uModelFactory: software para modelado de sistemas embebidos

Abstract

En el marco del proyecto de investigación XXXX, perteneciente al Departamento de XXX, se propuso desarrollar una aplicación de software para PC con interfaz gráfica, multiplataforma y de código abierto que permita el modelado de una aplicación de control orientada a sistemas embebidos.UML propone la utilización de diagramas de estado (statechart) para describir el comportamiento de una determinada secuencia de estados atravesada por transiciones, eventos y acciones. Como resultado del proceso de modelado, la aplicación propuesta permitió generar el código en lenguaje C asociado manteniendo el sincronismo entre el diagrama, su representación en código y la documentación asociada.

1. Introducción

En la actualidad el uso de modelos es ampliamente utilizado para describir el comportamiento y ciclo de vida de una aplicación. Los modelos no están pensados para visualizar código sino para representar un sistema con un nivel de abstracción superior al de los lenguajes de programación.

Los modelos ayudan a comprender el sistema a diseñar y favorecen el intercambio de ideas entre los miembros del equipo de diseño y sus clientes.

Un modelo es una representación simplificada de un sistema que contempla las propiedades importantes del mismo desde un determinado punto de vista.

Los modelos se caracterizan por ser abstractos**,** comprensibles, precisos y predictivos y económicos [1]

La mayoría de los enfoques actuales en el desarrollo de software basado en modelos coinciden en [2,3]:

- Utilizar una *representación gráfica* del sistema a desarrollar.
- Describir el sistema con un cierto grado de *abstracción***.**
- Generar *código* ejecutable para el sistema embebido *partiendo del propio modelo***.**

Las máquinas de estados finitos constituyen una herramienta gráfica que ha sido utilizada tradicionalmente para modelar el comportamiento de sistemas electrónicos e informáticos. Una máquina de estados es un modelo computacional, basado en la teoría de autómatas, que se utiliza para describir sistemas cuyo comportamiento depende de los eventos actuales y de los eventos que ocurrieron en el pasado.

Los diagramas de estado muestran el conjunto de estados por los cuales pasa un objeto durante su ciclo de vida en una aplicación en respuesta a eventos junto con sus respuestas y acciones. Generalmente se encuentran compuestos por estados y transiciones. [4,5].

2. Análisis de la herramienta

A continuación se presenta el software uModel Factory v2015.

El desarrollo cuenta con una interfaz compuesta por:

- Módulo de modelado gráfico del algoritmo de control.
- Generador de código C para diferentes implementaciones.
- Módulo generador de documentación asociada al proyecto.

uModel Factory v2015 fue desarrollado en C++ utilizando el framework Qt en su versión 5.3. De esta forma ha sido posible crear una aplicación multiplataforma destinada al ámbito académico como a la industria.

La aplicación ha sido probada en forma exitosa en múltiples plataformas:

- Windows XP
- Windows 7
- Windows 8
- KUbuntu
- Debian 6

La versión actual permite:

- Crear, editar y guardar un proyecto.
- Alta, baja y modificación de estados
- Alta, baja y modificación de eventos
- Alta, baja y modificación de acciones.
- Describir en forma gráfica el modelo
- Generar implementaciones del modelo en lenguaje C utilizando punteros a función o estructura switch-case.
- Generar la documentación del proyecto
- Mantener el sincronismo entre el modelo código y documentación.

La interfaz principal permite diseñar en forma gráfica el diagrama de estados (figura 1).

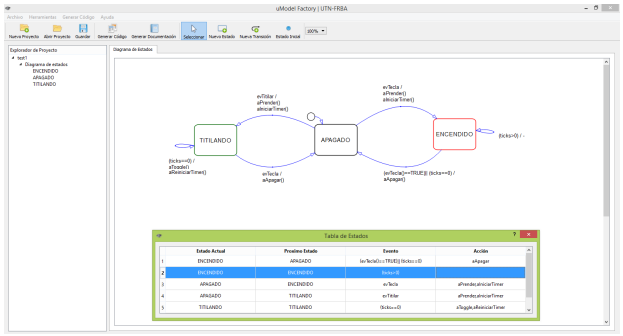

Figura 1

Dentro del entorno de desarrollo encontramos una barra de menú y herramientas (figura 2) que permiten realizar las acciones más habituales entre las cuales podemos destacar:

- Crear, abrir, cerrar y guardar proyecto
- Crear, editar y borrar estado
- Generar transiciones simples
- Generar código
- Generar documentación

Figura 2

En la barra lateral, se encuentra el árbol de proyecto el cual brinda información sobre los recursos del modelo (figura 3).

