Newcastle University

BANG & OLUFSEN

COMPASS

www.compass-research.eu

INCOSE International Symposium

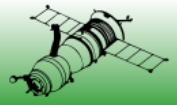
Las Vegas, NV
June 30 - July 3, 2014

# Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML


Jeremy Bryans;
John Fitzgerald; Richard Payne
(Newcastle University, UK)

Klaus Kristensen
(Bang & Olufsen, DK)

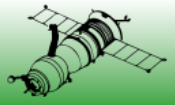





# Overview



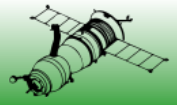
- Systems of Systems
- Case study
- Modelling
- Analysis
- Conclusions and future work


# Overview



- **Systems of Systems**
- Case study
- Modelling
- Analysis
- Conclusions and future work


# Systems of Systems



# Systems of Systems



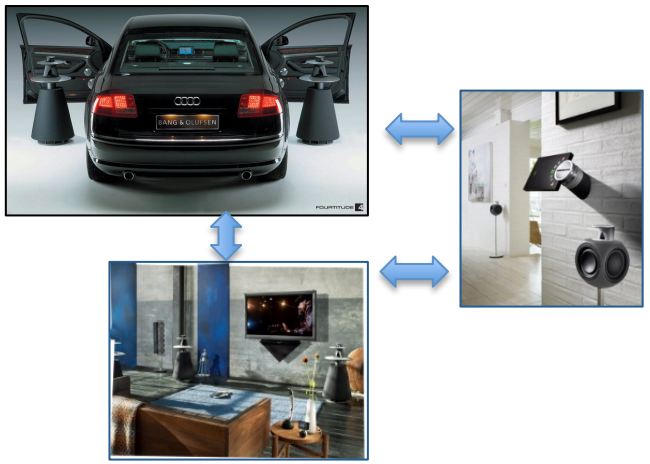

- Multiple content sources, DRMs,
- Multiple devices from multiple manufacturers
- Mobile and concurrent systems

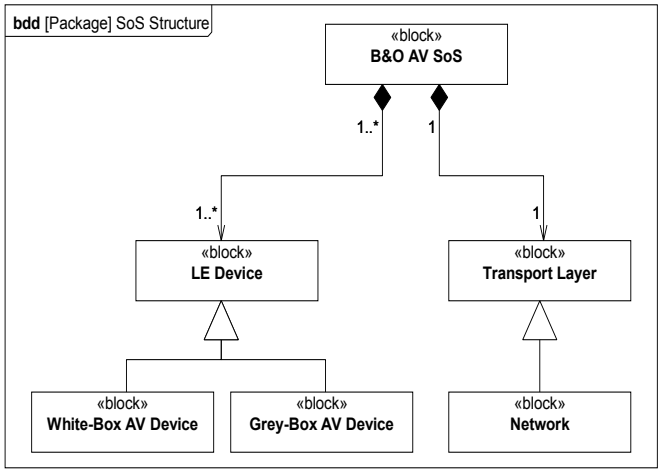
***Can we ensure consistent "user experience" as devices, content, DRM, etc., change?***

# Systems of Systems (SoS)



- Assembly and integration of independent systems that collectively offer a new ("emergent") service on which value and reliance is placed.
- Independence
- Distribution
- Evolution
- Emergence
- Model-based SoS Engineering as a way of mastering complexity





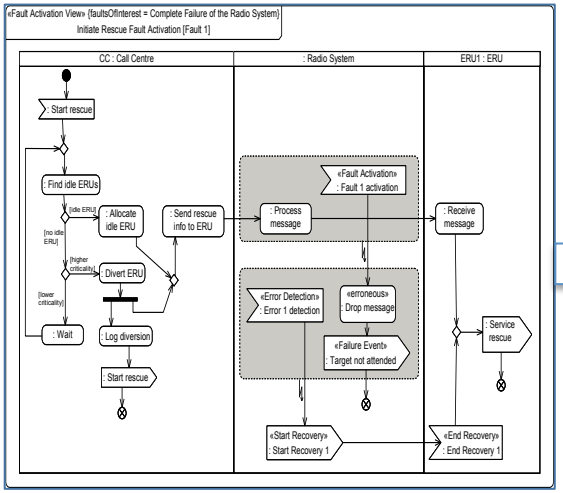
# SoS Engineering Challenges



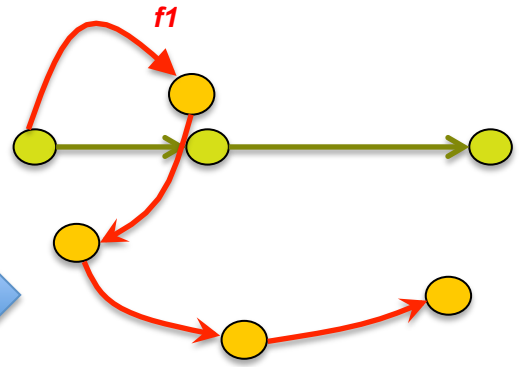
- Independence and autonomy of constituent systems
  - Constituent systems evolve at the behest of their owners
  - *Response: Collaborative SoS modelling by contractual (assume, commit) interface specification*
- Complexity of confirming/refuting SoS-level properties
  - Verification of emergence
  - *Response: verified refinement for engineering of emergent properties; simulation tools allow exploration for unanticipated behaviours*
- Semantic heterogeneity (integrating models)
  - Wide range of interacting features in models (e.g. location, time, concurrency, data, communication)
  - *Response: extensible semantic basis*

# COMPASS Technology

INCOSE International Symposium
Las Vegas, NV
June 30 - July 3, 2014




```
actions
 MERGE1(r) =
  (dcl e: set of ERUId @ e := findIdleERUs();
   (do
     e = {} -> DECISION2(r)
    |
     e <> {} ->
      (dcl e1: ERUId @ e1 :=
        allocateIdleERU(e, r); MERGE2(e1, r))
   end)) …
process InitiateRescue = CallCentreProc [|
SEND_CHANNELS |] RadioSystemProc [|
RCV_CHANNELS |] ERUsProc
```




(SoS || STOP) [= $\mathcal{L}_F$(SoS)

Symphony

## SysML modelling

- Guidelines for Requirements, Architecture, Integration
- SoS Modelling profiles, e.g. Fault-Error-Failure
- Architectural patterns and extensible frameworks

## Formal Modelling Language

- CML allows representation of behavioural semantics of the SoS
- Supports contract specification
- Describes functionality, object-orientation, concurrency, real-time, mobility.
- Can be extended to new paradigms

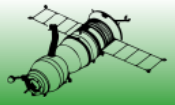
## Tool-supported Analysis

- Model-checker
- Automated proof
- Test generation
- Simulation
- Model-in-Loop Test
- Exploration of design space

# Overview



- Systems of Systems
- **Case study**
- Modelling
- Analysis
- Conclusions and future work


# AV SoS Case Study



- CSs are *heterogeneous* and may *evolve* (through software or firmware upgrades)
- New CSs may be *integrated* into SoS at any time
- CSs may be *legacy* or *non-B&O systems*

# AV SoS Case Study

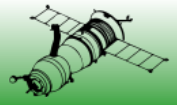


- Challenge: verifying emergence – can a single “leader” be established to maintain global clock, SoS architecture, streaming details, …?

