El modelo de software IEC61131-3: una propuesta de implementación y primera etapa del traductor asociado.

I. Plaza
C. Medrano
Dpto de Ingeniería Electrónica y Comunicaciones
E.U. Politécnica, Universidad de Zaragoza
C/ Ciudad Escolar s/n, 4003 Teruel
{iplaza, ctmedra}@unizar.es

Resumen

En este trabajo se presenta una propuesta para la implementación del modelo de software IEC61131-3. Este modelo considera los elementos de configuración de alto nivel de los autómatas programables. Damos las guías para su implementación sobre sistemas operativos de tiempo real compatibles con POSIX, en concreto sobre RTLinux corriendo en una arquitectura PC. Al considerar la implementación del modelo haremos una crítica de los principales puntos débiles. Se describe también la primera etapa de un traductor que acepta como entrada una descripción textual según la norma IEC61131-3.

Palabras Clave: Autómatas programables, IEC 61131-3, POSIX/C.

1 INTRODUCCIÓN

Los lenguajes de programación tradicionales para autómatas programables, tales como la lógica de relés, tenían varios inconvenientes. Entre ellos podemos citar [6]: débil estructuración del software, poca facilidad para manejar estructuras de datos, limitada facilidad para construir secuencias complejas, limitado control de la ejecución de los programas, ineficiencia en re-utilización de software y diferencias entre fabricantes. El estándar IEC 61131-3 [4-6] es un intento para mejorar la calidad del software en general, incorporando nuevas características. En este trabajo nos centraremos en el modelo de software y en sus unidades de organización de programa (POU). Consideramos el caso de un único autómata programable, por lo que dejaremos de lado las funciones de comunicación en esta versión. El uso de los elementos software de la norma facilita el desarrollo de programas mejor estructurados, la reutilización de código y la incorporación de algunas ideas de ingeniería de software como la encapsulación y el ocultamiento de datos.

Entre el trabajo previo relacionado destacamos el proyecto MatPLC [7,10] como un intento de utilizar Linux en sistemas de control. Sin embargo, MatPLC define un marco más general de trabajo y no es, en su inicio, de tiempo real (aunque el paso a tiempo real se está planteando [11]). En nuestro trabajo, nos ceñiremos al estándar IEC61131-3. La concepción más extendida del estándar es la de una serie de lenguajes de programación para describir algoritmos. Sin embargo, el estándar contiene también una serie de elementos de configuración y la posibilidad de definir bloques de programación (POU) o nuevos tipos de datos de usuario. Por tanto, en el presente trabajo revisaremos brevemente los conceptos del modelo de software del IEC61131-3 y mostraremos una propuesta de implementación utilizando un lenguaje de programación de propósito general como POSIX/C y un sistema operativo de tiempo real específico, RTLinux [9], que cumple los requerimiento mínimos de Tiempo Real POSIX 1013.13. Siguiendo estas guías de implementación, se está realizando un traductor de IEC61131-3 a POSIX/C, cuya primera etapa se describirá brevemente. Finalmente, mostraremos las principales conclusiones de este trabajo.

2 EL MODELO DE SOFTWARE IEC61131-3

El modelo de software se describe en la figura 1. La figura 2 presenta un ejemplo de una configuración según la sintaxis del IEC. Repasaremos brevemente los conceptos que vamos a utilizar e implementar, y que recogen los aspectos fundamentales del estándar.

- Configuración. Es un elemento de lenguaje que corresponde al sistema de autómata programable. Consideraremos una configuración única, que se ejecuta en un PC. La configuración permite la declaración de variables globales y de recursos.
- Recurso. El recurso proporciona un soporte para la ejecución de programas. El recurso puede declarar variables globales, tareas y programas

