Modelling a Smart Grid System-of-Systems using VDM

Stefan Hallerstede and Peter Gorm Larsen

Aarhus University

Aarhus University
Aarhus
28 August 2013

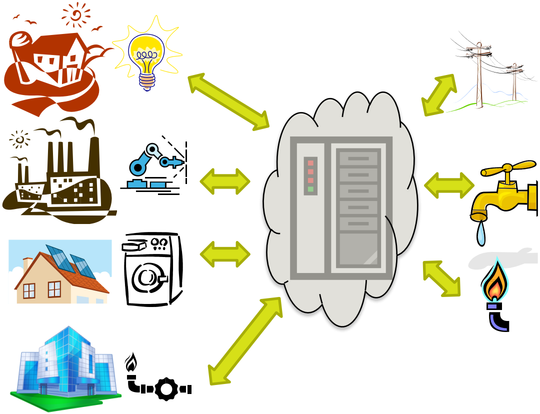
AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

Context

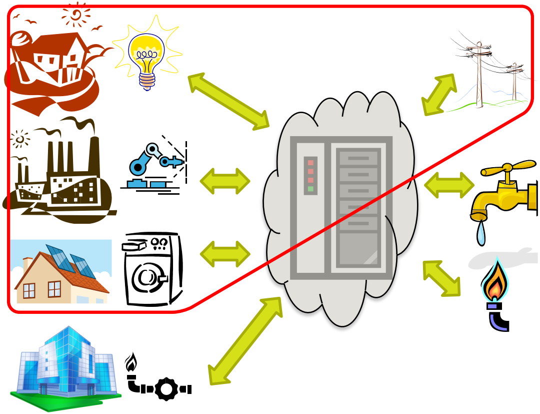
- Modelling of a System of Systems (SoS)
- Carried out within the COMPASS project (`http://www.compass-research.eu`)
- Purpose: evaluation of baseline technolgy (here: VDM++)
- Question: how well is VDM++ suited for this kind of problem?
- Question in the next phase:
  - how does the COMPASS modelling language (CML) improve the situation?
  - CML combines VDM++ and CSP (similar to Circus)
- Simple SoS case study from COMPASS Interest Group: Smart Grid

AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

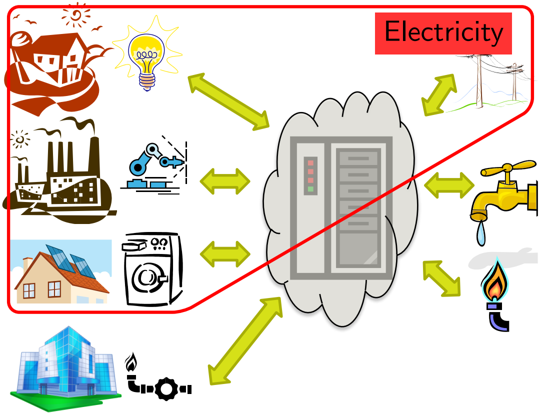
Problem Statement



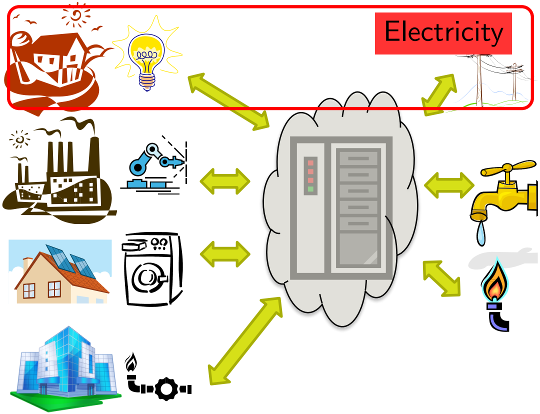
Problem Statement



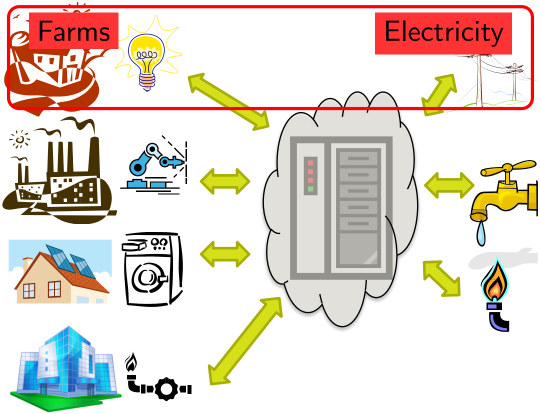
Problem Statement



Problem Statement

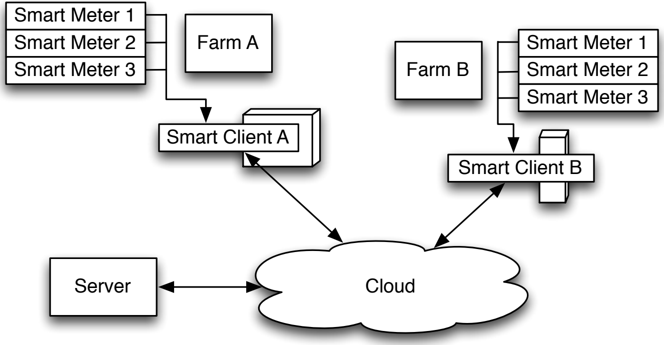


Problem Statement



AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

The Smart Grid Architecture (Sketch)

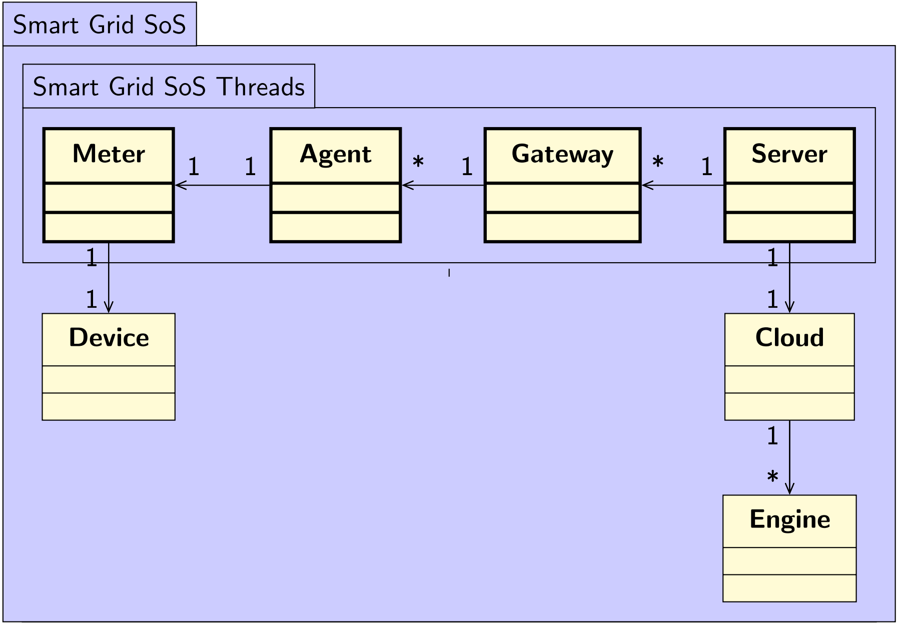


Challenges:

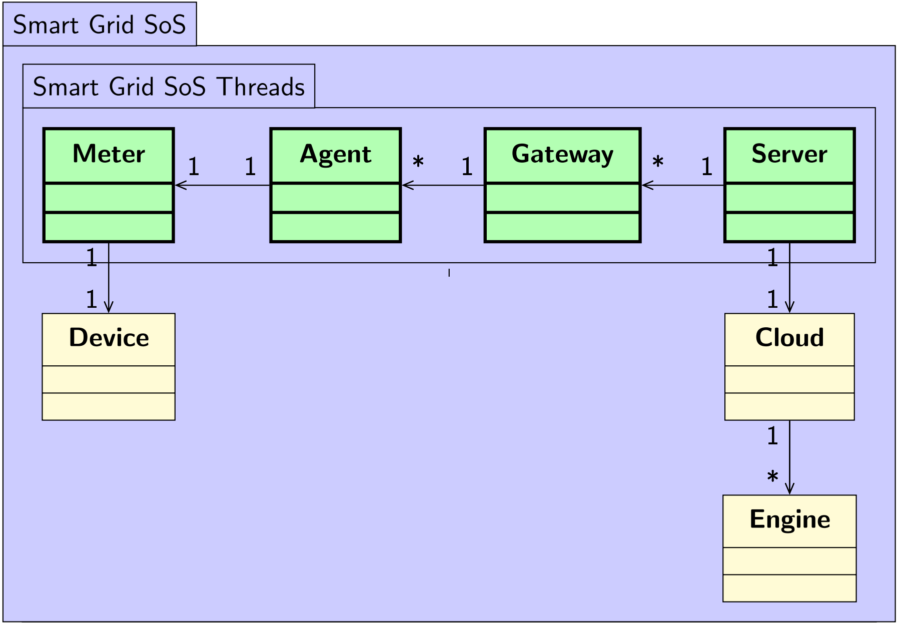
- Heterogeneity of the farms
- Frequent changes of constituent systems and control rules
- Many changing stakeholders
- Evolution and scale

AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

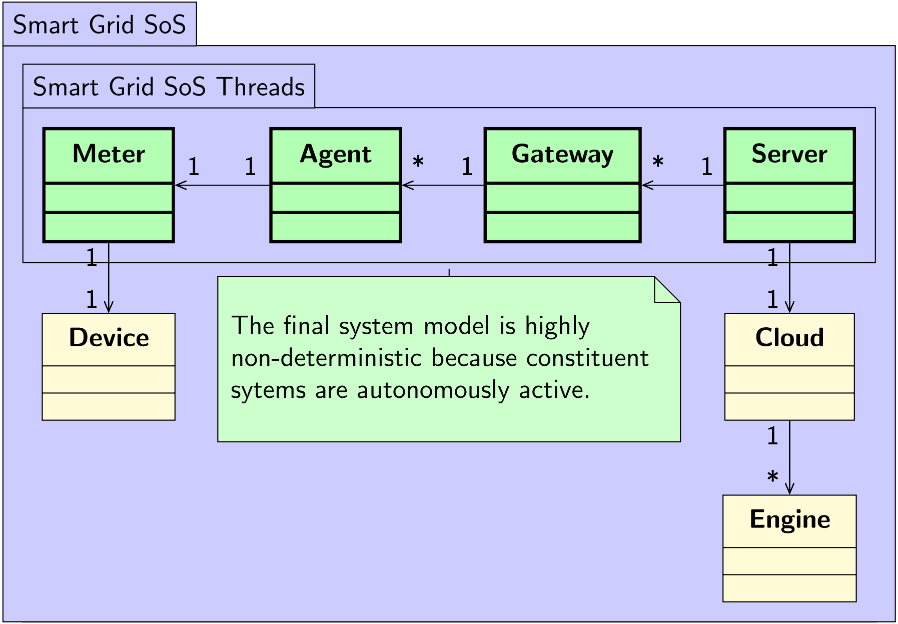
Generic Architecture of the SoS in VDM++



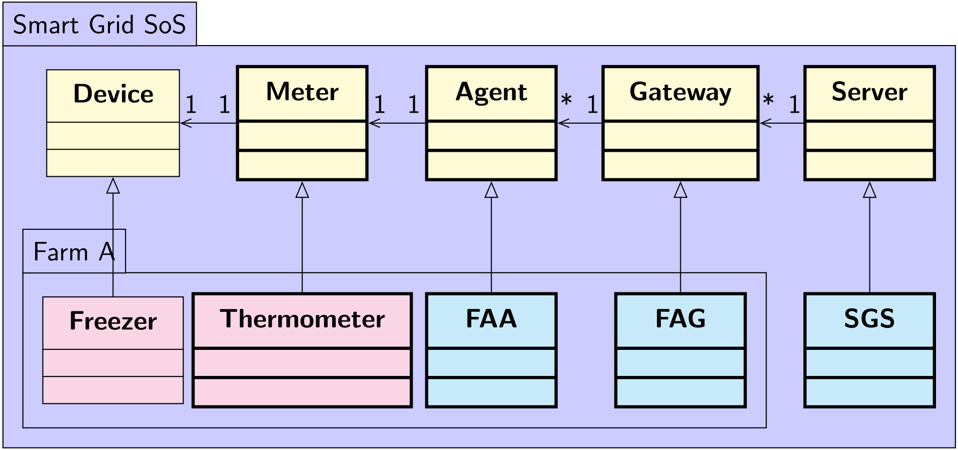
Generic Architecture of the SoS in VDM++



Generic Architecture of the SoS in VDM++



Concrete Architecture by Inheritance



VDM Example: Generic Device

```
class Device is subclass of Types

instance variables

public relay : map RelayID to Relay := {|->};

operations

public switch_relay : RelayID * bool ==> ()
switch_relay(id,s) ==
  relay(id).switch(s)
pre id in set dom(relay);

end Device
```

