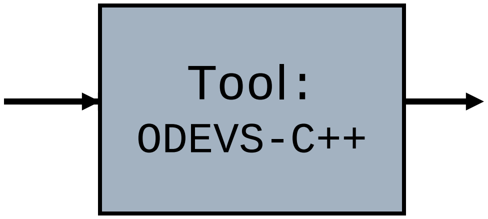
ECE575/ Chapter 7.
ODEVS-C++ Simulation Engine

Part2: System Simulation

Moon Ho Hwang
mhhwang@ece.arizona.edu

2006 Fall
ECE Department,
University of Arizona

Introduction of ODEVS-C++



- Motivation
    - Providing Open Source Project for DEVS
    - Providing a base library of DEVS-based Verification
- Strategy
    - As simple as possible
    - Testing in Visual.Net™ 2005 but not necessary => expanding platforms
    - Based on **classical** DEVS formalism

Tool:
ODEVS-C++

Directory Structure

- ODEVS (version 1.2.1)
  - *.h, *.cpp, ODEVS.sln, ODEVS.vcproj
  - Examples
    - Ex_ClientServer
    - Ex_DoublePingPong
    - Ex_PingPong
    - Ex_PingPongWithTable
    - Ex_Sim_All: Ex_Sim_All.sln //<- open this
    - Ex_Timer // atomic DEVS
    - Ex_TwoVendingMachine
    - Ex_VendingMachine // atomic DEVS

Three topics the user should understand.

Tool:
ODEVS-C++

- Event and its Couplings
- DEVS
  - Atomic DEVS
  - Coupled DEVS
- Scalable Real-time Engine: SRTEngine

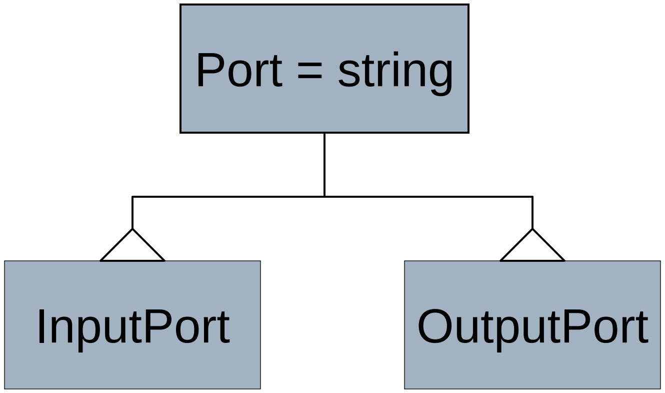
Event

- Example of Event
    - In HW#2, Y={!r:0, !r:1} of CC for red_off and red_on events.
    - In client-server example, X={?in:client} of buffer stands for an incoming client through in gate.
- Event=(**Port**, **Value**) in ODEVS-C++
    - **Port** is std::string.
    - **Value** is an abstract class.

Port Hierarchy (PortValue.h)

Tool:
ODEVS-C++

```
typedef std::string Port;

class InputPort: public Port
{
public:
	InputPort(){}
	InputPort(const std::string& name): Port(name){}
};

class OutputPort: public Port
{
public:
	OutputPort(){}
	OutputPort(const std::string& name): Port(name){}
};
```



Event

Value Class (PortValue.h)

Tool:
ODEVS-C++

```
/*-- abstract class of Value from which all the message value
    should be derived --*/
class Value
{
protected:
        Value(){}
public:
        //-- copy me to another instance
        virtual Value* Clone() const {return NULL;}
        //-- convert to string
        virtual std::string ToString() const {return std::string(); }
};
```

We can see `Client` class in `ClientServer` project as a derived class of Value.

These two virtual functions are **not** pure virtual. So override them for a concrete class **if needed**.

Event

Event=PortValue Class

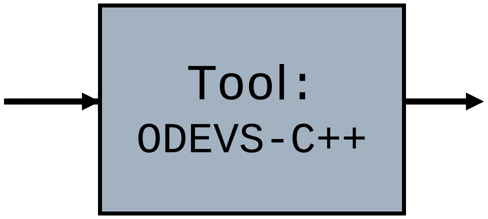
(PortValue.h)

Tool:
ODEVS-C++

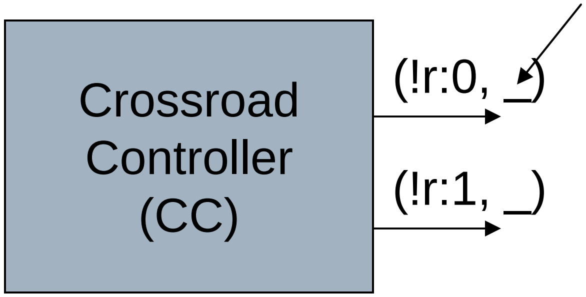
```
class PortValue
{
public:
        //--
        Port  port;
        //-- to use, safe dynamic_cast <>
        Value* value;
        //------------------------- constructors -------------------------
        PortValue(Port p="", Value* v=NULL){ SetPortValue(p, v); }
        PortValue(const PortValue& ob){ SetPortValue(ob.port, ob.value); }
        //---------------------- set or assign operator ------------------
        void SetPortValue(Port p, Value* v=NULL) { port = p; value = v;}
        const PortValue& operator=(const PortValue& ob)
        { SetPortValue(ob.port, ob.value); return *this; }

        std::string ToString() const
        {
                std::string str = port;
                if (value)
                        str += ":"+value->ToString();
                return str;
        }
};
```

Event

Representation of Y={ !r:0, !r:1}

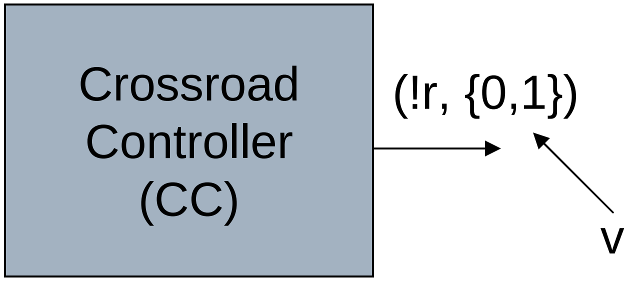


1. Use just port: Method 1



no value transmitted

2. Using a pair of (port, value): Method 2



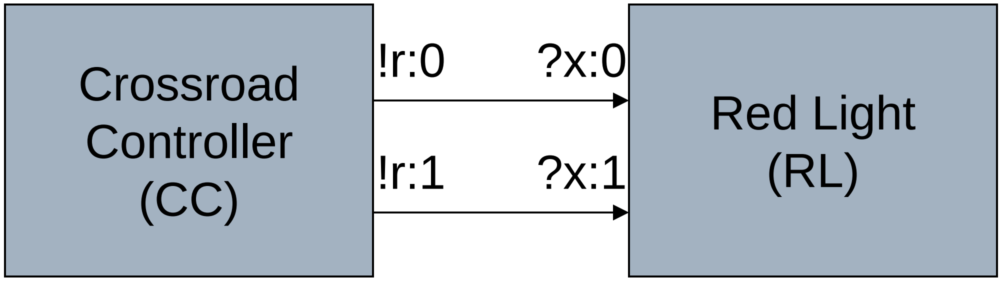
v∈{0,1} will be transmitted

Event

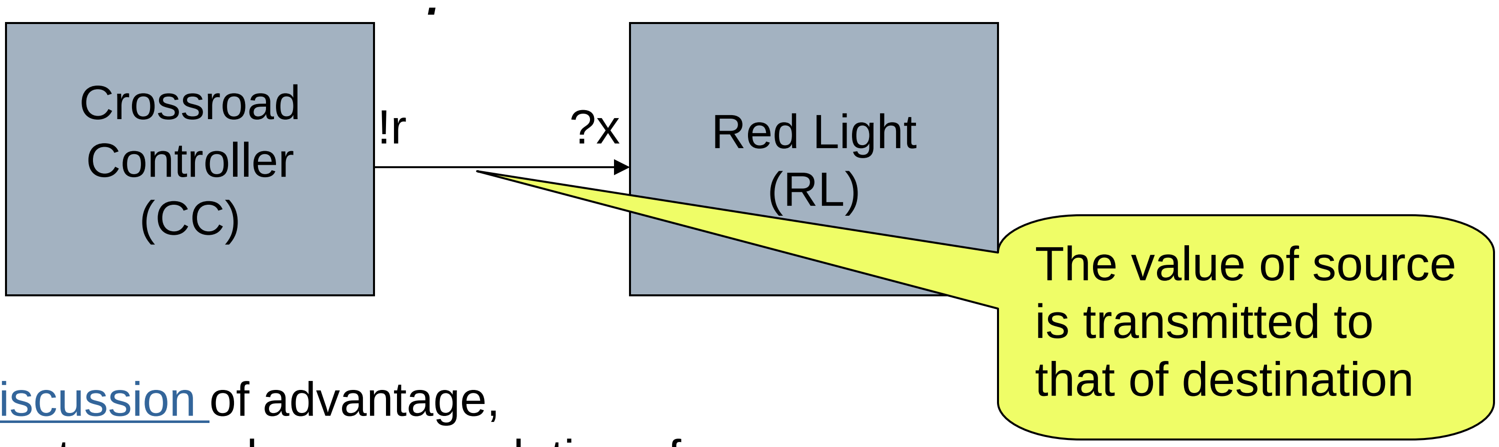
Coupling is, however, connection of two **ports**

