Protecting Critical Infrastructures While Preserving Each Organization's Autonomy

Yves Deswarte
deswarte@laas.fr

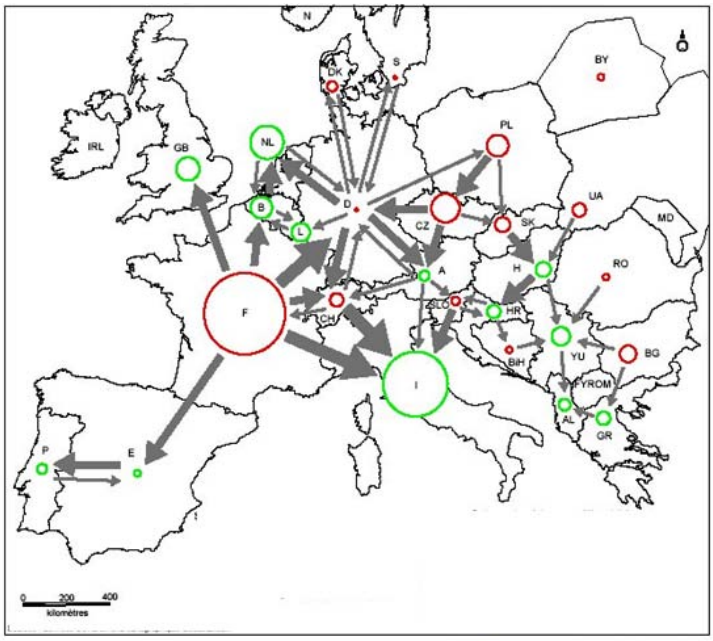
LAAS-CNRS

Toulouse, France

Critical Infrastructures (CI)

- Provide essential supplies
  - Electricity, water, transport, telecom, finance, …
- Large extension, multiple organizations
- These infrastructures are interconnected
  - Cascading failures
  - Escalading failures
  - Ex. North America black-out (Aug. 2003)
    - Bug in monitoring S/W + minor H/W failure -> loss: 7 to 14 US$ billions

Example: European Electric Grid

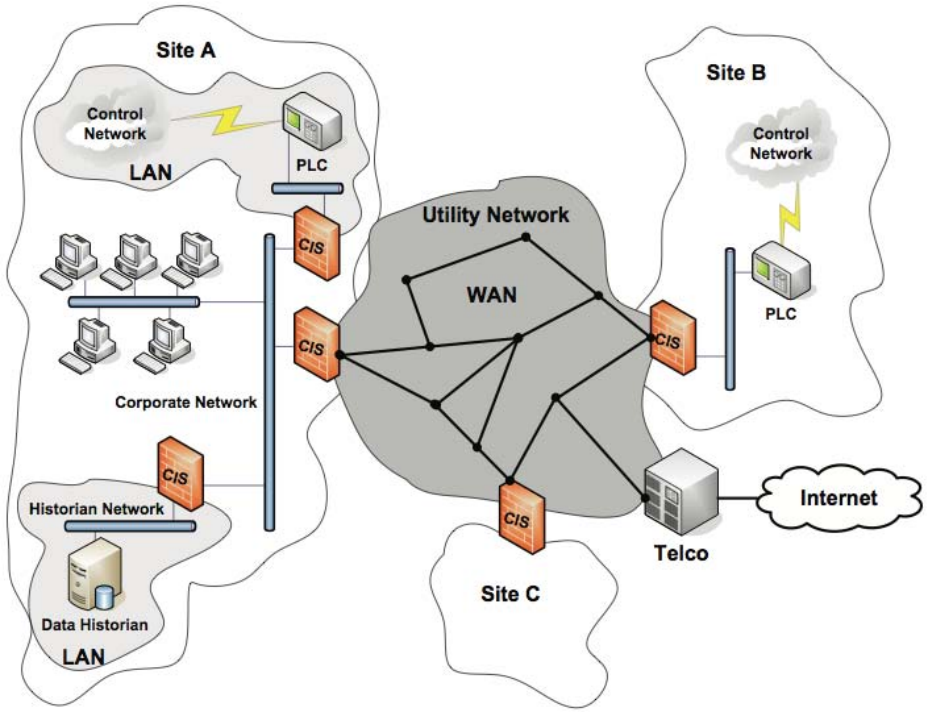


- Deregulated market
- Heterogeneous organizations:
  - Generation
  - Transmission
  - Distribution
- No global authority
- Cooperation & competition

Critical Information Infrastructure

- Each Critical Infrastructure (CI) is controlled by an underlying *Critical Information Infrastructure* (CII)
  - Computing Systems (e.g., SCADA, management information systems, …) interconnected through Networks
  - As critical as the CI itself
    - Plausible target for cyber-attacks
      - Ex. Stuxnet

Typical CII



CII Security

- Combination of good security practices
  - Security policies: properties + rules
  - Enforcement: authentication + AC mechanisms
  - Monitoring and Audit
- CII security requirements
  - Secure cooperation without a global authority
  - Autonomy, confidentiality, responsibility
  - Monitoring --> collection of evidence
  - Flexibility and scalability

How to manage security in multiple organizations?

Centralized vs. peer-to-peer

- Centralized:
  - Global security policy
  - Enforced by each cooperating organization
    - Requires adaptation of local policies
  - Global monitoring
  - Incompatible with autonomy and flexibility requirements

Centralized vs. peer-to-peer

- Peer-to-peer
  - Each organization defines and enforces its own security policy
  - For cooperation, each organization incorporates entities from other organizations (users or roles, objects, …)
    - Security policy consistency ?
    - Autonomy and sovereignty ?
    - Confidentiality ?
    - Flexibility and scalability ?

PolyOrBAC proposal

- Each organization defines and enforces its own security policy
  - Protects its assets by its own means
  - Is responsible for its users with respect to the other organizations
- Interactions between organizations through Web Services
  - Point-to-point agreement between the service provider and the service client
  - Service interface is incorporated within the provider and the client policies

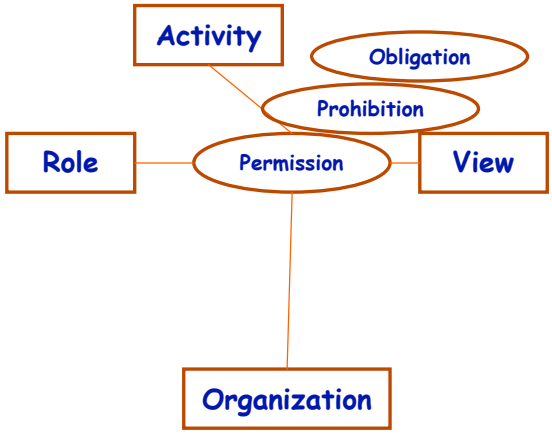
Local security policy

- OrBAC model
  - Abstractions
    - Users --> Roles
    - Objects --> Views
    - Actions --> Activities
  - Policy definition
    - Permission, prohibition, obligation rules expressed on abstract entities
  - Enforcement
    - At the interaction between concrete entities (users, actions, objects)

