 has-attribute ?attr1*
6. *Entity ?attr1 ISa name*
7. *Relationship ?attr1 has-value ?val :test(= ?val "Propiedad Raíz")*

Veamos como funciona; en la línea 1 se buscan los Web Services que sean Préstamos, y los que cumplan serán retornados en la variable ?proc (en PQL las variables se preceden con el signo ? y son variables de conjunto). En la línea 2 se examina el conjunto resultado ?proc y se pregunta cuales servicios tienen puertos (has-port) y el resultado queda en la variable ?port1.
En la línea 3 se pregunta cuales de los puertos retornados en ?port1 son de tipo entrada.

y en la línea 4 se pregunta cuales puertos son Colaterales y el conjunto resultante queda en ?port1 y el conjunto resultante queda en ?port1.
En la línea 5 se pregunta cuales de los puertos de entrada colaterales resultantes tienen atributos (valores) y el resultado queda en la variable ?attr1.
En la línea 6 se pregunta cuales atributos presentes en ?attr1 son names (nombres) Finalmente en la línea 7 se pregunta si el valor de alguno de los atributos es Propiedad Raíz.

Es una exploración en la estructura del Web Service que se va adentrando poco a poco hasta llegar a preguntar lo que se requiere en el "nivel" más interno.
En el lenguaje propuesto en nuestro proyecto la consulta se puede establecer similarmente así:

*General.Title='%Prestamo%' and Inputs.label='Colateral' and Inputs.Value='Propiedad Raíz'*

La expresión es más compacta y un poco más natural que PQL aunque en esta analogía el resultado serían por supuesto Coid's en el otro caso serían identificadores de los Web Services que cumplieron las condiciones, la esencia de la consulta es la misma: Hallar elementos (ya sean CE's o Web Services) que tengan una entrada llamada *Colateral* cuyo valor sea *Propiedad Raíz*.
(Se utilizó el *title* para buscar el nombre del servicio, también se pudo haber usado *description*).

Por supuesto que PQL es un lenguaje orientado a Web Services y fue diseñado de tal forma que hiciese frente a otro tipo de requerimientos (preguntas que involucran la consulta de excepciones y actividades) pero aún así la analogía es muy fuerte, por ejemplo en el lenguaje propuesto en nuestro proyecto, el uso de la cláusula With Recursive (ver sección 3.11.2) permite explorar la composición "hasta el fondo" (si así se desea o hasta un nivel específico) de un CEC, de una manera relativamente sencilla...

Precisamente la búsqueda de servicios basada en su descomposición no se había logrado hasta ese momento según [Klein 02], en nuestro caso el lenguaje propuesto permite tal tipo de búsqueda igualmente pero enfocada a recursos educativos y de una forma más compacta.

En esta comparación se manifiesta la importancia de proveer un lenguaje de búsqueda simple. Igualmente se muestra como en un "mundo" tan reciente como los Web Services se está trabajando con lenguajes, como PQL, que persiguen objetivos similares al aquí propuesto.

El lenguaje propuesto en nuestro proyecto, se concibió al estilo de un SQL simplificado (con elementos semiestructurados básicos), debido a la popularidad y relativa facilidad de dicho lenguaje, es de alto nivel, de complejidad computacional baja y está basado en elementos de búsqueda de más bajo nivel. Este "azúcar sintáctico" *facilita la expresión de muchas consultas*, las cuales sería posible expresar mediante el uso de un lenguaje de más bajo nivel, pero el costo de su escritura y aprendizaje sería muy alto.
Por supuesto y como lo expresa [Klein 02] "se requiere que el usuario se familiarice con el lenguaje...". Igualmente el uso de una interfaz deberá facilitar aun más esta labor y el prototipo así lo ha evidenciado.

### 4.3 El aspecto de los metadatos extensibles

Como se explicó en la sección 3.5.2 a diferencia de los proyectos de [El Saddik 01] cuya contribución fue la de extender LOM con 4 categorías adicionales y 16 propiedades para recursos multimediales y [Suthers 01] que extendió LOM para el sistema educativo K-12; en nuestro proyecto no se han predefinido todo los posibles metadatos que los recursos educativos puedan tener, en vez de ello, se ha dejado abierta la posibilidad para que en el evento de que surjan metadatos (aparte de aquellos del LOM básico que se han preservado para garantizar un núcleo estandarizado) que se consideren necesarios para la documentación, estos les puedan ser incorporados directamente por el autor sin la intervención de un DBA por ejemplo.

El lenguaje propuesto soporta y manipula estos metadatos "dinámicos". Independientemente de cómo se ha logrado esta característica (en este proyecto se logró usando bases de datos semiestructuradas vía XML, igualmente se pudo haberlo hecho vía bases de datos tradicionales) lo importante es ofrecer esta flexibilidad de documentación y manipulación de una manera "trasparente" para el lenguaje y para el usuario, es decir, aunque el esquema de cada CE puede variar con respecto a los demás, el lenguaje no cambia (por ejemplo los operadores *Get Components* y *Update* muestran la *extensibilidad del lenguaje*, véase secciones 3.11.2 y 3.11.4*).* Por supuesto que dependerá de un grupo el definir y mantener los metadatos adicionales que deben ser incorporados a determinados recursos.

Un aspecto menor es el de haber contribuido con una herramienta amigable para el ingreso de metadatos, dando respuesta a la necesidad planteada en [BECTA 01]: "*Las herramientas para asignar metadatos a los recursos educativos están surgiendo lentamente*".(ver sección 2.2.2.3.3)

## 4.4 Otros Aportes

### 4.4.1 Introducción de lógica descriptiva para los recursos educativos

La introducción de la LD ha dotado al lenguaje de consulta de más expresividad. Esta técnica no había sido incorporada al mundo de los recursos educativos. Lo más cercano que se encontró fue su introducción en el mundo de los Web Services como se presenta en [Hacid 02]. La inclusión del operador de *subsumption* permite **formalizar** la búsqueda de CE's que incluyan el desarrollo de determinados conceptos con ciertos roles. Existe un nivel de trasparencia alto para el usuario ya que en su forma más simple (véase sección 3.15) sólo se requiere establecer cuales conceptos se desean estudiar y basándose en el perfil del usuario (*learner group*) el operador automáticamente busca los CE's que satisfagan la consulta. Adicionalmente con el objetivo de flexibilizar el uso del operador, el usuario puede anular los roles que por defecto le son asociados por medio de su perfil y emitir la consulta con los roles y conceptos que desee.
Se pueden introducir otros operadores de la LD tal y como se menciona en la sección 6.

### 4.4.2 Fusión del concepto componente (en el sentido de componente de software) con los recursos educativos y explotación de tal modelo

Se ha planteado un modelo de CE's basado en conceptos del mundo de los componentes de software. El concepto de componentes educativos aparece en la literatura como un sinónimo de recurso educativo, el concepto que aquí se ha desarrollado va más allá porque dota a un recurso educativo de una interfaz y de metadatos. Igualmente se da nacimiento al concepto de CEC, lo cual permite la reutilización, característica fundamental de los modelos de componentes.
Para garantizar que el modelo propuesto cumple con las características del mundo de los componentes se establece la siguiente comparación basada en [Birngruber 01] donde se establecen **los requerimientos o elementos mínimos (y deseables) que debe poseer un modelo para ser considerado de componentes:**

1. *Conexión orientada a la composición*

Es un aspecto fundamental el conectar componentes (esto les permite interactuar y "comunicarse" con otros). Estos deben describir como pueden ser conectados. En nuestra propuesta, el nivel de

granularidad (ver sección 3.3.3.1 y 3.4) usado minimiza los requerimientos de conexión. (ver adicionalmente la restricción de secuencia, sección 3.8.1.1)

*2. Composición basada en agregación*

Nuestro modelo ofrece mecanismos para agregar componentes a un nivel mas alto y para integrar las interfaces de los componentes agregados.

*3. Reutilización*

En nuestro caso a través de los CEC's se reutilizan contenidos educativos presentes en los CEP's. Un CEP puede ser utilizado en muchos CEC's, e incluso independientemente.

*4. Construcción recursiva de componentes*

Un CEC es construido con base a CEP's u otros CEC'S (ver sección 3.8). Los autores pueden crear lecciones, cursos, workshops etc. (véase sección 3.9)

*5. Independencia de la plataforma*

Dado que se utilizó una especificación para los CEC's en XML, esta facilita la independencia de la plataforma y el intercambio de CEC's a través de aplicaciones para la web.

*6. Minimizar la necesidad de nuevas herramientas*

De nuevo, la descripción basada en XML minimiza este aspecto. Además se desarrolló una interfaz "fácil" de usar para los operadores de creación y composición fundamentalmente. Esta tarea es muy importante es nuestro trabajo, porque la creación de CEC's debe llegar a ser una tarea realizable por no expertos en computación.

*7. Reducir el proceso de aprendizaje del modelo de componentes*

La creación de una interfaz amigable que permite la creación de CEC's se espera contribuya de manera significativa en este aspecto. El lenguaje de creación es trasparente para el autor. Igualmente se pueden construir interfaces que hagan más amigables los otros operadores (ver características del prototipo en la sección 5). Además se utilizó LOM el cual es una de las propuestas más importantes para documentar a los recursos educativos y es bastante "intuitivo".

*8. Soporte de versiones*

Directamente nuestro modelo no maneja este aspecto. Sin embargo un CE puede ser la base para crear otro CEC que puede jugar el papel de una nueva versión. Igualmente se dispone del ME donde

podría documentarse lo anterior, aparte de algunos de los elementos (por ejemplo *contribute*) del LOM básico que pueden servir también para tal propósito.

