DEVS Modeling and Simulation

SYSC 5104 / SYSC 4906G

Smart Fridge Temperature Control System

Maryam Bazyari
February 2026

1. Overview

This project develops a hierarchical Discrete-Event System Specification (DEVS) model for a smart fridge temperature control system. The objective is to regulate the fridge temperature within an acceptable operating range using a hysteresis-based controller that issues ON/OFF commands to a compressor. Temperature evolves through event-driven updates generated by a periodic temperature sensor and compressor-status feedback.

The model is implemented in C++ using the Cadmium DEVS simulator and validated through atomic-level, coupled-subsystem, and full-system test cases. The implementation demonstrates closed-loop sensing-control-actuation behavior, hysteresis-based switching, and bounded temperature oscillation.

2. Conceptual Model

2.1 Problem Description

A refrigerator must maintain its internal temperature within a safe operating range for food storage. In practice, fridge temperature increases when cooling is OFF and decreases when cooling is ON. A controller must decide when to activate or deactivate cooling while avoiding rapid switching (chattering).

The goal of this project is to model a simplified smart fridge control loop as a DEVS system in which state changes occur at discrete event times driven by sensor outputs and control/actuator feedback.

2.2 Modeling Objectives

The main modeling objectives are:

- Regulate fridge temperature within a desired operating band.
- Use hysteresis control to reduce switching chatter.
- Model temperature evolution as event-driven updates (without continuous ODEs).
- Provide a clear hierarchical DEVS structure (atomic → coupled → top coupled) compatible with Cadmium.
- Support incremental testing from atomic models to full-system integration.
- Produce execution traces that can be analyzed to validate DEVS behavior.

2.3 Hierarchical Structure

The system is organized in three levels:

- **Top coupled model:** SmartFridgeSystem (SFS)
- **Coupled subsystem:** FridgeControlService (FCS)
- **Atomic models:** TempSensor (TS), Controller (CTRL), Compressor (COMP)

This decomposition supports modular design, easier debugging, and incremental verification.

2.4 Component Behavior (Informal)

TempSensor (TS)

- Stores the current temperature estimate.
- Periodically outputs temperature readings.
- Temperature increases when cooling is OFF.
- Temperature decreases when cooling is ON.
- Can optionally receive a disturbance input (e.g., door-open effect).

Controller (CTRL)

- Receives temperature readings from TS.
- Applies hysteresis logic with two thresholds:
  - If temperature is above the upper threshold, command ON.
  - If temperature is below the lower threshold, command OFF.

    - Otherwise, no command is sent (hold region).
- Can optionally generate an alarm if temperature exceeds a safety threshold.
- May suppress redundant repeated commands.

Compressor (COMP)

- Receives ON/OFF commands from CTRL.
- Updates cooling mode accordingly.
- Outputs cooling status (COOLING_ON / COOLING_OFF) to TS as feedback.

2.5 Assumptions and Scope

- Temperature is represented in °C.
- Thermal behavior is simplified as fixed increments per sensor sample.
- Sensor readings are ideal (no measurement noise or delay beyond DEVS event timing).
- Compressor is modeled as a two-state actuator (ON/OFF only).
- The model is event-driven and discrete-time in behavior (through periodic sensor events), not continuous-time.
- Disturbance and alarm behaviors are optional extensions and may be included as placeholders in the baseline implementation.

3. Formal DEVS Specification

Atomic models follow the classic DEVS form:

M = ⟨X, S, Y, δ ext, δ int,λ , ta⟩

Coupled models follow the DEVS coupled structure:

N = ⟨X, Y, D, {Mi}, EIC, EOC, IC, Select⟩

3.1 Atomic Models

3.1.1 TempSensor (TS)

Purpose
Maintains an internal estimate of fridge temperature and periodically emits temperature

readings. It reacts to compressor status feedback (cooling ON/OFF) and optionally disturbance inputs.

