

Introducción a la programación RTL en C++ para simulación de procesamientos digitales de señales

Ing. Leonardo Luis Ortiz

Instituto Balseiro
Centro Atómico Bariloche

19 de septiembre de 2025



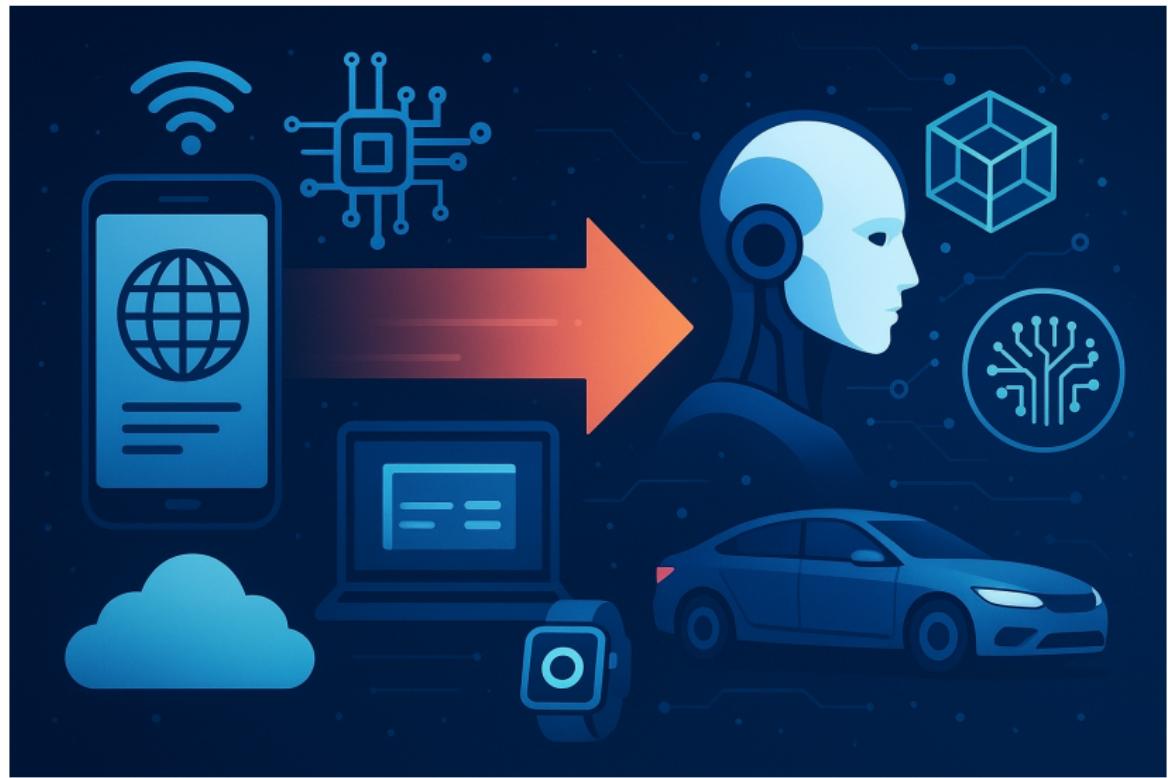
Contenido

- 1 Introducción
- 2 Lenguajes de Descripción de Hardware - HDL
- 3 Programación dirigida por eventos

Objetivos

- El dictado del curso se orienta a proveer al alumno de la capacidad de diseñar sistemas de procesamiento de señal, y utilizar el simulador HALCON como herramienta para la verificación y análisis.

Introducción



¿Por qué procesamiento de señales?

La siguientes áreas de Ciencia e Ingeniería se han beneficiado por el rápido crecimiento y avances de las técnicas de procesamiento de señales.

- Machine Learning
- Análisis de Datos
- Computer Vision
- Procesamiento de Imágenes e Imágenes Médicas
- Sistemas de Comunicación
- Electrónica de Potencia
- Probabilidad y Estadística
- Análisis Numérico
- Teoría de Decisión
- Circuitos Integrados