Tool:
ODEVS-C++

1. Method1



2. Method2

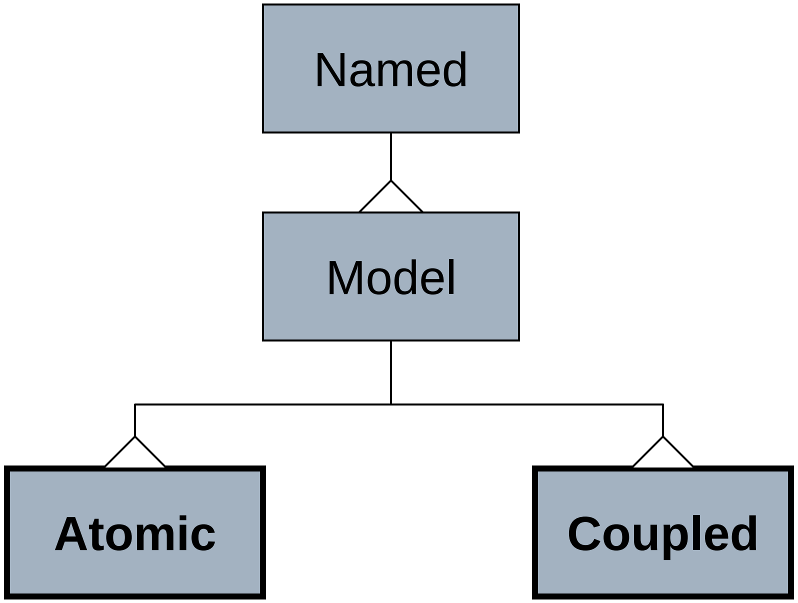


The value of source is transmitted to that of destination

See Discussion of advantage, disadvantage, and recommendation of these two methods.

Event

Hierarchy of DEVS Classes



Named

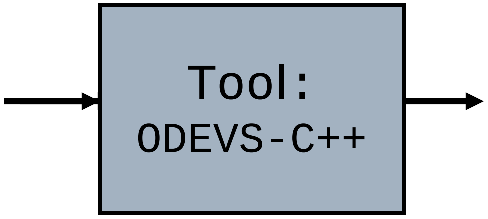
Model

Atomic

Coupled

DEVS Models

Named Class (Model.h)

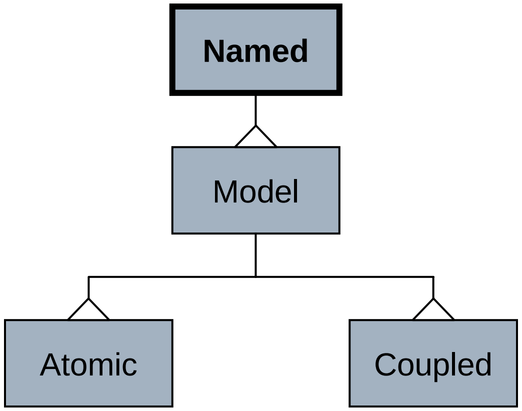


ODEVS_EXP is a macro for exporting and importing of DLL class for WIN32. See `ODEVS_Export.h`

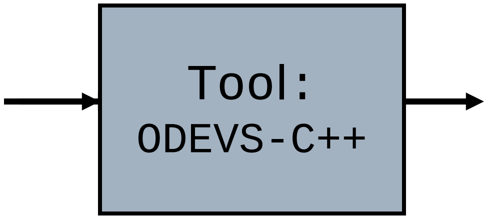
```
///--------A class having a name ------------------
class ODEVS_EXP Named
{
public:
        Named(const string& name):Name(name){}
        string Name;
};//----------------------------------------------
```

string is string of STL.

Derived class from Named class can have “public” data field Name.

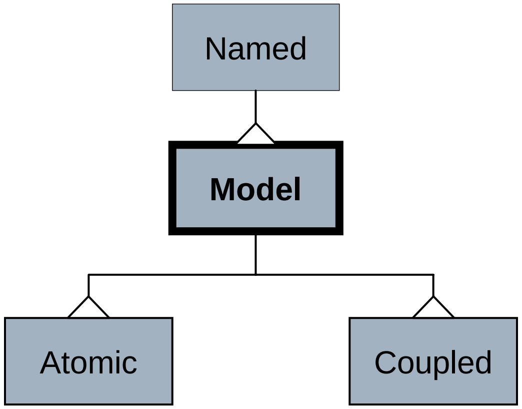


Model Class (`Model.h`)

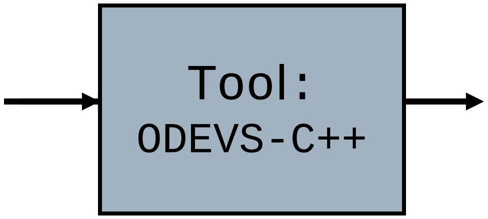


```
class ODEVS_EXP Model: public Named
{
public:
        /// get last-schedule-update time
        Time TimeLast() const ;
        /// get next-schedule time
        Time TimeNext() const ;
        /// get current time;
        static Time TimeCurrent() ;
        /// get reamining time to next schedule
        Time TimeRemaining() const ;
        /// get elapsed time since last schedule
        Time TimeElapsed() const ;

        // parent pointer
        Coupled*  Parent;
protected:
        static Time t_Current; // current time
};
```

typedef double Time;



Atomic Class (Atomic.h)



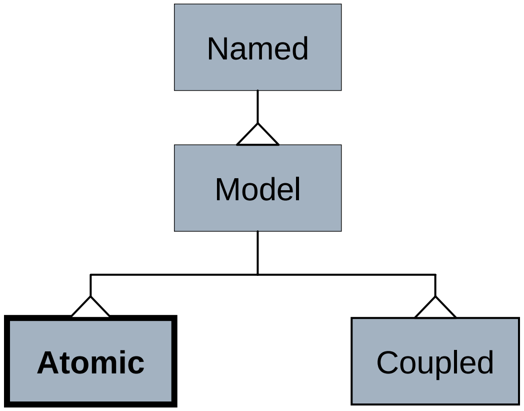
```
/// base class of atomic DEVS models
class ODEVS_EXP Atomic: public Model
{
public:
        /*-- 5 characteristic functions  --*/
        virtual void init() = 0;
        virtual void delta_ext(const PortValue& x) = 0;
        virtual void delta_int() = 0;
        virtual PortValue lambda() const = 0;
        virtual TimeSpan ta() const = 0;
        //---------------------------------------------

        //---- virtual function for tracing state ----------
        virtual string Get_s() const { return string(); }

protected:
        //-- function making reschedule after delta_ext
        void x_RescheduleMe() ;
…
};
```

5 Pure virtual functions. So we need to override them when deriving concrete class.

typedef double TimeSpan;

We need to override it for tracing execution run of derived class.

It is supposed to be used in overriding **delta_ext** when reschedule is needed.



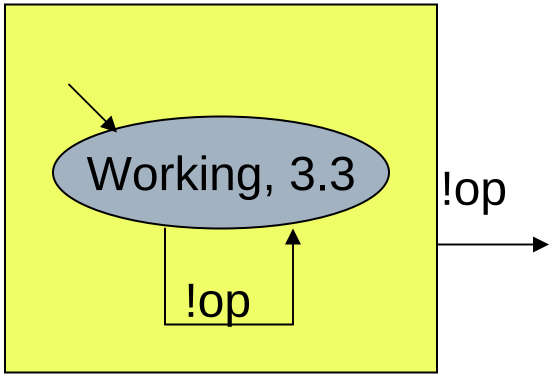
Ex1. Atomic: Ex_Timer

Tool:
ODEVS-C++

```
class SimplestTimer: public Atomic
{
public:
    const OutputPort op;
    SimplestTimer(const string& name=""): Atomic(name), op( "op" ){ }

    /*virtual*/ void init(){}
    /*virtual*/ void delta_ext( const PortValue& x) { }
    /*virtual*/ void delta_int() { };
    /*virtual*/ PortValue lambda() const { return PortValue(op); }
    /*virtual*/ Time ta() const { return 3.3; }
    /*virtual*/ string Get_s() const { return "Working"; }
};
```

Overriding 5 Pure virtual functions.

Overriding tracing state function.