Inputs (X)

- `cooling_status_in : CoolingStatus`
- `disturbance_in : Disturbance` (optional)

Outputs (Y)

- `temp_out : Temperature`

State (S)

- T (real): current temperature
- `cooling_on` (boolean): whether cooling is currently active
- `pending_disturbance` (real or flag): optional disturbance effect
- timing/phase state for periodic behavior

Time advance (ta)

- Periodic sampling interval Δ `ts` (e.g., 1 time unit)

External transition (δ ext)

- On `cooling_status_in`:
    - set `cooling_on = true` if `COOLING_ON`
    - set `cooling_on = false` if `COOLING_OFF`
- On `disturbance_in`:
    - store or accumulate disturbance effect (optional)

Internal transition (δ int)

- If `cooling_on = false` → `T := T + rwarm`
- If `cooling_on = true` → `T := T - rcool`
- Apply pending disturbance if any
- Schedule next periodic output after Δ `ts`

Output function (λ)

- Output `temp_out = T` at each scheduled internal event

3.1.2 Controller (CTRL)

Purpose
Applies hysteresis control to maintain fridge temperature within a band and optionally produces an alarm output.

Inputs (X)

- `temp_in : Temperature`

Outputs (Y)

- `cmd_out : CompressorCommand` (`CMD_ON / CMD_OFF`)
- `alarm_out : Alarm` (optional)

State (S)

- `last_temp` (real)
- `last_cmd` (`CMD_ON / CMD_OFF / unknown`)
- `command_pending` (bool)
- `pending_cmd` (`CMD_ON / CMD_OFF`)
- `alarm_pending` (bool)
- `pending_alarm` (optional)
- phase/mode (`passive` or `output`)
- σ (time advance state)

Time advance (ta)

- `ta = 0` when an output is pending
- `ta = ∞` otherwise (passive/reactive behavior)

External transition (δ ext)

- Receive `temp_in`
- Update `last_temp`
- Evaluate hysteresis:
  - If `T > Thigh` and `last_cmd != CMD_ON` → `pending_cmd = CMD_ON`
  - If `T < Tlow` and `last_cmd != CMD_OFF` → `pending_cmd = CMD_OFF`

    - Else no command
- Optionally evaluate alarm threshold
- If any output is pending → schedule immediate output (`ta = 0`)

Internal transition (δ int)

- Finalize command memory (update `last_cmd` if a command was emitted)
- Clear pending outputs
- Return to passive mode (`ta = ∞`)

Output function (λ)

- Emit `cmd_out` if `command_pending = true`
- Emit `alarm_out` if `alarm_pending = true`

3.1.3 Compressor (COMP)

Purpose
Models the compressor actuator. It receives ON/OFF commands and emits cooling status feedback.

Inputs (X)

- `cmd_in : CompressorCommand` (`CMD_ON / CMD_OFF`)

Outputs (Y)

- `status_out : CoolingStatus` (`COOLING_ON / COOLING_OFF`)

State (S)

- `mode ∈ {COOLING_ON, COOLING_OFF}`
- `output_pending` (bool)
- phase/mode (`passive` or `output`)
- σ (time advance state)

Time advance (ta)

- `ta = 0` when `output_pending = true`

- `ta = ∞` otherwise

External transition (δ ext)

- On `cmd_in = CMD_ON`:
  - `mode := COOLING_ON`
  - `output_pending := true`
- On `cmd_in = CMD_OFF`:
  - `mode := COOLING_OFF`
  - `output_pending := true`
- Schedule immediate output (`ta = 0`)

Internal transition (δ int)

- `output_pending := false`
- Return to passive mode (`ta = ∞`)

Output function (λ)

- Emit `status_out = mode` when output is pending

3.2 Transition Pseudocode

3.2.1 TempSensor Pseudocode

```
On internal transition every sample_time:
    if cooling_on == true:
        T := T - cooling_rate
    else:
        T := T + warming_rate

    if pending_disturbance exists:
        T := T + pending_disturbance
        clear/update pending_disturbance

    schedule next internal transition after sample_time

Output function:
    emit temp_out(T)
```

```
On external transition:
    if cooling_status_in received:
        set cooling_on based on status
    if disturbance_in received:
        store disturbance effect
```

