Introducing DEVS for Collaborative Building Simulation Development

Rhys Goldstein and Azam Khan

Autodesk Research
Introducing **DEVS** for Collaborative Building Simulation Development

Rhys Goldstein and Azam Khan

Autodesk Research
Simulation Use

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Simulation Use

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Simulation Use

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Simulation Use

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS ➔ SIMULATION SOFTWARE ➔ SIMULATION RESULTS

MODEL PARAMETERS ➔ SIMULATION SOFTWARE ➔ SIMULATION RESULTS

MODEL PARAMETERS ➔ SIMULATION SOFTWARE ➔ SIMULATION RESULTS

MODEL PARAMETERS ➔ SIMULATION SOFTWARE ➔ SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
start_time = 4014
datetime = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
Outdoor Climate Simulation

```python
start_time = 4014
del_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```
Outdoor Climate Simulation

start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
Outdoor Climate Simulation

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

STATE
TRANSITION
Outdoor Climate Simulation

start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1

time = start_time
outdoor_temperature = weather_data[int(start_time)]
indoor_temperature = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0

while time < end_time:
    outdoor_transition_time = int(time) + 1
    rate = wall_rate
    target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature

    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))

    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        time = indoor_transition_time
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        outdoor_temperature = weather_data[int(time)]
        output(time, ["outdoor_temperature", outdoor_temperature])
        output(time, ["indoor_temperature", indoor_temperature])
Outdoor Climate Simulation → Indoor Climate Simulation

Temperature (°F)

Outdoor Temperature
Outdoor Climate Simulation → Indoor Climate Simulation

Temperature (°F)

Outdoor Temperature

Indoor Temperature
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1

start_time
outdoor_temperature = weather_data[int(start_time)]
indoor_temperature = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0

while time < end_time:
    outdoor_transition_time = int(time) + 1
    rate = wall_rate
    target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        time = indoor_transition_time
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        outdoor_temperature = weather_data[int(time)]
        output(time, ["outdoor_temperature", outdoor_temperature])
        output(time, ["indoor_temperature", indoor_temperature])
Indoor Climate Simulation → Heating System Simulation

```
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75

time = start_time
outdoor_temperature = weather_data[int(start_time)]
indoor_temperature = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0
heater_is_on = False

while time < end_time:
    outdoor_transition_time = int(time) + 1
    if heater_is_on:
        rate = wall_rate + heater_rate
        target_temperature = (wall_rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
    else:
        rate = wall_rate
        target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        if indoor_temperature >= lower_sensor_threshold:
            heater_is_on = True
        elif indoor_temperature <= upper_sensor_threshold:
            heater_is_on = False
        time = indoor_transition_time
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
    output(time, ["indoor_temperature", indoor_temperature])

```
Indoor Climate Simulation → Heating System Simulation

Temperature (°F)

- Outdoor Temperature
- Indoor Temperature
Indoor Climate Simulation → Heating System Simulation
Heating System Simulation → Window Opening Simulation

start_time = 4014
d_end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
time = start_time
outdoor_temperature = weather_data[int(start_time)]
indoor_temperature = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0
heater_is_on = False

while time < end_time:
    outdoor_transition_time = int(time) + 1
    if heater_is_on:
        rate = wall_rate + heater_rate
        target_temperature = (wall_rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
    else:
        rate = wall_rate
        target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        if indoor_temperature <= lower_sensor_threshold:
            heater_is_on = True
        elif indoor_temperature >= upper_sensor_threshold:
            heater_is_on = False
        time = indoor_transition_time
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        outdoor_temperature = weather_data[int(time)]
        output(time, ["outdoor_temperature", outdoor_temperature])
        output(time, ["indoor_temperature", indoor_temperature])
Heating System Simulation → Window Opening Simulation

