

Mophoua

Lenguaje de Programación Orientado al Contexto del Objeto

20 de enero de 2008

<http://mophoua.sf.net>

Oscar Pérez Mora
Máster Internacional en Software Libre

Gregorio Robles Martínez
Director Proyecto de Fin de Máster

Copyright (c) 2008 Oscar Pérez Mora.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being just “Resumen”, “Modelo Orientado al Contexto del Objeto” and “Conclusiones”, one Front-Cover Texts: “Mophoua. Lenguaje de programación orientado al contexto del objeto. <http://mophoua.fs.net>”, and no Back-Cover Text. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; siendo las Secciones Invariantes “Resumen”, “Modelo Orientado al Contexto del Objeto” y “Conclusiones”, siendo los Textos de Cubierta Delantera “Mophoua: Lenguaje de programación orientado al contexto del objeto. <http://mophoua.fs.net>”, sin textos de Cubierta Trasera. Una copia de la licencia está incluida en la sección titulada *GNU Free Documentation License*.

En la Universidad de Guadalajara desarrollo un proyecto de investigación que contempla la creación de un modelo de programación que se ha denominado Context Object Oriented Programming(COOP). En el se integran los enfoques Orientado a Objetos, Orientado a Roles y Programación Dirigida por Eventos. El presente proyecto consiste desarrollar el lenguaje de programación Mophoua que soporta este modelo. La intención es construir una relación simbiótica entre el lenguaje y el modelo, que acelere la evolución de ambos mediante la experimentación, facilitando el trabajo de investigación.

Por su motivación principal, Mophoua es un Lenguaje Orientado al Contexto del Objeto (LOCO) o por sus siglas en ingles COOL (Context Object Oriented Language), pero también tiene como objetivo ser minimalista. Al inicio del proyecto Mophoua se observo que con extender lenguajes de programación como Java o C++ para incluir los conceptos COOP, se incrementa la complejidad sintáctica de estos lenguajes, y se obscurece la simplicidad pretendida por el Modelo Orientado al Contexto. Por este motivo, se opto por diseñar una nueva sintaxis para el lenguaje que mantenga una relación directa y limpia con la semántica subyacente.

Como estrategia experimental, Mophoua favorece el uso de símbolos y estructuras de texto por encima de palabras reservadas. Con ello se explora si el nivel de abstracción permite programas más legibles. El reto es mantener una curva de aprendizaje pequeña y un nivel de abstracción que equilibre la fácil comprensión humana y el poder expresivo del

lenguaje.

El desarrollo de Mophoua se inicio con la herramienta Eli y como precompilador de C++, al inicio del 2008 se encuentra en etapa de migración a la herramienta ANTLR y se implementa como precompilador de Java.

Licencia GFDL.....	3
Resumen.....	4
Contenido.....	6
1: Introducción.....	8
Objetivos.....	8
Motivación.....	9
Estado del Arte.....	10
Metodología general.....	12
Plan de Trabajo.....	13
Especificaciones generales para la versión 0.0.1 :.....	15
Estructura de esta memoria.....	16
2: Modelo Orientado al Contexto del Objeto.....	17
Antecedentes.....	17
Objetos Simples.....	21
Objetos: Dualidad instancia-clase.....	21
Propiedades: Dualidad atributo-operación.....	22
Encapsulamiento: Dualidad Interfaz-implementación.....	22
Herencia: Padre-Hijo.....	23
Objetos Complejos.....	25
Agregación: dualidad componentes-colaboración.....	25
Mensajes: solicitud – evento.	26
Contexto: Roles – Actores.....	27
Efecto esquizofrenia.....	29
3: Diseño e Implantación.....	31
Indentación.....	31
Identificadores	37
Clases y Roles.....	39
Sentencias.....	43
Resolución de métodos.....	45

Símbolos.....	46
4: Herramientas.....	48
Eli.....	48
Antl y AntiWorks.....	54
IDEs para Mophoua.....	57
5: Conclusiones.....	60
Referencias.....	61
Anexos.....	63
Mini tutorial de Mophoua.....	63
Objetos y Clases.....	63
Roles.....	65
Ejemplos.....	66
Implementación en Eli.....	68
GNU Free Documentation License.....	76

Objetivos

El objetivo general del proyecto es desarrollar un prototipo para el lenguaje de programación Mophoua en el que sea posible implementar programas modelados con Orientación a Roles y Programación dirigida por eventos de acuerdo al modelo propuesto en el presente trabajo. El prototipo debe ser lo suficientemente funcional para ser utilizado como lenguaje de propósito general. Sin embargo, este trabajo no contempla la utilización o desarrollo de librerías para el lenguaje.

La sintaxis del lenguaje debe tener una correspondencia directa y limpia con el modelo propuesto, de forma que su curva de aprendizaje no sea pronunciada. Un objetivo complementario es que el lenguaje pueda servir de herramienta didáctica para el aprendizaje de la programación, en técnicas como el refinamiento progresivo y diseño dirigido por responsabilidades, aplicables a la programación estructurada y orientada a objetos, de los cursos básicos de la formación en ciencias computacionales. El modelo subyacente de Mophoua permitirá además, una fácil introducción a la programación de roles y eventos.

Motivación

La programación orientada a objetos llegó con la promesa de ofrecer mecanismos de abstracción para disminuir la complejidad de los sistemas de cómputo. Sin embargo, el poder de dichas abstracciones es limitado. El modelo de roles ofrece una alternativa para enfrentar la complejidad en un nivel que la POO tradicional no puede.

Actualmente, las investigaciones sobre el modelado de roles se encuentran en proceso, y falta mucho aún para que constituyan un campo de conocimiento coherente y consensuado. Se han desarrollado algunas propuestas para implantar el concepto de rol en diversos lenguajes de programación, siendo Java el más concurrido. Para los objetivos del presente proyecto, son de especial interés PowerJava y ObjectTeams.

La programación orientada a aspectos, enfrenta la complejidad separando un sistema transversalmente desde varios enfoques. Esto es en gran medida similar a lo propuesto en el modelado de roles, sin embargo existen algunas diferencias conceptuales.

La intención del presente proyecto es definir un modelo de programación orientado a roles e implementarlo en un lenguaje de programación, con el propósito de evaluar sus ventajas y desventajas comparadas con la POO tradicional. En una primera etapa se pretende desarrollar y liberar el primer prototipo del lenguaje para su evaluación pública, en el periodo comprendido de agosto del 2007 a enero de 2008.

El proyecto inicia su desarrollo en febrero de 2007, tomando como base investigaciones anteriores al respecto. El presente trabajo es un plan de desarrollo que finaliza en enero del 2008 y se desarrolla en la Universidad de Guadalajara y la Universidad Abierta de Cataluña.

Otro problema identificado al analizar diversos lenguajes de programación modernos, es el ocasionado por el aumento de complejidad interna con el paso de los años. Esto debido a la incorporación de nuevas herramientas con el fin de satisfacer necesidades de mercado. Provocando que los lenguajes no siempre implementen una sintaxis y semántica simples.

Finalmente, la programación dirigida por eventos se sustenta en un modelo bastante desarrollado desde hace algún tiempo. Sin embargo, los modernos lenguajes de programación han incorporado constructos en el que no se reflejan de forma natural el modelo. Esto provoca que los programas sean difíciles de mantener, y que los estudiantes de programación no puedan apreciar el concepto de eventos en su sencillez .

Mophoua tiene entre sus objetivos, un diseño minimalista, pero completo que permita la implantación de programas orientados a objetos, roles y eventos. Buscando en su diseño una sintaxis limpia y una semántica lo más transparente posible. Se pretende lograr coherencia entre la semántica del lenguaje y el modelo de programación propuesto.

Estado del Arte

El modelo de roles ha sido motivo de diversas investigaciones y metodologías de desarrollo desde hace varios años [Reenskaug 1996] Es un concepto que se aplica en varias áreas del conocimiento, como son la lingüística, la sociología, teorías cognitivas, los sistemas multiagentes, el modelado orientado a objetos, entre otros. Una introducción al tema desde el punto de vista de las ciencias de la computación puede consultarse en Steimann [Steimann 2000].

En la literatura se pueden encontrar fundamentalmente dos enfoques: roles como relaciones, o *roles relacionales* [Balzer 2007] y el de roles como funcionalidades en un contexto [Loebe 2007] que llamaremos *roles contextuales*.

Del estudio de la cognición se ha propuesto importar el término *affordances* (una posible traducción es *permisiones*), que se refiere a lo que un objeto "se permite" ver y utilizar de otro objeto en una relación. Esta "permisión" está mediada por los intereses de la relación y en cierta forma es subjetiva al objeto que observa. Del campo de la teoría organizacional, se utiliza el término *institución* [Baldoni 2006b] para referirse al conjunto de objetos que colaboran para alcanzar una tarea. Una institución asigna *poderes* a los objetos que forman parte (apoderados), los roles son utilizados para distribuir responsabilidades, obligaciones y derechos a los agentes que trabajan en una organización. ObjectTeams/J [Herrmann 2007] es una extensión del lenguaje Java, que permite la separación de aspectos en clases especiales que denomina *equipos* y se declaran con la palabra reservada *team*. Utiliza el término *Crosscutting Collaborations* [Herrmann 2002] para referirse a separación del sistema por áreas de interés colaborativo. Las clases que se definen al interior de un *equipo* se denominan *roles*, y deben ser 'jugados' por los objetos que colaborarán en el equipo y que pertenecen a lo que se llama *clase base*. Los equipos y los roles pueden ser activados o desactivados mediante el uso de *guardias*.

Los roles restringen el acceso a los miembros de la clase base, pudiendo interceptar llamadas a métodos mediante el procedimiento que denomina *callin*. El proceso inverso se denomina *callout*, y consiste en el enlace de métodos solidados al rol y que son redirigidos a la clase base. Este enlace puede ser hacia un método oculto de la clase base, aplicando

lo que se llama *desencapsulación*.

Los objetos y los roles pueden accederse mutuamente mediante los procedimientos llamados *lifting* (obtener roles de un objeto) y *lowerling* (obtener objeto de un rol). De esta forma roles y objetos comparten una misma identidad.

ObjectTeams tiene planeado la inclusión de constructos sintácticos que permitan realizar consultas en *join points*, de esta forma dar soporte al paradigma de programación por Aspectos.

PowerJava es otro lenguaje de programación que debe su nombre a los términos empleados en teoría organizacional [Baldoni 2005], dónde una *institución* es la encargada de definir los *poderes* que confiere a un objeto, esta es la tarea de los roles.

Un rol puede asignarse a objetos que no comparten la misma superclase. En este sentido un rol es una clase con doble interfaz, la que ofrece para el servicio de la institución y la que requiere del objeto que pretende jugar dicho rol. Al ser una doble interfaz, los roles en powerjava no incluyen implementación, se definen como clases abstractas. Dónde la implantación se da en el interior de las clases que definen tipos naturales. Este es un mecanismo intrusivo de definición de roles. La institución es el conjunto de declaración de roles abstractos.

Metodología general

El proceso metodológico inicia con un estudio comparativo de los lenguajes de programación PowerJava [Baldoni 2006] y ObjectTeams/Java [Herrmann 2007], analizando el modelo conceptual que sustentan. Se analizaron también los modelos de objetos definidos por C++, Eiffel y

Java [Kolling 1998]. Como segundo paso, se adopta un modelo de roles y eventos con base en la revisión de la literatura [Steimann 2000][Reenskaug 1996] entre otros, y el resultado del estudio comparativo de lenguajes. El modelo se revisa y se realizan los ajustes para simplificarlo. Con base al modelo obtenido se define una gramática para el lenguaje de programación Mophoua. Finalmente se evalúa la implantación lograda.

El sistema de indicadores utilizado se centra en los constructos del lenguaje y en los elementos de modelado que no están presentes en otros lenguajes de programación. Se evalúa así mismo la claridad y potencia de modelado, aplicando los conceptos utilizados en este trabajo. Se pretende comparar, mediante casos de estudio, las soluciones propuestas en distintos lenguajes a un problema dado.

Para la implantación de Mophoua se utilizó la herramienta de construcción de compiladores Eli [Gray 1992] y se ha iniciado su reescritura en ANTLR [Parr 1995] por sus ventajas para desarrollar rápidamente prototipos.