# Overview



- Systems of Systems
- Case study
- **Modelling**
- Analysis
- Conclusions and future work

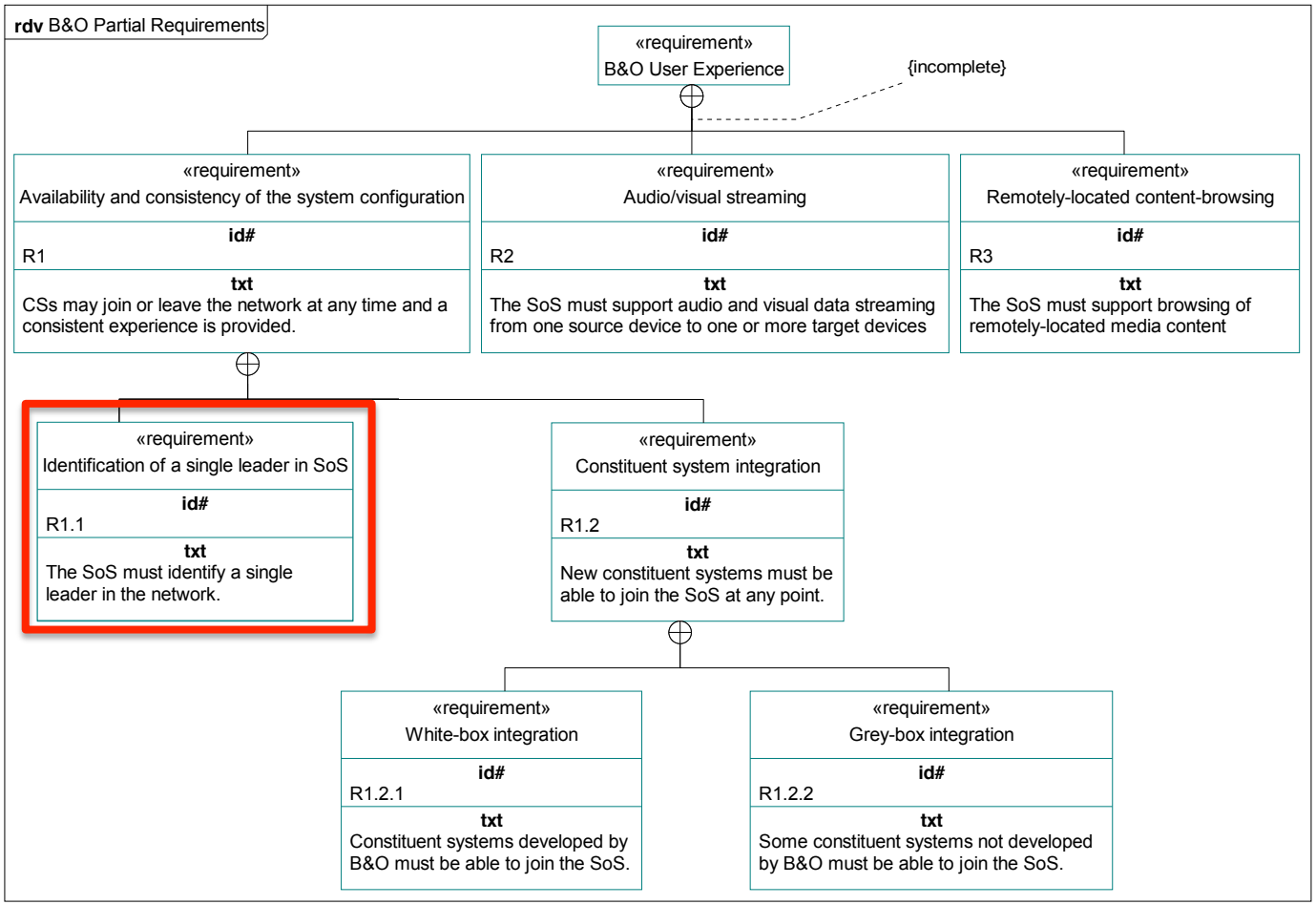

# Modelling Approach



- Use SoS-ACRE – COMPASS Requirements Engineering guidelines
- Define SoS composition and contracts in SysML using Contracts Pattern