```python
def weather_data(start_time):
    # Read weather data from file
    return weather_data

wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
lower_occupant_threshold = 72
upper_occupant_threshold = 76
window_rate = 0.4

start_time = 4014
end_time = 4306

wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
lower_occupant_threshold = 72
upper_occupant_threshold = 76
window_rate = 0.4

wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
lower_occupant_threshold = 72
upper_occupant_threshold = 76
window_rate = 0.4

while time < end_time:
    if window_is_open:
        rate = wall_rate + window_rate
    else:
        rate = wall_rate
    if heater_is_on:
        target_temperature = (rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
    else:
        target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        observed_temperature = indoor_temperature
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
    time = outdoor_transition_time
    while time < end_time:
        if window_is_open:
            envelope_rate = wall_rate + window_rate
        else:
            envelope_rate = wall_rate
        if heater_is_on:
            rate = envelope_rate + heater_rate
        else:
            rate = envelope_rate
        target_temperature = (envelope_rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
        dT = target_temperature - indoor_temperature
        if dT < 0:
            transition_dT = lower_transition_temperature - indoor_temperature
        else:
            transition_dT = upper_transition_temperature - indoor_temperature
        if abs(dT) <= abs(transition_dT):
            indoor_transition_time = infty
        else:
            indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
        if indoor_transition_time < outdoor_transition_time:
            if dT < 0:
                indoor_temperature = lower_transition_temperature
            else:
                indoor_temperature = upper_transition_temperature
            observed_temperature = indoor_temperature
        else:
            dt = outdoor_transition_time - time
            indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        output(time, ['outdoor_temperature', outdoor_temperature])
        output(time, ['indoor_temperature', indoor_temperature])
```

```python
```
```
Heating System Simulation → Window Opening Simulation

Graph showing temperature trends with lines for outdoor and indoor temperatures.
Heating System Simulation → Window Opening Simulation

Graph showing the comparison of Outdoor Temperature and Indoor Temperature.
Traditional Simulation Code

start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
lower_occupant_threshold = 72
upper_occupant_threshold = 76
window_rate = 0.4

time = start_time
outdoor_temperature = weather_data[int(start_time)]
indoor_temperature = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0
heater_is_on = False
window_is_open = False
observed_temperature = weather_data[int(start_time)]

while time < end_time:
    outdoor_transition_time = int(time) + 1
    if window_is_open:
        envelope_rate = wall_rate + window_rate
    else:
        envelope_rate = wall_rate
    if heater_is_on:
        rate = envelope_rate + heater_rate
        target_temperature = (envelope_rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
    else:
        rate = envelope_rate
        target_temperature = outdoor_temperature
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = infty
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        observed_temperature = indoor_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        if indoor_temperature <= lower_sensor_threshold:
            heater_is_on = True
        elif indoor_temperature >= upper_sensor_threshold:
            heater_is_on = False
        time = indoor_transition_time
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        outdoor_temperature = weather_data[int(time)]
        output(time, ['outdoor_temperature', outdoor_temperature])
        output(time, ['indoor_temperature', indoor_temperature])
    if observed_temperature >= max([outdoor_temperature, upper_occupant_threshold]):
        window_is_open = True
    if observed_temperature <= max([outdoor_temperature, lower_occupant_threshold]):
        window_is_open = False
Traditional Simulation Development

MODEL PARAMETERS \[\rightarrow\] SIMULATION SOFTWARE \[\rightarrow\] SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
Traditional Simulation Development

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS

MODEL PARAMETERS → SIMULATION SOFTWARE → SIMULATION RESULTS
DEVS Model Development
DEVS Model Development
DEVS Model Development

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS
DEVS Model Development

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS

MODEL PARAMETERS → MODEL → SIMULATOR → SIMULATION RESULTS
Outdoor Climate Model

OUTDOOR CLIMATE

outdoor temperature
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state
    def external_transition(state, elapsed_time, input_value):
        pass
    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]
    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
datetime = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]

    return [initialize, DEVS_model]
```
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

# Print the outdoor temperature at the start time
print(weather_data[int(start_time)])

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state
    
    def external_transition(state, elapsed_time, input_value):
        pass
    
    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]
    
    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time
    
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
```
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
```
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):

    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]

    return [initialize, DEVS_model]```
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return state, output_values

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]

    return [initialize, DEVS_model]
```
Outdoor Climate Model

Traditional Simulation Code:

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()

time = start_time
outdoor_temperature = weather_data[int(start_time)]

while time < end_time:
    time = time + 1
    outdoor_temperature = weather_data[int(time)]
    output(time, ["outdoor_temperature", outdoor_temperature])
```

DEVS Model Code:

```python
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = [["outdoor_temperature", outdoor_temperature]]
        return [state, output_values]