State Transition Diagram of SimplestTimer

Atomic

Implementation of S and Q in ODEVS-C++

Tool:
ODEVS-C++

- Since ODEVS-C++ use the C++, we define S as *member data* of atomic DEVS.
- There is an *predefined variable* $\rho\in\{0,1\}$ indicating *if reschedule of external transition x is needed or not*.
- Let's say the user defined state set as S_u so that the set of states S can be defined as

$$S=\{(s,\rho)\text{: } s\in S_u,\, \rho\in\{0,1\}\}.$$

- To describe *explicitly* the continue case of external input, we introduce the *schedule time span* t_s into the total state such that

$$Q=\{(s,\rho,t_e,\ t_s)\,|\,(s,\rho)\in S, t_e\in[0,t_s], t_s\in\bigcup_{s\in S_u}\{ta(s)\}$$

$$\cup\{(\text{error},0,t_e,\infty)\,|\,t_e\in[0,\infty]\}.$$

Atomic

Tool:
ODEVS-C++

Total State Transition

- Given an atomic DEVS A, the total state transition of A is defines as follows: for $q=((s,\rho), t_e, t_s)\in Q$ and $z\in Z=X\cup Y^{\varepsilon}$,

  $$\delta((s,*,t_e,t_s),z)=$$

  $$\begin{cases} (s',*,\mathbf{0},ta(s')) & \text{for } z\in X, \delta_{ext}(s,*,t_e,z)=(s',\rho'), \boldsymbol{\rho}'=\mathbf{1}, \\ (s',*,\mathbf{t_e},t_s) & \text{for } z\in X, \delta_{ext}(s,*,t_e,z)=(s',\rho'), \boldsymbol{\rho}'=\mathbf{0}, \\ (s',*,0,ta(s')) & \text{for } \mathbf{t_e}=\mathbf{t_s}, z=\lambda(s), \delta_{\text{int}}(s)=s' \\ (\text{error},*,t_e,\infty) & \text{otherwise.} \end{cases}$$

  where * means "don't care the value.

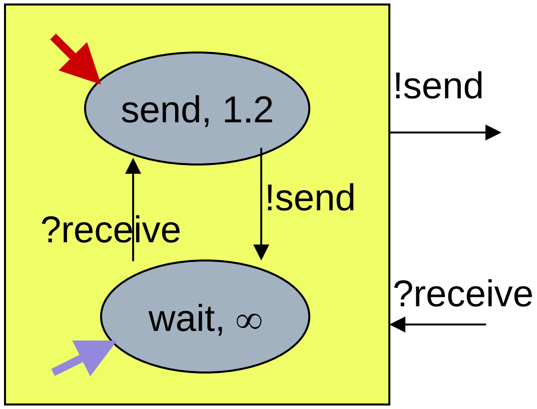
- Notice that t_e is a value of a clock changing continuously, while t_s is a variable changing when a discrete transition occurs.

Atomic

Ex2. Atomic: Ex_PingPong

Tool:
ODEVS-C++

```
const OutputPort  OP= "send";
const InputPort  IP= "receive";

const string  WAIT = "wait";
const string  SEND = "send";

class Player: public Atomic
{
protected:
      string    m_phase;      //
      bool      m_width_ball;
public:
      Player( const string& name="", bool with_ball= false):
               Atomic(name), m_phase(WAIT), m_width_ball(with_ball) { }

      /*virtual*/ void init()
      {
            if (m_width_ball)
                    m_phase = SEND;
            else
                    m_phase = WAIT;
       }
```



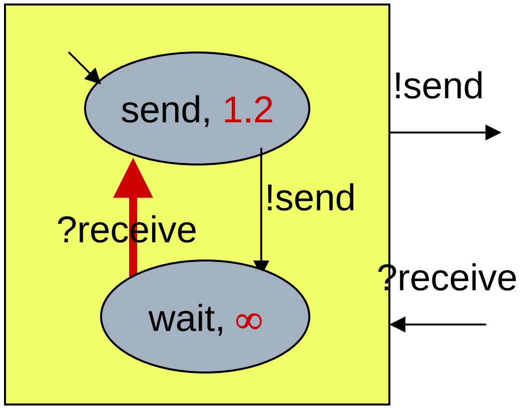
State Transition Diagram of Player

Atomic

Ex2. Atomic: Ex_PingPong

Tool:
ODEVS-C++

```
class Player: public Atomic
{
    …
    /*virtual*/ TimeSpan ta( ) const
    {
        if (m_phase == SEND)  return 1.2;
        else                  return DBL_MAX;
    }

    /*virtual*/ void delta_ext(const PortValue& x)
    {
        if (x.port == IP) // IP= "receive";
        {
            if (m_phase == WAIT) {
                m_phase = SEND;
                x_RescheduleMe(); // ← rho:=1
            }
        }
        /* the rest cases of
        delta_ext((s,0,t_s,t_e),x) = (s,0)*/
    }
}
```



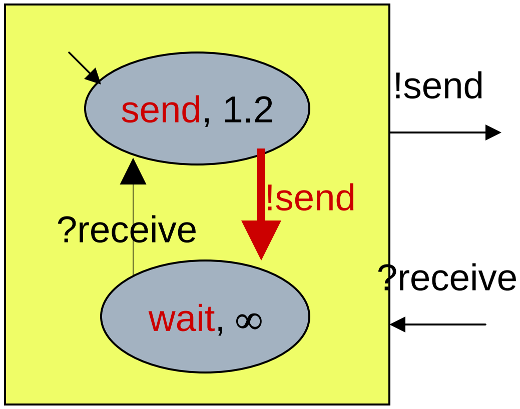
State Transition Diagram of Player

Atomic

Ex2. Atomic: Ex_PingPong

Tool:
ODEVS-C++

```
class Player: public Atomic
{
      …
      /*virtual*/ void delta_int( )
      {
            if (m_phase == SEND)
                  m_phase = WAIT;
      }

      /*virtual*/ PortValue lambda( ) const
      { return PortValue(OP); } // OP="send"


      /*virtual*/ string Get_s() const { return m_phase; }
};
```



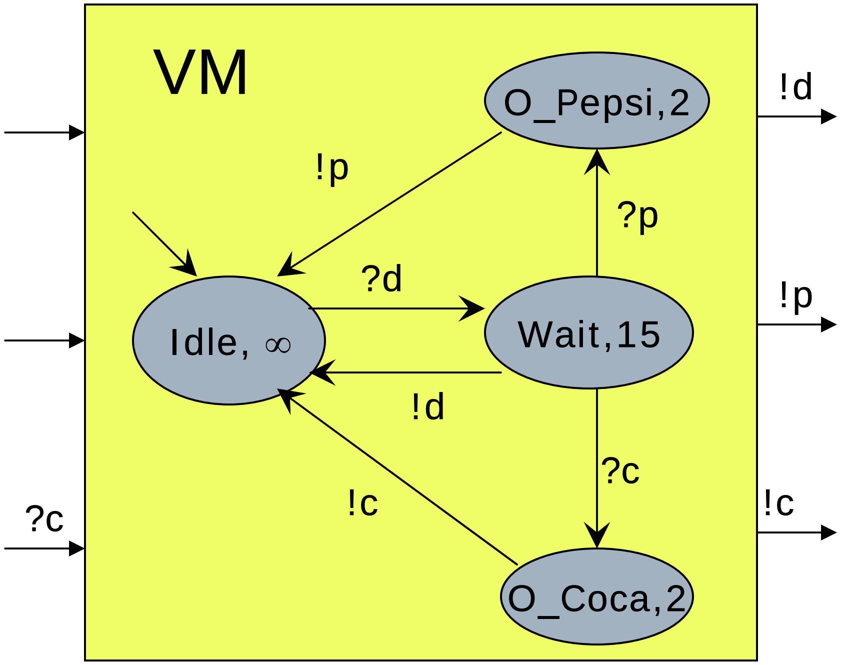
State Transition Diagram of Player

Atomic

Ex3. Atomic: Ex_VendingMachine

Tool:
ODEVS-C++

```
class VM: public Atomic
{
      …
    /*virtual*/ void delta_ext( const PortValue& x )
    {
        if (m_phase == IDLE && x.port == id){
            m_phase = WAIT;
            x_RescheduleMe(); // rho:=1
        } else if (m_phase == WAIT && x.port == ip) {
            m_phase = O_PEPSI;
            x_RescheduleMe(); // rho:=1
        } else if (m_phase == WAIT && x.port == ic) {
            m_phase = O_COCA;
            x_RescheduleMe(); // rho:=1
        }
        /* the rest cases of
           delta_ext((s,0,t_s,t_e),x) = (s,0)*/
    }
    …
};
```