# Requirements Definition





# The Contracts Pattern

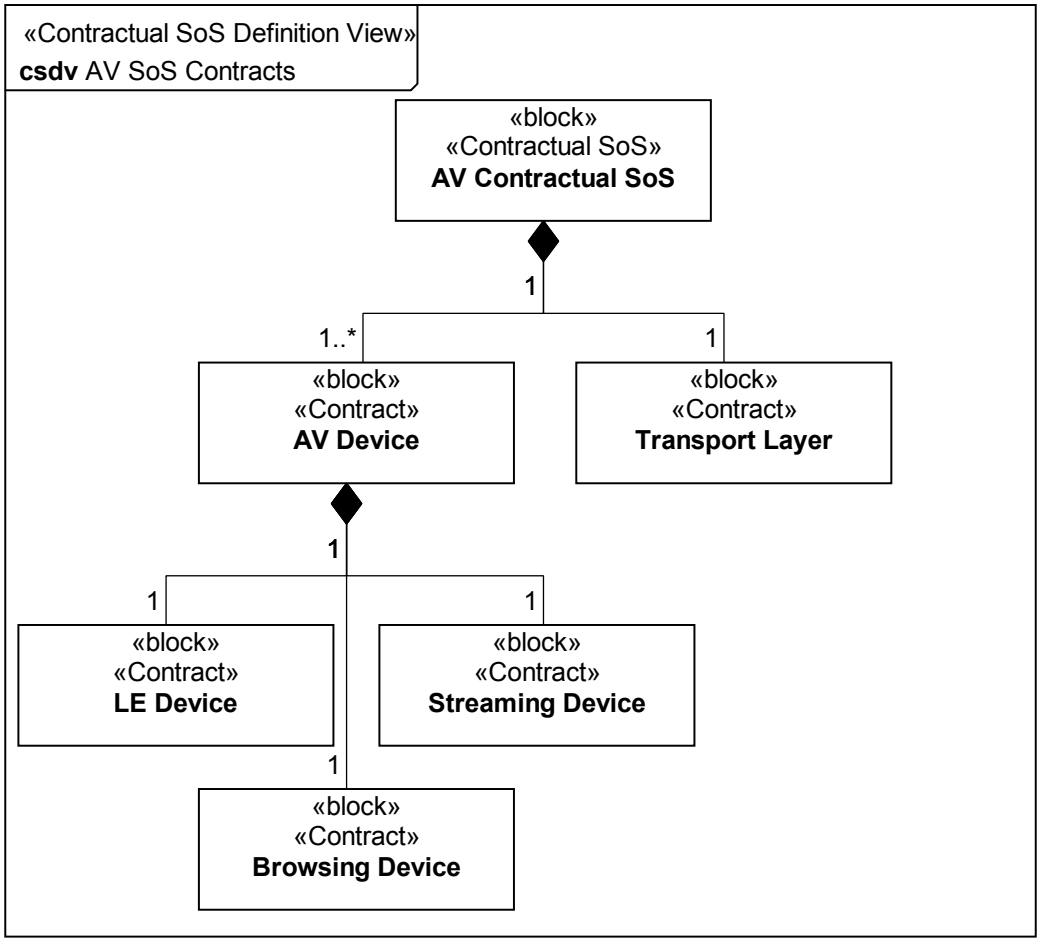


- What is a contract?
  - A description of the "minimum" behaviour that a CS must exhibit in order to be part of an SoS
- What is the Contract Pattern?
  - Collection of viewpoints for modelling the contracts of a SoS
  - Defined and implemented using SysML
  - Notation agnostic

See also: Bryans, J.; Fitzgerald, J.; Payne, R.; Miyazawa, A.; Kristensen, K. *SysML Contracts for Systems of Systems,* In Proceedings of IEEE SoSE 2014

