Modelado y Simulación con el Formalismo DEVS

Ernesto Kofman y Federico Bergero

Laboratorio de Sistemas Dinámicos y Procesamiento de la Información
FCEIA - Universidad Nacional de Rosario.
CIFASIS – CONICET. Argentina

Organización de la Presentación

1. Modelado y Simulación con el Formalismo DEVS
   - Conceptos Básicos de Modelado y Simulación
   - El Formalismo DEVS
   - Simulación de Modelos DEVS
2. Contribuciones de nuestro grupo
   - Simulación de Sistemas Continuos
   - Control Mediante DEVS
   - DEVS Estocásticos
3. PowerDEVS - Implementación del simulador DEVS
   - Modelo atómico
   - Modelo acoplado
   - Coordinador de la simulación
   - Simulación en Tiempo Real

Outline

Sistemas Dinámicos

Definición de Sistema

Un Sistema es una disposición delimitada de entidades interactuantes.

- Disposición: define la Estructura del Sistema.
- Delimitación: las acciones del resto del universo sobre el sistema se reemplazan por entradas.
- Entidades interactuante: son los componentes del sistema: procesos, elementos, subsistemas, etc.

Definición de Sistema Dinámico

Un Sistema Dinámico es un Sistema en el cual hay almacenamiento de energía, materia o información; es decir, un Sistema Dinámico es un sistema con Memoria.

Modelos Matemáticos

Por cuestiones de costo, riesgo o imposibilidad (si el sistema todavía no existe por ejemplo), en muchas ocasiones no se puede experimentar sobre los sistemas reales. En estos casos se recurre a la experimentación sobre Modelos del sistema.

Definición de Modelo

Un Modelo es una representación simplificada de un Sistema que permite responder interrogantes sobre este último sin recurrir a la experimentación sobre dicho sistema.

Definición de Modelo Matemático

Es un conjunto de expresiones matemáticas que describen las relaciones existentes entre las magnitudes caracterizantes del sistema.

Modelado y Simulación

Modelado

Es el proceso de obtención de modelos matemáticos.

Para esto, hay dos técnicas:

- A partir de principios analíticos y/o físicos.
- Mediante experimentación (identificación)

Simulación

Es la experimentación sobre un modelo matemático de un sistema, generalmente implementado en una computadora.

La simulación de un sistema, además del modelado, suele requerir la utilización de técnicas de aproximación (métodos numéricos de integración, por ejemplo).

Clasificación de Modelos Matemáticos

- Tiempo Continuo: Las variables evolucionan continuamente en el tiempo. Generalmente se representan mediante ecuaciones diferenciales. Ej:

$$\ddot{\theta}(t) + \frac{g}{l}\sin(\theta(t)) = 0 \quad \text{(péndulo sin fricción)}$$

- Tiempo Discreto: Las variables sólo pueden cambiar en determinados instantes de tiempo. Se suelen representar mediante ecuaciones en diferencias. Ej:

$$\begin{aligned} N(k+1) &= \lambda \cdot N(k) \cdot e^{-a \cdot P(k)} \\ P(k+1) &= N(k) \cdot (1 - e^{-a \cdot P(k)}) \end{aligned} \quad (Nicholson - Bailey)$$

- Eventos Discretos: Las variables pueden cambiar en cualquier momento, pero sólo puede haber números finitos de cambios en intervalos de tiempo finitos.

Representación de Modelos de Eventos Discretos

Existen varios formalismos de representación de sistemas por eventos discretos:

- Redes de Petri
- State Graphs
- Grafcet
- DEVS (Discrete EVent system Specification)

DEVS es el formalismo más general, ya que cualquier modelo de eventos discretos puede representarse mediante DEVS.

Formalismo DEVS

El formalismo DEVS, formulado por Bernard Zeigler a mediados de los 70, permite representar cualquier sistema que tenga un número finito de cambios en un intervalo finito de tiempo.

Un modelo DEVS procesa una secuencia de eventos de entrada y de acuerdo a su condición inicial produce una secuencia de eventos de salida.

DEVS – Modelo Atómico

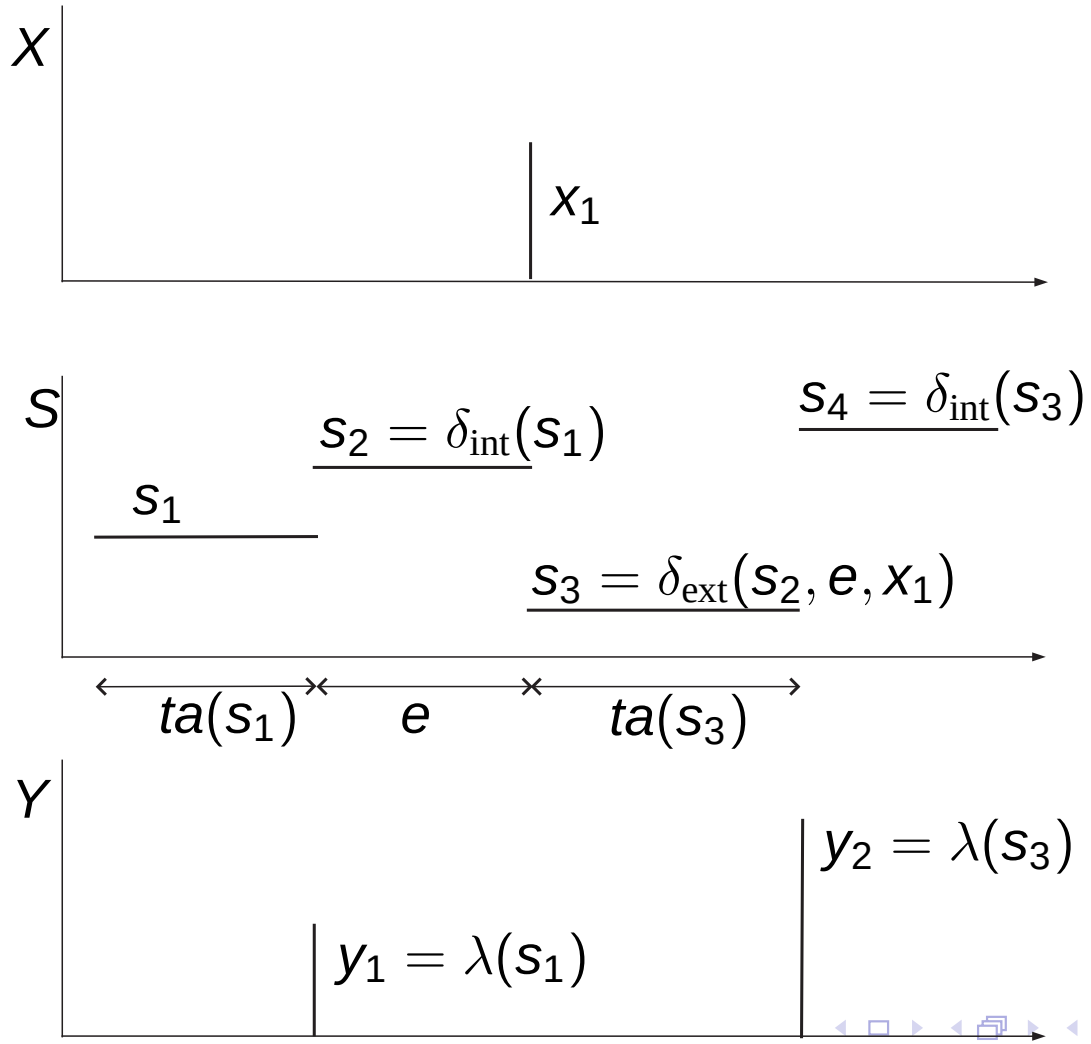
Un modelo atómico DEVS se define por la estructura:

$$M = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta)$$

donde:

- X es el conjunto de valores de entrada.
- Y es el conjunto de valores de salida.
- S es el conjunto de valores de estado.
- δ_{int}, δ_{ext}, λ y ta son las funciones que definen la dinámica del sistema.