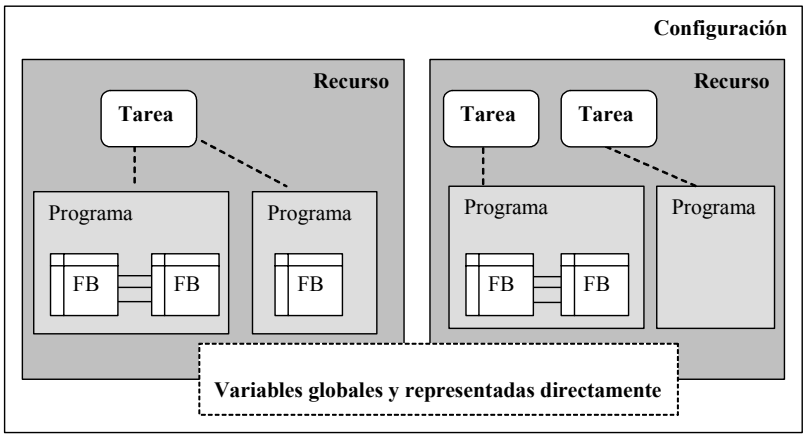

Figura 1: Modelo de software

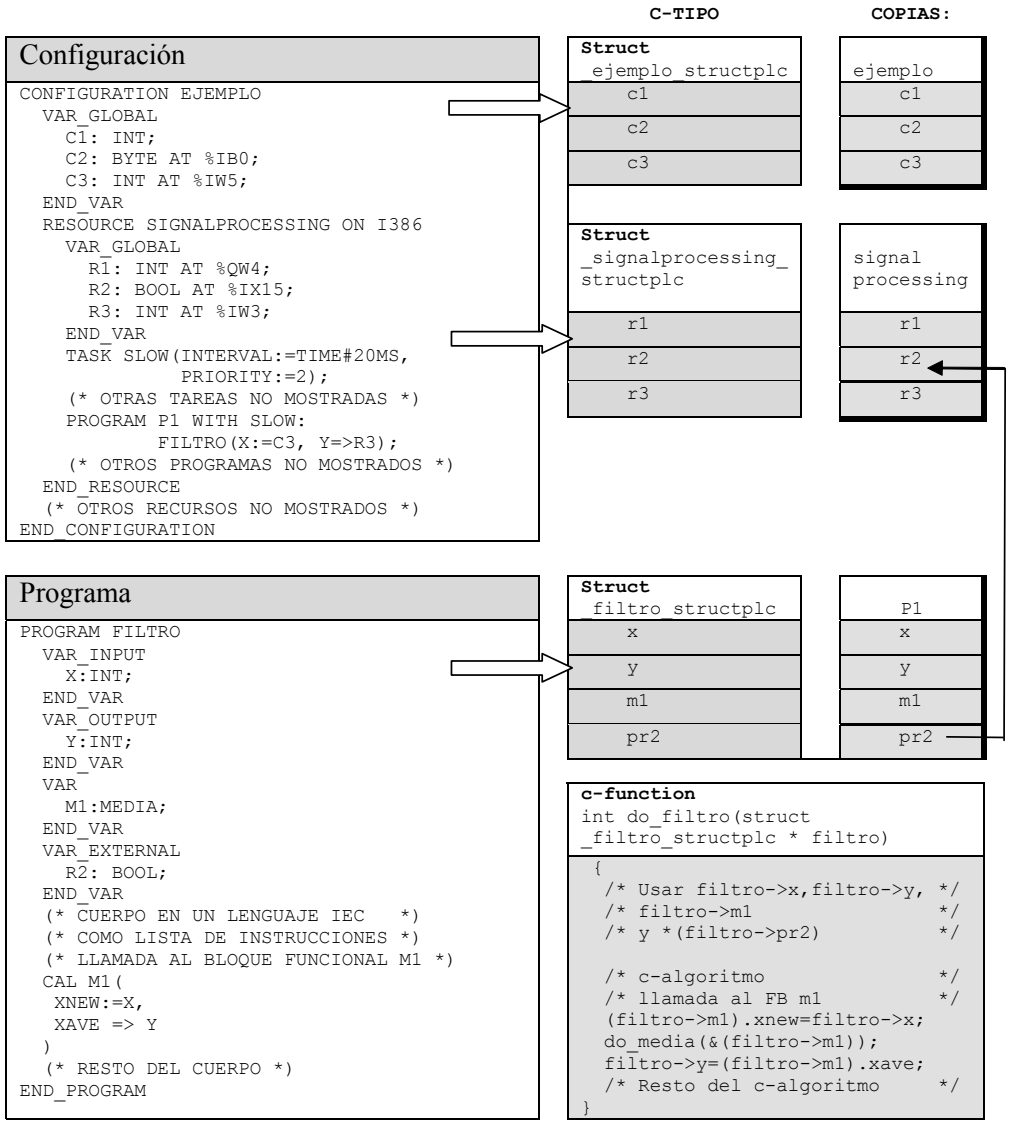

Figura 2: Ejemplo de Configuración IEC: configuración, recurso, programa y sus c-estructuras asociadas (se muestra esquemáticamente)

asociados a las tareas. El recurso puede ser asociado a un procesador determinado.