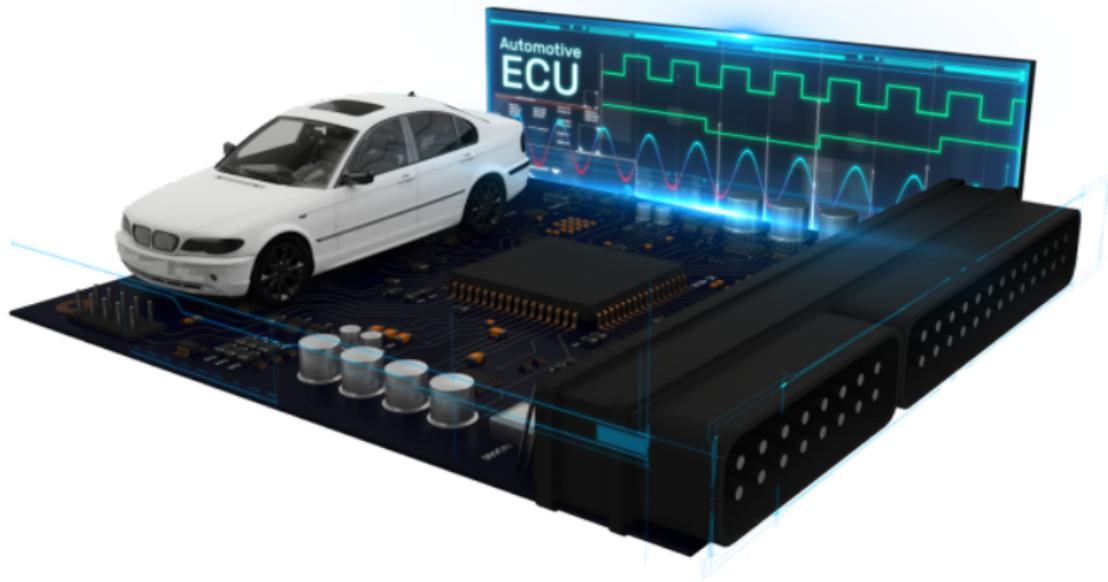
Procesamiento Digital de Señales - DSP

Contenidos mínimos necesarios para DSP

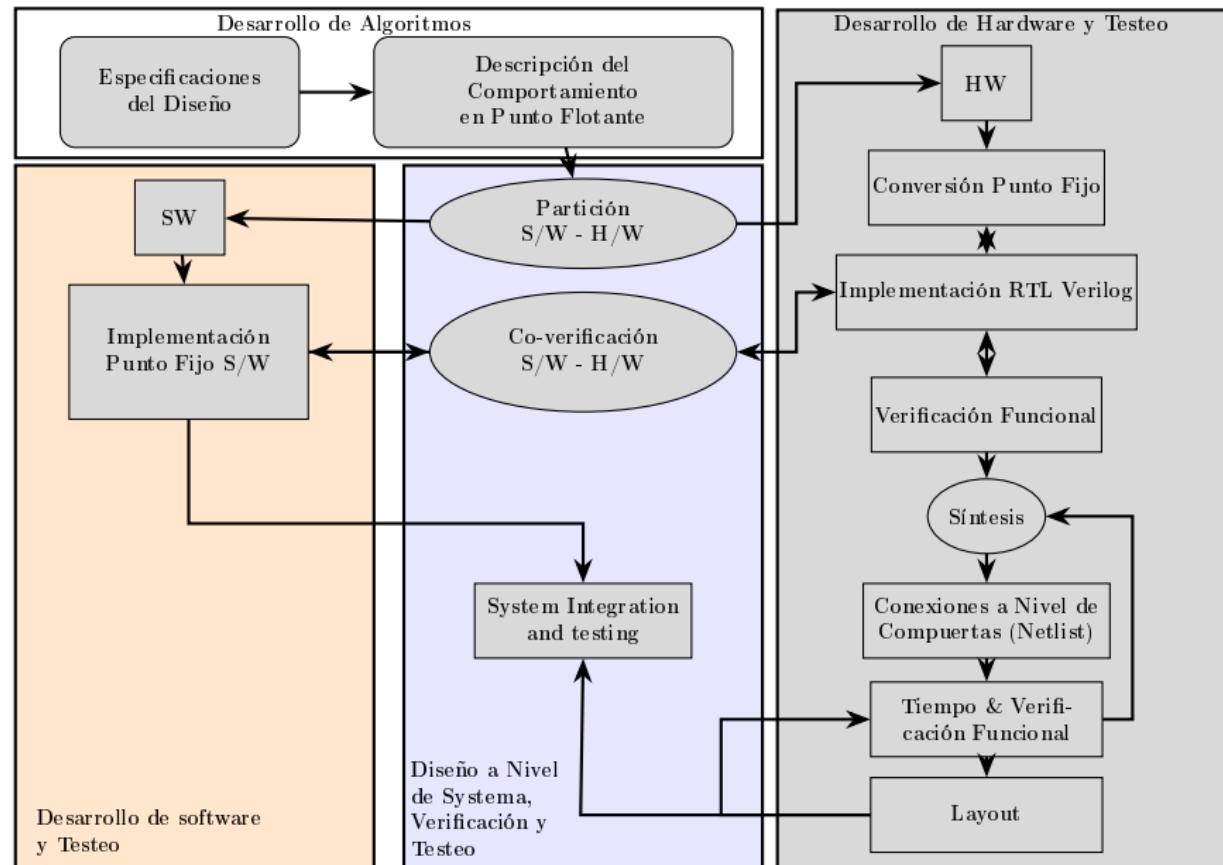
- Fundamentos de procesamiento de señales
- Conversión Analógica-Digital
- Sampleo y Reconstrucción
- Teorema de Nyquist
- Convolución
- Eliminación de ruido en señales
- Transformada de Fourier
- Diseño de filtros FIR e IIR

Diseño digital

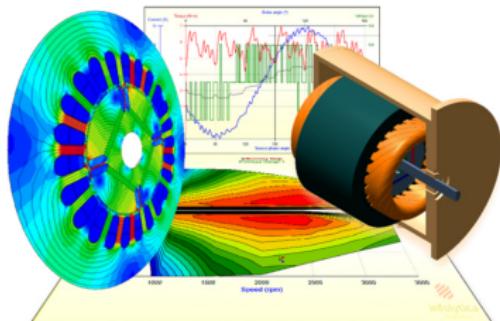
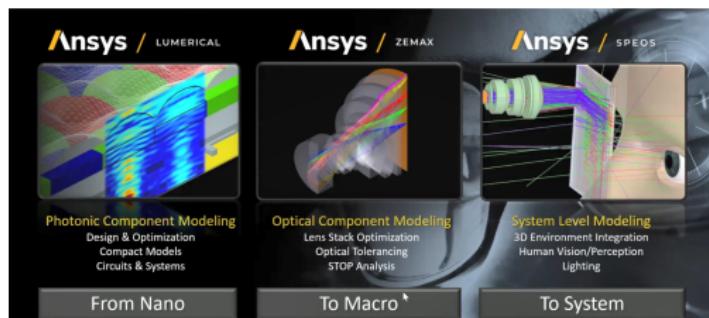
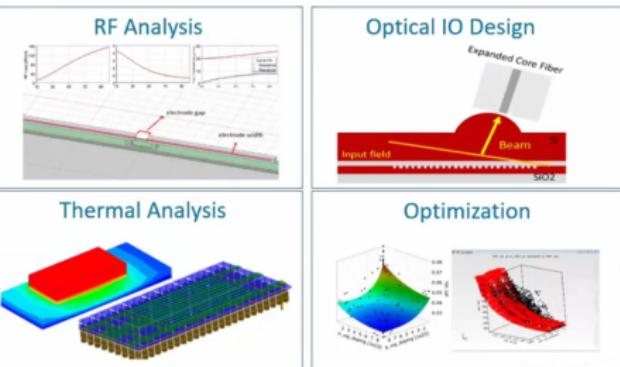
Diseño Digital es el proceso de diseño que utiliza componentes lógicos digitales para una aplicación específica