3.2.2 Controller Pseudocode

```
On external transition with temp_in(T):
    last_temp := T
    command_pending := false
    alarm_pending := false

    if T > T_high:
        if last_cmd != CMD_ON:
            pending_cmd := CMD_ON
            command_pending := true
    else if T < T_low:
        if last_cmd != CMD_OFF:
            pending_cmd := CMD_OFF
            command_pending := true

    if alarm_enabled and T > T_safe:
        pending_alarm := UNSAFE
        alarm_pending := true

    if command_pending or alarm_pending:
        sigma := 0
    else:
        sigma := INF

Output function:
    if command_pending: emit cmd_out(pending_cmd)
    if alarm_pending: emit alarm_out(pending_alarm)

On internal transition:
    if command_pending: last_cmd := pending_cmd
    clear pending outputs
    sigma := INF
```

3.2.3 Compressor Pseudocode

```
On external transition with cmd_in(c):
    if c == CMD_ON:
        mode := COOLING_ON
    else if c == CMD_OFF:
        mode := COOLING_OFF

    output_pending := true
    sigma := 0

Output function:
    if output_pending:
        emit status_out(mode)

On internal transition:
    output_pending := false
    sigma := INF
```

3.3 Formal Coupled Model Definitions

3.3.1 Coupled Model: FridgeControlService (FCS)

FridgeControlService encapsulates the smart fridge control loop (sensor, controller, and compressor).

Definition
FridgeControlService = ⟨X, Y, D, {Mi}, EIC, EOC, IC, Select⟩

X (external inputs)

- `disturbance_in` (optional)

Y (external outputs)

- `alarm_out` (optional)
- `temp_out` (recommended for observation)
- `status_out` (recommended for observation)

D (components)

- `{TempSensor, Controller, Compressor}`

Atomic models included

- `M_TempSensor`, `M_Controller`, `M_Compressor`

EIC (external input couplings)

- `(disturbance_in → TempSensor.disturbance_in)` [optional]

IC (internal couplings)

- `(TempSensor.temp_out → Controller.temp_in)`
- `(Controller.cmd_out → Compressor.cmd_in)`
- `(Compressor.status_out → TempSensor.cooling_status_in)`

EOC (external output couplings)

- `(Controller.alarm_out → alarm_out)` [optional]
- `(TempSensor.temp_out → temp_out)` [recommended]
- `(Compressor.status_out → status_out)` [recommended]

Select

- A fixed priority may be used if needed by the simulator; otherwise, default Cadmium behavior is acceptable.

3.3.2 Top Coupled Model: SmartFridgeSystem (SFS)

SmartFridgeSystem is the top-level coupled model used to run integrated experiments.

Definition
SmartFridgeSystem = ⟨X, Y, D, {Mi}, EIC, EOC, IC, Select⟩

X (external inputs)

- `disturbance_in` (optional top-level disturbance event)

Y (external outputs)

- `alarm_out` (optional)
- `temp_out` (recommended)
- `status_out` (recommended)

D (components)

- `{FridgeControlService}`

Coupled model included

- `M_FridgeControlService`

EIC (external input couplings)

- `(disturbance_in → FridgeControlService.disturbance_in)` [optional]

IC (internal couplings)

- none (single subsystem at top level)

EOC (external output couplings)

- `(FridgeControlService.alarm_out → alarm_out)` [optional]
- `(FridgeControlService.temp_out → temp_out)` [recommended]
- `(FridgeControlService.status_out → status_out)` [recommended]