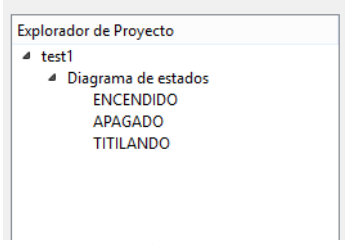

Figura 3

Al generar un nuevo proyecto, el usuario deberá cargar el nombre y el autor del proyecto como así también la ubicación donde se almacenará el modelo (figura 4).

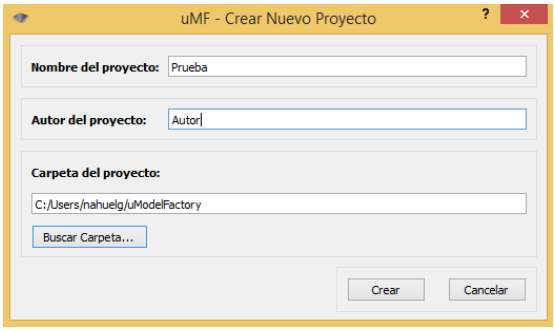

Figura 4

A partir de la definición de los estados pertenecientes al modelo, el software permite definir las transiciones entre estados a partir de la herramienta directa (figura 5).

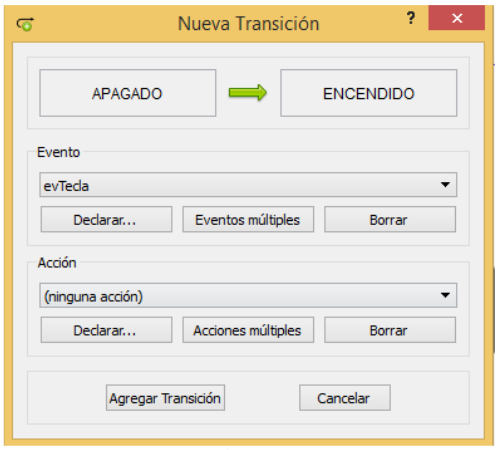

Figura 5

También es posible realizar la transición a partir de la edición de los atributos del estado (figura 6). Dentro de cada estado se definirá:

- El estado del que se parte.
- El estado al que se desea llegar.
- El evento que genera la transición.
- La acción asociada a dicho evento.

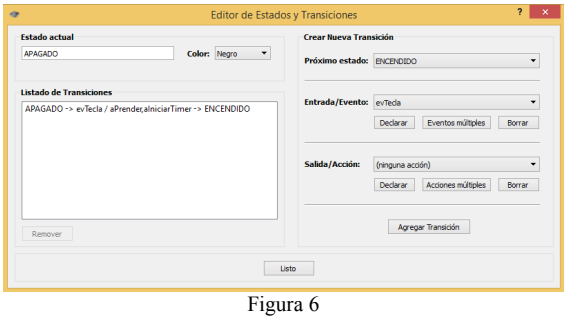

Figura 6

En la presente versión es posible definir eventos simples definidos por funciones booleanas como así también eventos de condición múltiple compuestos por variables, funciones booleanas y funciones de propósito general (figura 7).

Figura 7

En relación a las acciones, es posible trabajar con acciones simples o acciones múltiples (figura 8).

En el caso de modelos donde necesitemos asignar eventos y acciones previamente descriptas (reutilizar recursos) es posible generar un nuevo proyecto a partir de uno existente manteniendo todos los recursos generados por el usuario.

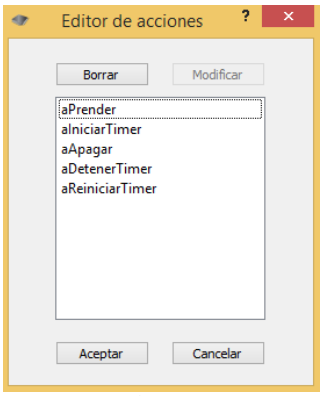

Figura 8

Una vez finalizado el diagrama podremos definir qué tipo de implementación deseamos obtener (figura 9):

- Mediante implementación con switch
- Mediante vector de punteros a función

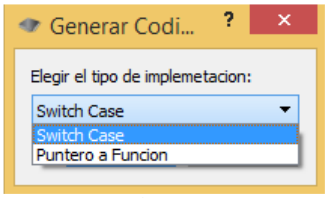

Figura 9

uModel Factory v2015 cuenta con un módulo generador de documentación dinámica del proyecto (figura 10). Dicho módulo genera un sitio web (local) el cual puede imprimirse y llevarse por ejemplo a formato PDF.
Dentro de la documentación generada se encontrará:

- Nombre del proyecto y autor
- Última fecha de modificación
- Diagrama de estados
- Listado de eventos
- Listado de acciones
- Tabla de estado asociada

