정형검증도구를 통한 요구공학기술을 연계한 스마트 살충제의 요구사항 분석과 개념적 설계

박선영 | IT융합정보보호학과 | 건국대학교

목차

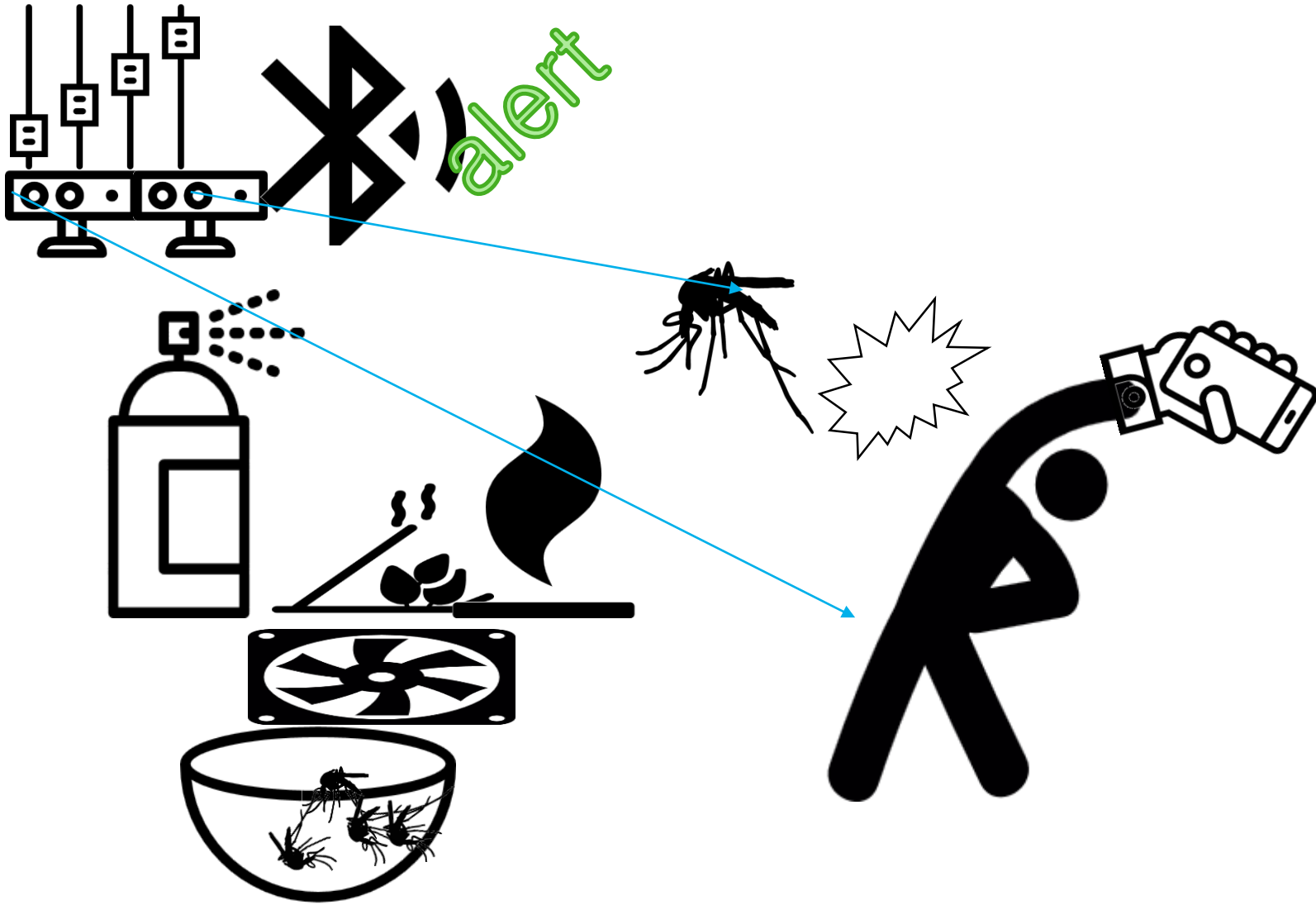
- 1. Motivation & introduction
- 2. Example
- 3. Modeling
- 4. Simulation
- 5. Property
- 6. Verification

1. Motivation & Introduction

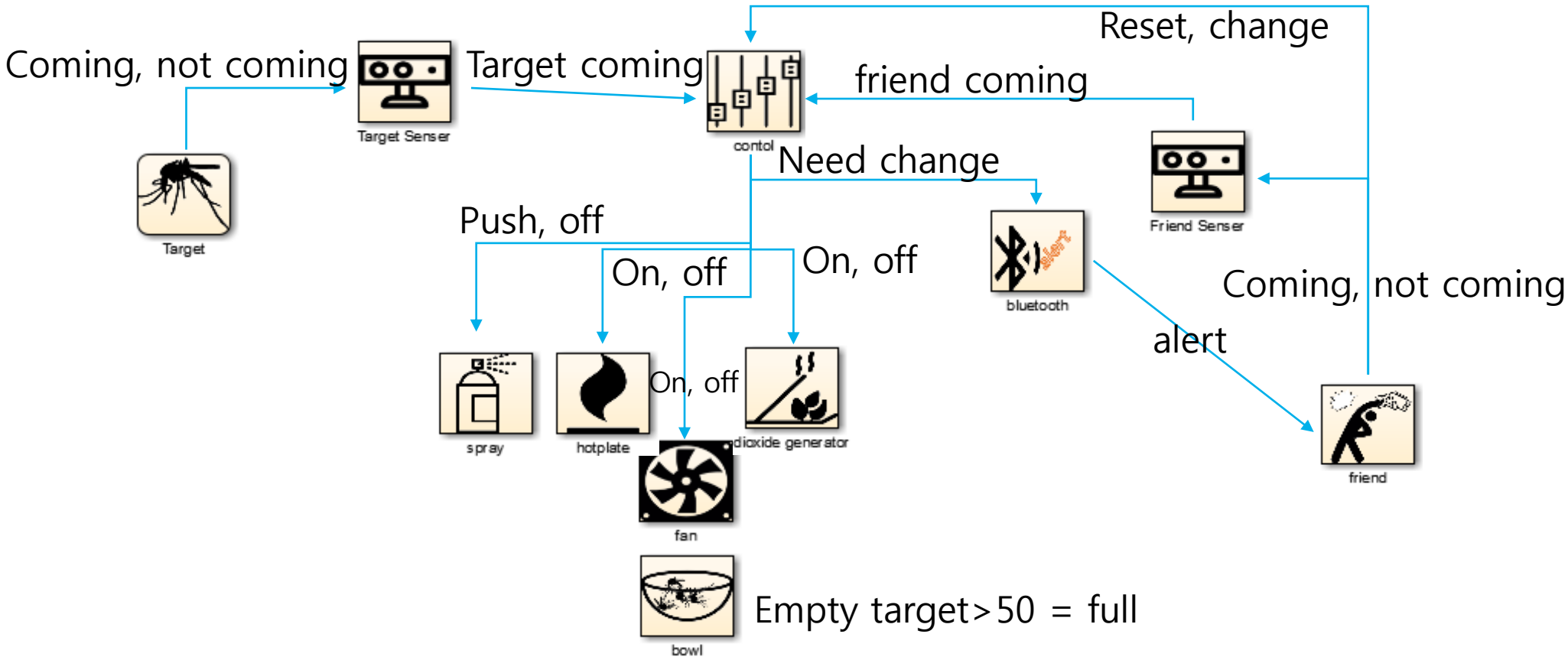
- iot가 대두 되면서 스마트 폰과 연동 하여 활용 가능한 장치들이 다수 발생 하고 있으나, 살충 시스템의 연동 시스템은 공공에서 방재용으로 사용하거나 단순 포집 단계의 장치에서 머무르고 있음
- 이에 자동 살충과 포집을 목적으로 하고 해당 항목을 스마트폰으로 연계하여 관리할 수 있도록 하는 시스템을 구성하고자 함
- 본 시스템은 기존에 상용화된 바가 없어 새로운 요구 사항의 도출이 필요
- 이에 소프트웨어 공학의 '요구공학' 기법을 통해 초기 도출 된 요구사항에 시나리오를 구성하고 모델링한 것을 SMV Model checker 를 통해 요구 사항의 검증을 진행하여 본 케이스의 필요성과 가능성을 검증