Plan de Trabajo

1. Publicar el 'Modelo Contextual' con soporte de roles y eventos en un congreso o revista.
 1. El 26 de octubre de 2007, se presentó la ponencia "Lenguaje de programación minimalista orientado a roles y eventos" en el marco del *XX Congreso Nacional y VI Congreso Internacional de Informática y Computación* organizado por la ANIEI. En a página

de las ponencias aceptadas se puede encontrar con el número 67¹, a la fecha de realización de este trabajo, las memorias del congreso no han sido publicadas aún en la página.

2. El 16 de noviembre de 2007 se presentó la ponencia “Programación Orientada a Roles” en el marco del *VI Festival GNU/Linux y Software Libre*² organizado por la Universidad de Guadalajara.
 3. Se encuentra en fase de pre-aceptación la ponencia “Creación de un lenguaje de programación Orientado a Roles” en el *Congreso Nacional de Software Libre 2008*³
 4. Se proyecta proponer un artículo en alguna revista indexada.
2. Búsqueda discreta de colaboradores. Antes de realizar un anuncio masivo en la comunidad de Software Libre, se planteó la promoción del proyecto dentro de círculos más cercanos. En las presentaciones realizadas se han recopilado comentarios de los participantes, y 2 de ellos han mantenido el interés en participar en el proyecto y se ha mantenido contacto con ellos.
 3. Estabilizar la versión 0.0.1 de Mophoua. Se ha definido un conjunto inicial de características del lenguaje para la versión 0.0.1, no se definirán nuevas características hasta que esté liberada la versión. El proceso de liberación se retrasó un poco por tomar la decisión de migrar a ANTLR.
 4. Publicar el proyecto en sourceforge.net. El espacio del proyecto se abrió en la dirección <http://mophoua.sf.net>. el espacio para la comunidad de desarrolladores es

1 <http://www.aniei.org.mx/portal/modules.php?name=cnciic&leng=es&op=11>

2 <http://fsl.udg.mx/programa.html>

3 <http://comas.consol.org.mx/2008/general/proposals>

<http://sourceforge.net/projects/mophoua/>

5. Realizar la documentación mínima. La documentación propuesta incluye el modelo subyacente, un mini-tutorial del lenguaje y la manual del desarrollador.
6. Crear la página del proyecto. La página del proyecto se encuentra en mophoua.fs.net
7. Establecer las líneas de desarrollo para la comunidad. Se contempla una vez estabilizada la versión, realizar un framework para desarrollo de interfaces GUI. Por lo pronto, la línea de desarrollo se establece por las especificaciones de la versión 0.0.1.
8. Presentación masiva. Se realizará su anuncio en freshmeat.net, cofradia.org y algunas listas de correo de grupos de usuarios. La fecha contemplada originalmente era mediados de enero, se contempla que será a fines de enero.

Especificaciones generales para la versión 0.0.1 :

1. Clases con herencia simple.
2. variables de instancia de clase privados y métodos públicos.
3. Tipos primitivos, sin soporte para arreglos (posiblemente).
4. Estructuras de control sin sentencia foreach para colecciones.
5. Expresiones copiadas de java.
6. Sin soporte para librerías ni archivos múltiples.
7. Sin soporte para roles múltiples, solo roles unitarios.
8. Soporte para eventos.

9. Genera código Java.

El desarrollo se limita a un prototipo, por lo cual se implementa solo un pre-compilador de Java. La creación de una comunidad alrededor se inicia con una difusión discrecional en círculos especializados, como universidades y congresos. Después de obtener los resultados iniciales del primer prototipo, se planteará una difusión masiva. El prototipo no implementa un modelo completo de POO pero si funcional.

Estructura de esta memoria

- El capítulo central que describe el modelo subyacente a Mophoua es “Modelo Orientado al Contexto del Objeto”, contiene la perspectiva personal sobre el tema y es la motivación central del proyecto Mophoua.
- Una descripción de la herramientas para generación de compiladores y los motivos de la elección se describen en “Herramientas”
- El capítulo “Mophoua: Diseño e Implementación” describe los aspectos relevantes y las decisiones de diseño consideradas en el prototipo de Mophoua.
- Una introducción rápida para los usuarios se describe en el anexo: “Mini tutorial de Mophoua”

2: Modelo Orientado al Contexto del Objeto

La labor del desarrollo de sistemas se sustenta en el concepto de modelado. Desarrollar un software implica realizar un modelo del problema a resolver, de tal forma que pueda ser codificado posteriormente en la computadora. Desafortunadamente, en ingeniería de software proliferan modelos con poco

El Modelo Orientado al Contexto del Objeto descrito en esta sección es tan solo una propuesta o hipótesis que sirva de respaldo en la definición del lenguaje de programación Mophoua. Se realizó tomando como base diversos trabajos de investigación que se mencionan en la bibliografía y algunos otros. La propuesta aquí expuesta es perfectible y se delimita con el objetivo de servir como punto de partida para futuras investigaciones.

El modelo de roles es un supraconjunto del modelo de objetos. Por lo que la descripción del modelo incluye la visión particular OO de forma que se integre a la visión Orientada al Contexto del Objeto.

Antecedentes

El software es una representación de una fracción del mundo como lo entendemos. La complejidad de los sistemas que se desarrollan va en aumento, el número de componentes y sus interrelaciones es cada día mayor, así como su necesidad de evolución constante. Cada elemento

en sí mismo, representa un subsistema complejo, que requiere interactuar con otros elementos maximizando la cohesión y minimizando el acoplamiento. Nos acercamos a la necesidad de un *Modelado Ecológico del Software*. Un enfoque ecológico se preocupa la interacción de los actores con su entorno (dentro del computador y fuera de él) de forma que se logre un equilibrio para la evolución del sistema. Alta cohesión que significa sinergia y bajo acoplamiento que significa flexibilidad en la evolución. Los objetos construidos en un software, adquieren cada día mayor responsabilidad hacia el sistema y requieren mayor autonomía para decidir la mejor forma de cumplirla e integrarse en otros sistemas. En un ecosistema, la configuración del contexto se determina por los actores, así como también las características de estos últimos son resignificados por el contexto.

La descripción de los objetos que se realiza en el paradigma actual OO, mantiene un enfoque 'objetivo', en el sentido en que la definición de un objeto y su comportamiento no cambia al ser utilizado en diferentes contextos. La interacción con el entorno no influye en la estructura de los objetos. Las investigaciones se orientan al modelado de los objetos en su *contexto*. Van de la 'objetividad' a la 'subjetividad', esto es, la definición y comportamiento de los objetos dependen del entorno que les rodea, de su particular punto de vista subjetivo. En cierta forma se evoluciona de los *objetos* a los *sujetos*, con mayor carga de responsabilidad, mayor autonomía e interacciones complejas con su entorno. Son precisamente éstas las interacciones las que permiten que el sistema sea mayor que la suma de sus partes.

Mophoua es una propuesta de exploración rumbo a la creación de un modelado ecológico del software, aunque dista mucho de alcanzarlo. Permite la extensión de los objetos de forma dinámica para que pueden

adaptarse a diversos contextos. En Mophoua, el *contexto* es la plataforma sobre la que se construye la colaboración entre objetos para realizar tareas complejas.

En la tradición OO, un objeto mantiene una estructura estática y otra dinámica. Para objetos simples, la parte estática es representada por *atributos*, la parte dinámica por *operaciones*. En objetos complejos, la parte estática es representada por componentes [Browne 2005], la dinámica, por la colaboración entre dichos componentes [Reenskaug 2007]. De esta forma el contexto es el interior de un objeto complejo.

Durante su vida, un objeto colabora en varios contextos para apoyar la realización de diversas tareas. En cada uno, adquiere responsabilidades diferentes y comportamientos distintos. Desempeña un *rol* distinto en cada contexto. Sin perder su identidad, los objetos, que llamaremos *actores*, adoptan y abandonan roles de forma dinámica, conforme participan en diversos contextos. Por ejemplo, un objeto *persona* al ingresar al contexto de una *escuela* puede adquirir el rol de *profesor*. Simultáneamente, puede ingresar al contexto de una *empresa* y adquirir el rol de *empleado*.

Un rol es una extensión de un objeto. Define la funcionalidad requerida por el contexto en que participa dicho objeto [Loebe 2007]. En cierta forma 'instruye' al objeto para que colabore en un contexto específico. En una *colaboración* no importa si el componente es un *actor* o es un *rol* jugado por un actor, lo importante de cada componente es la interfaz que éste provee y que ayuda a que se logren las metas de la colaboración.

Mophoua distingue dos categorías de tipos (o clases), los tipos naturales y los tipos rol [Genovese 2007]. Los *tipos naturales* tienen identidad propia, por lo que pueden crearse instancias sin mayores requisitos. Los *tipos rol* son *extensiones dinámicas* de la funcionalidad de los tipos

naturales, no tienen identidad propia y su instanciación esta subordinada a una instancia de un tipo natural (así pues, no puede existir un *profesor* sin una *persona*). Para cada contexto en el que participa un actor se puede instanciar un rol. Por ejemplo, una *persona* puede ser *empleado* en dos *empresas* distintas, en cada una de ellas mantiene un salario y horario diferente. También existe la posibilidad del *rol único* que es *incrustado* en el actor [Chemuchin 2005].

Mophoua incrementa el nivel de autonomía de un objeto definiendo la *solicitud* y el *evento* como dos tipos distintos de *mensajes*. Una *solicitud* es un mensaje con una intencionalidad y destinatario, que obedece a una necesidad del objeto solicitante. La solicitud o llamada a función es el mecanismo tradicional de mensajes que implementan los actuales lenguajes Orientados a Objetos. Un *evento* es un mensaje que no lleva intencionalidad ni destinatario [Baldoni 2005], simplemente es un aviso que se lanza al contexto desconociendo las consecuencias que pueda provocar. El concepto de evento, corresponde al patrón observador [Gamma 1995] en el que algunos objetos (observadores) se mantienen al tanto de ciertas transformaciones que ocurren en el objeto observado. Este aviso puede no ser interceptado por ningún observador y perderse, o puede provocar una reacción en un observador. Mophoua incluye constructor especiales para dar soporte directo al concepto de eventos.

En Mophoua existen dos tipos de herencia, de interfaz y de clase. La herencia de clase es el mecanismo tradicional de herencia en los actuales lenguajes OO. El objeto derivado adquiere la estructura y comportamiento de la clase padre, hereda su interfaz y su implantación. En la herencia de interfaz, el objeto derivado no adquiere la implantación de la clase padre, aún cuando esta exista.

Se diferencian también dos mecanismos de extensiones de una clase:

Extensión estática proporcionada por la herencia en el sentido conferido por Java y *extensión dinámica* proporcionada por la definición de roles.

A continuación se expone un resumen del modelo Orientado a Objetos que incluye la visión del Contexto, al cual se ha denominado Orientado al Contexto del Objeto. Se considera una propuesta o hipótesis de respaldo en la definición del lenguaje de programación Mophoua. Se realizó tomando como base diversos trabajos de investigación que se mencionan en la bibliografía y algunos otros. Como cualquier trabajo, es una propuesta perfectible y se delimita con el objetivo de utilizarse como punto de partida en futuras investigaciones.

Objetos Simples

Objetos: Dualidad instancia-clase

Los **objeto** son abstracciones que realizamos de porciones del mundo que nos rodea, y delimitamos por un conjunto de propiedades que podemos identificar para diferenciarlo de los demás. Esta abstracción de propiedades se integran gracias al concepto de **identidad**. Gracias a ésta, cada objeto se identifica con una existencia un tanto independiente de los demás objetos. La identidad de un objeto permanece con él a través de la vida del objeto, sin importa los cambios que este pueda sufrir.

El conjunto de objetos que comparten las mismas propiedades se agrupan en **clases**. Los lenguajes OO definen clases que son utilizadas posteriormente como plantillas para la creación de objetos. A este proceso se le conoce como **instanciación**.

instancia	clase
------------------	--------------

Propiedades: Dualidad atributo-operación

Las propiedades de todo objeto se definen en las dimensiones tiempo-espacio. Los **atributos** definiendo el estado, y las **operaciones** definiendo los cambios de estado. El **comportamiento** se define como la secuencia de estados por la que puede transitar un objeto al utilizar sus operaciones.

instancia	clase
estado	atributos
comportamiento	operaciones

Encapsulamiento: Dualidad Interfaz-implementación.