DEVS – Trayectorias



DEVS – Ejemplo 1

Un procesador recibe trabajos identificados por un número real positivo que indica cuanto tiempo demora en procesarse dicho trabajo. Transcurrido el tiempo de procesamiento, el procesador emite un evento con valor 0.

$$P_1 = < X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta >$$

$$X = Y = S = \mathbb{R}^+$$

$$\delta_{\text{ext}}(s, e, x) = x$$

$$\delta_{\text{int}}(s) = \infty$$

$$\lambda(s) = 0$$

$$ta(s) = s$$

Este modelo se olvida del trabajo que estaba procesando al recibir uno nuevo

DEVS – Ejemplo 1

Para ignorar los eventos de entrada mientras se está procesando un trabajo, podemos modificar el modelo anterior como sigue:

$$P_2 =< X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta >$$

$$X = Y = \mathbb{R}^+$$

$$S = \mathbb{R}^+ \times \{true, false\}$$

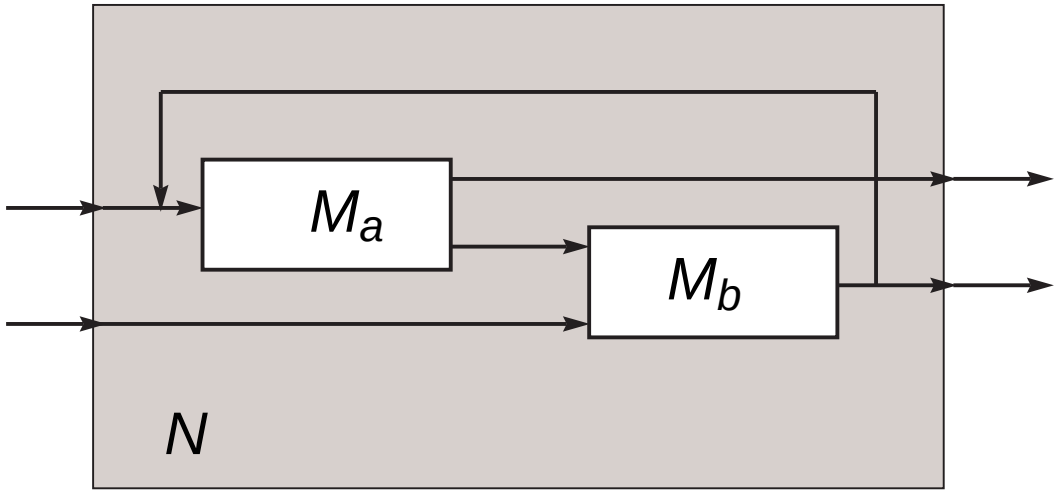
$$\delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}((\sigma, busy), e, x) = \begin{cases} (\sigma - e, true) & \text{si } busy = true \\ (x, true) & \text{en otro caso} \end{cases}$$

$$\delta_{\text{int}}(s) = \infty$$

$$\lambda(s) = 0$$

$$ta((\sigma, busy)) = \sigma$$

DEVS – Modelos Acoplados



En el acoplamiento modular los eventos de salida de un modelo DEVS se convierten en eventos de entrada de otro.

Clausura bajo acoplamiento

El acoplamiento de modelos atómicos DEVS define un modelo atómico equivalente.

Simulación de Modelos DEVS

Un simulador genérico de modelos DEVS funciona como sigue:

1. Para cada atómico $d \in N$, calculamos $tn_d = ta_d(s_d) - e_d$ (tiempo del próximo evento interno en el atómico d).
2. Buscamos el modelo atómico d^* que tiene el mínimo tn_d (d^* es el próximo atómico en realizar una transición interna).
3. Avanzamos el tiempo de la simulación t haciendo $t = tn_{d^*}$.
4. Calculamos el evento de salida $\lambda_{d^*}(s_{d^*})$.
5. Para cada modelo atómico d conectado a la salida del modelo d^*, ejecutamos la función de transición externa y recalculamos tn_d.
6. Ejecutamos la transición interna de d^* y recalculamos tn_{d^*}.
7. Volvemos al punto 2.

Software de Simulación de Modelos DEVS

Actualmente, existen varias herramientas de software basadas en DEVS, desarrollada por los distintos grupos de investigación que trabajan en el tema:

- ADEVS, DEVS/C++ y DEVSJAVA (University of Arizona).
- DEVSSim++ (KAIST, Corea).
- CD++ (Carleton University y UBA).
- LSIS-DME (LSIS, Marsella, Francia)
- VLE (Université du Littoral Côte d'Opale, Francia).
- SmallDEVS (Brno University of Technology, República Checa).
- JDEVS (Université de Corse).
- PowerDEVS (Universidad Nacional de Rosario).

Outline

Trabajo sobre DEVS en la EIE y CIFASIS

Desde el Departamento de Electrónica de la FCEIA y desde CIFASIS–CONCIET, nuestro grupo está trabajando en los siguientes temas relacionados a DEVS:

- Métodos de simulación de sistemas continuos mediante DEVS.
- Control de plantas continuas por DEVS.
- Modelos DEVS Estocásticos, y aplicación al modelado y simulación de redes de datos.
- Desarrollo de la herramienta PowerDEVS.
- Simulación y procesamiento de señales en tiempo real mediante DEVS.

Simulación de Sistemas Continuos

Una de las formas de representar sistemas continuos es a través de modelos de Ecuaciones Diferenciales Ordinarias:

$$\dot{x}(t) = f(x(t), t)$$

Para simular este sistema, hay que resolver la ecuación y obtener $x(t)$. Para esto, los métodos tradicionales de integración numérica aproximan este sistema mediante fórmulas del tipo:

$$x(t_{k+1}) = F(x(t_k), t_k)$$

es decir, una Euación en Diferencias (como si fuera un Sistema de Tiempo Discreto).

Integración por Cuantificación

Otro camino para simular el modelo

$$\dot{x}(t) = f(x(t), t) \tag{1}$$

es modificarlo como sigue

$$\dot{x}(t) = f(q(t), t) \tag{2}$$

donde cada componente $q_i(t)$ es una versión cuantificada de $x_i(t)$ (por ejemplo, podría tomarse $q_i(t) = \text{int}(x_i(t))$).