2. Example

- 모기 포집 순서
  - 유인 → 모기인식센서 등으로 타깃 발견 → 살충 제품 살포 → 팬 작동 하여 포집
- 사용자의 역할
  - 수동으로 중단
  - 사람센서 앞에 등장
  - 스마트 폰으로 알림을 받은 후 조치



3. Modeling



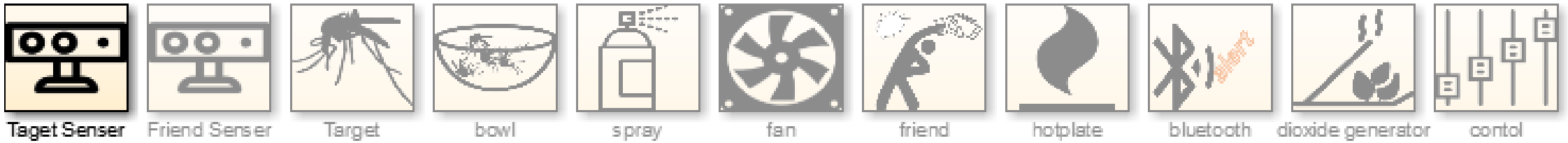
가상 변수

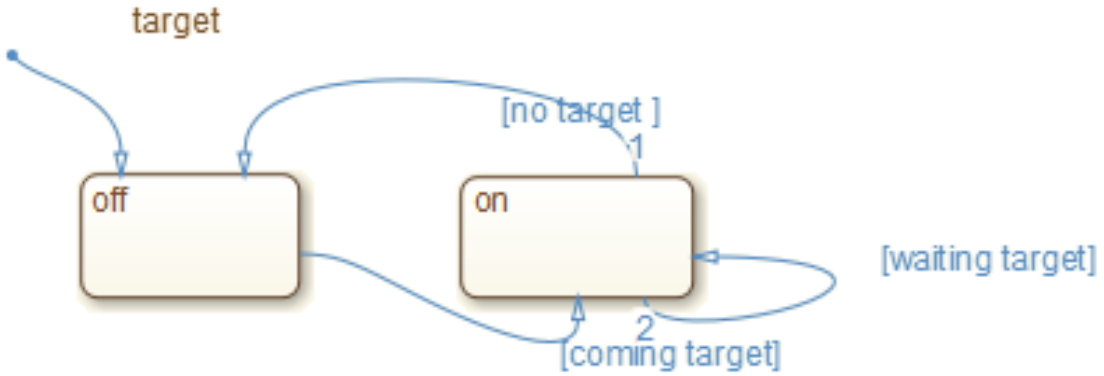
Alert : Spray capacity < 100

dioxide < 100

target count <50

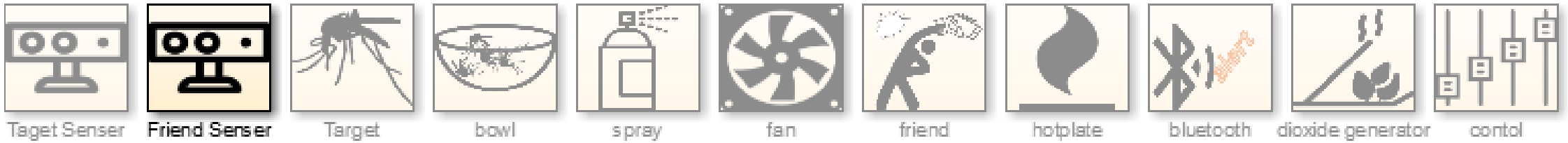
3. Modeling

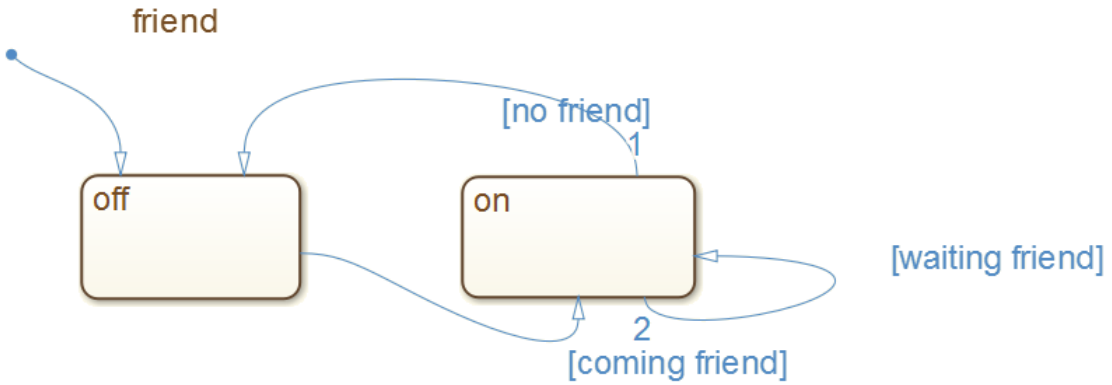





```
MODULE Target_Senser(target)
  VAR
    status : { off, on };
  ASSIGN
    init(status) := off;
    next(status) :=
      case
        target = coming          : on;
        target = wating          : on;
        target = not_coming      : off;
      esac;
```