UTN.BA FACULTAD REGIONAL BUENOS AIRES

uModel Factory - UTN FRBA

Software didactico-profesional para modelado, depuracion e implementacion
de aplicaciones industriales sobre sistemas embebidos

Proyecto: test1
Autor: utn
Modificacion: 22-08-2015 17:16:39

Diagrama de estados

Listado de eventos
void evTecla() ;
void evTitilar() ;

Listado de acciones
void aApagar() ;
void aIniciarTimer() ;
void aPrender() ;
void aPrender,aIniciarTimer() ;
void aReiniciarTimer() ;
void aToggle() ;
void aToggle,aReiniciarTimer() ;

Tabla de estados

Estado actual	Evento	Proximo estado	Accion
ENCENDIDO	(evTecla()==TRUE)|| !ticks==0\)	APAGADO	aApagar
ENCENDIDO	!ticks>0\)	ENCENDIDO	
APAGADO	evTecla	ENCENDIDO	aPrender,aIniciarTimer
APAGADO	evTitilar	TITILANDO	aPrender,aIniciarTimer
TITILANDO	!ticks==0\)	TITILANDO	aToggle,aReiniciarTimer
TITILANDO	evTecla	APAGADO	aApagar

Figura 10

3. Representación del modelo

En función del desarrollo y las mejoras introducidas en las diferentes versiones, se ha pensado la representación del modelo desde dos puntos de vista:

- Almacenamiento
- Representación gráfica
  - Diagramas de estado
  - Tabla de estados y transiciones

4.1 Almacenamiento

Hoy en día, la utilización de XML se ha convertido en un estándar para intercambiar información entre componentes y aplicaciones. XML es un meta-lenguaje utilizado para organizar y describir datos de forma que puedan ser interpretados por diferentes aplicaciones.

Con el objetivo de poder almacenar la información brindada por el modelo, se diseñó una estructura para representación de su información (document type definition). Para llevar adelante su manejo se diseñó la clase XMLhandler la cual permite realizar las operaciones básicas sobre el archivo XML (abrir, leer, escribir y guardar).

La estructura diseñada prevé una serie de tags (etiquetas) que posibilitan el almacenamiento de estados, eventos (funciones y variables), acciones y su vínculo mediante la transición (figura 11).

```
<?xml version='1.0' encoding='UTF-8'?>
<proyecto>
  <general>
    <nombre></nombre>
    <autores></autores>
    <creacion></creacion>
    <modificacion></modificacion>
    <carpeta></carpeta>
  </general>
  <maquina>
    <estado0>
      <diagrama>
        <nombre></nombre>
        <color></color>
        <pos_x></pos_x>
        <pos_y></pos_y>
        <ancho></ancho>
        <alto></alto>
      </diagrama>
      <transiciones>
        <tr0>
          <idEvento></idEvento>
          <idAcciones></idAcciones>
          <idProxEstado></idProxEstado>
          <midPos></midPos>
          <textPos></textPos>
        </tr0>

        ……
```

Figura 11

En esta estructura, se encuentra definido el estado inicial del sistema como así también la configuración del usuario (color y ubicación de los estados y transiciones).

Uno de los principales aspectos que se buscaron en el diseño de la estructura de almacenamiento, fue que la misma fuese escalable y adaptable a las futuras etapas del proyecto. El tipo de implementación elegida permite agregar nueva información en distintos niveles jerárquicos de manera muy simple.

4.2 Representación gráfica

Diagrama de estado

En el proyecto se planteó la clase *DiagramScene*, la cual es responsable de administrar los diferentes recursos al momento de describir nuestro modelo (estados y transiciones compuestas por eventos y acciones). Mediante dicha clase se logró representar gráficamente cada elemento del diagrama. Tanto los estados como las transiciones tienen una identidad definida, lo que permite que el usuario pueda interactuar con ellos de manera sencilla y editar sus propiedades y relaciones.

La clase *DiagramScene* funciona como un contenedor general de distintos objetos. Cada uno de ellos posee atributos propios vinculados con las características básicas de la representación del modelo, como lo son el tamaño del texto, su color y fuente. A su vez, posibilita el manejo de los eventos generados por el usuario mediante el uso del teclado y mouse (figura 12).

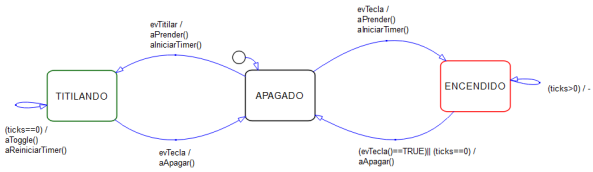

Figura 12

Tabla de estados y transiciones

La tabla de estados y transiciones permite representar la información vinculada al modelo a través de su representación organizada en cuatro columnas (figura 13)

- Estado actual
- Evento
- Próximo estado
- Acción

Figura 13

A medida que el usuario genera su modelo, la tabla se completa en forma dinámica, permitiendo evaluar rápidamente la lógica del modelo representado.