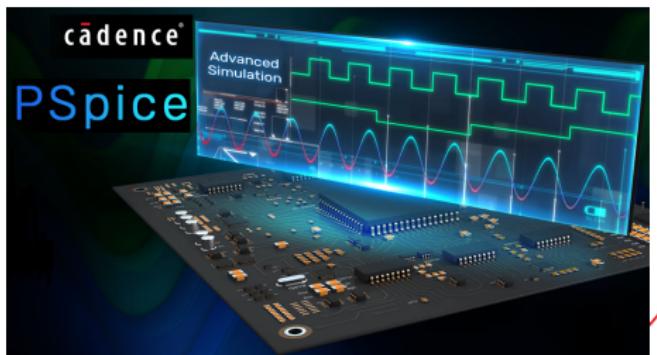
Diseño digital



Herramientas de Simulación



SIEMENS SIMCENTER
CAE/CFD SOFTWARE

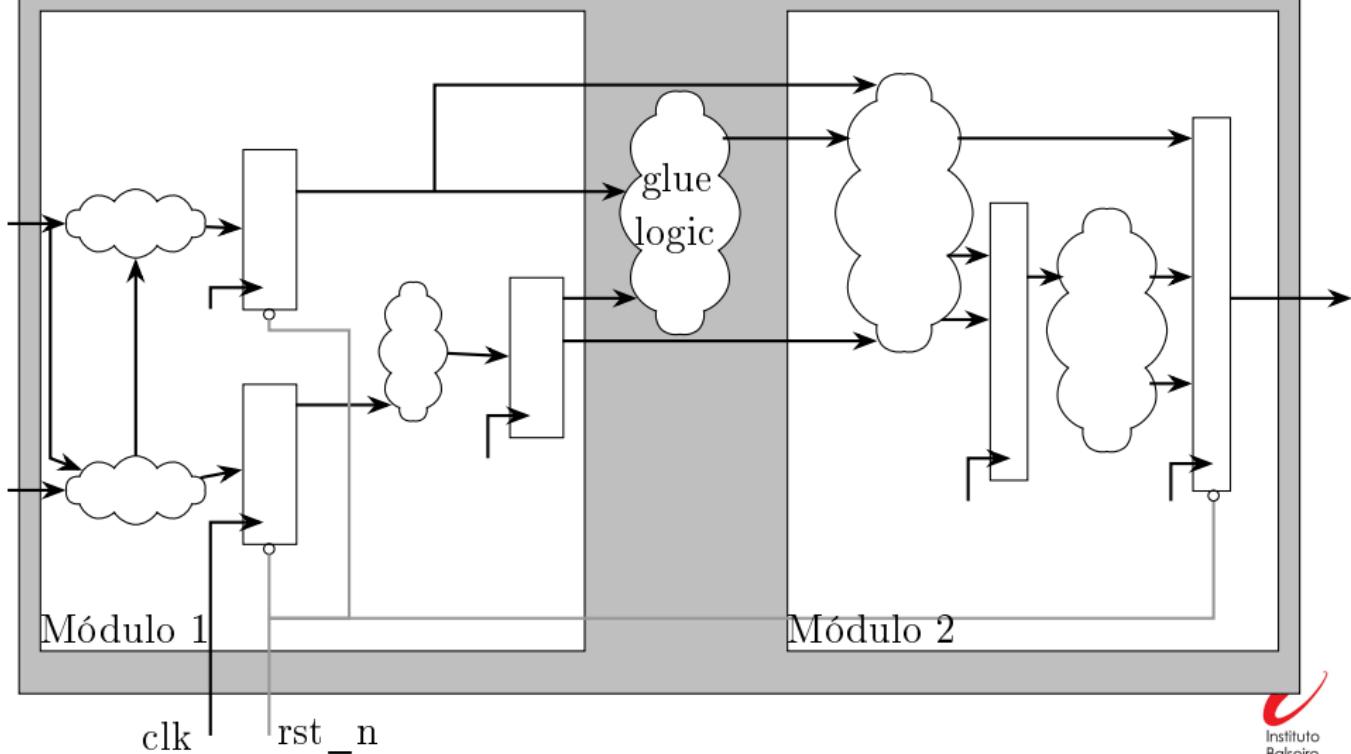


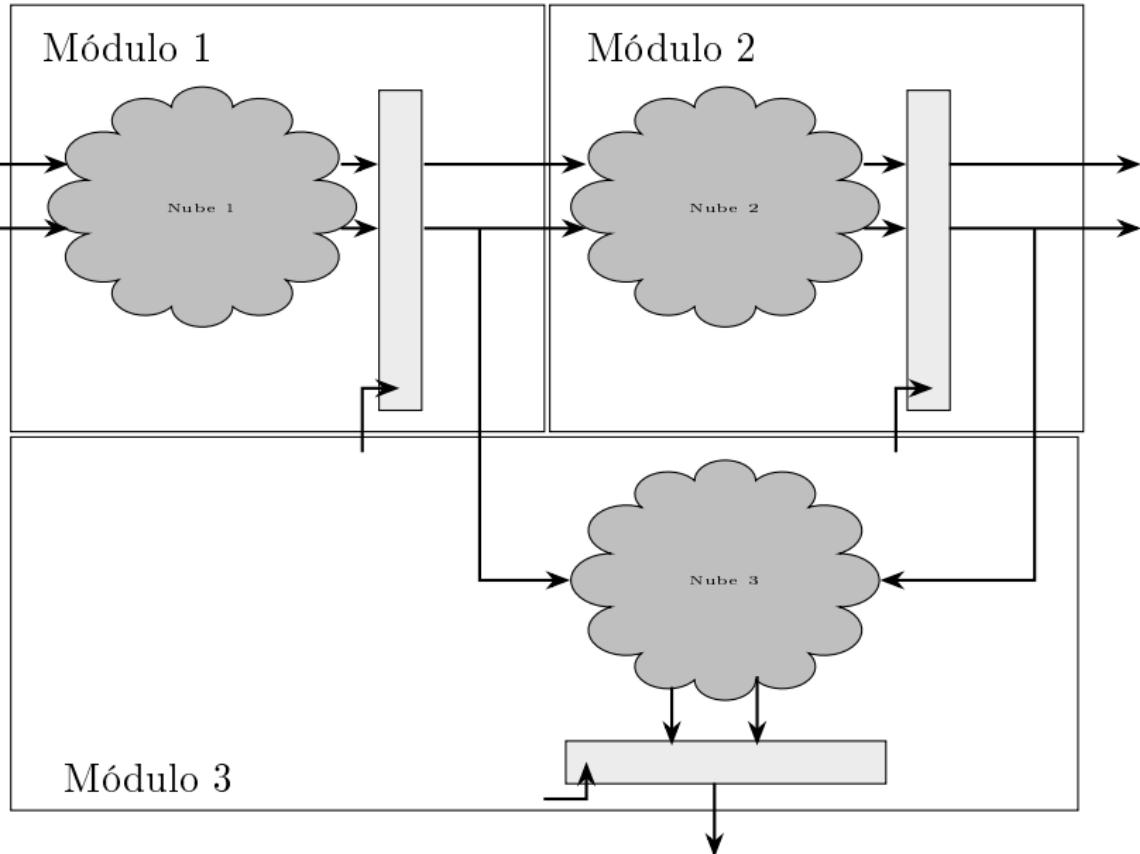
INSTITUTO
Balseiro

Lineamientos de Diseño RTL

- La partición de un diseño digital en una serie de módulos es importante.
- Un módulo no debe ser demasiado pequeño ni demasiado grande.
- Siempre que sea posible, el diseño debe dividirse de manera que los límites del módulo terminen en salidas de registros. Esto hará que sea más fácil sintetizar el módulo de nivel superior o síntesis jerárquica en cualquier nivel con limitaciones de tiempo.
- El diseñador también debe asegurarse de que ninguna combinación de nubes cruce los límites del módulo. Esto da a la herramienta de síntesis más facilidades para generar lógica optimizada.

Módulo de Nivel Superior





Lenguajes de Descripción de Hardware - HDL

En la actualidad se vuelve indispensable diseñar, modelar y simular el funcionamiento del hardware, principalmente en la industria ya que de este modo se evitan grandes pérdidas por errores de diseño. Para poder lograr esto existen los "Lenguajes de Descripción de Hardware".

