A Declarative Goal-oriented Framework for Smart Environments with LPaaS

Giuseppe Bisicchia, Stefano Forti and Antonio Brogi

giuseppe.bisicchia@outlook.it

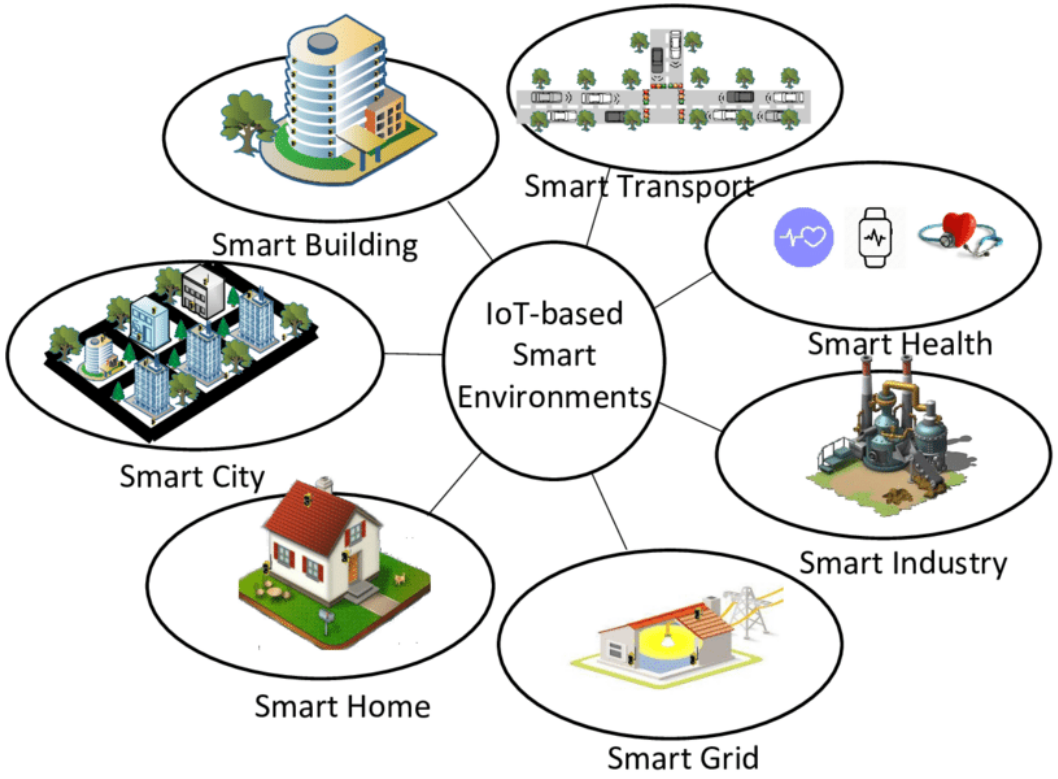
Service-oriented, Cloud and Fog Computing Research Group

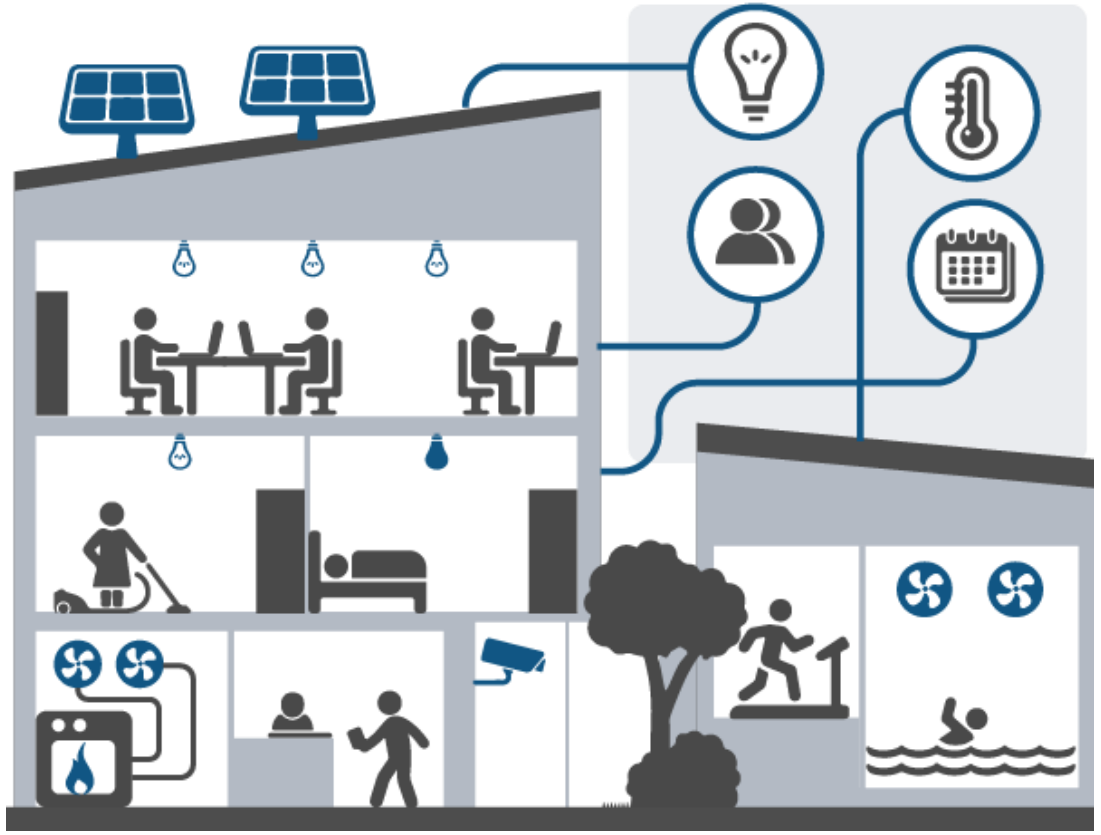
Department of Computer Science

University of Pisa, Italy

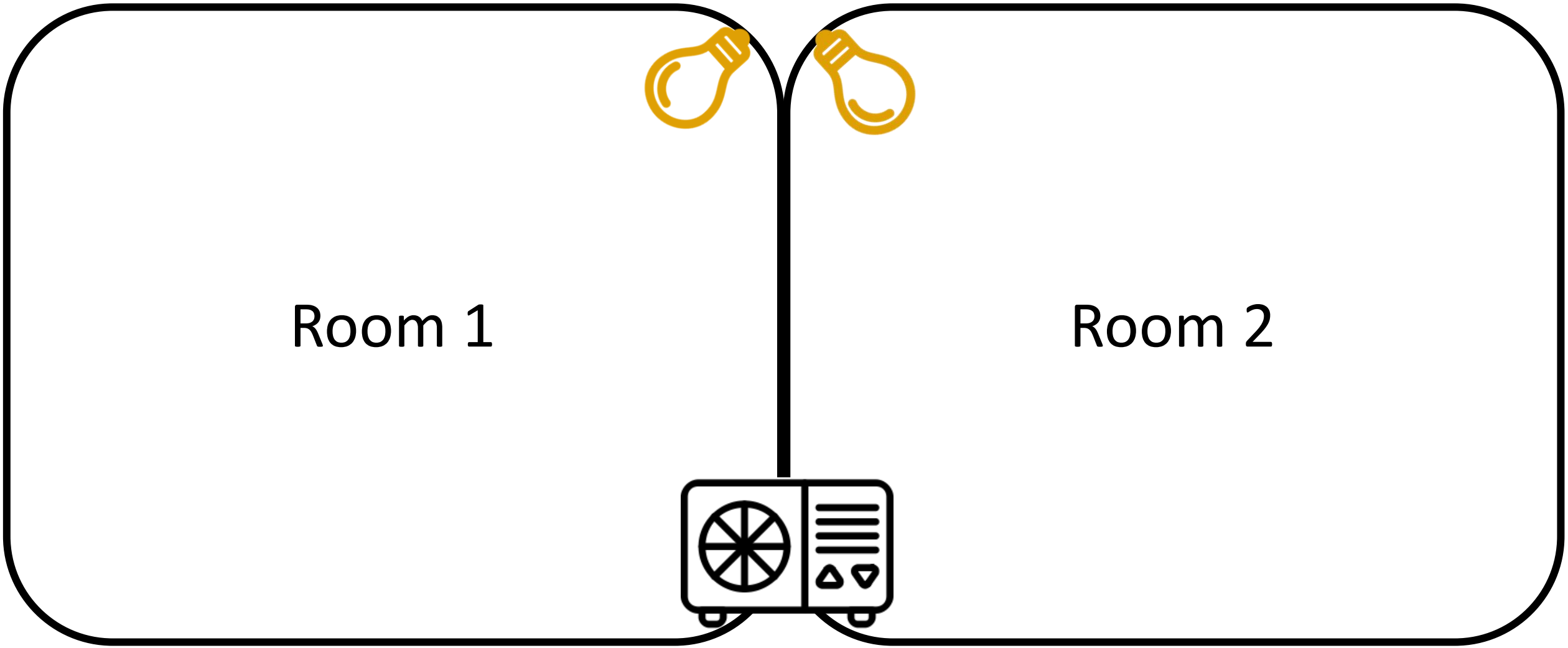
36th Edition of the Italian Conference on Computational Logic, 7-9 September 2021