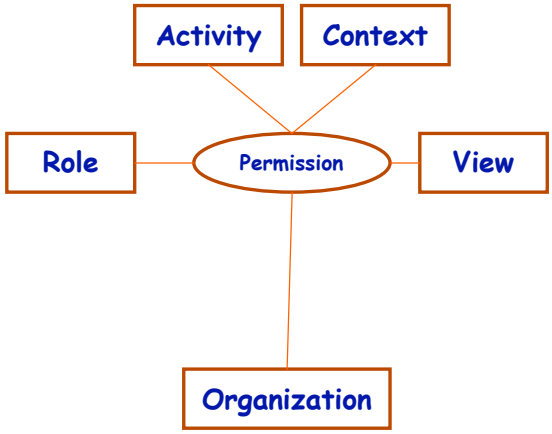
OrBAC

Organization

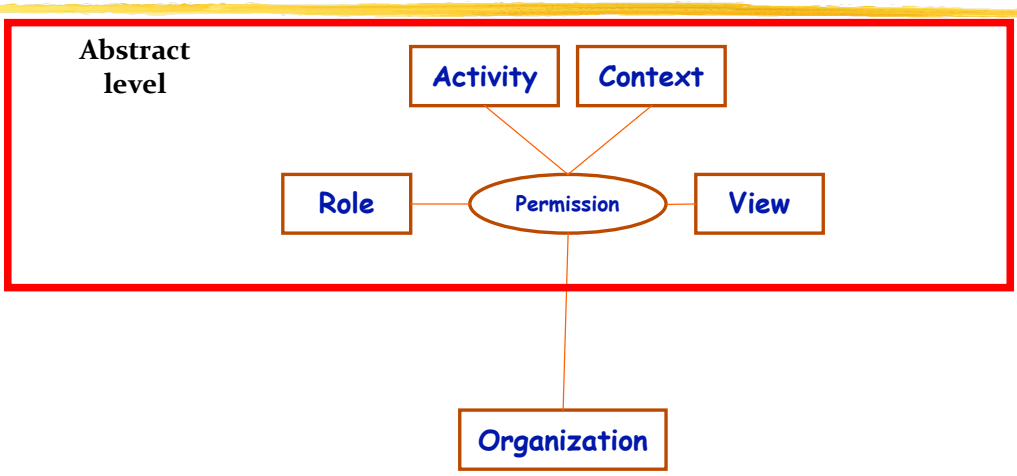
OrBAC



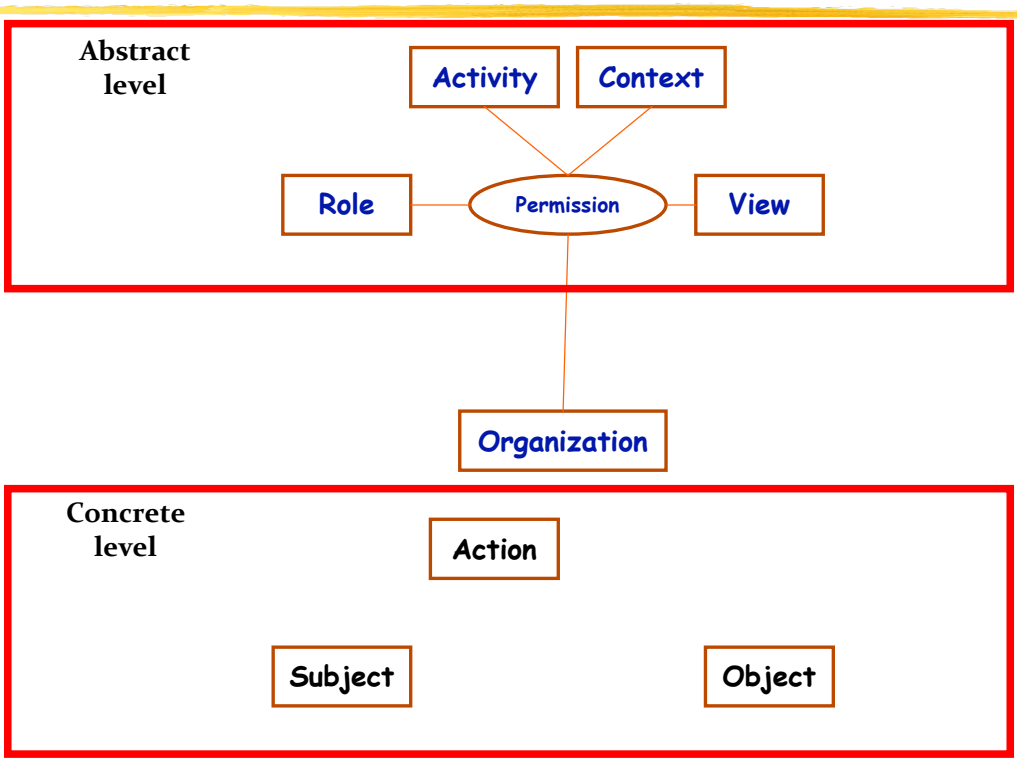
OrBAC


Activity
Context
Role
Permission
View
Organization

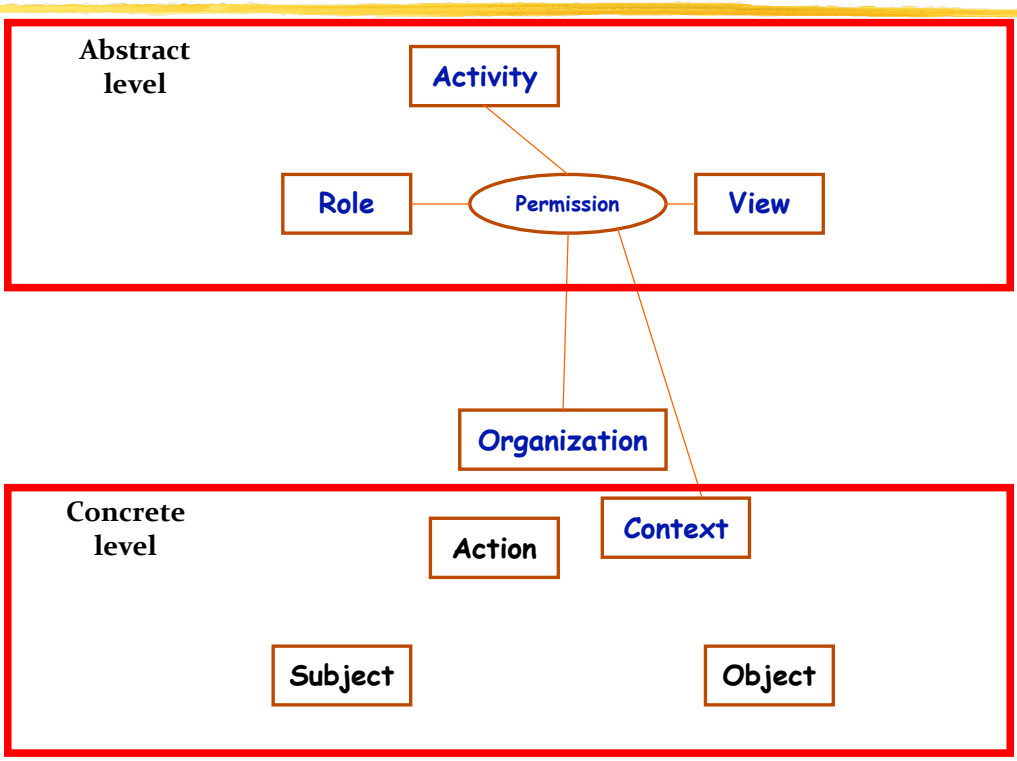
OrBAC


Abstract level
Activity
Context
Role
Permission
View
Organization

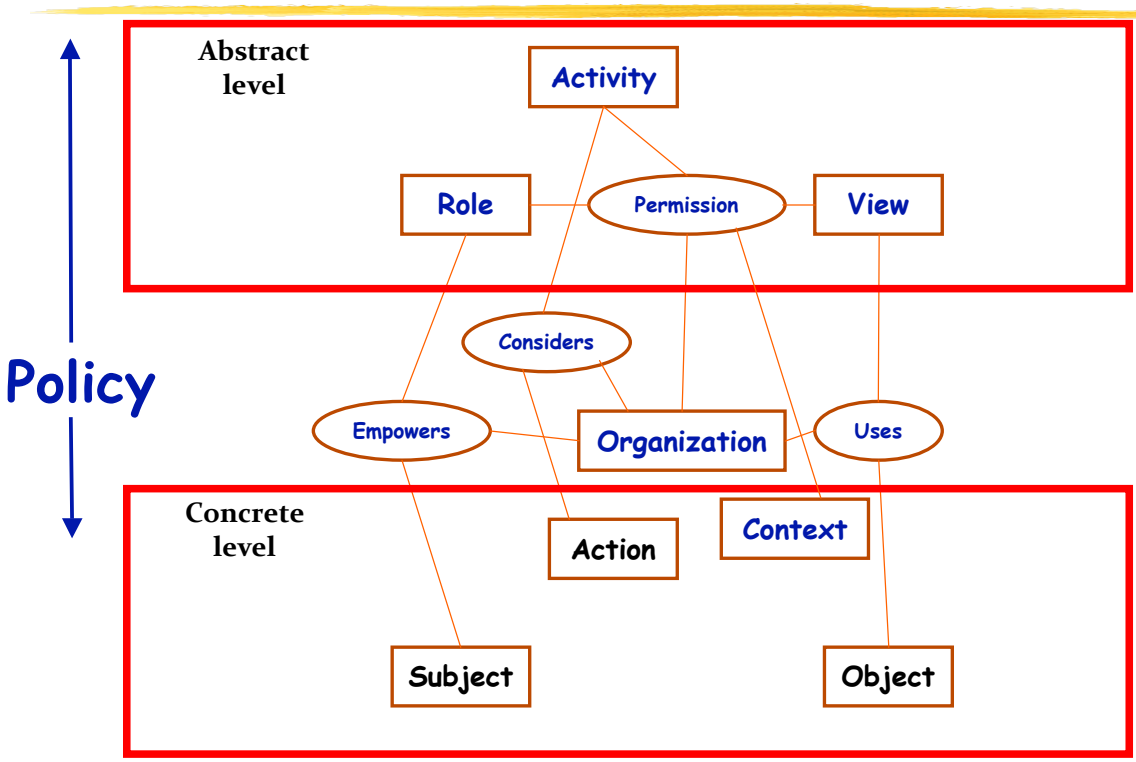
OrBAC


Abstract level
Activity
Context
Role
Permission
View
Organization
Concrete level
Action
Subject
Object

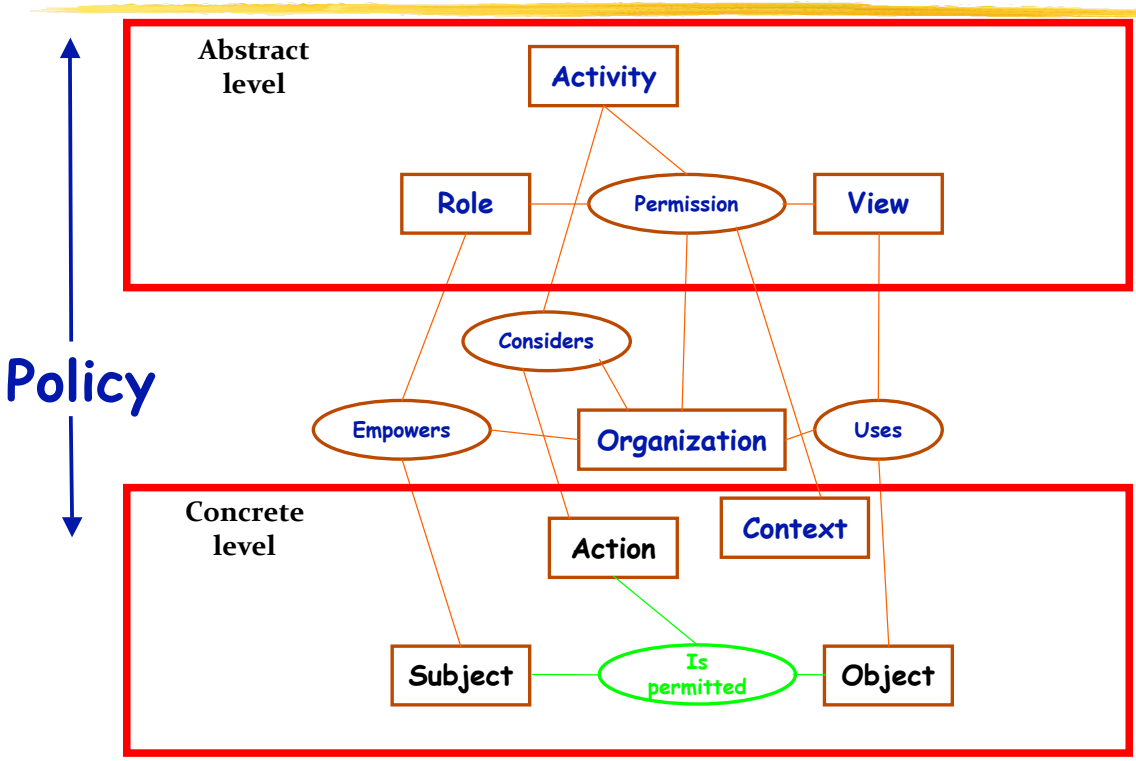
OrBAC


Abstract level
Activity
Role
Permission
View
Organization
Concrete level
Context
Action
Subject
Object