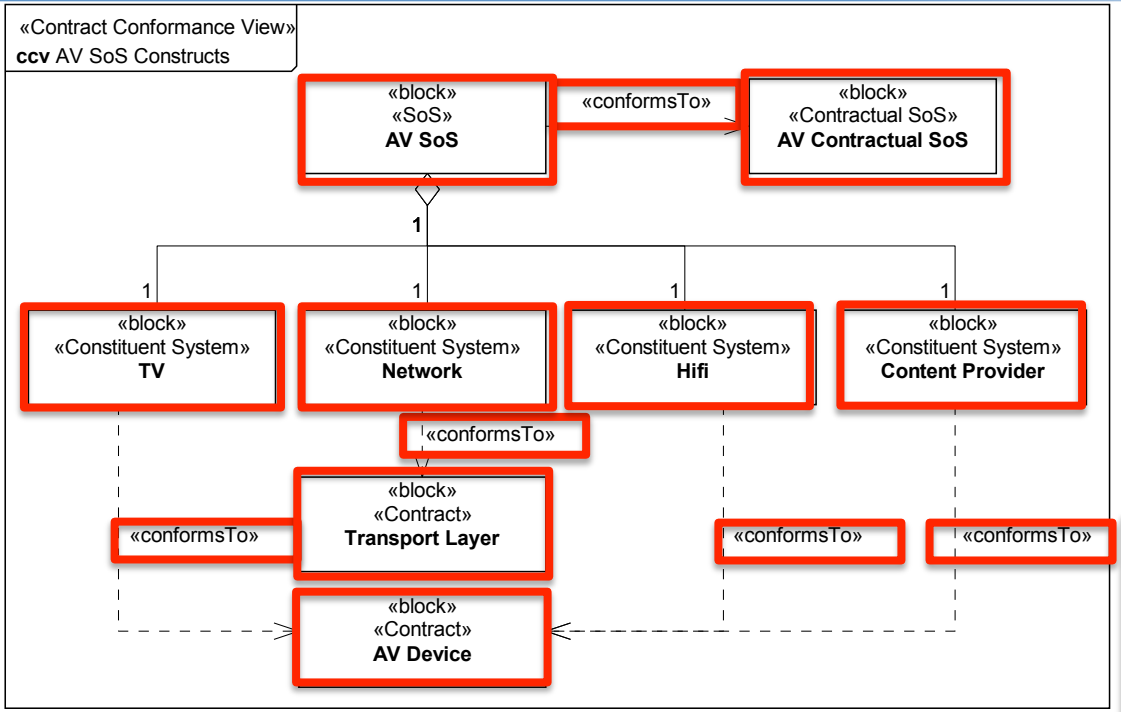
# Contractual SoS Definition View

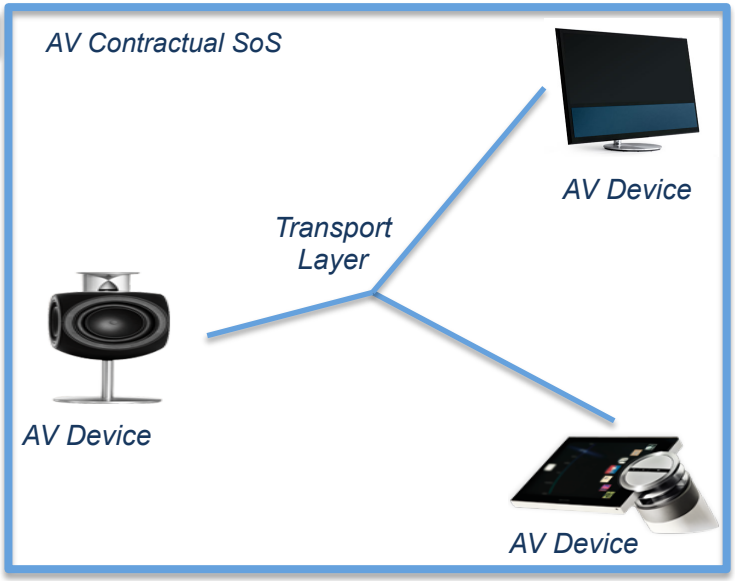




# Contract Conformance View

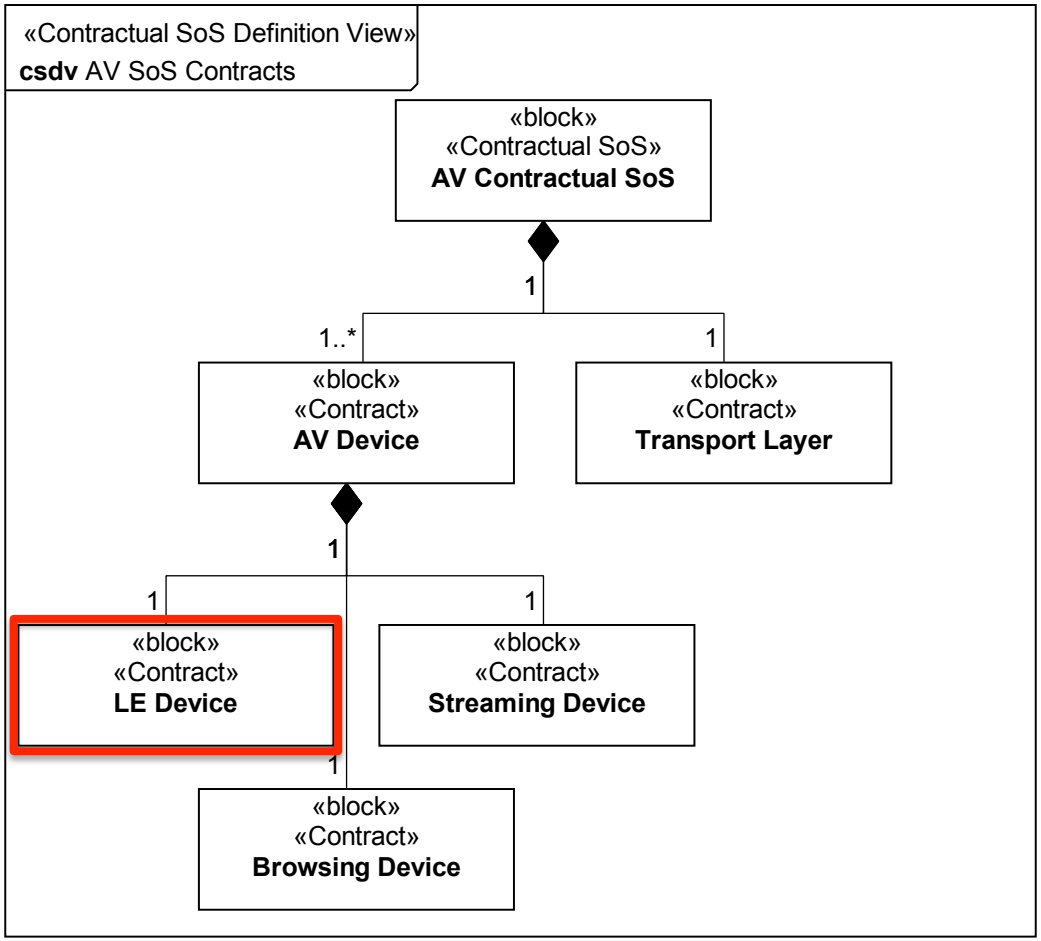




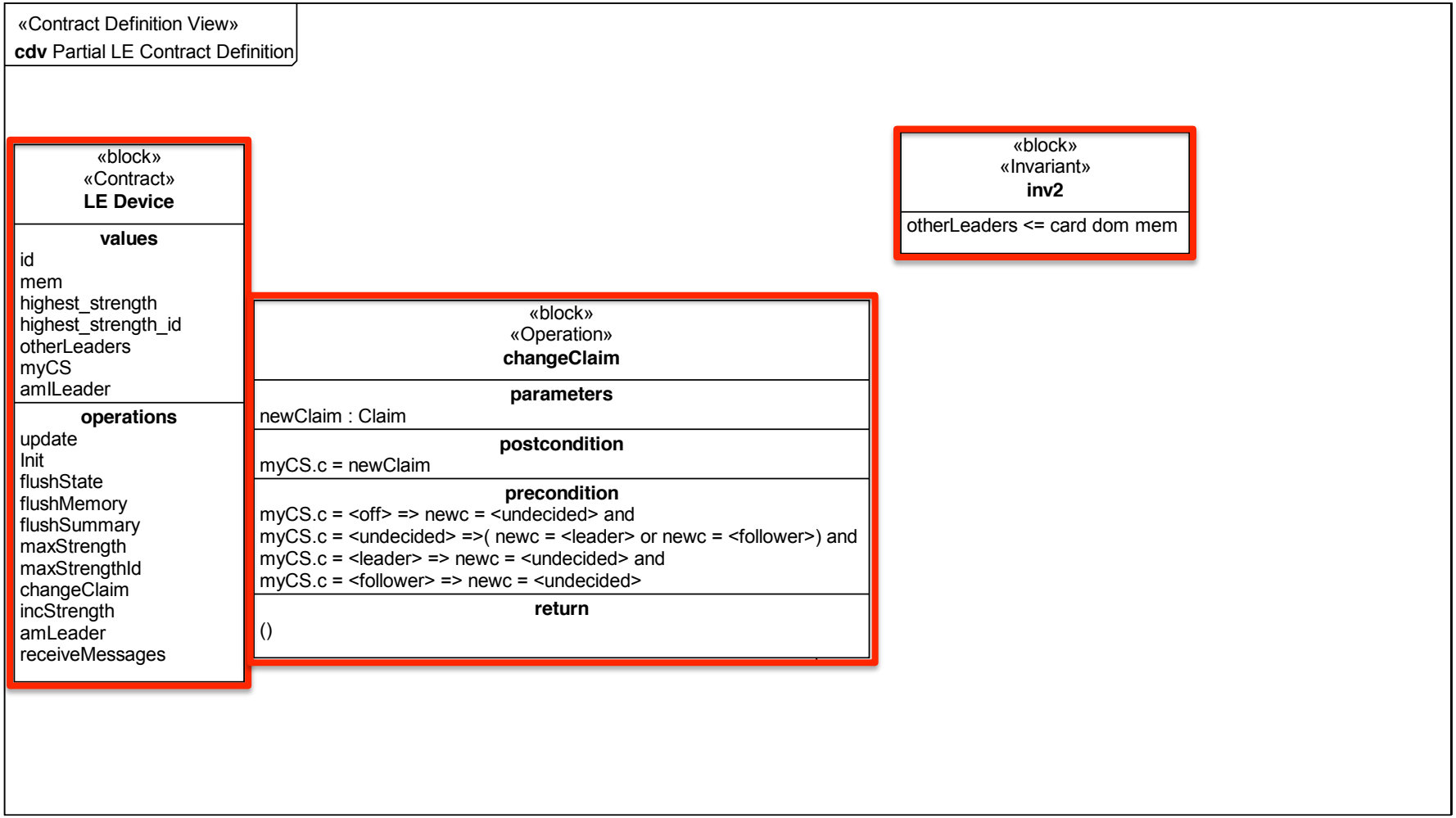
# Defining a Contract





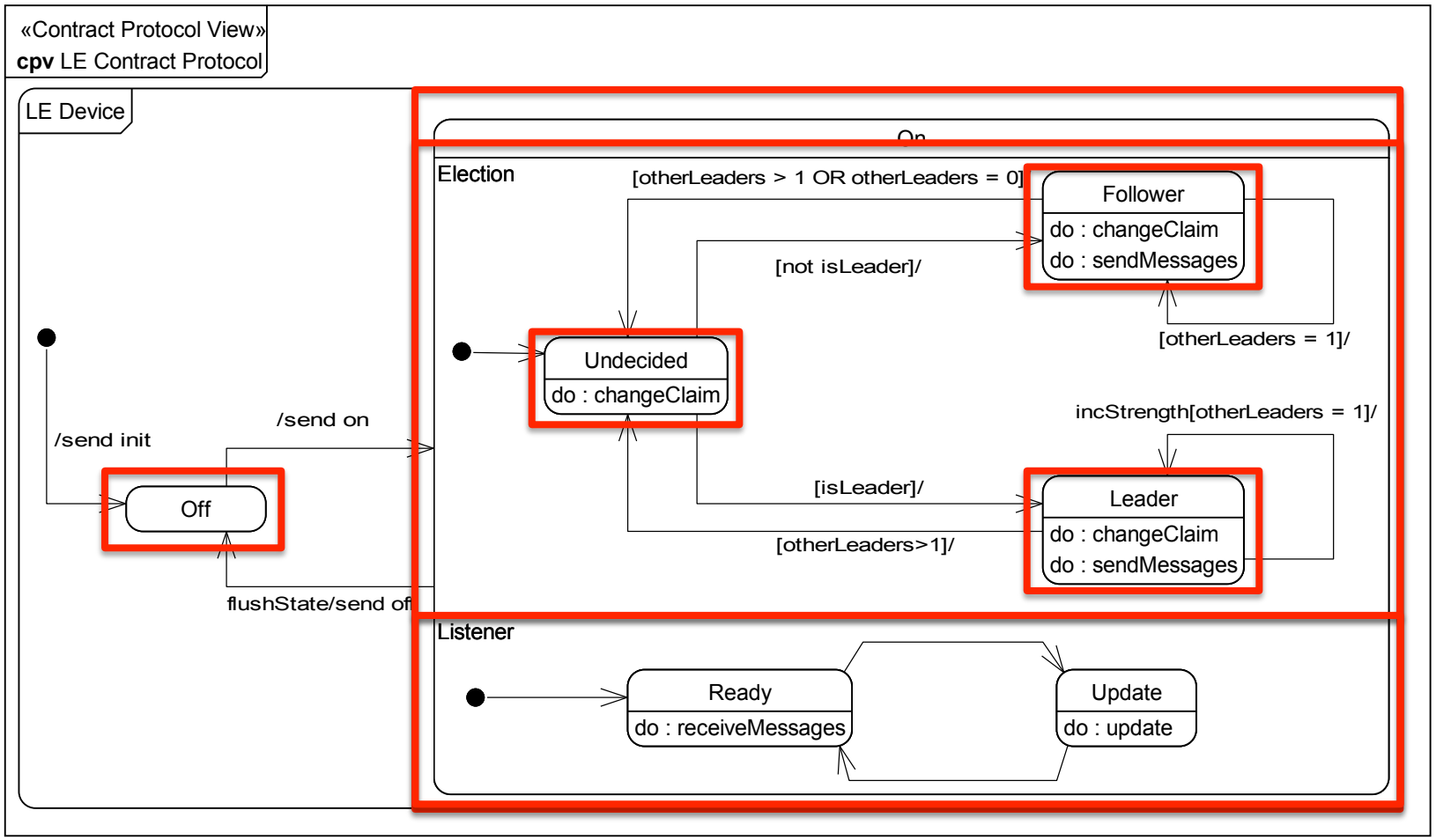
# LE Device Contract Definition View





# LE Device Contract Protocol View

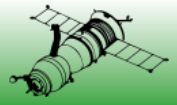




# Overview



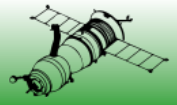
- Systems of Systems
- Case study
- Modelling
- **Analysis**
- Conclusions and future work


# Model-based Analysis

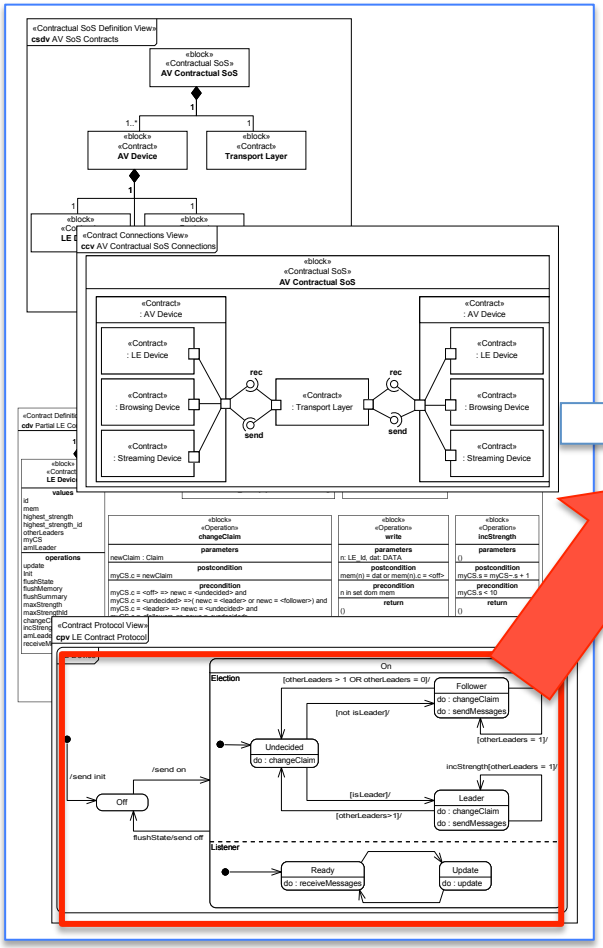


- Translate SysML contract model to formal notation COMPASS Modelling Language (CML)
  - Contracts are defined in terms of communicating *processes*