La Ec.(2) se denomina Sistemas de Estados Cuantificados (QSS) y puede demostrarse que es equivalente a un modelo DEVS, y por lo tanto, puede simularse exactamente.

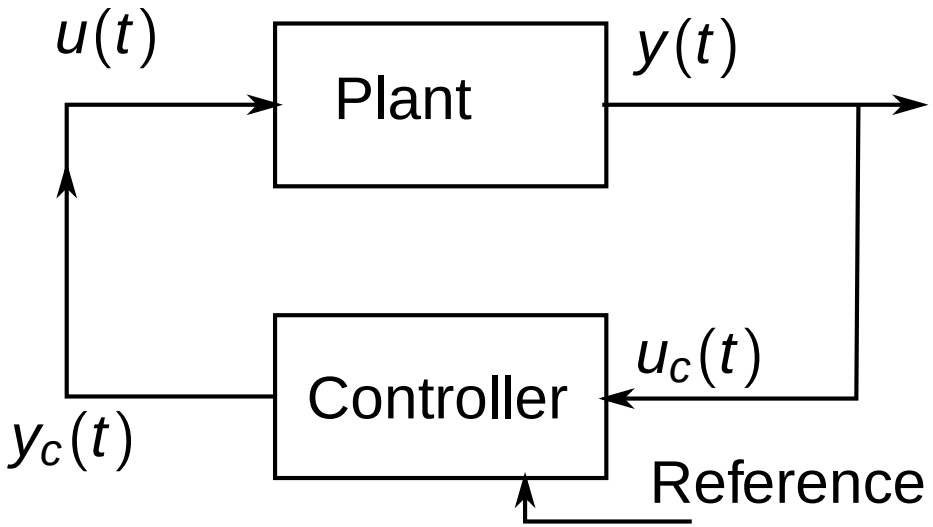
Métodos de QSS

Los métodos de integración por cuantificación (métodos de QSS) fueron desarrollados y estudiados por nuestro grupo. Actualmente existen los siguientes métodos:

- Método de QSS1 (2000)
- Método de QSS2 (2002)
- Método de QSS3 (2005)
- Método de Backward QSS (BQSS), para sistemas stiff (2006).
- Método de Centered QSS (CQSS), para sistemas marginalmente estables (2008).

Control por Retroalimentación

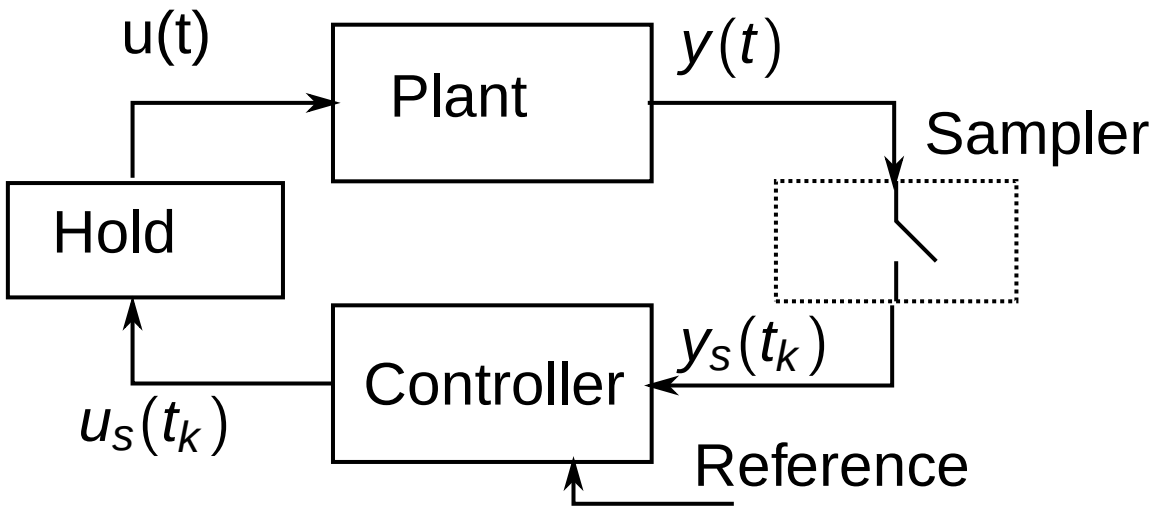
Una forma de lograr que un sistema se comporte como uno quiere, es conectándolo a un controlador que mida ciertas variables (salidas) y en base a esto, modifique otras variables (entradas), siguiendo alguna ecuación diferencial.



El problema es que es muy dificil construir un sistema físico que cumpla con las ecuaciones diferenciales del controlador.

Control Muestreado

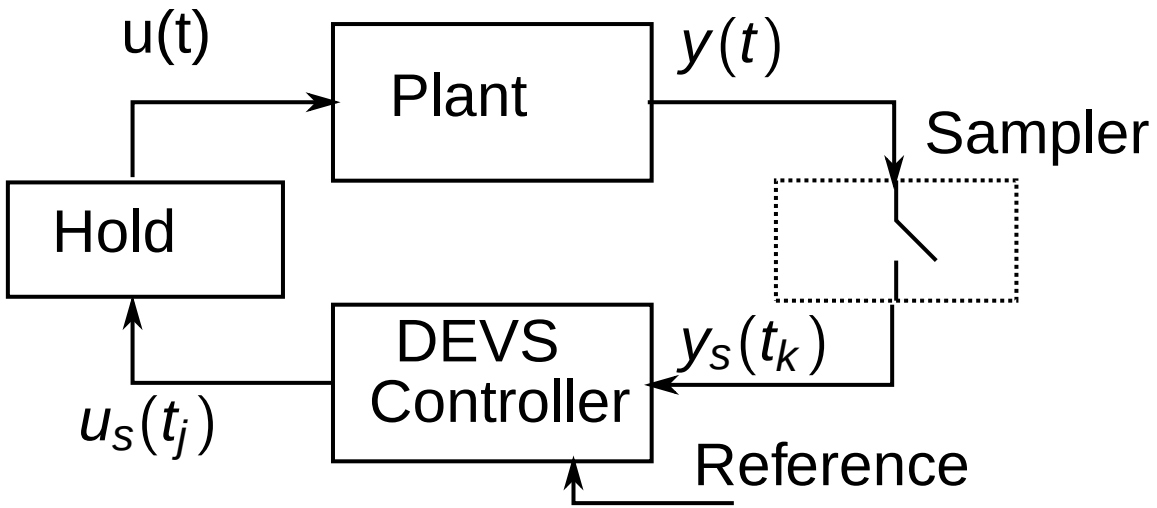
El controlador ideal entonces se suele reemplazar por un dispositivo digital que recibe las mediciones muestreadas periódicamente y envía de la misma forma los valores de las entradas.



El nuevo controlador satisface una ecuación en diferencias que aproxima la dinámica de la ecuación diferencial del controlador continuo ideal.