4. Generación de código

Tal como se ha explicado previamente, uModel Factory v2015 permite la generación de código para sistemas embebidos utilizando puntero a función o la estructura selectiva switch-case.

El diseño de un sistema embebido debe realizarse considerando la portabilidad, es decir, es recomendable trabajar con un modelo de capas que permita separar la lógica de la aplicación de los recursos de hardware asociados. Las capas de este modelo son:

Aplicación: contempla el diseño lógico

Primitivas: son las funciones que interactúan con la aplicación.

Buffers: se utilizan para independizar los tiempos entre las primitivas y el hardware.

Drivers: son funciones que interactúan en forma directa con el hardware:

Hardware: recursos propios del sistema.

uModel Factory v2015 permite trabajar generar código en la capa de aplicación y primitivas, de forma que el desarrollador complete las capas restantes de acuerdo a los recursos propios del sistema.

La estructura base de cualquier sistema embebido consta de una rutina de inicialización de periféricos y un ciclo continuo que sensa las entradas, las evalúa y acciona sobre las salidas (figura 14);

```
int main (void)
{
        inicializacion() ;

        while(1)
        {
            maquina_estados() ;
        }

       return 0 ;
}
```

Figura 14

Implementación mediante switch-case

```
void maquina_estado()
{
   static int estado = APAGADO;

   switch(estado)
   {
     case ENCENDIDO:
        // evaluación de evento
        // transición hacia otro estado
        // o generación de auto-transición
        break;

      case APAGADO:
        // evaluación de evento
        // transición hacia otro estado
        // o generación de auto-transición
        break;

       default: estado = APAGADO;

    }
}
```

Figura 15

La estructura anterior (figura 15) permite evaluar cada uno de los estados, dentro de cada estado se evalúa si el evento es verdadero o falso y se producirá una transición hacia el estado siguiente o una auto-transición. El último caso, permite contemplar modificaciones involuntarias sobre la variable *estado* y de esta forma garantizar la continuidad del sistema.

Implementación mediante puntero a función

Esta implementación requiere de dos partes: la declaración e inicialización del array y la máquina de estados propiamente dicha (figura 16).

```
int estado = APAGADO ;

(void*) arrayFunciones[] () = {
funcion_ENCENDIDO, funcion_APAGADO,
funcion_TITILANDO } ;
```

Figura 16

La utilización de puntero a función genera un código compacto que contempla dos situaciones: la evaluación de estado de emergencia y la invocación de la rutina correspondiente utilizando como parámetro la variable *estado* (figura 17).

```
void maquina_estado(void)
{

       if(estado > TITILANDO)
       {
              estado = APAGADO;
              return;
       }

       (*arrayFunciones[estado])();
}
```

Figura 17

El código generado resulta compatible con *Doxygen* que es una herramienta que permite generar documentación de código en diferentes formatos (Latex, pdf, sitio web).

5. Utilización de la herramienta

La herramienta presentada en este trabajo ha sido utilizada principalmente en forma didáctica para el dictado de seminarios y talleres orientados a estudiantes, docentes y profesionales.

Durante el presente año se ha utilizado:

- En el seminario para docentes – Informática II perteneciente al departamento de Electrónica.
- Como herramienta didáctica para el dictado de programación gobernada por eventos en la asignatura previamente mencionada.
- En el Workshop "Diseñando sistemas embebidos con uModel Factory" en el Simposio Argentino de Sistemas Embebidos.

6. Líneas de trabajo a futuro

Dentro del desarrollo de la herramienta presentada, se prevé la incorporación de un módulo de simulación a fin de poder evaluar el modelo planteado sin necesidad de contar con un hardware asociado. La simulación posibilita reducir los tiempos de desarrollo como así también los costos vinculados a un proyecto.

7. Referencias

[1] N. Gonzalez, J. Cruz, L. Sugezky, M. Giura, M. Trujillo, M. Prieto. *Analysis of a UML-based embedded system modeling software application.* Congreso Argentino de Sistemas Embebidos. 2014.

[2] G. Booch, J. Rumbaugh, I. Jacobson. "El Lenguaje Unificado de Modelado". Addison-Wesley 2nd Edition, 2006.

[3] G. Booch, J. Rumbaugh, I. Jacobson. "El Proceso Unificado de Desarrollo de Software". Addison-Wesley 1st Edition, 2000.

[4] C. Larman. "UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado". Prentice-Hall, 2003.

[5] Miro Samek. "Practical UML Statecharts in C/C++". Newnes 2nd Edition, 2009.