Los Lenguajes de Descripción de Hardware (HDL) son una metodología de diseño digital de hardware a través de software.

Ventajas

- INDEPENDIENTE DE LA TECNOLOGÍA
- SOPORTA VARIOS ESTILOS DE DESCRIPCIÓN (Behavioral, Dataflow, Structured, Switch)
- AHORRO DE TIEMPO Y COSTOS

Lenguajes de Descripción de Hardware - HDL

Módulos

El módulo es la unidad básica de diseño, en la que se describe el funcionamiento de alguna unidad lógica. Puede existir un sólo módulo o un conjunto de módulos que se instancian con un módulo estructural.

Entradas y Salidas

Un módulo define la información que describe la relación entre las entradas y salidas de un circuito lógico. Las entradas y salidas están definidas por las palabras reservadas según sea el caso del lenguaje utilizado.

Primitivas

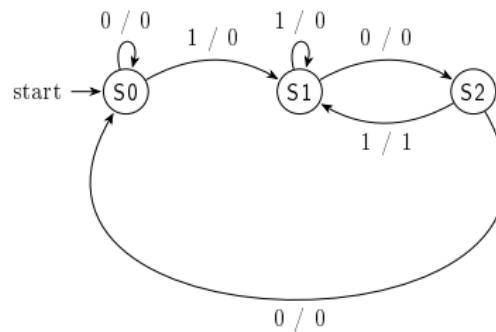
Elementos lógicos predefinidos, que son elementos estructurales que se pueden instanciar en un diseño más grande para formar una estructura más compleja.(Ej. en Verilog: and, or, xor, not)

Ejemplo de código - Detección de secuencia "101" de una cadena de bits

Entrada: *bit_in*

Salida: *seq_detected* (se activa cuando se detecta la secuencia 101)

Reset y reloj incluídos.



Etiquetas: "entrada / seq_detected" (Mealy)

Lenguajes de Descripción de Hardware - VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity seq_detector is
port (
clk : in STD_LOGIC;
rst : in STD_LOGIC;
bit_in : in STD_LOGIC;
seq_detected : out STD_LOGIC
);
end seq_detector;

architecture Behavioral of seq_detector is
type state_type is (S0, S1, S2);
signal state, next_state : state_type;
begin
process(clk, rst)
begin
if rst = '1' then
state <= S0;
elsif rising_edge(clk) then
state <= next_state;
end if;
end process;
end;
```

Lenguajes de Descripción de Hardware - VHDL

```
1      process(state, bit_in)           1
2      begin                           2
3          seq_detected <= '0';
4          case state is
5              when S0 =>
6                  if bit_in = '1' then
7                      next_state <= S1;
8                  else
9                      next_state <= S0;
10                 end if;
11             when S1 =>
12                 if bit_in = '0' then
13                     next_state <= S2;
14                 else
15                     next_state <= S1;
16                 end if;
17             when S2 =>
18                 if bit_in = '1' then
19                     next_state <= S1;
20                     seq_detected <= '1';
21                 else
22                     next_state <= S0;
23
24         end case;
25     end process;
26
27 end Behavioral;
```



Lenguajes de Descripción de Hardware - VHDL Testbench)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_seq_detector is
end tb_seq_detector;

architecture testbench of tb_seq_detector is

signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal bit_in : std_logic := '0';
signal seq_detected : std_logic;

-- Componente a testear
component seq_detector is
port (
clk : in std_logic;
rst : in std_logic;
bit_in : in std_logic;
seq_detected : out std_logic
);
end component;
```

Lenguajes de Descripción de Hardware - VHDL

Testbench

```
begin

    -- Instancia del DUT
    uut: seq_detector
        port map (
            clk          => clk,
            rst          => rst,
            bit_in       => bit_in,
            seq_detected => seq_detected
        );

    -- Generador de reloj
    clk_process: process
begin
    while true loop
        clk <= '0'; wait for 5 ns;
        clk <= '1'; wait for 5 ns;
    end loop;
end process;
```

Lenguajes de Descripción de Hardware - VHDL

Testbench

```
stim_proc: process
begin
    rst <= '1';
    wait for 12 ns;
    rst <= '0';

    bit_in <= '1'; wait for 10 ns;
    bit_in <= '0'; wait for 10 ns;
    bit_in <= '1'; wait for 10 ns;
    bit_in <= '0'; wait for 10 ns;
    bit_in <= '1'; wait for 10 ns;
    bit_in <= '1'; wait for 10 ns;
    bit_in <= '0'; wait for 10 ns;
    bit_in <= '1'; wait for 10 ns;

    wait for 20 ns;
```

Lenguajes de Descripción de Hardware - VHDL

Testbench

```
    wait;
end process;

end testbench;
```

Lenguajes de Descripción de Hardware - Verilog

```
1 module seq_detector (
2     input wire clk,
3     input wire rst,
4     input wire bit_in,
5     output reg seq_detected
6 );
7
8 parameter S0 = 2'b00,
9     S1 = 2'b01,
10    S2 = 2'b10;
11
12 reg [1:0] state, next_state;
13
14 // Registro de estado
15 always @(posedge clk or posedge rst) begin
16     if (rst)
17         state <= S0;
18     else
19         state <= next_state;
20 end
```

Lenguajes de Descripción de Hardware - Verilog

```
1 // State transition and  
2     output logic  
3 always @(*) begin  
4     seq_detected = 0;  
5     case (state)  
6         S0: begin  
7             if (bit_in)  
8                 next_state = S1;  
9             else  
10                next_state = S0;  
11        end  
12        S1: begin  
13            if (!bit_in)  
14                next_state = S2;  
15            else  
16                next_state = S1;  
17    end
```

```
1 S2: begin  
2     if (bit_in) begin  
3         next_state = S1;  
4         seq_detected = 1;  
5     end else  
6         next_state = S0;  
7     end  
8     default: next_state =  
9         S0;  
10    endcase  
11 end  
12 endmodule
```

Lenguajes de Descripción de Hardware - Verilog

Testbench

```
1  `timescale 1ns/1ps
2
3  module tb_seq_detector;
4      reg clk;
5      reg rst;
6      reg bit_in;
7      wire seq_detected;
8
9      seq_detector dut (      // Module instance
10         .clk(clk),
11         .rst(rst),
12         .bit_in(bit_in),
13         .seq_detected(seq_detected)
14     );
15
16    initial begin
17        clk = 0;
18        forever #5 clk = ~clk;
19    end
```

Lenguajes de Descripción de Hardware - Verilog

Testbench

```
1  initial begin
2      rst = 1;
3      bit_in = 0;
4      #12;
5      rst = 0;
6      #10 bit_in = 1; // ciclo 1
7      #10 bit_in = 0; // ciclo 2
8      #10 bit_in = 1; // ciclo 3
9      #10 bit_in = 0; // ciclo 4
10     #10 bit_in = 1; // ciclo 5
11     #20 $finish;
12 end
13
14 initial begin
15     $monitor("Time=%0t | bit_in=%b | seq_detected=%b |
16         state=%b",
17             $time, bit_in, seq_detected, dut.state);
18 end
19
20 endmodule
```