Un objeto es una abstracción que se construye con los elementos primitivos de una computadora, que son variables e instrucciones. Así, los atributos se implementan con **variables** y las operaciones con **métodos**, que son conjuntos de instrucciones. Esta división divide a la instancia-clase en interfaz (que especifica el 'que' puede hacer) y la implantación (que especifica el 'cómo').

implantación	interfaz
variables	atributos
métodos	operaciones

La encapsulación oculta la implementación al resto de los objetos. *La interfaz debe ser independiente de la implantación*, esto significa que

para realizar una operación dada, el objeto puede implementar varios métodos y seleccionar es que considere adecuado. Así mismo, para implementar un atributo puede realizarlo mediante una variable, conjunto de variables o incluso un método. Por fuera, se accede a operaciones y atributos, por dentro, el objeto puede implementar varias alternativas para ellos.

Gran parte de los lenguajes de programación soportan la independencia de operaciones-métodos, pero vinculan fuertemente los atributos-variables.

Los **atributos** de un objeto pertenecen a la **interfaz**, y deben ser independientes de la implementación.

El acceso a un atributo son en realidad operaciones, LeerAtributo() y EscribirAtributo(), por lo tanto, la **encapsulación** se define como el ocultamiento de métodos y variables, y acceso solo mediante operaciones.

Herencia: Padre-Hijo

La **herencia** es el mecanismo mediante el cual se utiliza la definición de una clase llamada padre para definir una nueva clase llamada hija. El objetivo es realizar una **extensión** de propiedades en la clase hija.

Existen dos tipos de básicos de herencia de tipo y estructural. En la **herencia de tipo** la clase hija adquiere los atributos y operaciones de la clase padre (interfaz), pero no implementación (variables y métodos). En la **herencia estructural** la clase hija adquiere los las variables y los

métodos de la clase padre (Implantación). La **herencia de clase** más que otro tipo de herencia, es la combinación de la herencia de tipo y herencia estructural.

La **herencia de tipo** se aplica en la **interfaz** de la clase, la **herencia estructural** se aplica en la **implementación**.

La herencia de tipo mantiene la una relación **es-un** entre clases, mientras que la herencia estructural establece una relación **funciona-como** en el sentido de la implantación o funcionamiento interno.

La **herencia múltiple** representa un conflicto en el caso de herencia estructural, mientras que en la herencia de tipo no representa inconveniente pues solo se utiliza la interfaz del padre.

La herencia estructural puede sustituir las librerías globales. Por ejemplo: la clase “Gráficos” define operaciones para realizar trazos en un lienzo, y puede heredar estructuralmente de la clase “Matemática” que permite realizar funciones trigonométricas. “Gráficos” funciona-como “Matemática” pues para hacer los cálculos internos utiliza todas las operaciones de ésta. Por otra parte, no podemos solicitar a “Gráficos” operaciones que realiza “Matemática”.

Un **tipo** es un conjunto de objetos que comparten la misma interfaz, y una **clase** es un conjunto de objetos que comparten interfaz e implementación.

Objetos Complejos

Agregación: dualidad componentes-colaboración

Un **objeto simple** se compone de atributos que representan su estado, y operaciones, que representan su comportamiento. Un **objeto complejo** se conforma de objetos **componentes** que representan su estado, y de **colaboración** entre objetos, que representan su comportamiento.

instancia	clase
estado	componentes
comportamiento	colaboración

Un objeto complejo al igual que un objeto simple debe implementar una interfaz con operaciones y atributos, que muestran al cliente del objeto un aspecto de sencillez.

El estado y comportamiento de un objeto complejo depende de sus componentes, y esta estructura debe ocultarse por una interfaz de atributos y operaciones.

La **responsabilidad** de un objeto esta definida por el *papel*⁴ que juega dentro del objeto complejo. Cada uno de los componentes se define en función del objetivo del objeto complejo.

⁴ Papel puede considerarse un sinónimo de rol

Mensajes: solicitud – evento.

Este apartado puede llamarse también estructura dinámica de un objeto complejo. La colaboración es posible, solo mediante mecanismos adecuados de comunicación y el principal son los mensajes. Existen dos tipos de mensajes dentro de un objeto complejo:

Solicitud. es un mensaje que un objeto **cliente** envía al **servidor**, para solicitar un servicio. El cliente intenta con ésto satisfacer alguna necesidad particular y conoce la identidad del servidor. La solicitud se realiza directamente a un *operación* del servidor, y esta es atendida con un *método*.

Evento. es un mensaje que un objeto **observado** envía al **contexto**, para informar de un cambio de estado. El objeto observado no necesariamente tiene algún interés o necesidad particular para enviar el mensaje y no conoce su destino ni la identidad del **observador**. Un evento activa en el observador un **sensor** que es atendido con un *método*.

Un **mensaje** puede ser de dos tipos: **solicitud**, interés de quien lo lanza y **evento**, interés de quien lo recibe.

Las *solicitudes* se distribuyen utilizando **referencias** que conectan la *identidad* de un objeto servidor para dar acceso desde el cliente. Los *eventos* conceptualmente no tienen referencias, el evento se lanza al contexto y se ignora su destino.

El mecanismo de eventos eleva el nivel de autonomía de los objetos, podemos utilizar la metáfora del paso de objetos “inanimados” a objetos

“animados”⁵.

objeto “inanimado”	objeto “animado”
operación	sensor
Actúa como un mecanismo, causa-efecto	Reacciona a los cambios de su entorno

Una operación programa al objeto como se comporta cuando lo manipulen. Un sensor programa al objeto su comportamiento para adaptarse al contexto. El mecanismo de eventos es por definición conceptual concurrente.

Contexto: Roles – Actores

El interior de un objeto complejo establece un **contexto** para sus componentes, en el los objetos internos pueden comunicarse y colaborar para alcanzar un objetivo común. *El contexto establece los lineamientos para el comportamiento de cada componente*, aquel objeto que no respeta dichos lineamientos, puede provocar un fallo en todo el mecanismo de colaboración que significa la estructura de un objeto complejo. De esta forma decimos que cada objeto componente de un objeto complejo desempeña un **rol** o papel delimitado o definido por el contexto, de acuerdo a las metas de la colaboración.

5 La inteligencia artificial es un factor importante para simular objetos “animados”, este tema escapa al alcance del proyecto.

Un **rol** es el comportamiento que adopta un objeto al entrar a un contexto dado.

Pero sucede que *los objetos pueden pertenecer o colaborar en varios contextos*. Una *persona* puede desempeñar el rol de *trabajador* en el contexto de una empresa y de *padre* en el contexto de una familia. En este ejemplo *Familia* y *Empresa* son objetos complejos.

Los componentes de un objeto complejo son esencialmente **roles**. Un objeto puede tomar o dejar de jugar un rol en el transcurso de su vida y puede tomar en mismo tipo de rol en contextos diferentes llamados **roles múltiples**, así como también puede topar un rol que cambiará su comportamiento en todos los contextos, llamados **roles universales**. Los roles universales son en cierta forma un mecanismo *intrusivo*, pues cambian el comportamiento el el núcleo mismo del actor.

Al ingresar a un nuevo contexto, un objeto debe “aprender” a comportarse de acuerdo a los lineamientos de dicho contexto y además de cumplir con una responsabilidad para beneficiarlo, esto es, tomar un rol. Los roles no tienen identidad propia ni pueden instanciarse de forma independiente a un objeto. Un rol es el “aprendizaje” de los objetos para colaborar en diversos contextos.

El comportamiento de un rol dentro de un contexto puede ser distinto en la interacción con cada componente particular. Por ejemplo, el rol *empleado* tiene distintos comportamientos con el *jefe* y con un *cliente*.

Al objeto sin roles lo llamamos **actor**. Un rol es la interfaz de un actor a un contexto, el rol determina los permisos y la forma de percibir al actor. Un rol actúa como un filtro para el acceso a un objeto.

Un rol es una interfaz que define el tipo de acceso que se puede tener a un actor. Esta interfaz esta en las referencias.

Los roles establecen un mecanismo de polimorfismo por clasificación dinámica. Al heredar de un objeto la clase deriva adquiere los roles de la clase padre. Un rol puede derivarse de otro, el rol derivado se aplica igualmente a la misma clase de actor a la que pertenece el rol padre.

Los objetos complejos establecen una construcción recursiva objeto-colaboración-objeto complejo, pues cada objeto complejo puede formar parte de otra colaboración. Para cada objeto se establecen tres tipos de contextos, el **contexto externo** en el que interactúa con otros objetos, el **contexto interno** en el que interactuar sus componentes y el **contexto de identidad** que corresponde al actor y todos los roles que juega en distintos contextos en un tiempo dado.

Efecto esquizofrenia

INSERTAR LA CITA A FERRET

En el contexto de identidad, el actor y todos sus roles deben coordinarse para trabajar como una unidad, y evitar el efecto de objetos esquizofrénicos que tienen “múltiples personalidad” y conflicto de identidad. En la relación actor-rol la autonomía de uno disminuye al aumentar la del otro. De esta forma podemos describir los siguientes niveles de autonomía que puede tomar un actor en la relación con sus roles.

0. Sin autonomía. El jugador sigue las instrucciones del rol. El jugador

soporta solo operaciones atómicas. El rol lleva la responsabilidad de la tarea. Incluso, puede ser abstracto, no ofrecer ninguna funcionalidad, toda la funcionalidad estará en los roles. El rol decide incluso el método con el que se debe atender una solicitud. El actor se convierte tan solo en un núcleo manipulable. Ejemplo: TDA como Pila.

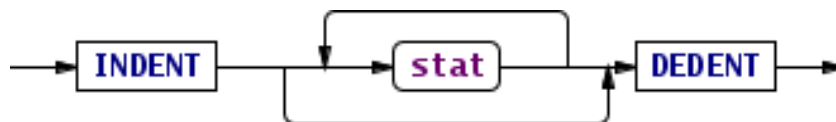
1. De procesos. El jugador puede seleccionar el método adecuado para un proceso. Los procesos se centran en la E/S del algoritmo, no importa el contexto, lo que importa que la mejor forma de realizarlo de acuerdo con los parámetros que éste recibe. TDA polimorficos y/o con sobrecarga de métodos.
 2. De metas. Puede detectar el contexto y las necesidades, para seleccionar un método que ayude a cumplir las metas que el jugador tiene planteadas. Las metas son del jugador (núcleo). Ejemplo: actor con sensores de los eventos de sus roles y con cierto conocimiento del contexto de sus roles.
1. Autónomo. Puede tener sus propios intereses independientes a los intereses de la organización, estos intereses nacen de la pertenencia simultánea a otros contextos. El objeto tiene una misión importante en el sistema, los roles que lo extienden son sus colaboradores para alcanzarla. Los roles se alinean a las metas del actor.

3: Diseño e Implantación

En esta sección se abordan algunos temas relevantes respecto al diseño de Mophoua. Se omiten aquellos aspectos de diseño que son bien conocidos en la escritura de compiladores y se exponen solo aquellos que representan una situación específica al diseño de Mophoua.

Indentación.

Mophoua utiliza la indentación para definir los bloques estructurales, de forma similar a Python. Un incremento de indentación inicia el bloque y un decremento lo termina.



Es tarea del Analizador Léxico (Lexer), emitir los léxicos (Tokens) imaginarios INDENT y DEDENT, cuando descubre un cambio de indentación. Sin embargo, a diferencia de Python, la gramática de Mophoua considera al carácter nueva línea '\n' como un espacio en blanco, es decir lo ignora. La separación entre sentencias se realiza con el punto y coma ';'. En analizador léxico de Mophoua emite un ';' al final de cada línea que contiene texto, de esta forma, es opcional para el programador utilizar el punto y coma.

La estructura sintáctica de mopohua se identifica por niveles de

indentación (el primero es el número 1). En el primer nivel de indentación, inician las definiciones de clase, en el segundo, los miembros de clase (atributos y operaciones) y en el tercero el cuerpo de funciones.

