Politecnico di Torino
Dipartimento di
Automatica e Informatica
Torino, Italy

e-Lite
http://elite.polito.it

Formal Verification of Device State Chart Models

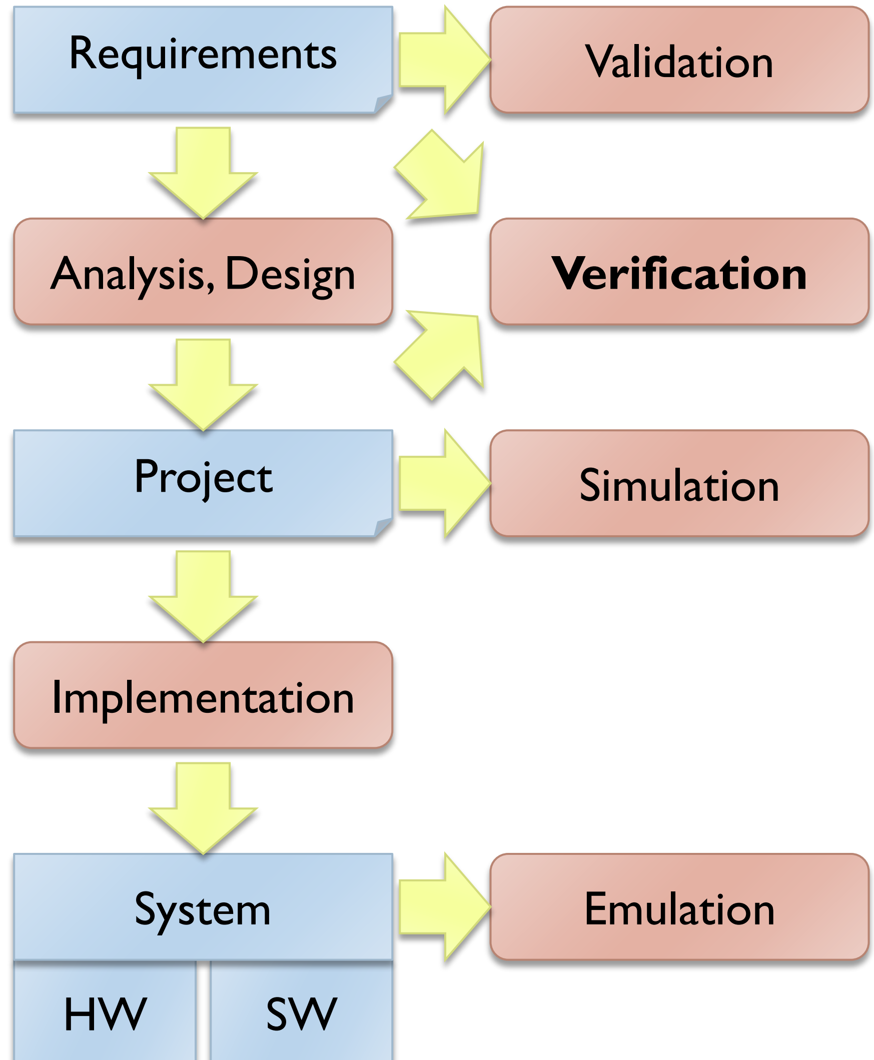
Fulvio Corno, Muhammad Sanaullah

Outline

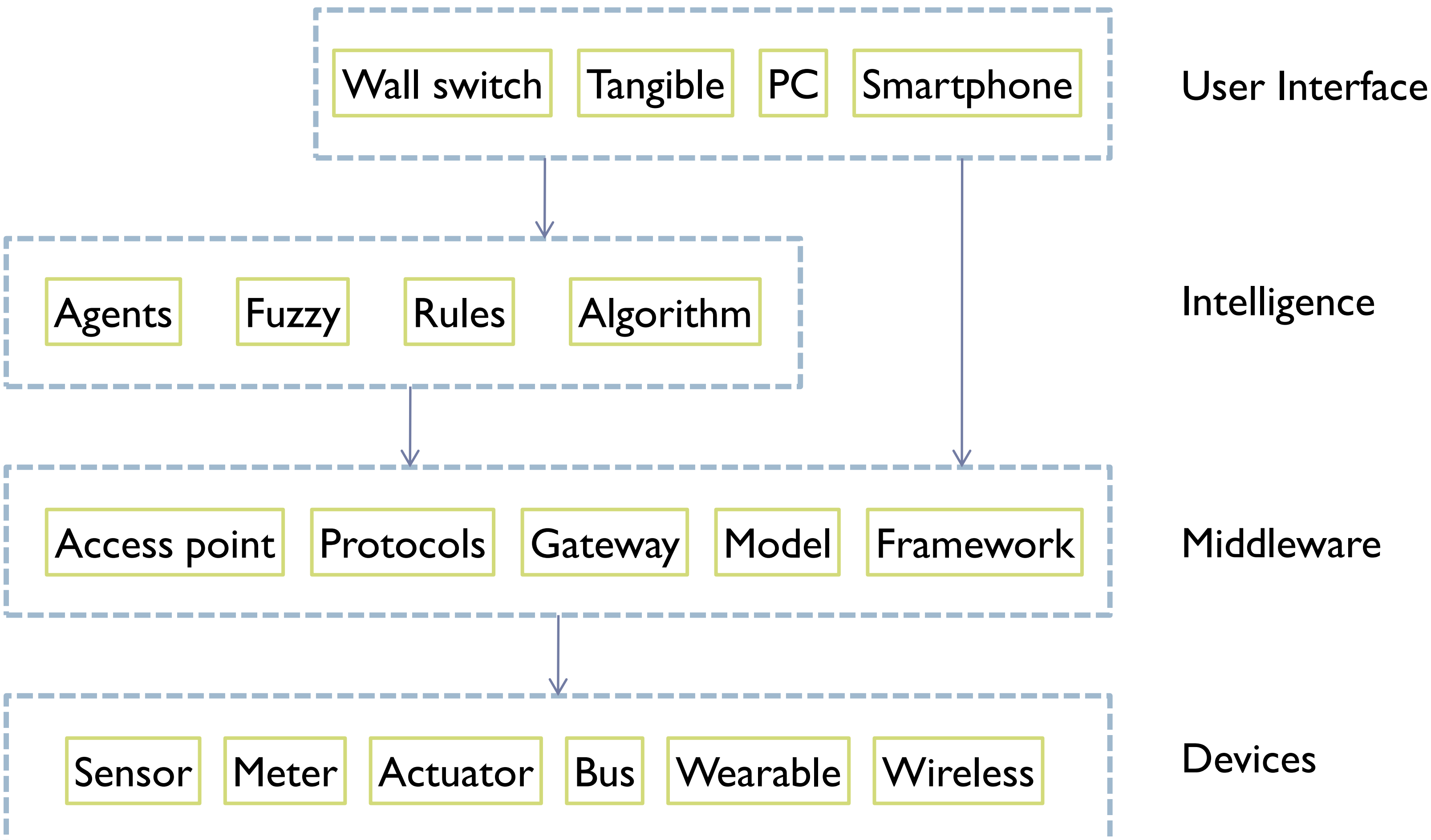
- Design process
- Formalisms
- Verification Methodology
- Results
- Conclusions