Lenguajes de Descripción de Hardware - SystemVerilog

```
1 module seq_detector(
2     input wire clk,
3     input wire rst,
4     input wire bit_in,
5     output reg seq_detected
6 );
7
8     typedef enum reg [1:0] { S0, S1, S2 } state_t;
9     state_t state, next_state;
10
11    always @ (posedge clk or posedge rst) begin
12        if (rst)
13            state <= S0;
14        else
15            state <= next_state;
16    end
17
18    always @ (*) begin
19        next_state = state;
20        seq_detected = 0;
```

Lenguajes de Descripción de Hardware - SystemVerilog

```
1    case (state)
2        S0: if (bit_in) next_state = S1;
3        S1: if (!bit_in) next_state = S2;
4        S2: begin
5            if (bit_in) begin
6                next_state = S1;
7                seq_detected = 1;
8            end else
9                next_state = S0;
10        end
11    endcase
12 end
13 endmodule
```

Lenguajes de Descripción de Hardware - SystemVerilog

Testbench

```
1 module tb_seq_detector;
2     logic clk = 0, rst;
3     logic bit_in;
4     logic seq_detected;
5
6     seq_detector uut (
7         .clk(clk), .rst(rst), .bit_in(bit_in),
8         .seq_detected(seq_detected)
9     );
10    always #5 clk = ~clk;
11
12    initial begin
13        rst = 1; bit_in = 0;
14        #10 rst = 0;
15
16        // Test sequence: 1 0 1
17        bit_in = 1; #10;
18        bit_in = 0; #10;
19        bit_in = 1; #10;
```

Lenguajes de Descripción de Hardware - SystemVerilog

Testbench

```
1      // Expected: seq_detected = 1
2      $display("Detected: %b", seq_detected);
3      $finish;
4  end
5 endmodule
```

Lenguajes de Descripción de Hardware - SystemC

```
1 // File: seq_detector.h
2 #ifndef SEQ_DETECTOR_H
3 #define SEQ_DETECTOR_H
4
5 #include <systemc.h>
6
7 SC_MODULE(SeqDetector) {
8     // Puertos
9     sc_in<bool> clk;
10    sc_in<bool> rst;
11    sc_in<bool> bit_in;
12    sc_out<bool> seq_detected;
13
14    // Estados internos (FSM)
15    enum State { S0, S1, S2 };
16    sc_signal<State> state;
```

Lenguajes de Descripción de Hardware - SystemC

```
1 // Proceso principal
2 void process_fsm() {
3     if (rst.read()) {
4         state.write(S0);
5         seq_detected.write(false);
6     } else if (clk.posedge()) {
7         State current_state = state.read();
8         bool next_detected = false;
9         State next_state = current_state;
```

Lenguajes de Descripción de Hardware - SystemC

```
1  switch (current_state) {
2      case S0:
3          next_state = (bit_in.read()) ? S1 : S0;
4          break;
5      case S1:
6          next_state = (!bit_in.read()) ? S2 : S1;
7          break;
8      case S2:
9          if (bit_in.read()) {
10              next_state = S1;
11              next_detected = true;
12          } else {
13              next_state = S0;
14          }
15          break;
16      }
```

Lenguajes de Descripción de Hardware - SystemC

```
1         state.write(next_state);
2         seq_detected.write(next_detected);
3     }
4 }
5
6 // Constructor
7 SC_CTOR(SeqDetector) {
8     SC_METHOD(process_fsm);
9     sensitive << clk.pos();
10    sensitive << rst;
11 }
12 };
13
14 #endif
```

Lenguajes de Descripción de Hardware - SystemC

Testbench

```
1 #include <systemc.h>
2 #include "seq_detector.h"
3
4 int sc_main(int argc, char* argv[]) {
5     // Signals
6     sc_signal<bool> rst;
7     sc_signal<bool> bit_in;
8     sc_signal<bool> seq_detected;
9     sc_clock clk("clk", 10, SC_NS); // Reloj de 10 ns
10
11     SeqDetector uut("uut"); // Instancia del modulo
12     uut.clk(clk);
13     uut.rst(rst);
14     uut.bit_in(bit_in);
15     uut.seq_detected(seq_detected);
16
17     // Trazado de signals (opcional)
18     sc_trace_file *wf =
        sc_create_vcd_trace_file("waveform");
```

Lenguajes de Descripción de Hardware - SystemC

Testbench

```
1 // Simulacion
2     rst = 1;
3     bit_in = 0;
4     sc_start(12, SC_NS); // primer ciclo
5
6     rst = 0;
7
8     // Aplicamos secuencia: 1 0 1 se debe detectar en el
9     // tercer bit
10    bit_in = 1; sc_start(10, SC_NS);
11    bit_in = 0; sc_start(10, SC_NS);
12    bit_in = 1; sc_start(10, SC_NS); // Aqui debe
13        detectarse
14
15    // Mas estímulos
16    bit_in = 0; sc_start(10, SC_NS);
17    bit_in = 1; sc_start(10, SC_NS);
18    bit_in = 1; sc_start(10, SC_NS);
19    bit_in = 0; sc_start(10, SC_NS);
```

C++ quasi-HDL

```
1 #include <iostream>
2 #include <vector>
3
4 enum State { S0, S1, S2 };
5
6 class SeqDetector {
7     private:
8         State state;
9         bool seq_detected;
10
11     public:
12         SeqDetector() {
13             reset();
14         }
15
16         void reset() {
17             state = S0;
18             seq_detected = false;
19         }
}
```

```
1
2         void clock_tick(bool bit_in) {
3             seq_detected = false;
4             switch (state) {
5                 case S0:
6                     state = (bit_in) ? S1 : S0;
7                     break;
8                 case S1:
9                     state = (!bit_in) ? S2 : S1;
10                    break;
11                 case S2:
12                     if (bit_in) {
13                         state = S1;
14                         seq_detected = true;
15                     } else {
16                         state = S0;
17                     }
18                     break;
19             }
}
```

C++ cuasi-HDL

```
1  bool output() const {
2      return seq_detected;
3  }
4
5
6  std::string state_str() const {
7      switch (state) {
8          case S0: return "S0";
9          case S1: return "S1";
10         case S2: return "S2";
11     }
12     return "ERROR";
13 }
14 };
```

C++ quasi-HDL- Testbench

```
1 SeqDetector fsm;
2
3 std::vector<bool> input_sequence = {1, 0, 1, 0, 1, 1, 0, 1};
4 std::cout << "Time | In | State | Detected\n";
5 std::cout << "-----\n";
6
7 for (size_t t = 0; t < input_sequence.size(); ++t) {
8     bool bit_in = input_sequence[t];
9     fsm.clock_tick(bit_in);
10    std::cout << " " << t << " | " << bit_in << " | "
11    << fsm.state_str() << " | "
12    << fsm.output() << "\n";
13 }
14
15 return 0;
16 }
```