  - Processes contain *datatypes*, *variables*, *operations* and *actions*
- Verify requirement of emergent behaviour using CML tool Symphony
  - Formal semantics allows range of formal analyses

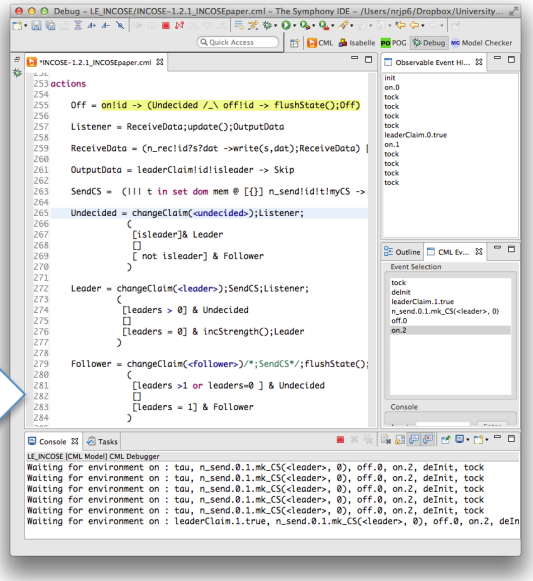




# Analysing the Model






```
process LE_Device = i : nat @
begin
  …
  actions
    Off = on!id -> (Undecided /_\ off!id
      -> flushState();Off)
    Undecided = changeClaim(<undecided>);
      Listener;([isleader]& Leader
                []
                [not isleader]& Follower)
    Leader = …
    Follower = …
    Listener = …
end

process TransportLayer =
begin
  …
end

process AllLEDevices =
      || i in set le_ids @ (LE_Device(i))

process AVSoS= AllLEDevices
                [|{|interface_channels|}|]
                 TransportLayer
```