### 4.4.3 Clasificación automática

Tal y como se menciona en [Klein 02], "*idealmente debería existir una clasificación automática de los servicios"*, **en nuestro trabajo se ha contribuido con la clasificación automática de un CEC a través de sus conjuntos $R_{prerequisite}$ y $R_{subject}$.** El conjunto $R_{prerequisite}$ de un CEC siempre se genera a partir de sus CEP's mientras que el $R_{subject}$ se realiza en forma automática siempre y cuando sea posible y lógico hacerlo (ver seccines 3.8.1 y 3.8.2**). Esto *simplifica* el trabajo para el creador de un CEC.**

### 4.4.4 Contribución de los operadores propuestos para el proceso de adaptación

En el apéndice 2 se puede observar como varios de los operadores concebidos aquí contribuyen de manera significativa al proceso de adaptación del sistema global educativo del cual hace parte este trabajo. En concreto los operadores *Find Similar*, *Get Components*, *Show* (Prerequisites), *Replace* y *GetPaths*; proveen elementos para guiar el proceso de adaptación tal y como lo muestran las figuras AP2 y AP3 del apéndice. De acuerdo a la literatura analizada lo más cercano a operadores que ayuden con el aspecto de adaptación fueron los de construcción de triggers de AHA (véase sección 2.2.3.3.2). Los operadores concebidos contribuyen a la automatización de la adaptación, confróntese frente a la escritura de guía dirigida que debe realizarse en AHA por ejemplo.

# 5 Presentación del prototipo

## 5.1 Introducción

No es el objetivo del prototipo implementar todo el modelo, tampoco lo es concebir una herramienta profesional para un ambiente de producción. Se ha concebido una herramienta sencilla, funcional y que permite probar los conceptos fundamentales. Se describe a continuación la funcionalidad, los aspectos técnicos y algunas conclusiones al respecto.
Una guía completa con un ejemplo de creación paso a paso de CE's por el prototipo en formato HTML se encuentra en el CD que acompaña a este trabajo (archivo tour.html). Igualmente todo el código fuente se encuentra disponible en el CD, en la carpeta software.

## 5.2 Funcionalidad

El objetivo principal del prototipo es implementar el modelo de CE's y algunos de sus operadores. El prototipo permite la creación de CEP's, CEC's y manipularlos con algunos de los operadores propuestos. Más que un manual de usuario, se describe lo que es posible realizar con la herramienta.

### 5.2.1 La creación de un CEP

La creación de un CEP se lleva a cabo en una interfaz web (ver figura 5.1) a través de la cual el autor documenta el recurso educativo que va a matricular en el sistema para convertirlo así en un CE.
El autor deberá ingresar la información de los metadatos, las relaciones con el árbol de dominio ($R_{subject}$ y $R_{prerequisite}$) y documentar las entradas y salidas de su recurso.
Para la entrada de la información se ha tratado de minimizar el "peso de documentar" mediante el uso de listas de selección, *radio buttons*, llenado de atributos por defecto etc.; ya que el ambiente está dirigido a autores de diversas áreas y debe ser amigable.
El factor "amigable" es difícil de medir y es algo subjetivo. Sin embargo se han empleado los elementos típicos que se supone hacen amigables a las herramientas de computación.

### 5.2.2 La creación de un CEC

La labor de creación de un CEC es mucho más compleja. En este caso el autor se enfrenta a una interfaz gráfica (ver figura 5.2) donde puede dibujar el grafo de composición que representa su CEC. El trabajo de creación se realiza mediante iconos y uso del mouse (arrastrar y soltar)**. Las instrucciones de creación son transparentes para el autor**, estas son generadas internamente por el sistema.

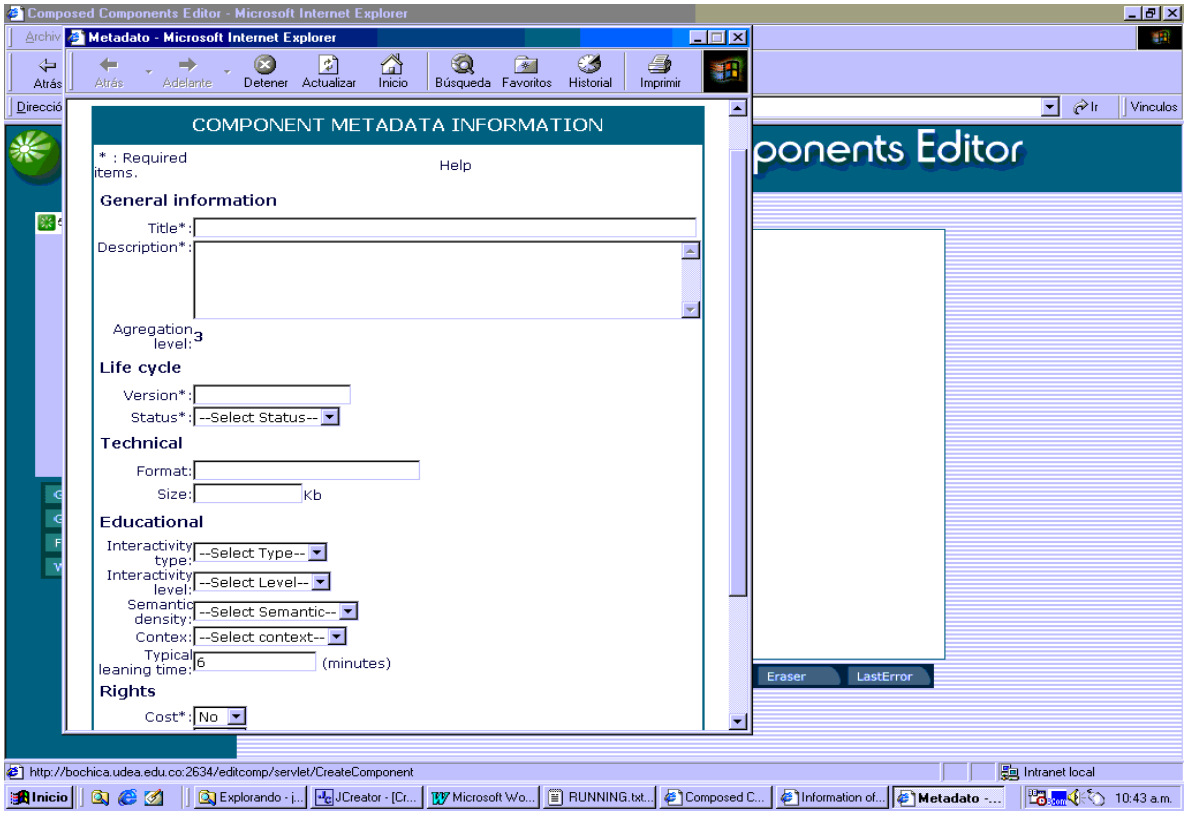



Figura 5.1: Interfaz web para la creación de un CEP

Como se muestra en la figura 5.2, en el panel izquierdo se visualizan todos los CE's existentes en la base de datos. Haciendo click derecho sobre cualquier CE el autor tiene acceso a toda la información del CE (en forma trasparente está haciendo uso del operador **Show**), para ello se despliega una ventana en la cual se puede elegir que se desea observar: metadatos, las entradas, las salidas, el conjunto $R_{subject}$, el conjunto $R_{prerequisite}$ o toda la información del CE. Esta información proporciona elementos al autor para determinar si un CE le es útil en la creación de su CEC.

Igualmente por medio de los botones que se observan en la parte inferior izquierda el autor puede hacer consultas sobre los metadatos de los CE's. Si se pulsa el botón *Get All Components* el panel izquierdo se poblará con todos los CE's. Sin embargo si el autor pulsa el botón *Get Components By*, se le despliega una ventana (véase figura 5.3) en la cual el autor puede realizar consultas sobre los metadatos y las otras secciones de los CE's mediante selección de las categorías deseadas y escribiendo la condición en la caja de diálogo. Lo único que el autor escribe es la condición, el sistema de forma trasparente construye internamente la sentencia **Get Components** correspondiente.