Lenguajes de Descripción de Hardware - Resumen

VHDL:

- Fuertemente tipeado (ADA-like verbose syntax)
- Muy estricto en tipos
- Considerado más robusto por algunos ingenieros

Verilog:

- Lenguaje clásico HDL
- Sintaxis similar a C
- No tiene muchas características modernas como typedef, class o interface

SystemVerilog:

- Extiende Verilog con mejoras como logic, enum, typedef, interface, class, randomization, etc.
- Sintaxis compatible con Verilog, pero más poderoso para testbenches y sistemas complejos
- Permite diseño orientado a objetos (especialmente en verificación)

SystemC:

- Mayor Abstracción
- Diseño de Hardware y Software Conjunto
- Exploración de Arquitecturas
- Productividad
- Simulación Rápida
- Flexibilidad

Lenguajes de Descripción de Hardware - Resumen

Ventajas de los Lenguajes de Descripción de Hardware (HDL)

- **Design Verification:** permiten la verificación funcional temprana de los diseños. Los diseñadores trabajan a nivel de transferencia de registros (RTL), optimizando y modificando la descripción RTL hasta que cumpla con la funcionalidad deseada, lo que ayuda a eliminar la mayoría de los errores de diseño
- **Simulation:** permiten simular diseños, lo que posibilita realizar pruebas exhaustivas antes de su implementación en hardware
- **Parallel Programming:** VHDL (Very High-Speed Integrated Circuit Hardware Description Language, lenguaje de descripción de hardware para circuitos integrados de muy alta velocidad) también puede servir como lenguaje de programación paralelo de uso general

Lenguajes de Descripción de Hardware - Resumen

Desventajas de los Lenguajes de Descripción de Hardware (HDL)

- **Learning Curve:** tienen una curva de aprendizaje pronunciada para los principiantes debido a su sintaxis y conceptos específicos.
- **Debugging Challenges:** La depuración de diseños basados en HDL puede resultar compleja, especialmente cuando se trata de circuitos de gran tamaño.
- **Limited Visualization:** A diferencia de los diseños basados en esquemas, los HDL carecen de representaciones visuales, lo que dificulta la visualización del circuito.
- **Hardware Constraints:** Los HDL no siempre convierten perfectamente hardware real, y ciertas construcciones pueden no ser sintetizables.

Ventajas de los Lenguajes de Software

- **Versatility:** Los lenguajes de programación pueden utilizarse para una amplia gama de aplicaciones más allá del diseño de hardware.
- **Abstraction:** Los lenguajes de programación de alto nivel proporcionan abstracción, lo que facilita la expresión de algoritmos y lógica complejos.
- **Rich Libraries:** Los lenguajes de programación ofrecen amplias bibliotecas y marcos de trabajo para diversas tareas.

Desventajas de los Lenguajes de Software

- **Performance Overhead:** Los lenguajes interpretados pueden tener una sobrecarga de rendimiento en comparación con los lenguajes de bajo nivel como C o lenguaje ensamblador.
- **Resource Consumption:** Las aplicaciones de software consumen recursos del sistema (CPU, memoria), lo que afecta al rendimiento general del sistema.
- **Security Risks:** Los lenguajes de programación pueden introducir vulnerabilidades de seguridad si no se manejan con cuidado.

Puntos claves: Concurrency

La concurrencia, en el contexto de la programación, se refiere a la capacidad de un programa para descomponerse en partes que pueden ejecutarse de forma independiente entre sí.

Puntos claves: Concurrency

HDLs:

- La concurrencia en la descripción del hardware se refiere a la ejecución simultánea de múltiples operaciones dentro de un circuito digital.
- Modelan la naturaleza paralela de las operaciones del hardware, donde las señales se propagan a través de diferentes caminos simultáneamente y los componentes interactúan entre sí en paralelo.
- VHDL y Verilog admiten de forma inherente la ejecución concurrente, lo cual es crucial para simular cómo funcionan simultáneamente las diferentes partes de un circuito.
- Los HDL incluyen construcciones (procesos en VHDL, bloques always en Verilog) para modelar el comportamiento concurrente.

Puntos claves: Concurrency

GPLs:

- Suelen ejecutar instrucciones de forma secuencial, aunque el paralelismo y la concurrencia pueden lograrse explícitamente mediante hilos (threads), procesos o técnicas de programación asíncrona.
- Sin embargo, operan a un nivel más alto de abstracción y son gestionadas por el programador.

Puntos claves: Timing

HDLs:

- El estado de tiempos en la descripción del hardware se refiere a los retrasos de propagación y las restricciones de sincronización asociados con las transiciones de señales y las interacciones de los componentes dentro de un circuito digital.
- Proporcionan construcciones de tiempo para especificar restricciones de sincronización y retrasos, reflejando el comportamiento en tiempo real de los circuitos digitales.
- La información de sincronización es fundamental para la simulación y síntesis precisas de los diseños de hardware.

Puntos claves: Timing

GPLs:

- Las consideraciones de temporización en los GPL suelen gestionarse en un nivel de abstracción superior, por ejemplo, mediante funciones de suspensión o retardo, o se abstraen por completo en las aplicaciones de software.
- No disponen de construcciones integradas para especificar restricciones de temporización al nivel de los HDL.
- La información de temporización en los GPL es gestionada por el sistema operativo y no refleja directamente el comportamiento de temporización de los circuitos digitales.

Puntos claves: Applications

HDLs:

- Incluyen amplias funciones para la simulación y la verificación. Los ingenieros crean testbenches y herramientas de simulación para validar el comportamiento de los diseños de hardware en diferentes escenarios.
- Las herramientas de síntesis convierten las descripciones HDL en implementaciones a nivel de compuertas, que luego se utilizan para fabricar el hardware real.

Puntos claves: Applications

GPLs:

- Se centran en la corrección, la eficiencia y la funcionalidad de los algoritmos. Los desarrolladores utilizan herramientas de depuración, frameworks de pruebas y herramientas de análisis de código para garantizar la calidad del software.

Programación controlada por eventos

Programación Controlada por Eventos Event-Driven Programming - (EDP)

Programación controlada por eventos

“Un evento es un hecho inmutable, algo que ha sucedido en el pasado y que no se puede cambiar”.