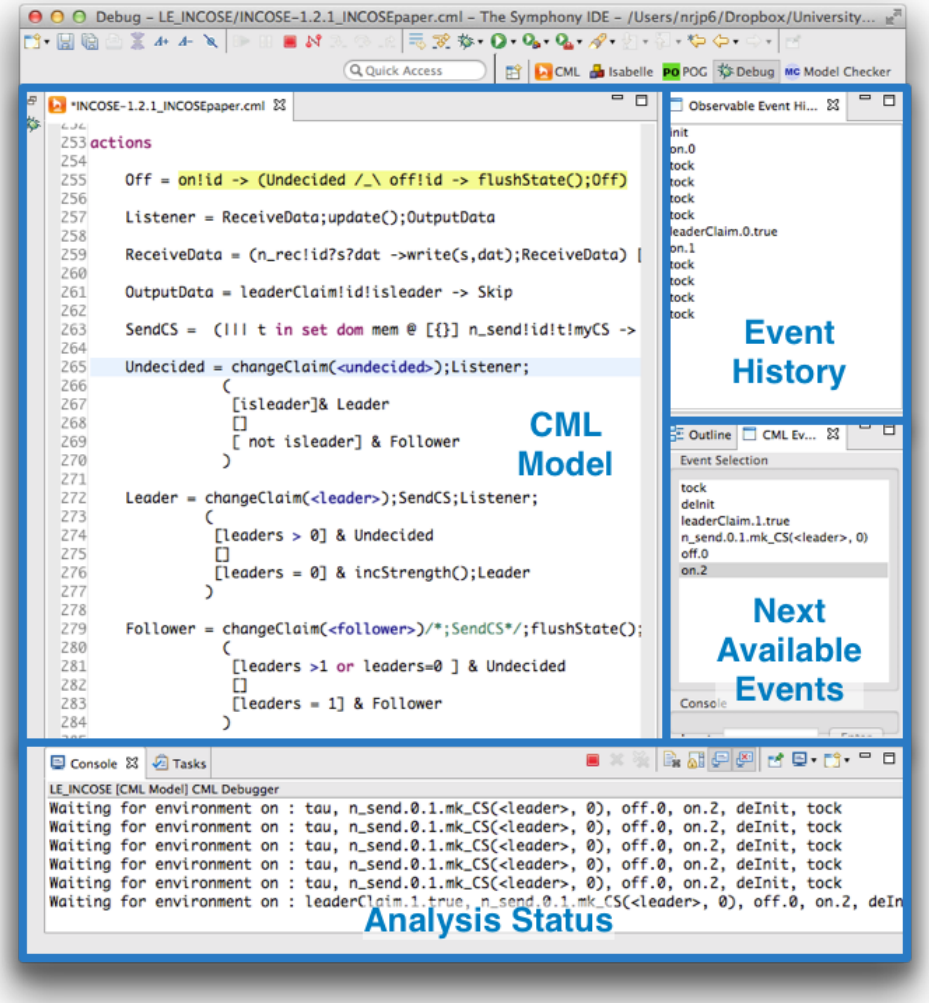



## Symphony tool

- Analyse leader election emergent behaviour
- Simulate execution of model
- Model checking

# CML Model Simulation



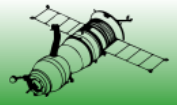



- Used Symphony simulator to execute traces of CML model
- Model does not meet requirement R1.1
  - Can have more than one leader
  - However, quickly resolved
  - Incorrect model or incorrect requirement?
- New CSs may be added and emergent behaviour maintained

# Overview



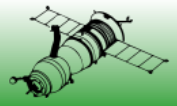
- Systems of Systems
- Case study
- Modelling
- Analysis
- **Conclusions and future work**


# Conclusions



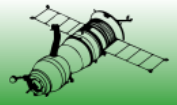
- Established need for contractual definition of constituent systems
- Defined and demonstrated contracts pattern with industrial proof of concept study
  - Using SysML and CML
- Demonstrated analysis of CS contracts to ensure required emergence is maintained
  - Simulation of CML model
  - Resulting in clarification of requirements


# Future Work



- Integrate SoS engineering frameworks
  - e.g. fault modelling and analysis, testing
- Enhance contract pattern
  - non-functional properties and security features
- Modelling SoS-level contracts in pattern
- Consequences of contract composition
- Automated contract conformance

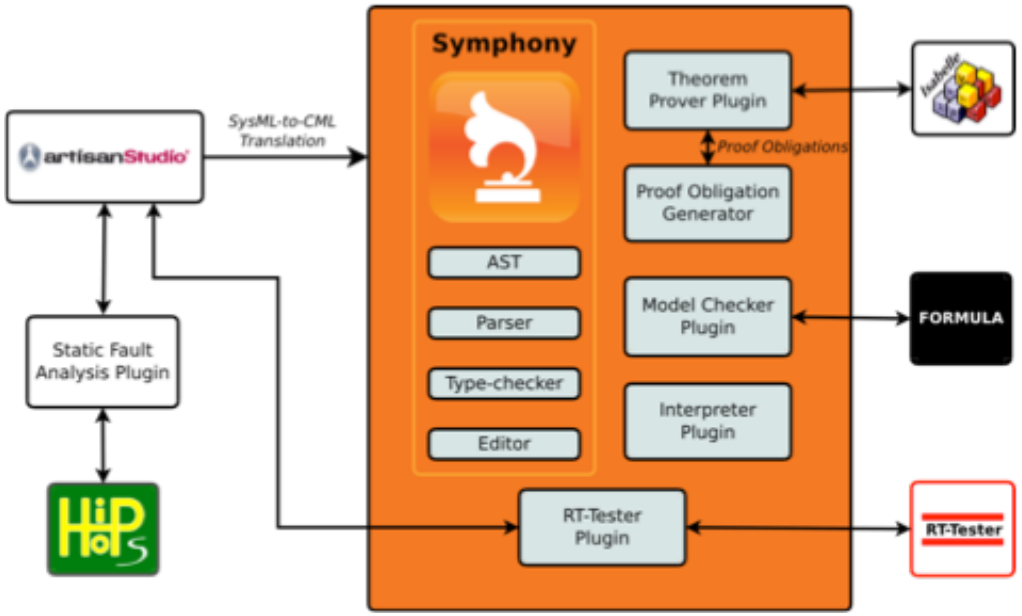

INCOSE International Symposium
Las Vegas, NV
June 30 - July 3, 2014

# COMPASS SUMMER SCHOOL
## New Developments in Model-Based SoS Engineering

Radisson Blu Edwardian Vanderbilt, London,
17-18 September 2014

Learn about the COMPASS methods and tools:

- Architectural SoS modelling in SysML & CML
- The Symphony tool
- Model-based testing




## More information:

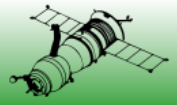
http://www.compass-research.eu/summerschool.html

