

---

# IrrigationController

## Report on Configuration MS\_VCPP

### Overridden Properties

#### Subjects:

CG

#### Metaclasses:

CGGeneral

#### Properties:

InitialLayoutAs2.3: False

CPP\_Roundtrip

#### Metaclasses:

General

#### Properties:

RoundtripScheme: Basic

CPP\_CG

#### Metaclasses:

General

#### Properties:

<<Friend>>ImplementationScheme: Rhapsody2.3

# PACKAGES

## HomeIrrigationController

Overridden Properties

Subjects:

CG

Metaclasses:

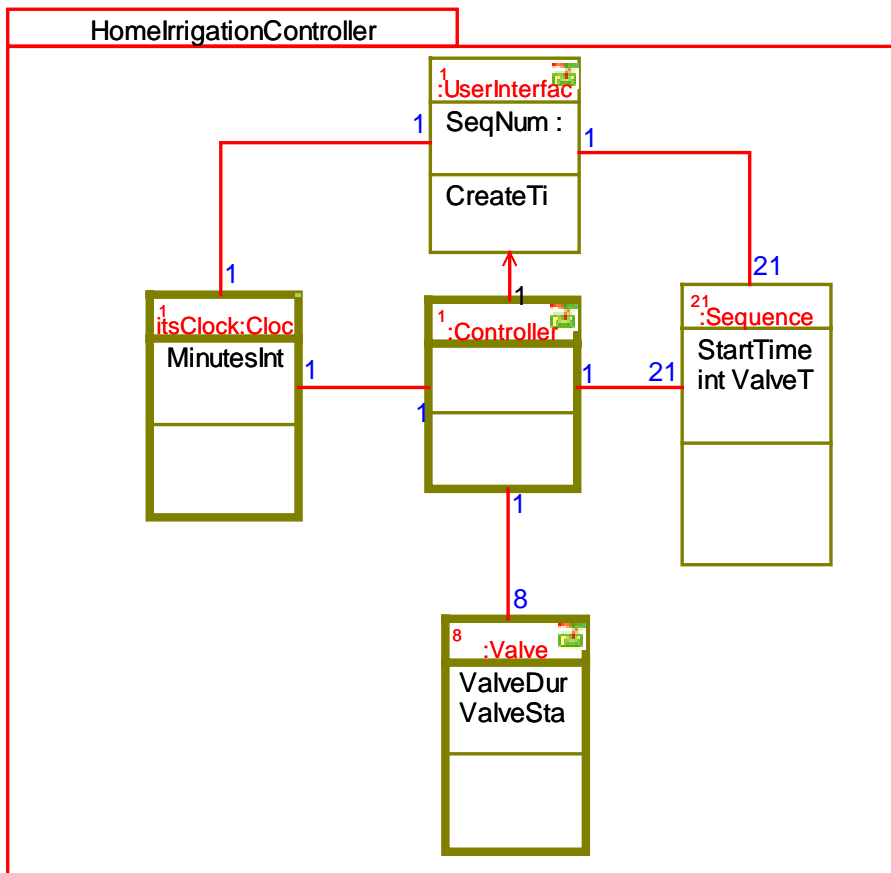
**Package**

Properties:

GenerateWithAggregates: True

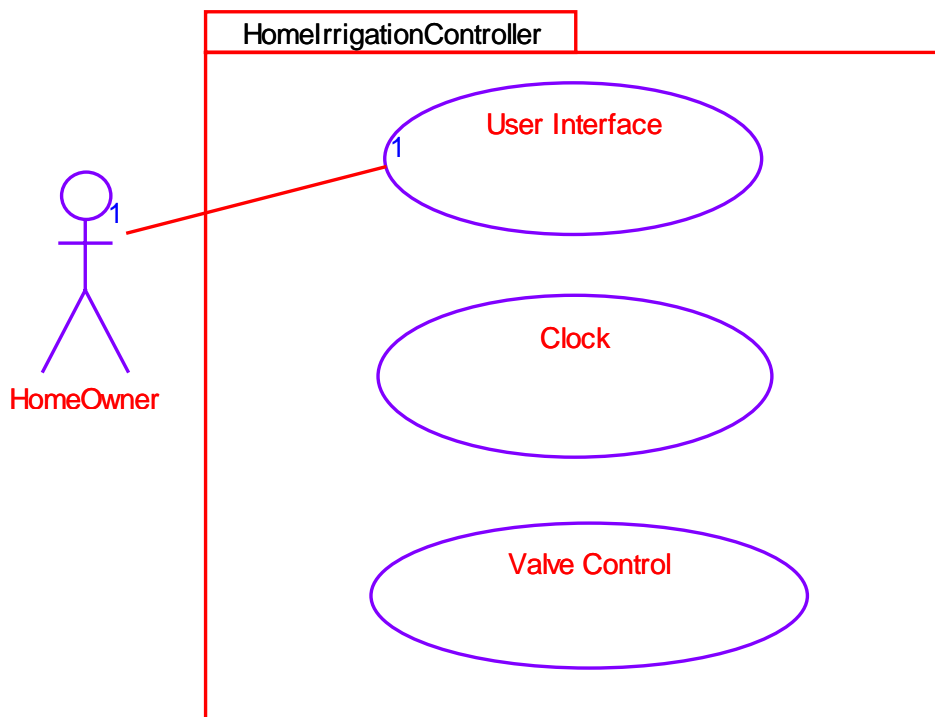
## OBJECT MODEL DIAGRAMS:

### IrrigationController



# USE CASE DIAGRAMS:

## IrrigationController



## EVENTS:

### evDoneSequence

*Done executing a sequence*

### evDoSequence

*Execute a sequence.*

### evMinuteElapsed

*event occuring every minute. It initiates sequence checking.*

### evValveDone

*Done with a valve. It is now off.*

### evValveON

*Open a valve.*

## GLOBALS:

### Relations:

#### itsUserInterface

Composition of UserInterface, Multiplicity of 1, Uni-directional

#### itsClock

Composition of ClockSchedule, Multiplicity of 1, Uni-directional

#### itsValve

Composition of Valve, Multiplicity of 8, Uni-directional

**itsSequence**

Composition of Sequence, Multiplicity of 21, Uni-directional

**itsController**

Composition of Controller, Multiplicity of 1, Uni-directional

**Instantiated Relations:****itsClockSchedule**

of itsController with itsClock

**itsController**

of itsClock with itsController

**itsSequence**

of itsController with itsSequence

**itsController**

of itsSequence with itsController

**itsValve**

of itsController with itsValve

**itsController**

of itsValve with itsController

**itsSequence**

of itsUserInterface with itsSequence

**itsUserInterface**

of itsSequence with itsUserInterface

**itsClockSchedule**

of itsUserInterface with itsClock

**itsUserInterface**

of itsClock with itsUserInterface

**itsUserInterface**

of itsController with itsUserInterface

**CLASSES:****ClockSchedule**

*This thread keeps track of time in the system.*

**Overridden Properties****Subjects:**

CG

**Metaclasses:**

Class

**Properties:**

Concurrency: active

**Relations:****itsUserInterface**

Association with UserInterface, Multiplicity of 1, Bi-directional

**itsController**

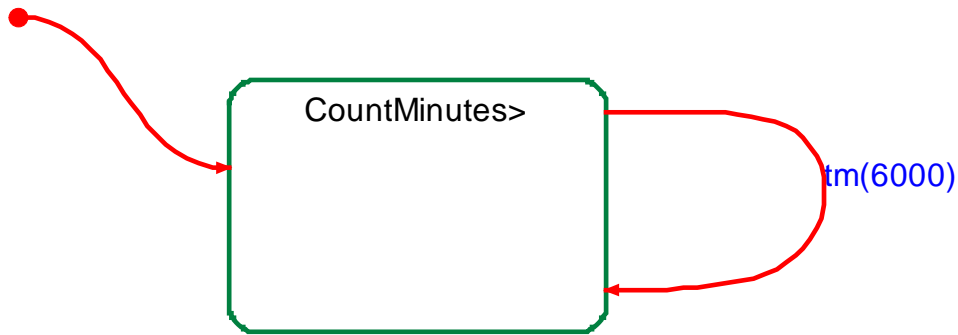
Association with Controller, Multiplicity of 1, Bi-directional