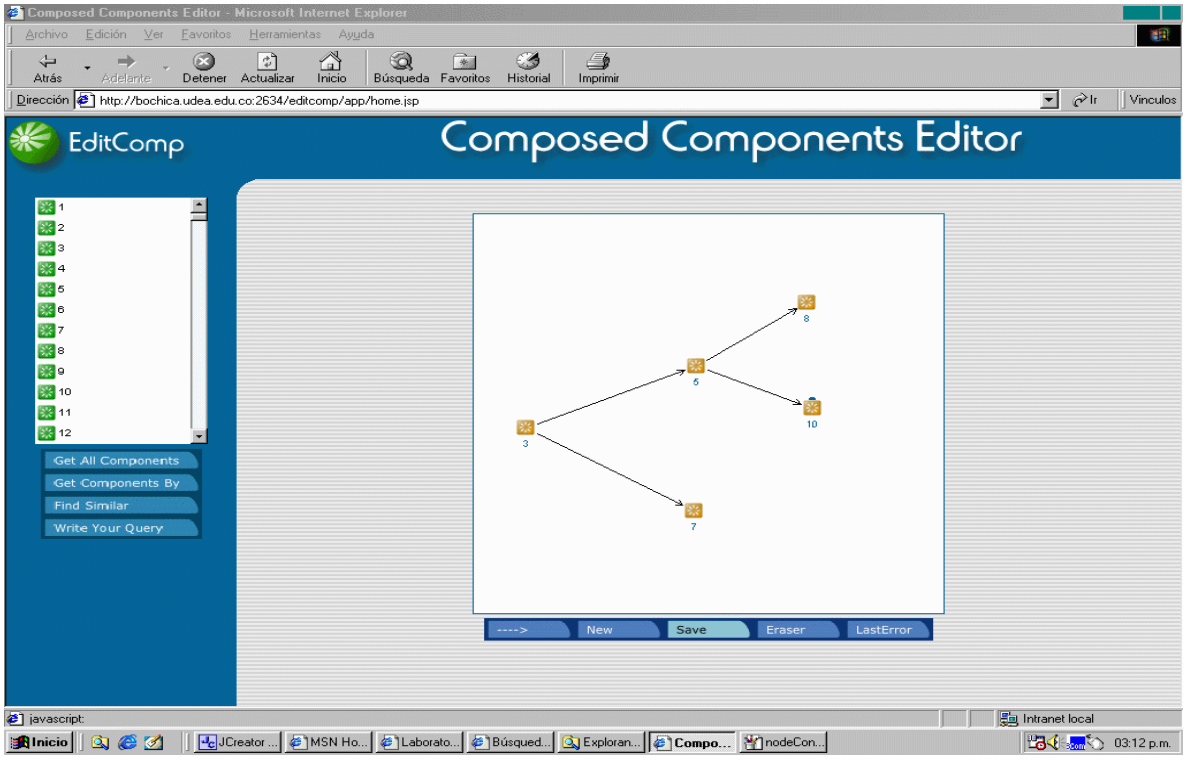



Figura 5.2: Interfaz web para la creación de un CEC

Por medio de los iconos (“pestañas”) que se observan en la parte inferior de la figura 5.2, el autor construye el grafo de composición. Haciendo click izquierdo sobre un CE del panel izquierdo, este se ubica en el panel de edición (panel central). Mediante el icono de la flecha se construye un arco entre dos CE’s.

Finalmente cuando el autor considera terminado su grafo de composición, se realizan dos verificaciones fundamentales: **La ausencia de ciclos y que el grafo dibujado sea conectado** pulsando el botón *Save*.

Si se cumple lo anterior se abre el formulario de metadatos (ver figura 5.1) el cual debe ser llenado en igual forma que para un CEP. Terminado este proceso el sistema genera la instrucción **create component** correspondiente con todos los parámetros y se crea el nuevo CEC. La instrucción generada se construye con base en los operadores **Sequence** y **Union**

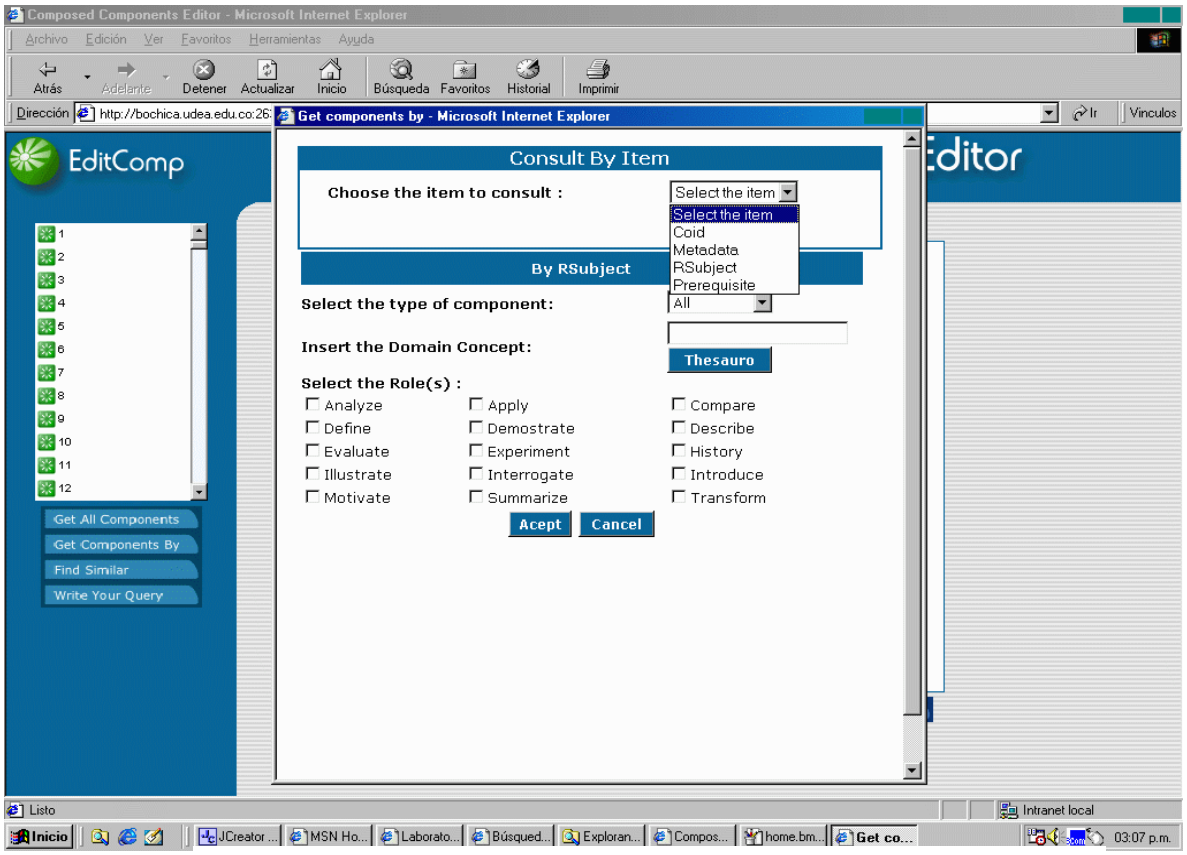


Figura 5.3: Interfaz web para el operador Get Components

### 5.2.3 Los operadores de manipulación

Como se explicó en la sección 5.2.2, por medio de los operadores *Show* y *Get Components*, la información de los CE's puede ser explorada. Al operador *Show* se tiene acceso haciendo click derecho sobre un CE. Para elaborar una consulta mediante el operador *Get Components* se tienen varias alternativas:

- Mediante el botón *Get All Components* se pueden traer todos los CE's
- Mediante el botón *Get Components By* se pueden construir consultas especificando condiciones simples mediante un menú (un pequeño wizard). (Ver figura 5.3.)

Las dos anteriores alternativas son útiles en especial para autores nuevos en el sistema, los autores avanzados podrían sacar provecho del botón *write your query*, mediante el cual pueden construir la consulta que deseen mediante el operador *Get Components*.

En este momento y como decisión de implementación dicho operador se construyó basado en el lenguaje XPath (véase sección 2.3.6). En una futura versión del prototipo una interfaz amigable deberá ser construida para la construcción de consultas **complejas** (véase sección 6).

Los operadores de *Delete* y *Update* no fueron incorporados al prototipo, ya que aunque son necesarios, estas no son las operaciones más frecuentes. Igualmente en una futura versión del prototipo deberían ser implementadas.

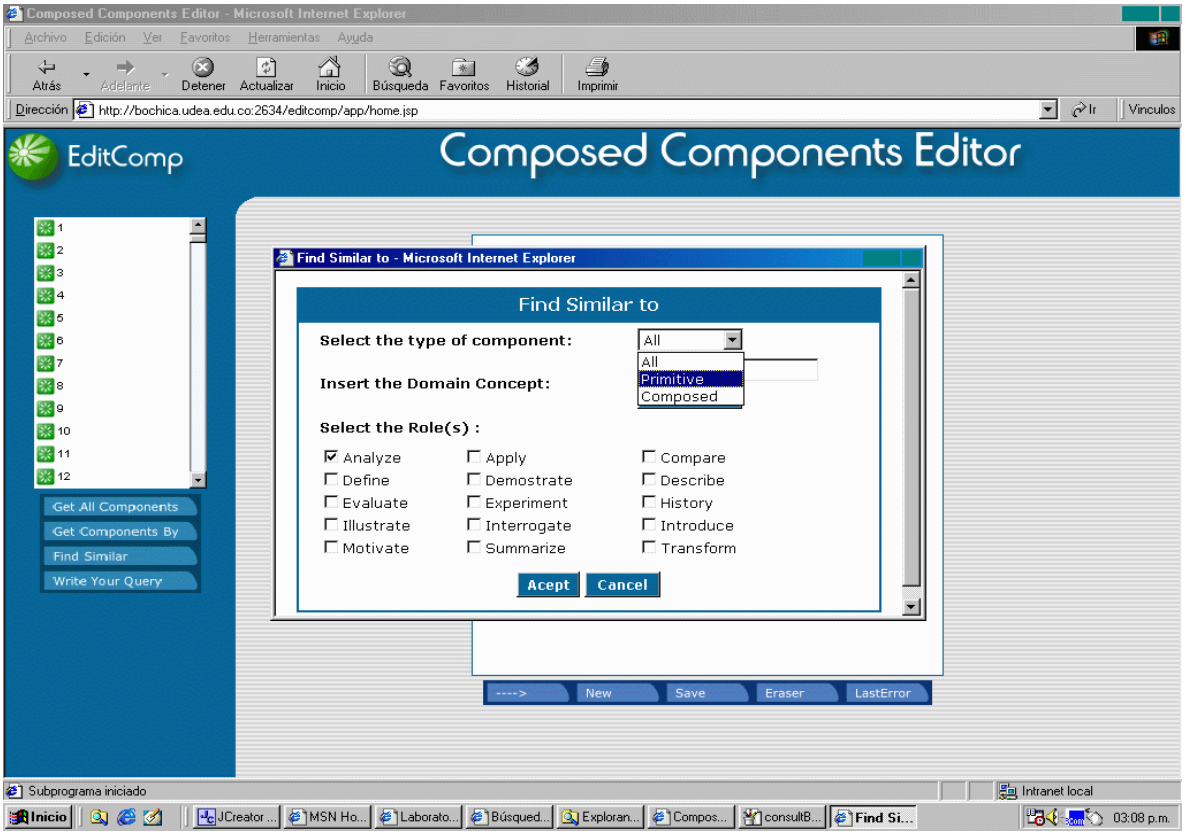



Figura 5.4: Interfaz web para el operador Find Similar

### 5.2.4 Sobre los operadores Find Similar, Replace y GetPaths

El objetivo de estos operadores es contribuir en el proceso de adaptación (véase apéndice 2). A través de la interfaz (ver figura 5.4) se puede usar el operador *find similar* de manera trasparente para el usuario pulsando el botón *find similar*.