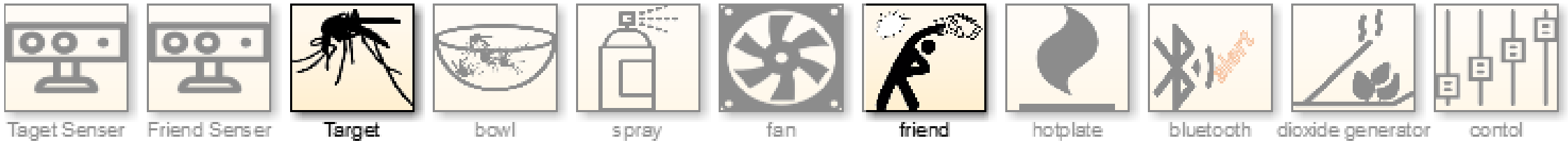
3. Modeling





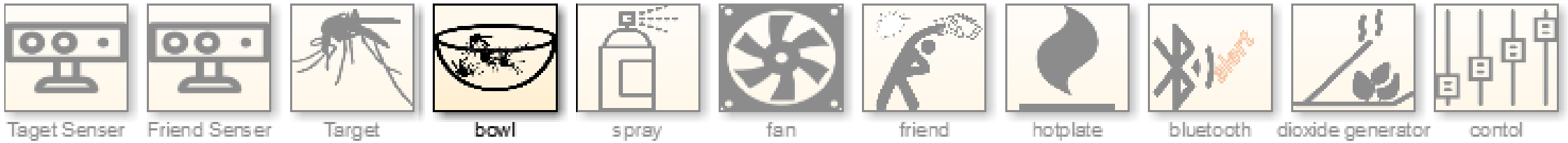

```
MODULE Friend_Senser(friend)
  VAR
    status : { off, on };
  ASSIGN
    init(status) := off;
    next(status) :=
      case
        friend = coming          : on;
        friend = wating          : on;
        friend = not_coming      : off;
      esac;
```

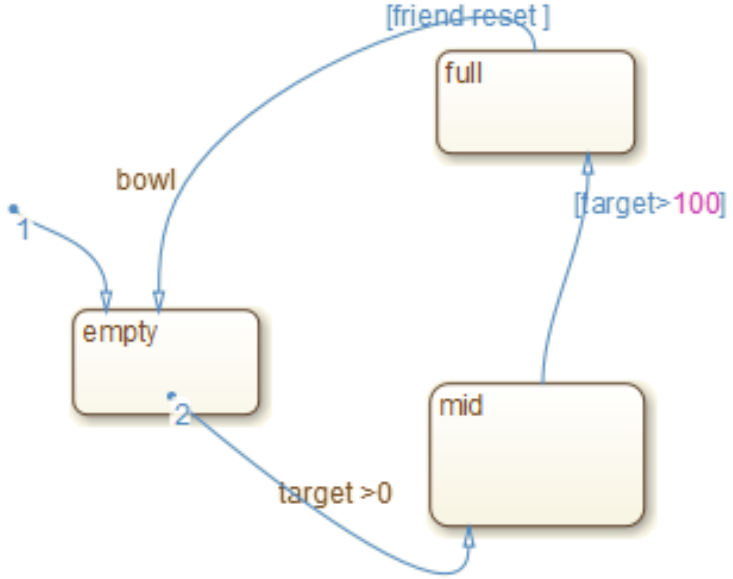
3. Modeling




```
target: {coming, not_coming, wating};
friend: {coming, not_coming, wating, need_reset, need_change, need_empty};
```

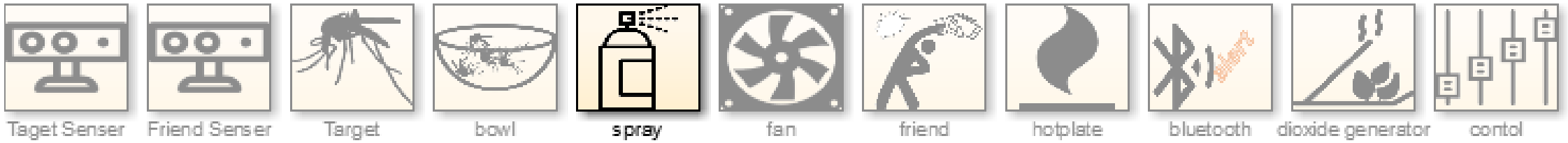
3. Modeling

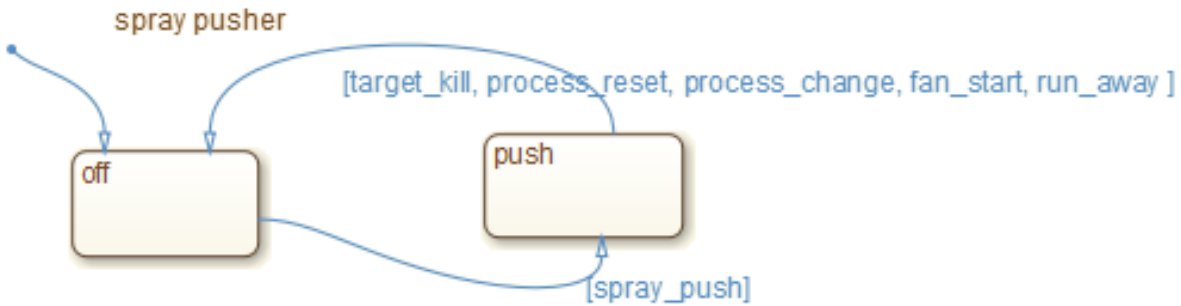





```
MODULE Bowl(target_count)
  VAR
    state : { empty, mid ,full };
  ASSIGN
    init(state) := empty;
    next(state) :=
      case
        target_count = 0     : empty;
        target_count > 0     :  mid;
        target_count >100    : full;
        TRUE:empty;
      esac;
```