Design process for IE

- Intelligent environments gaining acceptance
  - More installations
  - Standard solutions
- Need more structured design process
  - Less "art"
  - More "engineering"



Reference model



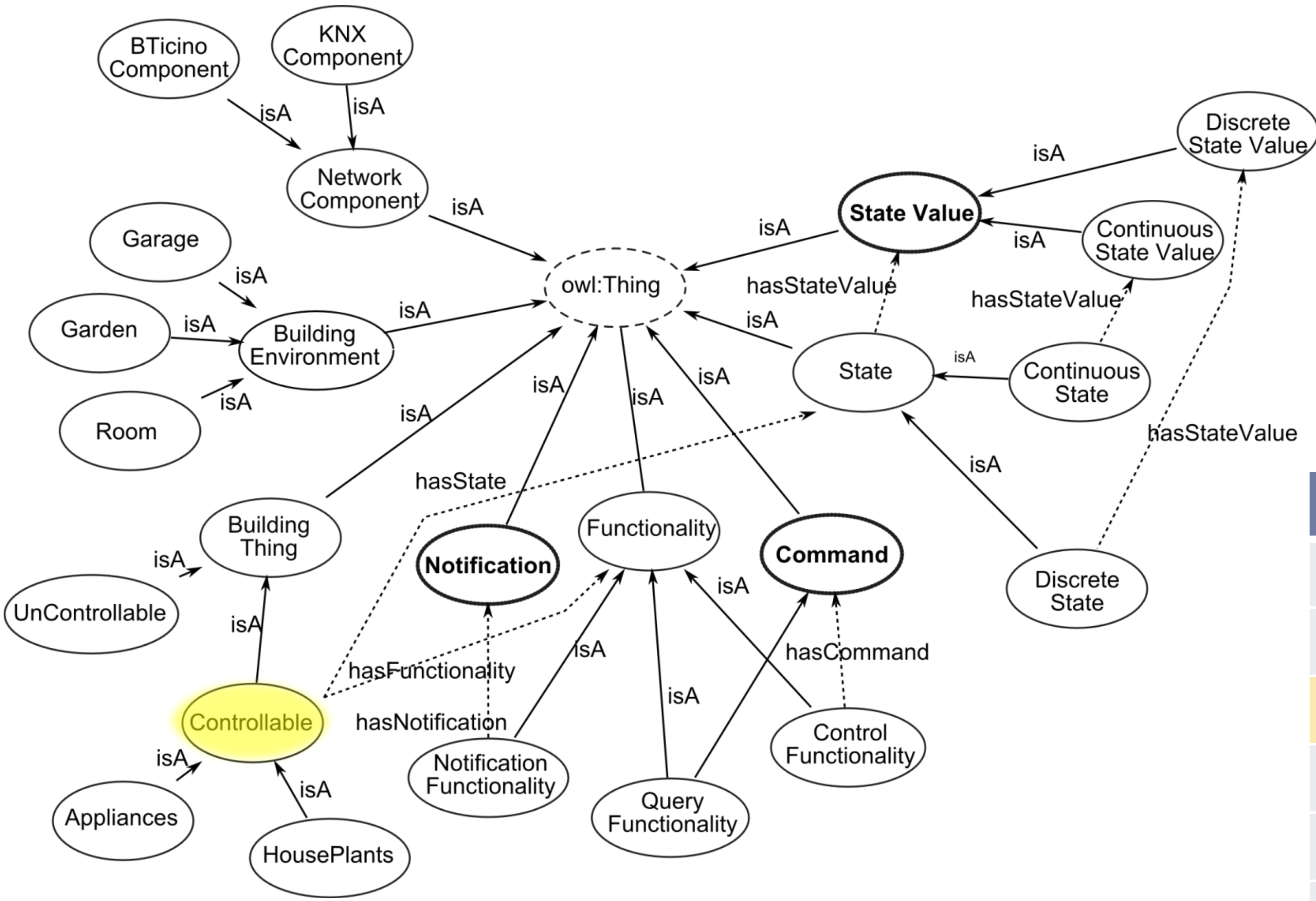
General Goals

- Adopt **formal representations** to allow a sound design process
- Enable validation and verification **throughout** the design process
- Integrate the solution in the Dog2.1gateway toolset