Respecto al aspecto del nivel de dificultad de un CE, en el operador *Replace* este se implementó tomando como base el número de prerrequisitos de cada CE (es decir un CE se considera más difícil que otro si tiene mas prerrequisitos, véanse secciones 3.14.1 y 6).

## 5.3 Aspectos técnicos

El sistema fue concebido para ser usado en la web. Se utilizó una base de datos Oracle 8i para almacenar la información de los CE's y lenguaje de programación Java. Los CE's fueron definidos mediante XML (Los DTD's definidos son entregados con el prototipo).

Para la creación de formularios se utilizaron *applets*, los cuales dieron el manejo gráfico requerido y *servlets* para acceder a la base de datos. Las librerías Java utilizadas fueron:

| | |
|---|---|
| *org.w3c.dom.*;* | // Para manipular xml |
| *org.xml.sax.*;* | // Para manipular xml |
| *javax.xml.parsers.*;* | // Para manipular xml |
| *org.apache.xpath.*;* | // Para manejar xpath |
| *java.util.StringTokenizer;* | // Para manejar un tipo especial de String |
| *javax.servlet.jsp.JspWriter;* | // Para escribir en servlets |
| *javax.servlet.*;* | // Panejo de servlets |
| *java.io.PrintWriter;* | // Para escribir en servlets |
| *java.io.IOException;* | // Para manejo de excepciones |
| *java.sql.ResultSet;* | // Para manejar el Resultset de una consulta de la base de datos |
| *java.sql.SQLException;* | // Para manejar excepciones en consultas a base de datos |

## 5.4 Conclusiones

El objetivo principal del prototipo es mostrar la factibilidad del modelo propuesto. Se implementaron los operadores que mayor funcionalidad ofrecen, siendo fundamental por supuesto el operador de creación de CEC's el cual requirió una interfaz web completa, la cual hace transparente el proceso de creación de un CEC. Se crearon CE tanto primitivos como compuestos y se realizaron consultas sobre ellos mediante los operadores propuestos lo cual ha servido para evidenciar varios de los aspectos enunciados en la sección sobre el aporte (sección 4) y de definición del modelo (sección 3).

Aunque no se trata de una herramienta profesional, destinada a un ambiente de producción, se espera haber dado un primer paso en la construcción de una herramienta más sólida y robusta.

# 6 Conclusiones y futuras líneas de trabajo

## 6.1 Conclusiones

Se ha propuesto un modelo *simple* para la manipulación de recursos educativos. El concepto de *simple* se refiere principalmente a que el modelo pueda ser entendido por un profesor perteneciente a cualquier área del conocimiento.
El prototipo construido y las pruebas realizadas así lo han evidenciado, aunque por supuesto se requiere un mayor trabajo desde el punto de vista de las interfaces gráficas.

Es posible concebir modelos complejos, que involucren muchos más elementos, pero los recientes desarrollos para la web han mostrado que sólo los modelos sencillos tienden a tener aceptación entre el público. (La idea general es que con un mínimo de clicks el usuario obtenga lo que desea)

El éxito de un sistema de este tipo no puede ser garantizado, tal y como lo expresa [Roschelle 99] se requiere un *proceso de evangelización* que de a conocer los beneficios potenciales que se pueden derivar de un sistema a gran escala de componentes (**reutilización de contenidos)**.

Respecto a la labor de documentación, esta es por supuesto una tarea pesada pero necesaria si se desea alimentar al sistema con información que se puede explotar posteriormente. Sólo una política implantada en un grupo de trabajo pedagógico puede ayudar a garantizar esta labor, **la cual se ha tratado de minimizar mediante formularios sencillos y llenado automático** (valores por defecto) siempre y cuando sea posible.

El lenguaje de manipulación propuesto junto con el ambiente de trabajo deben ser aprendidos por los usuarios no expertos en computación, si en verdad se quiere minimizar la presencia del programador de aplicaciones. De lo contrario no importa cuán amigable o profesional sea el sistema, seguirá existiendo dicha dependencia.

## 6.2 Futuras líneas de trabajo

- El lenguaje de consulta se puede hacer más expresivo: Permitir ordenamientos y agrupamientos (similar al operador Group By del SQL) de CE's por elementos de los metadatos, joins entre CE's, entre otros.

- Se requiere un trabajo más profundo para determinar el nivel de dificultad de un CE. Por ello se deja abierta la posibilidad de que se calcule mediante un procedimiento que se formalice.
- Aunque este no es un aspecto a desarrollar desde el punto de vista teórico o de aporte, en una futura versión se podrían implementar los operadores que quedaron faltando en el prototipo y realizar un trabajo de interfaz gráfica profesional.
- Un estudio de rendimiento se hace necesario, por ejemplo tratar aspectos como el indexamiento de los CE's, minimizar el gasto de espacio en disco, incrementar la seguridad etc.
- Realizar una comparación entre los modelos semiestructurado y relacional dejando claras las ventajas y desventajas de ambos modelos y bajo que criterios se debe usar el uno o el otro.
- La labor de adición y/o eliminación de una entrada o salida de un CE tal y como se explicó en la sección 3.11.4.1, es decir este proceso debe ir acompañado del cambio del recurso educativo asociado.

**Apéndice 1: Los metadatos del LOM**

Véase además [Web Page 16] y sección 2.2.2.3.3.

| Nro. | Nombre | Explicación | #Ocurren-cias | Rango de Valores |
|---|---|---|---|---|
| 1 | General | Información general que describe al RE. | 1 | - |
| 1.1* | Identifier | Identificador único del RE. | 1 | - |
| 1.2 | Title | Nombre del RE | 1 | |
| 1.3* | Catalog Entry | Describe al RE de acuerdo a algún sistema de catálogo establecido | Max 10 | - |
| 1.3.1* | Catalog | Nombre del catálogo | 1 | - |
| 1.3.2* | Entry | Valor de la entrada en el catálogo correspondiente | 1 | - |
| 1.4 | Language | Lenguaje en el que fue elaborado el RE | Max 10 | - |
| 1.5 | Description | Descripción breve del contenido del RE | Max 10 | - |
| 1.6 | Keywords | Palabras o frases que describen al RE | Max 10 | - |
| 1.7* | Coverage | Gama de elementos tales como región, cultura etc. propios del RE | Max 10 | - |
| 1.8* | Structure | Estructura subyacente del RE | 1 | Collection, Mixed, Linear, Hierarchical, Networked, Branched, Parceled, Atomic |
| 1.9 | Aggregation Level | Granularidad del RE | 1 | 1= Atomos Ej: Fotos, Doc Texto simple etc.<br>2= Colección de átomos. Ej: Doc HTML.<br>3= Coleccíon de elementos de nivel 2. Ej: Doc HTML con enlaces hacia otros.<br>4= Conjunto de elementos de nivel 3. |
| 2 | Life Cycle | Describe la evolución del RE | 1 | - |
| 2.1 | Version | Versión del RE | 1 | - |
| 2.2 | Status | Estado del RE | 1 | Draft, Final, Revised, Unavailable |

| | | | | |
|---|---|---|---|---|
| 2.3 | Contribute | Personas que cambian al RE | Max 30 | - |
| 2.3.1 | Role | Tipo de Contribución | 1 | Author, Publisher, Unknown, Inititiator, Validator, Terminator, Editor, Graphical Designer,Tecnical Implementer, Content Provider, Technical Validator, Educational Validator, Scrip Writer, Instructional Designer. |
| 2.3.2 | Entity | Naturaleza del contribuyente | Max 40 | Ej: Si Role es Author, entonces Entity es Person.<br>Si Role es Publisher, entonces Entity es Organisation (University, Department etc.) |
| 2.3.3 | Date | Fecha de la contribución | 1 | - |
| 3* | Metametadata | Información sobre la creación de los metadatos | 1 | - |
| 3.1* | Identifier | Identificador único del registro de MM | 1 | - |
| 3.2* | Catalog Entry | Describe al MM de acuerdo a algún sistema de catálogo establecido | 1 | - |
| 3.2.1* | Catalog | Nombre del catálogo | 1 | - |
| 3.2.2* | Entry | Valor de la entrada en el catálogo correspondiente | 1 | - |
| 3.3* | Contribute | Personas que cambian al MM | Max 10 | |
| 3.3.1* | Role | Tipo de Contribución | 1 | Creator, Validator |
| 3.3.2* | Entity | Naturaleza del contribuyente | 1 | Idem 2.3.2 |
| 3.3.3* | Date | Fecha de la contribución | 1 | - |
| 3.4* | Metadata Scheme | Nombre y versión de la especificación usada | 1 | Rango provisto por la ISO/IEC 10646 1 |
| 3.5* | Language | Lenguaje en el cual se escribe el registro MM | 1 | - |
| 4 | Technical | Características técnicas del RE | 1 | - |
| 4.1 | Format | Tipos de todos los elementos que conforman al RE | Max 40 | MIME Types (Ver RFC2048) |
| 4.2 | Size | Tamaño del RE en Bytes | 1 | - |
| 4.3 | Location | Localización del RE.( Ej: URL) | Max 10 | Rango provisto por la ISO/IEC 10646 1 |