**Attributes:****MinutesIntoWeek**

*Keep track of the number of minutes that have elapsed for the current week.*

Type of int, Public, Static, Initial Value: 0

Statechart



## ROOT

Or-state

Substates:

CountMinutes

Default Transition

Target:

CountMinutes

## CountMinutes

*This keeps track of time, rolling it over at the end of a week.*

*It currently is in fast-time where 1 sec = 100 milliseconds.*

Or-state

EntryAction

```
if(++MinutesIntoWeek == 10080) MinutesIntoWeek=0;
itsController->GEN(evMinuteElapsed);
```

Out Transition

tm(6000)

Target:

CountMinutes

---

## Controller

*The object that checks to see if it is time to execute a sequence.*

*It also prints out the time and status of all valves.*

## Overridden Properties

Subjects:

CG

Metaclasses:

Class

Properties:

Concurrency: active

Relations:

**itsClockSchedule**

Association with ClockSchedule, Multiplicity of 1, Bi-directional

**itsValve**

Association with Valve, Multiplicity of 8, Bi-directional

**itsSequence**

Association with Sequence, Multiplicity of 21, Bi-directional

**itsUserInterface**

Association with UserInterface, Multiplicity of 1, Uni-directional

Operations:

**DisplayTime**

Primitive-operation, Public, Return type is void

Body

```
cout << "Day: " << dDay << " Hr:Mn " << dHour << ":" << dMinute << " ";
```

**evDoneSequence**

*Done executing a sequence*

Event

**evDoSequence**

*Execute a sequence.*

Event

**evMinuteElapsed**

*event occurring every minute. It initiates sequence checking.*

Event

**evValveDone**

*Done with a valve. It is now off.*

Event

**ExtractTime**

*Calculates the Day, Hour, and Minutes from the passed time.*

Primitive-operation , Public, Return type is void

Args:

int tminutes

*time to be converted*

Body

```
dDay = tminutes/1440;
dHour = (tminutes % 1440)/60;
dMinute = (tminutes % 1440) % 60;
```

**NewMinute**

Primitive-operation , Public, Return type is void

Args:

int tminutes

*Value of current minute of the week.*

Body

```
ExtractTime(tminutes);
DisplayTime();
ValveSummary();
cout << endl;
```

**ValveSummary**

Primitive-operation , Public, Return type is void

Body

```
for(int i=0; i<8; ++i)
{ if (itsValve[i]->getValveStatus()) cout << " ON ";
else cout << "OFF ";
}
```