Adopted formalisms

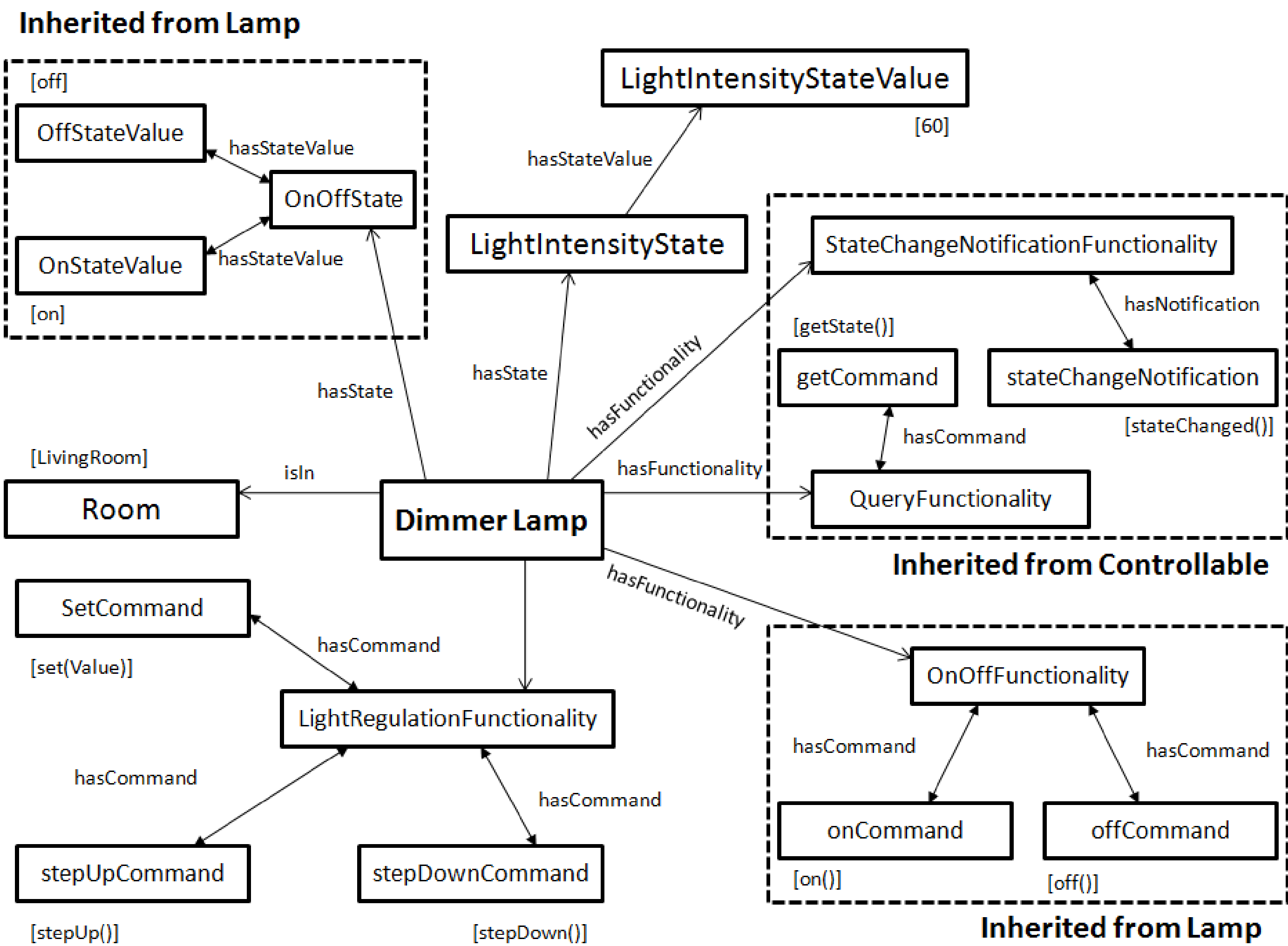
Level	Design artifact	Technique	Formalism
User Interface	System requirements	Temporal Logics	UCTL
Intelligence	Intelligent algorithms	State machines	UML Statecharts
Middleware	Device categories	Ontology	DogOnt classes
	System configuration	Ontology	DogOnt instances
Devices	Device models	State machines	UML Statecharts
	Whole system behavior	Parallel state machines	UML Statecharts

The DogOnt ontology



Formalism
UCTL
UML Statecharts
DogOnt classes
DogOnt instances
UML Statecharts
UML Statecharts

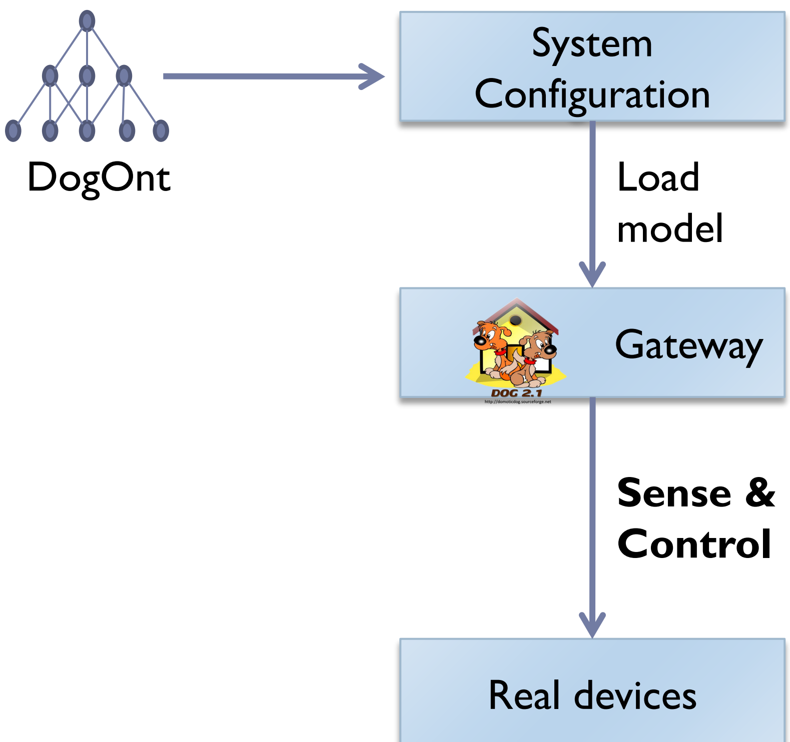
DogOnt instances: DimmerLamp



Formalism
UCTL
UML Statecharts
DogOnt classes
DogOnt instances
UML Statecharts
UML Statecharts