Considerando el siguiente programa fuente entregado al compilador:

```
$clase
--- metodo()

    sentencia
    sentencia ;;;
    sentencia ;
    sentencia ; sentencia ; sentencia ;
```

El analizador léxico entrega algo parecido a lo siguiente al analizador sintáctico (parser):

```
$clase
--- metodo()

INDENT   sentencia ;
          sentencia ;;;;
          sentencia ;;
DEDENT   sentencia ; sentencia ; sentencia ;;
```

Este es el código Java que realiza la tarea de insertar INDENT y DEDENT. Primero mide la la profundidad de la indentación (depth) después de encontrar un caracter '\n'.

```
int depth=0; // text is it in the firts columnn
while(input.LA(1) == ' ' ){
    depth++;
    input.consume();
}
```

Al aumentar el nivel de profundidad emite un INDENT.


```
if ( depth > indent[currentlevel] ) {  
    if ( input.LA(1) != '/' && input.LA(2) != '*' ) {  
        indent[++currentlevel] = depth;  
        emit(INDENT,"INDENT",input.getLine(),depth+1);  
    }  
}
```

Al disminuir el nivel de profundidad, verifica que corresponda a uno de los niveles de indentación anteriores, y emite los DEDENT necesarios según el número de niveles decrecidos.

```
else {  
    // emite un INDENT por cada nivel decrecido  
    while ( depth < indent[currentlevel] ) {  
        emit(DEMENT,"DEMENT",input.getLine(),depth+1);  
        currentlevel--;  
    }  
    // verifica que la profundidad corresponda a un nivel  
anterior  
    if ( depth != indent[currentlevel] ){  
        System.out.println("line " + input.getLine() +  
";:"  
        + (depth+1) + " Incorrect DEDENT level" );  
    }  
}
```

Mophoua permite iniciar un bloque después de ':'

```
$clase  
--- método()  
  
    // sentencia if  
    ? ( condición ):  sentencia  
                    sentencia  
                    sentencia
```

Para este caso el analizador léxico emite el INDENT al encontrar ':'

```
// mide la profundidad
int depth = input.getCharPositionInLine() ;
while(input.LA(1) == ' '){
    depth++;
    input.consume();
}

// if the rest of line is not empty and aren't a comment
int LA1 = input.LA(1);
if ( LA1!= '\n' && !( LA1 == '/' || input.LA(2) == '*')
    && ! (LA1=='/' && input.LA(2)=='/')){

    emit(INDENT,"INDENT",input.getLine(),depth+1);
    indent[++currentlevel] = depth;
}
}
```

Para insertar los ';' se verifica que se encuentre dentro del cuerpo de una función

```
if ( depth <= indent[currentlevel]){
    // Insert "SCOLON" in body functions
    // TODO: Use flag inFuction
    if ( currentlevel > 1 )
        emit(SCOLON,"SCOLON",input.getLine()-1,depth+1);
    else
        skip();
}
}
```

El código que se encuentra dentro de cualquier signo de agrupación, no es afectado por léxicos imaginarios.

```
if ( inMultiLine > 0 ) {
    skip();
    return;
}
```

El tratamiento de los comentarios debe ser considerado en el reconicimiento de la indentación, pues estos no deben aplicarse dichas

reglas. Un comentario de una línea equivale a un '\n' y un comentario multilínea debe equivale a espacios en blanco y los respectivos '\n'. Los comentarios multilinea /* */ realizan la verificación de indentación al cerrar el comentario.

```
int depth = input.getCharPositionInLine() ;
while(input.LA(1) == ' '){
    depth++;
    input.consume();
}
int LA1 = input.LA(1);
if ( LA1!= '\n' && !( LA1 == '/' || input.LA(2) == '*')
    && ! (LA1=='/' && input.LA(2) =='/') ){

    emit(INDENT,"INDENT",input.getLine(),depth+1);
    indent[++currentlevel] = depth;

}

skip();
```

El código Java en Antlr es mucho más simple que el que se utilizó para realizar la misma función en Eli (ver Apéndice “Implementación en Eli”) pues aún cuando Eli provee de un mecanismo para manejar estos casos, los requisitos de la sintaxis de Mophoua, requirieron realizar la implantación a mano,

Antlr v3.0 no incluye la función emit(), por razones de desempeño, pues implica que el método nextToken() verifique si hay lexicos imaginarios que lanzar primero y después tomar léxicos del flujo de entrada normal. Por lo tanto se requiere sobrecargar el método nextToken() y crear la función emit() de acuerdo a la documentación de Antlr.

```
@lexer::members {
```

```
int indent[] = new int[100];
int currentlevel = 0; // of intentation

int inMultiLine = 0; // flag to detect () [] {} inside

List<Token> tokens = new ArrayList<Token>(); // allow emit()

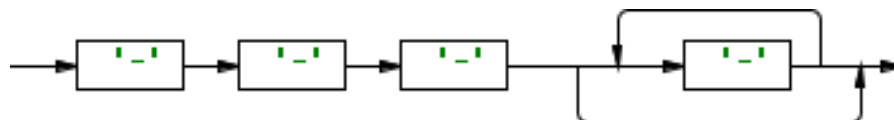
public void emit(int type, String name, int line, int col) { // insert
imaginary tokens
    // The line number it's parameter because the imaginary ';' is
reporter one line before
    Token tkn = new CommonToken(type,name);
    tkn.setLine(line);
    tkn.setCharPositionInLine(col);
    this.token = tkn;
    tokens.add(tkn);

    //System.out.println("emit(" + name + ") linea: "+ input.getLine()
+ ":" + line);
}

public void emit(Token tkn) { // insert imaginary tokens
    this.token = tkn;
    tokens.add(tkn);
}

public Token nextToken() { // allow emit()
    super.nextToken();
    if ( tokens.size()==0 ) {
        return Token.EOF_TOKEN;
    }
    return (Token)tokens.remove(0);
}
}
```

La definición de funciones inicia con el léxico LINEA1 que tiene la siguiente estructura:



El analizador léxico emite sus coordenadas en la posición del segundo carácter, esto provoca que se detecte un cambio de indentación y se emita el lexico INDENT. Con este truco, la definición de funciones se escribe en el mismo nivel que la definición de clases.

Complementariamente, el emisor de DEDENT considera a la primera y segunda columna en el mismo nivel de indentación.

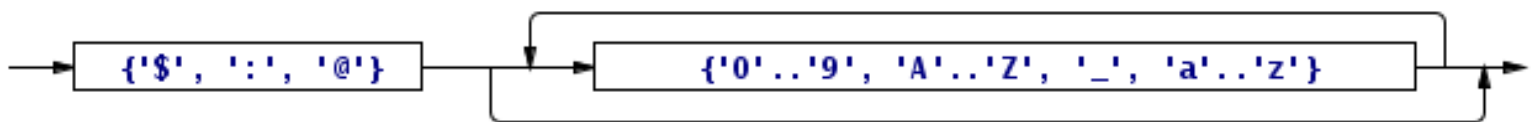
```
$clase
--- metodo ()
--- metodo ()
```

Identificadores

Existen dos espacios de nombres para los identificadores en mophoua, uno para los tipos de datos y otro para funciones, variables, constantes y etiquetas. Los identificadores de tipo poseen una sintaxis especial pues inician con los caracteres '\$' ':' o '@' seguido de una secuencia de caracteres.

```
ID_TIPO:  ( '$' | ':' | '@' ) ( 'A'..'Z' | 'a'..'z' | '0'..'9' | '_' ) *
;
ID       :  ( 'A'..'Z' | 'a'..'z' ) ( 'A'..'Z' | 'a'..'z' | '0'..'9' | '_' )
*;
```

Sintaxis de identificadores de tipo:



Esta sintaxis permite identificar fácilmente los tipos de datos, tanto en el código fuente como en la documentación. Así, al referirse a los identificadores *:persona* y *persona*, el primero es identificado inmediatamente como un tipo.

Por regla general la notación `:tipo` es para referirnos a un tipo, y la notación `$tipo` es para referirnos a una instancia de tipo. De esta forma no se utiliza el operador `new`.

```
x      $int          // declara x como tipo :int además de
instanciar un objeto
y      :int          // y es una referencia a :int

y = x          // inicializa y
y = $int       // crea una nueva instancia de :int
```

Esta notación tiene la ventaja de simplificar la aplicación del operador `typeof`, para evaluar si una referencia pertenece a una clase dada.

```
? ( numero == :int ) // aplica el operador typeof
```

Otra ventaja es la aplicación del operador `cast`

```
numero :float        // declara una variable tipo :float
edad :int            // declara edad como tipo :int

edad = numero :int   // aplica el operador cast a numero
```

Y sobre todo, la aplicación y definición de roles, que es un tema importante para Mophoua:

```
yo      $persona     // crea una instancia de persona
yo      :profesor    // asigna el rol de :profesor a yo que es
de tipo :persona

tu      $persona :profesor // instancia una :persona con el rol
de :profesor
```

Clases y Roles

La definición de una clase inicia con un identificador de tipo escrito en la primera columna de la línea, el siguiente código ejemplifica la sintaxis requerida para escribir miembros de clase:

```
$persona
    edad $int
    estatura $int
--- saludar()
--- metodo()

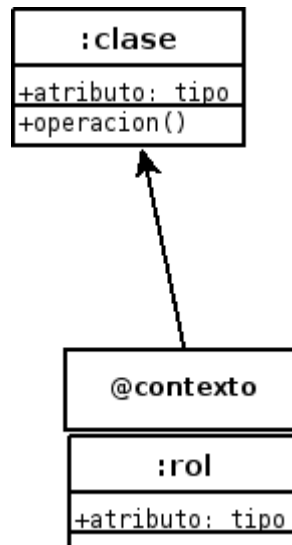
$escuela
    nombre $string
```

El código anterior define dos clases, *\$persona* y *\$escuela*. Si se hubiera querido definir solo interfaces se utiliza la notación *:persona* y *:escuela*.

Definamos ahora el rol *:profesor* que puede ser jugado por *\$persona* en el contexto de una *:escuela*

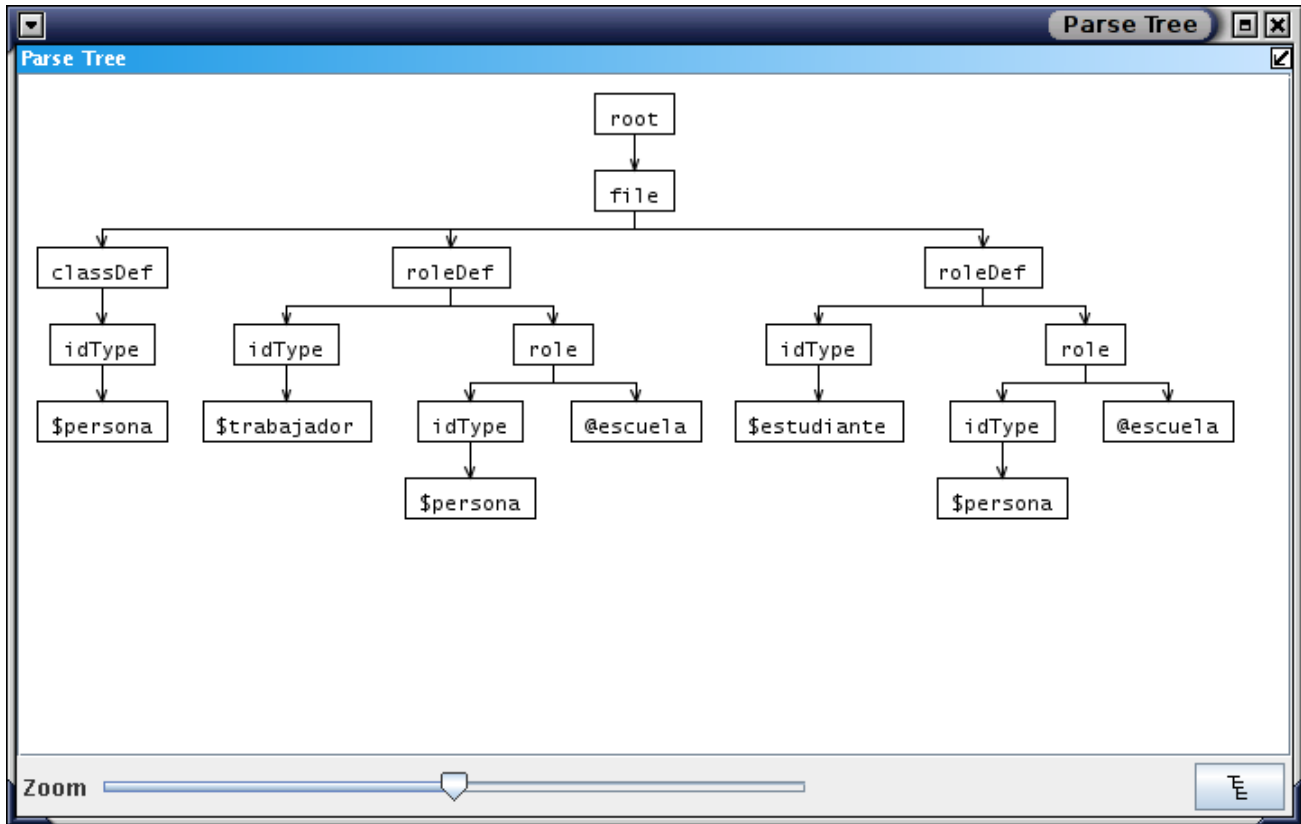
```
:profesor $persona@escuela
    codigo $int
--- trabajar()
```

A efectos de implementación un rol es una relación entre entidades o tipos que deben trabajar con bajo una misma identidad. Un diagrama propuesto para describir esta relación se puede apreciar en la siguiente figura.