Select

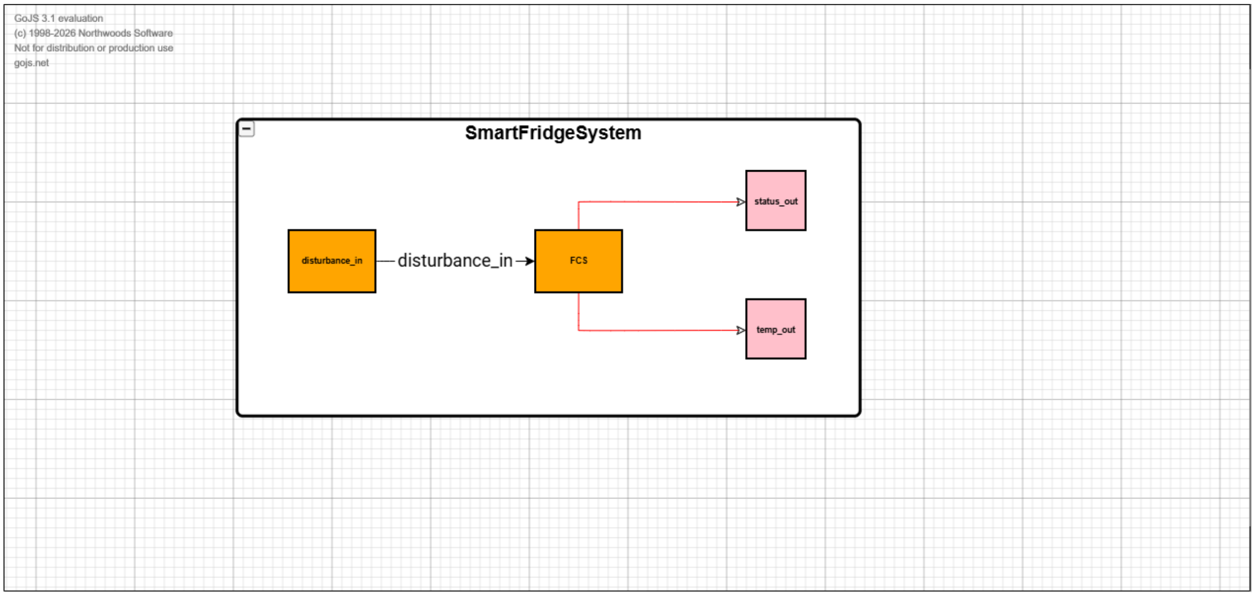
- Default simulator tie-breaking policy is acceptable for the baseline system.

4. DEVS-Graph Diagrams

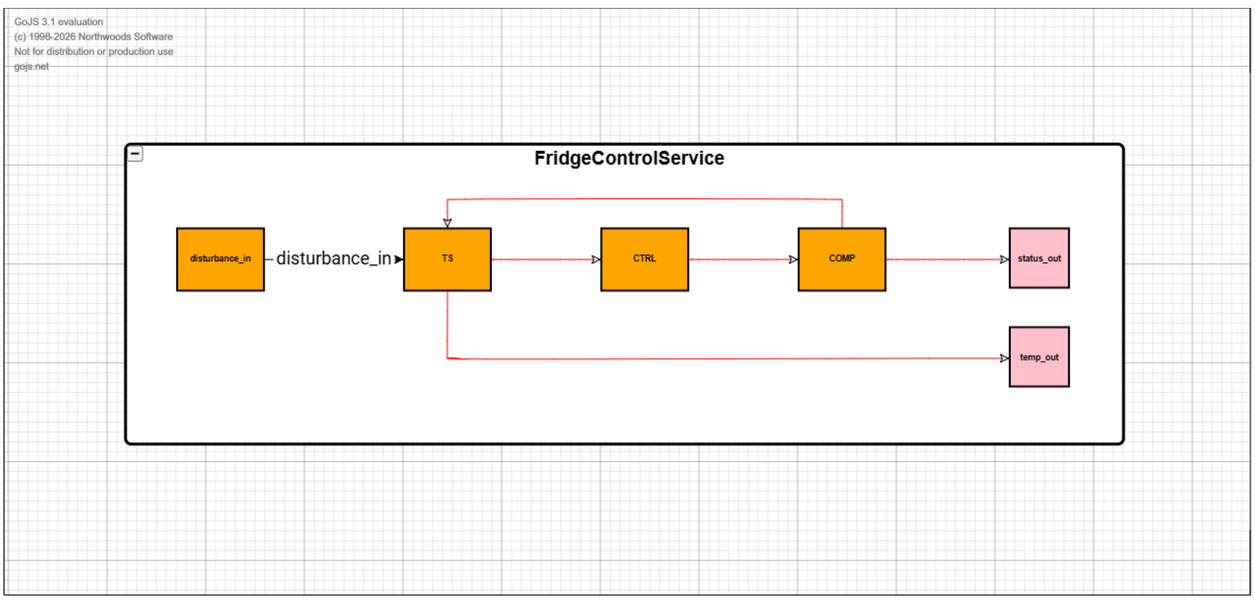
The following DEVS-Graph diagrams are included to document the hierarchical structure and state-based logic of the model.