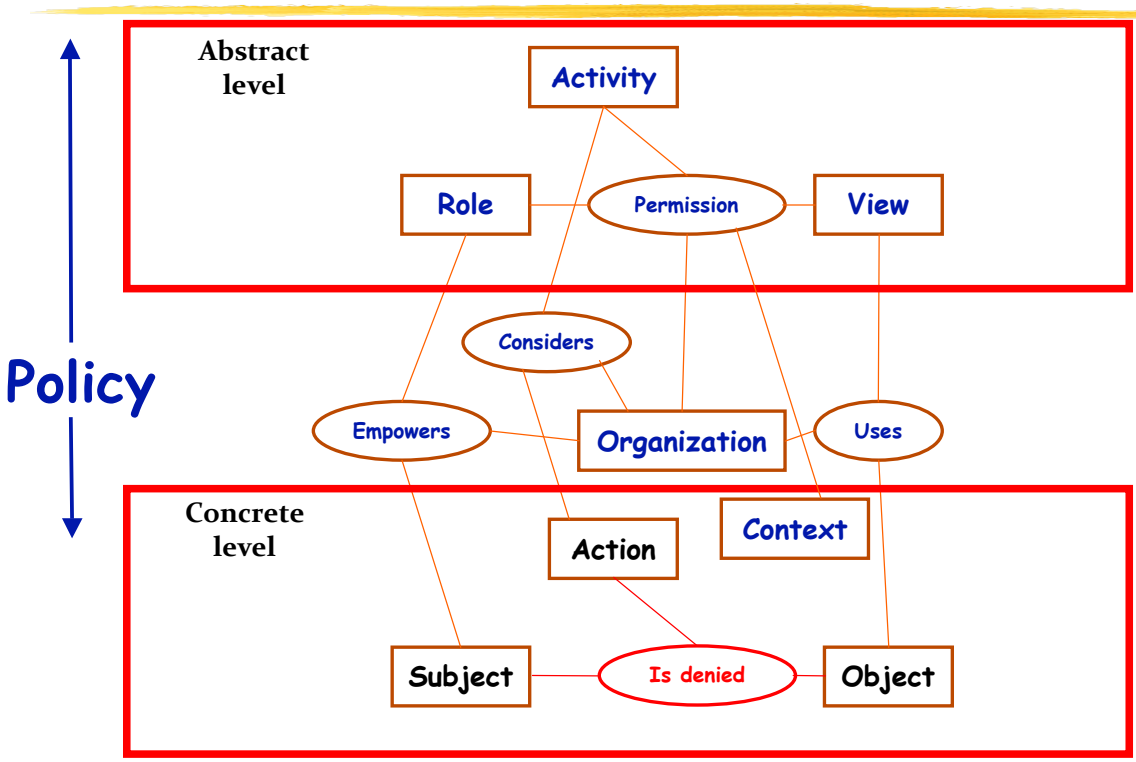
OrBAC


Abstract level
Activity
Role
Permission
View
Considers
Policy
Empowers
Organization
Uses
Concrete level
Action
Context
Subject
Object

OrBAC


Abstract level
Activity
Role
Permission
View
Considers
Policy
Empowers
Organization
Uses
Concrete level
Action
Context
Subject
Is permitted
Object

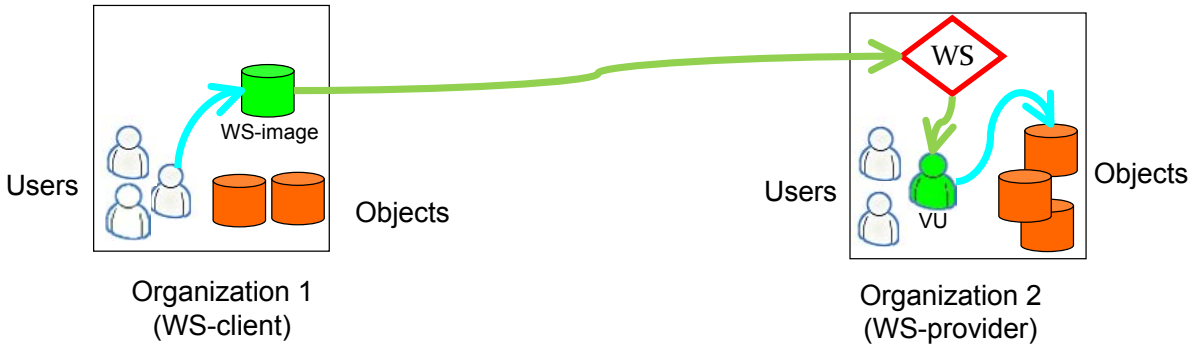
OrBAC



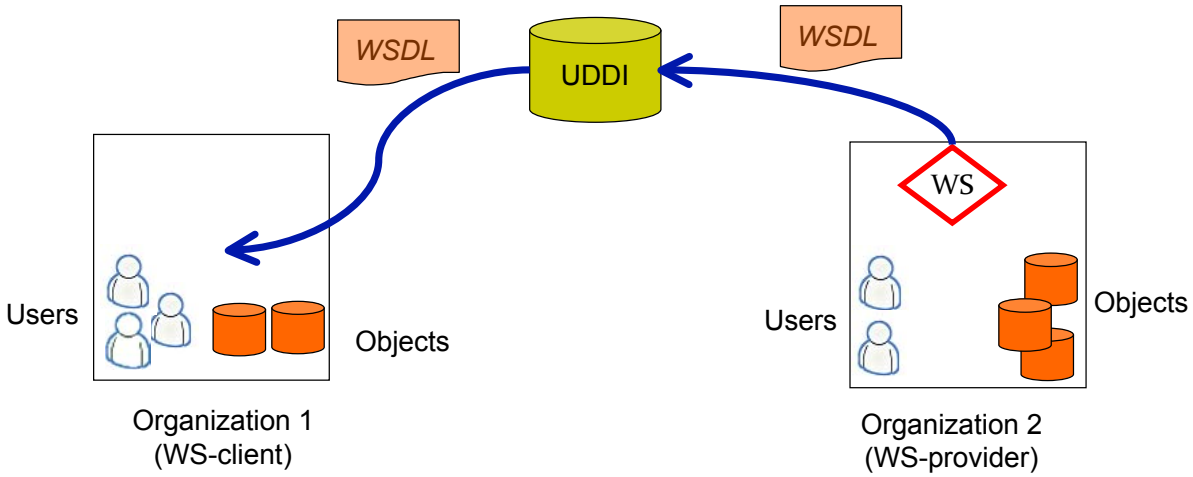
Interactions between organizations

- Web Services
  - Platform independent protocols & standards: XML, SOAP, WSDL, UDDI
  - Integration within local OrBAC policies
    - For the service provider: the client organization is viewed as a "*virtual user*"
    - For the service client: the service invocation is an action on a local "*service-image object*"

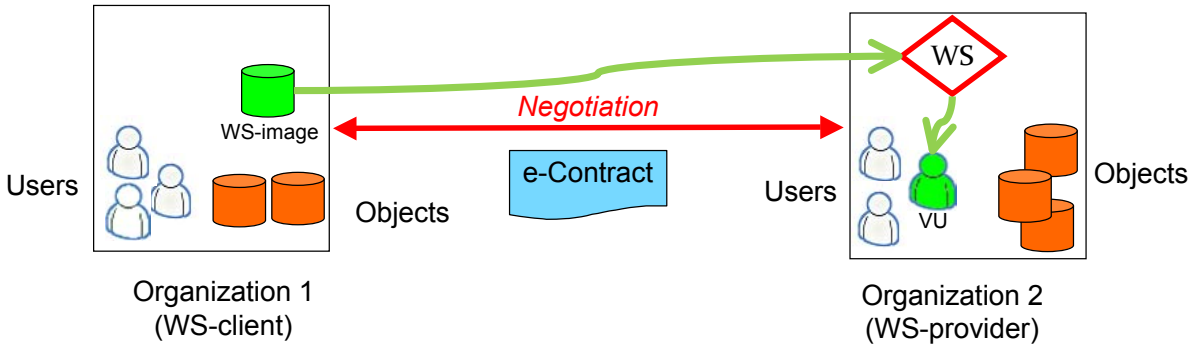
Virtual users and WS-images



Publication of a Web Service



Publication of a Web Service



e-Contract

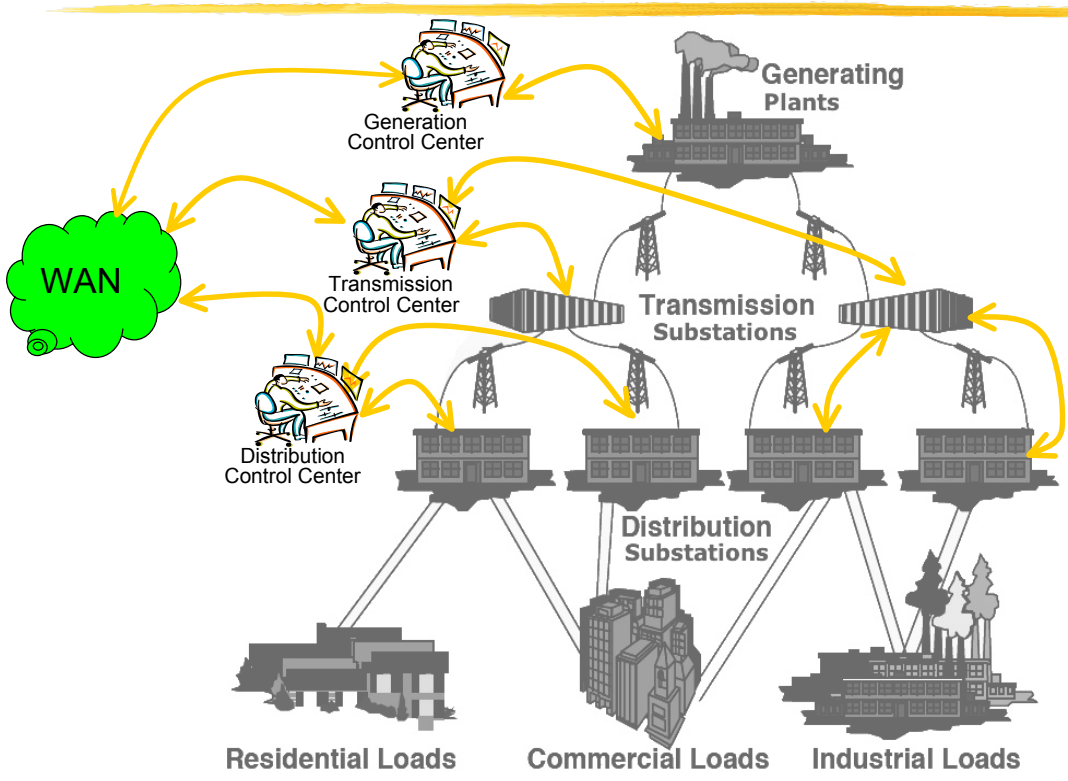
- Describes the agreed service
  - Functionality and time constraints (QoS), payment, penalty if service failure or abuse
    - The client organization is liable for its users (authentication, access control)
    - The provider is liable for the service (compliance with the contract)
  - Agreed security rules
    - Permissions, Prohibitions, Obligations
    - Consistency checked when signing the contract (static)
    - Rules enforced at run-time, with collection of evidence if a rule is violated