Smart Environments

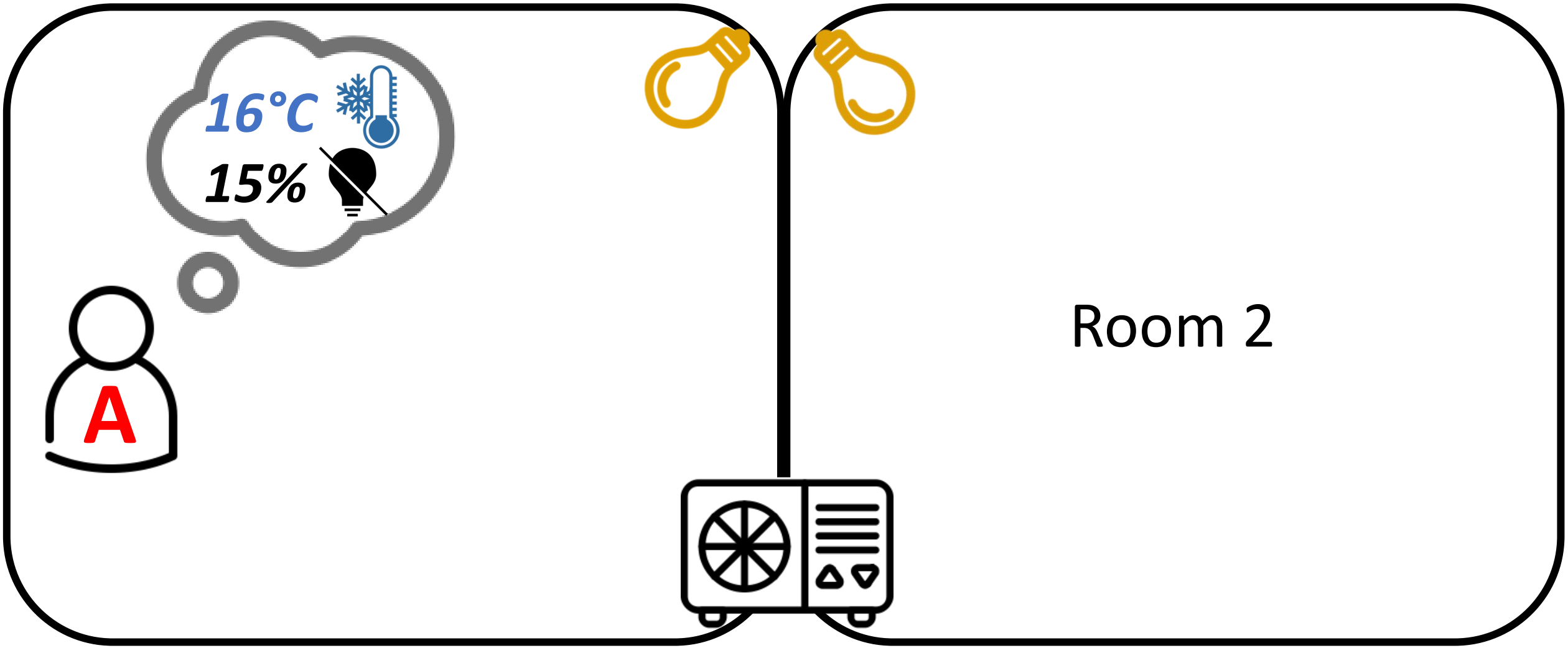




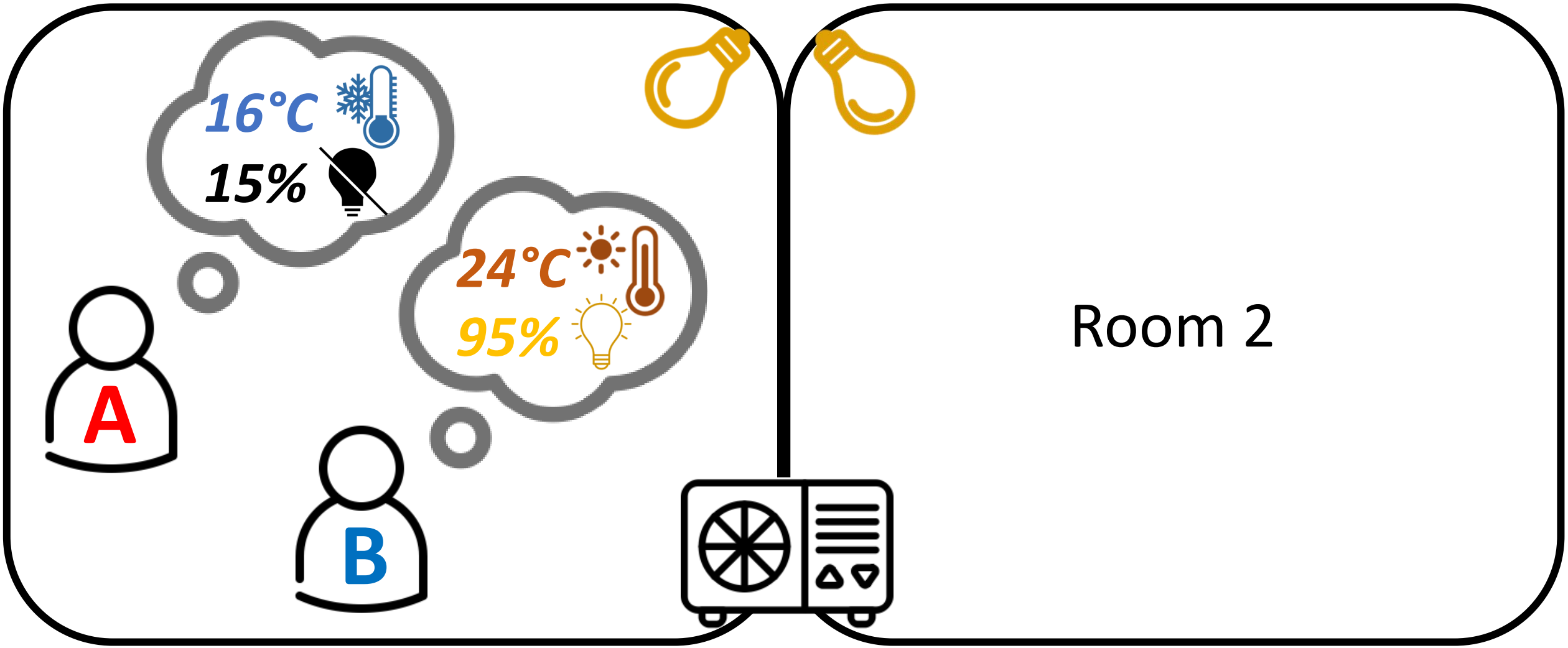
Motivating Scenario



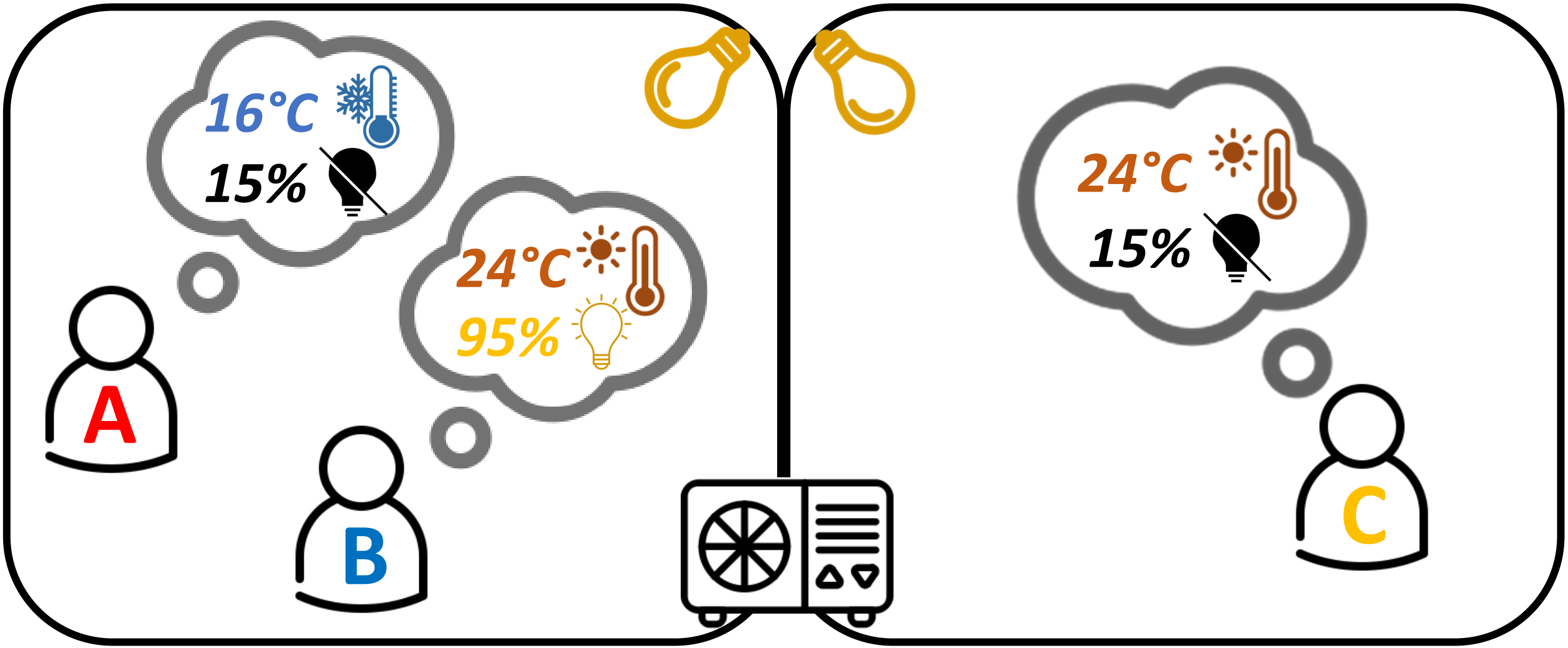
Motivating Scenario



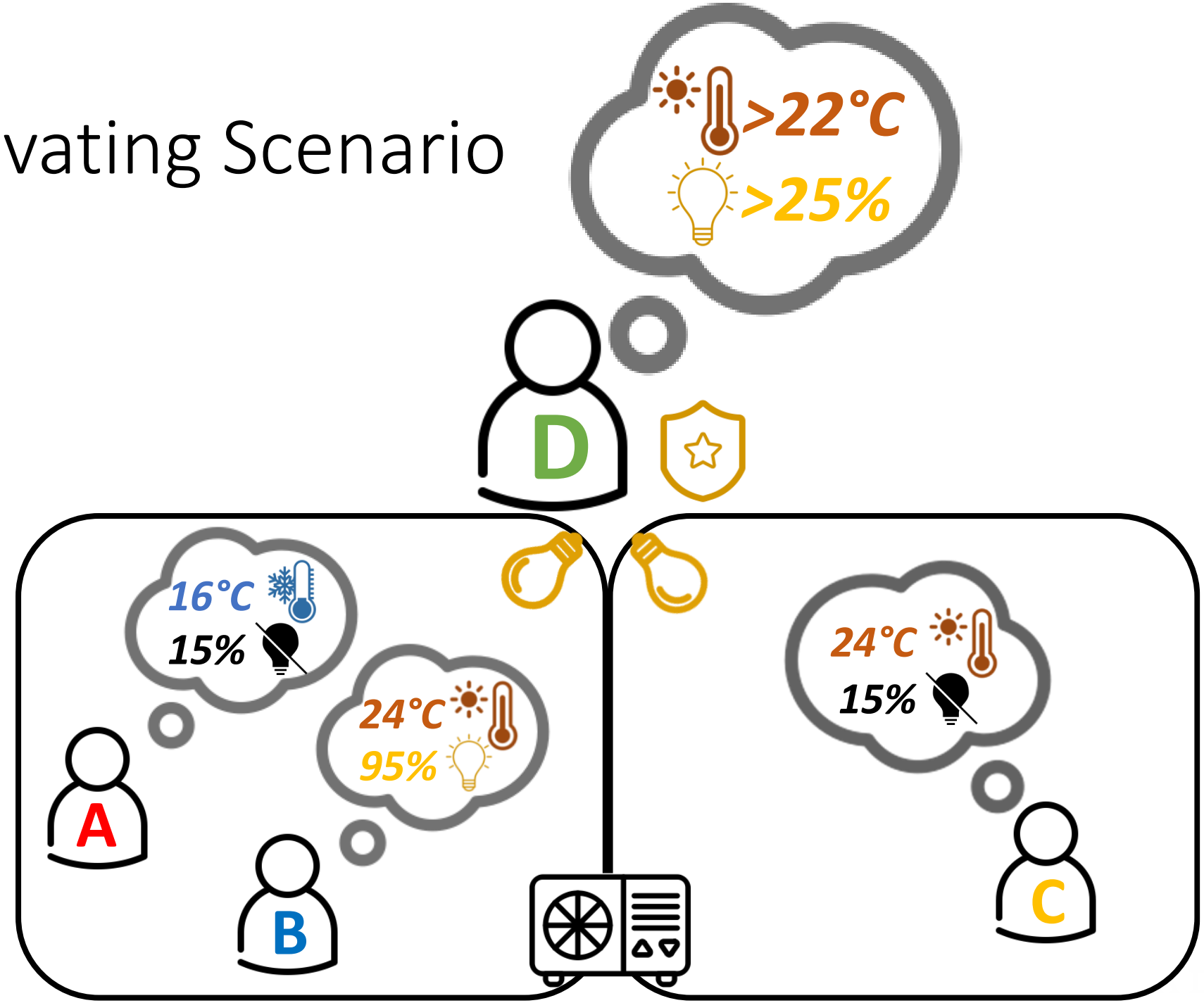
Motivating Scenario



Motivating Scenario



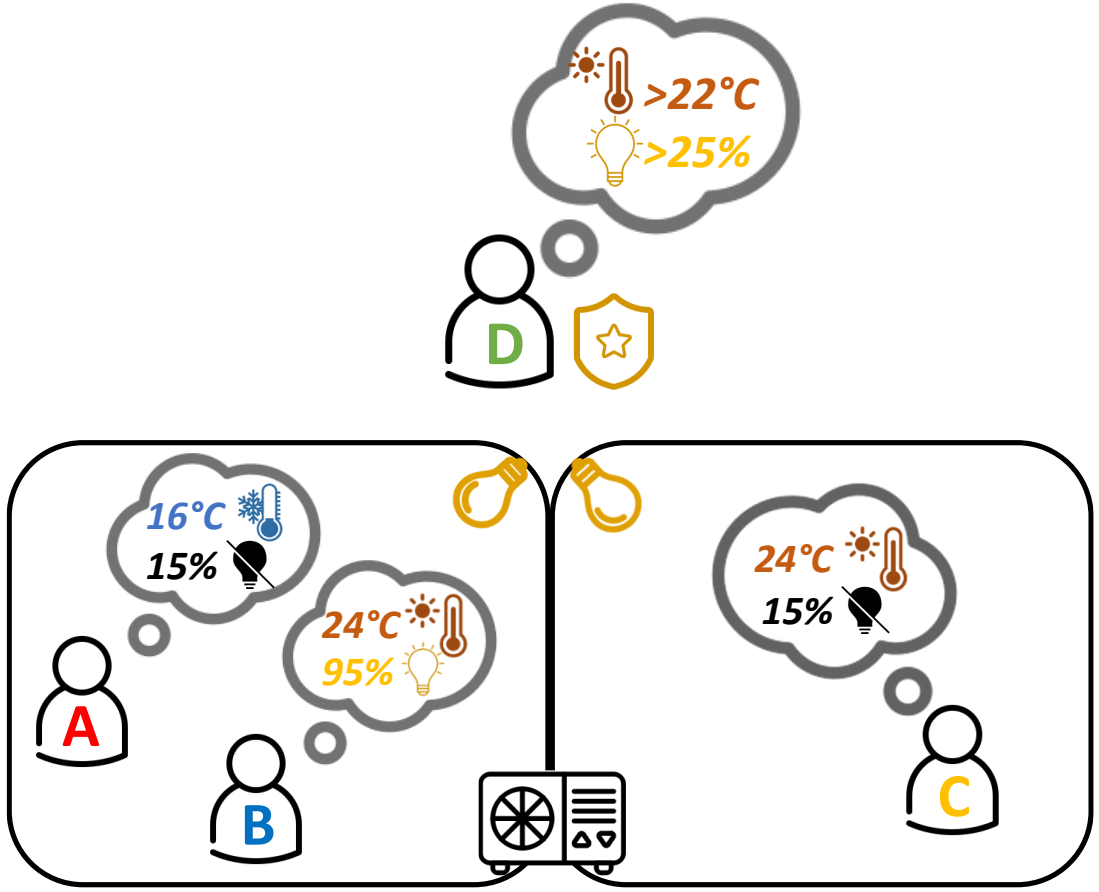
Motivating Scenario



Open Questions

Three types of **conflicts** can arise:

- **[User-User]** *How to **mediate all users' preferences** to satisfy them in the best possible way?*