4.1 Included Figures

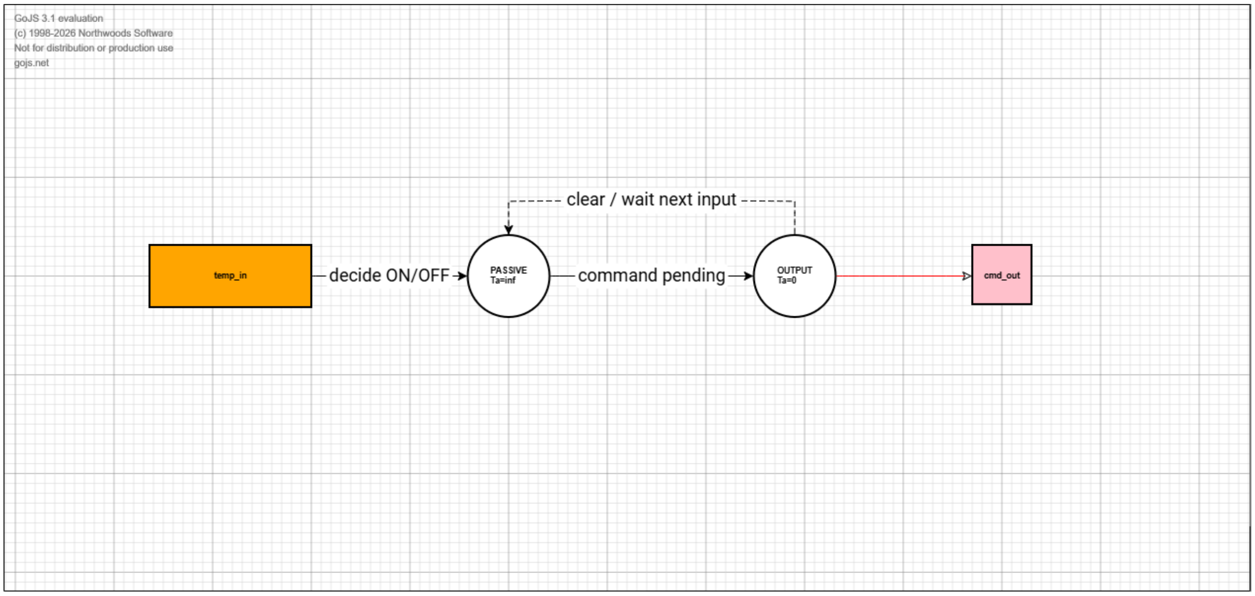
- **Figure 1:** SmartFridgeSystem (top coupled model)



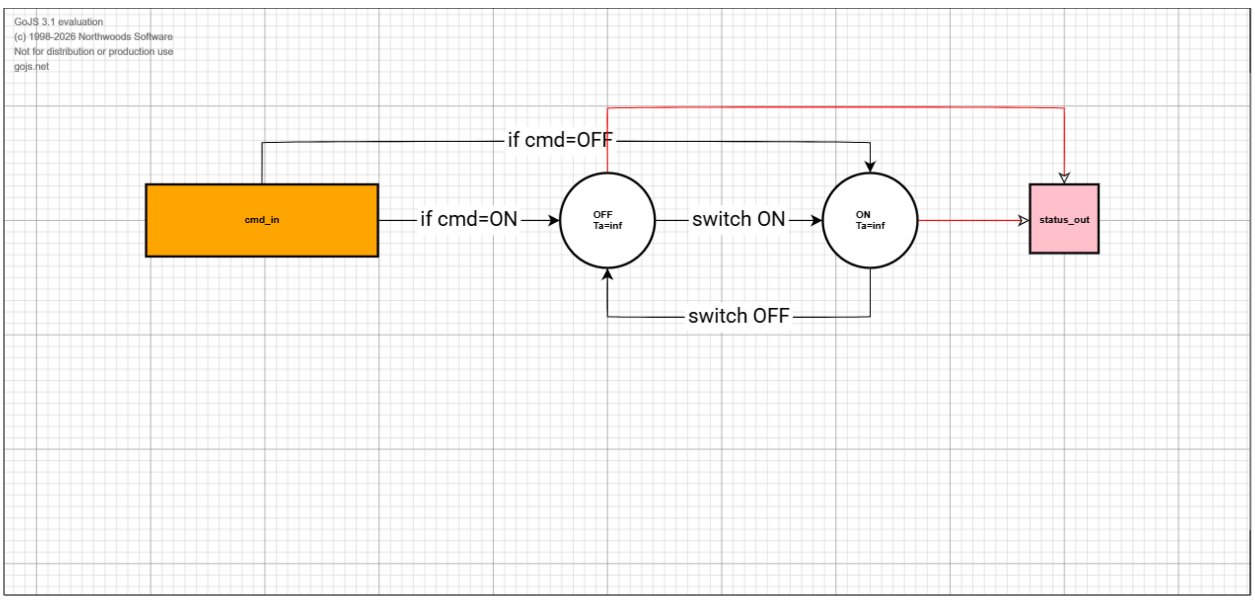
- **Figure 2:** FridgeControlService (coupled subsystem)