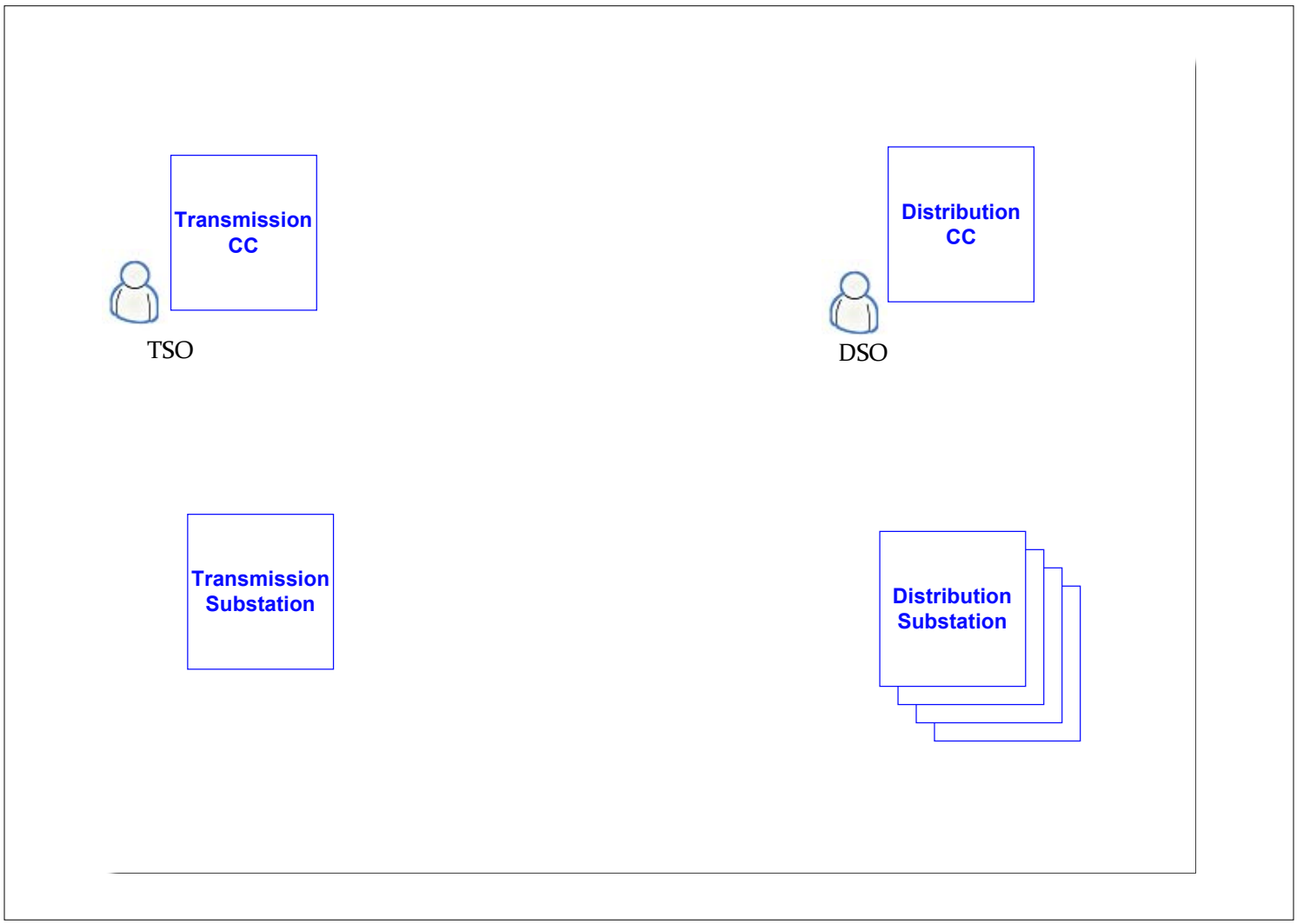
    → formal representation:
    1 Timed-automaton on each side of the WS

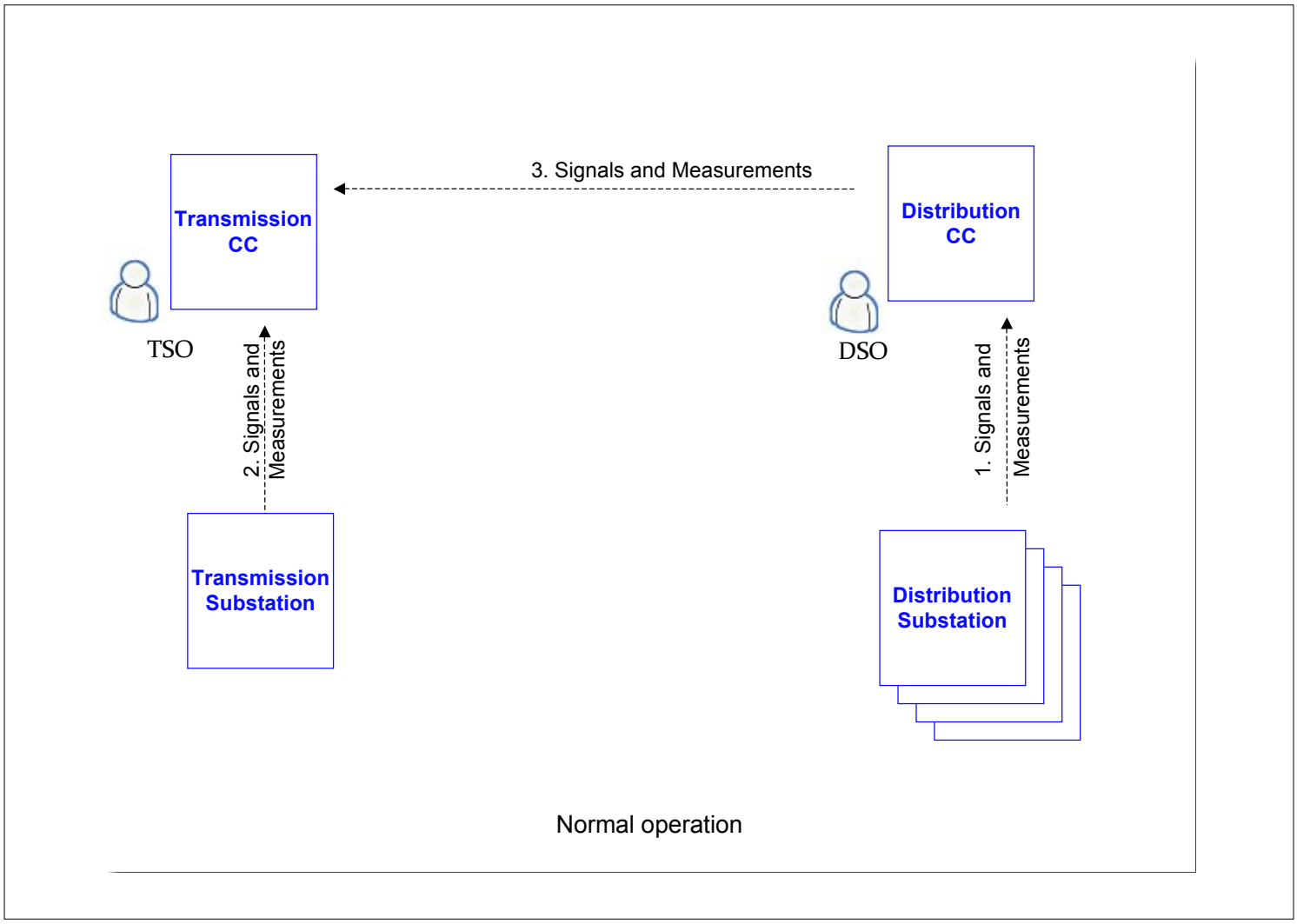
Timed automata model

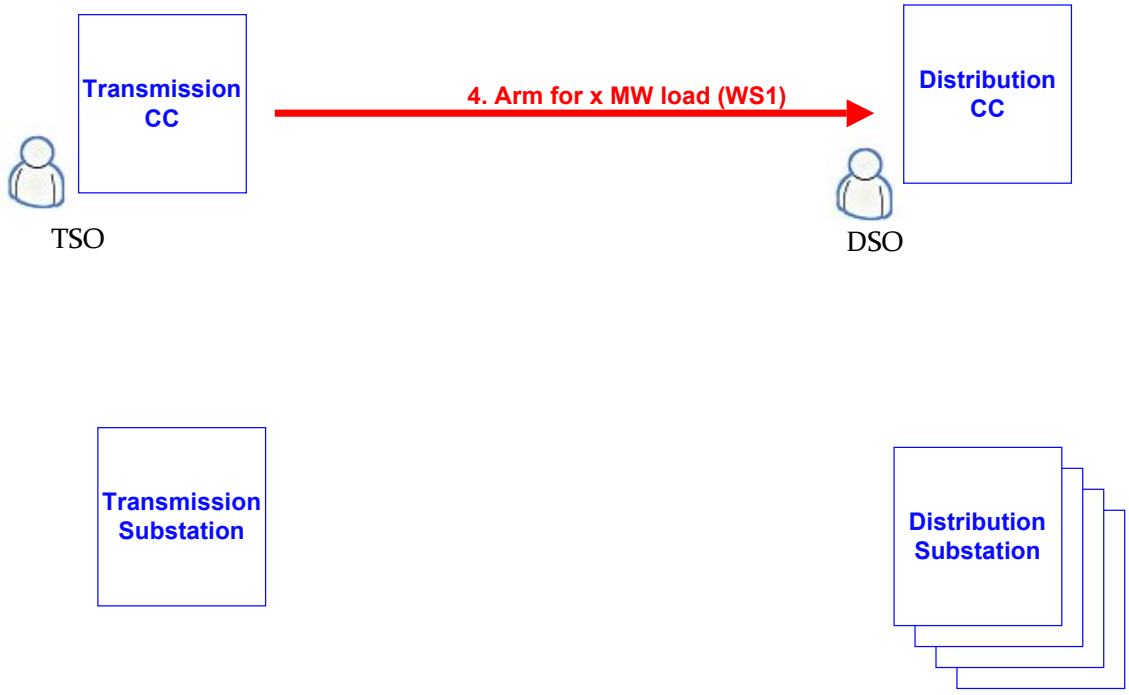
- **Permissions** = transitions between states,
  triggered by a valid WS message or by a local event
- **Prohibitions**
  - Implicit: no transition
  - Explicit: transitions to "failure states"
- **Obligations**
  - "Internal obligations" = transition automatically triggered by local events (guarded by time-out)
  - "External obligations" = those that must be realized by the other party, guarded by a local time-out:
    if a proof of achievement is not received before the time-out --> transition to an exception (obligation)

