An Application Conflict Detection and Resolution Method for Smart Homes

Miki Yagita
1st year master's course
@ The University of Tokyo, Japan

Supervisers:
Assoc. Prof. Fuyuki Ishikawa, Prof. Shinichi Honiden

IoT – Internet of Things

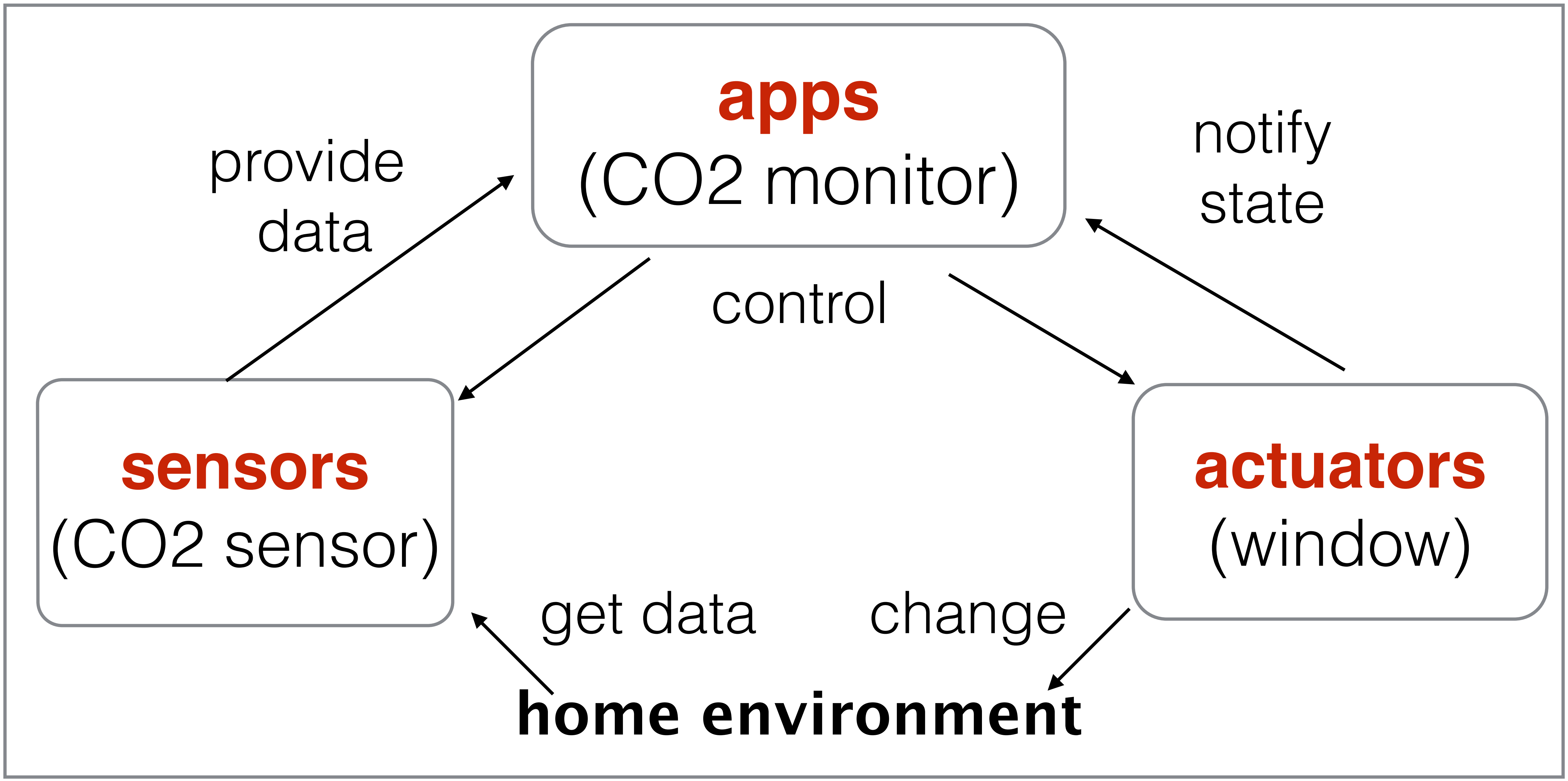
A relatively new concept: “**inter-connecting devices that were not connected before.**”

U.S. National Intelligence Council chooses IoT to be **one of the six technologies that will most influence the world by 2025.**

One application of IoT is the **Smart Homes**.