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]

    return [initialize, DEVS_model]
```
Outdoor Climate Model → Indoor Climate Model

OUTDOOR CLIMATE

outdoor temperature

52
Outdoor Climate Model → Indoor Climate Model
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state

    def external_transition(state, elapsed_time, input_value):
        pass

    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = [[”outdoor_temperature”, outdoor_temperature]]
        return [state, output_values]

    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time

    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def BUILDING_ENVELOPE(wall_rate):
    def initialize(initial_temperature):
        state_has_changed = False
        outdoor_temperature = initial_temperature
        state = [state_has_changed, outdoor_temperature]
        return state
    def external_transition(state, elapsed_time, input_value):
        state_has_changed = True
        [port, outdoor_temperature] = input_value
        state = [state_has_changed, outdoor_temperature]
        return state
    def internal_transition(state):
        [state_has_changed, outdoor_temperature] = state
        state_has_changed = False
        state = [state_has_changed, outdoor_temperature]
        output_values = ["outdoor_heat_transfer", [outdoor_temperature, wall_rate]]
        return [state, output_values]
    def time_advance(state):
        [state_has_changed, outdoor_temperature] = state
        if state_has_changed:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def initialize(initial_temperature, initial_rate):
    indoor_temperature = initial_temperature
    lower_transition_temperature = indoor_temperature - 1.0
    upper_transition_temperature = indoor_temperature + 1.0
    outdoor_temperature = initial_temperature
    envelope_rate = initial_rate
    rate = initial_rate
    target_temperature = initial_temperature
    dT = 0
    indoor_transition_remaining_time = infty
    outdoor_has_changed = False
    state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
             outdoor_temperature, envelope_rate, rate, target_temperature, dT,
             indoor_transition_remaining_time, outdoor_has_changed]
    return state

def external_transition(state, elapsed_time, input_value):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, rate, target_temperature, dT,
     indoor_transition_remaining_time, outdoor_has_changed] = state
    dt = elapsed_time
    indoor_temperature = target_temperature - dT * exp(-rate * dt)
    [port, message] = input_value
    if port == "outdoor_heat_transfer":
        [new_outdoor_temperature, new_envelope_rate] = message
        if new_envelope_rate == envelope_rate or not (new_outdoor_temperature == outdoor_temperature):
            outdoor_has_changed = True
            outdoor_temperature = new_outdoor_temperature
            envelope_rate = new_envelope_rate
            rate = envelope_rate
            target_temperature = outdoor_temperature
            dT = target_temperature - indoor_temperature
            if dT < 0:
                transition_dT = lower_transition_temperature - indoor_temperature
            else:
                transition_dT = upper_transition_temperature - indoor_temperature
            if abs(dT) <= abs(transition_dT):
                indoor_transition_remaining_time = infty
            else:
                indoor_transition_remaining_time = (1.0 / rate) * log(abs(dT) / (abs(dT) - abs(transition_dT)))
    state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
             outdoor_temperature, envelope_rate, rate, target_temperature, dT,
             indoor_transition_remaining_time, outdoor_has_changed]
    return state

def internal_transition(state):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, rate, target_temperature, dT,
     indoor_transition_remaining_time, outdoor_has_changed] = state
    if outdoor_has_changed:
        outdoor_has_changed = False
        output_values = ["indoor_temperature", indoor_temperature]
    else:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        dT = target_temperature - indoor_temperature
        if dT < 0:
            transition_dT = lower_transition_temperature - indoor_temperature
        else:
            transition_dT = upper_transition_temperature - indoor_temperature
        if abs(dT) <= abs(transition_dT):
            indoor_transition_remaining_time = infty
        else:
            indoor_transition_remaining_time = (1.0 / rate) * log(abs(dT) / (abs(dT) - abs(transition_dT)))
        output_values = ["indoor_temperature_transition", indoor_temperature]
    state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
             outdoor_temperature, envelope_rate, rate, target_temperature, dT,
             indoor_transition_remaining_time, outdoor_has_changed]
    return [state, output_values]

def time_advance(state):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, rate, target_temperature, dT,
     indoor_transition_remaining_time, outdoor_has_changed] = state
    if outdoor_has_changed:
        remaining_time = 0
    else:
        remaining_time = indoor_transition_remaining_time
    return remaining_time

DEVS_model = [external_transition, internal_transition, time_advance]
return [initialize, DEVS_model]
Indoor Climate Model → Heating System Model

- OUTDOOR CLIMATE
- BUILDING ENVELOPE
- INDOOR CLIMATE

Outdoor temperature → Building Envelope
Building Envelope → Indoor heat transfer
Indoor heat transfer → Indoor climate

Indoor temperature
Indoor Climate Model → Heating System Model

OUTDOOR CLIMATE

→

BUILDING ENVELOPE

→

INDOOR CLIMATE WITH HEATER

INDOOR CLIMATE

SENSOR

HEATER

OUTDOOR temperature

BUILDING heat transfer

INDOOR CLIMATE WITH HEATER

heater heat transfer

heater change

indoor temperature
def SENSOR(lower_sensor_threshold, upper_sensor_threshold):
    def initialize():
        heater_should_change = False
        heater_should_be_on = False
        state = [heater_should_change, heater_should_be_on]
        return state
    def external_transition(state, elapsed_time, input_value):
        [heater_should_change, heater_should_be_on] = state
        [port, indoor_temperature] = input_value
        if (not heater_should_be_on) and indoor_temperature <= lower_sensor_threshold:
            heater_should_change = True
            heater_should_be_on = True
        elif heater_should_be_on and indoor_temperature >= upper_sensor_threshold:
            heater_should_change = True
            heater_should_be_on = False
        state = [heater_should_change, heater_should_be_on]
        return state
    def internal_transition(state):
        [heater_should_change, heater_should_be_on] = state
        heater_should_change = False
        state = [heater_should_change, heater_should_be_on]
        output_values = [['heater_change', heater_should_be_on]]
        return [state, output_values]
    def time_advance(state):
        [heater_should_change, heater_should_be_on] = state
        if heater_should_change:
            remaining_time = 0
        else:
            remaining_time = inf
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def HEATER(heater_temperature, heater_rate):
    def initialize():
        heater_has_changed = False
        heater_is_on = False
        state = [heater_has_changed, heater_is_on]
        return state
    def external_transition(state, elapsed_time, input_value):
        [heater_has_changed, heater_is_on] = state
        heater_has_changed = True
        [port, heater_is_on] = input_value
        state = [heater_has_changed, heater_is_on]
        return state
    def internal_transition(state):
        [heater_has_changed, heater_is_on] = state
        heater_has_changed = False
        state = [heater_has_changed, heater_is_on]
        if heater_is_on:
            rate = heater_rate
        else:
            rate = 0
        output_values = ["heater_heat_transfer", [heater_temperature, rate]]
        return [state, output_values]
    def time_advance(state):
        [heater_has_changed, heater_is_on] = state
        if heater_has_changed:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
**Indoor Climate Model → Heating System Model**

**OUTDOOR CLIMATE**

**outdoor temperature**

**BUILDING ENVELOPE**

**outdoor heat transfer**

**INDOOR CLIMATE WITH HEATER**

**heater heat transfer**

**HEATER**