Ex: Emergency scenario in electric grid

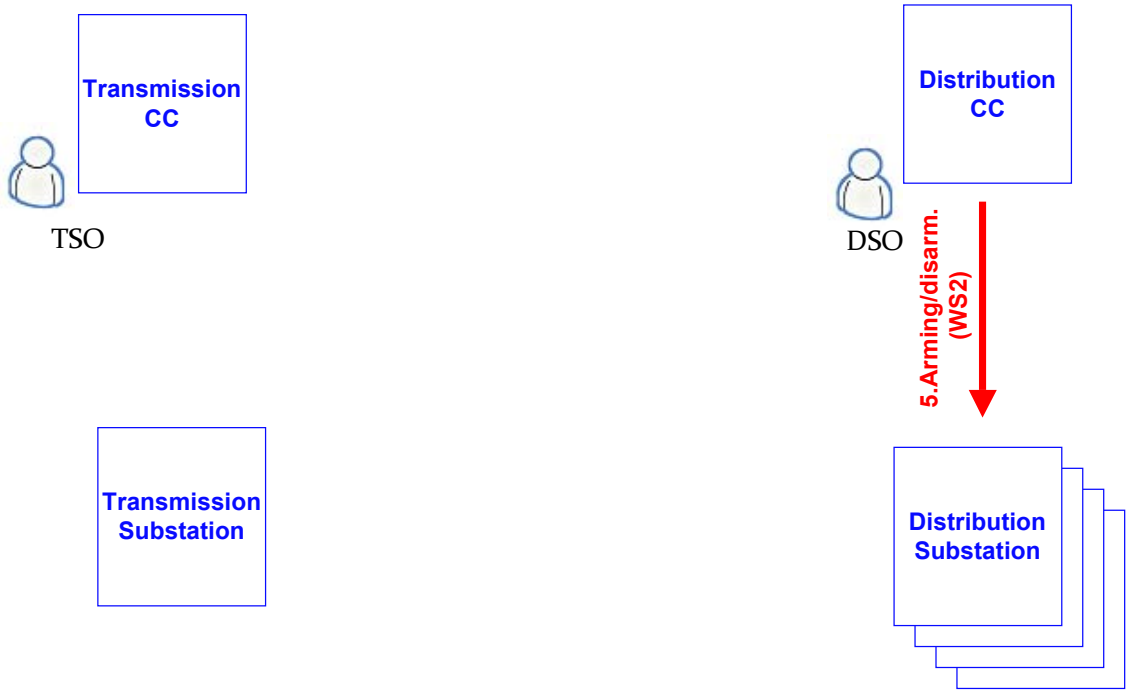



Transmission CC
TSO
Distribution CC
DSO
Transmission Substation
Distribution Substation

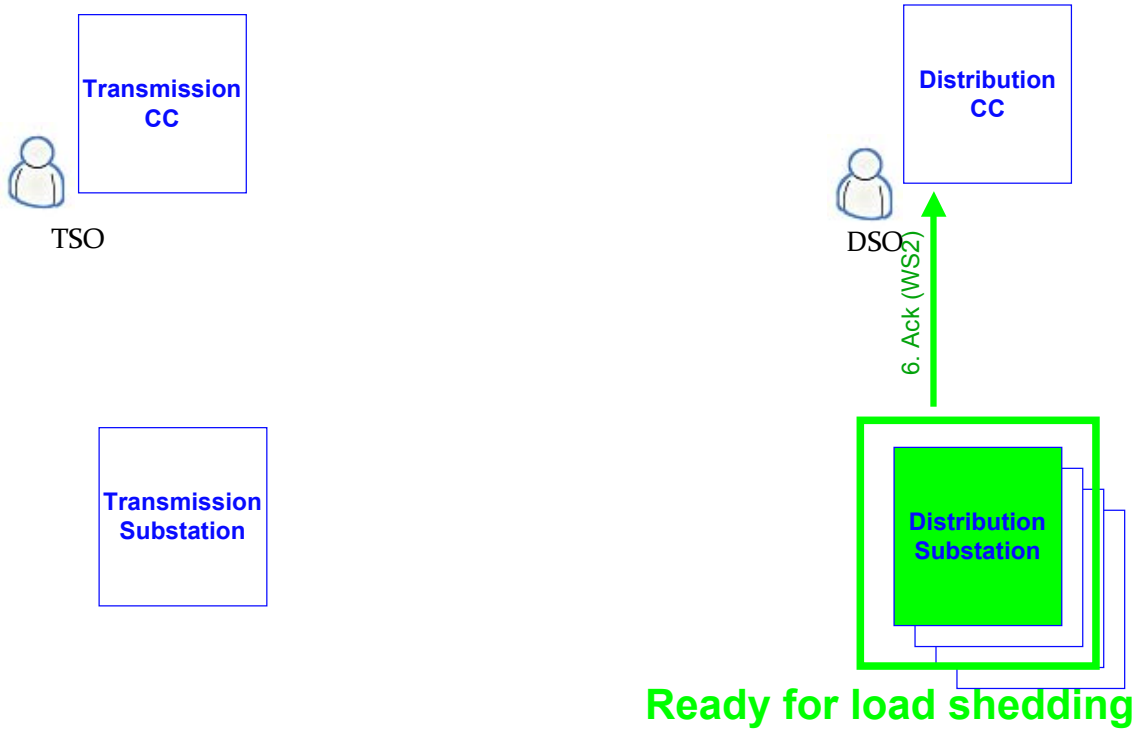

3. Signals and Measurements
Transmission CC
TSO
Distribution CC
DSO
2. Signals and Measurements
1. Signals and Measurements
Transmission Substation
Distribution Substation
Normal operation