Control por Estados Cuantificados

Una alternativa, desarrollada por nuestro grupo, es reemplazar el controlador continuo por su aproximación QSS, es decir, por un modelo DEVS.



En este esquema, denominado Quantized State Control (QSC), el muestreo se realiza además por cruces de nivel (de manera asíncrona).

DEVS Estocásticos

Una limitación del formalismo DEVS es que está planteado sólo para modelos determinísticos. Por esto, desarrollamos una extensión del formalismo que permite modelar dinámicas estocásticas, que denominamos STDEVS. Un modelo atómico STDEVS tiene la forma:

$$M_{ST} =< X, Y, S, \mathcal{G}_{int}, \mathcal{G}_{ext}, P_{int}, P_{ext}, \lambda, ta >$$

donde X, Y, S, λ y ta tienen la misma definición que en DEVS, y ahora

- Para cada $s \in S$, $\mathcal{G}_{int}(s)$ es una colección de subconjuntos de S.
- Para cada $G \in \mathcal{G}_{int}(s)$ ($G \subseteq S$), la función $P_{int}(s, G)$ da la probabilidad de transicionar desde s hasta G.
- ... de manera similar se definen $\mathcal{G}_{ext}$ y P_{ext}.

DEVS Estocásticos

Además de definir el nuevo formalismo STDEVS,

- se demostró la clausura bajo acomplamiento de STDEVS.
- se probó que DEVS es un caso particular de STDEVS.
- se demostró que un modelo DEVS en el cual se añaden funciones tipo random en las transiciones define un modelo STDEVS.
- se está aplicando la metodología para modelado y simulación de redes de datos.

Outline

PowerDEVS

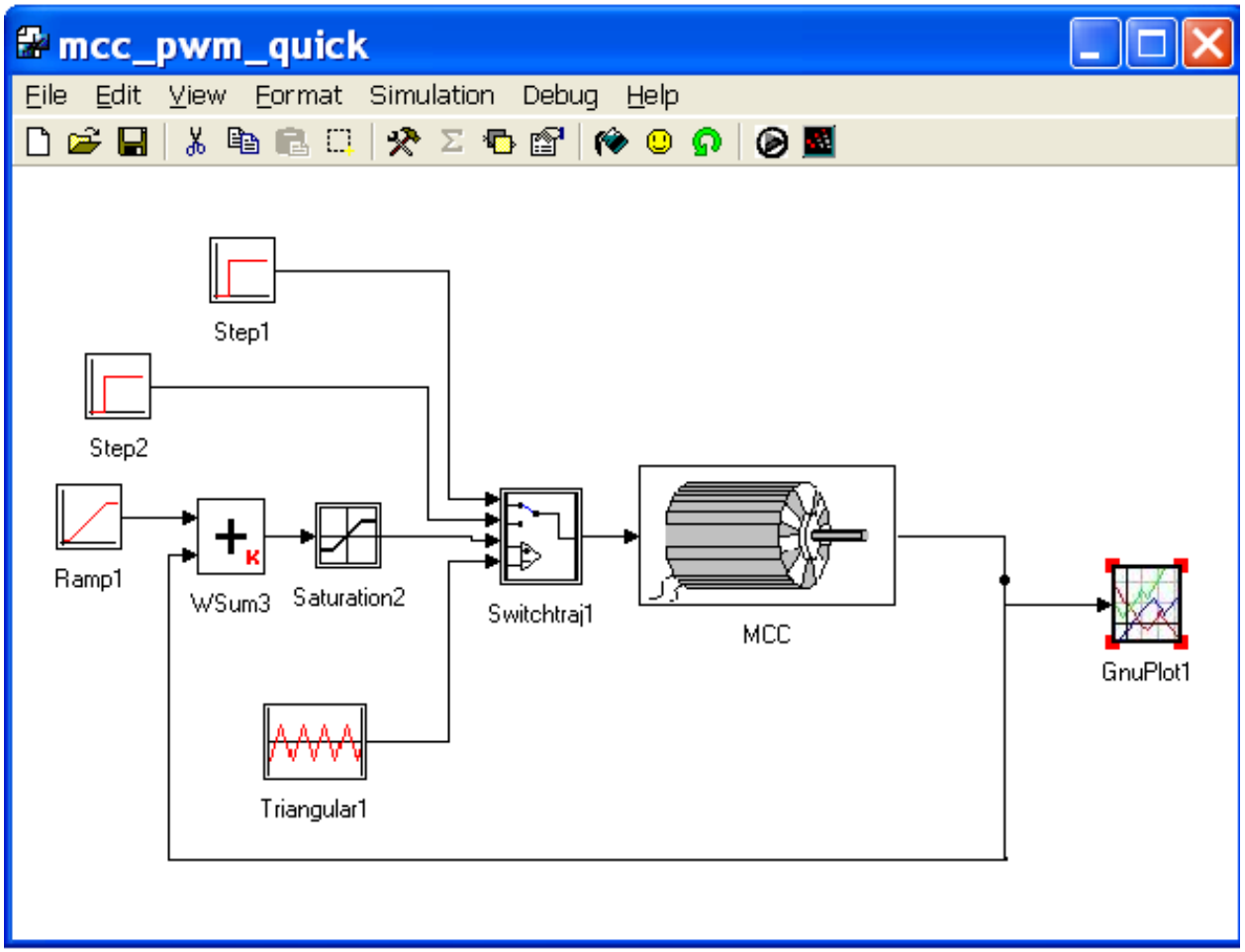
PowerDEVS es un entorno de simulación de modelos DEVS de propósito general, orientado a la simulación de sistemas híbridos, realizado y mantenido por nuestro grupo de investigación en la FCEIA.

Cuenta con dos partes principales:

Entorno gráfico Es la interfaz que ve el usuario regular de PowerDEVS. Permite describir modelos DEVS en forma gráfica y sencilla.

Motor de simulación Este módulo es el núcleo de toda la herramienta. El código generado por la interfaz, luego es compilado junto con el motor de simulación obteniéndose el programa que simula el modelo DEVS.

PowerDEVS – Ventana de Modelo



Simulación de un modelo DEVS

Implementación del simulador DEVS

Un modelo DEVS puede ser simulado en una computadora. Veremos una implementación abstracta y luego una implementación en C++.