```python
def initialize(initial_temperature, initial_rate):
    indoor_temperature = initial_temperature
    lower_transition_temperature = indoor_temperature - 1.0
    upper_transition_temperature = indoor_temperature + 1.0
    outdoor_temperature = initial_temperature
    envelope_rate = initial_rate
    heater_temperature = initial_temperature
    heater_rate = 0
    rate = initial_rate
    target_temperature = initial_temperature
    dT = 0
    indoor_transition_remaining_time = infty
    outdoor_has_changed = False
    state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
             outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed]
    return state

def external_transition(state, elapsed_time, input_value):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed] = state
    dt = elapsed_time
    indoor_temperature = target_temperature - dT*exp(-rate*dt)
    [port, message] = input_value
    if port == "outdoor_heat_transfer":
        [new_outdoor_temperature, new_envelope_rate] = message
        if new_envelope_rate == envelope_rate or not (new_outdoor_temperature == outdoor_temperature):
            outdoor_has_changed = True
            outdoor_temperature = new_outdoor_temperature
            envelope_rate = new_envelope_rate
        else:
            pass
    elif port == "heater_heat_transfer":
        [heater_temperature, heater_rate] = message
        rate = envelope_rate + heater_rate
        target_temperature = (envelope_rate*outdoor_temperature + heater_rate*heater_temperature)/float(rate)
        dT = target_temperature - indoor_temperature
        if dT < 0:
            transition_dT = lower_transition_temperature - indoor_temperature
        else:
            transition_dT = upper_transition_temperature - indoor_temperature
        if abs(dT) <= abs(transition_dT):
            indoor_transition_remaining_time = infty
        else:
            indoor_transition_remaining_time = (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
            state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
                     outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed]
            return state