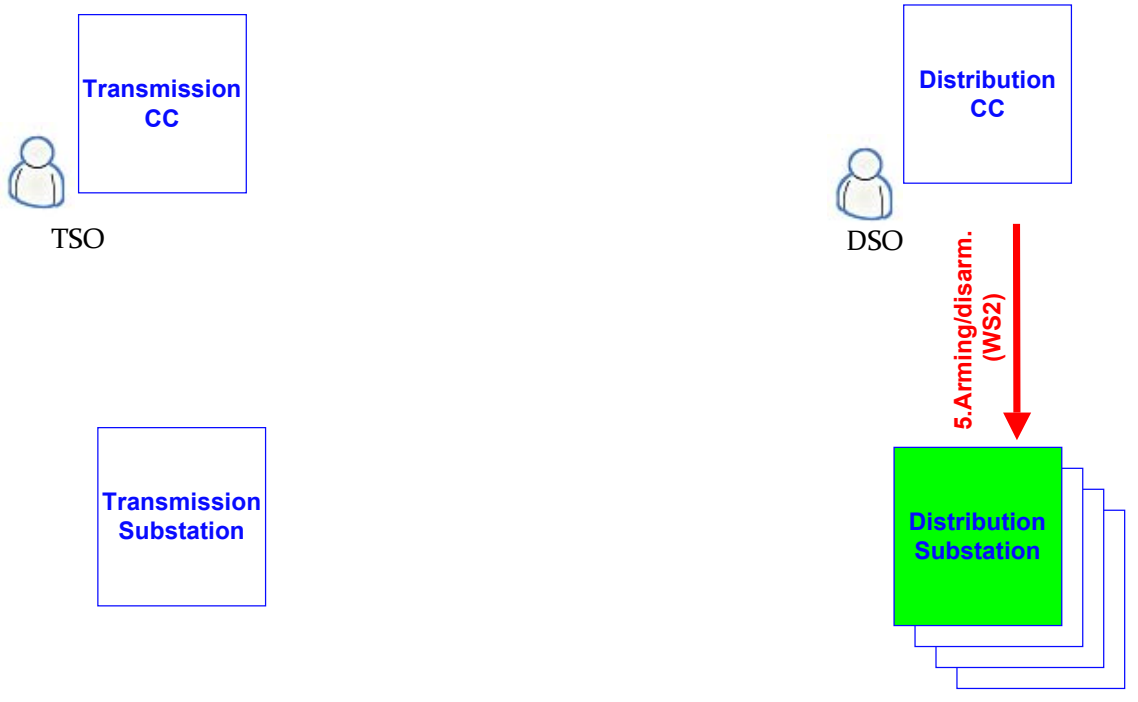
Prepare for possible load shedding



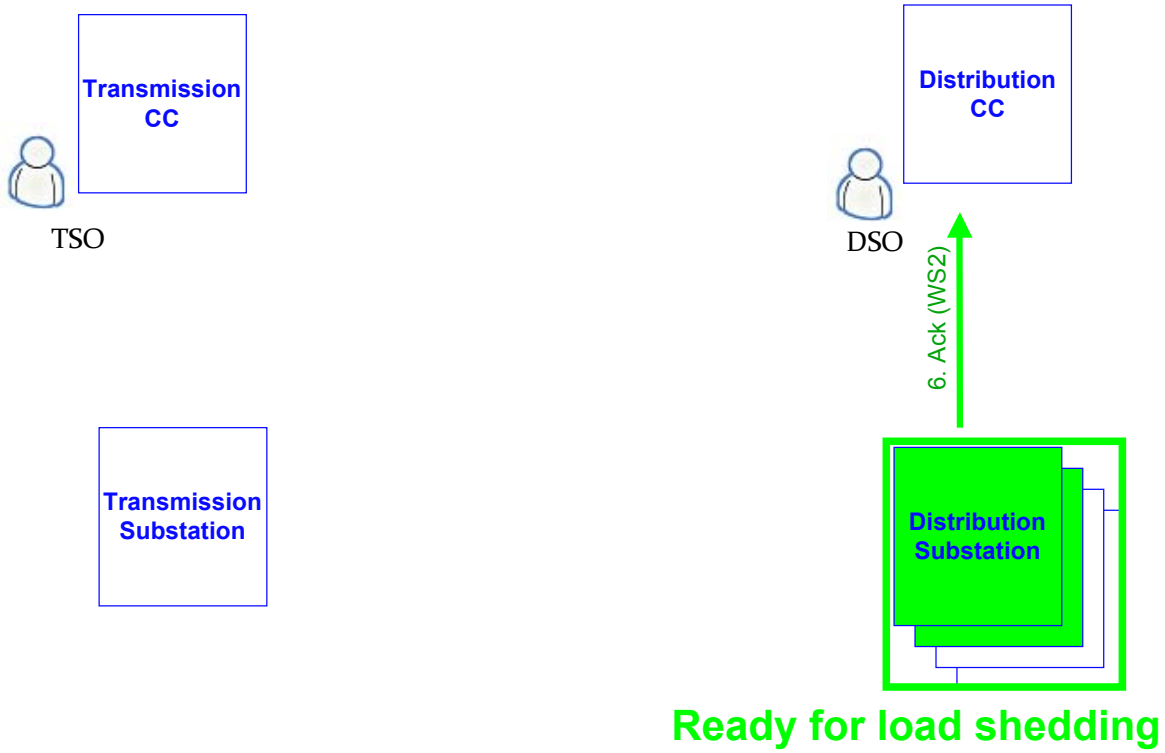
Prepare for possible load shedding



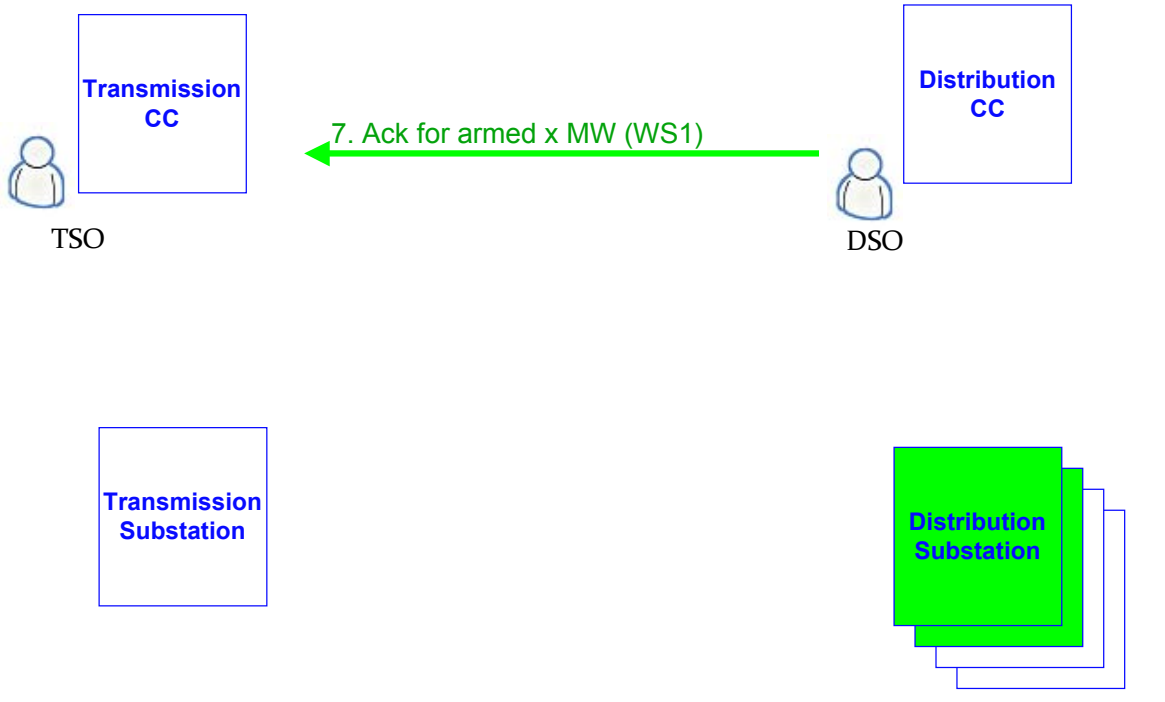
Prepare for possible load shedding



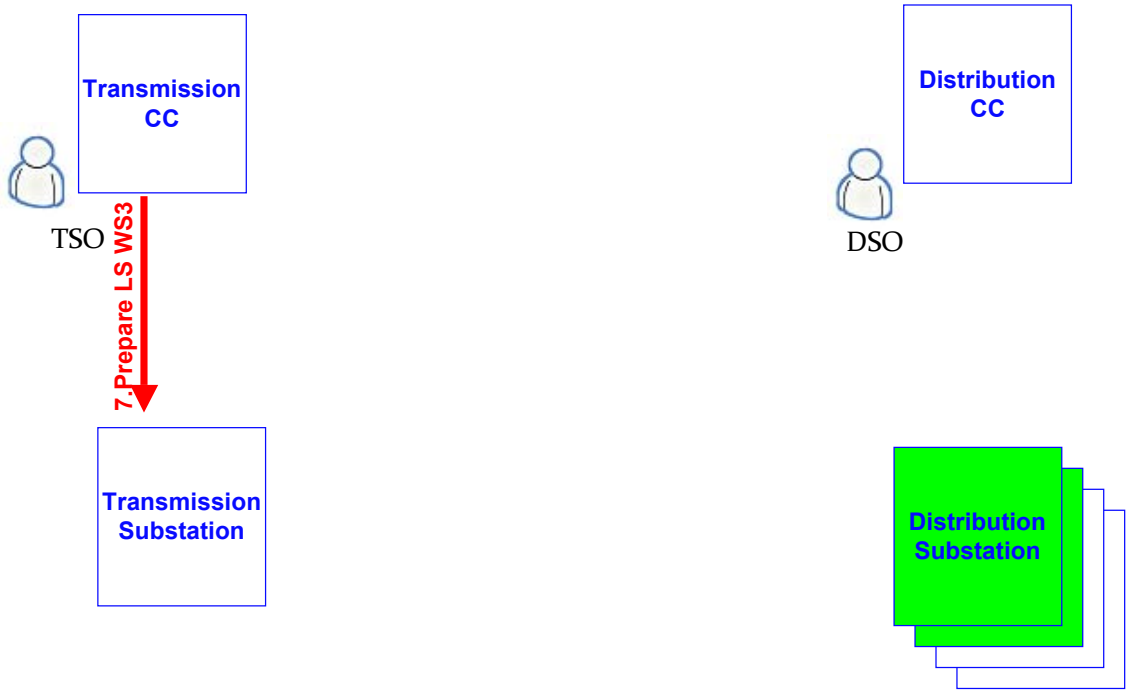
Prepare for possible load shedding



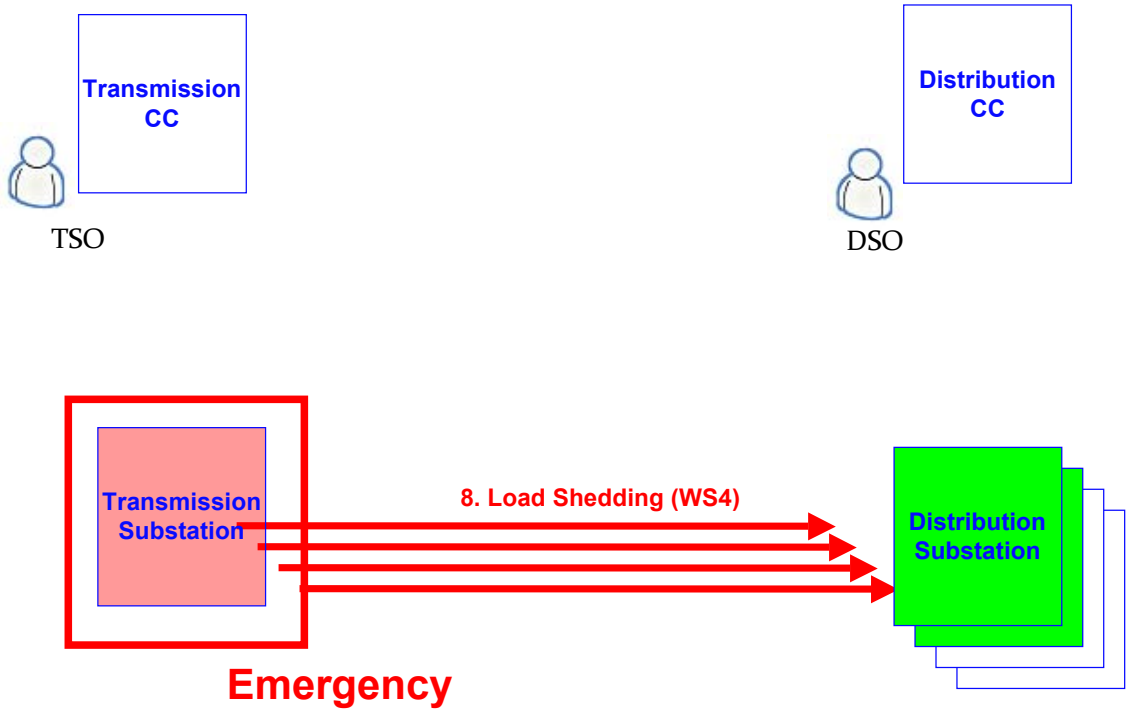
Prepare for possible load shedding

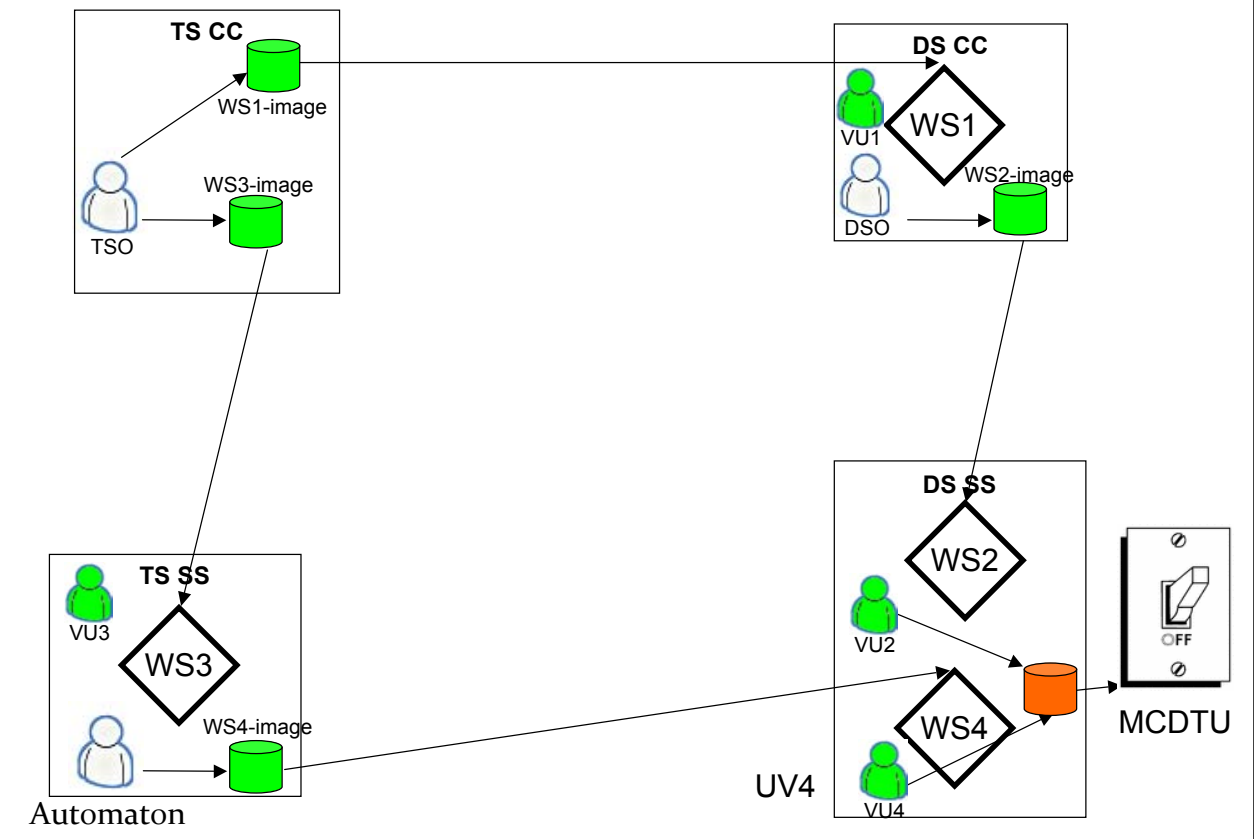


Prepare for possible load shedding



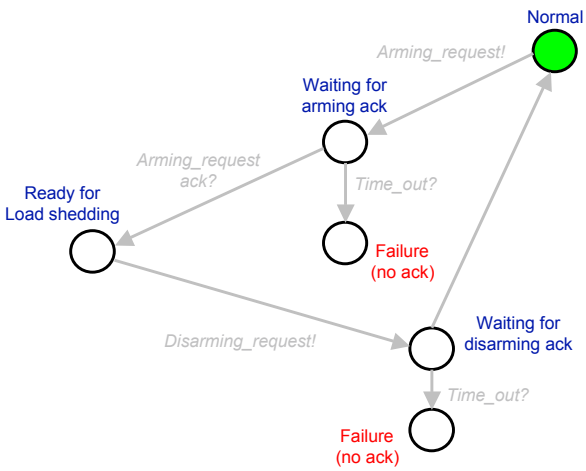
Prepare for possible load shedding



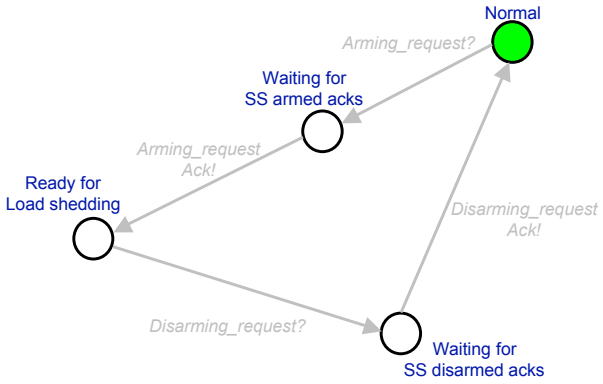


Run-time model checking: WS1

TS CC

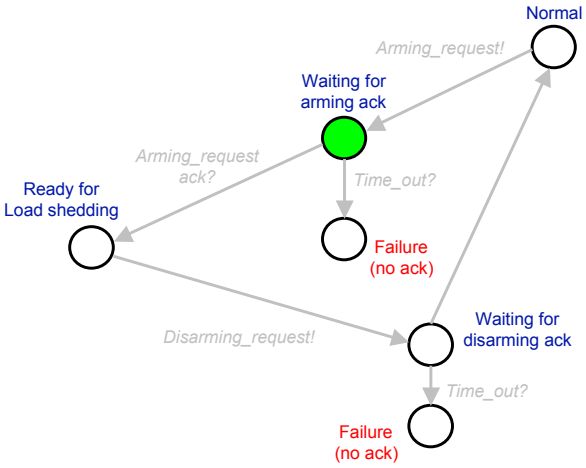


DS CC

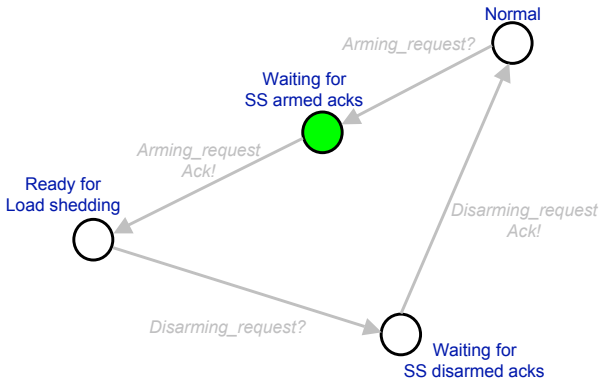


Run-time model checking: WS1