| | | | | |
|---|---|---|---|---|
| 4.4* | Requirements | Características requeridas para usar el RE | Max 40 | - |
| 4.4.1* | Type | Tipo de Tecnología requerida por el RE. | 1 | Browser, Operating System |
| 4.4.2* | Name | Nombre de la tecnología requerida | 1 | Ej: Si Type=Browser entonces: Netscape Communicator, Opera etc. |
| 4.4.3* | Minimun Version | Mínima versión de la tecnología requerida | 1 | Rango provisto por la ISO/IEC 10646 1 |
| 4.4.4* | Maximun Version | Máxima versión de la tecnología requerida | 1 | Rango provisto por la ISO/IEC 10646 1 |
| 4.5* | Installation Remarks | Descripción de cómo instalar el RE | 1 | - |
| 4.6* | Other Platform Requirements | Información sobre otros requerimientos | 1 | - |
| 4.7* | Duration | Tiempo que dura un RE (Por ej. un vídeo) | 1 | - |
| 5 | Educational | Describe las características pedagógicas de un RE | 1 | - |
| 5.1 | Interactivity Type | Tipo de interactividad entre el RE y el usuario | 1 | Active, Expositive, Mixed, Undefined |
| 5.2* | Learning Resource Type | Tipo de RE | Max 10 | Exercise, Simmulation, Questionnaire, Diagram, Figure, Graph, Index, Slide, Table, Narrative Test, Exam, Experiment, Problem Statement, Self Assesssment |
| 5.3 | Interactivity Level | Grado de interactividad entre el RE y el usuario | 1 | Very Low, Low, Medium, High, Very High |
| 5.4 | Semantic Density | Cantidad de información que el RE contiene en proporción a su tamaño o duración | 1 | Very Low, Low, Medium, High, Very High |
| 5.5* | Intended End User Role | Tipos de usuarios a los cuales va dirigido el RE | Max 10 | Teacher, Author, Learner, Manager |
| 5.6 | Context | Entorno principal donde será usado el RE | Max 10 | Primary Education, Secondary Education, Higher Education, University First Cycle, University Second Cycle, University Postgrade, Technical School First Cycle, Technical School Second Cycle, Professional |

| | | | | |
|---|---|---|---|---|
| | | | | Formation, Continuos Formation, Vocational Training |
| 5.7* | Typical Age Range | Rango de edad para el cual se considera adecuado el RE | Max 5 | - |
| 5.8* | Difficulty | Nivel considerado de dificultad del RE con respecto a su audiencia | 1 | Very Easy, Easy, Medium, Difficult, Very Difficult |
| 5.9 | Typical Learning Time | Tiempo aproximado que se requiere para trabajar con el RE | 1 | - |
| 5.10* | Description | Comentarios respecto a como va a ser usado el RE | 1 | - |
| 5.11* | Language | Lenguaje típico usado por la audiencia que experimenta el RE | Max 10 | - |
| 6 | Rights | Características de propiedad intelectual | 1 | - |
| 6.1 | Cost | Si el uso del RE es gratuito | 1 | Yes, No |
| 6.2 | Copyright and Other Restrictions | Si leyes de Copyright u otras restricciones aplican al uso del RE | 1 | Yes, No |
| 6.3 | Description | Comentarios sobre las condiciones de uso del RE | 1 | - |
| 7* | Relation | Describe la relación (si la hay) entre un RE y otro RE | Max 100 | - |
| 7.1* | Kind | Tipo de la relación | 1 | IsPartOf, HasPart, IsVersionOf, HasVersionOf, IsFormatOf, HasFormat, References, IsReferencedBy, IsBasedOn, IsBasisFor, Requieres, IsRequieredBy |
| 7.2* | Resource | El tipo de objeto requerido por la relación establecida | 1 | - |
| 7.2.1* | Identifier | Identificador del RE que establece la relación | 1 | - |
| 7.2.2* | Description | Descripción de 7.2 | 1 | - |
| 7.2.3* | Catalog Entry | Idem 1.3 | 1 | - |
| 8* | Annotation | Comentarios sobre el uso del RE | Max 30 | |
| 8.1* | Person | Persona que hace el comentario | 1 | - |

| | | | | |
|---|---|---|---|---|
| 8.2* | Date | Fecha del comentario | 1 | - |
| 8.3* | Description | Contenido del comentario | 1 | - |
| 9* | Classification | Describe la ubicación del RE dentro de un sistema de clasificación particular | Max 30 | - |
| 9.1* | Purpose | Propósito de clasificar este RE | 1 | Discipline, Idea, Prerequisite, Educational Objective, Accessibility Restrictions, Educational Level, Skill Level, Security Level |
| 9.2* | Taxon Path | Describe un camino de clasificación del RE en un sistema de clasificación establecido | Max 15 | - |
| 9.2.1* | Source | El nombre del sistema de clasificación | 1 | Rango provisto por la ISO/IEC 10646 1 |
| 9.2.2* | Taxon | Describe un término particular dentro de una taxonomía | Max 15 | |
| 9.2.2.1* | Id | Identificador de 9.2.2 | 1 | Rango provisto por la ISO/IEC 10646 1 |
| 9.2.2.2* | Entry | Nombre textual de 9.2.2 | 1 | - |
| 9.3* | Description | Descripción del RE respecto a 9.1 | 1 | - |
| 9.4* | Keywords | Palabras y frases claves del RE con respecto a 9.1 | Max 40 | - |

**Notas:**

1. En este trabajo no se utilizan los ítems marcados con * (véase sección 3.5.1 al respecto).

2. El ítem 1.6 Keywords es remplazado por $R_{subject}$.

3. El ítem 1.9 Aggregation Level puede tener dos posibles valores: 2 (CEP) y 3 (CEC).

4. El ítem 4.3 Location será nulo si Aggregation Level representa un CEC.

**Apéndice 2: Conexión con el módulo adaptativo**

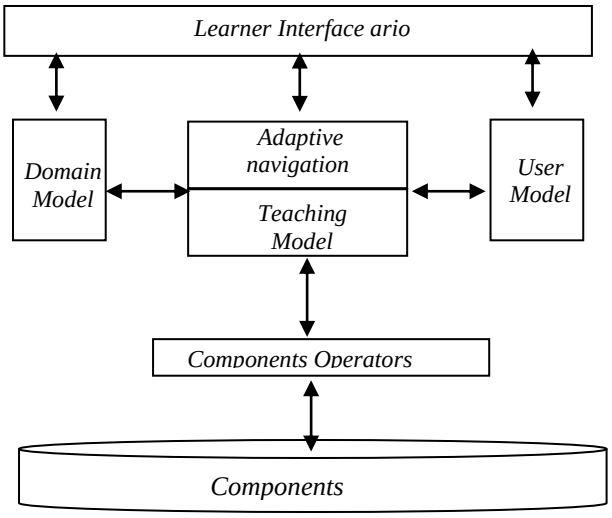
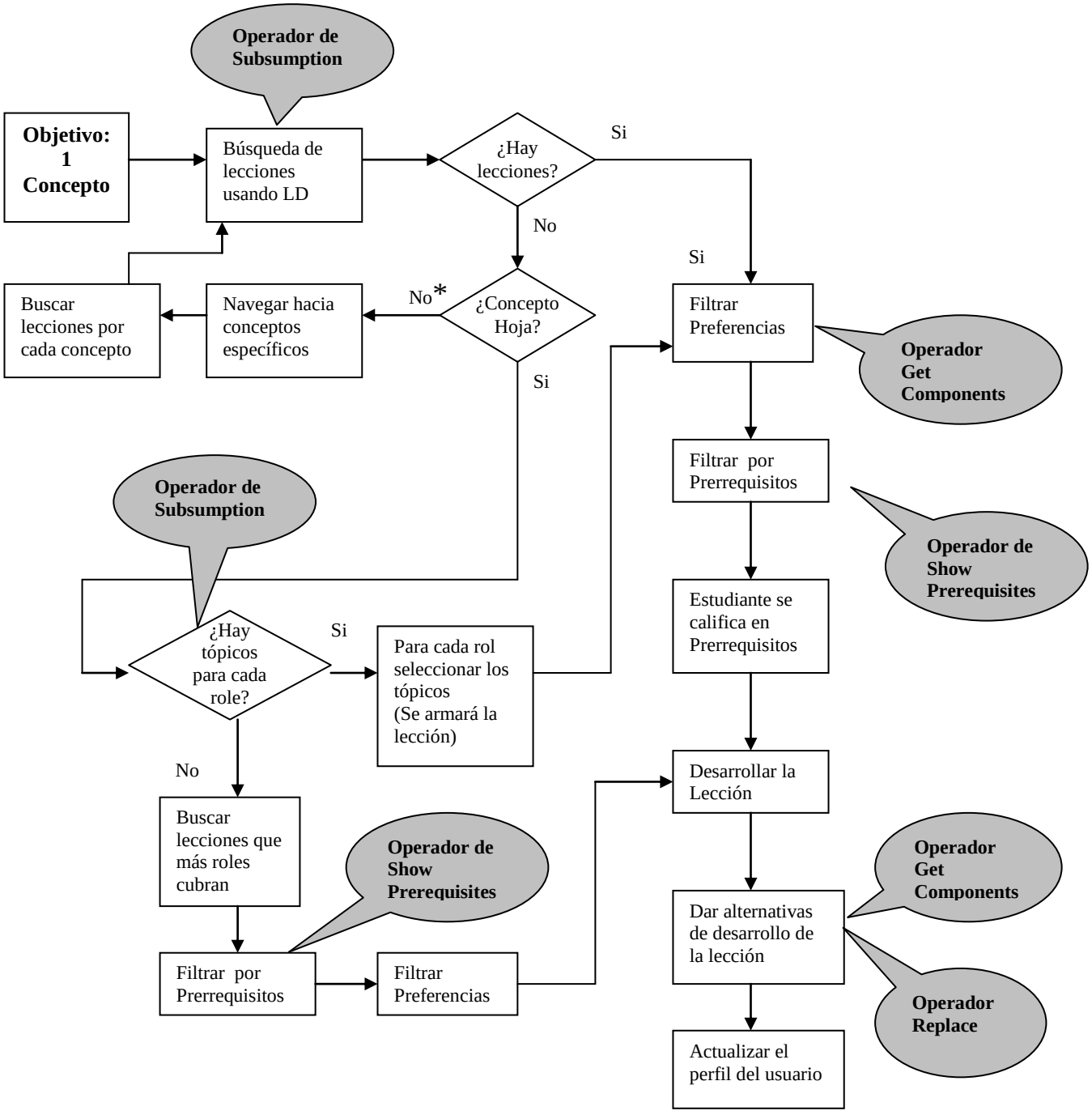