Attributes:**dDay**

*Current Day for display interface.*

Type of int, Public

**dHour**

*Current Hour for display interface.*

Type of int, Public

**dMinute**

*Current Minute for display interface.*

Type of int, Public

**doValve**

*index of the valve to turn on/off*

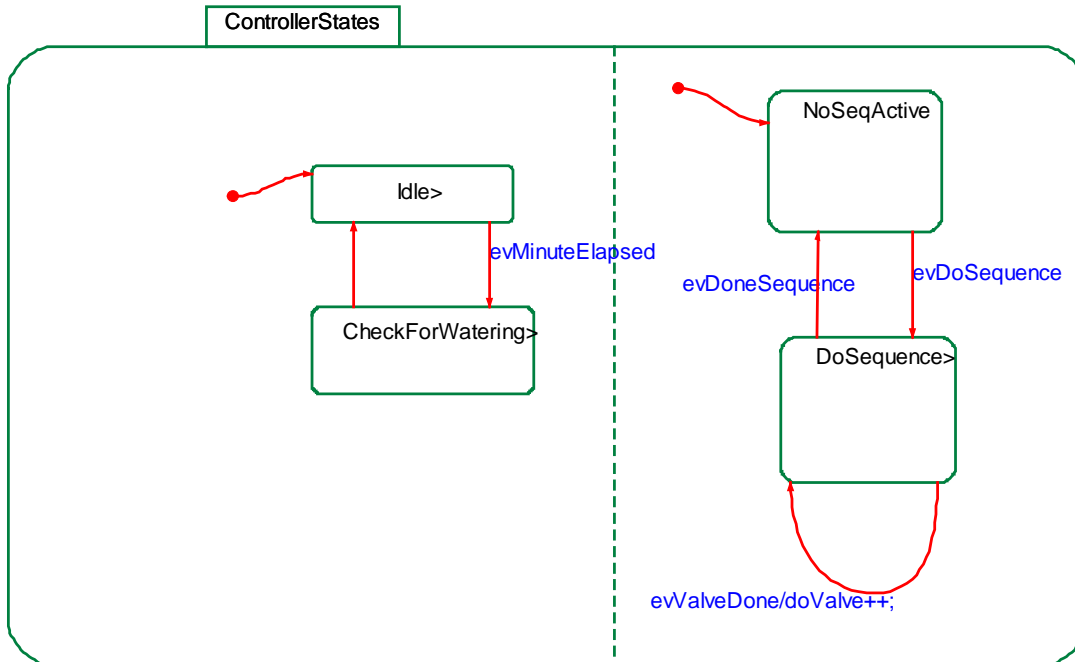
Type of int, Public

**trySeqNum**

*iteration counter to check sequences.*

Type of int, Public

# Statechart



## ROOT

Or-state

Substates:

ControllerStates

## ControllerStates

And-state

Substates:

state\_4

state\_5

## state\_4

Or-state

Substates:

CheckForWatering

Idle

Default Transition

Target:

Idle

## CheckForWatering

Or-state

EntryAction

```
for (int i =0; i < 21; ++i)
{
    if (itsSequence[i]->getStartTime() == itsClock->getMinutesIntoWeek())
    { //time to do a sequence
        trySeqNum = i;
        doValve=0;
        GEN(evDoSequence);
    }
}
```

Out Transition

Target:

Idle

## Idle

Or-state

### EntryAction

```
if( !itsUserInterface->getUIInProgress() )
{
    ExtractTime(itsClock->getMinutesIntoWeek());
    DisplayTime();
    ValveSummary();
    cout << (char)13 ;
    cout.flush();
}
```

### ExitAction

```
trySeqNum=0;
```

### Out Transition

```
evMinuteElapsed
```

### Target:

```
CheckForWatering
```

### state\_5

Or-state

### Substates:

DoSequence

NoSeqActive

Default Transition

### Target:

NoSeqActive

### DoSequence

Or-state

### EntryAction

```
if (doValve < 8)
{
    if( itsSequence[trySeqNum]->getValveTime(doValve) > 0 )
    {
        itsValve[doValve]->setValveDuration( (itsSequence[trySeqNum]-
>getValveTime(doValve))*6000);
        itsValve[doValve]->GEN(evValveON);
        cout <<endl << "Turning on valve# " << doValve << endl;
    }
    else
    {
        GEN(evValveDone);
    }
}
else
{
    GEN(evDoneSequence);
}
```

### Out Transition

```
evDoneSequence
```

### Target:

NoSeqActive

### Out Transition

```
evValveDone/doValve++;
```

### Target:

DoSequence

### NoSeqActive

Or-state

### Out Transition

```
evDoSequence
```

### Target:

DoSequence

---

## Sequence

Each sequence contains a start time for the sequence and a duration for each of the valves available to the controller.  
This is just a container class for each of the 21 sequences.  
There are not states of this class.