def internal_transition(state):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed] = state
    if outdoor_has_changed:
        outdoor_has_changed = False
        output_values = ["indoor_temperature", indoor_temperature]
    else:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        lower_transition_temperature = indoor_temperature - 1.0
        upper_transition_temperature = indoor_temperature + 1.0
        dT = target_temperature - indoor_temperature
        if dT < 0:
            transition_dT = lower_transition_temperature - indoor_temperature
        else:
            transition_dT = upper_transition_temperature - indoor_temperature
        if abs(dT) <= abs(transition_dT):
            indoor_transition_remaining_time = infty
        else:
            indoor_transition_remaining_time = (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
            output_values = ["indoor_temperature_transition", indoor_temperature]
            state = [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
                     outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed]
            return state, output_values

def time_advance(state):
    [indoor_temperature, lower_transition_temperature, upper_transition_temperature,
     outdoor_temperature, envelope_rate, heater_temperature, heater_rate, rate, target_temperature, dT, indoor_transition_remaining_time, outdoor_has_changed] = state
    if outdoor_has_changed:
        remaining_time = 0
    else:
        remaining_time = indoor_transition_remaining_time
    return remaining_time
```

**DEVS_model**

```python
[initialize, DEVS_model]
```
DEVS Simulation Output

Temperature (F)

Outdoor Temperature

Indoor Temperature
Heating System Model → Window Opening Model

- OUTDOOR CLIMATE
- BUILDING ENVELOPE
  - outdoor temperature
  - outdoor heat transfer
- INDOOR CLIMATE WITH HEATER
  - indoor temperature
- SENSOR
  - indoor temperature
  - heater change
- HEATER
  - heater heat transfer
Heating System Model → Window Opening Model

OUTDOOR CLIMATE → BUILDING ENVELOPE WITH WINDOW

outdoor temperature → outdoor heat transfer

BUILDING ENVELOPE WITH WINDOW → INDOOR CLIMATE WITH HEATER

window change → indoor temperature

INDOOR CLIMATE WITH HEATER → SENSOR

indoor temperature → heater heat transfer

SENSOR → HEATER

heater change → HEATER
def OCCUPANT(lower_sensor_threshold, upper_sensor_threshold):
    def initialize(initial_temperature):
        window_should_change = False
        window_should_be_open = False
        outdoor_temperature = initial_temperature
        indoor_temperature = initial_temperature
        state = [window_should_change, window_should_be_open,
                 outdoor_temperature, indoor_temperature]
        return state
    def external_transition(state, elapsed_time, input_value):
        [window_should_change, window_should_be_open,
         outdoor_temperature, indoor_temperature] = state
        [port, input_temperature] = input_value
        if port == "outdoor_temperature":
            outdoor_temperature = input_temperature
        elif port == "indoor_temperature_transition":
            indoor_temperature = input_temperature
        if (not window_should_be_open) and indoor_temperature >= max([outdoor_temperature, upper_sensor_threshold]):
            window_should_change = True
            window_should_be_open = True
        elif window_should_be_open and indoor_temperature <= max([outdoor_temperature, lower_sensor_threshold]):
            window_should_change = True
            window_should_be_open = False
        state = [window_should_change, window_should_be_open,
                 outdoor_temperature, indoor_temperature]
        return state
    def internal_transition(state):
        [window_should_change, window_should_be_open,
         outdoor_temperature, indoor_temperature] = state
        window_should_change = False
        state = [window_should_change, window_should_be_open,
                 outdoor_temperature, indoor_temperature]
        output_values = ["window_change", window_should_be_open]
        return [state, output_values]
    def time_advance(state):
        [window_should_change, window_should_be_open,
         outdoor_temperature, indoor_temperature] = state
        if window_should_change:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def BUILDING_ENVELOPE_WITH_WINDOW(wall_rate, window_rate):
    def initialize(initial_temperature):
        state_has_changed = False
        outdoor_temperature = initial_temperature
        window_is_open = False
        state = [state_has_changed, outdoor_temperature, window_is_open]
        return state
    def external_transition(state, elapsed_time, input_value):
        state_has_changed = True
        if port == "outdoor_temperature":
            outdoor_temperature = message
        elif port == "window_change":
            window_is_open = message
        state = [state_has_changed, outdoor_temperature, window_is_open]
        return state
    def internal_transition(state):
        state_has_changed = False
        state = [state_has_changed, outdoor_temperature, window_is_open]
        if window_is_open:
            envelope_rate = wall_rate + window_rate
        else:
            envelope_rate = wall_rate
        output_values = [["outdoor_heat_transfer", [outdoor_temperature, envelope_rate]]]
        return [state, output_values]
    def time_advance(state):
        state_has_changed = False
        state = [state_has_changed, outdoor_temperature, window_is_open]
        if state_has_changed:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
DEVS Simulation Output

Graph showing temperature variations over time for outdoor and indoor environments.

- **Outdoor Temperature** (blue line)
- **Indoor Temperature** (red line)

Temperature (°F) vs. Time (days or whatever time units are used in the graph).
Traditional Simulation Code