[1] National Intelligence Council, Disruptive Civil Technologies April 2008, http://www.fas.org/irp/nic/disruptive.pdf

Smart Home


apps
(CO2 monitor)
provide data
notify state
control
sensors
(CO2 sensor)
actuators
(window)
get data
change
home environment

App conflict

Generally, there are multiple apps installed

→ conflict between apps

Two types of App conflict

- Sensor control conflict
- Actuator control conflict

App conflict

Generally, there are multiple apps installed

→ conflict between apps

Two types of App conflict

- Sensor control conflict : relatively easy to solve
- **Actuator control conflict**

Our work focuses on this type of conflict

Example: Actuator control conflict

CO2Monitor_App

what: opens window1

when: status of CO2_Sensor is "high"

secureWindow_App

what: closes window1

when: status of the residents are "Away" or "Asleep"

→ possible conflict regarding window1 (actuator)

Related Work: DepSys[8]

Actuator conflict detection/resolution at install-time

Problems:

1. Only apps that run synchronously (with time) are supported
2. Creates a total order between apps which is not flexible
3. Does not specify how the app priority is created

→ **Our system overcomes these three limitations**

[8] Munir et al., “DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes” ICCPS’14

Problem (1)

No conflict detection/resolution for apps that run asynchronously

CO2Monitor_App

what: opens window1

when: the status of CO2_Sensor is "high"

Notice that even this simple app is operating asynchronously

Problem (2)

Creating a total order between all apps does not provide a flexible resolution of conflicts.

CO2Monitor_App

what: opens window1

when: the status of CO2_Sensor is “high”

secureWindow_App

what: closes window1

when: the status of the residents are “Away” or “Asleep”

Users may want:

residents “Away” → secureWindow_App > CO2Monitor_App

residents “Asleep” → secureWindow_App < CO2Monitor_App

Approach

With metadata of **apps**, **actuators**, and **sensors**, check **each situations of conflicts** by **model-checking**