- **[User-Admin]** *How to **achieve goals** set by the Sys. Admin. (e.g. energy savings)?*
- **[IoT-IoT]** *How to **reach a mediated target state** by suitably settings the available actuators?*



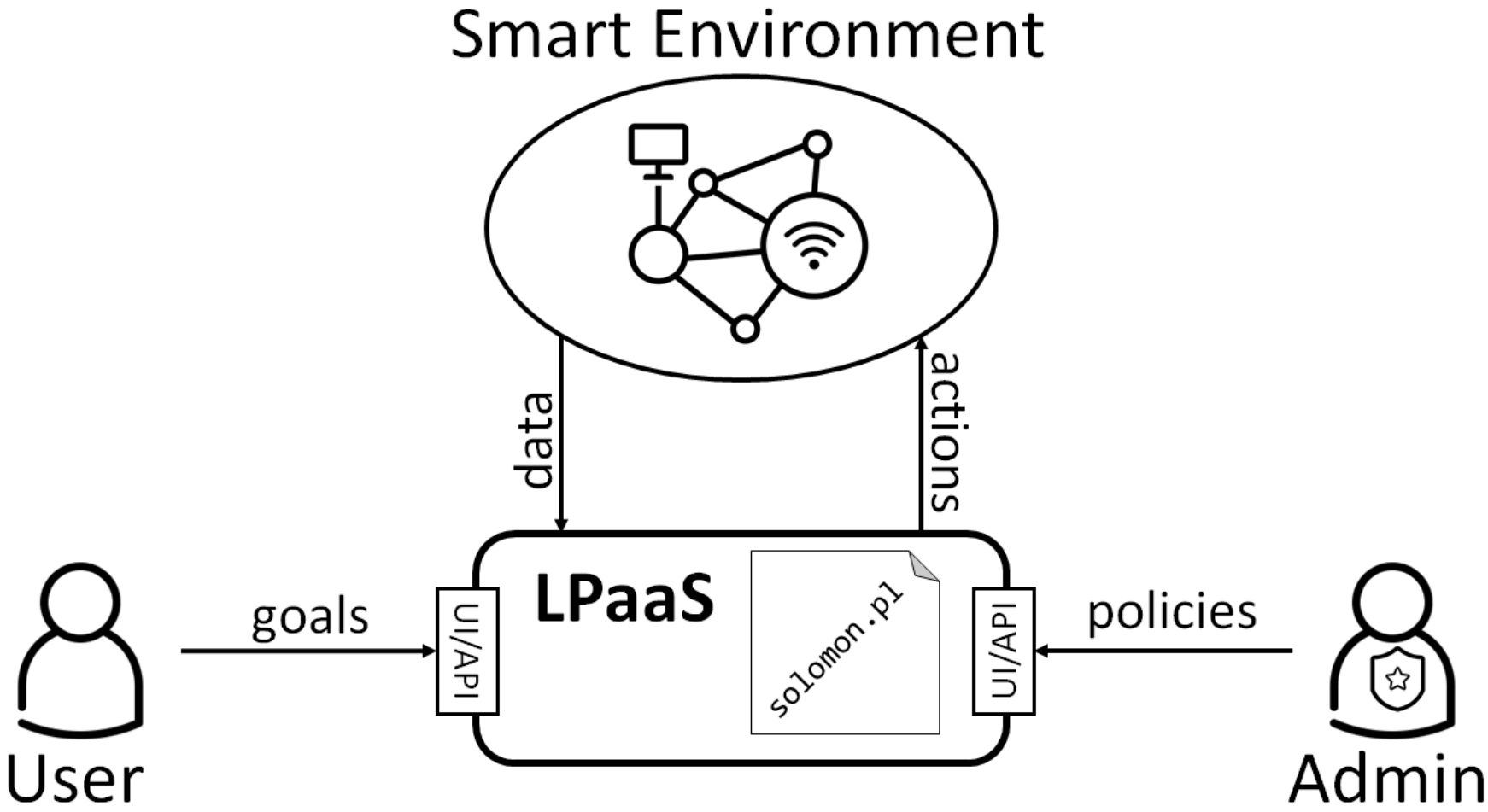
Our Proposal

A **declarative framework** -- and its prototype **Solomon** -- to specify **customisable mediation policies** for reconciling **contrasting goals** and actuator settings in **smart environments**

How?