### Relations:

#### **itsUserInterface**

Association with UserInterface, Multiplicity of 1, Bi-directional

#### **itsController**

Association with Controller, Multiplicity of 1, Bi-directional

### Operations:

#### **getEndTime**

Primitive-operation , Public, Return type is int

#### Body

```
int temp=StartTime;
for(int i=0; i<8; ++i)
    temp += ValveTime[i];

return temp;
```

#### **printSequence**

Primitive-operation , Public, Return type is void

#### Body

```
cout << "Start Day: " << StartTime/1440
    << " Start Time: " << (StartTime % 1440)/60 << ":" << ((StartTime % 1440) % 60)
    << " Valve Times: "
    << ValveTime[0] << " "
    << ValveTime[1] << " "
    << ValveTime[2] << " "
    << ValveTime[3] << " "
    << ValveTime[4] << " "
    << ValveTime[5] << " "
    << ValveTime[6] << " "
    << ValveTime[7]
    << endl;
```

## Sequence

Constructor , Public

#### Initializers

```
StartTime(-1)
```

#### Body

```
for (int i=0; i < 8; i++)
    {ValveTime[i]=0;}
```

### Attributes:

#### **StartTime**

*The start time in minutes-into-the-week of this sequence.*

Type of int, Public

#### **ValveTime**

*Allow for up to 8 valves per sequence.*

Type of 'int %s[8];', Public

---

## UserInterface

*Allows the user to:*

- 1 - Set the current Time and Day of week.
- 2 - Create a new watering sequence.
- 3 - Summarize existing watering sequences.
- 4 - Delete a watering sequence.
- 5 - Exit Program.

### **Overridden Properties**

#### Subjects:

CG

#### Metaclasses:

Class

Properties:

Concurrency: sequential

Relations:

**itsClockSchedule**

Association with ClockSchedule, Multiplicity of 1, Bi-directional

**itsSequence**

Association with Sequence, Multiplicity of 21, Bi-directional

Operations:

**CreateTime**

*Returns the number of minutes since the beginning of the week.*

Primitive-operation , Public, Return type is int

Args:

int tD

*Day of the week*

int tH

*Hour of the day*

int tM

*Minute of the hour*

Body

```
return (tD*1440 + tH*60 + tM);
```

Attributes:

**num\_input**

*input from user on menu selection*

Type of int, Public

**SeqNum**

*Sequence Number currently being dealt with.*

Type of int, Public

**tDay**

*Temp Day*

Type of int, Public

**tHour**

*Temp hour*

Type of int, Public

**tMinute**

*Temp minute*

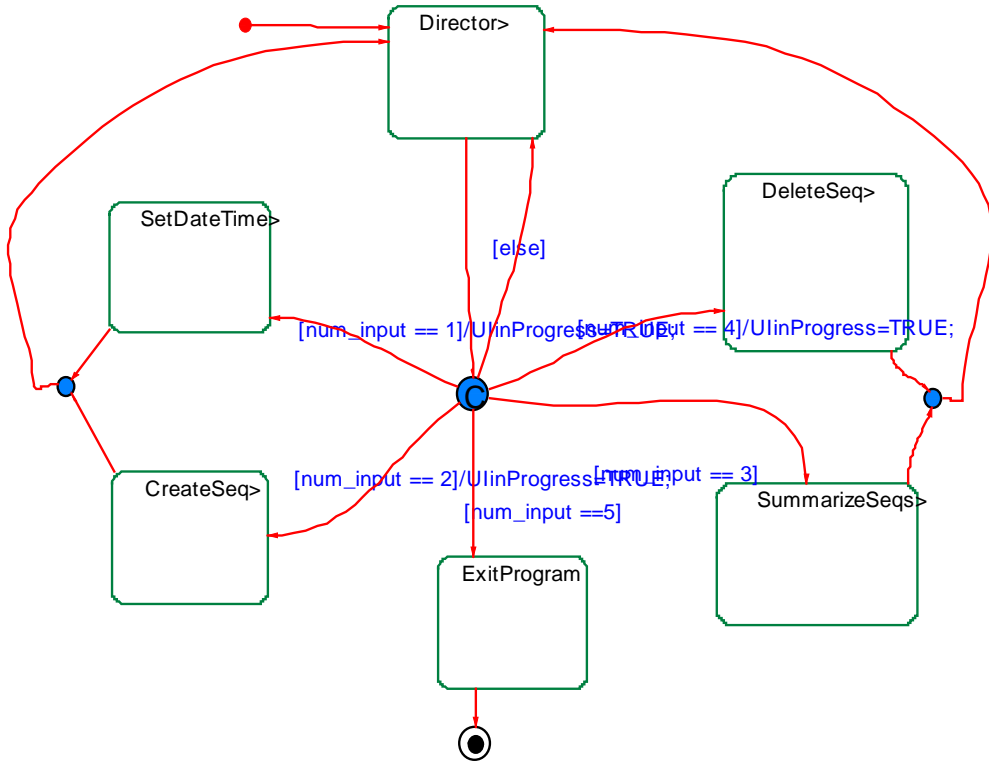
Type of int, Public

**UlinProgress**

*This indicates if user input is currently active.*

Type of OMBoolean, Public, Static, Initial Value: FALSE

Statechart



**ROOT**

Or-state

Substates:

- CreateSeq
- DeleteSeq
- Director
- ExitProgram
- SetDateTime
- state\_7
- SummarizeSeqs

Default Transition

Target:

Director

**CreateSeq**

Or-state

EntryAction

```

// find empty sequence;
SeqNum = -1;
for (int i=0; i<21; ++i)
{
    if (itsSequence[i]->getStartTime() < 0)
    {
        SeqNum = i;
        i=21; // end loop
    }
}

// is there an empty sequence?
if (SeqNum == -1)
{
    cout << "No new sequences available, you will have to delete an existing one first." << endl;
}
else
// an empty sequence is found!!!
{

```

```

cout << "You are now defining sequence # " << SeqNum << endl;
cout << "Enter Start Day: ";
cin >> tDay;
cout << "Enter Start Hour: ";
cin >> tHour;
cout << "Enter Start Minute: ";
cin >> tMinute;

// set sequence start time to be the number of minutes into the week.
itsSequence[SeqNum]->setStartTime(CreateTime(tDay, tHour, tMinute));

// get duration times for each valve
cout << endl;
cout << "Enter the duration times for each of the valves: ";
int vt;
for ( i=0; i<8; ++i)
{
cout << endl;
cout << "Valve #" << i << ":";
cin >> vt;
itsSequence[SeqNum]->setValveTime(i, vt);
}

// check for sequence overlap.
// if the starting time of this sequence is between the starting and ending times
// of any sequence, then we have an error.
// also if the ending time of this sequence is between the starting and ending times
// of any sequence, then we have an error!!!

// check start times
for ( i=0; i<21; ++i)
{
if( itsSequence[i]->getStartTime() > 0 )
{
if( (itsSequence[SeqNum]->getStartTime() > itsSequence[i]->getStartTime())
&& (itsSequence[SeqNum]->getStartTime() < itsSequence[i]->getEndTime()) )
{ cout << "Overlapping start times, try again." << endl;
itsSequence[SeqNum]->setStartTime(-1);
}
}
}

// check end times
for ( i=0; i<21; ++i)
{
if( itsSequence[i]->getStartTime() > 0 )
{
if( (itsSequence[SeqNum]->getEndTime() > itsSequence[i]->getStartTime())
&& (itsSequence[SeqNum]->getEndTime() < itsSequence[i]->getEndTime()) )
{ cout << "Overlapping start times, try again." << endl;
itsSequence[SeqNum]->setStartTime(-1);
}
}
}
}

```

### ExitAction

UIInProgress=FALSE;