State Transition Diagram of VM

Atomic

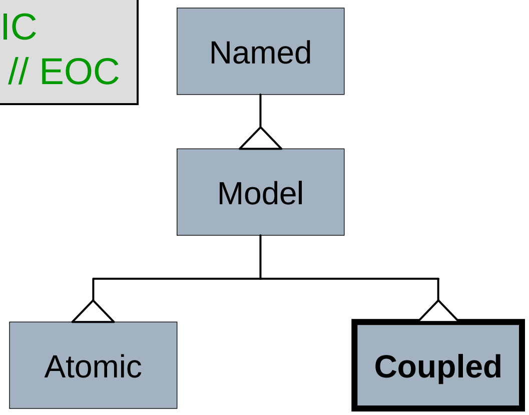
Coupled Class `(Coupled.h)`

Tool:
ODEVS-C++

```
//-- class of DEVS Network
class ODEVS_EXP Coupled: public Model
{
public:
    // constructor
    Coupled(const string& name=""): Model(name) {}
    // destructor
    virtual ~Coupled();

    // modelling related
    void AddModel(Model* md);
    Model* GetModel(const string& name) const;
    void Couple(Model* smd,  InputPort spt, Model* dmd, InputPort dpt);//EIC
    void Couple(Model* smd, OutputPort spt, Model* dmd, InputPort dpt);// IC
    void Couple(Model* smd, OutputPort spt, Model* dmd, OutputPort dpt); // EOC

    //-- print all couplings in a hierarchical manner
    void PrintCouplings() const;
}
```

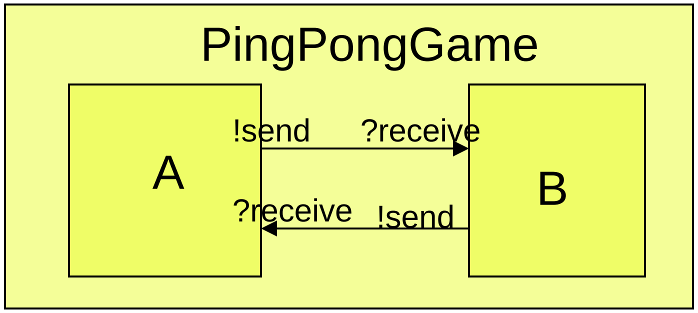
Adding a couple in either EIC, IC or EOC, respectively.

To be use for checking coupling structure.



Ex3. PingPong Network

Tool:
ODEVS-C++



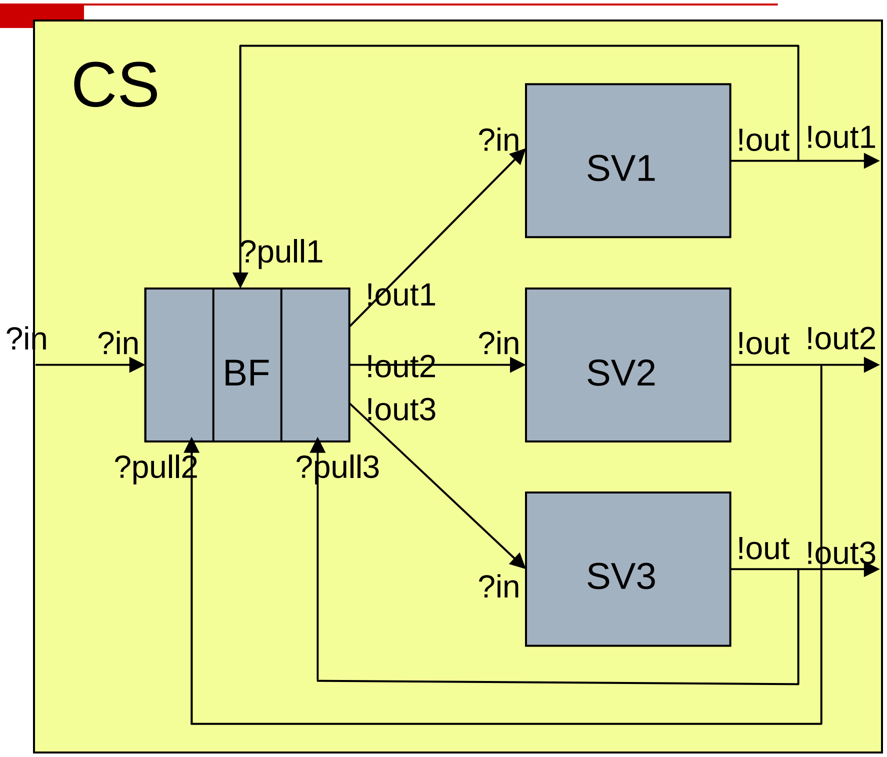
```
// Ex_PingPong.cpp

const OutputPort  OP= "send";
const InputPort  IP= "receive";

Coupled* MakePingPongGame( const string& name)
{
    //-- PingPong Game: Coupled Model ------ (1)
    Coupled* PingPong =  new Coupled(name);
    Player* A = new Player( "A" , true);
    Player* B = new Player( "B" , false);
    PingPong->AddModel(A);
    PingPong->AddModel(B);
    //-- Internal Coupling -------- (2)
    PingPong->Couple(A, OP, B, IP);
    PingPong->Couple(B, OP, A, IP);
    return PingPong;
}
```

Coupled

Ex4. Client-Server Network

Tool:
ODEVS-C++

```
Coupled* MakeClientServer( const string& name, int nservers)
{
    Coupled* CS = new Coupled(name);
    Buffer* bf = new Buffer("BF", nservers);
    CS->AddModel(bf);
    CS->Couple(CS, InputPort( "in" ), bf, bf->in); // EIC

    for( int i=0; i < nservers; i++) {
       char buffer[10]; _itoa_s(i, buffer, 9);
       Server* srv = new Server(string( "SV" )+buffer);
       CS->AddModel(srv);
       CS->Couple(bf, bf->out[i], srv, srv->in); // IC
       CS->Couple(srv, srv->out, bf, bf->pull[i]); // IC
       CS->Couple(srv, srv->out,
                       CS, OutputPort(string( "out" )+buffer)); // EOC
    }
    CS->PrintCouplings();
    return CS;
}
```



Coupled

Scalable Real-time Engine: SRTEngine (SRTEngine.{h,cpp})

Tool:
ODEVS-C++

- Functionalities
  1. Scalable Real-time: (Debugging Possible)
     Slower <-> Real-time <-> Faster
  2. Run-Through vs Step-by-Step
  3. Pause (at), Reset, and Rerun
  4. Execution of External Event as well as Internal Event
  5. Tracing Discrete State Transition
  6. Tracing Continuous State Transition
  7. Print Current Total State
  8. Print Hierarchical Coupling
  9. Providing a Text Menu in Console

SRTEngine

How to use SRTEngine
(Ex_Timer.cpp)

Tool:
ODEVS-C++

```
/* Model& modl: DEVS Model to be simulated;
   CallBack cbf: function pointer whose form is PortValue (*CallBack)() which
                 is supposed to return an input event to be used for injection.
   Time ending_t: simulation termination time;
*/
SRTEngine(Model& modl, CallBack cbf = NULL, Time ending_t = DBL_MAX);// constructor
...
void main( void )
{
        SimplestTimer* STimer1 = new SimplestTimer( "STimer" ) ; //-- simulation model
        SRTEngine simEngine(*STimer1); // plug-in sim. model to sim. engine
        simEngine.RunConsoleMenu(); // run the interactive menu in console
        delete STimer1;
}
```

- You can see the following menu items in the console window.

  scale, run, step, [p]ause, pause_at, reset, rerun, inject, ctmode, dtmode, print, cls, exit

SRTEngine

Tool:
ODEVS-C++

scale *value*

- If *value* = 1, it runs as real-time (default).
- If *value* < 1, it runs slower than real-time.
- If *value* > 1, it runs faster than real-time.
- If *value* <= 0 or *value* > 1.0E+06, it runs as fast as possible.

- Related Function

```
void SRTEngine::SetTimeScale(double value);
```

SRTEngine

Tool:
ODEVS-C++

run

- `run` performs a simulation run unless (1) stopped by `pause` command or (2) reaches the simulation termination time which is set by `pause_at` command.
- `run` can be useful for *long-run simulation*.
- During the simulation, it shows the model status depending on modes which are selected by `trace_d` and/or `tace_c` commands.
- Related Function

```
void SRTEngine::Run();
```

SRTEngine

Tool:
ODEVS-C++

step

- step performs a step-by-step simulation which means it stops when it find a *discrete state transition*.
- step command can be useful during *model debugging*.
- During the simulation, it shows the model status depending on modes which are selected by `trace_d` and/or `tace_c` commands.
- Related Function

```
void SRTEngine::Step();
```

SRTEngine

Tool:
ODEVS-C++

`[p]ause & pause_at` *time*

- pause or just p stops a simulation run instantly.
- Related Function

```
void SRTEngine::Pause();
```

- pause_at *time* sets the simulation termination time at *time*.
- *time* should be grater than the current time.
- Related Function

```
void SRTEngine::SetEndingTime(Time time);
```

SRTEngine

Tool:
ODEVS-C++

reset & rerun

- `reset` stops the simulation run instantly, and initializes the model.
- Related Function

```
void SRTEngine::Reset();
```

- `rerun` performs reset and then run.
- Related Function

```
void SRTEngine::Rerun();
```

SRTEngine

inject [x]

Tool:
ODEVS-C++

- `inject` transmits an input events to the simulation model through the simulation engine.
- `inject` command is related to the callback function whose type is

`PortValue call_back_function()`. The following is an example of `Ex_ClientServer`.

```
PortValue InjectMsg()
{
	return PortValue(InputPort("in"), new Client(60));
}
```

- The callback function is passed as the second argument when instancing of SRTEngine.