TS CC

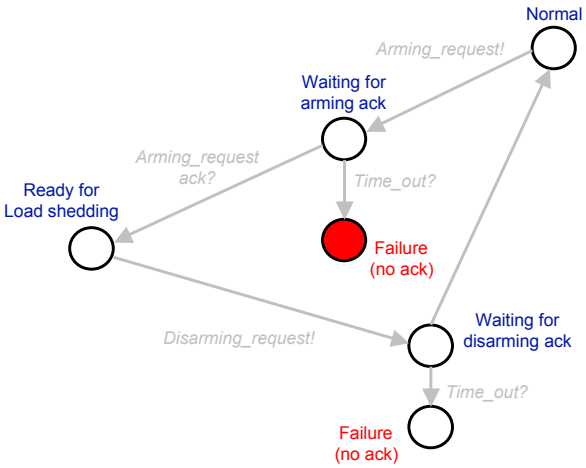


DS CC

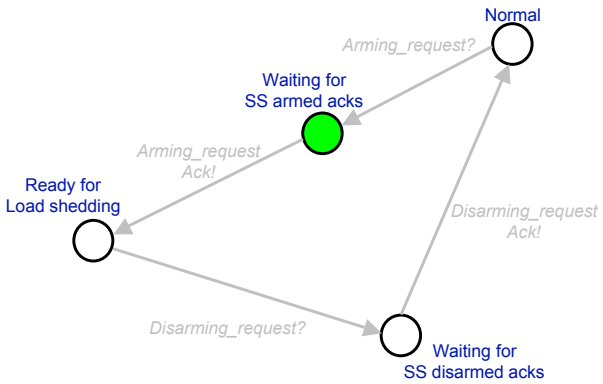


Run-time model checking: WS1

TS CC

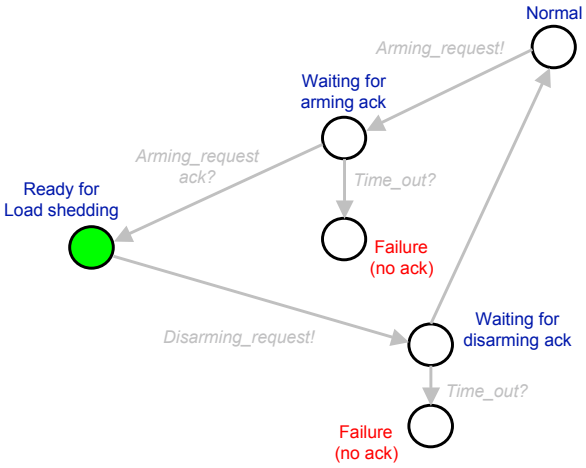


DS CC

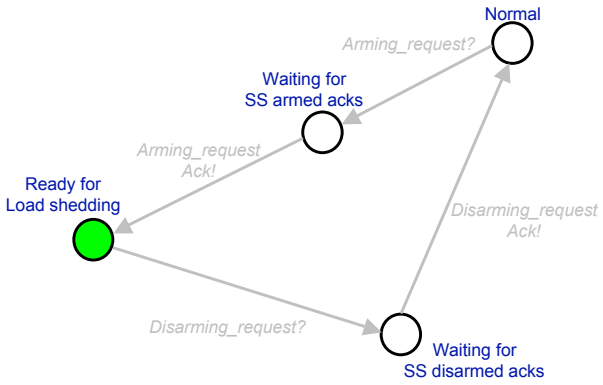


Run-time model checking: WS1

TS CC

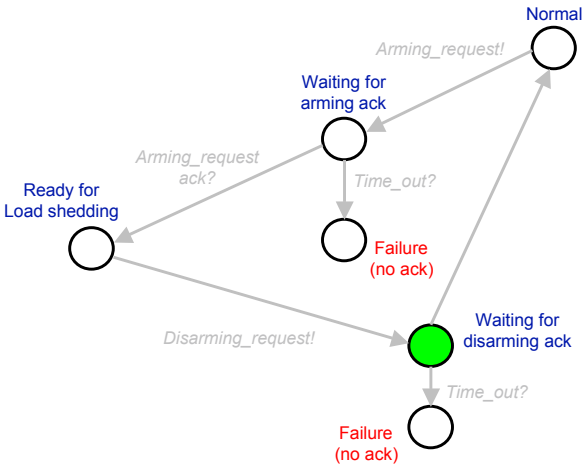


DS CC

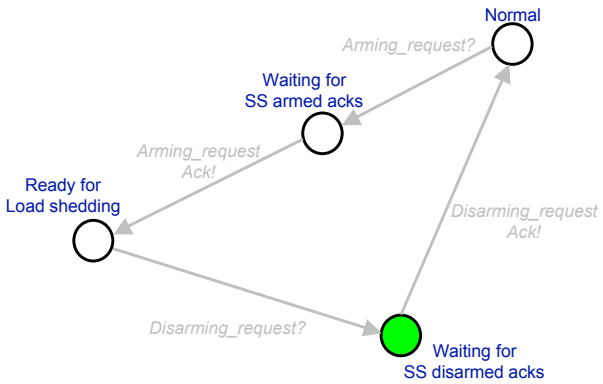


Run-time model checking: WS1

TS CC

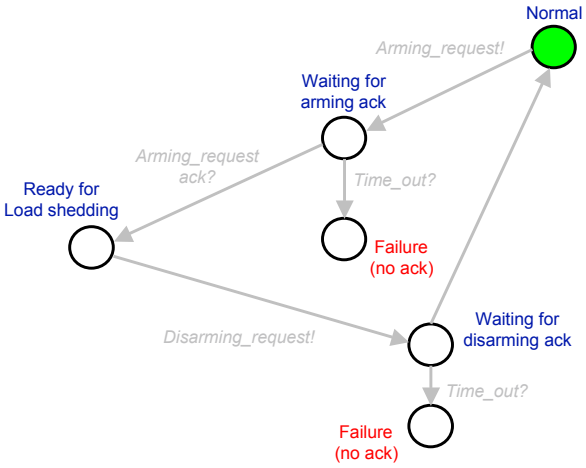


DS CC

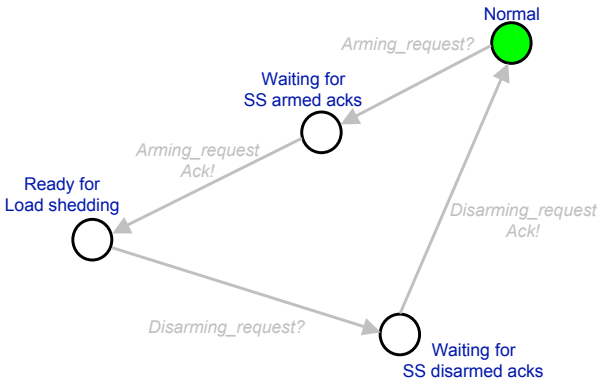


Run-time model checking: WS1

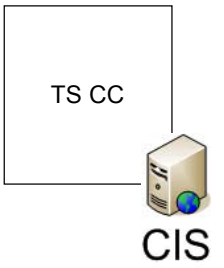
TS CC

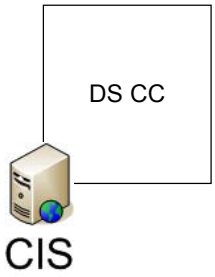


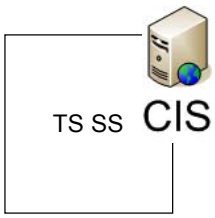
DS CC

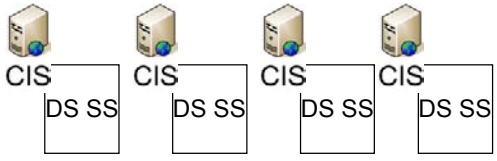


Experiment (simulation)