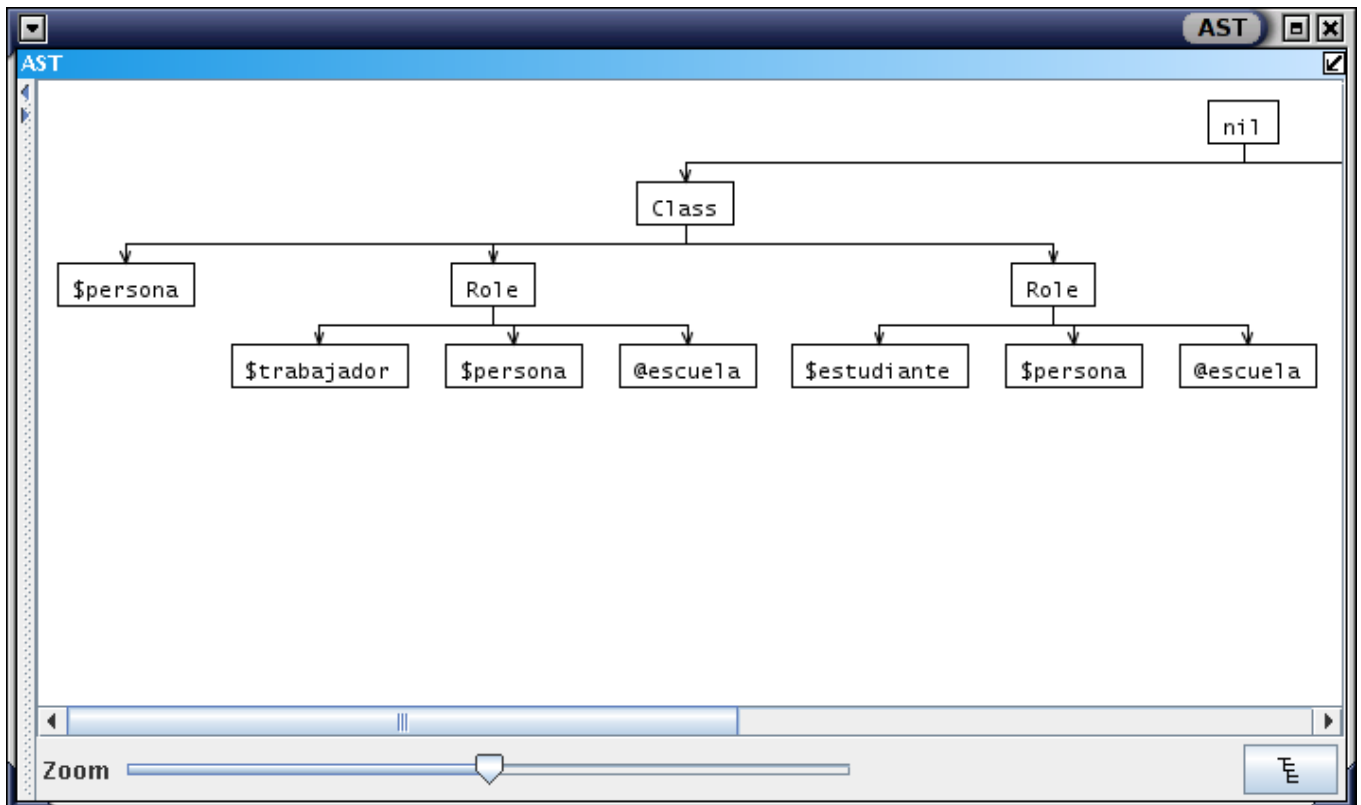


Nótese que al aplicar la notación `@tipo` convierte al `:tipo` en el contexto de un rol.

El analizador sintáctico crea un árbol de reconocimiento dónde la definición de los roles es separada de la definición de la clase que los puede jugar:



Con esta entrada, el analizador sintáctico crea un árbol abstracto dónde las clases incluyen como hijos cada definición de rol que pueden jugar:



El código necesario para realizar ésta tarea en Antlr es el siguiente:

```
file options{ backtrack= true; }
:(c=classDef
  {
    // lo agrega a la lista
    // Agrega la clase a la lista, los roles seran agregados a
    esa lista.
    CommonTree t = (CommonTree)c.getTree();
    class_list.add(t);

  }
| r=roleDef
  {
    CommonTree tRol = (CommonTree)r.getTree();

    class_list.add(tRol);

    // role nodes are inserted in her class actor
```

```
for ( CommonTree cur: class_list )

    if(
cur.getChild(0).toString().compareTo(tRol.getChild(1).toString() ) == 0)
        adaptor.addChild(cur, tRol);

    }
)+ ;
```

La generación de código a partir de éste árbol es sencilla, pues se limita a recorrer el árbol y escribir clases anidadas en Java que es el código objeto. Antlr facilita la escritura con la librería StringTemplate. A la fecha de escribir este reporte la generación de código está en proceso de escritura.

Sentencias

Conceptualmente mopohua solo reconoce dos tipos de estructuras de control, la selección y el ciclo, aún cuando cada tipo de estructura puede tomar distintas formas. El símbolo '?' se utiliza para iniciar la construcción de una sentencia de selección, y los ':' para indicar el inicio del bloque controlado. El siguiente ejemplo muestra un caso de selección simple.

```
? edad > 18 :
    sentencia
    sentencia
```

La estructura equivalente a la sentencia switch o selección múltiple tiene la siguiente forma.

```
?* edad :
```

```
** 1 .. 10 :      sentencia
                sentencia
                sentencia

** 18:           sentencia
```

Esta notación representa un problema en la implantación del analizador sintáctico, pues la diferencia entre la forma de selección simple y selección múltiple, no se encuentra sino hasta después del léxico ':'. Por lo pronto la implementación de estas sentencias se realiza utilizando la opción backtrack de Antl, que implica un considerable impacto en el desempeño, posteriormente puede optarse por una implantación a nivel léxico, donde el analizador léxico utilice un LA(*) hasta encontrar el símbolo después de ':' e inserte un léxico imaginario después de '?'. También existe la posibilidad de cambiar la gramática para que se construya fácilmente por la izquierda. Por lo pronto, para los objetivos del prototipo la solución tomada es satisfactoria.

Existe una situación similar con las estructuras de ciclo, que utilizan el carácter '@' para iniciar su construcción.

```
@ cont < 10 :    // ciclo while
    sentencia

@ x = 1, x+1 ... 10 : // ciclo for
```

El analizador léxico requiere encontrar un ':' para reconocer un while, y encontrar una ',' para reconocer un for, pues el signo de '=' es un operador válido en las expresiones de Mophoua. En una versión posterior se estudiarán estos casos para ofrecer una solución mejor.

Resolución de métodos

Un problema para implantar un lenguaje orientado al contexto es definir para cada referencia, el conjunto exacto de métodos que pueden ser solicitados.

En la programación modular, lenguajes como C, existe una correspondencia directa, pues cada operación o solicitud se asocia directamente con un método.

N= Conjunto de operaciones o solicitudes

M= Conjunto de métodos

$N \rightarrow M$

El polimorfismo requiere un mecanismo de resolución de métodos, la tiempo de respuesta está en función del número de clases polimorfas, el número de operaciones y el número de métodos.

C= Conjunto de clases polimorfas.

Modelo: $[(N \times C) \rightarrow M]$

En un modelo de roles, la función se complica.

R = Conjunto de roles

[(NxCxR) -> M]

El compilador debe conocer el nombre de la referencia, la clase de objeto a la que apunta(polimorfo) el rol que desempeña, con esto localiza el método a ejecutar.

Se están buscando soluciones que puedan mejorar está situación.

Símbolos

Uno de los objetivos de Mophoua es permitir una transparencia de la sintaxis con la semántica del lenguaje y con el modelo del contexto subyacente. La estrategia tomada para lograrlo se basa en una notación simbólica y estructural del texto. En el caso de la sintaxis de tipos, este objetivo se alcanza plenamente, con el uso de los símbolos '@', ':' y '\$' se cubre una amplia gama de situaciones para la operación con tipos, al mismo tiempo que la implementación del compilador es simple.

En el caso de las estructuras de control, la situación no es muy clara, pues aún cuando los símbolos '?' y '@' permiten una identificación directa, la posibilidad de realizar combinaciones anidadas, repercute en algunos casos en la claridad del código. Con la ayuda de los editores que reconocen la sintaxis, posiblemente esta situación no sea tan grave, se requiere aún de continuar la investigación en este sentido para explorar diversos escenarios y sus repercusiones en la facilidad de lectura y escritura de código fuente utilizando símbolos.

Otro problema es que las sentencias de control no se están optimizadas para una construcción por la izquierda, impactando en el desempeño del

compilador.

Existe además un conflicto en el uso del símbolo '@', pues se había asignada originalmente a los ciclos. Recientemente, se incorporó para describir contextos, esta incorporación significo el éxito de la notación de tipos. Sin embargo, no se ha eliminado de la notación de ciclos porque aún se está buscando que carácter símbolo o construcción sintáctica puede suplirlo.

No se descarta la posibilidad que el lenguaje incluya palabras reservadas para describir las estructuras de control y mantener una notación simbólica para el manejo de tipos. Sin embargo, esa es una desición que será tomada de acuerdo con los resultados de la investigación. El estado actual de Mopohua es proporcionar un marco que apoye dicha investigación, por lo tanto la opción es la implementación simbólica.

Eli

Eli⁶ es un compilador de compiladores [Kantens 2007] que utiliza gramáticas diferentes para fase del proceso de creación de compilador.