Simulando un modelo atómico

DEVS-simulator

variables :

tl // time of last event

tn// time of next event

$e = 0$// elapsed time in the actual state

s// actual state of the atomic

$y = (y.value, y.port)$ // current output of the atomic

when recieves $i - message(i, t)$ *at time* t

$tl = t - e$

$tn = tl + ta(s)$

Simulando un modelo atómico - Cont

when recieve $* - message(*, t)$ *at time* t
  $y = \lambda(s)$
  send $y - message(y, t)$ *to parent coordinator*
  $s = \delta_{int}(s)$
  $tl = t$
  $tn = t + ta(s)$
when recieve $x - message(x, t)$ *at time* t
  $e = t - tl$
  $s = \lambda_{ext}(s, e, x)$
  $tl = t$
  $tn = t + ta(s)$
end DEVS-simulator

Implementación de un atómico en C

```
class simulator
{
    public:
        Coupling * father;
        int myself;
        double e;
        double tl;
        double tn;
        Event output
        void imessage(Coupling*, int, double);
        Event lambdamessage(double);
        void dintmessage (double);
        void dextmessage(Event, double);

        virtual void dint(double)=0;
        virtual Event lambda(double)=0;
        virtual void dext(Event, double)=0;
        virtual void init(double, ...)=0;
        virtual double ta(double)=0;
}
```

Implementación de un atómico en C - Cont

```
void simulator::imessage(Coupling* f, int my, double t){
    myself=my;
    father=f;
    e=0;
    tl=t;
    tn=t+ta(t);
}

Event simulator::lambdamessage(double t){
    Event out=lambda(t);
    return out;
}
```

Implementación de un atómico en C - Cont

```
void simulator::dintmessage(double t){
    e=t-tl;
    dint(t);
    tl=t;
    tn=t+ta(t);
}

void simulator::dextmessage(Event x, double t){
    e=t-tl;
    dext(x, t);
    tl=t;
    tn=t+ta(t);
}
```

Ejemplo de un modelo atómico

```
class sinegen:public simulator{
    double sigma;
    double y[10];
    double a,phi,k,dt,w,f;
    public:
        void init(double, ...);
        double ta(double t);
        void dint(double);
        void dext(Event , double );
        Event lambda(double);
}
```

Ejemplo de un modelo atómico - Cont

```
void sinegen::init(double t, ...){
    a= va_arg(parameters,double);
    f = va_arg(parameters,double);
    phi= va_arg(parameters,double);
    k= va_arg(parameters,double);
    dt=1/f/k;
    w=3.14159*2*f;
    sigma=0;
    for(int i=0;i<10;i++){y[i]=0;};
}
double sinegen::ta(double t){   return sigma; }

void sinegen::dint(double t){ sigma=dt; }

Event sinegen::lambda(double t){
    y[0]=a*sin(w*t+phi);
    return Event(&y[0],0);
}
```

Simulando un modelo acoplado

DEVS-coordinator

variables:

tl // time of last event

tn// time of next event

$y = (y.value, y.port)$ // current output of the atomic

D // list of children

IC// list of connections of the form $[(d_i, port_1), (d_j, port_2)]$

EIC// list of connections of the form $[(N, port_1), (d_j, port_2)]$

EOC// list of connections of the form $[(d_i, port_1), (N, port_2)]$

when recieve $i - message(i, t)$ *at time* t

send $i - message(i, t)$ *to all children* $\in D$

Simulando un modelo acoplado - Cont

when recieve $* - message(*, t)$ *at time* t
  send $* - message(*, t)$ *to* d^*
  $d^* = arg[min_{d \in D}(d.tn)]$
  $tl = t$
  $tn = t + d^*.tn$
when recieve $x - message((x.value, x.port), t)$ *as time*
  $(v, p) = (x.value, x.value)$
  for each connection $[(N, p), (d, q)]$
  send $x - message((v, q), t)$ *to child* d
  $d^* = arg[min_{d \in D}(d.tn)]$
  $tl = tn$
  $tn = t + d^*.tn$

Simulando un modelo acoplado - Cont

when recieve $y - message((y.value, y.port), t)$ *from* d^*
  if a connection $[(d^*, y.port), (N, q)]$ *exists*
    send $y - message((y.value, q), t)$ *to parent coordinator*
  foreach connection $[(d^*, p), (d, q)]$
  send $x - message((y.value, q), t)$ *to child* d
end $DEVS - coordinator$

Implementación de un acoplado en C

```
class Coupling:public simulator
{
    public:
                simulator ** D;
                connection** IC;
                connection** EIC;
                connection** EOC;
                int dast;
                void init(double t, ...);
                void exit();
                void dint(double);
                void propagate(Event, double);
                Event lambda(double);
                void dext(Event, double);
}
```

Implementación de un acoplado en C - Cont

```
void Coupling::init(double t, ...){
                for(int i=0;i<Dsize;i++){
                                D[i]->imessage(this, i, t);
                };
};

void Coupling::exit(){
                for(int i=0;i<Dsize;i++){
                                D[i]->exit();
                };
};

void Coupling::dint(double t){
                D[dast]->dintmessage(t);
};
```

Implementación de un acoplado en C - Cont

```
void Coupling::dext(Event x, double t){
                int port=x.port;
                for(int i=0;i<EICsize;i++)
                                if (EIC[i]->port1==port){
                                                x.port=(EIC[i])->port2
                                                int mod=EIC[i]->child2
                                                D[mod]->dextmessage(x,
                                };
};
double Coupling::ta(double t){
                tn=4e10;
                for(int i=0;i<Dsize;i++)
                                if (D[i]->tn<tn){
                                                tn=D[i]->tn;
                                                dast=i;
                                };
                double tal=tn-t;
                return tal;
};
```

Implementación de un acoplado en C - Cont

```
Event Coupling::lambda(double t){
        output=D[dast]->lambdamessage(t);
        int p=output.port;
        if (output.IsNotNull()){
                propagate(output,t);
        };
        Event x;
        x.SetNullEvent();
        return(x);
}
```

Implementación de un acoplado en C - Cont

```
void Coupling::propagate(Event x, double t){
    output=x;
    int p=output.port;
    if(x.IsNotNull()){
        for(int i=0;i<ICsize;i++){
                if ((IC[i])->child1==dast) && (IC[i]->port1==p)){
                        output.port=IC[i]->port2;
                        int mod=IC[i]->child2;
                        D[mod]->dextmessage(output, t);
                };
        };
        for(int k=0;k<EOCsize;k++){
                if ((EOC[k]->child1==dast) && (EOC[k]->port1==p)){
                        output.port=EOC[k]->port2;
                        father->propagate(output, t);
                };
         };
      };
}
```

Coordinando la simulación

DEVS-root-coordinator
  t // global simulation time
  d // child coordinator
  $t = t_0$
  $send\ i - message(i, t)\ to\ d$
  $t = d.tn$ // Se adelanta el tiempo!
  $do\ while\ (t < tf)$
    $send\ *-message(*, t)\ to\ d$
    $t = d.tn$
end DEVS-root-coordinator

Implementación del coordinador en C

```
void root_simulator::init(){
    t=ti;
    MainCoupling->imessage(0,0,t); // Este es el acoplado hijo d
    MainCoupling->ta(t);
}

bool root_simulator::step(){
    t=MainCoupling->tn; // Se adelanta el tiempo!
    if (t<=tf) {
        MainCoupling->lambdamessage(t);
        MainCoupling->dintmessage(t);
        return false;
    }
    else {
        MainCoupling->exit();
        return true;
    };
}
void root_simulator::run() {    while(!step()); }
```

Simulación en Tiempo Real

¿Porqué simular en Tiempo Real?