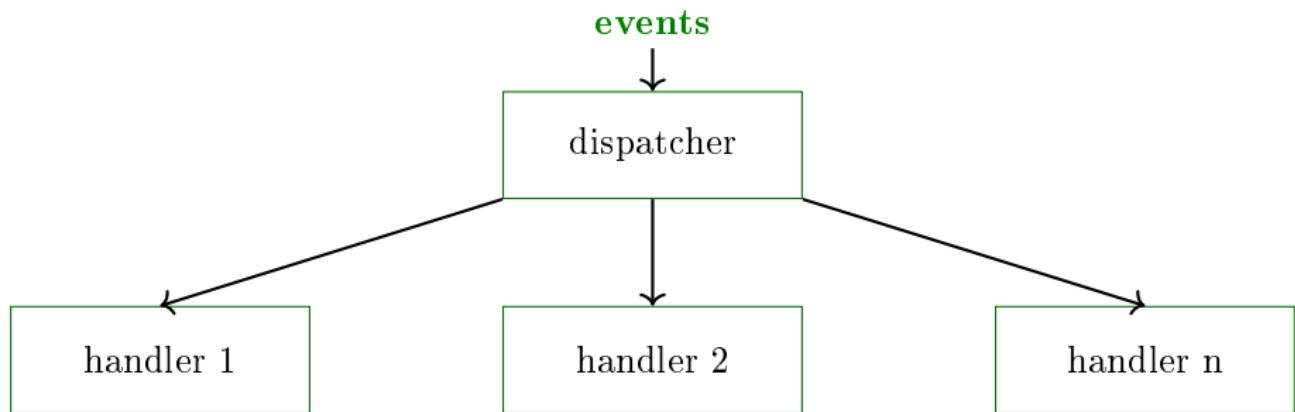
Un evento es un tipo de mensaje. Los mensajes pueden clasificarse en:

Commands

Events

Queries

Programación controlada por eventos



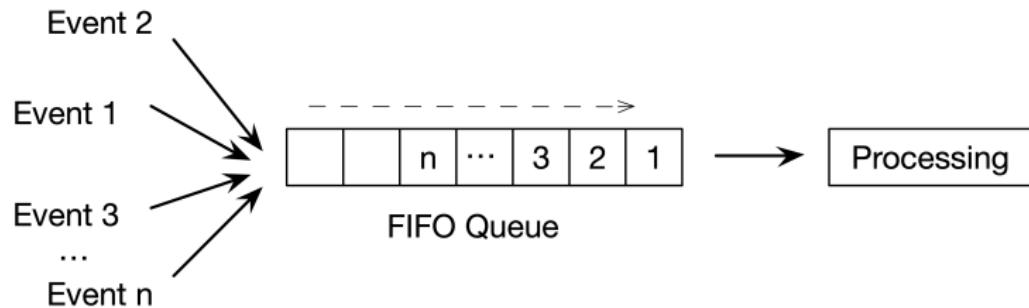
HANDLERS PATTERN

Programación controlada por eventos

```
1  #
2 do forever:    # the event loop
3     get an event from the input stream
4
5     if event type == EndOfEventStream :
6         quit # break out of event loop
7     if event type == ... :
8         call the appropriate handler subroutine, passing it event information as an
9             argument
10    elif event type == ... :
11        call the appropriate handler subroutine, passing it event information as an
12            argument
13    else:      # handle an unrecognized type of event
14        ignore the event, or raise an exception
15 #
```

Programación controlada por eventos

Event queue

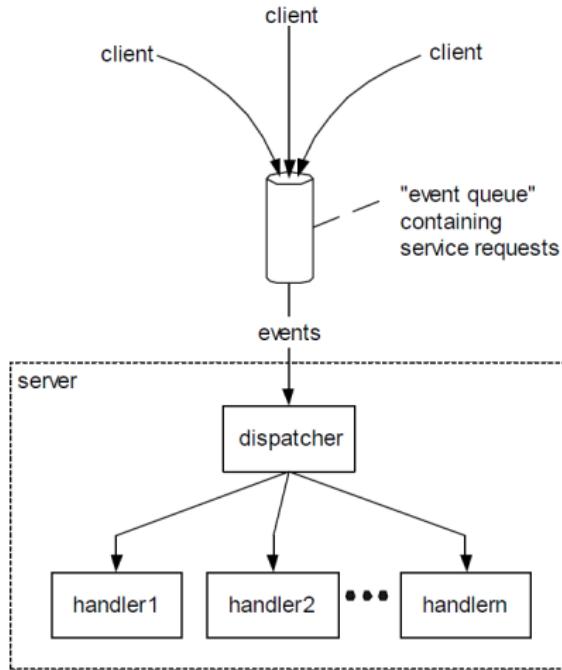


Arquitecturas controladas por eventos

- Arquitectura Cliente-Servidor
- Sistemas de mensajería
- OOP

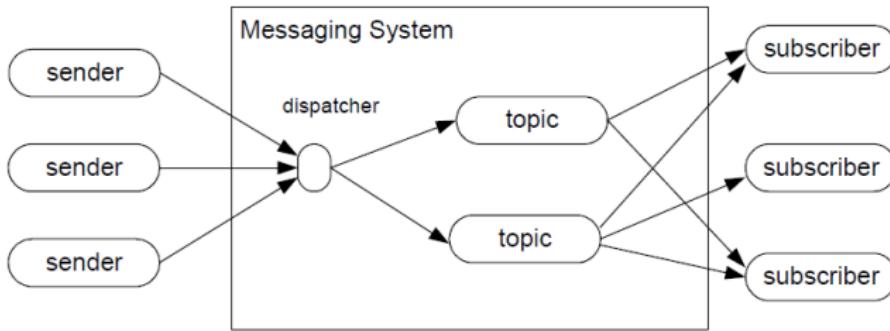
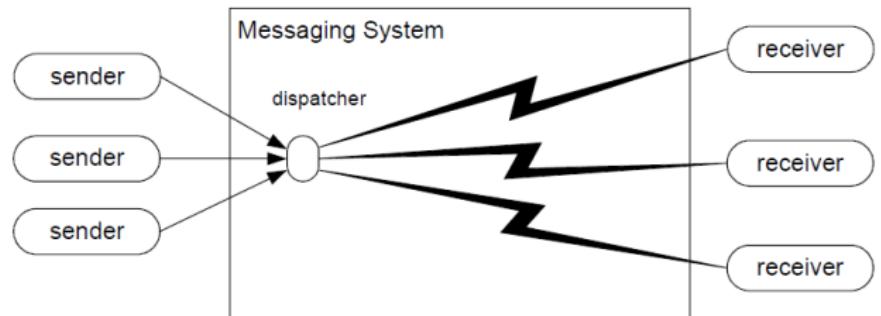
Arquitecturas controladas por eventos