Figura AP1: Arquitectura del Sistema Global

La figura AP1 muestra la arquitectura del sistema de autoaprendizaje, los CE's, sus operadores y el modelo del dominio fueron definidos en la sección 3.
En este apéndice se mostrará como contribuyen estos módulos con el resto del sistema.

Existe un modelo del usuario el cual contiene información que incluye objetivos, preferencias y conocimientos previos, con base en estos elementos los CE's son instanciados para cada usuario.
Cada usuario pertenece a un *learner group* que consta de roles, por ejemplo pueden existir los siguientes grupos predefinidos:
*Student:* Usa los roles Define, Example y Evaluate
*Investigator*: Usa los roles Define, Apply y Experiment.
*Employee:* Usa los roles Define y Motivate

Respecto al Teaching Model (Modelo de aprendizaje) se usarán dos métodos: *Basado en objetivos* y *basado en cursos*. En el primero el estudiante selecciona el o los objetivos deseados. Cada objetivo estará expresado como un concepto tomado desde el modelo del dominio. El sistema buscará el o los CE's que permitan llenar estos objetivos, con el mínimo de información extra (prerrequisitos) y teniendo presente que en lo posible el aprendiz esté preparado para enfrentar estos contenidos. Para la segunda modalidad, la basada en cursos, el sistema buscará ofrecer el máximo de ayudas para permitir al estudiante cumplir los objetivos definidos para el curso. Por ejemplo suministrando información acerca de prerrequisitos no cumplidos ó módulos complementarios para cubrir tópicos relacionados con el curso. Por ello los prerrequisitos no constituyen un impedimento para que un estudiante aborde un CE, se consideran mas bien una guía.
La siguientes figuras muestran el proceso de adaptación y para los propósitos de nuestro proyecto se muestra cómo (en que punto) y cuales operadores contribuyen en el proceso.




* Este paso sólo se realiza para los hijos directos de *un concepto no hoja*. (es decir no se realiza para descendientes de sus hijos)

Figura AP2: Resumen del Proceso de Adaptación: 1 Concepto

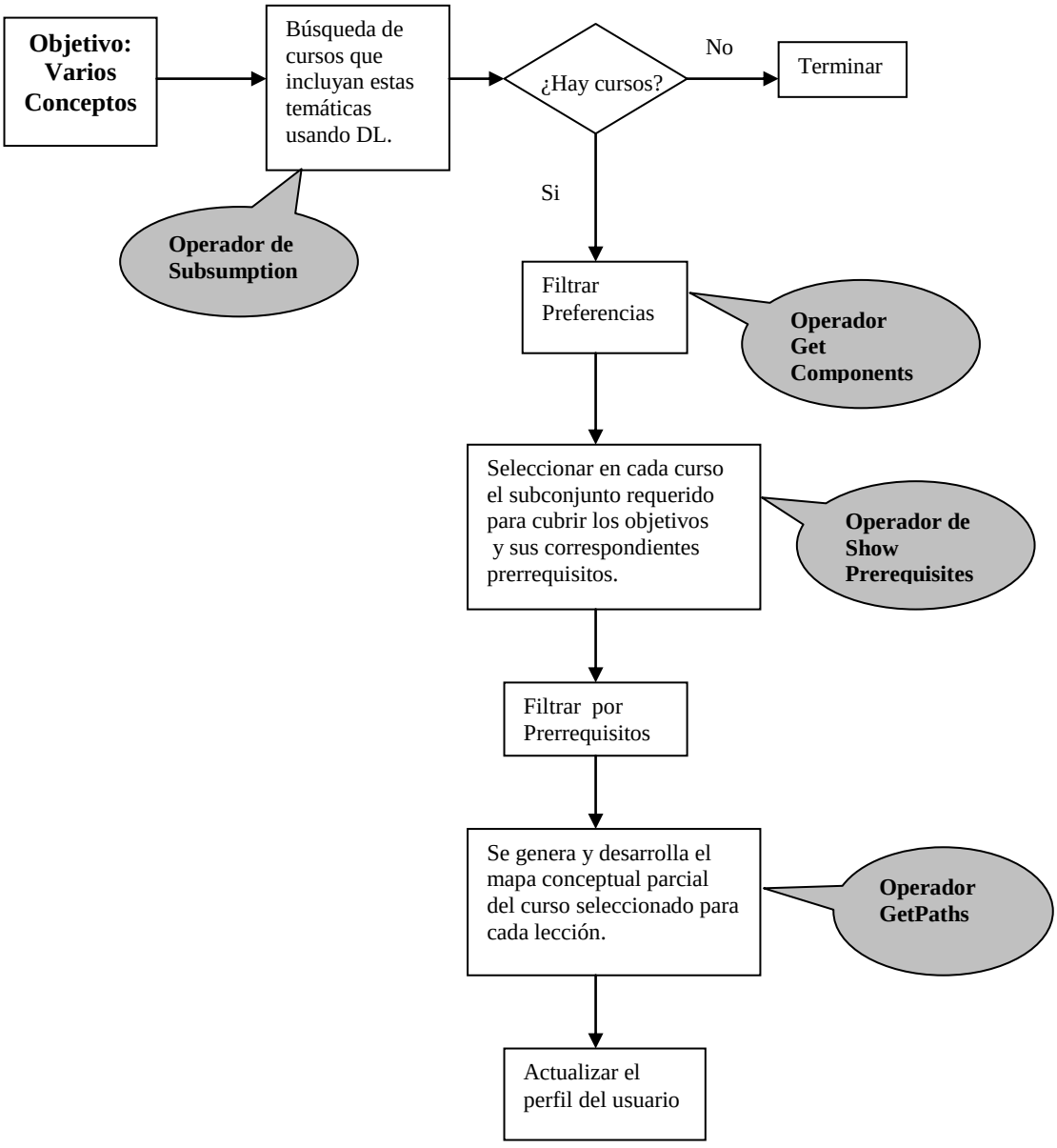



Figura AP3: Resumen del Proceso de Adaptación: Varios Conceptos

## Apéndice 3: Gramática BNF del lenguaje

### Operadores

1. <operadores> → <opget>
2. <operadores> → <opdel>
3. <operadores> → <opupdate>
4. <operadores> → <opshow>
5. <operadores> → <opreplace>
6. <operadores> → <opfind>
7. <operadores> → <opcreate>

### Operador get components

8. <opget> → **get components**<resto-opget>
9. <resto-opget> → **Є**
10. <resto-opget> → <option>**where**<condiciones>
11. <option> → **all | specific | pattern | primitive | composite | Є**
12. <condiciones> → <condiciones><más-condiciones>
13. <más-condiciones> → <op-lógico><condiciones><más-condiciones>
14. <más-condiciones> → **Є**
15. <condiciones> → <cond>
16. <condiciones> → <cond1>
17. <cond> → <cond><más-cond>
18. <más-cond> → <op-lógico><cond><más-cond>
19. <más-cond> → **Є**
20. <cond> → <cond-comp>
21. <cond1> → **(**<cond-comp><ot-cond-comp>**)with recursive**<rwr>
22. <ot-cond-comp> → <op-lógico><cond-comp><ot-cond-comp>
23. <ot-cond-comp> → **Є**
24. <cond-comp> → <exp-cam><opdor-comp><expresión>
25. <op-lógico> → **and** | **or**
26. <opdor-comp> → > | < | ≠ | **like** | =