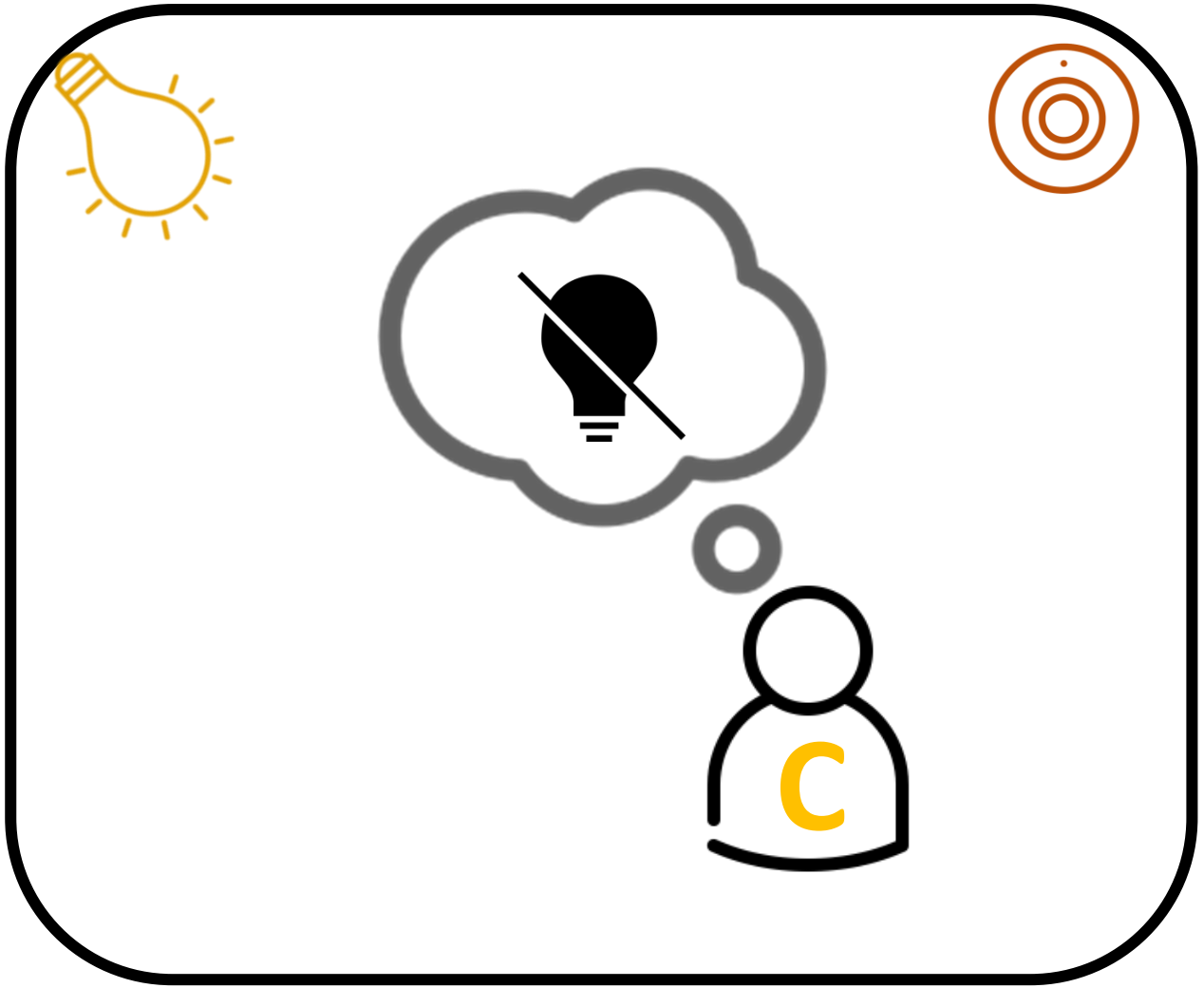
- By **reasoning on a model** of the available **IoT infrastructure** and on (possibly contrasting) goals
- Specifying **ad-hoc mediation policies** for *User-User*, *User-Admin* and *IoT-IoT* conflicts

Our Prototype



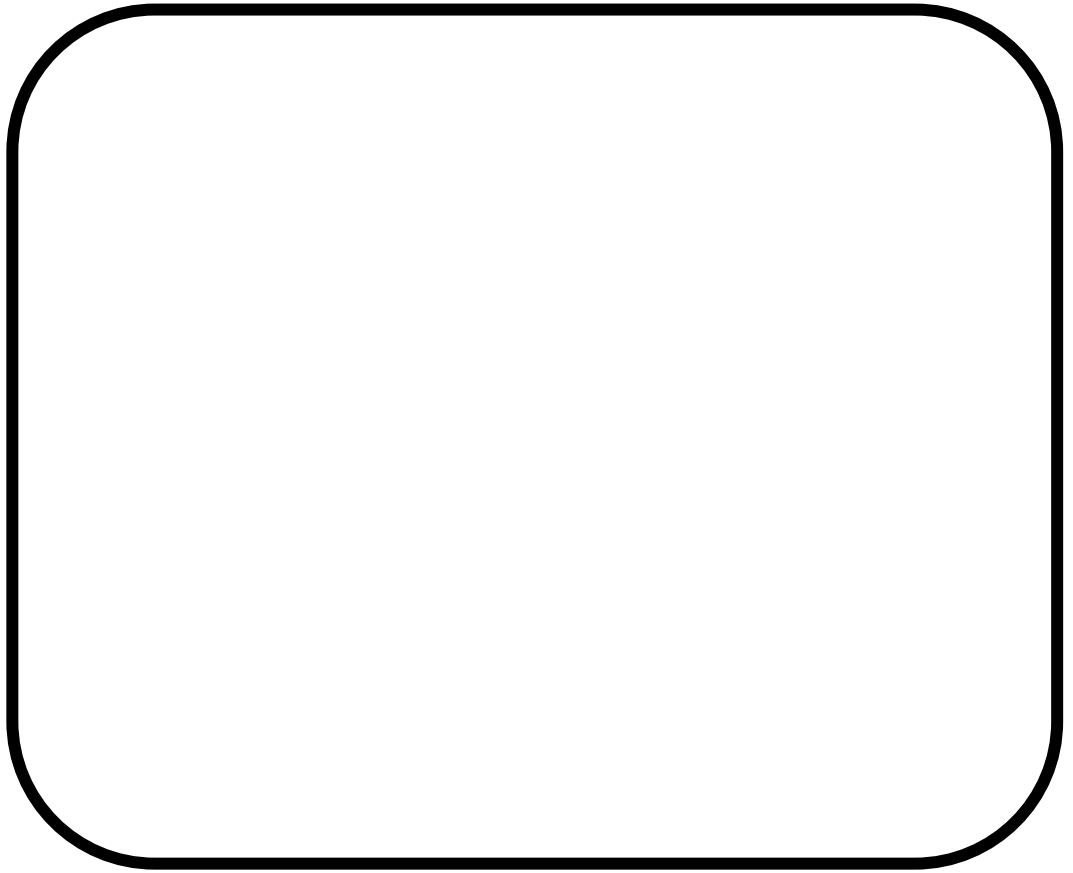
Our **declarative methodology** has been **prototyped in Prolog** and offered **as-a-service** through the **LPaaS paradigm**. The code is **open-source** and available at: https://github.com/di-unipi-socc/Solomon

A Simple Smart Environment



A Simple Smart Environment

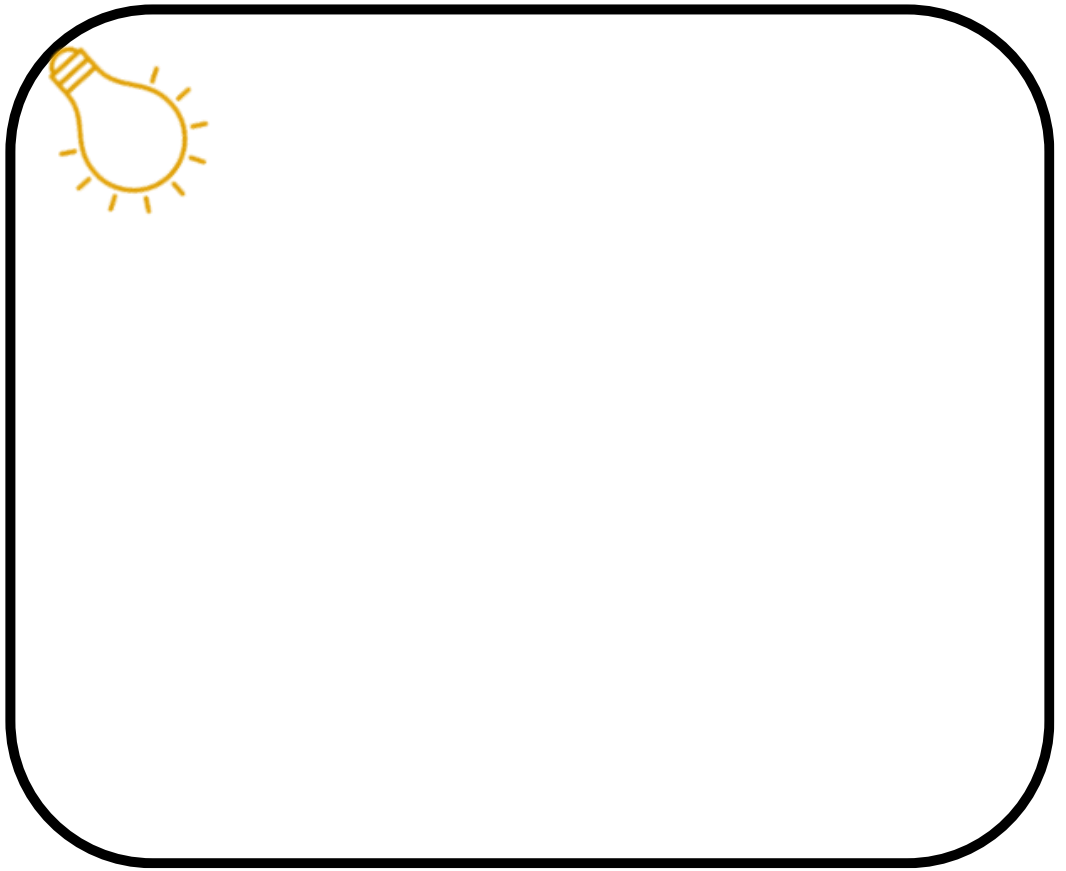
```
propertyType(light).
```



A Simple Smart Environment

```
propertyType(light).

actuator(mainLamp,light).
```