SRTEngine simEngine(*vm, **InjectMsg**); // in `Ex_ClientServer`

SRTEngine

Tool:
ODEVS-C++

inject [x] (continued)

- We can generates the input event depending on the user input. (see Ex_VendingMachine)

```
PortValue InjectMsg()
{
        string input;
        cout << "[d]ollar [p]epsi_botton [c]oca_botton > " ;
        cin >> input;
        if (input == "d" )  return PortValue( "dollar" );
        else if (input == "p" )  return PortValue( "pepsi_btn" );
        else if (input == "c" ) return PortValue( "coca_btn" );
        else {cout << "Invalid input! Try again! \n" ;
            return PortValue();
        }
}
```

- Related Function

`void SRTEngine::Inject(PortValue pv);`

SRTEngine

dtmode {none, tr, te,}

Tool:
ODEVS-C++

- dtmode sets the *discrete* trace mode as none, tr, or te
  - none : no trace
  - tr : trace with remaining time
  - te : trace with elapsed time (Default)
- The *discrete trace* means tracing when a discrete state transition occurs.
- Related Function

```
void SRTEngine::Set_dtmode(PrintStateMode
    md );
```

SRTEngine

Tool:
ODEVS-C++

ctmode {none, *value*}

- ctmode sets the *continuous* trace mode as none or *value*
  - none : no trace
  - *value* : tracing interval in second (default *value*=0.25, i.e. 4 frames/sec)

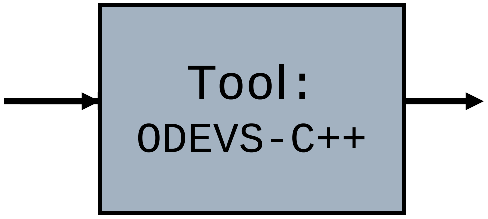
- The *continuous trace* means tracing when the *value* amount time has been passed.

- Related Function

```
void SRTEngine::SetAnimationFlag(bool flg);
void SRTEngine::SetAnimationInterval(TimeSpan ai);
```

SRTEngine

print {q, cpl, s}



- print shows the status of model in terms of
  - q : the current total state
  - cpl : hierarchical couplings if the model is Coupled
  - s : the current settings of the simulation environment

- Related Function

```
String Model::Get_q(bool remaining) const;
void Coupled::PrintCouplings() const;
void SRTEngine::PrintSettings() const;
```

SRTEngine

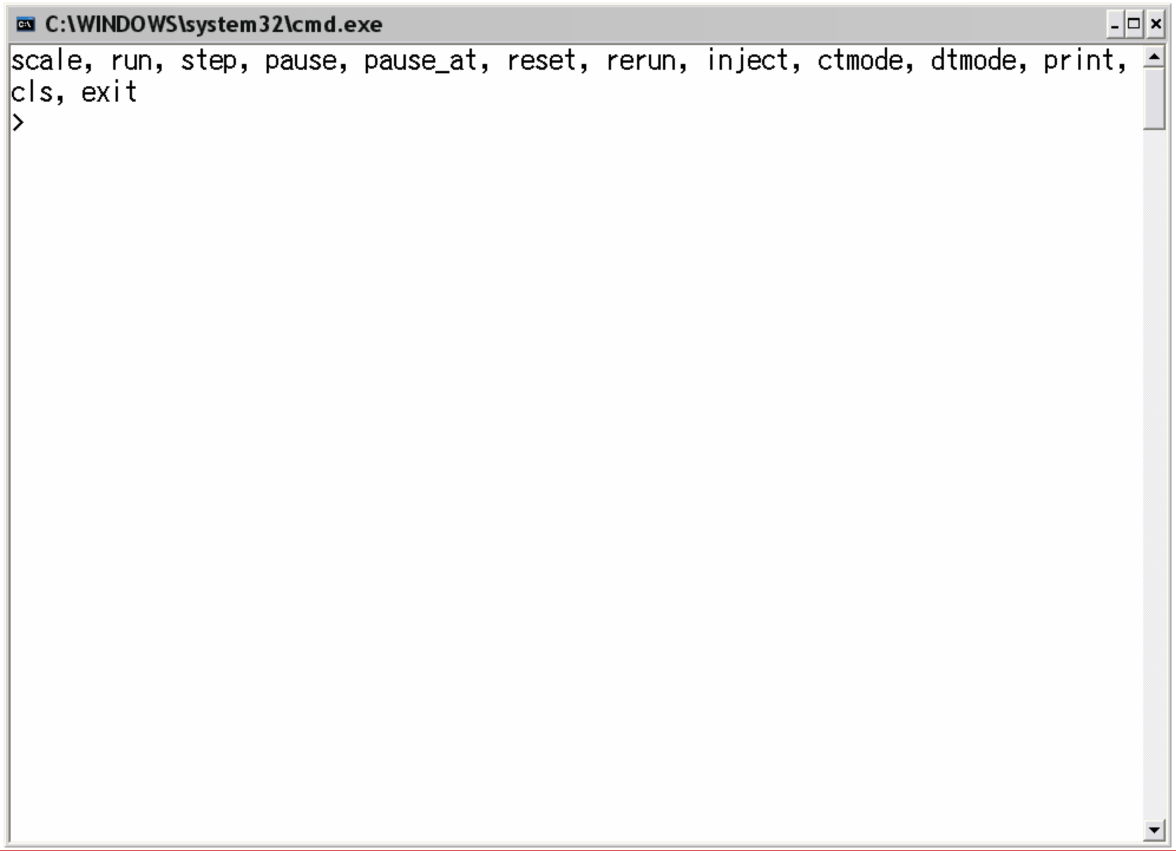
`cls` & `exit`

- `cls` clears the screen.
- `exit` exits the loop of console menu.

Case Study of Ex_Timer: Active Atomic DEVS

Tool: ODEVS-C++

- Open `ODEVS/Examples/Ex_Sim_All/*.sln` file.
- Set `Ex_Timer` as StartUp Project and Start it.

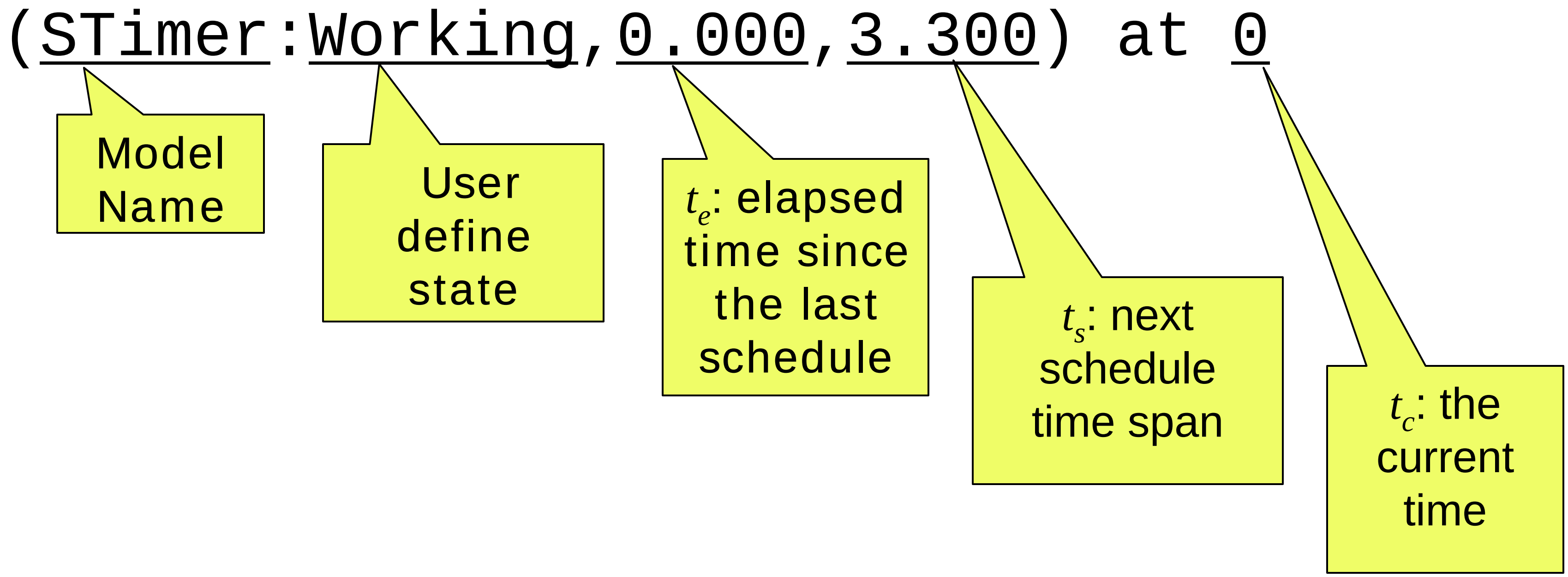


```
scale, run, step, pause, pause_at, reset, rerun, inject, ctmode, dtmode, print,
cls, exit
>
```

Case Study

Case Study of Ex_Timer: Active Atomic DEVS

Tool: ODEVS-C++



```
>print
options: q cpl > q
(STimer:Working,0.000,3.300) at 0
```

Model Name

User define state

t_e: elapsed time since the last schedule

t_s: next schedule time span

t_c: the current time

* Notice that ρ is not printed but used inside for rescheduling of $\delta_{ext}(q,z)$.