### Out Transition

#### Target:

Director

### DeleteSeq

#### Or-state

#### EntryAction

```

num_input = -1;
while (num_input<0 || num_input>21)
{
cout << "Enter the number of the sequence to delete: ";
cin >> num_input;
}

```

```
itsSequence[num_input]->setStartTime(-1); // a start time of -1 signifies not used
;
```

#### ExitAction

```
UIinProgress=FALSE;
```

Out Transition

#### Target:

Director

#### **Director**

Or-state

#### EntryAction

```
cout << endl;
cout << "Welcome to the Home Irrigation Controller " << endl;
cout << "Please enter one of the following selections:" << endl;
cout << " 1 - Set the current Time and Day of week." << endl;
cout << " 2 - Create a new watering sequence." << endl;
cout << " 3 - Summarize existing watering sequences." << endl;
cout << " 4 - Delete a watering sequence." << endl;
cout << " 5 - Exit Program." << endl;
cin >> num_input;
if (cin)
    cout << "You selected " << num_input << endl;
else
    cout << "You entered an invalid entry, please try again." << endl;
```

Out Transition

Condition Connector

#### Branches:

```
[num_input == 1]/UIinProgress=TRUE;
```

Target:

SetDateTime

```
[num_input ==5]
```

Target:

ExitProgram

```
[else]
```

Target:

Director

```
[num_input == 2]/UIinProgress=TRUE;
```

Target:

CreateSeq

```
[num_input == 3]
```

Target:

SummarizeSeqs

```
[num_input == 4]/UIinProgress=TRUE;
```

Target:

DeleteSeq

#### **ExitProgram**

Or-state

Out Transition

#### Target:

state\_7

#### **SetDateTime**

Or-state

#### EntryAction

```
cout << "Enter the day of the week; 0=Sun, 1=Mon, ... 6=Sat: ";
cin >> tDay;
cout << endl << "Enter the current Hour: ";
cin >> tHour;
cout << endl << "Enter the current Minute: ";
cin >> tMinute;
```

```
itsClock->setMinutesIntoWeek (CreateTime(tDay, tHour, tMinute));
```

#### ExitAction

```
UIinProgress=FALSE;
```

Out Transition

#### Target:

Director

#### **state\_7**

Local Termination State

#### **SummarizeSeqs**

Or-state

#### EntryAction

```
for (int i=0; i<21; ++i)
{
if (itsSequence[i]->getStartTime() >= 0)
{
cout << "sequence# " << i << " " ;
itsSequence[i]->printSequence();
cout <<endl;
}
}
```

Out Transition

#### Target:

Director

---

### **Valve**

*Controls the turning on and off of a specific valve.*

*This is initiated by the controller class, and signals the controller when the valve is off.*