Muchas veces la simulación debe inter-actuar con eventos del mundo real, (medir un sensor, o ejecutar una acción sobre algún componente, etc). En este caso la simulación debe ejecutarse restricta a cotas temporales impuestas por estos eventos externos.

Esto es lo que se denomina un sistema de tiempo real, que en el sentido más general, es aquel en el cual el resultado obtenido no sólo debe ser correcto sino que debe ser también “entregado” en el momento correcto y debe responder a estímulos cumpliendo también restricciones temporales.

Sincronización de la simulación

Tiempo Real

En la simulación de un modelo DEVS antes planteada, el tiempo real no tiene ninguna relación con el tiempo de simulación.

Cuando simulamos un modelo pueden ocurrir dos cosas:

- Que la simulación consuma más tiempo (real) que el tiempo simulado (simular 10 s en 40 s).
- Que la simulación consuma menos tiempo (real) que el tiempo simulado (simular 10 s en 1 s).

En cualquiera de los casos el modelos esta desfasado con el tiempo real.

Sincronización de la simulación - Cont

La idea entonces es, sincronizar los eventos de la simulación con el tiempo real. Esto puede realizarse esperando que se alcance el tiempo real en el cual tiene que ser emitido un evento.

Sincronización

Si un evento e debe emitirse en un tiempo de simulación $t_i > T$ (donde T es el Tiempo Real) debemos esperar $t_i - T$ s y emitir el evento e. De esta forma los eventos serán emitidos sincronizado con el tiempo real. En el caso que $T > t_i$ la sincronización no puede realizarse, este suceso se denomina overrun.

Simulando modelos DEVS en Tiempo Real

DEVS-simulator

when recieve $*-message(*,t)$ *at time* t

$y = \lambda(s)$

if y *is a realtime event*

wait for real time t

send $y - message(y,t)$ *to parent coordinator*

$s = \delta_{int}(s)$

$tl = t$

$tn = t + ta(s)$

Implementación de un atómico en C con RT

```
Event simulator::lambdamessage(double t){
    Event out=lambda(t);
    if (out.isRealTimeEvent() ) {
        waitFor(t,out.realtimemode);
    }
    return out;
}
```

La precisión de la sincronización depende cómo esté implementada la rutina `waitFor(double t)`.