- Tarea. La tarea es el elemento que controla la ejecución de programas y de bloques funcionales. En nuestro caso, la restringiremos al control de ejecución de programas.
- Unidades de organización de programa (POU): funciones, bloques funcionales y programas. Las funciones son similares a otros lenguajes, aceptando entradas y devolviendo un valor. La definición de bloque funcional (FB) puede contener variables de entrada, de salida, externas e internas. El cuerpo del bloque funcional es un algoritmo en alguno de los lenguajes IEC. Los FBs aceptan entradas y proporcionan varias variables de salida. Las variables externas se refieren a variables globales que han tenido que ser declaradas en un programa, recurso o configuración. Dado que los FBs tienen variables internas, la llamada a un FB con las mismas entradas no siempre proporciona las mismas salidas. Se pueden declarar copias de los FBs en los programas o incluso en configuraciones y recursos; cada copia corresponde a una zona de datos distinta. El estándar establece las reglas para el acceso a las entradas y salidas de un FB, mientras que el cuerpo de la declaración de un FB constituye el algoritmo que permite modificar las salidas de una copia dada. Los programas son similares a los FBs, y son declarados en los recursos, donde se conectan sus salidas y entradas. Además, pueden declarar variables globales y usar variables representadas directamente.
- Variables globales. Pueden ser declaradas en configuraciones, recursos o programas. Esto permite su uso dentro de programas o FB siempre que hayan sido declaradas como externas en el propio programa o FB y éstos se encuentren dentro del alcance de la variable global.
- Variables representadas directamente. Las variables directas ocupan posiciones determinadas de la memoria del autómata. Sus identificadores comienzan con el carácter '%', tras el cual se indica la posición de memoria ocupada. También se les pueden asociar otros identificadores. Pueden referirse a entradas o salidas físicas, o bien a memoria interna. Su uso es inevitable para asociar variables a puntos de E/S.

Dentro de una misma configuración, la comunicación entre programas debe realizarse por medio de variables globales. Sin embargo, el uso de variables directas, al referenciar posiciones en la memoria, puede constituir una manera de comunicación indirecta. Este aspecto no es recogido explícitamente por la norma. Finalmente, tampoco queda claro si estas variables deben declararse, ya que hay ejemplos en la norma donde no lo hace.

Otro de los aspectos que no está suficientemente detallado en la norma es el de la concurrencia. Podemos encontrar estas reglas para la ejecución de bloques funcionales:

- *If a function block receives more than one input from another function block, then when the former is executed, all inputs from the latter shall represent the results of the same evaluation.*
- *If two or more function blocks receive inputs from the same function block, and if the "destination" blocks are all explicitly or implicitly associated with the same task, then the inputs to all such "destination" blocks at the time of their evaluation shall represent the results of the same evaluation of the "source" block.*

Estas reglas pueden ser extendidas a los programas, que es la clase de POU controlado por la tarea en nuestra implementación.
El estándar también define tipos de datos elementales, así como la posibilidad de que el usuario pueda definir nuevos tipos (arreglos, estructuras y enumeraciones). Para la presente discusión, baste con considerar los tipos elementales numéricos o de cadenas de bits, cuyo equivalente en C es fácil de encontrar.

3 UNA PROPUESTA DE IMPLEMENTACIÓN

Antes de entrar en detalles de implementación, clarificamos el uso de variables representadas directamente y la comunicación entre programas. Para ello, proponemos estas reglas:

1) Las variables directas sólo podrán ser usadas en la parte declarativa de configuraciones y recursos, así como en la conexión de entradas y salidas de los programas.
2) Una determinada dirección sólo podrá accederse por una variable.
3) Dado que la implementación se va a basar en un lenguaje de alto nivel y utilizando PCs, restringiremos el uso de variables directas a aquellas que tengan asociada una E/S física. Las direcciones del resto de variables serán asignadas por el sistema operativo.