Arquitectura Cliente-Servidor



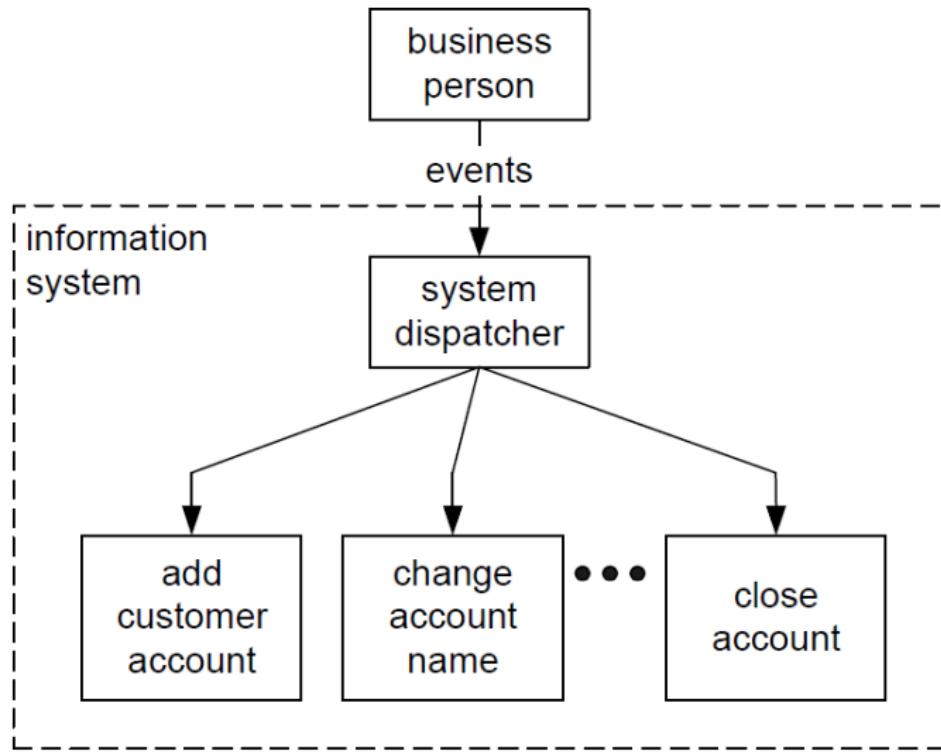
Arquitecturas controladas por eventos

Sistema de mensajería



Arquitecturas controladas por eventos

Object-Oriented Event-Driven Programming



Arquitecturas controladas por eventos

Frameworks

El término general para una pieza de software que funciona de esta manera (que define puntos de conexión y requiere complementos (plug-ins) es “framework”. Estas son algunas de las definiciones que encontrará si busca “framework” en Google.

Cada definición captura parte del concepto de un framework.

Arquitecturas controladas por eventos - Frameworks

Un framework es:

- una estructura básica que soporta o engloba algo más.
- una visión general, esquema o esqueleto, dentro del cual se pueden añadir detalles.
- un entorno de software extensible que se puede adaptar a las necesidades de un dominio específico.
- un conjunto de clases que proporciona una solución general a un problema de aplicación. Un framework suele perfeccionarse para abordar el problema específico mediante la especialización o mediante tipos o clases adicionales.
- Un componente que permite extender su funcionalidad mediante la escritura de módulos complementarios (“extensiones del framework”). El desarrollador de extensiones escribe clases derivadas de las interfaces definidas por el framework.

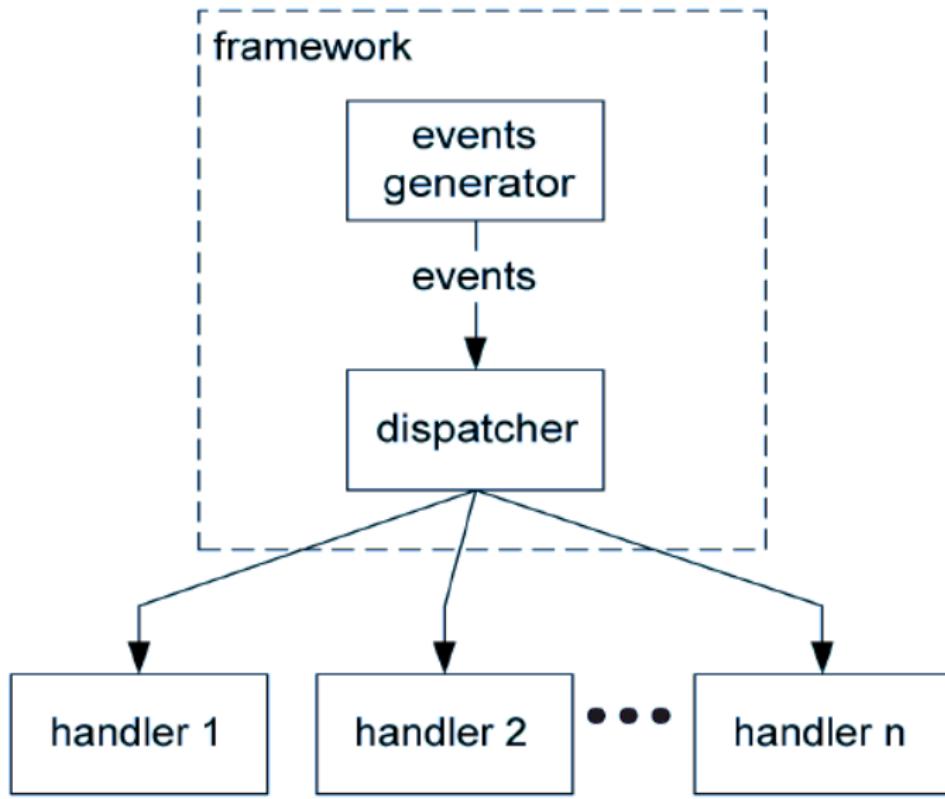
Arquitecturas controladas por eventos - Frameworks

Un framework es: (continuación)

- Un conjunto de software diseñado para una alta reutilización, con plug-points específicos para la funcionalidad requerida por un sistema en particular. Una vez proporcionados los plug-points, el sistema mostrará un comportamiento centrado en ellos.

El patrón que subyace al concepto de un framework es el patrón Handlers. Las extensiones del framework o complementos son controladores de eventos.

Arquitecturas controladas por eventos - Frameworks



Programación controlada por eventos

“La mayoría, si no todos, los sistemas y kits de herramientas GUI están diseñados para ser controlados por eventos, lo que significa que el flujo principal de su programa no es secuencial de principio a fin. Si nunca ha programado con GUI, este es uno de los cambios de paradigma más complejos.”

— Robin Dunn, hablando sobre programación GUI en OSCON2004

“Principio de Hollywood: "No nos llame; nosotros le llamaremos"… Implementa las interfaces, se registra. Se le llama cuando es el momento adecuado. Esto requiere una forma de pensar muy diferente a la que se enseña en la programación introductoria, donde el estudiante dicta el flujo de control.”

— Dafydd Rees, <http://c2.com/cgi/wiki?HollywoodPrinciple>

Programación controlada por eventos

Next ...

HALCON Simulator

¿Preguntas?