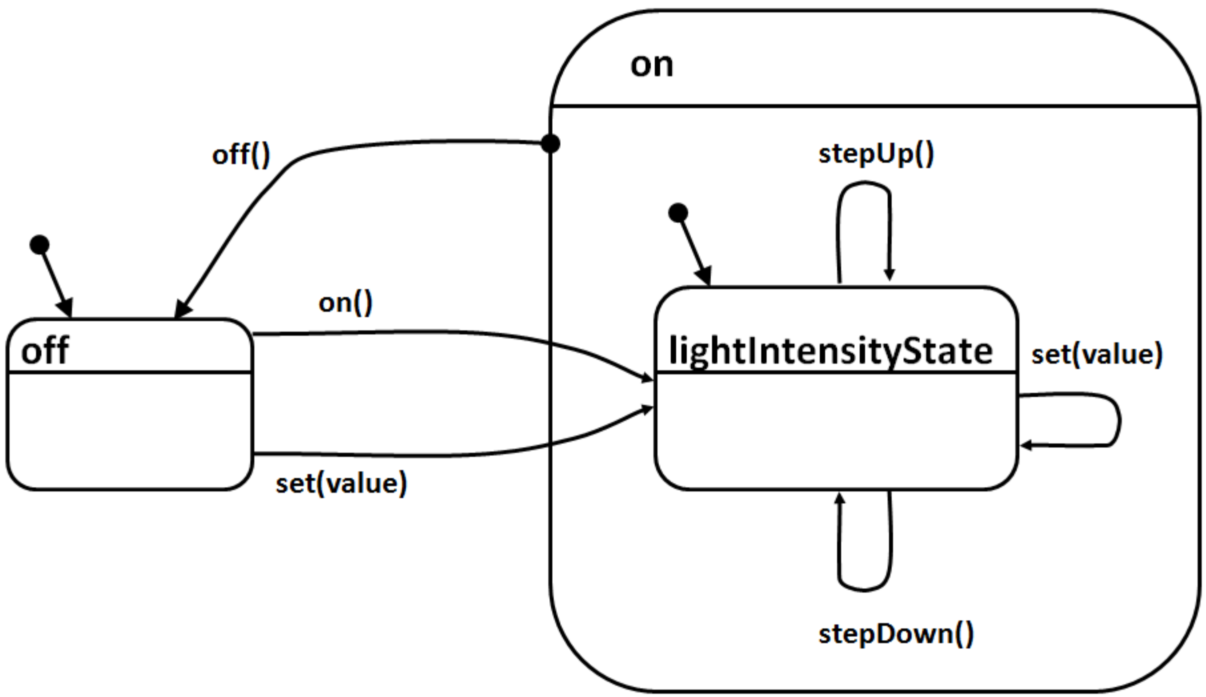
Overall system components

…to be continued…



Device modeling

- Ontologies are declarative formalisms: device properties
- For device behavior we need an operational formalism
  - Statecharts (Harel, 1987, now in UML 2.0)



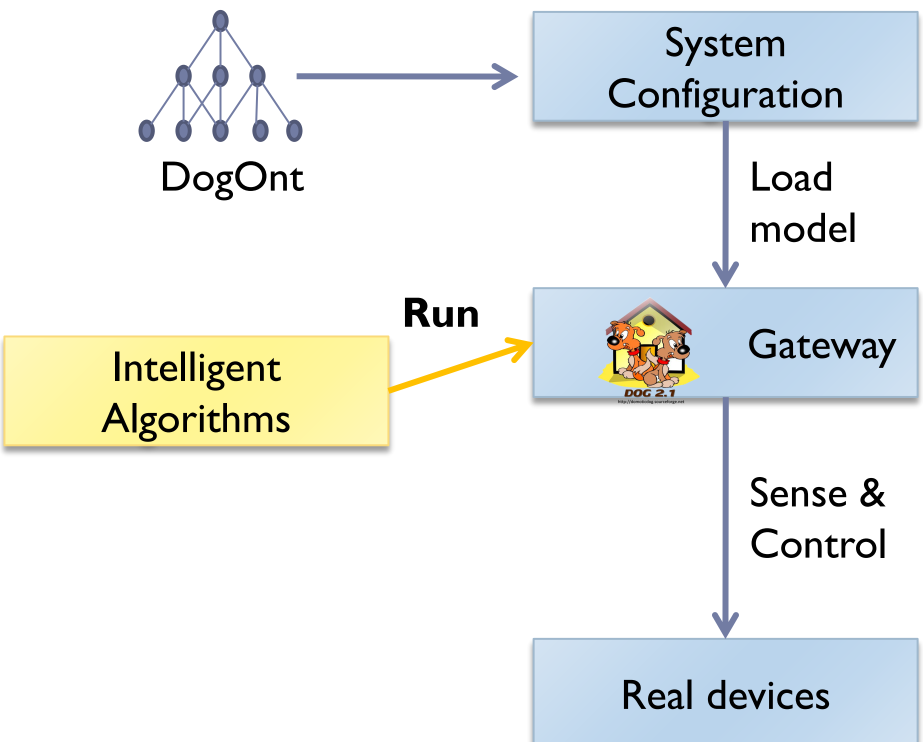
Formalism
UCTL
UML Statecharts
DogOnt classes
DogOnt instances
UML Statecharts
UML Statecharts

Use cases

- Ontologies are declarative formalisms: device properties
- For device behavior we need an operational formalism
  - Statecharts (Harel, 1987, now in UML 2.0)
- We use Statecharts for
  - Modeling the behavior of each **device** type
  - Implementing the **Intelligent Algorithms** within the gateway
  - Building a **whole-system model** allowing simulation and emulation
- Statecharts have a formal semantics: formal verification is possible