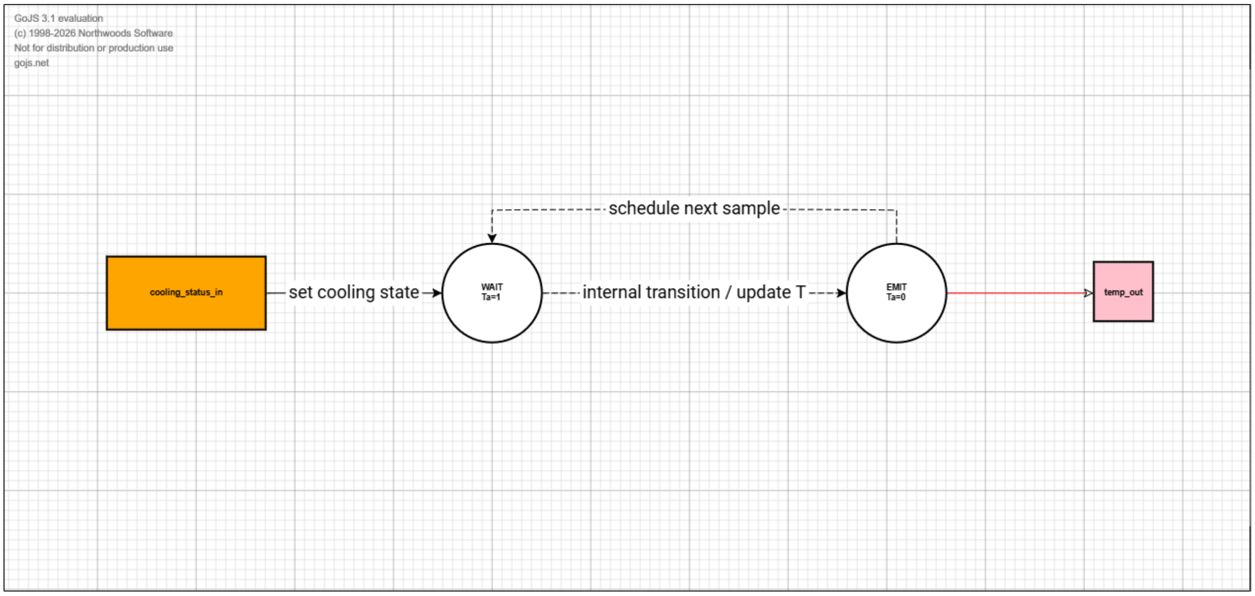
- **Figure 3:** Controller (CTRL) atomic/state-machine diagram



- **Figure 4:** Compressor (COMP) atomic/state-machine diagram



- **Figure 5:** TempSensor (TS) atomic/state-machine diagram



4.2 Relation to the Formal Model

The coupled-model diagrams correspond to the formal coupled definitions in Section 3.3, while the atomic diagrams represent the state-based realization of the transition logic and pseudocode described in Sections 3.1 and 3.2.

5. Implementation in Cadmium

Each atomic model was implemented in C++ as a Cadmium dynamic atomic model using the standard DEVS functions:

- `internal_transition`
- `external_transition`
- `confluence_transition`
- `output`
- `time_advance`

The coupled models were then composed using Cadmium dynamic coupled definitions consistent with the couplings listed in Section 3.3.