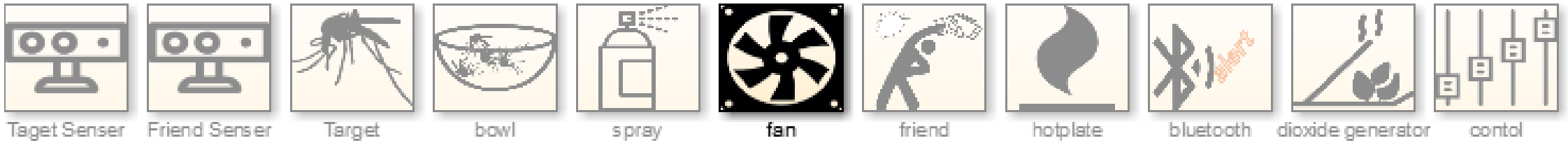
3. Modeling

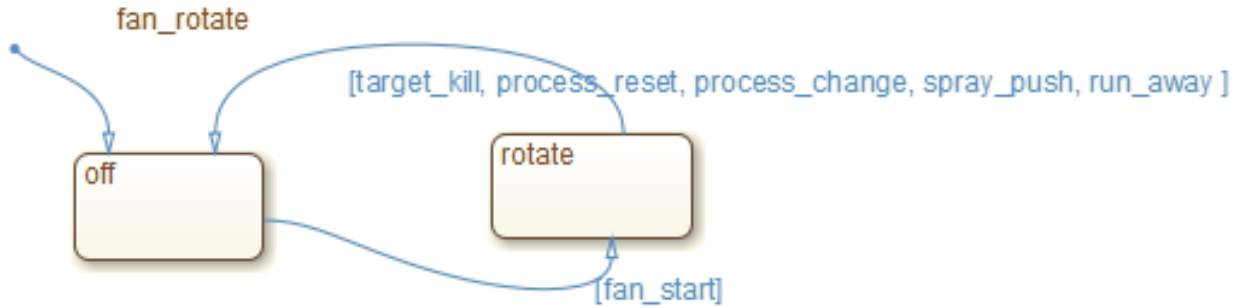





```
MODULE Spray(con_request)
  VAR
    Spray_pusher : { off, on };
  ASSIGN
    init(Spray_pusher) := off;
    next(Spray_pusher) :=
      case
        con_request = spray_push          : on;
        con_request = target_kill         : off;
        con_request = process_reset       : off;
        con_request = process_change      : off;
        con_request = fan_start           : off;
        con_request = run_away            : off;
        TRUE : off;
      esac;
```

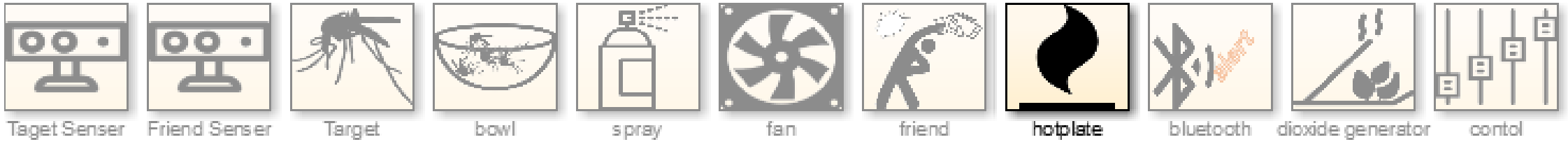
3. Modeling

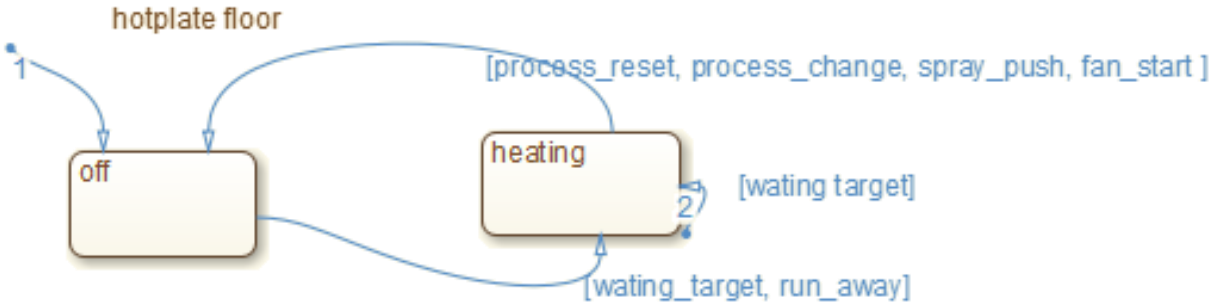





```
MODULE Fan(con_request)
  VAR
    Fan_rotate : { off, rotate };
  ASSIGN
    init(Fan_rotate) := off;
    next(Fan_rotate) :=
      case
        con_request = fan_start          : rotate;
        con_request = target_kill        : off;
        con_request = process_reset     : off;
        con_request = process_change    : off;
        con_request = spray_push     : off;
        con_request = run_away      : off;
        TRUE : off;
      esac;
```

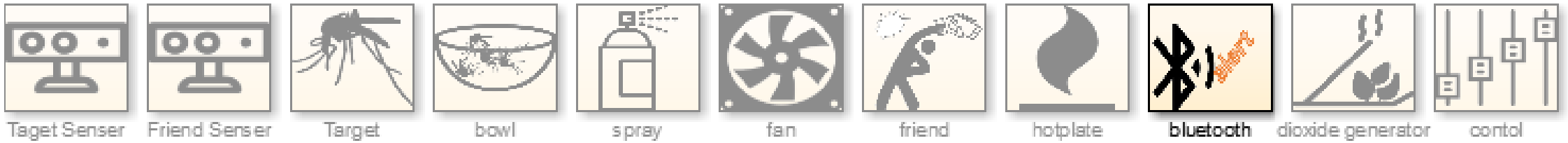
3. Modeling

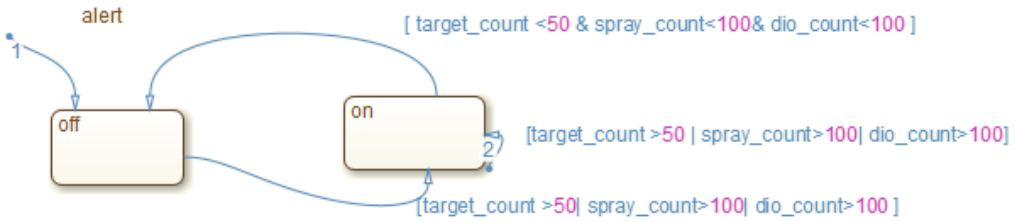





```
MODULE HotPlate(con_request)
  VAR
    HotPlate_floor : { off, heating };
  ASSIGN
    init(HotPlate_floor) := off;
    next(HotPlate_floor) :=
      case
        con_request = wating_target     : heating;
        con_request = run_away      : heating;
        con_request = process_reset     : off;
        con_request = process_change    : off;
        con_request = spray_push     : off;
        con_request = fan_start : off;
        TRUE : off;
      esac;
```