VDM Example: Farm A Freezer Device

```
class Farm_A_Freezer_Device is subclass of Device

instance variables

  public Relay_Cool : RelayID;
  public Relay_Hold : RelayID;

  inv Relay_Hold in set dom relay and Relay_Cool in set dom relay
        => (not relay(Relay_Cool).state) or (not relay(Relay_Hold).state);

  public static device : Farm_A_Freezer_Device
        := new Farm_A_Freezer_Device();

operations
  private Farm_A_Freezer_Device : () ==> Farm_A_Freezer_Device
  Farm_A_Freezer_Device() == (
   Relay_Cool := mk_RelayID(0);
   relay := {Relay_Cool |-> new Relay()};
   Relay_Hold := mk_RelayID(1);
   relay := relay munion {Relay_Hold |-> new Relay()};
  )
end Farm_A_Freezer_Device
```

VDM Example: Farm A Freezer Meter (aka Thermometer)

```
class Farm_A_Freezer_Meter is subclass of Meter

values

  private min_temp : real = -25.0;
  private max_temp : real = 37.0;

instance variables
  private initial_temp : real := -20;
  private hold_curve : seq of real := [0,-0.5,-1,-0.5,0,0.5,1.0,0.5];
  private temp : real := initial_temp;
  public static meter : Meter := new Farm_A_Freezer_Meter();
  inv min_temp < temp and temp < max_temp;

operations
...

protected imp : nat ==> ()
imp (now) == (
  if (device.relay(Farm_A_Freezer_Device`device.Relay_Cool).state) then (
     temp := temp - ((now-last_time)*rate);
  ) elseif (device.relay(Farm_A_Freezer_Device`device.Relay_Hold).state) then (
    temp := temp + (hd hold_curve);
    hold_curve := (tl hold_curve) ^ [hd hold_curve];
  ) else (
    temp := temp + ((now-last_time)*rate);
  );
);

...
end Farm_A_Freezer_Meter
```

VDM Example: Smart Grid Configuration

```
{
  -- switch freezer to hold
  "Farm_A_Gateway" |->
  {
    mk_Types`TriggerRule(
      {},
      mk_Types`Interval(4,4),
      mk_Types`Activity(
        [mk_Types`RelayAction(mk_Types`RelayID(0),<OFF>),mk_Types`RelayAct
        1,
        []),
      mk_Types`Dates({},{})
    )
  },

  "Farm_B_Gateway" |->
  {
    mk_Types`TriggerRule(
      {},
      mk_Types`Interval(20,20),
      mk_Types`Activity(
        [mk_Types`RelayAction(mk_Types`RelayID(2),<ON>)],
        40,
        [mk_Types`RelayAction(mk_Types`RelayID(2),<OFF>)]),
      mk_Types`Dates({},{})
    )
  }
 }
```

Control rules

```
{
  -- enable the safety switch
  "Farm_A_Freezer_Meter" |->
  [
    mk_ (0.0, 1.0, 0.0),
    mk_ (3.0, 1.0, 2.0)
  ],

  -- keep the direction switch neutral
  "Farm_B_Battery_Meter" |->
  [
    mk_ (0.0, 0.0, 0.0),
    mk_ (4.0, 0.0, 1.0),
    mk_ (7.0, 3.0, -2.0)
  ]
}
```

Scenario

Concluding Remarks

- Property Specification:
  - **SoS-wide properties** can be conveniently expressed by referring to **instance variables**
  - They need to be “property public”: *visible by property specifications but not by operations*
- Communication:
  - Communication is modelled by operation calls:
    `Send...()`
    `Receive...()`
  - This means data is “pushed” or “pulled”
  - CSP-like channel communication would avoid this

Concluding Remarks

- Stakeholder Modelling:
  - We have tried to take care of stakeholders by arranging the model in files and folders
  - This could be used with a usual **configuration control system**
  - This does not however provide **confidentiality**
  - It does also not allow to restrict shown models to those parts **relevant** to specific stakeholders

Concluding Remarks

- Correctness:
  - In the current model the SoS fails if any CS fails
  - This is unrealistic for an SoS model
  - However, it permits observing **failures** more easily as if the model was **fault-tolerant** but more realistic
  - More than one model may be needed! How could this be done?
- More Correctness:
  - Correctness of the Smart Grid SoS is (partly) based on its current configuration
  - Behaviour can be changed if it is unused in the current configuration