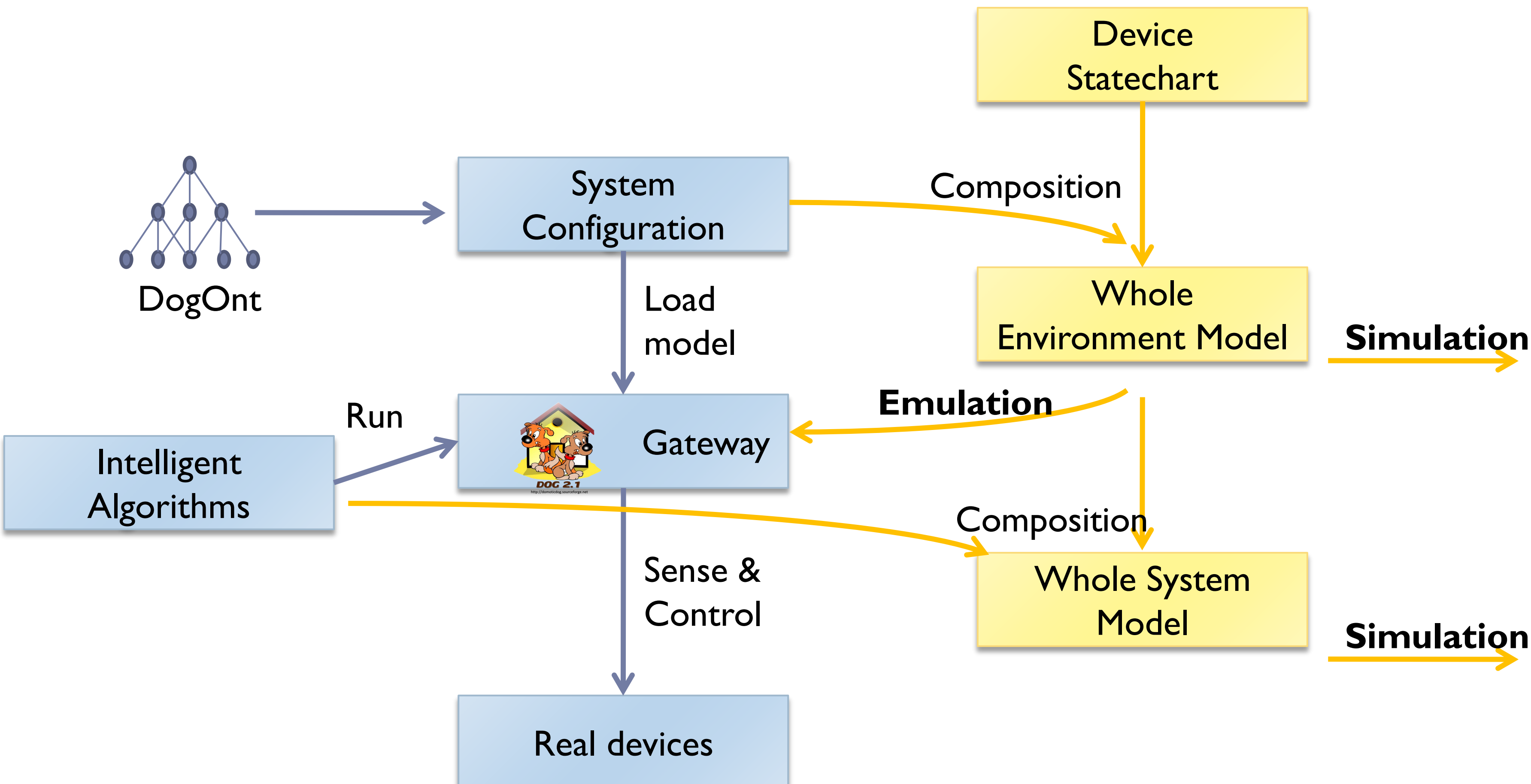
Overall system components

…to be continued…



Overall system components

…to be continued…



Temporal logic

- UCTL logic
  - Branching-time
  - State-based and action-based
  - Operators
    - Next (X,N)
    - Future (F)
    - Globally (G)
    - All (A)
    - Exists (E)
    - Until (U)
- UMC Model Checker
  - Supports Statecharts as a model

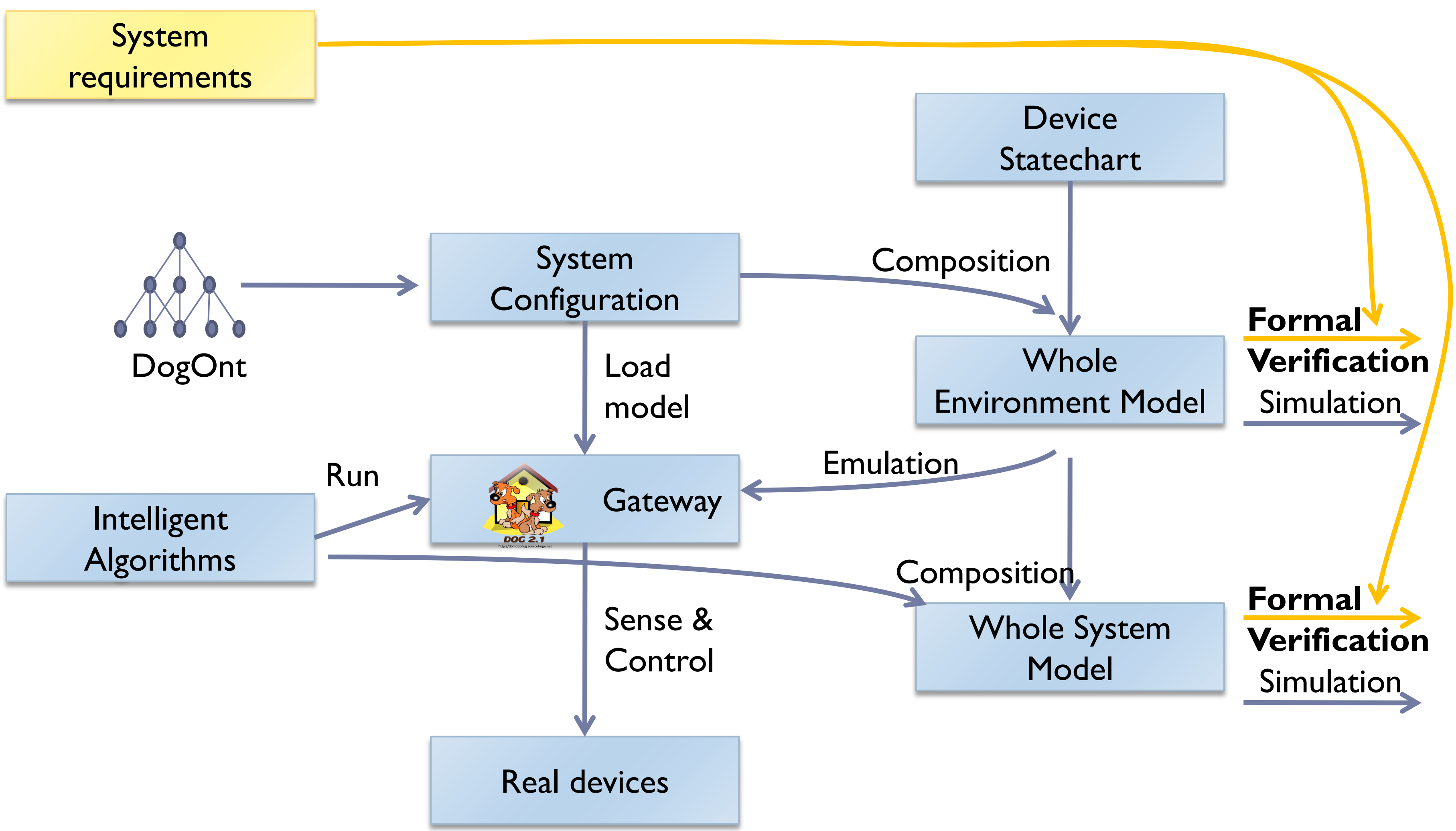
Examples

$$AG[\![openRequest(T1)]\!]$$
$$A\left[\top\ {}_{\{\neg openRequest(T1)\}}U_{\{tsDone(T1)\}}\ \top\right]$$

$$AG[\![daDoorOpen(DAExt)]\!]$$
$$A\left[\top\ {}_{\{\neg daDoorOpen(DAInner)\}}U_{\{extDoorClosed()\}}\ \top\right]$$