→ supports asynchronous apps (sol. to prob. 1) by using **model-checking**

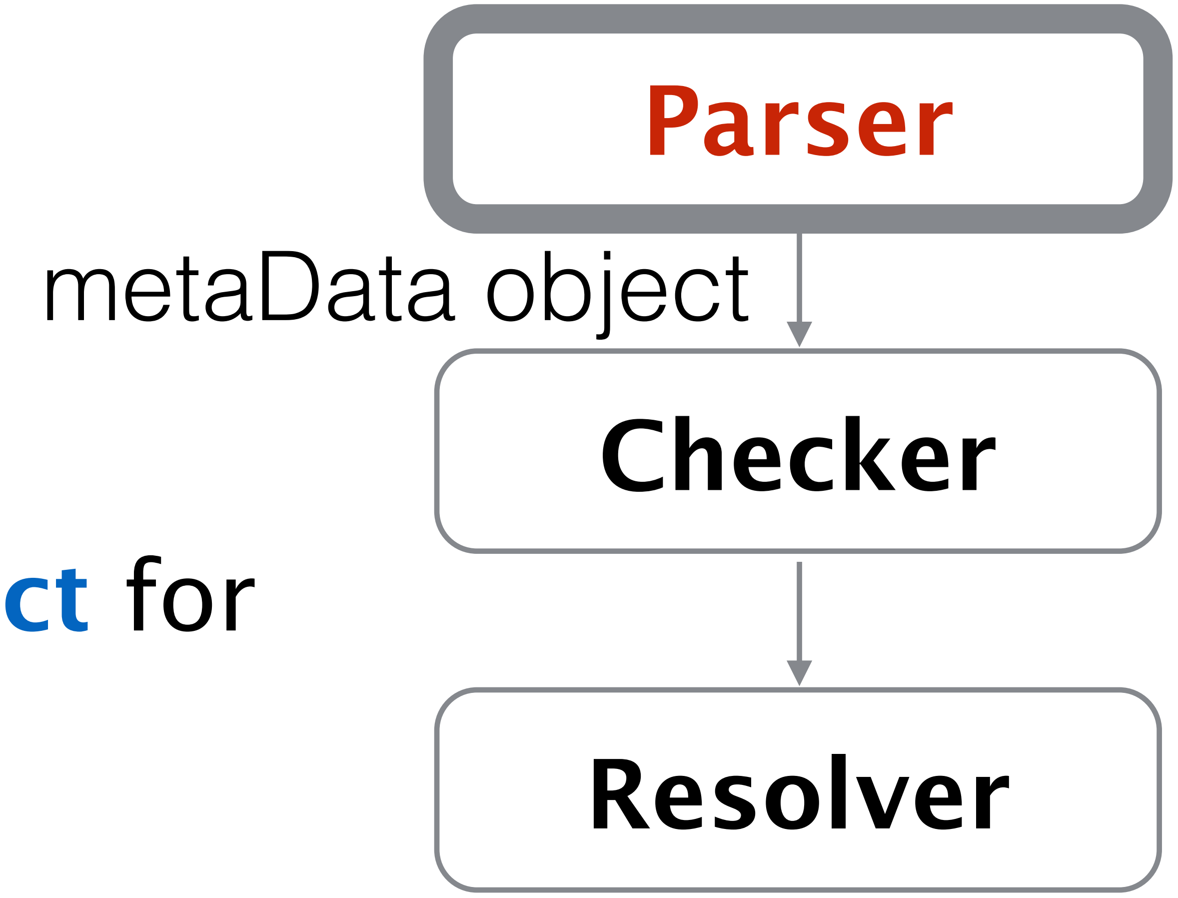
→ a flexible conflict resolution (sol. to prob. 2) by using **situations** (explained later) **of conflicts**

System Overview

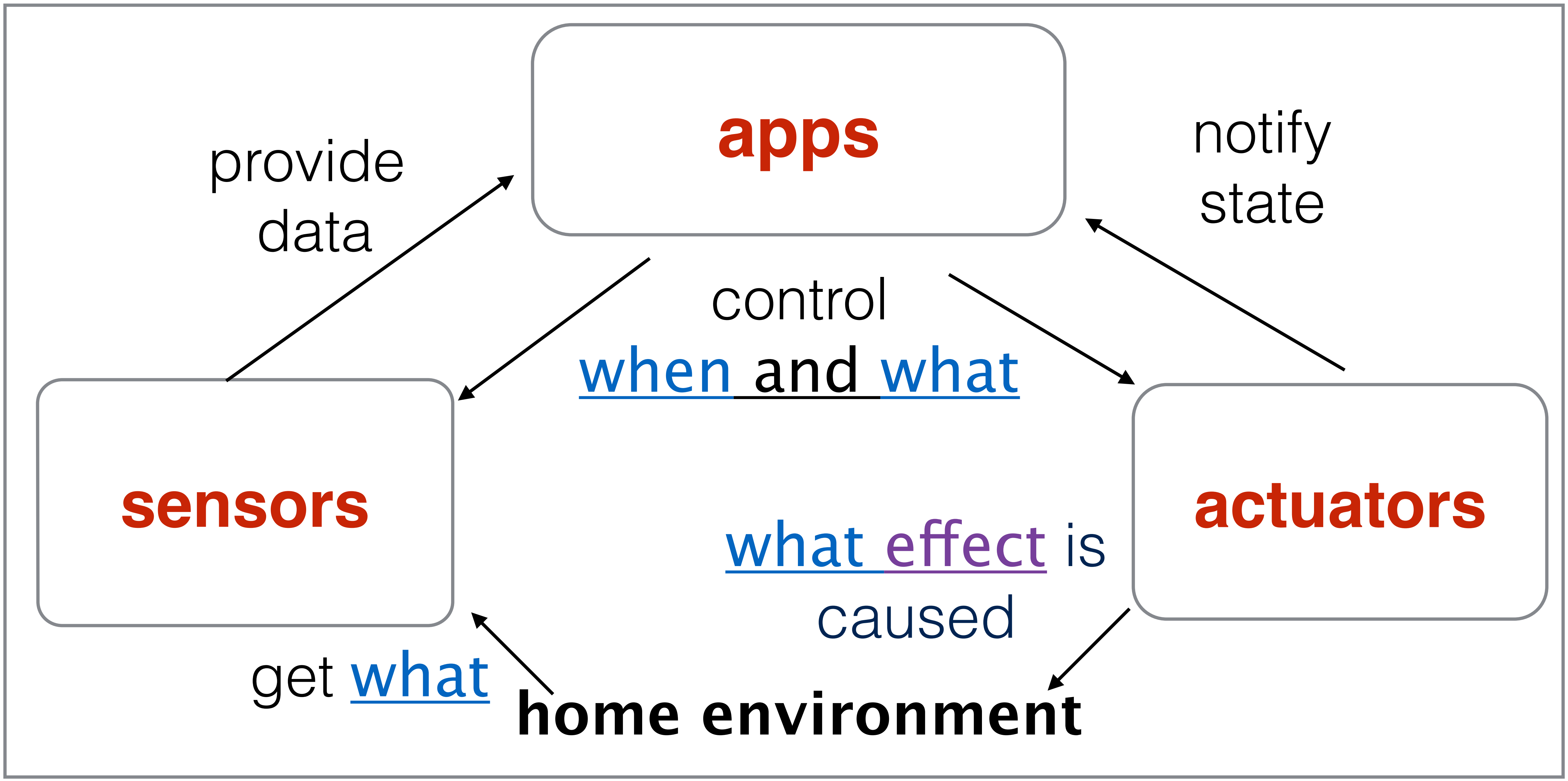
The Parser module:

Inputs metadata of apps, actuators, and sensors

Outputs a **metaData object** for further use in the system



‘metaData’ object



‘effect’

An effect is how actuator affects the environment

a **direct** effect

e.g.) “heater = On” → “Temperature = Higher”

a **non-direct** effect

e.g.) “heater = On” → “Temperature = Higher”

→ “Humidity = Lower” i.e. highschool physics

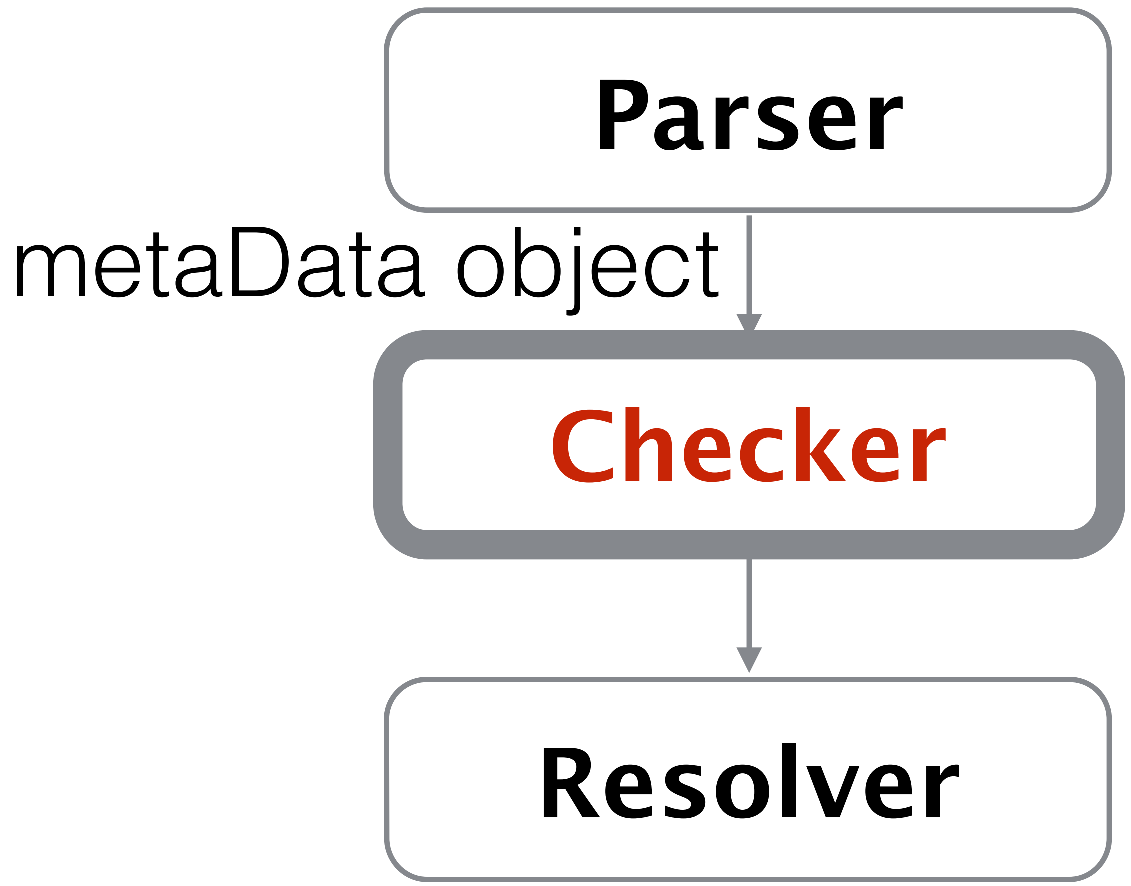
→ we only take into account direct effects