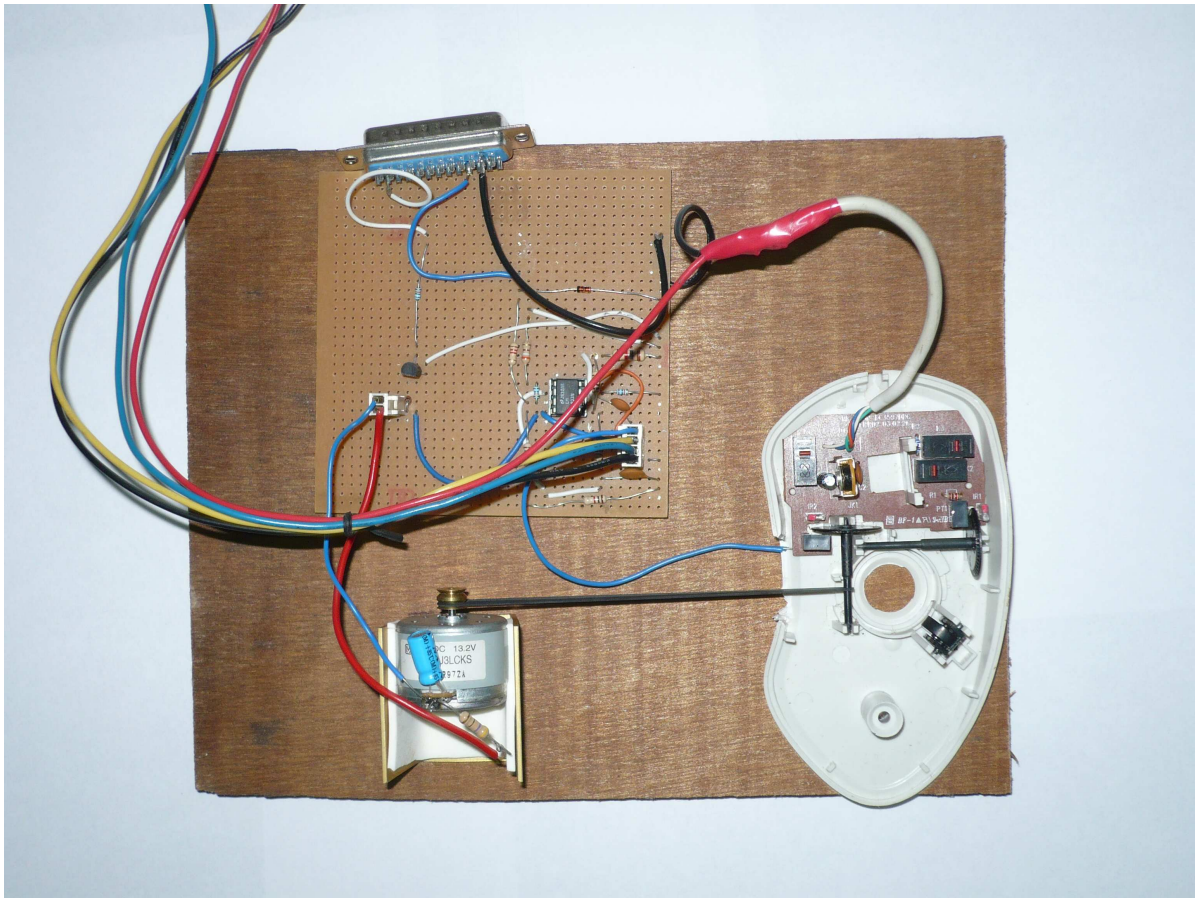
PowerDEVS – Distribución RTAI-Linux

RTAI Linux

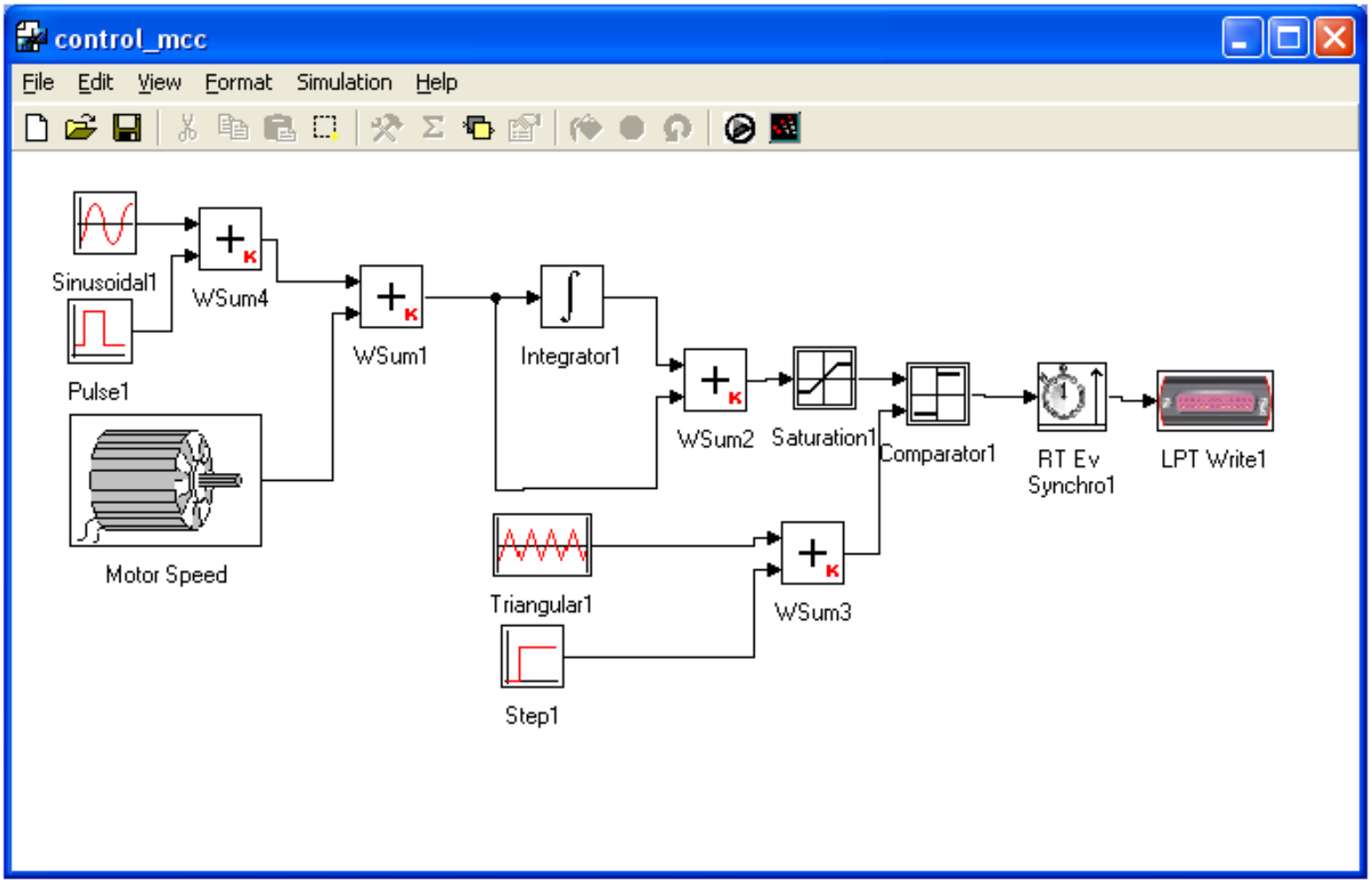
Implementamos el simulador en Tiempo Real sobre un Kernel de Linux modificado para correr en Tiempo Real (RTAI).

- Utilizando la herramienta desarrollada obtenemos latencias de simulación del orden de los 2000 ηs a 5000 ηs.
- En cuanto a la latencia de interrupción (respuesta a estímulos externos) obtenemos una latencia cercana a 15000 ηs