Case Study

Case Study of Ex_Timer: Active Atomic DEVS

Tool:
ODEVS-C++

```
>dtmode
options: none tr te > tr
>print
options: q cpl > q
(STimer:Working,3.300,3.300) at 0
```

Trace with remaining time

t_s: next schedule time span

t_r: remaining time to the next schedule

t_c: the current time

```
>dtmode
options: none tr te > te
```

Get back to the elapsed time mode

Case Study

Case Study of Ex_Timer: Active Atomic DEVS

Tool: ODEVS-C++

```
>pause_at
ending time > 2
>run
(STimer:Working,2.000,3.300) at 2
The current time has reached to the
   simulation ending time.
>pause_at 7
```

Set the simulation-termination time as 2.

Simulation run stops at 2.

Set the simulation-termination time as 7.

Case Study

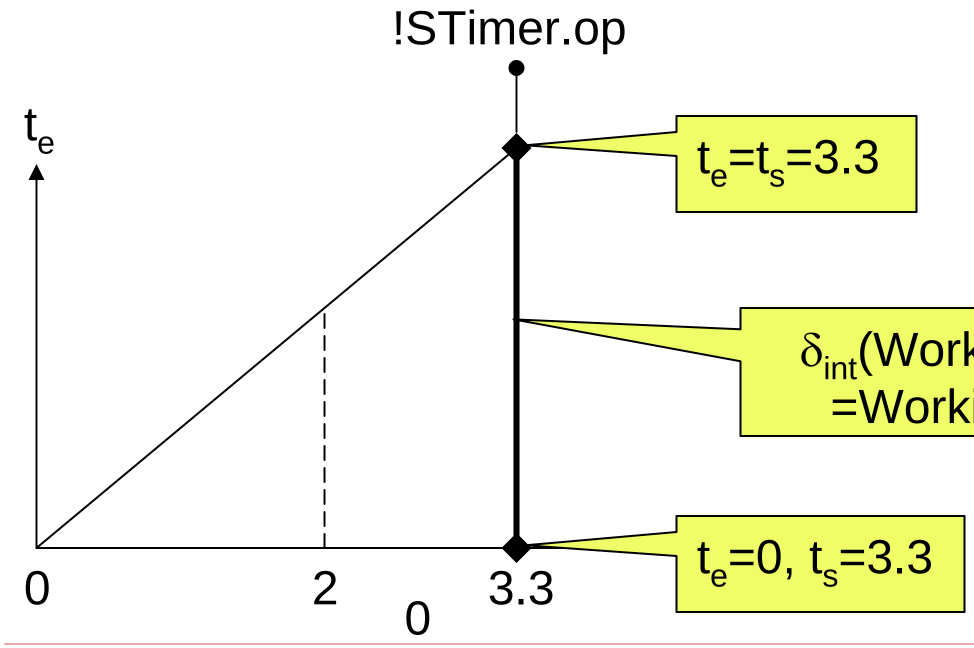
Case Study of Ex_Timer: Active Atomic DEVS

Tool:
ODEVS-C++

```
>step
(STimer:Working,3.300,3.300)
 --({!STimer.op},3.3)-->
(STimer:Working,0.000,3.300)
```

Triggering event i.e. lambda(Working);

At t=3.3



$t_e=t_s=3.3$

δ_{int}(Working) =Working

$t_e=0$, $t_s=3.3$

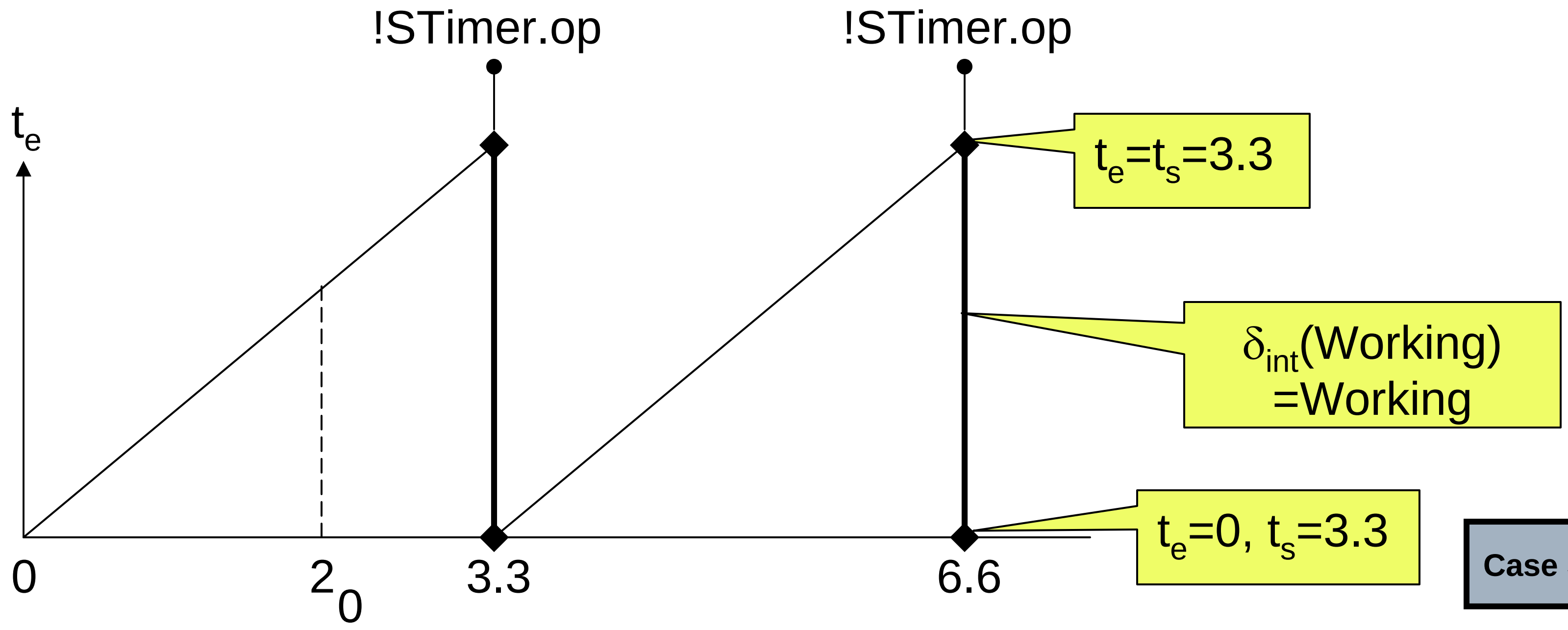
Case Study

Case Study of Ex_Timer: Active Atomic DEVS

Tool: ODEVS-C++

```
>step
(STimer:Working,3.300,3.300)
 --({!STimer.op},6.6)-->
(STimer:Working,0.000,3.300)
```