```python
start_time = 4014
end_time = 4306
weather_data = read_weather_from_file()
wall_rate = 0.1
heater_rate = 0.4
heater_temperature = 100
lower_sensor_threshold = 70
upper_sensor_threshold = 75
lower_occupant_threshold = 72
upper_occupant_threshold = 76
window_rate = 0.4

start_time = weather_data[int(start_time)]
end_time = weather_data[int(start_time)]
lower_transition_temperature = indoor_temperature - 1.0
upper_transition_temperature = indoor_temperature + 1.0

for time in range(start_time, end_time):
    outdoor_temperature = weather_data[int(time)]
    indoor_temperature = weather_data[int(time)]
    lower_transition_temperature = indoor_temperature - 1.0
    upper_transition_temperature = indoor_temperature + 1.0
    indoor_transition_time = float('inf')
    dT = target_temperature - indoor_temperature
    if dT < 0:
        transition_dT = lower_transition_temperature - indoor_temperature
    else:
        transition_dT = upper_transition_temperature - indoor_temperature
    if abs(dT) <= abs(transition_dT):
        indoor_transition_time = float('inf')
    else:
        indoor_transition_time = time + (1.0/rate)*log(abs(dT)/(abs(dT) - abs(transition_dT)))
    if indoor_transition_time < outdoor_transition_time:
        if dT < 0:
            indoor_temperature = lower_transition_temperature
        else:
            indoor_temperature = upper_transition_temperature
        observed_temperature = indoor_temperature
        if indoor_temperature <= lower_sensor_threshold:
            heater_is_on = True
        elif indoor_temperature >= upper_sensor_threshold:
            heater_is_on = False
    else:
        dt = outdoor_transition_time - time
        indoor_temperature = target_temperature - dT*exp(-rate*dt)
        time = outdoor_transition_time
        outdoor_temperature = weather_data[int(time)]
        output(time, ['outdoor_temperature', outdoor_temperature])
        output(time, ['indoor_temperature', indoor_temperature])
```
DEVS Model Code

OUTDOOR CLIMATE

BUILDING ENVELOPE WITH WINDOW

INDOOR CLIMATE WITH HEATER

OCCUPANT

SENSOR

HEATER

outdoor temperature

outdoor heat transfer

indoor temperature

heater heat transfer

window change

heater change
def OUTDOOR_CLIMATE(start_time, end_time, weather_data):
    def initialize():
        time = start_time
        outdoor_temperature = weather_data[int(start_time)]
        state = [time, outdoor_temperature]
        return state
    def external_transition(state, elapsed_time, input_value):
        pass
    def internal_transition(state):
        [time, outdoor_temperature] = state
        time = time + 1
        outdoor_temperature = weather_data[int(time)]
        state = [time, outdoor_temperature]
        output_values = ["outdoor_temperature", outdoor_temperature]
        return [state, output_values]
    def time_advance(state):
        [time, outdoor_temperature] = state
        if time < end_time:
            remaining_time = 1
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def BUILDING_ENVELOPE_WITH_WINDOW(wall_rate, window_rate):
    def initialize(initial_temperature):
        state_has_changed = False
        outdoor_temperature = initial_temperature
        window_is_open = False
        state = [state_has_changed, outdoor_temperature, window_is_open]
        return state
    def external_transition(state, elapsed_time, input_value):
        [state_has_changed, outdoor_temperature, window_is_open] = state
        [port, message] = input_value
        state_has_changed = True
        if port == "outdoor_temperature":
            outdoor_temperature = message
        elif port == "window_change":
            window_is_open = message
        state = [state_has_changed, outdoor_temperature, window_is_open]
        return state
    def internal_transition(state):
        [state_has_changed, outdoor_temperature, window_is_open] = state
        state_has_changed = False
        state = [state_has_changed, outdoor_temperature, window_is_open]
        if window_is_open:
            envelope_rate = wall_rate + window_rate
        else:
            envelope_rate = wall_rate
        output_values = ["outdoor_heat_transfer", [outdoor_temperature, envelope_rate]]
        return [state, output_values]
    def time_advance(state):
        [state_has_changed, outdoor_temperature, window_is_open] = state
        if state_has_changed:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