Newcastle University

john.fitzgerald@ncl.ac.uk @NclFitz
richard.payne@ncl.ac.uk @riffio

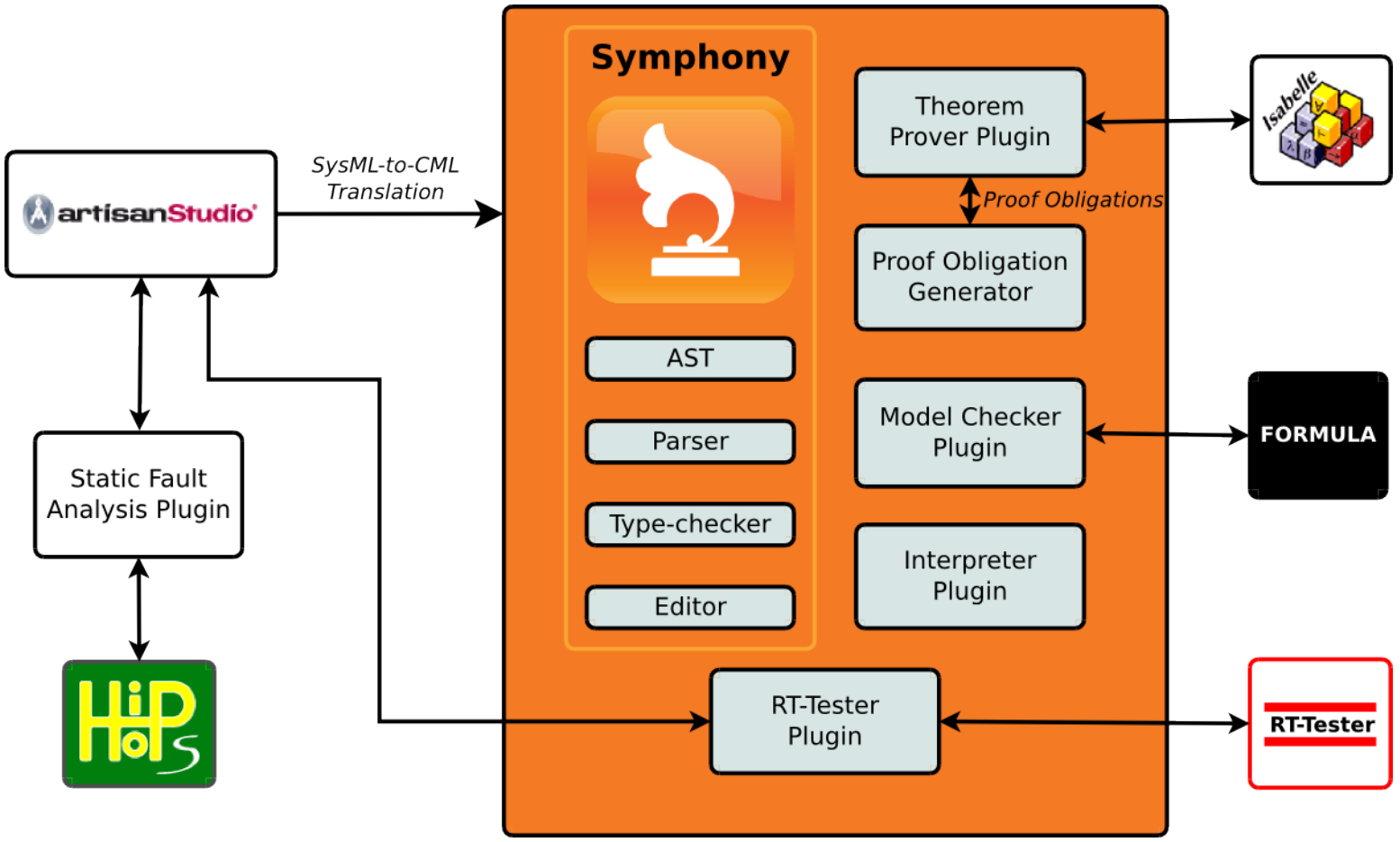
COMPASS

www.compass-research.eu
www.compass-research.eu/summerschool.html

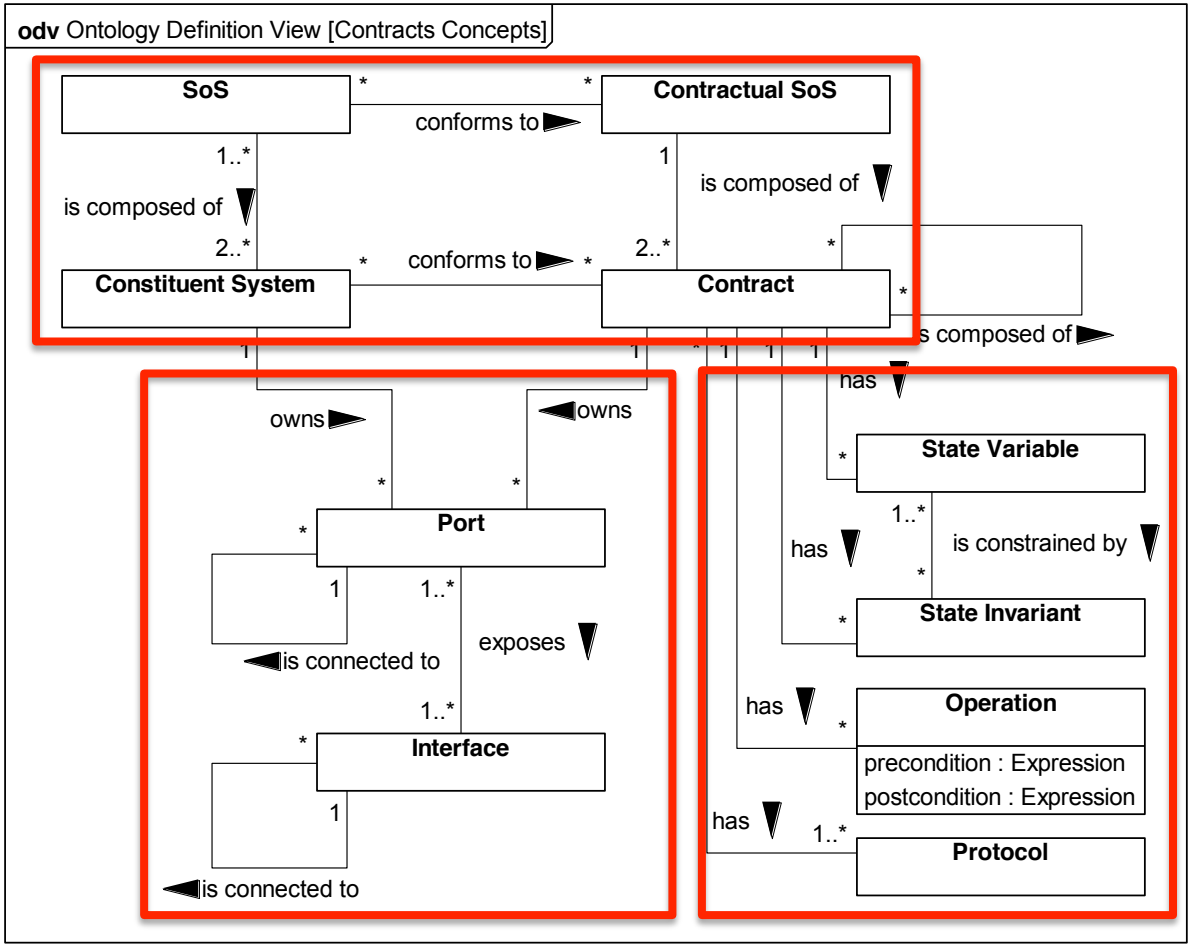

# COMPASS Tool Architecture





# Contract Pattern - Ontology

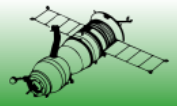




# Contract Pattern - Views



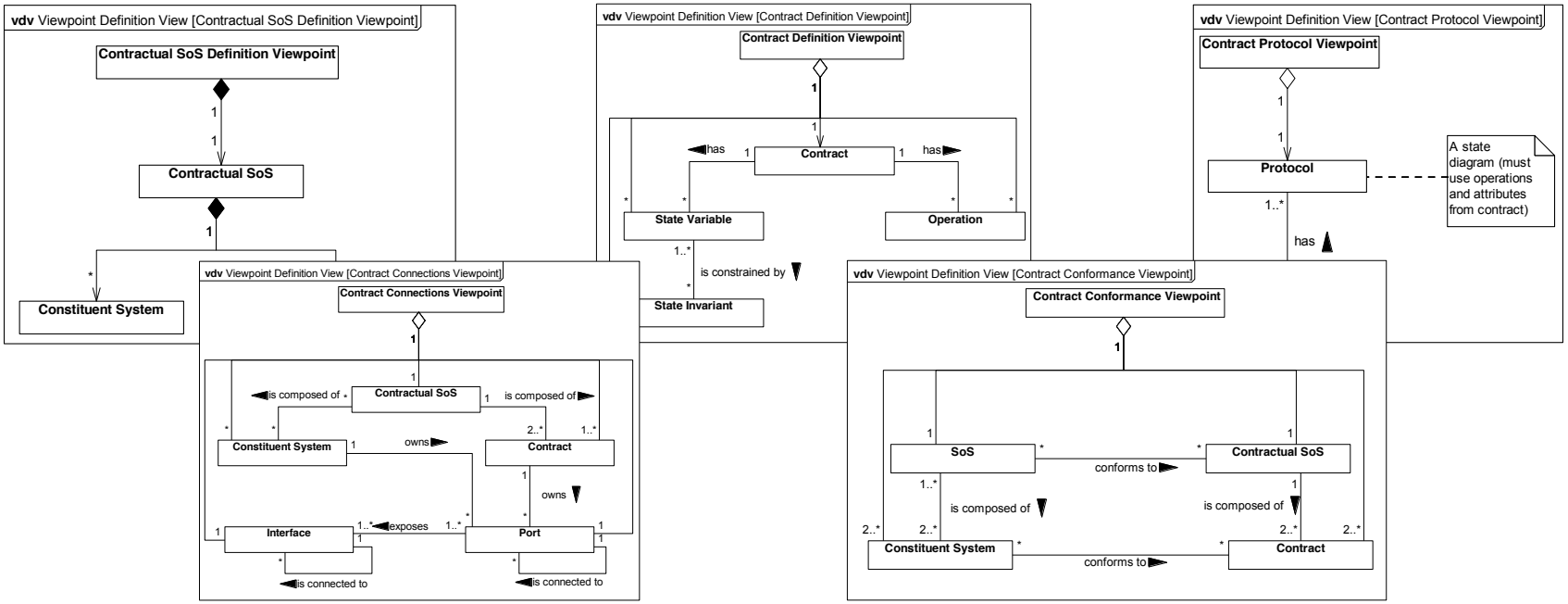
| Contract Pattern Viewpoints | |
|---|---|
| Name | Overview |
| Contractual SoS Definition | Identifies the contracts which comprise the Contractual SoS |
| Contract Conformance | Denotes the contracts each CS conforms to |
| Contract Connections | Shows connections and interfaces between contracts |
| Contract Definition | Defines operations, state variables and state invariants of a contract |
| Contract Protocol | Protocol specification of a contract |


# Contract Pattern – Viewpoint Definitions



- Define the model elements on a view and their relationships
- Consistent with ontology

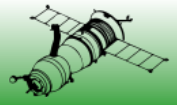