27. <rwr> → **Є**
28. <rwr> → **(**<entero>**) | (~)**
29. <exp-cam> → *****<cont-exp-cam>
30. <exp-cam> → **_**<cont-exp-cam>
31. <exp-cam> → <letra><fin-exp-cam>
32. <cont-exp-cam> → **.**<exp-cam>
33. <fin-exp-cam> → <lyd><fin-exp-cam>
34. <fin-exp-cam> → **.**<exp-cam>
35. <fin-exp-cam> → **Є**
36. <expresión> → **’**<hilera>**’**
37. <expresión> → <cte>
38. <hilera> → <símbolo><hilera>
39. <hilera> → <símbolo>
40. <letra> → **a..z**
41. <lyd> → **a..z** | **0..9** | **Є**
42. <símbolo> → **a..z** | **0..9** | **+** | **/** | **&** | …
43. <cte> → <signo><n-d>
44. <cte> → <signo><n-d>**E**<entero>
45. <entero> → <signo><ess>
46. <ess> → <dígito><ess>
47. <ess> → <dígito>
48. <signo> → **+** | **-** | **Є**
49. <dígito> → **0..9**
50. <n-d> → <ess>
51. <n-d> → <ess>**.**
52. <n-d> → **.**<ess>
53. <n-d> → <ess>**.**<ess>

**Operador delete**

54. <opdel> → **delete (**<opget>**)**
55. <opdel> → **delete** <identificador><masid>
56. <masid> → **,**<identificador><masid>
57. <masid> → **Є**
58. <identificador> → **c**<ess>

**Operador update**

59. <opupdate> → **update** <identificador>**add**<resto-add>
60. <opupdate> → **update** <identificador>**delete**<resto-delete>
61. <opupdate> → **update** <identificador>**change**><resto-change>
62. <resto-add> → <exp-cam>**:**<expresión>
63. <resto-add> → <exp-cam>**:(**<value-comp>**)**
64. <resto-add> → <exp-cam>
65. <value-comp> → <letra><lyd>**:**<expresión><resto-value-comp>
66. <resto-value-comp> → **,**<letra><lyd>**:**<expresión><resto-value-comp>
67. <resto-value-comp> → **,**<letra><lyd>**:(**<value-comp>**)**<resto-value-comp>
68. <resto-value-comp> → **Є**
69. <resto-add> → **subject:(**<rol>**,'**<hilera>**')**
70. <rol> → **define** | **motivate** | **evaluate** | **demonstrate** | **experiment | apply | example**
71. <resto-add> → **prerequisite:(‘**<hilera>'**,** <nivel>**)**
72. <nivel> → **low** | **medium** | **high**
73. <resto-delete> → <exp-cam>**:**<expresión>
74. <resto-delete> → <exp-cam>**:(**<value-comp>**)**
75. <resto-delete> → <exp-cam>
76. <resto-delete> → **subject:(**<rol>**,'**<hilera>**')**
77. <resto-delete> → **prerequisite:(**'<hilera>**',** <nivel>**)**
78. <resto-change> → <exp-cam>**:**<expresión>
79. <resto-change> → **input:(**<letra><lyd>**, label** | **value,'**<hilera>**')**
80. <resto-change> → **output:(**<letra><lyd>**, label** | **value,'**<hilera>**')**

**Operador show**

81. <opshow> → **show**<pal-show> **of components** <identificador>
82. <opshow> → **show prerequisites of** <getpaths>
83. <lista-hileras> → **‘**<hilera>**'**
84. <lista-hileras> → **‘**<hilera>**'**<lista-hileras>
85. <pal-show> → **composition** | **lom-metadata** | **prerequisite** | **all** | **prerequisites** | **subject** | **input** | **output** | **domain-metadata**

**Operador getpaths**

86. <getpaths> → **getpaths from** <identificador>**with{**<lista-hileras>**}**

**Operador replace**

87. <opreplace> → **replace in** <identificador>**(n**<ess>**) where** <resto-where>
88. <lista> → <exp-cam> <opdor-comp><expresión><resto-lista>
89. <resto-lista> → <op-lógico><lista>
90. <resto-lista> → **Є**
91. <resto-where> → <nivel-sin-and>
92. <resto-where> → <lista> <nivel-con-and>
93. <nivel-sin-and> → <opdor-comp> **difficulty**
94. <nivel-con-and> → **Є**
95. <nivel-con-and> → **and** <nivel-sin-and>

Nota: En el operador replace expresión de camino relacionada con preferencias (formato, idioma etc.)

**Operador find similar**

96. <opfind> → **find similar**<opción-find>**components contents** <resto-find>
97. <opción-find> → **all** | **primitive** | **composed** | **Є**
98. <resto-find> → **(**<lista-roles>**)'**<hilera>**'**<resto-resto-find>
99. <resto-resto-find> → **and** <resto-find>
100. <resto-resto-find> → **Є**

**Operador create**

101. <opcreate> → **create component(‘**<file-metadata>**',**<comp-create>**)**
102. <file-metadata> → <hilera>
103. <comp-create> → <subjects>**,**<input-output>**,**<prereq><resto-comp-create>
104. <subjects> → **{**<lista-ele1>**}**
105. <lista-ele1> → <ele1>
106. <lista-ele1> → <ele1>**,**<lista-ele1>
107. <ele1> → **((**lista-roles>**),'**<hilera>**')**
108. <lista-roles> → <rol>
109. <lista-roles> → <rol>**,**<lista-roles>
110. <input-output> → **{**<lista-ele2>**}**
111. <lista-ele2> → <ele2> | **Є**
112. <lista-ele2> → <ele2>**,**<lista-ele2>

113. <ele2> → **(´**<hilera>**’,’**<hilera>**’,’**<hilera>**’)**
114. <prereq> → **{**<lista-ele3>**}**
115. <lista-ele3> → <ele3> | **Є**
116. <lista-ele3> → <ele3>**,**<lista-ele3>
117. <ele3> → **(‘**<hilera>**’,**<nivel>**)**
118. <resto-comp-create> → **,**<composición>
119. <resto-comp-create> → **Є**
120. <composición> → **sequence(**<lista-nodos>**)**
121. <composición> → **union(**<lista-secuencias>**)**
122. <lista-nodos> → <nodo>
123. <lista-nodos> → <nodo>**,**<lista-nodos>
124. <lista-secuencias> → **sequence(**<lista-nodos>**),**<lista-secuencias>
125. <lista-secuencias> → **sequence(**<lista-nodos>**)**
126. <nodo> → **(c**<ess>**,**<weight>**,**<interchangeable>**,**<mandatory>**,**<ele1>**)**
127. <weight> → **0..100**
128. <interchangeable> → <boolean>
129. <mandatory> → <boolean>
130. <boolean> → **true** | **false**

**Glosario**

**AHA:** Adaptive Hypermedia Architecture. Un sistema para la creación de recursos educativos el cual usa como base un lenguaje de scripting.

**BNF:** Backus Naur Form, notación estándar para escribir lenguajes (gramáticas).

**Componente** (de software): Un componente es una *caja negra* que *encapsula servicios* detrás de *interfaces* bien definidas.

**CE:** Componente Educativo.

**CEC:** Componente Educativo Compuesto, es decir, que está formado por otros CE's

**CEP:** Componente Educativo Primitivo, es implementado por un archivo ubicado en un servidor el cual es el recurso educativo en sí.

**CEM:** Componente Educativo Miembro o partícipe de un CEC.

**Componente Educativo:** Un recurso educativo digital visible a través de un browser que contiene una interfaz, una documentación y un contenido pedagógico

**DTD:** (Document Type Definition). Gramática o reglas que debe cumplir un documento XML para ser considerado bien formado (válido).

**ELM-ART:** Episodic Learner Model - The Adaptive Remote Tutor. Un sistema educativo evolucionado para la enseñanza del lenguaje LISP.

**HTML:** (HyperText Markup Language). Lenguaje de Marcas de Hipertexto. Lenguaje para elaborar páginas web.

**Interbook:** Sistema orientado a la web para la crear libros electrónicos adaptables. (interbooks)

**KBS:** Proyecto que permite la creación de sistemas de hypermedia adaptativa e implementa el aprendizaje basado en proyectos.

**LD:** Lógica descriptiva

**Lógica descriptiva:** Formalismo utilizado para representar conocimiento.

**LOM:** Propuesta de la IEEE para estandarizar los metadatos educativos.

**LORE:** Proyecto para manipular datos semiestructurados desarrollado en la Universidad de Stanford.

**ME:** Metadatos extensibles de un CE, es decir, metadatos adicionales a los propuestos por el LOM.

**Multibook:** Un robusto sistema hypermedial adaptativo de aprendizaje.

**Recurso Educativo:** Cualquier entidad digital o no digital que sirve para llevar a cabo un proceso de enseñanza

**Semiestructurado (Dato):** Dato que se describe a sí mismo (self-describing), es decir, la información que está normalmente asociada a un esquema se encuentra dentro de los datos y puede modelar datos heterogéneos más naturalmente que el modelo relacional o el modelo orientado a objetos.

**URL**: (Uniform Resource Locator). Denominación que no sólo representa una dirección de Internet sino que apunta a un recurso concreto dentro de esa dirección.

**Web:** Ver WWW.

**WWW:** (World Wide Web). Telaraña mundial, para muchos la WWW es Internet, para otros es sólo una parte de esta. Podríamos decir estrictamente que la web es la parte de Internet a la que accedemos a través del protocolo HTTP y en consecuencia gracias a browsers normalmente gráficos como Netscape.

**XML:** (eXtensible Markup Language). Lenguaje que complementa a HTML ya que permite describir contenido *no* presentación.

**XPath:** Un lenguaje de consulta para XML.

**XSL:**(Extensible Stylesheet Language). Lenguaje par crear hojas de estilo. Sirve para "traducir" documentos XML a HTML, dándoles así presentación.