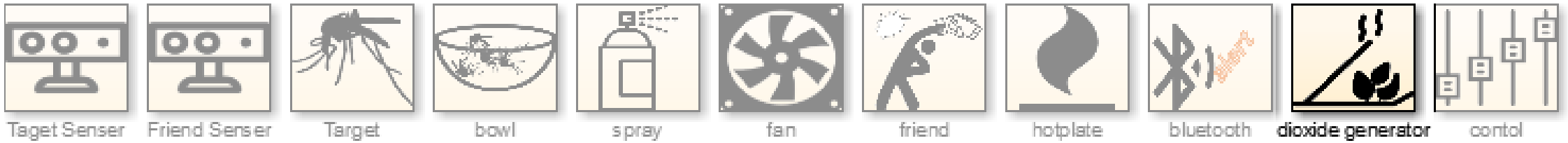
3. Modeling

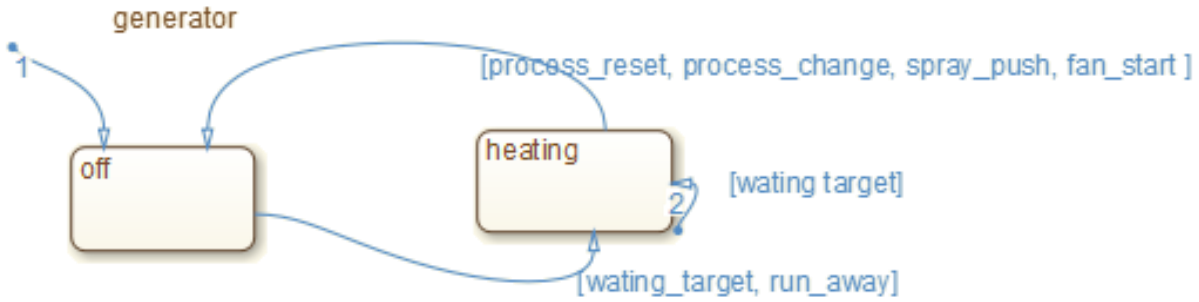





```
MODULE BlueTooth(target_count, spray_count, dio_count)
  VAR
    alert : {off, on};
  DEFINE
        Serviceable_bowl := target_count <50;
        Serviceable_spray := spray_count <100;
        Serviceable_dio := dio_count <100;
  ASSIGN
    init(alert) := off;
    next(alert) :=
        case
            !(Serviceable_bowl | Serviceable_spray | Serviceable_dio) : on;
            TRUE : off;
        esac;
```

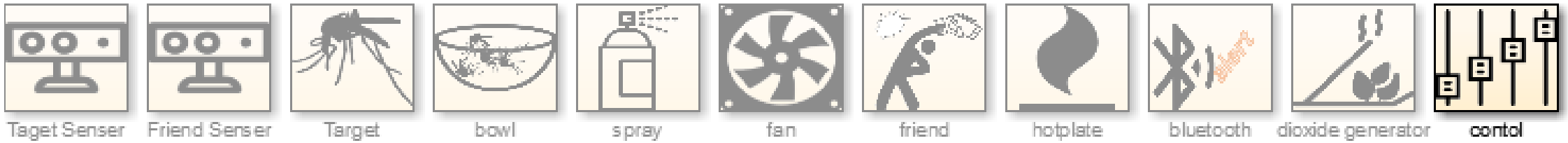
3. Modeling

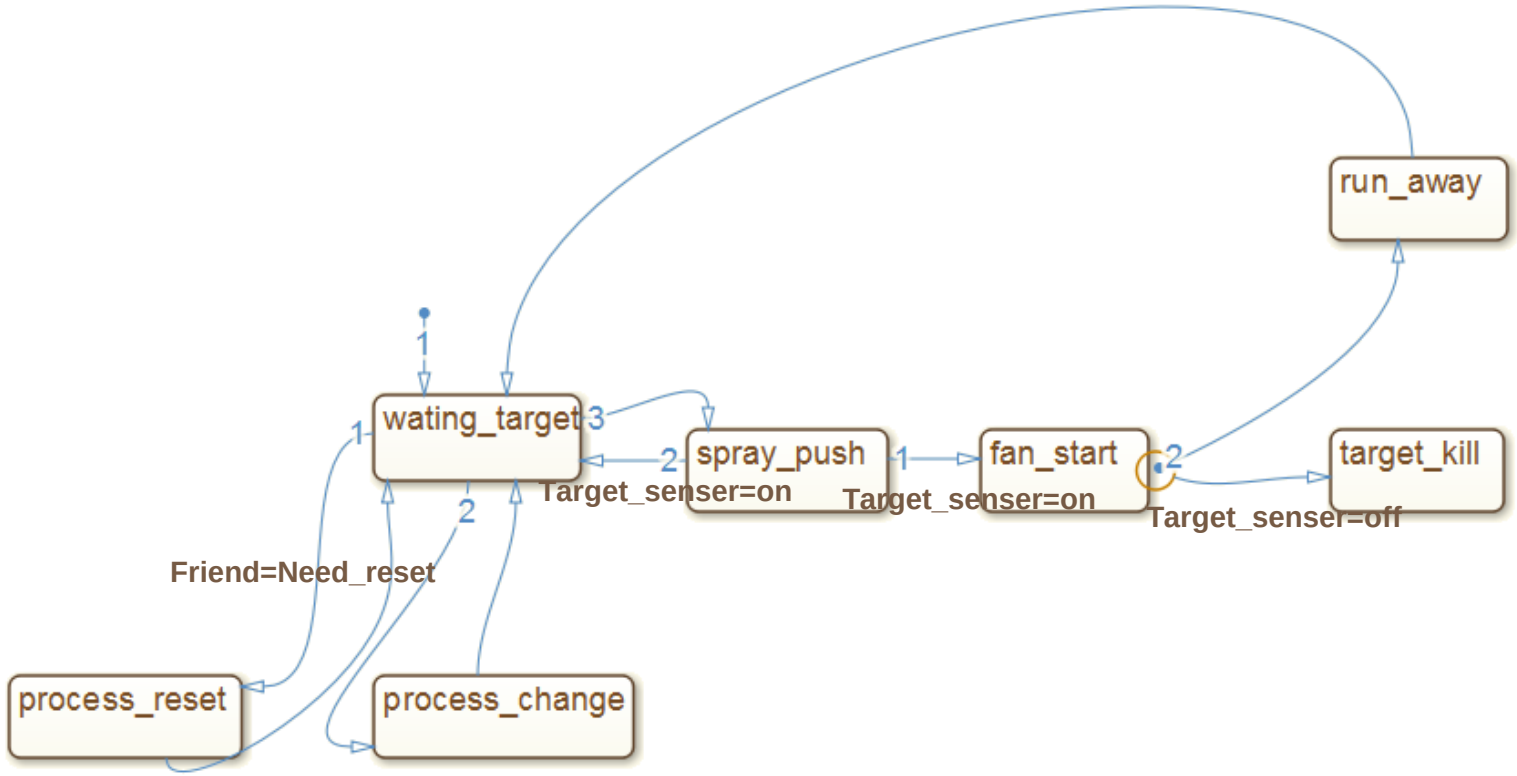


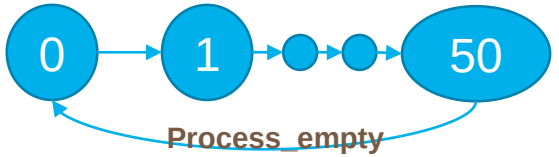


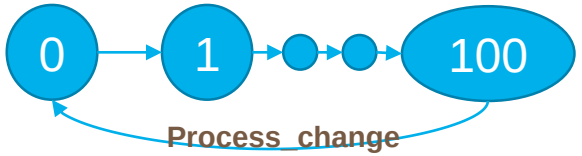

```
MODULE Dioxide_Generator(con_request)
  VAR
    generator : { off, heating };
  ASSIGN
    init(generator) := off;
    next(generator) :=
      case
        con_request = wating_target     : heating;
        con_request = run_away     : heating;
        con_request = process_reset    : off;
        con_request = process_change   : off;
        con_request = spray_push    : off;
        con_request = fan_start : off;
        TRUE : off;
      esac;
```