System Overview

The Checker module:

Inputs the **metaData object**

Outputs one conflict from each **situation**

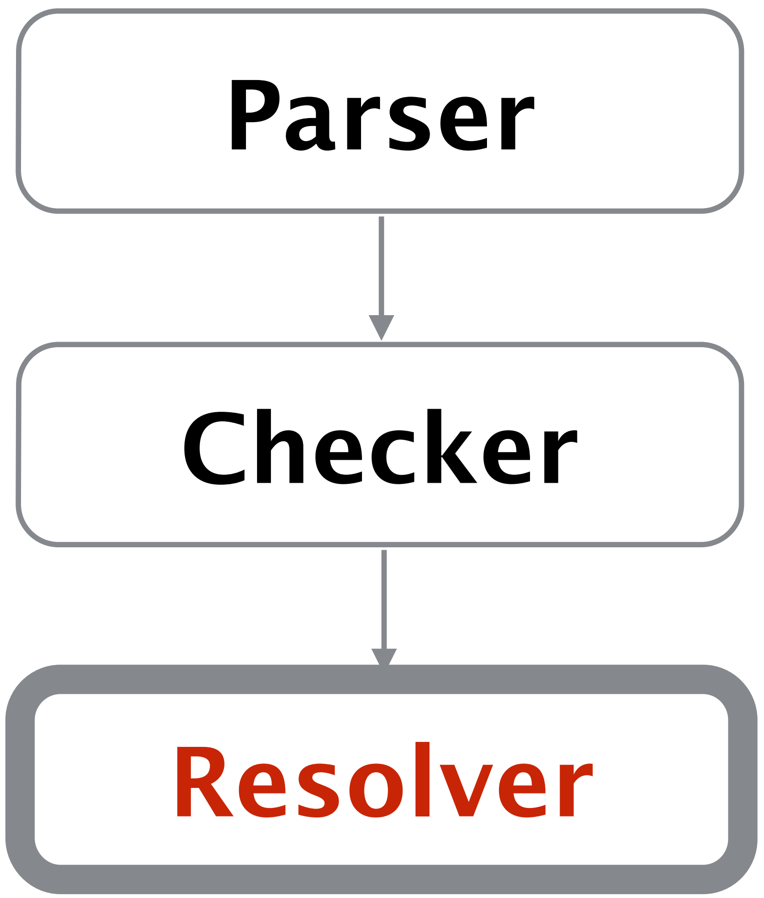


System Overview

The Resolver module:

Inputs one conflict from each **situation**

Outputs queries for the users



‘situation’

situations: equivalence classes to categorise actuator conflicts

1. which app was running first

   → especially: when priority of two apps are same/similar

2. why did the app run

   → secureWindow_App closes window1 when the status of the residents are **“Away”** or **“Asleep”**