5.1 Implemented Files (Core)

- `message_types.hpp`

- `TempSensor.hpp`
- `Controller.hpp`
- `Compressor.hpp`
- `main.cpp`
- `CMakeLists.txt`

5.2 Implementation Notes

The implementation uses explicit state structs with flags such as:

- `command_pending`
- `alarm_pending`
- `output_pending`
- `has_last_cmd`

Zero-time outputs are implemented through pending-output flags and `time_advance = 0` where needed, especially in `Controller` and `Compressor`.

`NDTime` is used as the simulation time type, and the code was adjusted during debugging to ensure compatibility with Cadmium/DESTimes.

6. Test Cases and Validation

This section presents reproducible test cases, expected behavior, and validation outcomes for atomic and integrated model execution.

6.1 Atomic-Level Test Cases

6.1.1 TempSensor Periodic Warming (TS-01)

Input/Setup
Initialize `TempSensor` with cooling OFF and temperature `T0`. Run for multiple sample intervals with no cooling-status change.

Expected Behavior
At each sampling event, `TempSensor` emits `temp_out`, and temperature increases by the warming increment.

Validation Criterion
The execution trace shows periodic `temp_out` messages and monotonic temperature increase while `cooling_on = false`.

6.1.2 TempSensor Cooling Response (TS-02)

Input/Setup
Send `cooling_status_in = COOLING_ON` to `TempSensor` and continue simulation.

Expected Behavior
Subsequent periodic outputs show decreasing temperature.

Validation Criterion
The execution trace shows temperature decreasing at each sample while `cooling_on = true`.

6.1.3 Controller Hysteresis ON/OFF Decisions (CTRL-01 / CTRL-02 / CTRL-03)

Input/Setup
Provide representative temperature inputs to `Controller`:

- above upper threshold,
- within hysteresis band,
- below lower threshold.

Expected Behavior

- Above upper threshold → `CMD_ON`
- Within hysteresis band → no command
- Below lower threshold → `CMD_OFF`

Validation Criterion
Commands are generated only at threshold crossings, with no repeated switching inside the hysteresis band.

6.1.4 Compressor Command-to-Status Mapping (COMP-01 / COMP-02)

Input/Setup

Send `CMD_ON` and then `CMD_OFF` to `Compressor`.

Expected Behavior

- `CMD_ON` produces `status_out = COOLING_ON`
- `CMD_OFF` produces `status_out = COOLING_OFF`

Validation Criterion

The execution trace shows correct status outputs aligned with command inputs.

6.2 System-Level Test Cases (Integrated Control Loop)

6.2.1 Experiment E1 – Warm Startup and Closed-Loop Cooling

Input/Setup

- Initial temperature approximately 8°C
- Compressor initially OFF
- No external disturbance

Expected Behavior

- TS outputs a high temperature reading
- CTRL issues `CMD_ON`
- COMP switches to `COOLING_ON`
- TS temperature decreases over subsequent samples
- CTRL issues `CMD_OFF` when temperature falls below the lower threshold
- Temperature rises again while cooling is OFF
- The cycle repeats with bounded oscillation

Validation Criterion

The execution log shows a complete ON/OFF control cycle with feedback and bounded temperature variation.

6.2.2 Experiment E2 – Hysteresis Behavior Verification

Input/Setup
Run the integrated model long enough to observe at least one full ON/OFF cycle.

Expected Behavior

- Compressor remains ON across multiple samples while temperature decreases
- Compressor remains OFF across multiple samples while temperature increases
- Switching occurs near distinct upper/lower thresholds
- Redundant commands are not emitted repeatedly at every sample

Validation Criterion
The trace confirms hysteresis behavior and reduced switching chatter.

6.2.3 Experiment E3 – Atomic Execution Printout Validation

Input/Setup
Run the integrated model and inspect the printed state traces for each atomic model.

Expected Behavior
State printouts include meaningful variables for:

- TS (temperature, cooling flag, disturbance state)
- CTRL (pending flags, last command, last temperature)
- COMP (mode, output pending)

Validation Criterion
The output log contains interpretable atomic state traces consistent with DEVS message flow and transitions.