3. Modeling

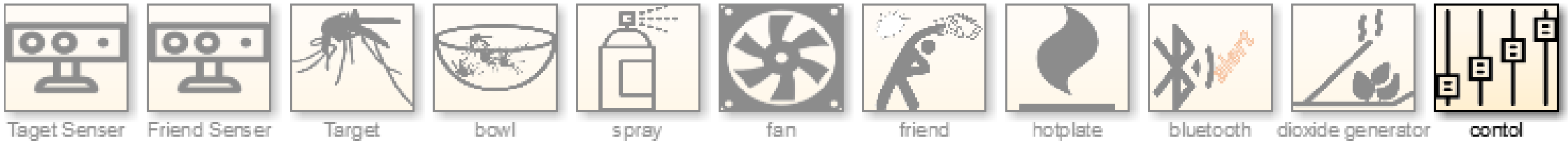

Taget Senser
Friend Senser
Target
bowl
spray
fan
friend
hotplate
bluetooth
dioxide generator
contol


run_away
wating_target
spray_push
fan_start
target_kill
process_reset
process_change
Target_senser=on
Target_senser=on
Target_senser=off
Friend=Need_reset


0
1
50
Process_empty


0
1
100
Process_change

3. Modeling




```
--current state 기준으로 next state를 찾기 쉽도록 우선->표준 순으로 이름 하기 위해 이전 상태를 체크 하여 같을 이름 사카드록 한다.
MODULE Controller(target_senser, friend_senser, friend, bluetooth)
    VAR
        con_request : {wating_target, process_reset, process_empty ,process_change, spray_push, fan_start ,target_kill, run_away};
        target_count: 0..50;
        spray_count: 0..100;
        dio_count:0..100;

    DEFINE
        Serviceable_bowl := target_count <50;
        Serviceable_spray := spray_count <100;
        Serviceable_dio := dio_count <100;
        ready := target_senser.status=on & friend_senser.status=off ;
        wait  := (target_senser.status=off & friend_senser.status=on) | (target_senser.status=on & friend_senser.status=on);
        serv_on := Serviceable_bowl | Serviceable_spray | Serviceable_dio;


    ASSIGN
        --기본 상태는 wating_target임 / 이후 상태에 따라 spray_push -> fan_start 혹은 run_away -> 이후 target_kill 혹은 run_away 혹은 타겟이 죽지 않았으면 다시 spray_push 순으로 진행
        init(con_request) := wating_target;
        next(con_request) :=
            case
                --타겟이 없으면 타겟을 계속 기다린다
                target_senser.status=off & friend_senser.status=off & Serviceable_dio  : wating_target;

                --타겟이 없어 있어도 사람이 없어 있으면 살충제를 뿌리면 안된다.(reset 시킴)
                target_senser.status=on & friend_senser.status=on    : process_reset;

                --타겟이 없어 있어야 하고 사람이 없으면서 살충제 용량이 있으면 살충제를 뿌리도록 한다
                target_senser.status=on & friend_senser.status=off
                &con_request=wating_target & Serviceable_spray       : spray_push;

                --spray가 뿌려지면 실은 fan을 동작 시킨다.
                con_request=spray_push       : fan_start;
                --이전 요청이 fan start 였고 타겟이 사라졌다면 타겟이 표적된 것으로 본다.
                target_senser.status=off
                &con_request=fan_start                      : target_kill;

                --fan이 작동하고 이전 컨트롤러의 요청이 spray를 작동 했었고 센서에서 타겟이 체거 되지 않았으면 타겟이 죽지 않고 도망간 것으로 간주 한다.
                target_senser.status=off & friend_senser.status=off
                &con_request=fan_start & Serviceable_spray    : run_away;

                --현재 상태가 run_away 상태면 모든 동작을 한번 중단 하고 wating_target 상태로 한번 변경 한다.
                target_senser.status=off & friend_senser.status=off
                &con_request=run_away                         : wating_target;


                --블루투스의 alert가 on이고 friend가 need_reset/ need_change / need empty 각 상태를 진행 할수 있도록 한다.
                !(Serviceable_spray | Serviceable_dio) & friend=need_change      : process_change;
                friend=need_reset                             : process_reset;
                --비루기를 동하나 이미 비어 있는 상태라면 또 비울 필요 없음.
                friend=need_empty & target_count > 0          : process_empty;
                TRUE : run_away;

            esac;
        --타겟 카운트가 계속 모아 bowl이 꽉이 되면 bowl을 비워줘야 한다.(process_empty)
        init(target_count) :=0;
        next(target_count) :=
            case
                --target_count 의 max가 될때까지 count를 +함
                con_request = target_kill & target_count < 50   : target_count +1;
                --friend 가 비루는 프로세스를 수행 할 시에 0이 됨
                con_request = process_empty                     : 0;
                TRUE : 0;
            esac;

        init(spray_count) :=0;
        next(spray_count) :=
            case
                --target_count 의 max가 될때까지 count를 +함
                con_request = target_kill & spray_count < 100   : spray_count +1;
                --friend 가 비루는 프로세스를 수행 할 시에 0이 됨
                con_request = process_change                    : 0;
                con_request = process_reset                     : spray_count;
                TRUE : 0;
            esac;

        init(dio_count) :=0;
        next(dio_count) :=
            case
                --target_count 의 max가 될때까지 count를 +함
                con_request = wating_target & dio_count < 100   : dio_count +1;
                --friend 가 비루는 프로세스를 수행 할 시에 0이 됨
                con_request = process_change                    : 0;
                --단순히 reset 만 한 경우 가졌랑은 그대로 둔다 한다.
                con_request = process_reset                     : dio_count;
                TRUE : 0;
            esac;
```