def initialize(initial_temperature):
    window_should_change = False
    window_should_be_open = False
    outdoor_temperature = initial_temperature
    indoor_temperature = initial_temperature
    state = [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature]
    return state

def external_transition(state, elapsed_time, input_value):
    [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature] = state
    [port, input_temperature] = input_value
    if port == "outdoor_temperature":
        outdoor_temperature = input_temperature
    elif port == "indoor_temperature_transition":
        indoor_temperature = input_temperature
    if (not window_should_be_open) and indoor_temperature >= max([outdoor_temperature, upper_sensor_threshold]):
        window_should_change = True
        window_should_be_open = True
    elif window_should_be_open and indoor_temperature <= max([outdoor_temperature, lower_sensor_threshold]):
        window_should_change = True
        window_should_be_open = False
    state = [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature]
    return state

def internal_transition(state):
    [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature] = state
    window_should_change = False
    state = [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature]
    output_values = ["window_change", window_should_be_open]
    return [state, output_values]

def time_advance(state):
    [window_should_change, window_should_be_open, outdoor_temperature, indoor_temperature] = state
    if window_should_change:
        remaining_time = 0
    else:
        remaining_time = infty
    return remaining_time

DEVS_model = [initialize, internal_transition, time_advance]
DEVS Model Code

def SENSOR(lower_sensor_threshold, upper_sensor_threshold):

    def initialize():
        heater_should_change = False
        heater_should_be_on = False
        state = [heater_should_change, heater_should_be_on]
        return state

    def external_transition(state, elapsed_time, input_value):
        [heater_should_change, heater_should_be_on] = state
        [port, indoor_temperature] = input_value
        if (not heater_should_be_on) and indoor_temperature <= lower_sensor_threshold:
            heater_should_change = True
            heater_should_be_on = True
        elif heater_should_be_on and indoor_temperature >= upper_sensor_threshold:
            heater_should_change = True
            heater_should_be_on = False
        state = [heater_should_change, heater_should_be_on]
        return state

    def internal_transition(state):
        [heater_should_change, heater_should_be_on] = state
        heater_should_change = False
        state = [heater_should_change, heater_should_be_on]
        output_values = ["heater_change", heater_should_be_on]
        return [state, output_values]

    def time_advance(state):
        [heater_should_change, heater_should_be_on] = state
        if heater_should_change:
            remaining_time = 0
        else:
            remaining_time = infty
        return remaining_time

DEVS_model = [external_transition, internal_transition, time_advance]

return [initialize, DEVS_model]
def HEATER(heater_temperature, heater_rate):
    def initialize():
        heater_has_changed = False
        heater_is_on = False
        state = [heater_has_changed, heater_is_on]
        return state
    def external_transition(state, elapsed_time, input_value):
        [heater_has_changed, heater_is_on] = state
        heater_has_changed = True
        [port, heater_is_on] = input_value
        state = [heater_has_changed, heater_is_on]
        return state
    def internal_transition(state):
        [heater_has_changed, heater_is_on] = state
        heater_has_changed = False
        state = [heater_has_changed, heater_is_on]
        if heater_is_on:
            rate = heater_rate
        else:
            rate = 0
        output_values = ["heater_heat_transfer", [heater_temperature, rate]]
        return [state, output_values]
    def time_advance(state):
        [heater_has_changed, heater_is_on] = state
        if heater_has_changed:
            remaining_time = 0
        else:
            remaining_time = inf
        return remaining_time
    DEVS_model = [external_transition, internal_transition, time_advance]
    return [initialize, DEVS_model]
Introducing DEVS for Collaborative Building Simulation Development

Rhys Goldstein and Azam Khan

Autodesk Research