6.3 Observed Results from the Executed Simulation

A successful integrated run was obtained. The execution trace shows:

- Initial temperature around **8.0°C**

- Cooling initially **OFF**
- Temperature rising to **8.4°C**
- Controller issuing **CMD_ON**
- Compressor switching to **COOLING_ON**
- Temperature decreasing in steps (e.g., **8.4 → 7.8 → 7.2 → ... → 1.8 → 1.2**)
- Controller issuing **CMD_OFF**
- Compressor switching to **COOLING_OFF**
- Temperature increasing again in steps (e.g., **1.2 → 1.6 → 2.0 → ... → 6.0 → 6.4**)
- Controller issuing **CMD_ON** again
- Beginning of the next cycle

These results confirm correct closed-loop feedback and hysteresis-based regulation.

6.4 Validation Summary

- **E1 (Warm startup integrated loop):** Pass
- **E2 (Hysteresis switching behavior):** Pass
- **E3 (Atomic execution printout validation):** Pass

Overall, the baseline integrated model operates as intended.

7. Results Analysis and Discussion

7.1 System Behavior

The implemented model exhibits the expected refrigerator control behavior:

- While cooling is OFF, temperature increases at a fixed warming rate.
- Once the upper threshold is exceeded, the controller issues `CMD_ON`.
- The compressor turns cooling ON and reports status feedback.
- While cooling is ON, temperature decreases at a fixed cooling rate.
- Once the lower threshold is crossed, the controller issues `CMD_OFF`.
- The cycle repeats, resulting in bounded oscillation.

This is the intended behavior of a hysteresis-controlled regulation system.

7.2 Strengths of the Model

- Clear modular decomposition (sensor, controller, compressor)
- Strong correspondence between DEVS formalization and Cadmium implementation
- Traceable execution through atomic model state printouts
- Supports incremental testing and future extensions
- Demonstrates hysteresis-based switching behavior clearly

7.3 Limitations

- Thermal dynamics are simplified (fixed increments instead of physics-based heat transfer)
- Disturbance behavior is optional and not fully exercised in the baseline run
- No automatic CSV/plot export in the baseline implementation (trace-based validation is used)

7.4 Possible Extensions

- Add explicit disturbance input scenarios (e.g., door-open events)
- Add alarm generation and corresponding validation experiments
- Export simulation traces to CSV and generate plots
- Compare different thresholds and sample times
- Add sensor noise or delays for robustness testing

8. Changes from the Initial Specification During Implementation

During implementation and debugging, several practical refinements were made:

8.1 Code-Level State Representation

The implementation uses explicit state flags (`command_pending`, `output_pending`, `has_last_cmd`) instead of relying only on abstract tuple descriptions. This improves clarity and compatibility with Cadmium coding patterns.

8.2 Zero-Time Output Handling

`Controller` and `Compressor` use pending-output flags and immediate output scheduling (`ta = 0`) to represent DEVS zero-time outputs correctly in Cadmium.

8.3 NDTime Compatibility Adjustments

Time handling was updated to match Cadmium's `NDTime` type after debugging and build corrections.

8.4 Parameter Selection for Visible Hysteresis Cycles

Baseline values (initial temperature, warming/cooling increments, thresholds) were selected to produce clear and stable ON/OFF cycles in the simulation trace.

These refinements preserve the conceptual design while improving implementation correctness and simulator compatibility.

9. Reproducibility: Build and Run Procedure

The project was built and executed successfully using **MSYS2 UCRT64** with **CMake** and **MinGW Makefiles**.

9.1 Build and Run Steps

1. Open **MSYS2 UCRT64**
2. Navigate to the project folder (`cadmium_generated`)
3. Create a clean build directory
4. Configure and build the project
5. Run the executable

9.2 Example Commands

```
cd cadmium_generated
rm -rf build
mkdir build
cd build
/c/msys64/ucrt64/bin/cmake.exe -G "MinGW Makefiles" ..
```

```
mingw32-make -j1
./smart_fridge.exe
```

10.Conclusion

This project developed a DEVS-based smart fridge temperature control system and implemented it in Cadmium. The final model includes a conceptual system definition, formal atomic and coupled DEVS specifications, transition pseudocode, hierarchical structure, and a tested executable implementation.

The simulation results confirm correct hysteresis-based ON/OFF regulation and bounded temperature behavior. The model is modular and can be extended with disturbances, alarms, and richer thermal dynamics in future work.

12. References

- Course lecture notes on DEVS modeling and simulation
- Cadmium DEVS simulator documentation and examples
- DEVS-Graph tool documentation and resources