#### **Overridden Properties**

##### Subjects:

CG

##### Metaclasses:

Class

##### Properties:

Concurrency: active

##### Relations:

#### **itsController**

Association with Controller, Multiplicity of 1, Bi-directional

##### Operations:

#### **evValveON**

*Open a valve.*

Event

#### **Valve**

Constructor , Public

##### Initializers

```
ValveStatus(false)
```

##### Attributes:

#### **ValveDuration**

*This is the duration of this valve.*

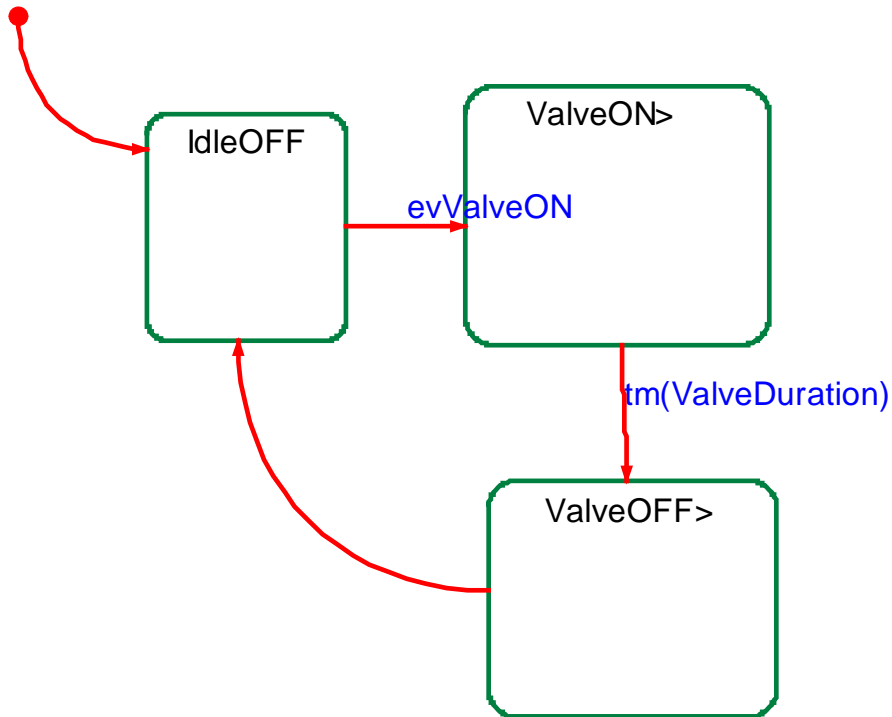
Type of int, Public

#### **ValveStatus**

*Indicate the current status of the valve. true ==> ON, false ==> OFF.*

Type of OMBBoolean, Public

# Statechart



## ROOT

Or-state

Substates:

IdleOFF

ValveOFF

ValveON

Default Transition

Target:

IdleOFF

## IdleOFF

Or-state

Out Transition

evValveON

Target:

ValveON

## ValveOFF

Or-state

EntryAction

```
ValveStatus = false;
cout << endl << "Turning off valve. " << endl;
itsController->GEN(evValveDone);
```

Out Transition

Target:

IdleOFF

## ValveON

Or-state

EntryAction

```
if(ValveDuration != 0)ValveStatus = true;
```

Out Transition

tm(ValveDuration)

Target:  
ValveOFF

## ACTORS:

---

### HomeOwner

Relations:

**itsUser Interface**

Association with User Interface, Multiplicity of 1, Bi-directional

## USE CASES:

---

### Clock

*Keep track of the current time into the week.  
This is a weekly clock, so it rolls over to zero at the end of a week.  
The days of the week are:  
0 = Sun  
1 = Mon  
...  
6 = Sat  
Time is kept in 0 - 24 hours.*

---

### User Interface

*Allow the Home owner to do one of the following.  
1 - Set the current Time and Day of week.  
2 - Create a new watering sequence.  
3 - Summarize existing watering sequences  
4 - Delete a watering sequence.*

*This performs various consistency checks, e.g.  
No overlapping sequences.  
Number of sequences defined.*

Relations:

**itsHomeOwner**

---

### Valve Control

*This is the heart of the valve control.  
There are 21 watering sequences allowed.  
Each watering sequence contains a start time and a set of valve ontime durations for each valve.  
Every minute the sequences are checked to determine if any of them should be activated.  
The time and valve statuses are displayed each minute.  
There can be up to 8 valves on this system.  
If a valve is not used, it should be assigned a 0 duration.*

# COMPONENTS

## MS\_VisualCPP

### COMPONENT SETTINGS:

Build type: Executable

### CONFIGURATIONS:

---

#### **Animated\_MS\_CPP**

Scope type: Explicit  
Instrumentation type: Animation  
Time-model type: Real-time  
Statechart generation type: Flat  
Standard headers: iostream.h

---

#### **MS\_VCPP**

Scope type: Explicit  
Instrumentation type: None  
Time-model type: Real-time  
Statechart generation type: Flat  
Standard headers: iostream.h

### FILES AND FOLDERS:

#### Explicit scope:

Package: HomeIrrigationController