Formalism
UCTL
UML Statecharts
DogOnt classes
DogOnt instances
UML Statecharts
UML Statecharts

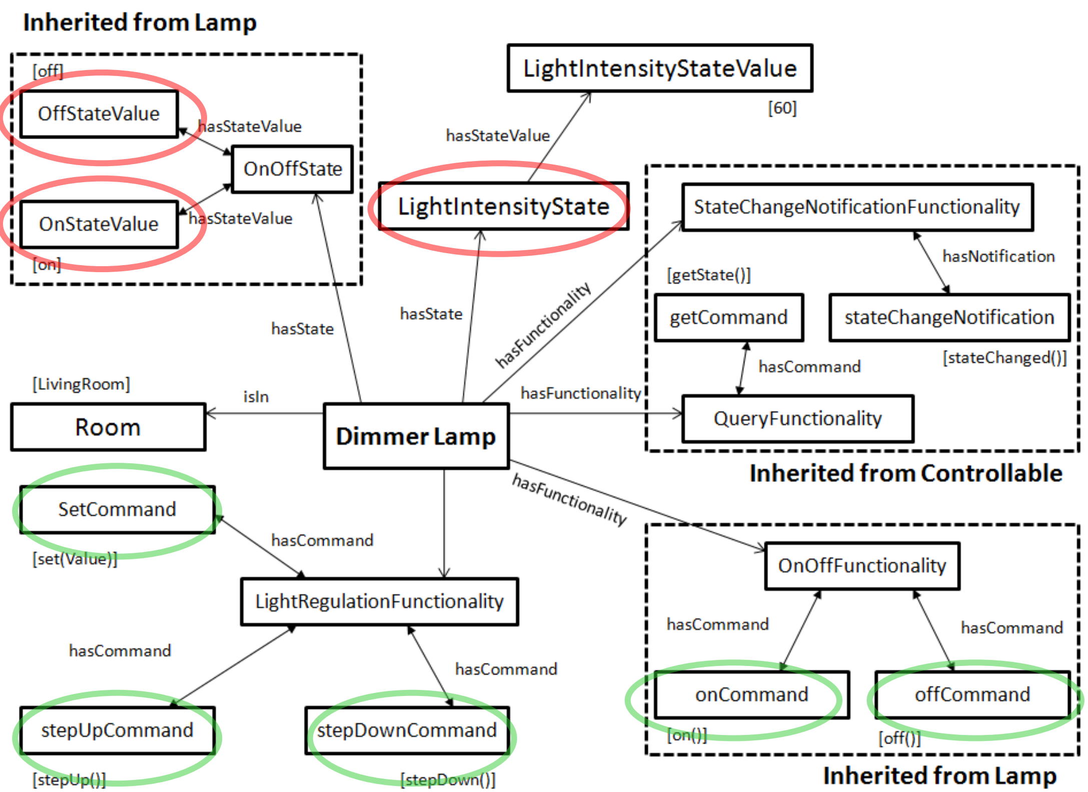
Overall system components



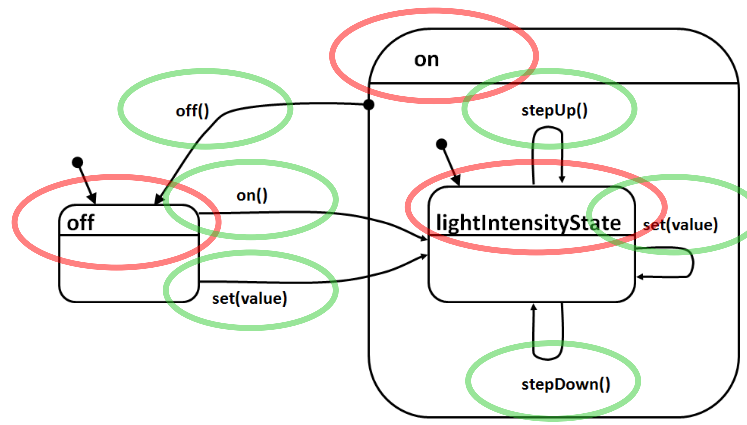
But… (goal of this paper)

- Formal verification relies on the composition of device state charts
- Environment control relies on information in DogOnt device properties
- How to ensure their consistency?
- Solution: use formal verification, too