4. Simulation

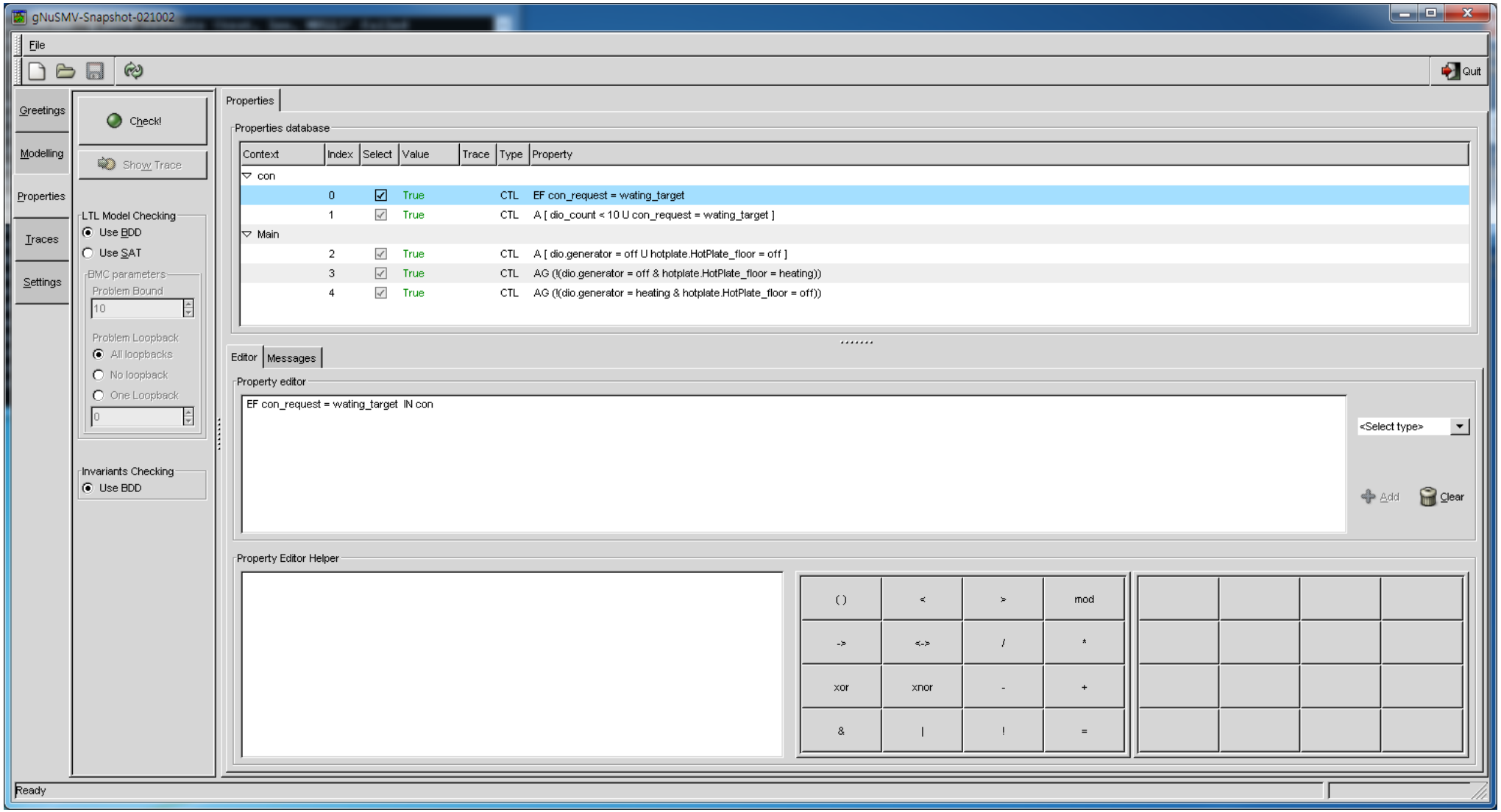
```
NuSMV > go
NuSMV > check_fsm

############################################################
The transition relation is total: No deadlock state exists
############################################################
NuSMV > print_fsm_stats
Statistics on BDD FSM machine.
BDD nodes representing init set of states: 33
BDD nodes representing state constraints: 1
BDD nodes representing input constraints: 1
Forward Partitioning Schedule BDD cluster size (#nodes):
cluster 1       :       size 1115
cluster 2       :       size 169
Backward Partitioning Schedule BDD cluster size (#nodes):
cluster 1       :       size 1115
cluster 2       :       size 169
NuSMV > check_ctlspec
-- specification EF con_request = spray_push IN con is true
-- specification EF con_request = wating_target IN con is true
-- specification EF con_request = process_reset IN con is true
-- specification EF con_request = process_change IN con is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  target = coming
  friend = coming
  target_senser.status = off
  friend_senser.status = off
  con.con_request = wating_target
  con.target_count = 0
  con.spray_count = 0
  con.dio_count = 0
```

```
############################################################################
system diameter: 102
reachable states: 52596 (2^15.6827) out of 2.87678e+010 (2^34.7437)
------- State    1 ------
  target = wating
  friend = need_empty
  target_senser.status = on
```