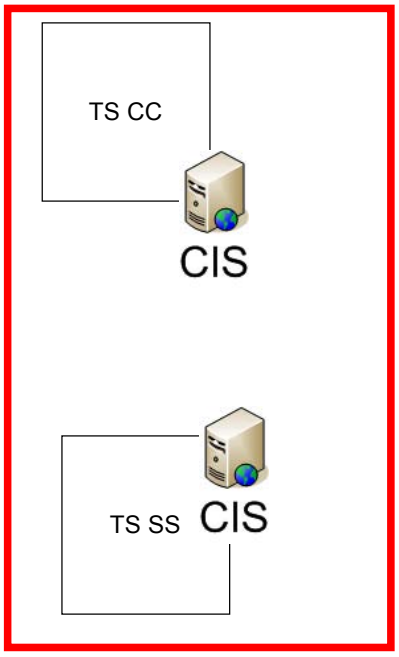




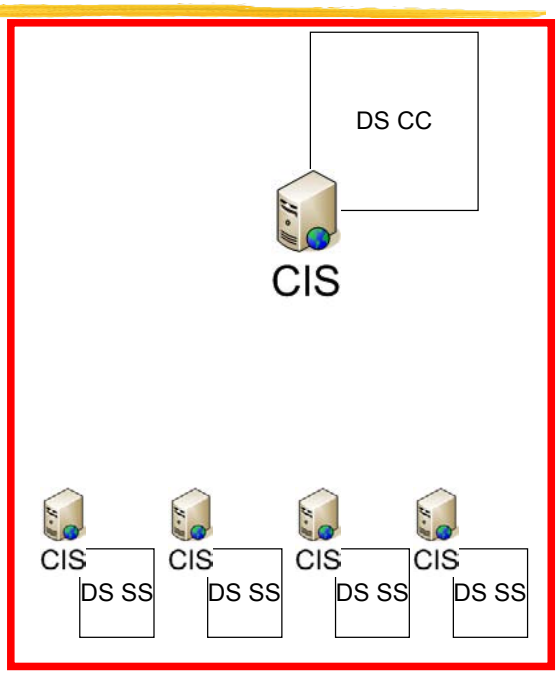




Experiment (simulation)

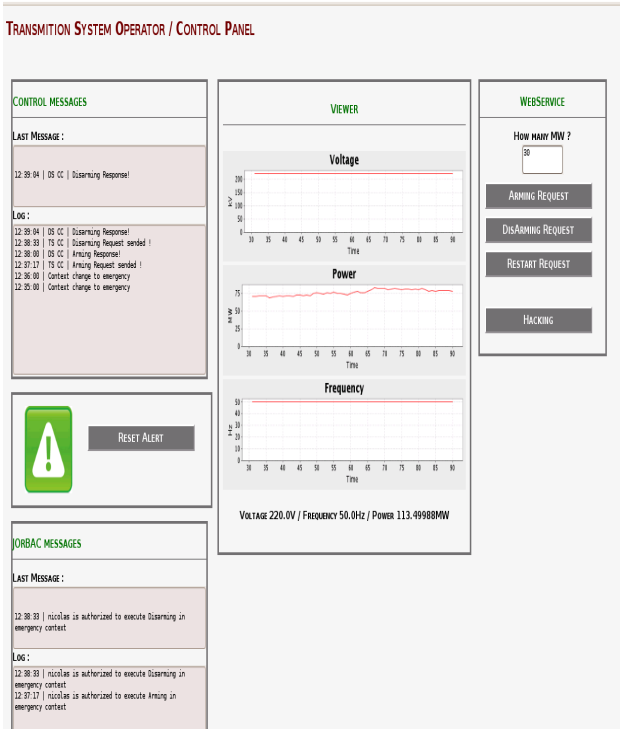


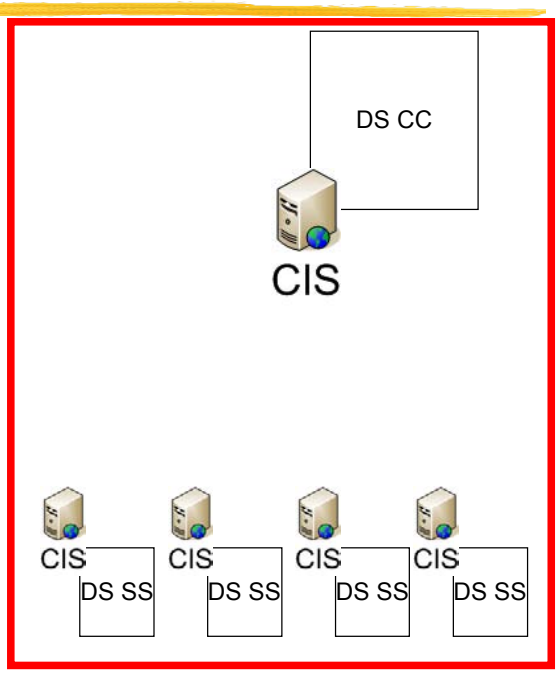
1 physical computer
(4 virtual machines)



1 physical computer
(10 virtual machines)

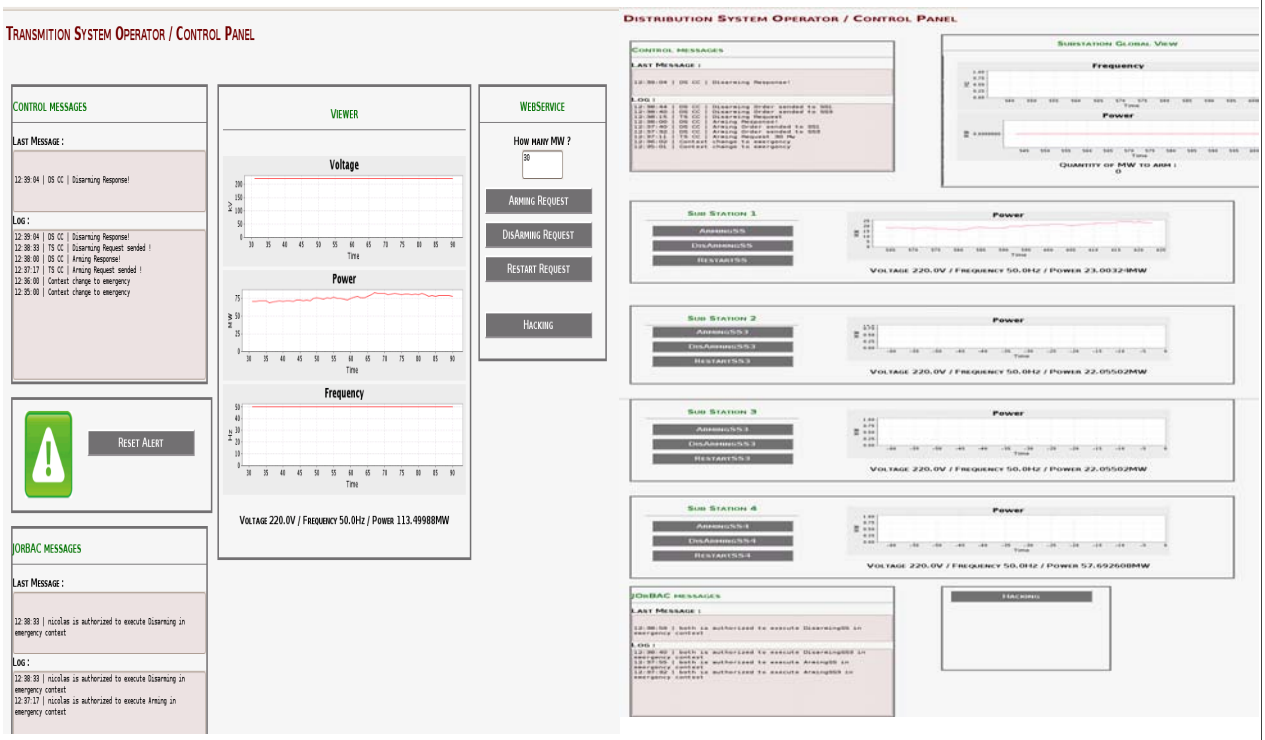
Experiment (simulation)





1 physical computer
(10 virtual machines)

Experiment (simulation)



Implementation

- 7 Web-Services (including simulated signals and measures), 4 users (TSO, DSO, + 2 low privilege users)
- 2 timed automata per WS, static verification of contracts (UPAAL)
- Operational code for 1 TCC (incl. Control Panel), 1 DCC (incl. Control Panel), 1 TSS, 4 DSS : WS execution, access control (OrBAC)
- 7 CIS (firewall, secure channels), interception of WS exchanges (messages) --> run-time model-checking
- One experiment control panel (configuration, experiment management, injection of external attacks)

Conclusion

- Autonomy, by local security provision
  - Each organization protects its own assets
  - Each organization is liable for its users
- Secure cooperation
  - Web Services, e-Contracts, run-time verification
- Monitoring and audit
  - Collection of evidence: message logs
- Scalability
  - Point-to-point => complexity $O(n)$ instead of $O(n^2)$
  - Very simple timed automata