The problem

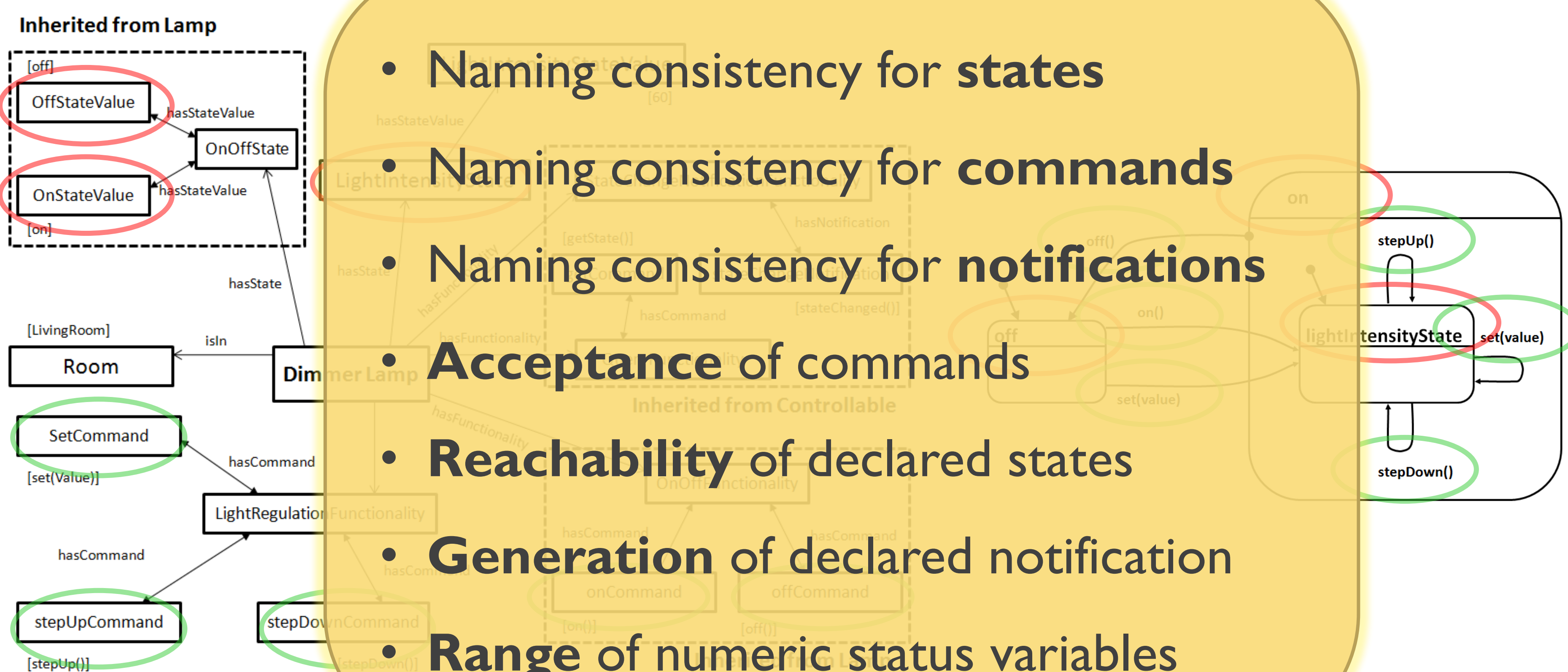


Inherited from Lamp
[off]
OffStateValue
hasStateValue
OnOffState
OnStateValue
hasStateValue
[on]
LightIntensityStateValue
[60]
hasStateValue
LightIntensityState
StateChangeNotificationFunctionality
hasNotification
[getState()]
getCommand
stateChangeNotification
[stateChanged()]
hasCommand
QueryFunctionality
hasFunctionality
hasFunctionality
hasState
hasState
[LivingRoom]
isIn
Room
Dimmer Lamp
Inherited from Controllable
hasFunctionality
SetCommand
[set(Value)]
hasCommand
LightRegulationFunctionality
OnOffFunctionality
hasCommand
hasCommand
hasCommand
hasCommand
stepUpCommand
[stepUp()]
stepDownCommand
[stepDown()]
onCommand
[on()]
offCommand
[off()]
Inherited from Lamp



on
off()
stepUp()
on()
off
lightIntensityState
set(value)
set(value)
stepDown()

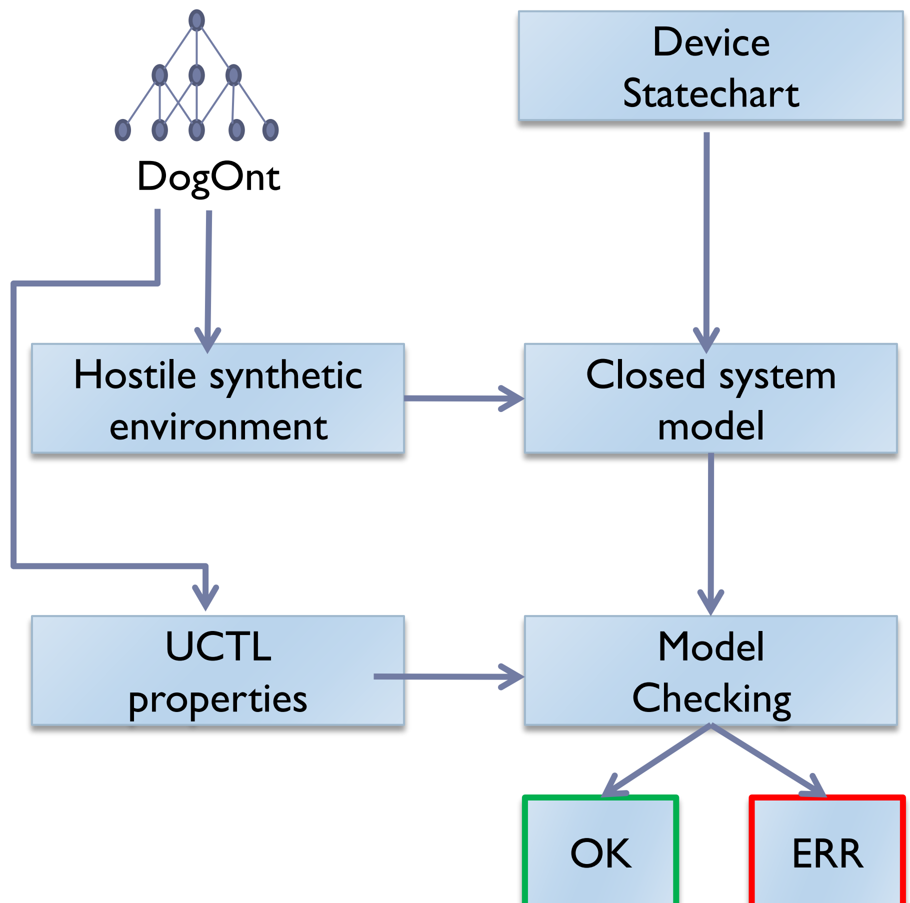
The problem



- Naming consistency for **states**
- Naming consistency for **commands**
- Naming consistency for **notifications**
- **Acceptance** of commands
- **Reachability** of declared states
- **Generation** of declared notification
- **Range** of numeric status variables