“Away” and **“Asleep”** create two different situations

Evaluation (1)

Through implementation (by hand) of the following case:

CO2Monitor_App

what: opens window1

when: the status of CO2_Sensor is “high”

secureWindow_App

what: closes window1

when: the status of the residents are “Away” or “Asleep”

Evaluation (2)

Some situations are as follows:

1. CO2Monitor_App controlling window1 → residents go “Away”
2. CO2Monitor_App controlling window1 → residents go “Asleep”

Created LTLs for above two situations

→ model checking with SPIN [9]

→ conflicts found within 0.01 seconds.

From ‘trail’ of model-checker :

> “When residents become Away while CO2Monitor_App is opening window1, there will be a conflict between that and secureWindow_App. Which app do you want to prefer?”

Conclusion

- We proposed the use of model checking in order to detect more conflicts
- By using situations our system allows a more flexible resolution of conflicts

Future Work:

- Support of indirect effects in conflict detection and resolution
- Evaluation with a larger test case

(We are doing this now!)

additional slides

References

[2] P. a. Vicaire, E. Hoque, Z. Xie, and J. a. Stankovic, “Bundle: A group-based programming abstraction for cyber-physical systems,” IEEE Transactions on Industrial Informatics, vol. 8, no. 2, pp. 379–392, 2012

[3] P. Vicaire and Z. Xie, “Physicalnet: A generic framework for managing and programming across pervasive computing networks,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, 2010, pp. 269–278.

[4] C. Dixon, R. Mahajan, S. Agarwal, A. J. B. Bongshin, L. Stefan, and S. Paramvir, “An Operating System for the Home,” NSDI, vol. 7, 2012.

[5] A.D.Wood,J.a.Stankovic,G.Virone,L.Selavo,Z.He,Q.Cao,T.Doan, Y. Wu, L. Fang, and R. Stoleru, “Context-aware wireless sensor networks for assisted living and residential monitoring,” IEEE Network, vol. 22, no. 4, pp. 26–33, 2008.

[6] R. Dickerson, E. Gorlin, and J. Stankovic, “Empath: a continuous remote emotional health monitoring system for depressive illness,” in Wireless Health 2011, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2077552

[7] M. L. Mazurek, J. P. Arsenault, J. Bresee, N. Gupta, I. Ion, C. Johns, D. Lee, Y. Liang, J. Olsen, B. Salmon, R. Shay, K. Vaniea, L. Bauer, L. F. Cranor, G. R. Ganger, M. K. Reiter, and Z. Eth, “Access Control for Home Data Sharing : Attitudes , Needs and Practices,” in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2010, pp. 645–654.

[8] Munir et al., “DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes” ICCPS’14

Problem (3)

Many existing works on self adaptive systems are a generalisation of the actuator conflict problem.

However, to our best knowledge, there has been no previous work on assisting users' resolution of conflicts by providing useful information.

Existing Approach (1)

Bundle [2] and Physicalnet [3]:

- actuator conflicts are resolved with the “Resolver”
- The “Resolver” is a Java method, and can be freely modified

→ A resolver written as a Java method can be used by programmers but not by general users

→ Our approach allows general users to resolve actuator conflicts

Existing Approach (2)

HomeOS [4], AlarmNet [5], Empath [6]:

- architectures for Smart Homes
- actuator conflicts are not resolved at install-time

  → why not install time?

HomeOS claims: “Studies show that users prefer this flexibility (permit or deny access interactively) **rather than having to specify all possible legal accesses a priori** [7].” [4]

→ However, users may not always be at home to resolve the conflict at run-time, so in some cases install-time conflict resolution is necessary.

Trigger

A trigger is what causes an app to run

trigger ::= <time> | <event>

A time object supports synchronous apps

An event object supports asynchronous apps

eg.) AND ({ sensorId = CO2_Sensor, sensorData = 0.1ppm,
comparator = HigherThan },
{ actuatorId = Window1,
actuatorEffect = { effect = { window = closed },
location = livingRoom } })