De este modo, los programas sólo usarán nombres simbólicos, lo que mejorará su reutilización. Además, las únicas variables globales accesibles en todo el autómata serán aquellas explícitamente declaradas en la configuración. Como consecuencia adicional de la regla 2, podemos asociar de forma exclusiva el uso de posiciones de E/S a un recurso o programa dado.

Los conceptos introducidos por el IEC61131-3 pueden ser implementados usando un lenguaje de

propósito general. Utilizaremos POSIX/C [2], dado que sus extensiones de Tiempo Real permiten obtener la respuesta temporal deseada en sistemas de control. El prefijo 'c-' será utilizado para distinguir conceptos de programación usual en C de conceptos del IEC61131-3 que se designan con el mismo nombre. Una tarea se referirá siempre a la tarea IEC, mientras que usaremos la denominación Pthread para los hilos de POSIX.

Consideramos primero los FB, el POU más característico de los autómatas. Los FBs tienen datos y un algoritmo. Los datos pueden ser contenidos en un nuevo tipo de datos, una nueva c-estructura. El algoritmo es una c-función que recibe un puntero al nuevo c-tipo (fig. 3). De este modo, el prototipo de FB puede ser compilado a un fichero objeto. Cada copia de un FB es una nueva variable del c-tipo correspondiente, entre cuyos campos estarán, en particular, las variables internas (un estado en definitiva), y las variables externas que serán referenciadas por su dirección (puntero). En la c-estructura, todas las variables son tratadas de la misma forma independientemente de su modo (entrada, salida etc). Es misión del traductor IEC a POSIX el asegurar que son usadas convenientemente, y que, por ejemplo, una variable de entrada no se sobrescribe. La traducción de los programas se puede hacer de forma similar.

Una llamada a un FB (una copia dada) necesita por lo tanto estos pasos (figura 2):

- Rellenar los campos de entrada de la c-estructura.
- Llamar a la c-función con el puntero a la copia. Los campos de salida se actualizan dentro de la c-función.
- Actualizar las variables a las que pudiesen estar conectadas las salidas del FB.

Una configuración como la mostrada en la figura 2 contiene recursos y variables globales. En la figura 4 proponemos una división de la configuración en módulos de RTLinux. Recordamos brevemente que los módulos son programas en C con una función de inicio (utilizada en la inserción del módulo) y una función de limpieza (llamada al eliminar el módulo). Estos programas pueden ser compilados a un fichero objeto cuya inserción en el kernel de Linux permite dotarlo de nuevas capacidades, pasando los módulos a formar parte del propio kernel. RTLinux se basa en el uso de módulos. El conjunto de la configuración incluyendo los recursos podría dar lugar a un único módulo, pero la división en varios módulos proporciona cierta flexibilidad como vamos a ver a continuación.

El módulo de configuración no hace otra cosa que declarar una serie de variables globales, que podrán ser referenciadas en cualquier otro recurso (y en realidad en cualquier parte del kernel). Además, define una tabla global de funciones (GFT) donde se encuentran punteros a funciones en C, que corresponden a los diferentes tipos de programas usados en la configuración. Conviene recordar que todos los módulos comparten el mismo espacio de memoria.

La configuración pasa a formar parte del sistema operativo mismo, que proporciona un área de almacenamiento, además de muchas otras capacidades necesarias para la operación del PLC.

Cada recurso da lugar a un módulo de RTLinux. En cada uno de ellos, las tareas del recurso se implementan como Pthreads. En el caso de tareas periódicas, la periodicidad y la prioridad pueden ser asociadas al Pthread, teniendo en cuenta que la convención de prioridades es opuesta en el IEC61131-3 y en POSIX. En el caso de tareas por evento, las cuales están asociadas al flanco de subida de una variable booleana y tienen también una prioridad, se asocian a un manejador de interrupción y a un PThread. El manejador de interrupción despierta al hilo asociado. Dependiendo del hardware específico, sólo algunas variables booleanas podrán activar las tareas. Las variables globales del recurso se implementan como una c-estructura, declarada como estática (static en C) en el módulo. De este modo, es visible directamente sólo dentro del mismo recurso. Las variables globales de configuración y la GFT están definidas como externas (extern en C). Los recursos pueden estar asociados a un procesador dado. Para ello, todos los Pthreads correspondientes, serán asociados a ese procesador, usando la función de RTLinux pthread_attr_setcpu_np (no compatible con POSIX).