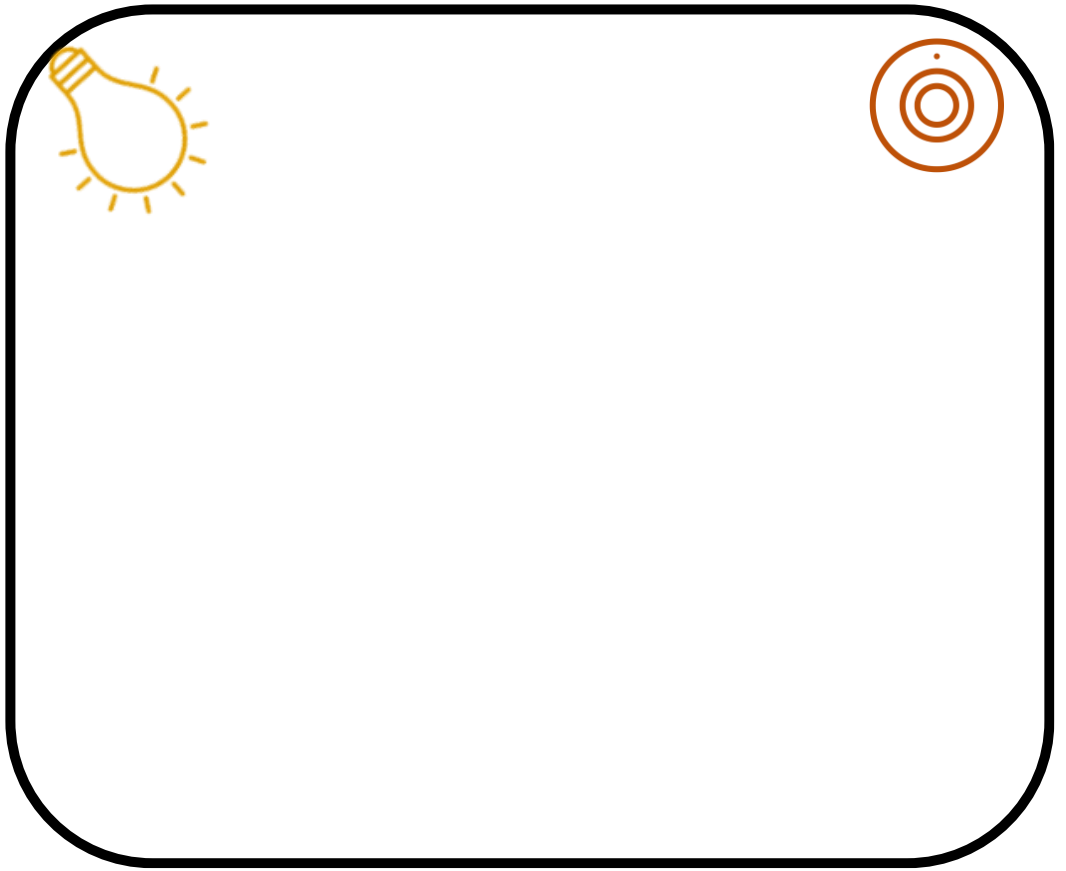
A Simple Smart Environment

```
propertyType(light).

actuator(mainLamp,light).

sensor(brightness, light).

sensorValue(brightness, 90).
```



A Simple Smart Environment

```
propertyType(light).
```

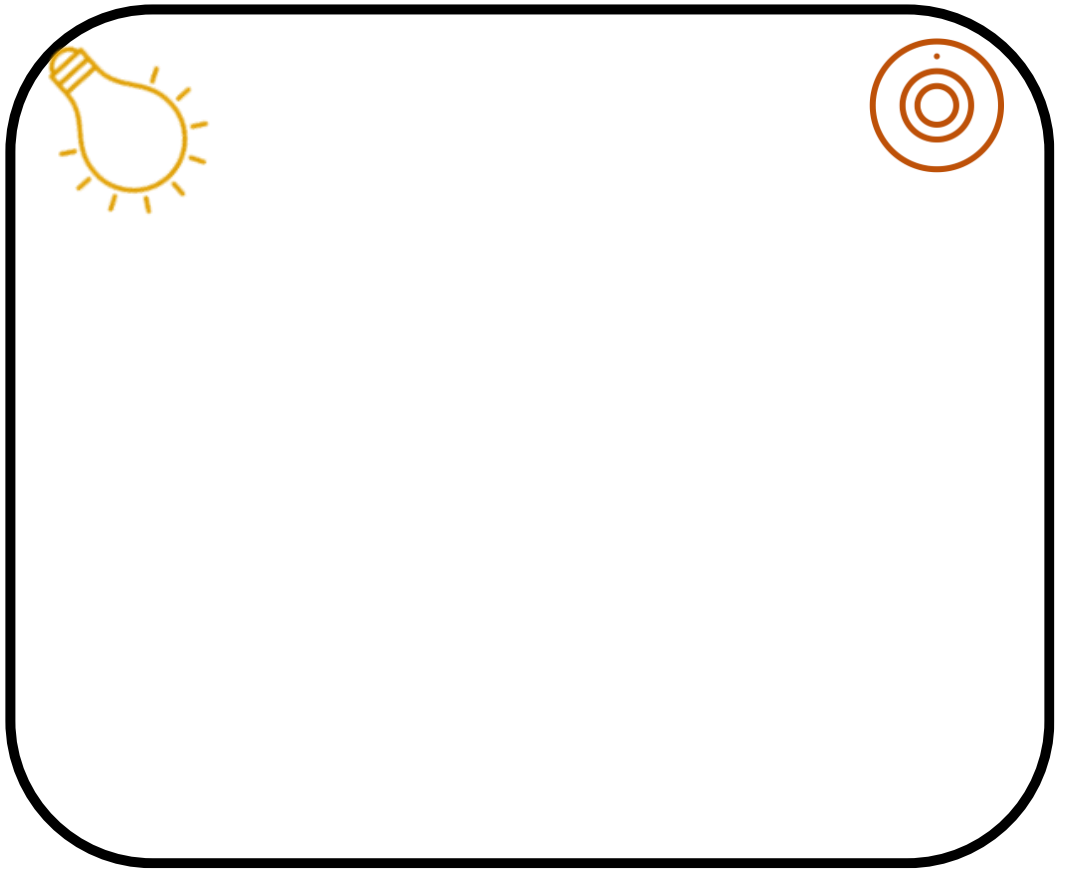
```
actuator(mainLamp,light).
```

```
sensor(brightness, light).
```

```
sensorValue(brightness, 90).
```

```
zone(room2, defaultPolicy).
```

```
propertyInstance(room2, roomLight, light,
        [mainLamp], [brightness]).
```



A Simple Smart Environment

```
propertyType(light).
```

```
actuator(mainLamp,light).
```

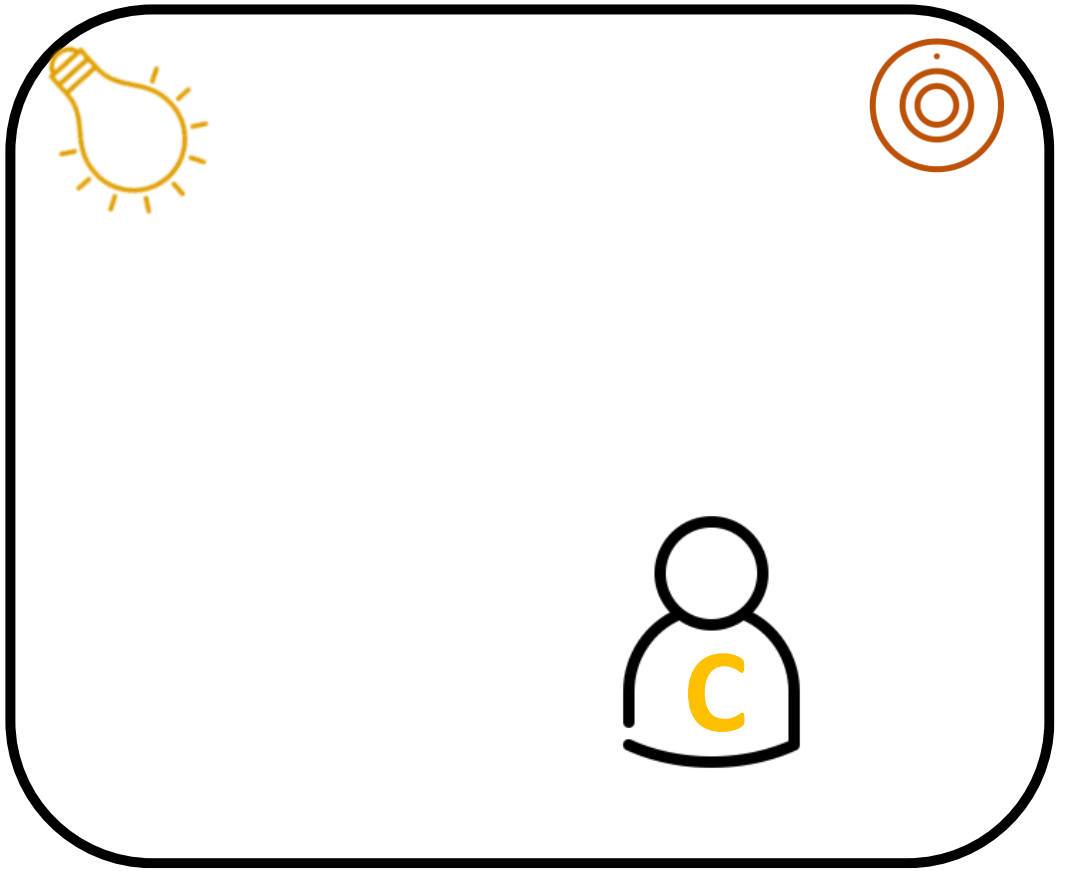
```
sensor(brightness, light).
```

```
sensorValue(brightness, 90).
```

```
zone(room2, defaultPolicy).
```

```
propertyInstance(room2, roomLight, light,
[mainLamp], [brightness]).
```

```
user(userC, [room2]).
```