Approach

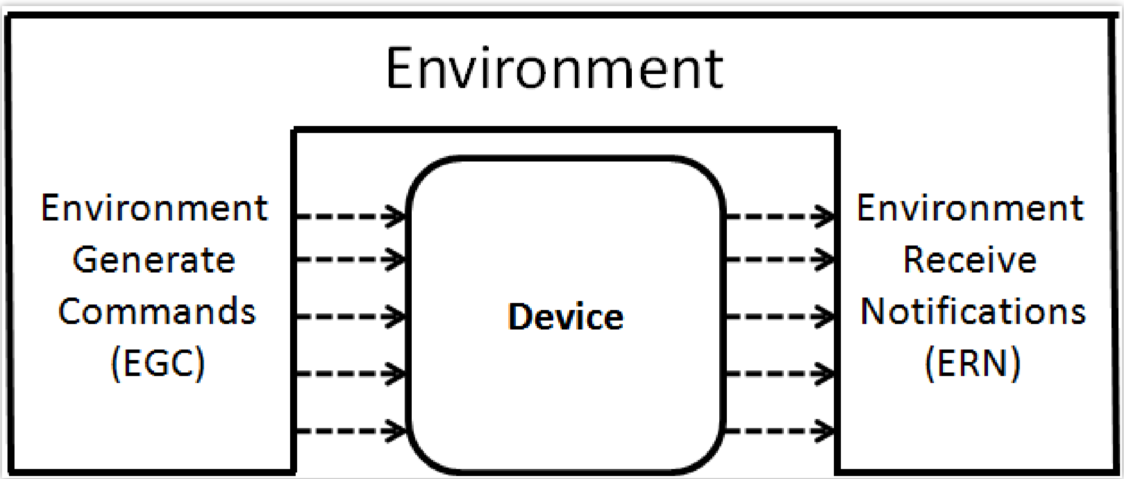
- From DogOnt, extract UCTL properties
- From DogOnt, build a synthetic environment for the device
- Integrate Device State Chart in the synthetic environment
- For every property
  - Run Model checher



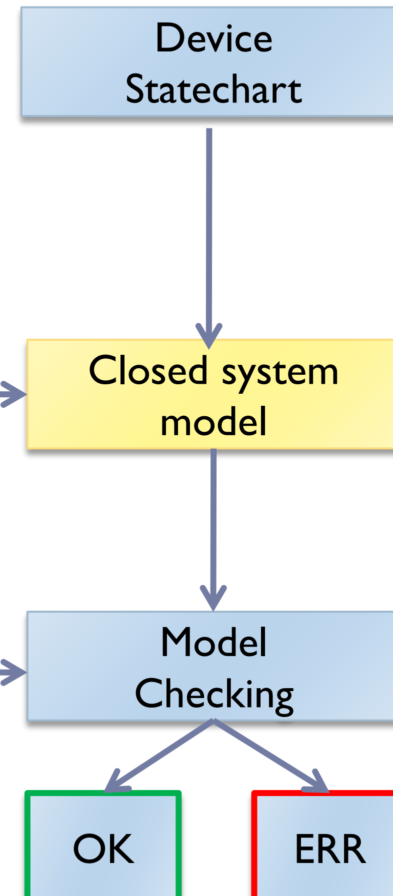
Approach

- From DogOnt, extract

Building a closed system model, ready for verification



- Run Model checher



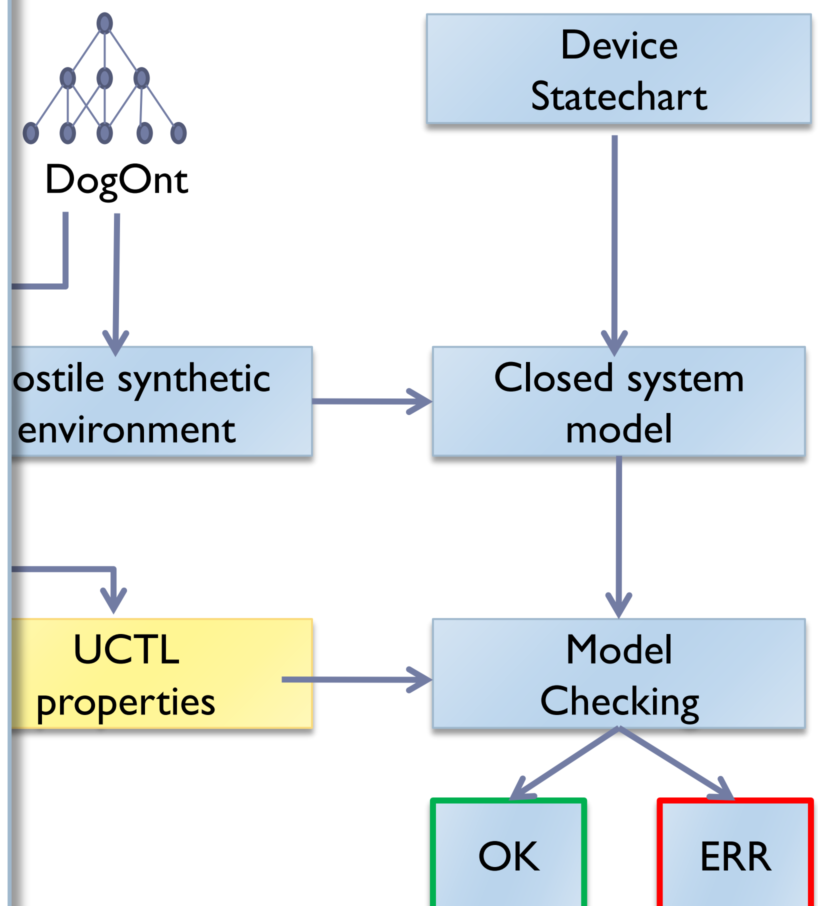
Approach

Example: DimmerLamp generated & verified properties

```
--Action Properties
  --the acceptance of all the commands in DSC
    EF {sending(stepDown)} true
    EF {sending(stepUp)} true
    EF {sending(set)} true
    EF {sending(off)} true
    EF {sending(on)} true
  --
    EF {accepting (stepDown)} true
    EF {accepting (stepUp)} true
    EF {accepting (set)} true
    EF {accepting (off)} true
    EF {accepting (on)} true

  --the generation of all the notifications in DSC
    EF {sending(stateChanged)} true
    EF {accepting(stateChanged)} true

--State Properties
  --the reachability of all the states in DSC
    EF (offState)
    EF (onState)
    EF (LightIntensityState)
```



Experimental Results

- UCTL Model Checker
- Dog2.1 standard device classes
- Device classes verified: 11
- Number of verifies properties: 114
  - Some design errors found and corrected
- CPU time: < 1 sec / property

- **Formally validated device statechart library in Dog2.1**

Conclusions

- Engineering the Design Process for Intelligent Environments
- Formalisms and tools are needed
- Ontologies, Statecharts, Temporal Logics

Thank you!

Questions?

http://elite.polito.it
http://domoticdog.sourceforge.net
fulvio.corno@polito.it