Para tener en cuenta los efectos de concurrencia mencionados en el estándar, utilizamos copias locales de las variables dentro de los Pthread. Las variables globales se utilizan sólo al principio y al final del algoritmo. Por tanto, el código de cada Pthread puede dividirse en varias partes. Consideremos un Pthread periódico. Las partes serían:

- Una parte declarativa donde se declaran variables locales destinadas a copiar las variables globales de recurso y de configuración. También se declararán las copias de los programas.
- Una parte de referencia de las variables externas.

Las variables externas en los programas y en los FB dentro de ellos se tratan como punteros, cuyo contenido es usado. Por ejemplo, supongamos que una variable externa es declarada en un prototipo de programa. Una vez que se crea una copia del programa, la variable externa deberá referirse a una variable global del recurso o de la configuración donde el programa ha sido instanciado.

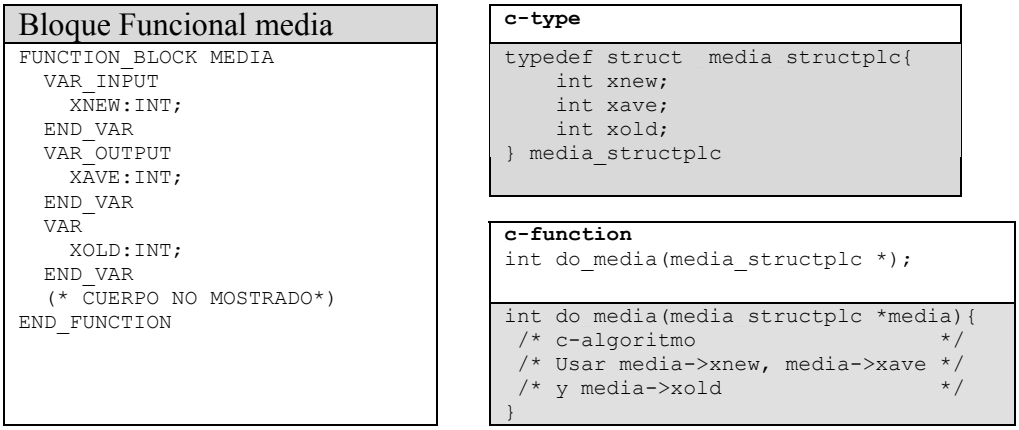

Figura 3: Bloque funcional tipo "media" y su correspondiente c-estructura y c-programa (se muestra esquemáticamente)

Es necesario establecer la dirección a la que apuntan esas variables externas. El traductor IEC a POSIX/C debe ser capaz de generar la interfaz de compilación [4] de cada POU, donde se guarda información acerca de las variables (tipo, modo, nombre).

- La tercera parte es la repetición periódica de estos pasos:

- Comunicación con un proceso gestor (para detener/arrancar el recurso, recibir comandos etc).
- Copia de variables globales en variables locales. Cuando las variables están asociadas a E/S (variables directas) deberá haber un acceso al hardware. La copia debe hacerse de forma atómica.
- Leer los punteros a funciones de la GFT. También esta operación debe ser atómica.
- Ejecutar los programas. Para ello rellenaremos los campos de entrada de la c-estructura y llamaremos a la función correspondiente. Después de cada llamada, los campos de la c-estructura que corresponden a la salida del programa deberán copiarse en las variables según su conexión que aparece en el recurso.
- Finalmente, las variables globales que deban ser modificadas por los programas (salidas de programas o variables definidas como externas en ellos o sus FB) se refrescan a partir de las copias locales. De nuevo, en el caso de variables directas pueden ser necesarios accesos al hardware. Este paso debe ser atómico.

En este esquema, la típica acción cíclica de los autómatas es realizada en cada tarea de forma independiente y las áreas globales se actualizan al final de cada ciclo. Otros esquemas más flexibles necesitarían algún mecanismo para permitir que el usuario decida cuando actualizar una variable o cómo sincronizar las tareas [10].