La creación del parser o analizador sintactico se realiza en archivos CON

⁶ <http://eli-project.sourceforge.net/>


```
*/
sentencia  : asigna      term
           / expr      term
           / seleccion
           / ciclo      term
           / romper     term
           / continuar  term
           / entrada    term
           / salida     term
           / bloque     term
           / '!'        term
           / retorno    term
           .

bloque     : '<' ID '>' [ secuencia fin ]
           / '!<' ID '>' [ secuencia fin ].

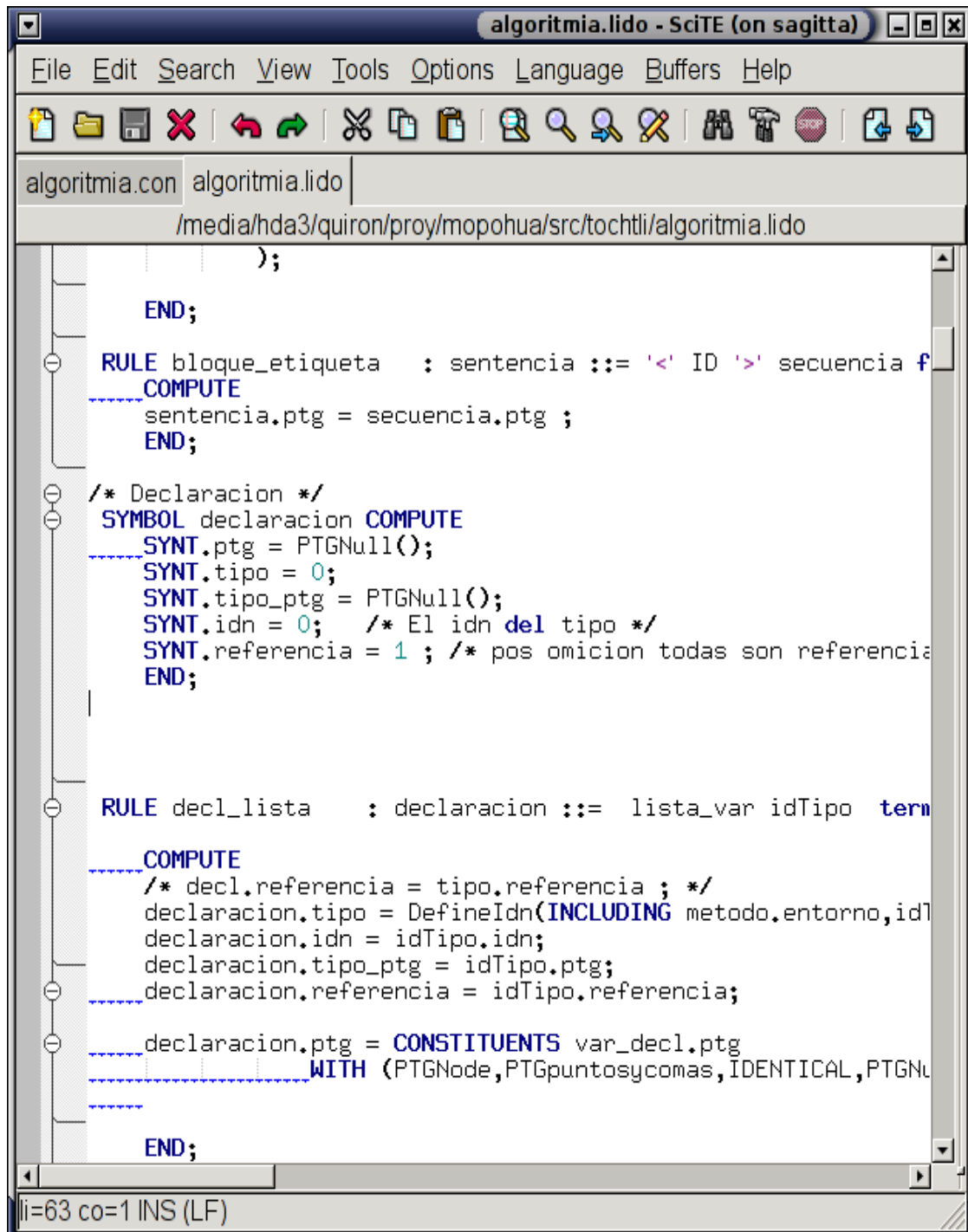
/* las asignaciones requieren una revisión de reglas de producción */
asigna    :
           elem asigna_op expr
           / idUso ':' idTipo /* Operador new */
           / idUso ':' '!' /* asignacion a nulo */
           / idUso '$=' expr /* clonacion */
           / nombre '[' expr ']' ':' expr
           / idUso ':' expr
           .

/*
Seleccion
=====
*/
```

li=1 co=1 INS (LF)

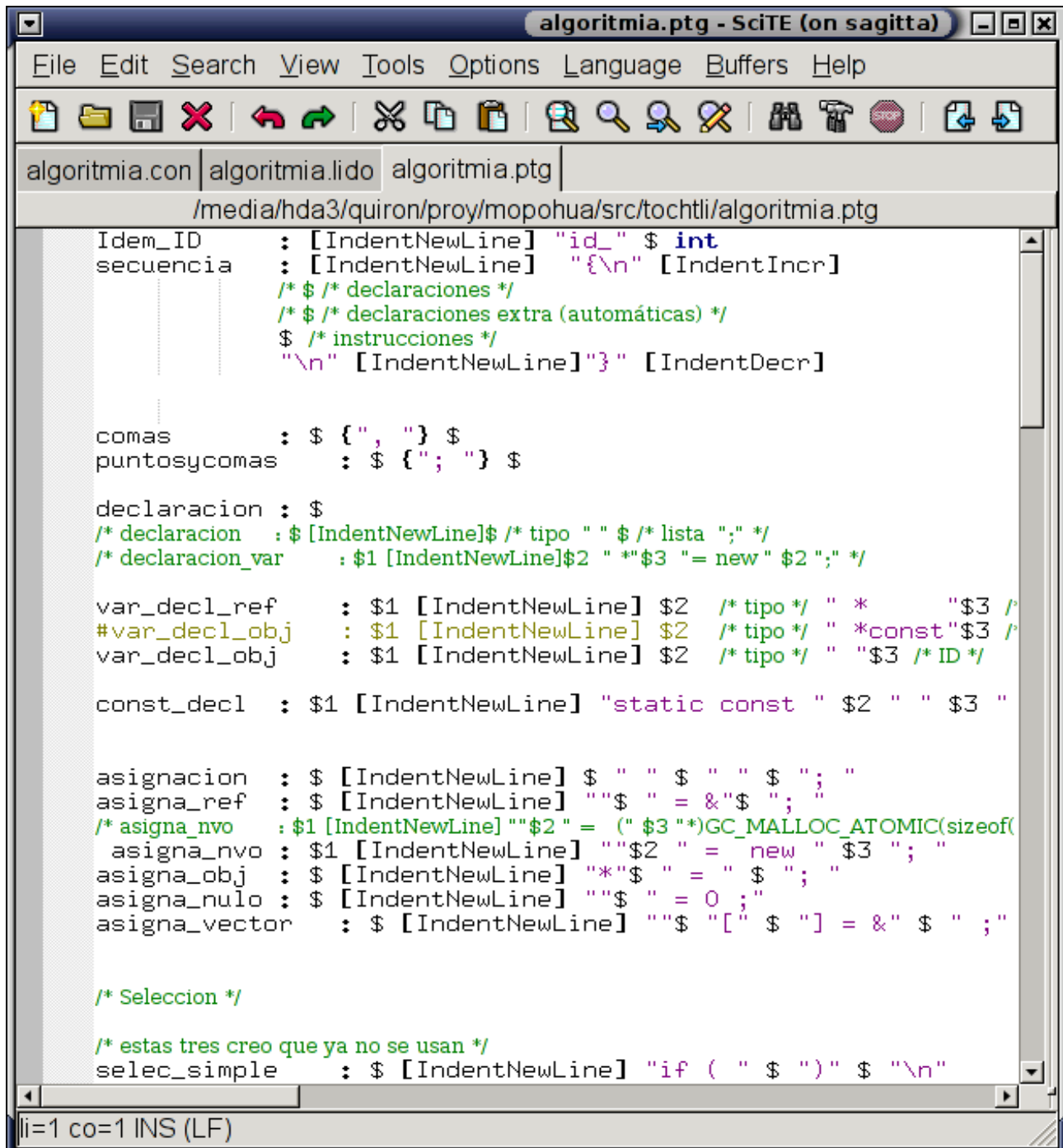
El lenguaje para crear árboles abstractos se denomina LIDO, al mismo

tiempo se puede adornar el árbol, utilizando dependencias.



```
);  
  
    END;  
  
    RULE bloque_etiqueta  : sentencia ::= '<' ID '>' secuencia f  
    COMPUTE  
    -----  
    sentencia.ptg = secuencia.ptg ;  
    END;  
  
    /* Declaracion */  
    SYMBOL declaracion COMPUTE  
    -----  
    SYNT.ptg = PTGNull();  
    SYNT.tipo = 0;  
    SYNT.tipo_ptg = PTGNull();  
    SYNT.idn = 0; /* El idn del tipo */  
    SYNT.referencia = 1 ; /* pos omision todas son referencia  
    END;  
  
    RULE decl_lista      : declaracion ::= lista_var idTipo term  
  
    COMPUTE  
    -----  
    /* decl.referencia = tipo.referencia ; */  
    declaracion.tipo = DefineIdn(INCLUDING metodo.entorno,idT  
    declaracion.idn = idTipo.idn;  
    declaracion.tipo_ptg = idTipo.ptg;  
    -----  
    declaracion.referencia = idTipo.referencia;  
  
    -----  
    declaracion.ptg = CONSTITUENTS var_decl.ptg  
    -----  
    WITH (PTGNode,PTGpuntosycomas,IDENTICAL,PTGNu  
  
    END;  
  
li=63 co=1 INS (LF)
```

La generación de código se realiza con un sistema de plantillas que se definen con un lenguaje llamado PTG.



```
Idem_ID      : [IndentNewLine] "id_" $ int
secuencia   : [IndentNewLine] "{\n" [IndentIncr]
             /* $ /* declaraciones */
             /* $ /* declaraciones extra (automáticas) */
             $ /* instrucciones */
             "\n" [IndentNewLine]"}" [IndentDecr]

comas       : $ {" , " } $
puntosycomas : $ {" ; " } $

declaracion : $
/* declaracion  :$ [IndentNewLine]$ /* tipo " " $ /* lista ";" */
/* declaracion_var :$1 [IndentNewLine]$2 " *"$3 "= new" $2 ";" */

var_decl_ref   : $1 [IndentNewLine] $2 /* tipo */ " * " "$3 /*
#var_decl_obj  : $1 [IndentNewLine] $2 /* tipo */ " *const"$3 /*
var_decl_obj   : $1 [IndentNewLine] $2 /* tipo */ " "$3 /* ID */

const_decl    : $1 [IndentNewLine] "static const " $2 " " $3 "

asignacion    : $ [IndentNewLine] $ " " $ " " $ ";" "
asigna_ref    : $ [IndentNewLine] ""$ " = &"$ ";" "
/* asigna_nvo :$1 [IndentNewLine] ""$2 " = (" $3 *)GC_MALLOC_ATOMIC(sizeof(
asigna_nvo    : $1 [IndentNewLine] ""$2 " = new " $3 ";" "
asigna_obj    : $ [IndentNewLine] ""*$ " = " $ ";" "
asigna_nulo   : $ [IndentNewLine] ""$ " = 0 ";" "
asigna_vector : $ [IndentNewLine] ""$ "[" $ "]" = &" $ ";" "

/* Seleccion */

/* estas tres creo que ya no se usan */
selec_simple  : $ [IndentNewLine] "if ( " $ ")" $ "\n"
```

li=1 co=1 INS (LF)

Eli genera analizadores tipo LALR(1) en código C que puede ser compilado en el C original de Kernigran, por lo que son muy eficientes y portables. Sin embargo, para las necesidades de desarrollo de Mophoua, se requiere un rápido prototipado. Cosa que puede ser complicada en Eli por algunos requisitos especiales que requiere la gramática de Mophoua.

Antl y AntlWorks

Antlr⁷ es un generador de reconocedores de lenguajes [Parr 2007], utiliza una estrategia de reconocimiento que denomina LL(*) que es una extensión de LL(k), dónde k se extiende arbitrariamente según sea necesario por la gramática. Esta implementado en Java y generacódigo Java, C#, Python entre otros.

Cuenta con una IDE de nombre AntlrWorks que asiste en el diseño de gramáticas y auxilia en su depuración.

⁷ <http://www.antrl.org/>

The screenshot displays the Mopohua IDE interface. The top window shows a grammar file with the following content:

```
242 assignment
243     //:      expression ; // join java expression ;
244     :      id '='^ expr term!;
245
246
247 ifStat // big locked! in LL(*)
248     :      '?' expr COLON term? block?
249     ->     ^(If expr block?)
250     ;
251
252 switchStat
253     :      '?' expr term? (caseBlock* defaultBlock)?
254     ->     ^(Switch expr (caseBlock*)? defaultBlock? )
255     ;
256 //case_list:      case_block* ;
257
258 caseBlock:      '**' expr COLON term? block
259     ->     ^(Case expr block);
260
261 defaultBlock
262     :      '**' COLON term? block
263     ->     ^(Case block);
264
265 whileStat
266     :      ARROBA expr COLON block?
267     ->     ^(While expr block?);
268
269 doWhile:      ARROBA '...' expr COLON block
270     ->     ^(DoWhile expr block);
271
272 forStat
273     options{
274         backtrack = true;
```

The left sidebar shows a project tree with categories like 'classes', 'Algorithmia', 'Expressions', and 'Indent'. The 'classes' category is expanded, showing various grammar elements like 'file', 'roleDef', 'classDef', 'inheritance', 'role', 'idType', 'attribute', 'method', 'params', 'param', 'IDClass', 'IDRef', 'IDContext', and 'LINE1'. The 'Algorithmia' category is also expanded, showing 'stat', 'assignment', 'ifStat', 'switchStat', 'caseBlock', 'defaultBlock', 'whileStat', 'doWhile', 'forStat', 'forStatC', 'varUse', 'block', and 'term'. The 'Expressions' category is expanded, showing 'NL' and 'COLON'. The 'Indent' category is expanded, showing 'NL' and 'COLON'.