**Bibliografía**


**[Abiteboul 97]** Abiteboul S., Quass D., McHugh J., Widom J., Wiener J.; *"The LOREL Query Language for Semistructured Data"*, Technical Report, Stanford University, 1997.

**[Abiteboul 00]** Abiteboul S., Suciu D., Buneman P.; *"Data on the Web: From Relations to Semistructured Data and Xml"*, Morgan Kaufmann, 2000.

**[Baader 03]** Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P.; ***"****The Description Logic Handbook: Theory, Implementation and Applications"*, Cambridge University Press, 2003.

**[Barbacci 89]** Barbacci M., Doubleday D., Weinstock C.; "*Durra: A Task-Level Description Language User's Manual"*, Technical Report, Universidad de Carnegie Mellon,1989.

**[Barroca 00]** Barroca L.(Editor), Hall J. (Editor), Hall P.; *"Software Architecture : Advances and Applications*", Springer Verlag, 2000.

**[Basu 01]** Basu J., Morriseau-Leroy N., Solomon M.; "*ORACLE8i Programación de Componentes Java*", McGraw Hill & Osborne, 2001.

**[BECTA 01]** BECTA; *"Metadata in Education"*, Technical Report, British Educational Communications and Technology Agency, Agosto de 2001.

**[Birngruber 01]** Birngruber D.; *"CoML: Yet Another, But Simple Component Composition Language"*, Workshop on Composition Languages, Viena, Austria, Septiembre de 2001.

**[Brusilovsky 97]** Brusilovsky P., Schwarz E.; *"InterBook User and Author Manual"*, Technical Report Universidad de Carnegie Mellon – Universidad de Trier, 1997.

**[Buyens 01]** Buyens J. *"Aprenda Desarrollo de Bases de Datos Web Ya"*, McGraw Hill – Microsoft, 2001.

**[Chang 01]** Chang B., Scardina M.,Karun K., Kiritzov S., Macky I., Ramakrishnan N.; *"Manual de Oracle XML"* , Oracle Press-Osborne-McGraw Hill, 2001.

**[Cramer 91]** Cramer J., Fey W., Goedicke M., Große-Rhode M.; "*Towards a Formally Based Component Description Language: A Foundation for Reuse"*, Structured Programming, Vol 12, No. 2, 1991.

**[Duitama 02]** Duitama F., Defude B.; "*Web Services for Educational Contents*",
Technical Report. Institut National des Telecommunications, Dept Informatique,
Evry-France, Diciembre de 2002.

**[De Bra 98]** De Bra P., Calvi L.; *"AHA: a Generic Adaptive Hypermedia System"*, Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia HYPERTEXT'98, Pittsburgh, Junio de 1998.

**[El Saddik 00]** El Saddik A., Ghavam A., Fischer S., Steinmetz R.*; "Metadata for Smart Multimedia Learning Objects"*, Proceedings of the fourth Australasian Computing Education Conference. ACM-CSE, Melbourne, Diciembre de 2000.

**[El Saddik 01]** El Saddik A., Fischer S., Steinmetz R.; *"Reusability and Adaptability of Interactive Resources In Web-Based Educational Systems",* ACM Journal of Educational Resources in Computing, Vol. 1, No. 1, 2001.

**[Fischer 98]** Fischer, D.; "*From Thesauri towards Ontologies"*, Structures and Relations in Knowledge Organization. Proceedings 5th Int. ISKO-Conference, Lille, 1998.

**[Goguen 86]** Goguen J.; *"Reusing and Interconnecting Software Components"* ,IEEE Computer; Vol 19, No 2, Febrero de 1986.

**[Hacid 02]** Hacid M., Leger A., Rey C., Toumani F.; *"Dynamic Discovery of e-Services: a Description Logics Based Approach"*, Symposium on the Effectiveness of Logic in Computer Sciences in Honour of Moshe Vardi, Marzo de 2002.

**[Henze 98]** Henze N., Nejdl W.; *"Adaptivity in the KBS Hyperbook System"*, Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW. 1998.

**[Henze 00]** Henze N.; "Adaptive Hyperbooks: Adaptation for Project-Based Learning Resources", PhD Tesis, Universidad de Hannover, 2000.

**[Hodgins 00]** Hodgins W.,Weibel S.; *"Memorandum of Understanding between the Dublin Core Metadata Initiative (DCMI) and the IEEE Learning Technology Standards Committee"*,Technical Report, DCMI: Dublin Core Metadata Initiative, 2000.

**[Iribarne 00]** Iribarne L., Valecillo A.; "*Sobre la Búsqueda y Emparejamiento de Componentes COTS con Múltiples Interfaces"*, Actas de JISDB'2000: V Jornadas de Ingeniería del Software y Bases de Datos, Noviembre de 2000.

**[Klein 02]** Klein M., Bernstein A.; *"Searching for Services on the Semantic Web Using Process Ontologies"*, The Emerging Semantic Web – Selected Papers from the First Semantic Web Working Symposium, Amsterdam, 2002.

**[Lahiri 99]** Lahiri T. , Abiteboul S., Widom J.; *"Ozone: Integrating Structured and Semistructured Data"*, Proceedings of the Seventh International Conference on Database Programming Languages, Kinloch,1999.

**[Magee 94]** Magee J., Dulay N., Kramer J.; "*A Constructive Development Environment for Parallel and Distributed Programs*", Technical Report, Colegio Imperial de Ciencias de Londres,1994.

**[Mann 87]** Mann W., Thomson S.A.; "*Rhetorical Structure Theory: A theory of text organization"*, Technical Report, Information Science Institute, USA 1987

**[Marvie 00]** Marvie R., Pellegrini M, *"Etat de l'art des modèles de composants"*, Technical Report, Proyecto RNRT CESURE, 2000.

**[Nierstrasz 98]** Nierstrasz O.; *"A Small Composition Language"*, Dagstuhl Workshop 9817 on Generic Programming,1998.

**[Roschelle 99]** Roschelle J., DiGiano C., Koutlis M., Reppening A., Phillips J., Jackiw N., Suthers D.; *"Developing Educational Software Components*", IEEE Computer, Septiembre 1999.

**[Schwarz 97]** Schwarz E., Brusilovsky P., Eklund J.; *"Adaptive Textbooks on the World Wide Web"*, Third Australian World Wide Web Conference AusWeb97, Julio de 1997.

**[Seeberg 97]** Seeberg C., El Saddik A., Steinacker1 A., Reichenberger K., Fischer S., Steinmetz R.; *"From the User's Needs to Adaptive Documents"* Proceedings of the Integrated Design & Process Technology IDPT, June 1999.

**[Seeberg 00]** Seeberg C., Steinacker A., Steinmetz R*., "Coherence in Modularly Composed Adaptive learning Documents*", Adaptive Hipermedia and Adaptive Web-Based Systems. Proceedings of the AH, Heidelberg, 2000.

**[Shaw 95]** Shaw M., DeLine R., Klein D, Ross T, Young D., Zelesnik G.; *"Abstractions for Software Architecture and Tools to Support them"*, Technical Report, Universidad de Carnegie Mellon,1995.

**[Sreedhar 00]** Sreedhar V.; *"ACOEL on CORAL: A Component Requirement and Abstraction Language"*, Technical Report, IBM TJ Watson Research Center, 2000.

**[Suthers 01]** Suthers D.; *"Evaluating the Learning Object Metadata for K-12 Educational Resources"*, Proceedings of the IEEE International Conference on Advanced Learning Technologies, Madison, Agosto de 2001.

**[Szyperski 99]** Szyperski C.; *"Components and Objects Together",* Software Development Magazine, Mayo de 1999.

**[Weber 97]** Weber G., Specht M*.; "User Modeling and Adaptive Navigation Support in www-based tutoring systems"*, User Modeling: Proceedings of the Sixth International Conference, Vienna, Departamento de Sicología Universidad de Trier.1997.

**[Wu 01]** Wu H.,De Kort E., De Bra P.; *"Design Issues for General Purpose Adaptive Hypermedia Systems",* Proceedings of the 12th ACM Conference on Hypertext and Hypermedia, Arhus, Universidad Tecnológica de Eindhoven, 2001.

**Páginas Web.**

[**Web Page 1**] http://www.iath.virginia.edu/elab/hfl0097.html

[**Web Page 2**] http://www.psychologie.uni-trier.de:8000/projects/ELM/elmart.html

[**Web Page 3**] http://www.multibook.dk/Demo/demo.htm

[**Web Page 4]** http://wwwis.win.tue.nl/2L690

[**Web Page 5]** http://www.contrib.andrew.cmu.edu/~plb/InterBook.html

[**Web Page 6]** http://www.kbs.uni-hannover.de/hyperbook

[**Web Page 7]** http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-45

**[Web Page 8]** http://www-db.stanford.edu/~widom/xml-whitepaper.html

**[Web Page 9]** http://www-db.stanford.edu/lore

**[Web Page 10]** http://www.w3.org/XML

**[Web Page 11]** http://www.research.att.com/~mff/xmlql-demo/html

**[Web Page 12]** http://www.w3schools.com/dom/default.asp

**[Web Page 13]** http://exist.sourceforge.net

**[Web Page 14]** http://dublincore.org/groups/education

**[Web Page 15]** http://ariadne.unil.ch/metadata

**[Web Page 16]** http://ltsc.ieee.org/wg12/index.html

**[Web Page 17**] http://www.research.att.com/~mff/xmlql

**[Web Page 18]** http://www.dbxml.com

**[Web Page 19]** http://www.xmldb.org/xupdate