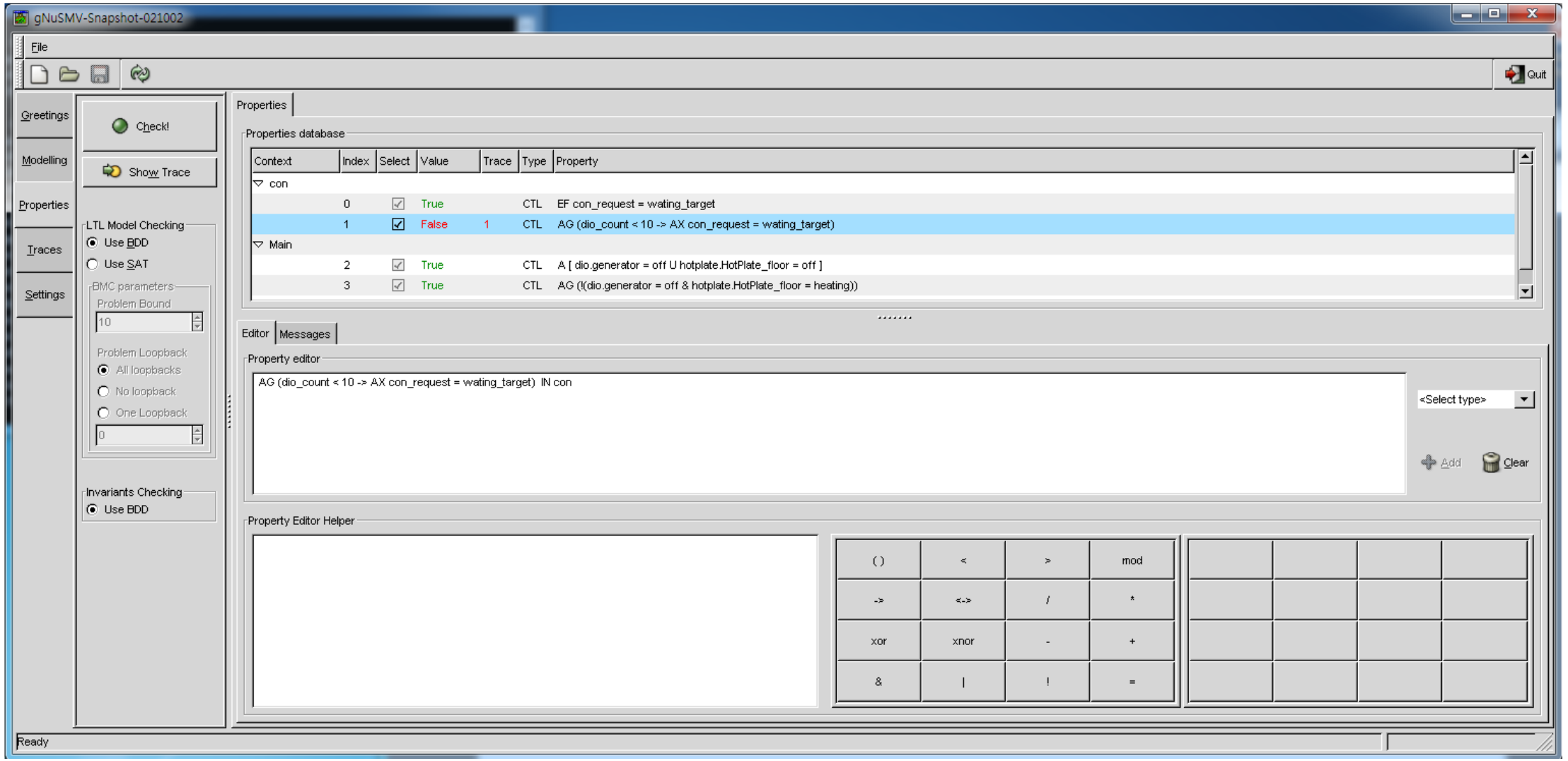
5. Property

- 컨트롤러에서 각 상태로 이동이 가능한지(유인→포집) 확인
- 이산화 탄소 발생기와 열판이 동시에 구동되어야 함
- 알림은 각 요소들이 모자를 때 발생 할 수 있어야 하며 각 요소들이 남아 있을 경우 계속 타깃을 기다릴 수 있어야 함
- 살충제는 사람이 인식 되면 뿌려지면 안됨
- 사용자는 언제든 기기를 중단 시킬 수 있어야 함

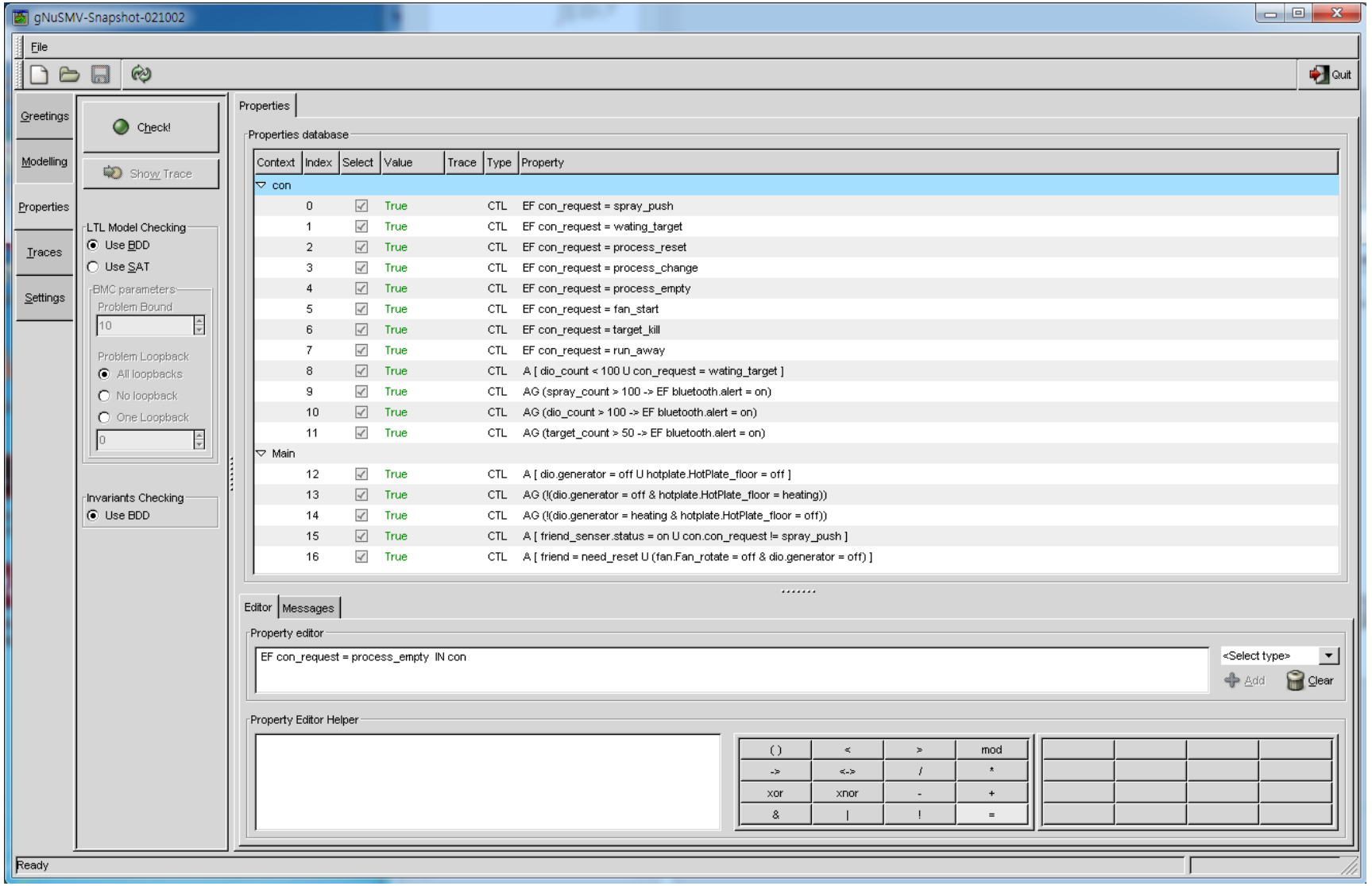
6. Verification



6. Verification



6. Verification



6. Verification

☑	True	CTL	EF con_request = spray_push
☑	True	CTL	EF con_request = wating_target
☑	True	CTL	EF con_request = process_reset
☑	True	CTL	EF con_request = process_change
☑	True	CTL	EF con_request = process_empty
☑	True	CTL	EF con_request = fan_start
☑	True	CTL	EF con_request = target_kill
☑	True	CTL	EF con_request = run_away
☑	True	CTL	A [dio_count < 100 U con_request = wating_target]
☑	True	CTL	AG (spray_count > 100 -> EF bluetooth.alert = on)
☑	True	CTL	AG (dio_count > 100 -> EF bluetooth.alert = on)
☑	True	CTL	AG (target_count > 50 -> EF bluetooth.alert = on)
☑	True	CTL	A [dio.generator = off U hotplate.HotPlate_floor = off]
☑	True	CTL	AG (!(dio.generator = off & hotplate.HotPlate_floor = heating))
☑	True	CTL	AG (!(dio.generator = heating & hotplate.HotPlate_floor = off))
☑	True	CTL	A [friend_senser.status = on U con.con_request != spray_push]
☑	True	CTL	A [friend = need_reset U (fan.Fan_rotate = off & dio.generator = off)]

각 항목으로 잘 이동하는지 확인

이산화 탄소 발생기가 소모되지 않았을때만 wating 상태가 유지된다.

각 요소들이 소모되거나 찬 경우 알림을 해야 한다

이산화탄소 발생기와 열판은 함께 움직여야 한다

사용자가 등장시에 스프레이가 움직이면 안된다

사용자는 항상 원할 때 기기를 reset 시킬 수 있으며 Reset 시에는 모든 장치가 동작하지 않음