Below the code editor, there is a 'Zoom' slider and a 'Show NFA' checkbox. The NFA diagram is displayed below the slider, showing the following states and transitions:

```
graph LR
    S(( )) --> E1[']
    E1 --> E2[expr]
    E2 --> C[COLON]
    C --> S1[']
    S1 --> T[term]
    T --> B[block]
    B --> E1
    C --> S2[']
    S2 --> T
    T --> B
    B --> E1
```

The NFA diagram shows a sequence of states: ']' (start), 'expr', 'COLON', ']' (loop), 'term', and 'block'. Transitions are: ']' to 'expr', 'expr' to 'COLON', 'COLON' to ']' (loop), ']' (loop) to 'term', 'term' to 'block', and 'block' to ']' (loop). There is also a direct transition from ']' to 'term'.

At the bottom of the IDE, there are tabs for 'Syntax Diagram', 'Interpreter', 'Debugger', and 'Console'. The 'Syntax Diagram' tab is active, showing '104 rules', '258:16', and 'Writable'.

IDEs para Mophoua

Se ha codificado un analizador sintáctico en C para extender el editor scite⁸ y que permita el reconocimiento de la sintaxis de Mophoua. Aunque se cree conveniente iniciar en un futuro, la creación de un plugin para el entorno de desarrollo Eclipse.

En la siguiente imagen se muestra un archivo de la versión de Mophoua implementada en Eli. Que incluye bloques etiquetados '<'>' para reconocimiento de secciones. El editor permite colapsar los bloques con base a la indentación y el reconocimiento de la gramática. SciTE permite la compilación directa de Mophoua con solo pulsar 'F5'.

⁸ <http://www.scintilla.org/SciTE.html>

```

burbuja.cx
/media/hda3/quiron/proy/mopohua/ejem/burbuja.cx
ordenador de enteros

/* Utiliza el método de ordenación por burbuja */

=== Ordenar()

a[cur] $vector[ $int]
tmp $int
algo, x,v $int
n :int

<Crea un arreglo de numeros>

    @ algo = 0 ... 9:|
      n := $int
      n = algo
      a[-] <<= n

<Impreme original>
    <<< "Original ",
    @ x = 0 ... 9 :
      <<< a[x] , " ",
      <<< " "

<Los ordena>
    @ v = 8, v-1 ... -1:
      @ n = 0 ... v :
        ? a[n] > a[n+1] :

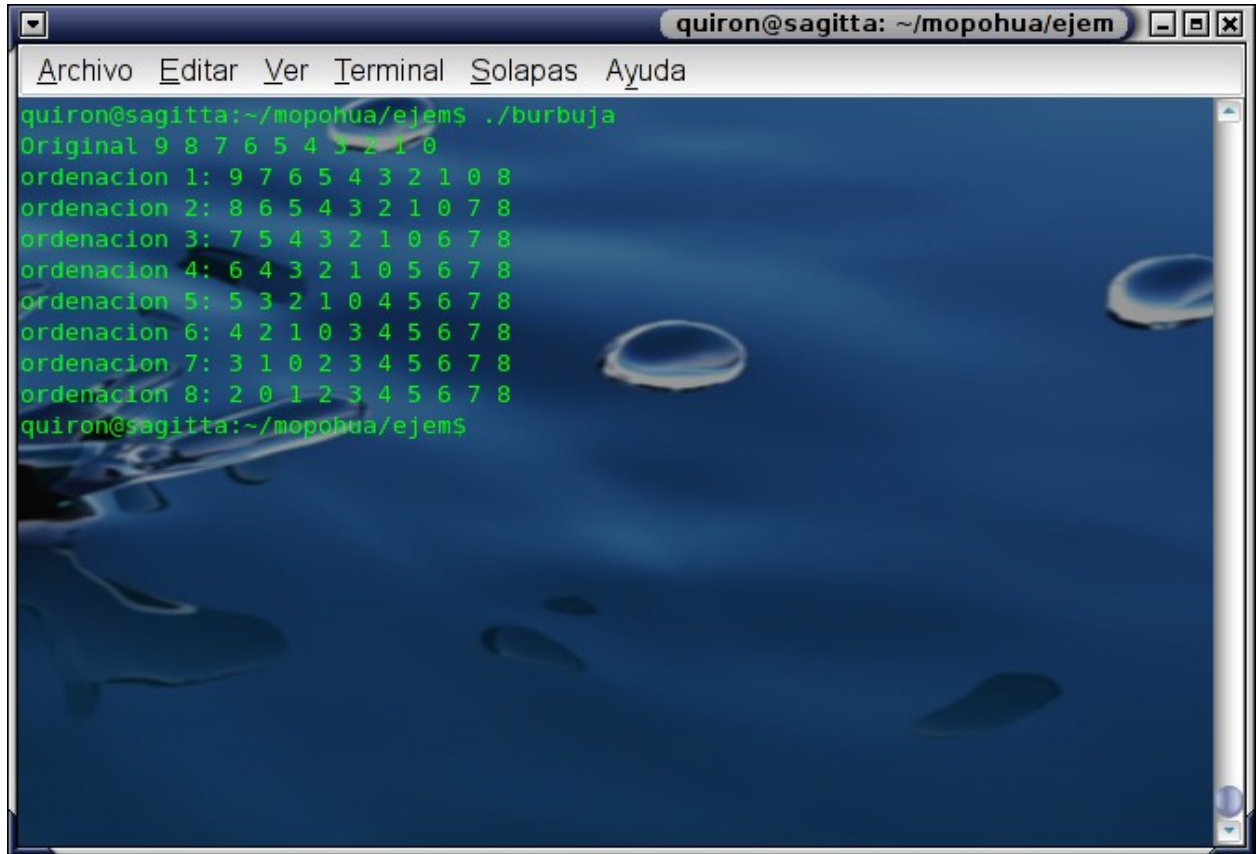
          // Invertir
          tmp = a[n]
          a[n] = a[n+1]
          a[n+1] = tmp

    <Impreme ordenado>
      <<< "ordenacion ", 9-v, ":",
      @ x = 0 ... 9 :
        <<< a[x] , " ",
        <<< " "

```

li=14 co=26 INS (LF)

Este es el resultado de la ejecución del programa binario generado:



```
quiron@sagitta: ~/mopohua/ejem
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
quiron@sagitta:~/mopohua/ejem$ ./burbuja
Original 9 8 7 6 5 4 3 2 1 0
ordenacion 1: 9 7 6 5 4 3 2 1 0 8
ordenacion 2: 8 6 5 4 3 2 1 0 7 8
ordenacion 3: 7 5 4 3 2 1 0 6 7 8
ordenacion 4: 6 4 3 2 1 0 5 6 7 8
ordenacion 5: 5 3 2 1 0 4 5 6 7 8
ordenacion 6: 4 2 1 0 3 4 5 6 7 8
ordenacion 7: 3 1 0 2 3 4 5 6 7 8
ordenacion 8: 2 0 1 2 3 4 5 6 7 8
quiron@sagitta:~/mopohua/ejem$
```


5: Conclusiones

Mopohua es un proyecto que ha dado sus primeros pasos con bases firmes, aún se encuentra en una fase temprana de su desarrollo. Las investigaciones del modelado de sistemas orientado a roles y la programación orientada a aspectos, evolucionan rápidamente. Una gramática específicamente diseñada para un nuevo modelado ofrece una mejor relación conceptual con el diseño de roles, que la extensión de gramáticas existentes. Quedan muchos temas para futuras investigaciones. Entre otros, concurrencia, sesiones, protocolos, máquinas de estados, dependencia entre roles, Aspectos y los temas relacionados con la enseñanza y el aprendizaje.

En el campo educativo Mopohua abre igualmente oportunidades de investigación, pues su estrategia de diseño de sintaxis minimalista, puede ser útil para todos aquellos que se inician en el estudio de la programación. Además, el modelo subyacente es suficientemente simple para ser comprendido por nuevos programadores.

La formación de la comunidad es trabajo que ha dado inicio con los primeros interesados, con el objetivo de ganar la mayor cantidad de usuarios, su lanzamiento será ofreciendo una versión de funcionalidad mínima pero suficiente para exponer el alcance del proyecto, y esta se espera realizar en los primeros meses del 2008.

- [Reenskaug 1996] Trygve Reenskaug, *Working with objects. The Ooram Software Engineering Method*, 1996
- [Steimann 2000] Friedrich Steimann, *On the representation of roles in object-oriented and conceptual modelling*, 2000
- [Balzer 2007] Stephanie Balzer and Thomas R. Gross, *Member Interposition: How Roles Can Define Class Members*, 2007
- [Loebe 2007] Frank Loebe, *Towards a Definition of Roles for Software Engineering and Programming Lang*, 2007
- [Baldoni 2006b] Matteo Baldoni, Guido Boella, and Leendert van der, *Modelling the Interaction between Objects: Roles as Affordances*, 2006
- [Herrmann 2007] Stephan Herrmann, Christine Hundt, Marco Mosconi, *ObjectTeams/Java Language Definition version 1.0. Technical Report*, 2007
- [Herrmann 2002] Stephan Herrmann, *Object Teams: Improving Modularity for Crosscutting Collaborations.*,
- [Baldoni 2005] Baldoni, M; Boella, G.; and van der Torre, L., *Rosel as a coordination construct: Introducing powerJava.*, 2007
- [Baldoni 2006] Matteo Baldoni and Guido Boella and Leendert van der Torre, *powerJava- ontologically founded roles in object oriented programming langu*, 2006
- [Kolling 1998] M. Kolling, *The design of an object-oriented environment and language for teaching*, 1998
- [Gray 1992] RW Gray, SP Levi, VP Heuring, AM Sloane, WM Waite,

Eli: a complete, flexible compiler construction system., 1992

[Parr 1995] T.J. Parr and R.W. Quong, *ANTLR: A Predicated-LL(k) ParserGenerator*,

[Browne 2005] James C. Browne, Kevin Kane and Nasim Mahmood, *Role Based Programming Systems*, 2005

[Reenskaug 2007] Trygve Reenskaug, *Roles and Classes in Object Oriented Programming*, 2007

[Genovese 2007] Valerio Genovese, *A Meta-model for Roles: Introducing Session*, 2007

[Chernuchin 2005] Chernuchin, D. and Ditrich, G., *Role Types and their Dependencies of Natural Types.*, 2005

[Gamma 1995] Gamma E, Helm R, Jhonson R, Vlissides J, *Design Patterns*, 1995

Mini tutorial de Mophoua

Tanto si eres un programador experimentado como nuevo en el apasionante mundo del desarrollo de software. Te invito a que conozcas la particular forma de escribir programas en Mophoua.

Objetos y Clases

Mophoua es un lenguaje orientado a objetos, por lo tanto para escribir el programa *Hola Mundo*, se requiere primero tomar dos decisiones, el nombre del objeto y el nombre de lo que hará:

```
$saludador  
=== Saludar()  
    <<< "Hola mundo"
```

\$saludador es el nombre de una **clase** o tipo de dato, se reconoce por el caracter especial '\$' como parte del nombre y al inicio de éste. *Saludar()* es una operación principal (equivalente a `main()`), es decir en ese punto comenzará la ejecución del programa, se reconoce por '=== ' antes del nombre. Finalmente, el operador '<<<' escribe el texto "Hola mundo" en la salida estándar del objeto.