At t=6.6



$t_e=t_s=3.3$

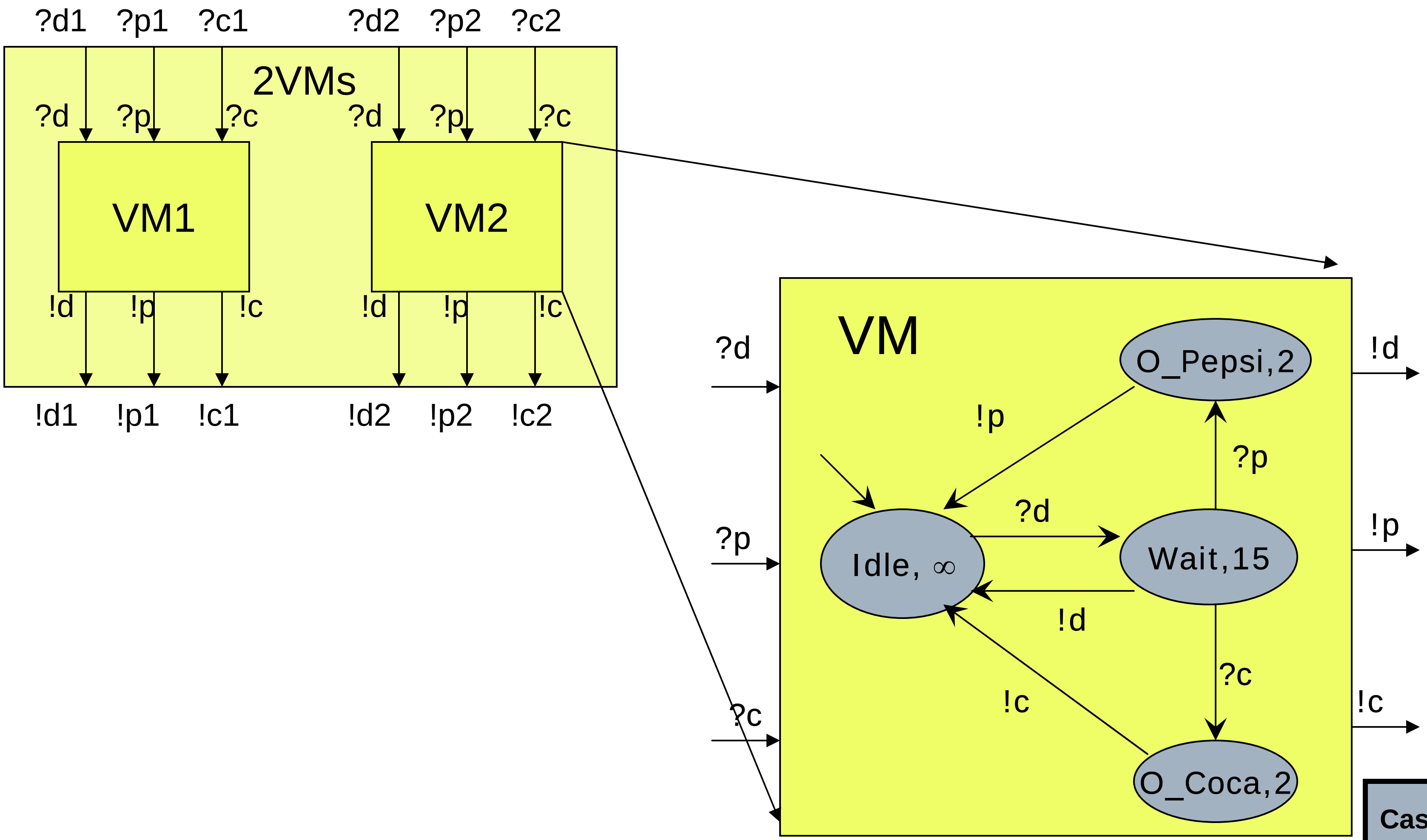
δ_{int}(Working) =Working

$t_e=0$, $t_s=3.3$

Case Study

Case Study of `Ex_2VMs`: Active Atomic DEVS

Tool: `ODEVS-C++`



Case Study

0

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

```
>print
options: q cpl > cpl
Inside of 2VMs
2VMs.c1-->VM1.coca_btn
2VMs.c2-->VM2.coca_btn
2VMs.d1-->VM1.dollar
2VMs.d2-->VM2.dollar
2VMs.p1-->VM1.pepsi_btn
2VMs.p2-->VM2.pepsi_btn
VM1.coca-->2VMs.c1
VM1.dollar-->2VMs.d1
VM1.pepsi-->2VMs.p1
VM2.coca-->2VMs.c2
VM2.dollar-->2VMs.d2
VM2.pepsi-->2VMs.p2
```

Print couplings

EIC of 2VMs

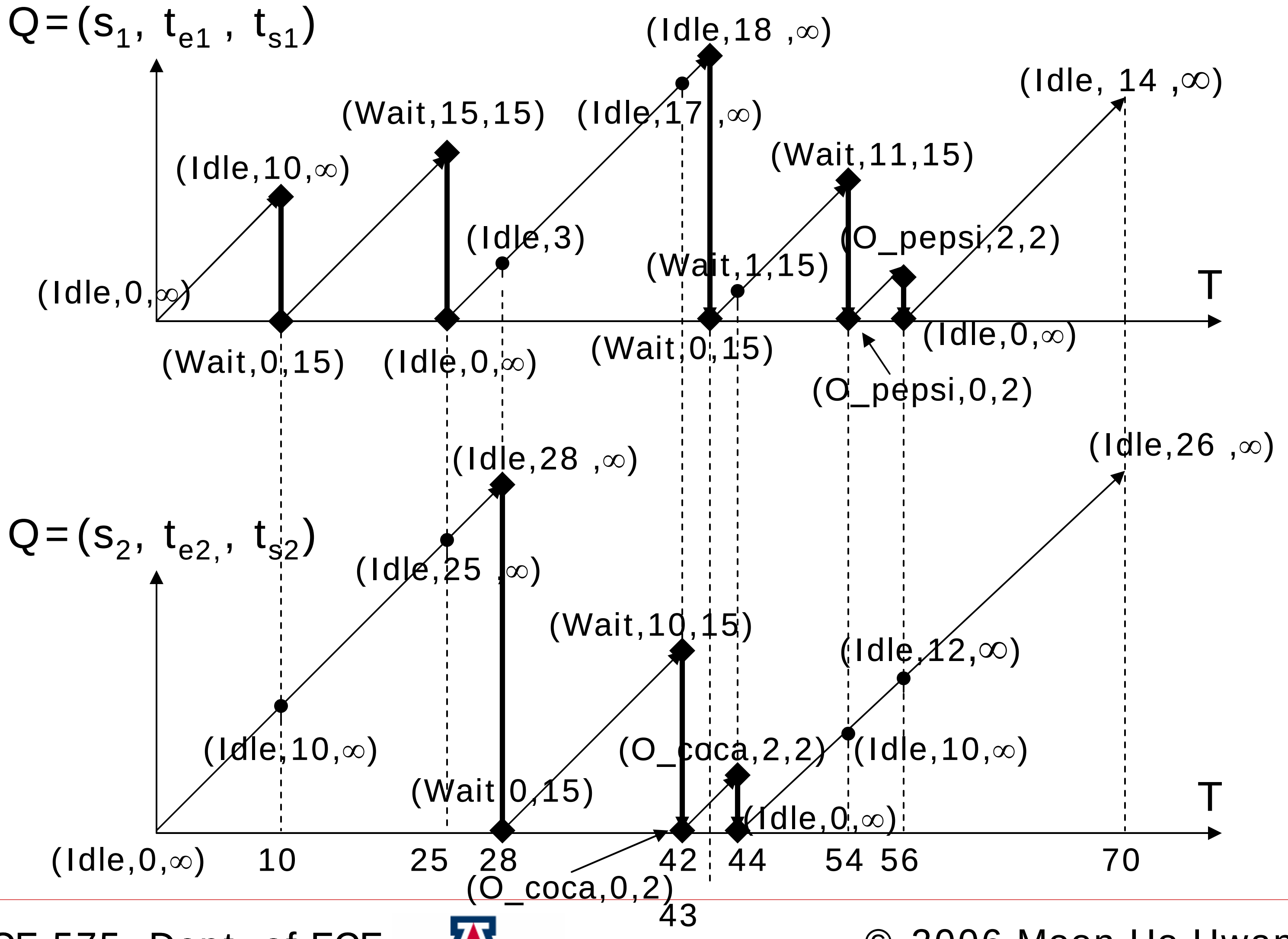
EOC of 2VMs

Case Study

Execution Run of

Tool:
ODEVS-C++

$\omega_{[0,70]}$ = (?d1,10)(!VM1.d,25)(?d2,28)(?c2,42)(?d1,43)(!VM2.c,44)(?p1.54)(!VM1.p,56)

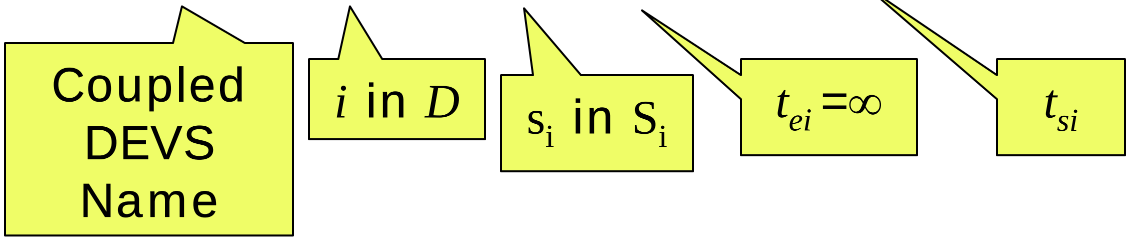


Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- ☐ Let’s check $\omega_{[0,70]}$ = (?d1,10)(!VM1.d,25)(?d2,28)(?c2,42)(?d1,43)(!VM2.c,44)(?p1.54)(!VM1.p, 56) $\in L(2VMs)$.
- ☐ To input ?d1 at 10, set ending time at 10 using `pause_at 10` and step until t=10.

```
>pause_at 10
>step
(2VMs:(VM1:Idle,10.000,inf),(VM2:Idle,10.000,inf)) at 10
```



Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let’s check $\omega_{[0,10]} = (?d1,10)$

```
>inject
[d1]ollar [p1]epsi_botton [c1]oca_botton [d2]ollar
    [p2]epsi_botton [c2]oca_botton> d1
(2VMs:(VM1:Idle,10.000,inf),(VM2:Idle,10.000,inf))
 --({?d1,?2VMs.VM1.dollar},10)-->
(2VMs:(VM1:Wait,0.000,15.000),(VM2:Idle,10.000,inf)
```

Triggering event

Synchronized Event by EIC

Printed in the user-defined callback function `InjectMsg`. See `Ex_TwoVMs.cpp`

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let’s check $\omega_{[0,25]}$ = (?d1,10)(!VM1.d,25)

```
>pause_at 28
>step
(2VMs:(VM1:Wait,15.000,15.000),(VM2:Idle,25.000,inf)
   )
 --({!2VMs.VM1.dollar},25)-->
(2VMs:(VM1:Idle,0.000,inf),(VM2:Idle,25.000,inf))
```

Next input time
(?d2,28)

Triggering
event

t=25

Case Study

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let's check $\omega_{[0,28]} = (?d1,10)(!VM1.d,25)\underline{(?d2,28)}$

```
>step
(2VMs:(VM1:Idle,3.000,inf),(VM2:Idle,28.000,inf)) at 28
The current time has reached to the simulation ending time.
>inject d2
(2VMs:(VM1:Idle,3.000,inf),(VM2:Idle,28.000,inf))
 --({?d2,?2VMs.VM2.dollar},28)-->
(2VMs:(VM1:Idle,3.000,inf),(VM2:Wait,0.000,15.000))
```

Input ?d2 which is transmitted to ?2VMs.VM2.dollar at t=28

Case Study

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let's check $\omega_{[0,42]} = (?d1,10)(!VM1.d,25)(?d2,28)\underline{(?c2,42)}$

```
>pause_at 42
>step
(2VMs:(VM1:Idle,17.000,inf),(VM2:Wait,14.000,15.000)) at 42
The current time has reached to the simulation ending time.
>inject c2
(2VMs:(VM1:Idle,17.000,inf),(VM2:Wait,14.000,15.000))
 --({?c2,??2VMs.VM2.coca_btn},42)-->
(2VMs:(VM1:Idle,17.000,inf),(VM2:O_coca,0.000,2.000))
```

Input ?c2 which is transmitted to ??2VMs.VM2.coca_btn at t=42

Case Study

Case Study of Ex_2VMs: Passive DEVS Network

Tool: ODEVS-C++

- ☐ Let's check $\omega_{[0,43]} =$ (?d1,10)(!VM1.d,25)(?d2,28)(?c2,42)$\underline{\text{(?d1,43)}}$

```
>pause_at 43
>step
(2VMs:(VM1:Idle,18.000,inf),(VM2:O_coca,1.000,2.000)) at 43
The current time has reached to the simulation ending time.
>inject d1
(2VMs:(VM1:Idle,18.000,inf),(VM2:O_coca,1.000,2.000))
 --({?d1,?2VMs.VM1.dollar},43)-->
(2VMs:(VM1:Wait,0.000,15.000 (VM2:O_coca,1.000,2.000))
```

Input ?d1 which is transmitted to ?2VMs.VM1.dollar at t=43

Case Study

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let's check $\omega_{[0,54]} =$ (?d1,10)(!VM1.d,25)(?d2,28)(?c2,42)(?d1,43)(!VM2.c,44)(?p1.54)

```
>pause_at 54
>step
(2VMs:(VM1:Wait,1.000,15.000),(VM2:O_coca,2.000,2.000))
 --({!2VMs.VM2.coca},44)-->
(2VMs:(VM1:Wait,1.000,15.000),(VM2:Idle,0.000,inf))
>step
(2VMs:(VM1:Wait,11.000,15.000),(VM2:Idle,10.000,inf)) at 54
The current time has reached to the simulation ending time.
>inject p1
(2VMs:(VM1:Wait,11.000,15.000),(VM2:Idle,10.000,inf))
 --({?p1,?2VMs.VM1.pepsi_btn},54)-->
(2VMs:(VM1:O_pepsi,0.000,2.000),(VM2:Idle,10.000,inf))
```

Next input time for (?p1,54)

lambda(s)=! 2VMs.VM2.coca at 44

Input ?p1 which is transmitted to ?2VMs.VM1.pepsi_btn at t=54

Case Study

Case Study of Ex_2VMs: Passive DEVS Network

Tool:
ODEVS-C++

- Let's check $\omega_{[0,70]}$ = (?d1,10)(!VM1.d,25)(?d2,28)(?c2,42)(?d1,43)(!VM2.c,44)(?p1.54)(!VM1.p,56)

```
>pause_at 70
>step
(2VMs:(VM1:O_pepsi,2.000,2.000),(VM2:Idle,12.000,inf))
 --({!2VMs.VM1.pepsi},56)-->
(2VMs:(VM1:Idle,0.000,inf),(VM2:Idle,12.000,inf))
>step
(2VMs:(VM1:Idle,14.000,inf),(VM2:Idle,26.000,inf)) at 70
The current time has reached to the simulation ending time.
```

lambda(s)=! 2VMs.VM1.pepsi at 56

Ending status of system at 70

- Thus, $\omega_{[0,70]}$ can be generated by ODEVS, $\omega_{[0,70]} \in L(2VMs)$.
- If you want to check faster than RT, you can use scale >1 such as scale 20.

Case Study

Discussion: Advantage and Disadvantage of each way

Tool:
ODEVS-C++

1. Method1
    - Advantage: Simple to Implement
    - Disadvantage: If |x:{value}| is large or infinite, couplings of them can be a burden or impossible.
2. Method2
    - Disadvantage: Difficult to Implement
    - Advantage: |{couplings of them}| is constant as |(x, y)| even if |x:{value}| is large or infinite,

Discussion

Discussion: Recommendation

1. If $|x{:}\{value\}| = |x{:}\{\}| = |x| = 1 \Rightarrow$ Use Method 1. (see `TwoVendingMachine` project)
2. If $|x{:}\{value\}| = |x{:}\{0, 1\}| = 2 \Rightarrow$ Use one of either Method 1 and Method 2.
3. If $|x{:}\{value\}| = |x{:}\{0, 1, \ldots, 999\}| = 1000$ or $|x{:}\{value\}| = \infty \Rightarrow$ Use Method 2. (see `ClientServer` project)

Discussion

Summary

1. Event = (Port, Value*)=PortValue
2. Atomic DEVS and Coupled DEVS
3. SRTEngine and its menu structure
4. Case Studies:
    1. Atomic DEVS (`Ex_Timer`)
    2. Coupled DEVS (`Ex_TwoVendingMachines`)

Appendix: Features of Examples

Tool: ODEVS-C++

- Ex_Timer
  - Model: Atomic DEVS
    - Six overriding functions
      - 4 characteristic functions of DEVS Formalism: `ta()`, `delta_ext()`, `delta_int()`, `lambda()`,
      - Initializing function: `init()`,
      - State-tracking function: `Get_s()`.
  - SRTEngine:
    - Basic Features such as Model plug-in, Console Menu

Summary

Appendix:
Features of Examples

- Ex_VendingMachine
  - Model: Atomic DEVS
    - `delta_ext()` shows to control update (or continue) schedule against an external input using `x_RescheduleMe()`.
  - SRTEngine:
    - Callback Function whose form is

```
PortValue Callback();
```

to define user input to model is illustrated.

    - The callback function can be passed as second arguments of SRTEngine constructor such as

```
SRTEngine simEngine(*vm, Callback);
```

Summary

Appendix:
Features of Examples

Tool:
`ODEVS-C++`

- Ex_TwoVendingMachine
    - Model: Coupled DEVS
        - Showing a DEVS network which as sets of external input couplings and external output couplings.
    - SRTEngine:
        - Same as `Ex_TwoVendingMachine` but the number of inputs are double.

Summary

Appendix: Features of Examples

Tool:
`ODEVS-C++`

- Ex_PingPong
  - Model: Coupled DEVS
    - Showing a single match game of ping pong.
- Ex_PingPongWithTable
  - Model: Coupled DEVS
    - Showing a single match game of ping pong with the model of table.
- Ex_DoublePingPong
  - Model: Coupled DEVS
    - Showing two matches of double game in which one team has two players, and the order to hit ball is changing from one to other in a team.
    - Recommending to print all couplings by `print cpl` command.

Summary

Appendix:
Features of Examples

Tool:
ODEVS-C++

- Ex_ClientServer
  - Model: Coupled DEVS
    - Consisting of two atomic DEVS classes such as Buffer and Server; one Value class for Client.
    - Showing how to derive a user-define class from Value class which is supposed to be transmitted through coupling.
  - SREngine:
    - A callback function shows to make a instance of the user-defined class, and inject to the simulator.

Summary