Los programas tipo tienen dos partes: datos y algoritmo. Las copias de los programas se declaran dentro de los recursos. El algoritmo se traduce a una c-función que se implementa como un módulo RTLinux. El módulo define la función y ejecuta un Pthread cuyo único objetivo es rellenar una entrada en la GFT con un puntero a la función. Un módulo es insertado por cada tipo de programa usado en una configuración. El uso de diferentes módulos para cada programa tipo, recurso y área global de configuración en lugar de un único módulo, no es obligatorio, pero permite realizar cambios 'en caliente' hasta un cierto punto. En nuestro modelo, un programa puede cambiar su algoritmo, pero no su interfaz. Una vez que el nuevo programa se ha compilado en un módulo, el gestor lo inserta. El nuevo módulo rellena su entrada en la GFT de forma que cualquier recurso que deba ejecutarlo leerá el nuevo puntero. Se ha establecido un mecanismo para que el gestor pueda eliminar el módulo antiguo en el momento adecuado.

Hemos citado un proceso denominado gestor. Está previsto que sea un programa Linux en el espacio de usuario habitual que estaría a cargo de insertar y eliminar los módulos, obtener información de los módulos de RTLinux para poder monitorizar variables, controlar la ejecución de los recursos etc, y de funciones de comunicación de alto nivel hacia el exterior. Sus requerimientos no serían de tiempo real, y su descripción detallada se realizará en un trabajo posterior.

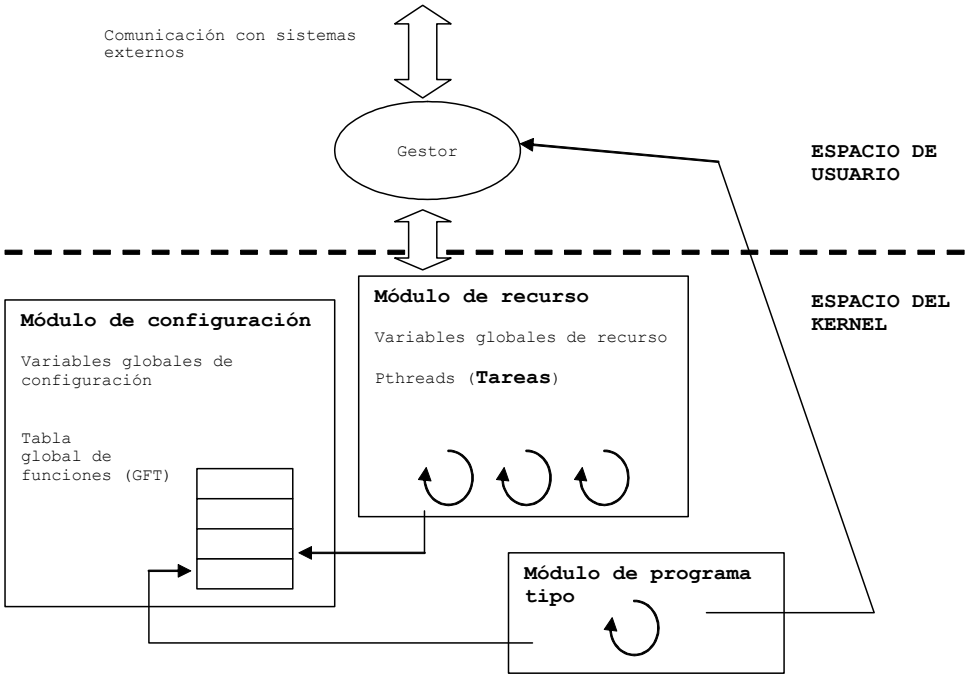

Figura 4: El modelo IEC61131-3 en RTLinux

4 PRIMEROS PASOS HACIA UN TRADUCTOR IEC61131-3

Para su aplicación práctica, las ideas citadas en el apartado anterior deben concretarse en un traductor que genere de forma automática el código en POSIX/C a partir de una descripción textual siguiendo el formato IEC.

El trabajo de los compiladores se suelen dividir en varias partes [1]: análisis léxico, análisis sintáctico, análisis semántico, generación de representación intermedia y generación del código final. En nuestro caso el código final no es un código máquina, sino un lenguaje de alto nivel, POSIX/C. Posteriormente este código en C será compilado utilizando la herramienta habitual en entornos Linux, gcc. Se prefiere utilizar como lenguaje final un lenguaje de alto nivel, puesto que esto facilita la optimización de código y la gestión de la memoria, que son efectuadas fundamentalmente por gcc.