A Simple Smart Environment

```
propertyType(light).
```

```
actuator(mainLamp,light).
```

```
sensor(brightness, light).
```

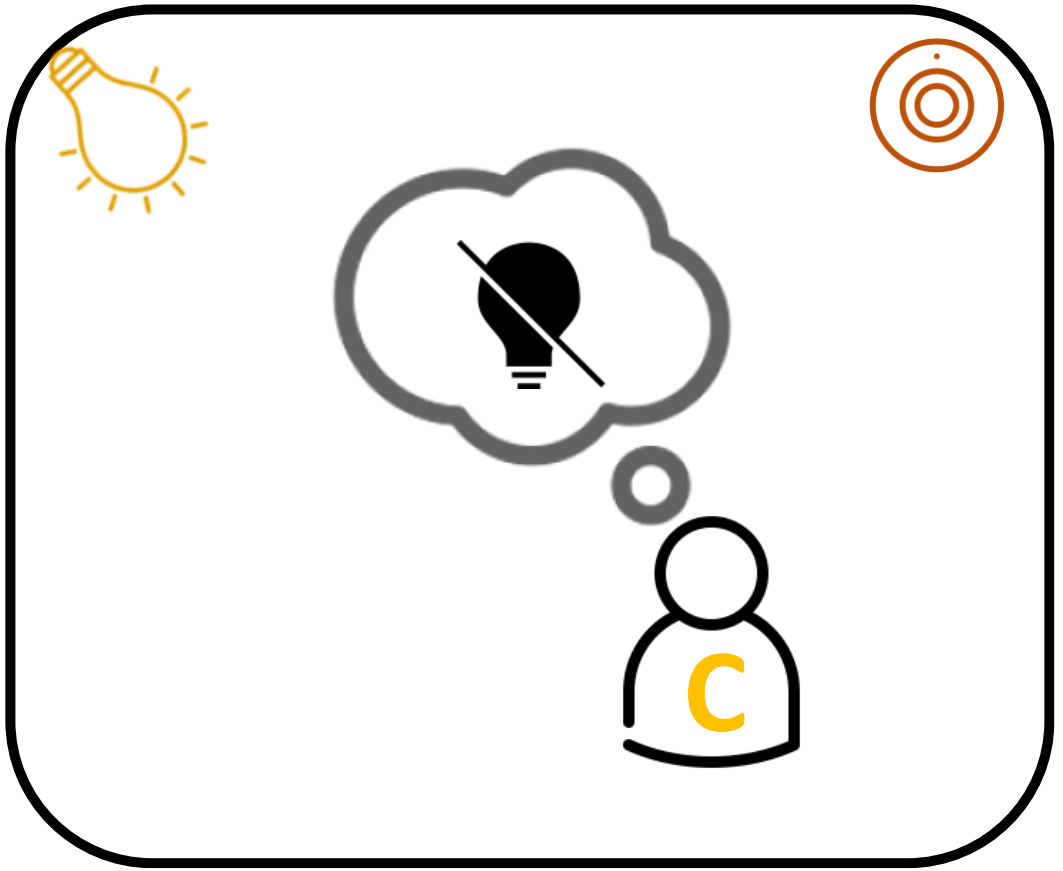
```
sensorValue(brightness, 90).
```

```
zone(room2, defaultPolicy).
```

```
propertyInstance(room2, roomLight, light,
[mainLamp], [brightness]).
```

```
user(userC, [room2]).
```

```
set(userC, room2, roomLight, 15).
```



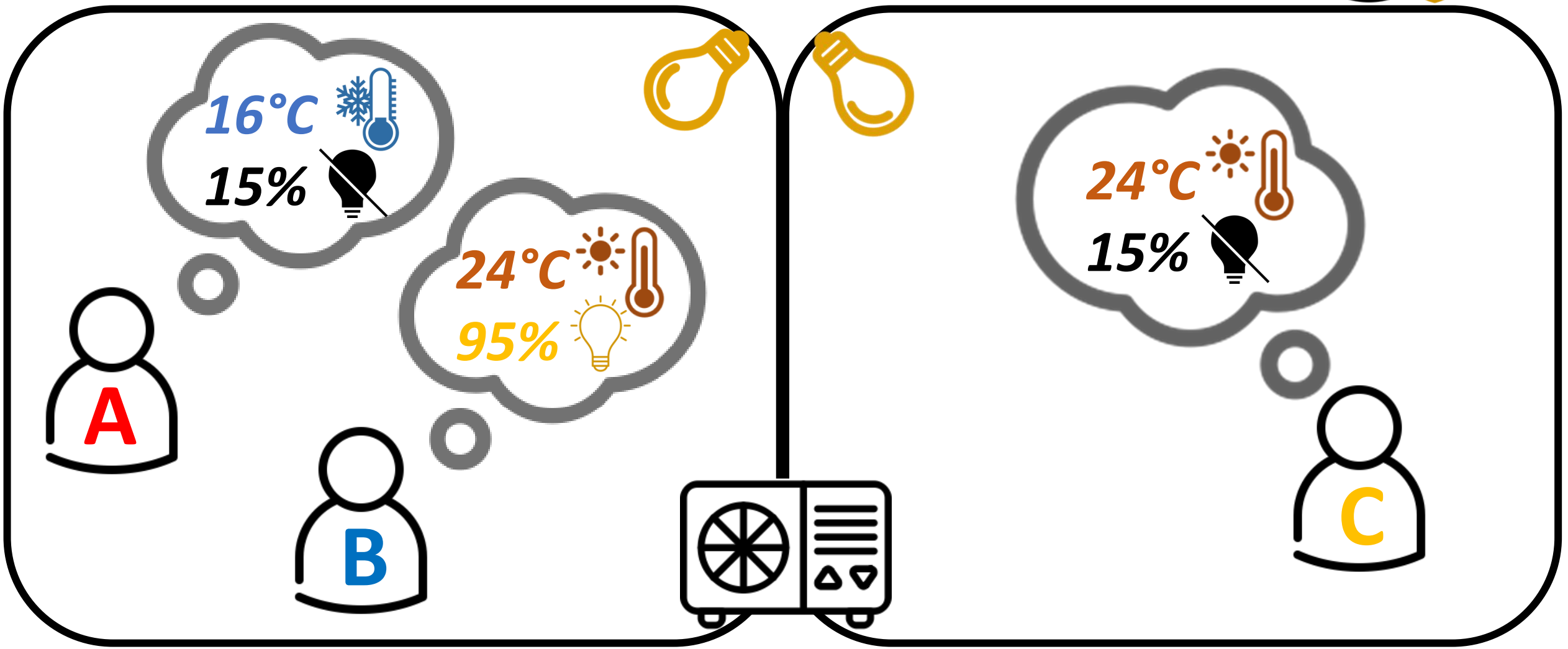
Solomon Functioning

1. **Collecting all user requests**
2. **Mediating the requests**
3. **Determining actions** for **individual IoT actuators**

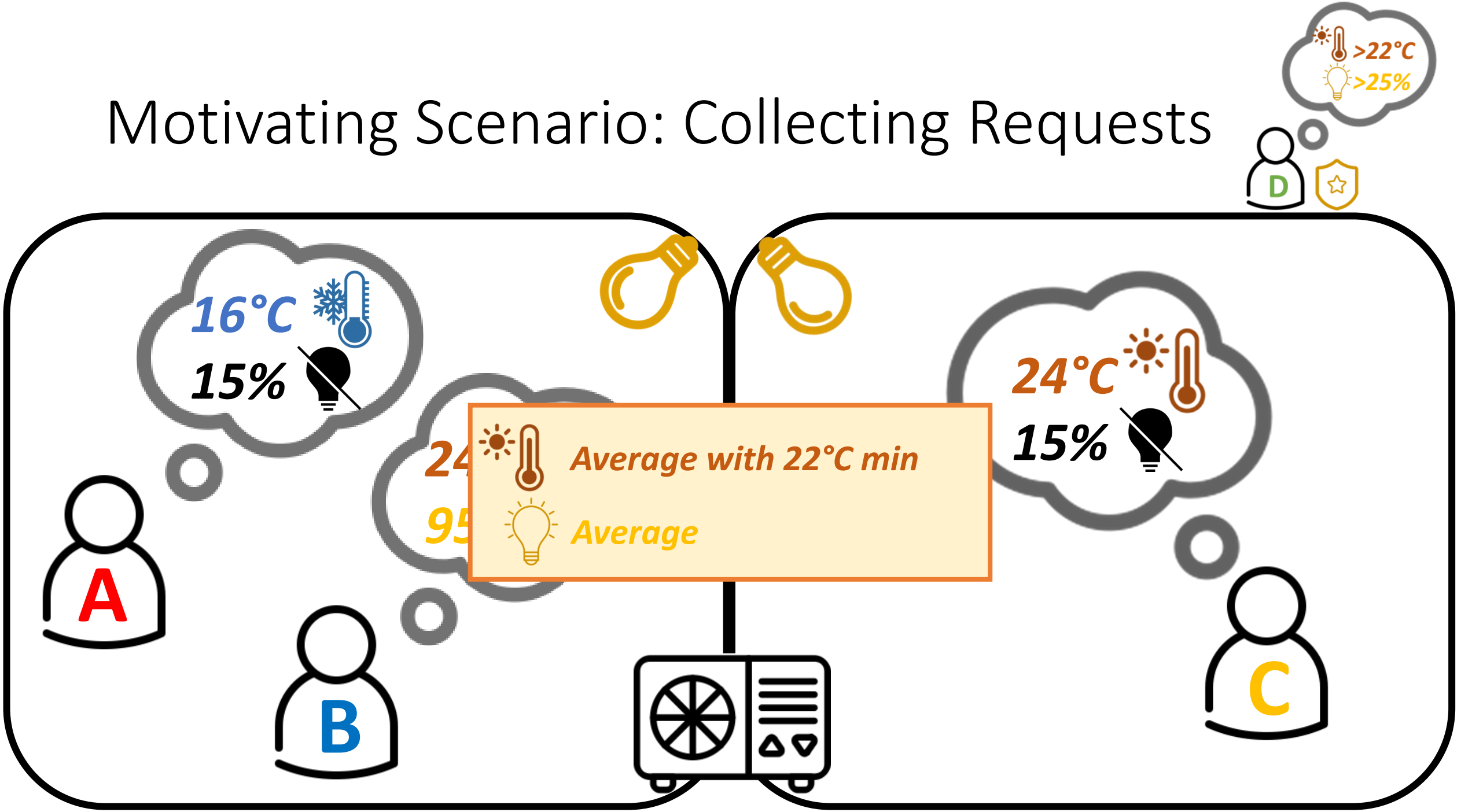
```
react(Requests, MediatedRequests, Actions) :-
    getRequests(Requests, ValidRequests),
    mediateRequests(ValidRequests, MediatedRequests),    % Defined by SysAdmin
    validMediation(MediatedRequests),
    associateActions(MediatedRequests, Actions),         % Defined by SyAdmin
    validActions(Actions).
```