Para crear nuevos tipos de datos o clases en Mophoua, basta con escribir un nombre de clase en la primera columna de un renglón. El siguiente archivo define 3 nuevas clases.

```
$persona
$gato
$saludador
```

Las clases anteriores no tienen **miembros**, por lo que conviene definir algunas operaciones o métodos.

```
$persona
--- Hablar()
--- Comer()

$gato
--- Maullar()

$saludador
--- Saludar()
```

Aún así, las clases todavía son inútiles, pues las operaciones no incluyen ninguna instrucción. Mophoua facilita escribir tu código por refinamientos progresivos.

```
$persona
--- Hablar()
    <<< "Bla Bla"

--- Comer()
    <<< "Chomp Chop"

$gato
--- Maullar()
    <<< "Muiauuuu"

$saludador
--- Saludar()
    <<< "Hola Mundo Loco"
```

Las instrucciones de una operación o función, deben escribirse con una sangría que permita identificar que están “dentro” del método. Hasta aquí, tenemos 3 nuevas clases, pero con ellas no se ha construido ningún objeto. Así que agregaremos una clase con una función principal y objetos.

```
$teatro
  pepe $persona
  juan $persona
  fufu $gato
  yo   $saludador

=== Actuar()

  pepe.Hablar()
  juan.Comer()
  fufu.Maullar()
  yo.Saludar()
```

En el \$teatro hay dos \$persona un \$gato y un \$saludador, y a la hora de Actuar(), se solicita a cada uno algo de lo que sabe hacer. Veamos el resultado de la ejecución del programa.

```
Bla Bla
Chomp Chomp
Miauuuu
Hola Mundo Loco
```

Roles

La característica especial que tiene Mophoua es la capacidad de programar comportamientos diferentes de los objetos para contextos diferentes.

```
$persona
--- Hablar()
```

```
<<< "Bla Bla"
```

Primero defino a una \$persona normal, que habla normal.

```
:payaso $persona@circo  
--- Hablar()  
    <<< "Quieren que les cuente un cuento? Jo Jo"
```

En el código anterior define el comportamiento especial :payaso que puede tomar \$persona en el contexto de un \$circo, al aplicar el carácter '@' en una clase, esta se utiliza como contexto.

```
$circo  
--- Actuar()  
    yo $persona // Yo soy una $persona normal  
    yo.Hablar() // Y hablo como una $persona normal  
  
    yo :payaso // El circo puede asignarme el rol de :payaso  
    yo.Hablar() // Y con él, aprendo a hablar como :payaso
```

En Mopohua, los objetos pueden aprender a comportarse de acuerdo al contexto en el que participan mediante la definición de **roles**. Esto provee de objetos más competentes en las tareas para las que se les requiere. Además, los roles se programan en el contexto, no en el objeto. Un :payaso se programa en un \$circo, y no en una \$persona.

Ejemplos

El siguiente ejemplo del algoritmo de ordenación por burbuja es un programa funcional de Tochtli, la versión anterior de Mophoua escrita en

Eli, es un prototipo para el soporte de contenedores. Incluye el operador de inserción (<<=) y cursores. Las secciones de código se diferencian por las etiquetas.

```
$Ordenacion por burbuja
=== Ordenar()

a[cur] :vector[] :int
tmp :int

<Crea un arreglo de numeros>

@ algo = 0 ... 9:
    a[-] <<= algo

<Impreme antes de ordenar>

<<< "Original \n"
@ x = 0 ... 9 :
    <<< a[x], " "

<<< "\n"

<ordena por burbuja>

@ v = 8, v-1 ... 1:
    @ n = 0 ... v :
        ? a[n] > a[n+1] :

            // Invertir
            tmp = a[n]
            a[n] = a[n+1]
            a[n+1] = tmp

<Impreme ordendado>

@ x = 0 ... 9 :
    <<< a[x], " "

<<< "\n"
```


Implementación en Eli

A efecto de comparación, se incluye un módulo de la implementación en Eli.

```
/*
 * Tochtli, Auxiliar de Indentación para el analex
 *
 * Esta es la función que permite a el analex revisar que la indentación
 * sea correcta. El analex entrega un texto con bloques que terminan con
 *
 * ; FIN ; ' Cada sentencia termina con ';'
 *
 *
 * Regresa : - ';' FIN y ';' al disminuir un nivel - ';' si está en
emismo
 * nivel. Con ':' aumenta de nivel. Si despues de los dos puntos esta
 * vacio no aumenta de nivel. Despues de dos puntos si la linea está
vacía
 * no entrega ';' . Si los ':' aparecen en los dos primeros niveles no
 * aumenta de nivel, porque lo usa 'clase' y 'funcion'
 *
 * Con '?' y '@' aumenta el nivel. No se permite que el resto de la
línea
 * este vacía.
 *
 *
 * ---- los lexicos que aumentan el nivel se definen en gla como :
 * ${\040\t} para que incluya todos los espacios en el lexico y suba el
 * nivel hasta la primera columna de texto.
 *
 *
 *
 *
 */

#include "litcode.h"                /* generado automaticamente por Eli, debe
 * contener los TERMINALES */

#include "err.h"
#include "gla.h"
#include <stdio.h>

#define MAXNIVEL 50

#define DIAGNOSTICO
#ifdef DIAGNOSTICO
```

```
#define diag(s,num) msg(s,num)
#else
#define diag(s,num)
#endif

extern char *auxNUL();
extern char *coordAdjust();

/*
 * para depuración
 */
void msg(char *s, int num)
{
    char m[250];
    sprintf(m, "%s %d", s, num);
    message(NOTE, m, 0, &curpos);
}

#if defined(__cplusplus) || defined(__STDC__)
void Indentar(char *lexico, int largo, int *tipo, int *s)
#else
void Indentar(lexico, largo, tipo, s)
char *lexico;
int largo, *tipo;
int *s;
#endif
/*
 * Entrada: lexico apuntador al buffer, donde inicia el lexico
largo
 * longitud del lexico tipo apunta al entero que almacenara el tipo
 * entregado s no se... =(
 *
 * Salida: tipo FIN si decrementa un nivel el sangrado SEP si se
 * mantiene en el mismo nivel NORETURN si se incrementa
 */
{

    static int pila[MAXNIVEL] = { 1 };
    static int *nivel = pila;

    static int parentesis = 0;

    char *antLexico;          /* nivel anterior */
    int antLin;              /* linea anterior */
    int pos;

    char cadena[10000];      /* diagnostico */
    char *tmp;               /* diagnostico */
}
```

```
/*
 * si llega al final del buffer ...
 */
if (lexico[largo] == '\0') {

    /*
     * ... llenarlo
     */
    lexico = auxNUL(lexico, largo);
    TokenEnd = lexico + largo;

    /*
     * si hay algo mas .. regresarlo
     */
    if (lexico[largo] != '\0') {
        TokenEnd = lexico;
        return;
    }
}

/*
 * al final del archivo y en el primer nivel no hay nada que hacer
 */
if (*TokenEnd == '\0' && nivel == pila) {
    *tipo = SEP;          /* regresa un ';' para la última
                          * instrucción en el primer nivel, si la
                          * hubiera */
    return;
}

antLexico = lexico;
antLin = LineNum;

coordAdjust(lexico, largo);
pos = TokenEnd - StartLine; /* numero de espacios en la línea OJO:
                             * TABs */

/*
 * -----
 */
if ( *lexico == '\n' ) diag("\n : ", nivel);
switch (*lexico) {
case '\n':

    if (parentesis)      /* entre parentesis no cuenta */

        *tipo = NORETURN;
}
```

```
/*
 * si la linea esta vacia no la cuenta
-----
 */
else if (*TokenEnd == '\n') {
    /* diag("linea vacia",*nivel); */
    *tipo = NORETURN;
} else if (*nivel == pos) {    /* mismo nivel
                               * ----- */
    /*
     * (eliminado en la version piti 06)
     *
     * if ( nivel == pila ){ /* el primer nivel no se cuenta /*
     * *tipo = NORETURN; *tipo = SEP;
     *
     * diag("nivel uno",*nivel); } else{
     */
    diag("';' separador en nivel", *nivel);
    *tipo = SEP;
    /*
     * }
     */
} else if (*nivel < pos) {    /* subio el nivel*/
    diag("subir al nivel", pos);
    if (nivel == pila + MAXNIVEL)
        message(DEADLY, "Demasiados niveles de anidacion", 0,
                &curpos);
    *++nivel = pos;
    *tipo = NORETURN;
}
else if (*nivel > pos)      {
                               /* bajo el nivel
                               * ----- */
    static int deuda = 0;    /* regresa dos lexicos ';' y
                               */
    /*
     * tarea : revisar las coordenadas
     */
    /*
     * cambiar a switch
     */
'FIN'
```

```

switch (deuda) {
case 0:
    diag("ultimo ';' del nivel", *nivel);
    deuda = 2;
    *tipo = SEP;
    break;

case 2:
    diag("FIN", *nivel);
    deuda = 1;
    *tipo = FIN;
    break;

case 1:
    diag("';' despues de Fin", *nivel);
    deuda = 0;
    nivel--;
    *tipo = SEP;
    break;
}

/*
 * if ( deuda == 1 ){
 *
 * deuda --; nivel --; *tipo = SEP;
 *
 * } if (deuda == 2){
 *
 * deuda --; *tipo = FIN; diag("';' despues de FIN",*nivel);
 *
 * }else {
 *
 * diag("FIN :",*nivel); deuda = 2; *tipo = FIN; }
 */

/*
 * Vuelve a procesar el mismo lexico por si hay que,
disminuir
 * más niveles.
 */

/*
 * comparar el nivel anterior con el nuevo nivel =====
 */

if (*nivel < pos) {
    message(ERROR, "Indentación Incorrecta", 0, &curpos);
}
/*

```

```
        * si retrocede más de un nivel
        */
        if (*nivel != pos) {
            if (nivel != pila) {

                /*
                 * vuelve a procesar el mismo léxico
                 */
                LineNum = antLin;
                StartLine = antLexico;
                TokenEnd = lexico;
            }
        }

    }
    else
        diag("ERROR aux_indent.c nignun caso al procesar \n ",
*nivel);
    break;

    /*
     * en una versión futura se puede modificar para - al entrar en
     * una llave incremente el nivel de anidación - en una anidación
     * con parntesis no entrege ';' como separador - no entrege FIN
     * despues de parentesis.
     */

    case '(':
        *tipo = PAR_IZQ;
        parentesis++;
        break;
    case ')':
        *tipo = PAR_DER;
        parentesis--;
        break;
    case '[':
        *tipo = COR_IZQ;
        parentesis++;
        break;
    case ']':
        *tipo = COR_DER;
        parentesis--;
        break;
    case '{':
        *tipo = LLA_IZQ;
        parentesis++;
        break;
    case '}':
        *tipo = LLA_DER;
        parentesis--;
        break;
```

```

case '?':
    *tipo = SELEC;
    *++nivel = pos;
    diag("'?' subiendo a ", *nivel);
    break;
case '@': // Parece que nunca entra aquí
    *tipo = CICLO;
    *++nivel = pos;
    diag("'@" subiendo a ", *nivel);

    break;
case ':':
    *tipo = DOS_PTS;
    diag("se encontraron ':' ", *nivel);
    /*
     * Este nivel esta por espacios y por num de nivel
     */
    /*
     * if (nivel - pila < 2 ) return;
     */

    /*
     * se requiere subir de nivel para permitir una sola sent
despues de ':'
     * en otro caso, el nivel debe subir con el sangrado de la
sentencia
     * en la siguiente linea
     */
    /*
     * los espacios siguientes los brinca en el analizador lexico
con
     * ':[\040\t]*'
     */
    /*
     * si el siguiente caracter es \n o /* o // no sube de nivel

     * *TokenEnd es el siguiente caracter despues del token
     */

    if (*TokenEnd == '\n') {
        diag("Omitiendo cambio de nivel, ':' linea vacia", *nivel);
        return;
    }
    /* Si hay un comentario despues de ':' no cambia el nivel,
    El siguiente caso es un error sintactico :(
    condicion: / * comentario * / sentencia
    */

```

```
else if ( *TokenEnd == '/' ) return ;
else if (parentesis) {

    diag("Omitiendo cambio de nivel, ':' parentesis", *nivel);
    return;

} else {

    /*
    * si no encuentro \n sube un nivel
    */
    if (nivel == pila + MAXNIVEL)
        message(DEADLY, "Demasiados niveles de anidacion",
0,&curpos);

    *++nivel = pos;
    diag("':' subiendo a ", *nivel);
    /*
    * diag(TokenEnd,0);
    */

}

break;

}

/*
Nota: no verifica el desbordamiento del buffer, puede ocasionar
violación de segmento

diag(cadena, *nivel);

sprintf(cadena, 30, "^^^ %s ^^^", TokenEnd);
for (tmp = cadena; *tmp != '\0'; tmp++);
*tmp = '\0';

*/

return;
}
```


GNU Free Documentation License

GNU Free Documentation License Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the

Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve

the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice

- giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
 - H. Include an unaltered copy of this License.
 - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
 - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
 - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
 - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
 - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
 - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or

by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate

and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c) YEAR YOUR NAME.  
Permission is granted to copy, distribute and/or modify this  
document  
under the terms of the GNU Free Documentation License, Version 1.2  
or any later version published by the Free Software Foundation;  
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover  
Texts.  
A copy of the license is included in the section entitled "GNU  
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the  
Front-Cover Texts being LIST, and with the Back-Cover Texts being  
LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.