Los primeros pasos que se han realizado corresponden a la etapa inicial o front end: análisis léxico, sintáctico y semántico con generación de código intermedio. Se han utilizado las herramientas automáticas flex y bison [3] para ayudar en el proceso. El programa generado es capaz de procesar y encontrar errores en:

- Declaraciones de tipos de usuario.
- Declaraciones de tipos de FBs y programas.

Para la parte algorítmica se ha elegido el lenguaje Lista de Instrucciones, por ser el más sencillo y por la experiencia previa [8]. No obstante, aunque la apariencia exterior de este lenguaje es la de un ensamblador, se ha tomado simplemente como una notación rudimentaria para la descripción de algoritmos.

- Declaraciones de configuraciones.

También se ha realizado la generación de código intermedio, que se ha representado como un árbol sintáctico abstracto. En una primera versión, se ha buscado la sencillez más que la optimización.

La descripción formal del estándar se encuentra en el anexo B de la norma [5]. Ha sido necesario realizar algunos cambios menores para evitar conflictos en la realización del traductor. Junto a ellos, el nivel de detalle alcanzado al realizar el traductor, permite percatarse de algunas mejoras a realizar, entre las que citamos:

1) Inclusión de declaración de variables globales en los programas, posibilidad citada en numerosas ocasiones en el estándar, pero no recogida en la descripción formal.

2) Ejemplos aclaratorios sobre la conexión de las entradas de un programa a variables de un recurso distinto a aquel en el que está declarado el propio programa. Esto se saltaría aparentemente las normas de alcance de las variables y no hay ninguna referencia en el texto a tal posibilidad, aunque las reglas sintácticas lo permiten.

5 CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se ha mostrado una propuesta para implementar el modelo de software IEC61131-3 utilizando POSIX/C. Hemos comentado los aspectos principales del estándar que no están completamente claros, como la implicación del uso de variables directas, y la concurrencia entre las tareas, indicando la realización prevista. El modelo propuesto ha sido traducido a POSIX/C. La traducción no siempre es directa, de forma que varias funciones y estructuras de datos necesitan ser definidos para implementar POUs, Configuraciones y Recursos. En el caso concreto de RTLinux, una división en módulos ayuda a estructurar la implementación y favorece el ocultamiento de datos, así como la posibilidad de hacer cambios 'en caliente'.

También se han realizado las primeras etapas de un traductor IEC a POSIX/C, incluyendo el análisis léxico, sintáctico y semántico, así como la generación de código intermedio. Para la parte algorítmica, utilizamos Lista de Instrucciones.

Como líneas futuras de trabajo, se pretende implementar el traductor completo y ampliar las posibilidades de lenguajes de programación. Paralelamente, se analizará la norma sugiriendo interpretaciones o modificaciones allí donde sea conveniente.

Referencias

[1] Bennet J.P., (1990) Introduction to compiling techniques: a first course using ANSI C, LEX and YACC. McGraw-Hill International (UK).

[2] Gallmeister, B.O., (1995) POSIX.4: Programming for the Real World, O'Reilly & Associates.

[3] Flex y bison, Manuales accesibles en http://www.gnu.org/manual/manual.html, última visita en Julio de 2004

[4] IEC 61131-8, (2003) Programmable Controllers: Part 8: Guidelines for the application and implementation of programming languages.

[5] IEC 61131-3, (2003) Programmable controllers. Part 3: Programming Languages.

[6] Lewis, R.W., (1998) Programming industrial control systems using IEC 1131-3, The Institution of Electrical Engineerings, London, United Kingdom.

[7] MatPLC project home page, http://mat.sourceforge.net/, última visita en Julio de 2004.

[8] Plaza, I., Medrano, C., (2003) "Organización de un autómata programable bajo RTLinux", *XXIV Jornadas de Automática*, León, 10-12 de septiembre de 2003.

[9] Ripoll, I., Tutorial de las API de RTLinux, accesible a través de la dirección web http://rtportal.upv.es/, el portal RTLinux de la Universidad Politécnica de Valencia, última visita en Julio de 2004.

[10] Sousa, M., (2001) "Linux-based PLC for industrial Control", *Embedded Linux Journal*, May (2001) (online at http://www.linuxdevices.com/articles/AT3681700515.html, última visita en Julio de 2004).

[11] Sousa, M., Baum, J., Romanenko, A., (2003), "MatPLC: Towards Real-Time Performance", *Fifth Real-Time Linux Workshop*, Valencia, Spain.