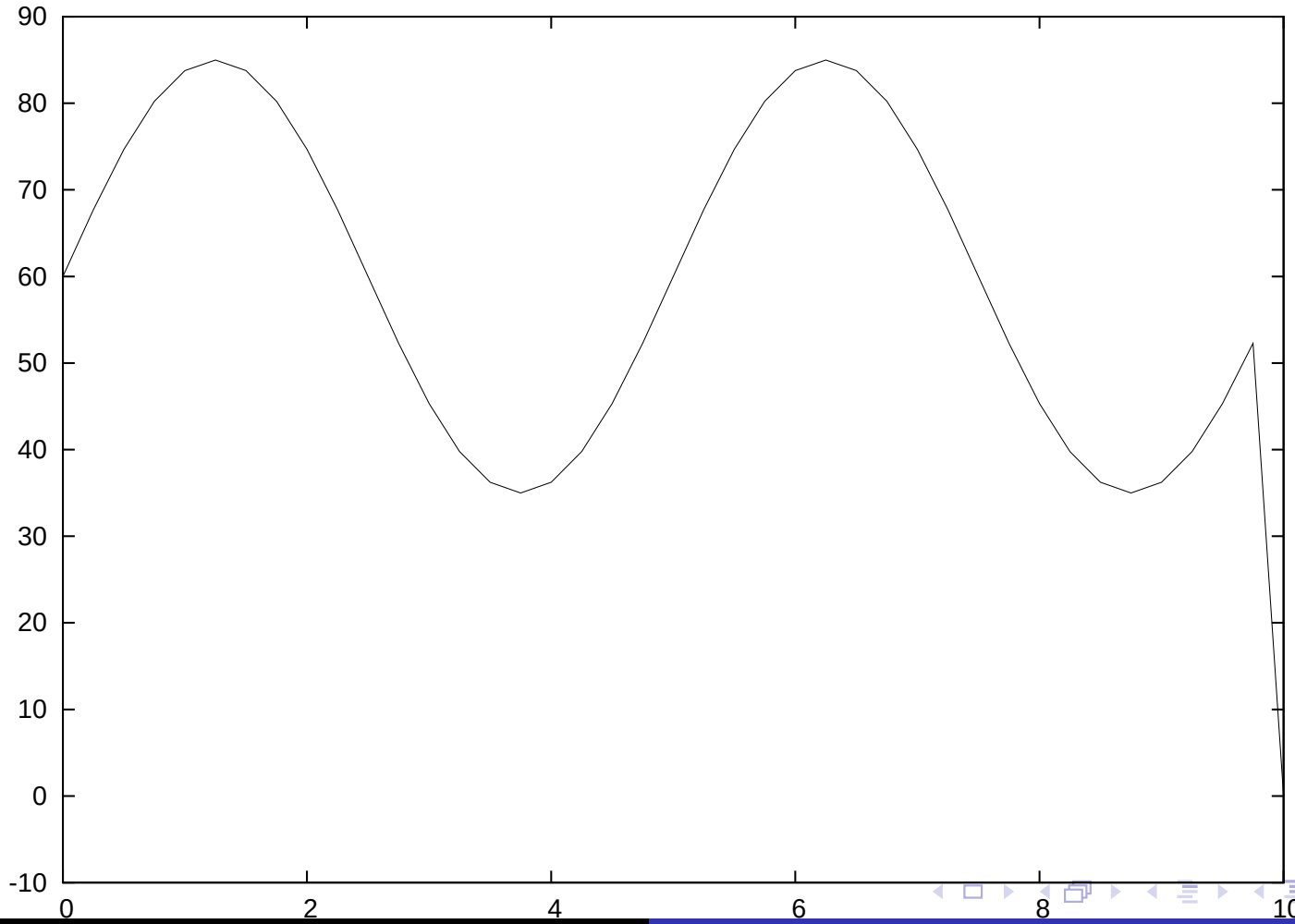
Control de un motor - Dispositivo



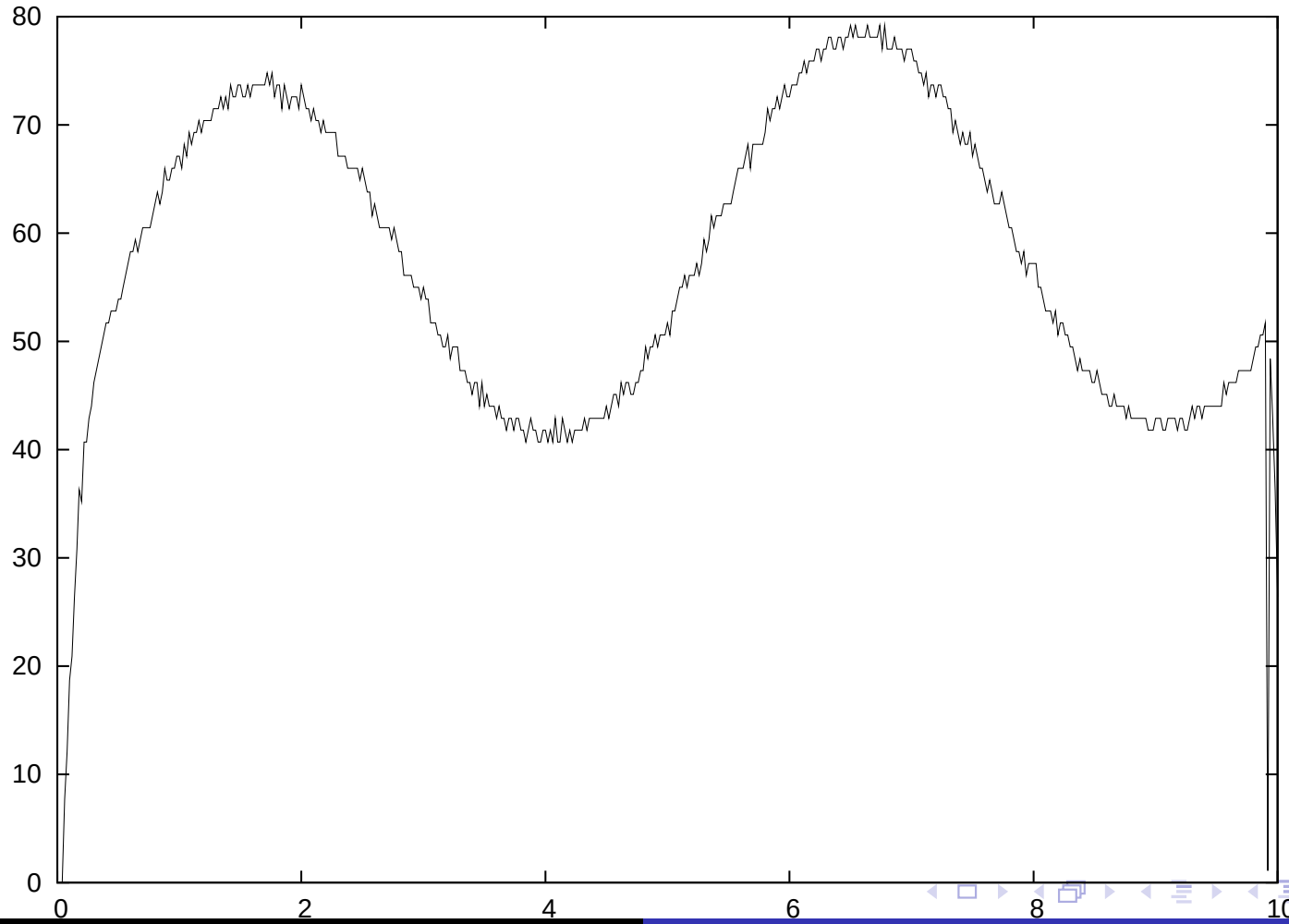
Control de un motor - Modelo DEVS



Control de un motor - Velocidad de referencia



Control de un motor - Velocidad medida



Publicaciones del Grupo sobre Métodos de QSS


E. Kofman and B. Zeigler.
DEVS Simulation of Marginally Stable Systems.
In *Proceedings of IMACS'05*, Paris, France, 2005.


E. Kofman.
A Third Order Discrete Event Simulation Method for Continuous System Simulation.
Latin American Applied Research, 36(2):101–108, 2006.


G. Migoni, E. Kofman, and F.E. Cellier.
Integración por Cuantificación de Sistemas Stiff.
Revista Iberoam. de Autom. e Inf. Industrial, 4(3):97–106, 2007.


F. Cellier and E. Kofman.
Continuous System Simulation.
Springer, New York, 2006.

Tesis y Trabajos de Conclusión de Grado

E. Pagliero and M. Lapadula.
Herramienta Integrada de Modelado y Simulación de Sistemas de Eventos Discretos.
Proyecto Final Ingeniería Electrónica. FCEIA, UNR, Argentina, Septiembre 2002.

E. Kofman.
Simulación y Control de Sistemas Continuos por Eventos Discretos.
PhD thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional de Rosario., Agosto 2003.

M. Bortolotto and F. Fontenla.
Fuentes Conmutadas DC-DC en PowerDEVS. Modelización y simulación.
Proyecto Final Ingeniería Electrónica. FCEIA, UNR, Argentina, Marzo 2007.

F. Bergero.
Desarrollo de una Plataforma de Simulación en Tiempo Real por Eventos Discretos.
Tesina de Licenciatura en Ciencias de la Computación. FCEIA, UNR, Argentina, Marzo 2008.

Publicaciones del Grupo sobre Control Muestreado

Ernesto Kofman.
Quantized-State Control. A Method for Discrete Event Control of Continuous Systems.
Latin American Applied Research, 33(4):399–406, 2003.

E. Kofman.
Discrete Event Control of Time Varying Plants.
Latin American Applied Research, 35(2):135–141, 2005.

J. Braslavsky, E. Kofman, and F. Felicioni.
Effects of Time Quantization and Noise in Level Crossing Sampling Stabilization.
In *Proceedings of AADECA 2006*, Buenos Aires, Argentina, 2006.

E. Kofman and J. Braslavsky.
Level Crossing Sampling in Feedback Stabilization under Data-Rate Constraints.
In *Proceedings of CDC'06, IEEE Conference on Decision and Control*, pages 4423–4428, San Diego, 2006.

S. Dormido, J. Sánchez, and E. Kofman.
Muestreo, control y comunicaciń basados en eventos.
Revista Iberoamericana de Automática e Informática Industrial, 5(1):5–26, 2008.

Publicaciones del Grupo sobre DEVS Estocásticos

E. Kofman and R.D. Castro.
STDEVS, A Novel Formalism for Modeling and Simulation of Stochastic Discrete Event Systems.
In *Proceedings of AADECA 2006*, Buenos Aires, Argentina, 2006.

R. Castro, E. Kofman, and G. Wainer.
A Formal Framework for Stochastic DEVS Modeling and Simulation.
In *Proceedings of HPCS 2008 (High Performance Computing and Simulation Symposium)*, Ottawa, Canada, 2008.