*Solomon also offers a **library of standard predicates** to implement **mediation policies** (e.g. average, consensus, min/max)*

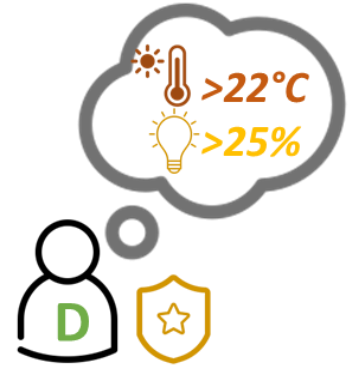
Motivating Scenario: Collecting Requests



Motivating Scenario: Collecting Requests



Motivating Scenario: Mediating Requests



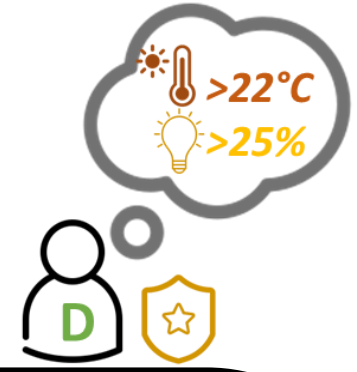
22°C

55%

24°C

15%

Motivating Scenario: Mediating Requests



22°C

55%

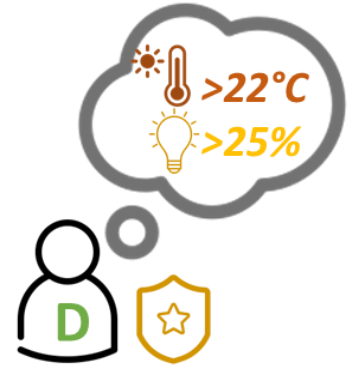
Maximum

Consensus with 25% min

24°C

15%

Motivating Scenario: Determing Actions

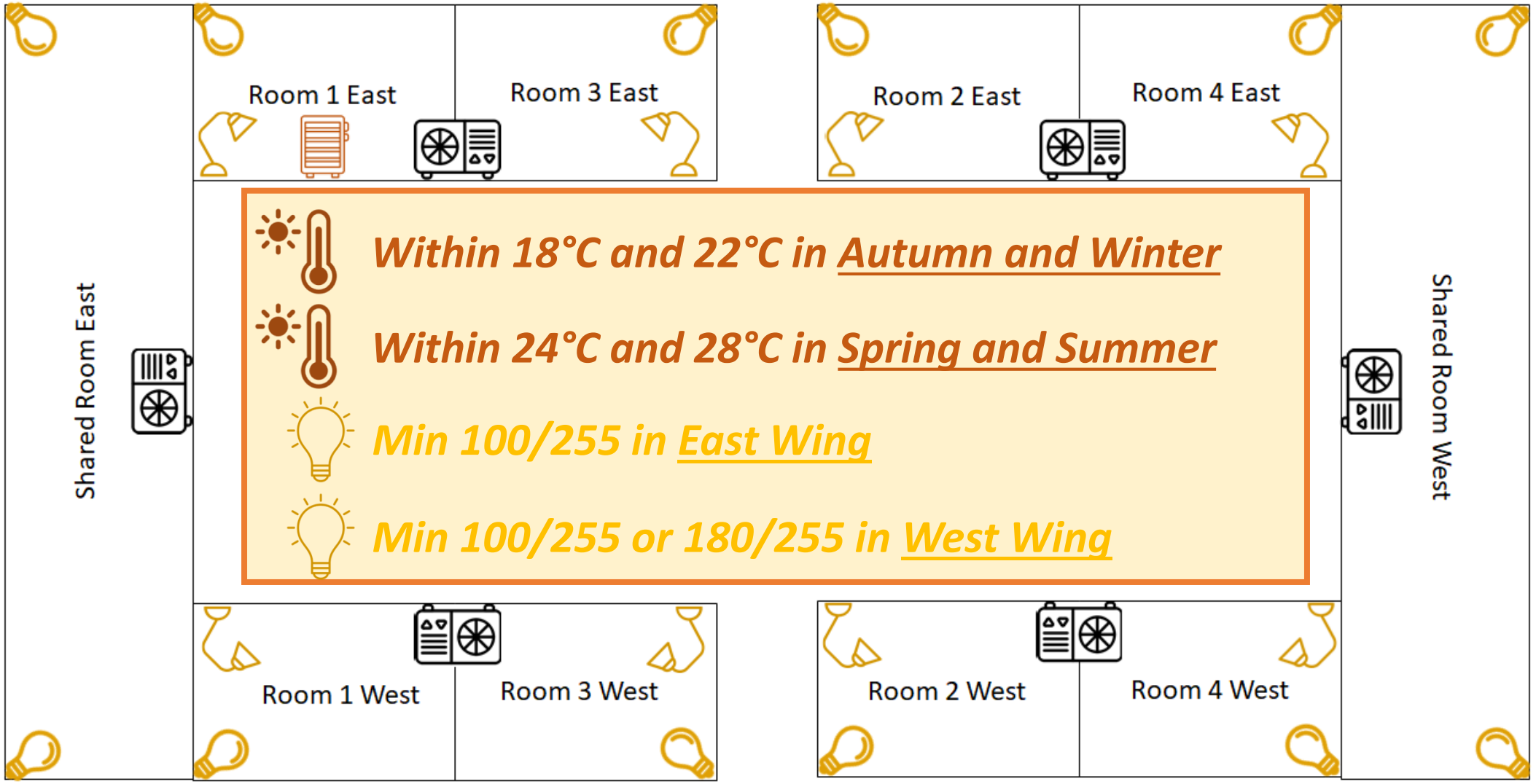


55%

24°C

25%

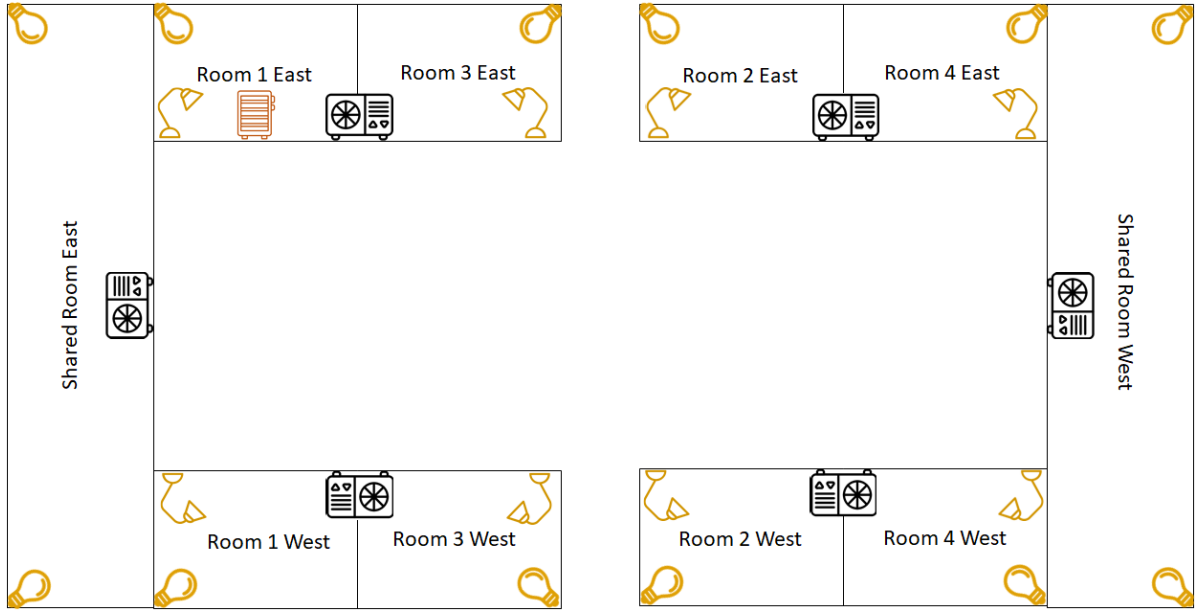
A More Complex Scenario: Smart Building



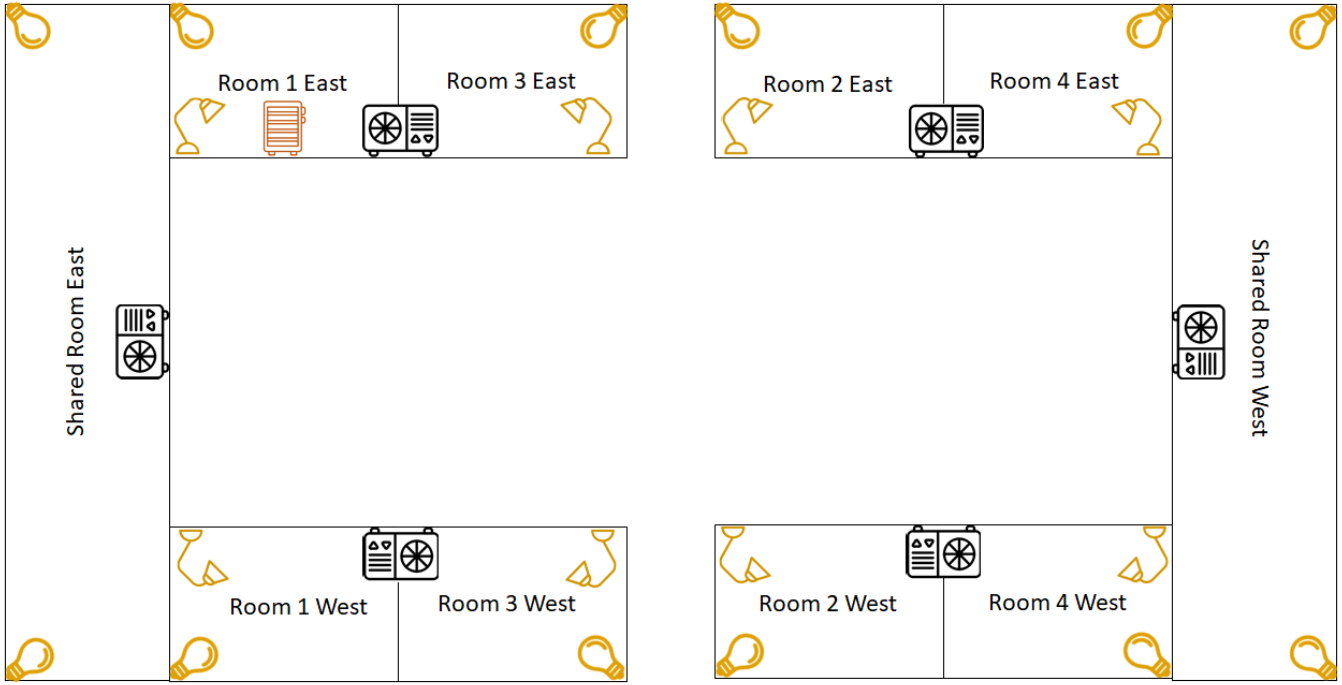
Describing the Smart Building

```
zone(room_E_1, east).
zone(room_E_2, east).
zone(room_E_3, east).
zone(room_E_4, east).
zone(commonRoom_E, east).

zone(room_W_1, west).
zone(room_W_2, west).
zone(room_W_3, west).
zone(room_W_4, west).
zone(commonRoom_W, west).
```



Describing the Smart Building




```
propertyInstance(room_E_1,roomTemp,temp,[acOdd_E,heater],[tempOdd_E]).

propertyInstance(room_E_1,roomLight,light,[biglightRoom_E_1,smalllightRoom_E_1],
[lightRoom_E_1]).

propertyInstance(room_E_3,roomTemp,temp,[acOdd_E],[tempOdd_E]).

propertyInstance(room_E_3,roomLight,light,[biglightRoom_E_3,smalllightRoom_E_3],[lightRoom_E_
3]).
```

mediateRequests/2


```
mediateRequests(Requests, Mediated) :-
    groupPerPI(Requests, NewRequests), mediateRequest(NewRequests, Mediated).

mediateRequest([],[]).
mediateRequest([(Z,PI,Ls)|Reqs], [Mediated|OtherMediatedReqs]) :-
    mediatePI(Z,PI,Ls,Mediated), mediateRequest(Reqs, OtherMediatedReqs).

mediatePI(_, _, [], undef).
mediatePI(Z, PI, Ls, (Z, PI, Avg)) :-
    findall(V, member((V,_),Ls), Values), avg(Values,AvgTmp),
    zone(Z, Policy), propertyInstance(Z, PI, Prop, _, [Sensor]),
    sensorValue(Sensor, SensedValue),
    findValue(Policy, Prop, SensedValue, AvgTmp, Avg).
```

mediateRequests/2


```
findValue(_, temp, _, TempValue, Value) :-
    season(S),
    (((S = winter ; S = autumn), (TempValue > 22, Value is 22; TempValue < 18, Value is 18;
Value is TempValue));
    ((S = summer ; S = spring), (TempValue > 28, Value is 28; TempValue < 24, Value is 24;
Value is TempValue))).


findValue(east, light, _, LightValue, Value) :-
    (LightValue > 255, Value is 255; LightValue < 100, Value is 100; Value is LightValue).


findValue(west, light, Brightness, LightValue, Value) :-
    ((Brightness > 100, (LightValue > 255, Value is 255; LightValue < 100, Value is 100;
Value is LightValue));
   (LightValue > 255, Value is 255; LightValue < 180, Value is 180; Value is LightValue)).
```

associateActions/2


```
associateActions(Requests, ExecutableActions) :-
    actionsFor(Requests, Actions), setActuators(Actions, ExecutableActions).


actionsFor([],[]).
actionsFor([undef|Reqs], Actions) :- actionsFor(Reqs, Actions).
actionsFor([(Z, PI, V)|Reqs], Actions) :-
    propertyInstance(Z, PI, _, ActuatorList, SensorList),
    selectActionsForPI(Z, PI, V, ActuatorList, SensorList, Actions1),
    actionsFor(Reqs, Actions2), append(Actions1, Actions2, Actions).


selectActionsForPI(_, _, V, ActuatorList, _, Actions) :-
    length(ActuatorList, ActuatorsNumber),
    triggerAllActuators(V, ActuatorsNumber, ActuatorList, Actions).


setActuators(Actions, ExecutableActions) :- setActuatorsWithMax(Actions, -inf,inf,
ExecutableActions).
```

associateActions/2


```
triggerAllActuators(_, _, [], []).


triggerAllActuators(V, ActuatorsNumber, [Actuator|ActuatorList],
[(Actuator,VNew)|Actions]) :-
    dif(Actuator, heater),
    VNew is V/ActuatorsNumber,
    triggerAllActuators(V, ActuatorsNumber, ActuatorList, Actions).


triggerAllActuators(V, ActuatorsNumber, [heater|ActuatorList],
[(heater,100)|Actions]) :-
    V > 0, triggerAllActuators(V, ActuatorsNumber, ActuatorList, Actions).


triggerAllActuators(V, ActuatorsNumber, [heater|ActuatorList],
[(heater,0)|Actions]) :-
    V =< 0, triggerAllActuators(V, ActuatorsNumber, ActuatorList, Actions).
```

A Working Example: Determining Actions


```
season(winter).
user(u1, [room_E_1,commonRoom_E,commonRoom_W]).
user(u3, [room_E_3,commonRoom_E,commonRoom_W]).
user(u8, [room_W_4,commonRoom_E,commonRoom_W]).
set(u1, room_E_1, roomLight, 0).
set(u1, room_E_1, roomTemp, 18).
set(u3, room_E_3, roomTemp, 28).
set(u4, room_E_2, roomLight, 0).
set(u4, room_E_2, roomTemp, 18).
```

```
sensorValue(lightCommonRoom_W, 160).
user(u2, [room_E_2,commonRoom_E,commonRoom_W]).
user(u4, [room_E_4,commonRoom_E,commonRoom_W]).

set(u2, commonRoom_W, commonRoomLight, 255).
set(u2, commonRoom_W, commonRoomTemp, 23).
set(u8, commonRoom_W, commonRoomLight, 255).
set(u8, commonRoom_W, commonRoomTemp, 18).
```

```
MediatedRequest = [(room_E_1,roomLight,100), (room_E_1,roomTemp,18),
(room_E_3,roomTemp,22),(commonRoom_W,commonRoomLight,255), (commonRoom_W,commonRoomTemp,20.5)].
```

```
Actions = [(acCommonRoom_W, 20.5), (biglightCommonRoom_W_1, 127.5),(biglightCommonRoom_W_2,
127.5), (acOdd_E, 22), (heater, 100), (biglightRoom_E_1, 50), (smalllightRoom_E_1, 50)].
```

Conclusions

goal-driven

It **considers** and **mediates** among them **goals**, from all (human and machine) **stakeholders** involved in a Smart Environment, to reach a **target state**

customisable

Being **open-source** and **enabling customisation** from its end-users. **Code** and **Docs** at: https://github.com/di-unipi-socc/Solomon

As it is **Prolog** code: **concise** (around **50 sloc**) and featuring a good level of **abstraction** and **flexibility** to **accommodate new emerging needs of Smart Environments**

declarative

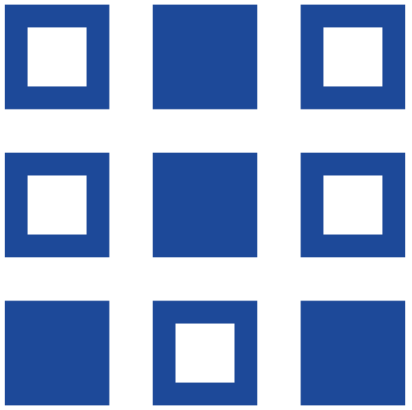
As it features a **well-defined REST API** based on **LPaaS**, it enables **interoperability** with other systems through **remote interactions**

as-a-Service

Future Work

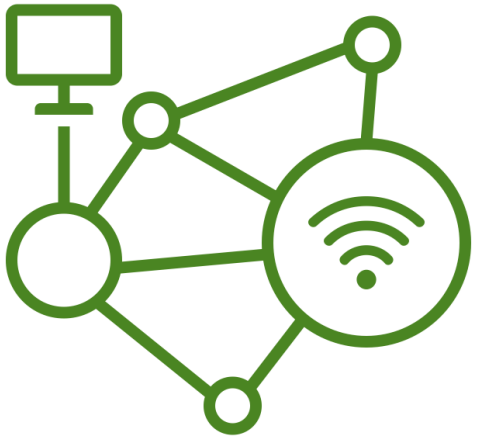
New Policies

by also proposing a set of modular building blocks



Answer Set Programming

to allow processing more expressive policies



Web of Things

to exploit Solomon in actual smart environments

A Declarative Goal-oriented Framework for Smart Environments with LPaaS

Giuseppe Bisicchia, Stefano Forti and Antonio Brogi

giuseppe.bisicchia@outlook.it

Service-oriented, Cloud and Fog Computing Research Group

Department of Computer Science

University of Pisa, Italy

36th Edition of the Italian Conference on Computational Logic, 7-9 September 2021