# Why Contracts May Help



- SoSs present significant challenges
  - CS integration: cannot justifiably rely on the behaviour of the CSs
  - Bound behaviours that can be relied upon *without over-constraining them*
  - *Promote desirable* and *limit undesirable* emergent behaviours
- *Contractual description* of CSs
  - CSs free to choose the way in which they meet these contracts
  - Free to adhere to other contracts

- We present a definition of a contract as a SysML pattern


# Proof of Concept Study

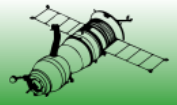


- Based on a Bang & Olufsen (B&O) home Audio Visual (AV) network linking multiple AV devices.
- The network exhibits the characteristic properties of a SoS;
  - Constituent systems are *heterogeneous* and may *evolve* (through software or firmware upgrades),
  - New CSs may be *integrated* into SoS at any time
  - CSs may be *legacy* or *non-B&O systems*, potentially limiting their controllability within the SoS.

# Proof of Concept Study

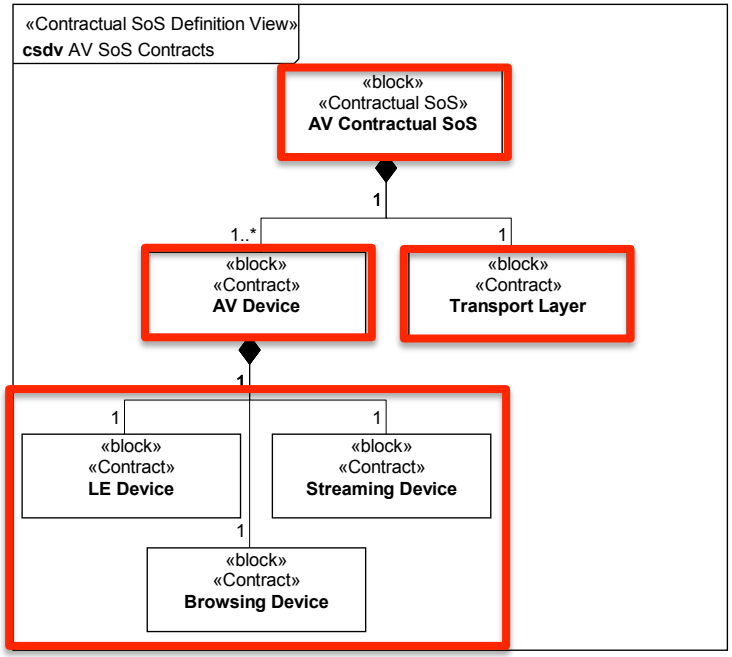


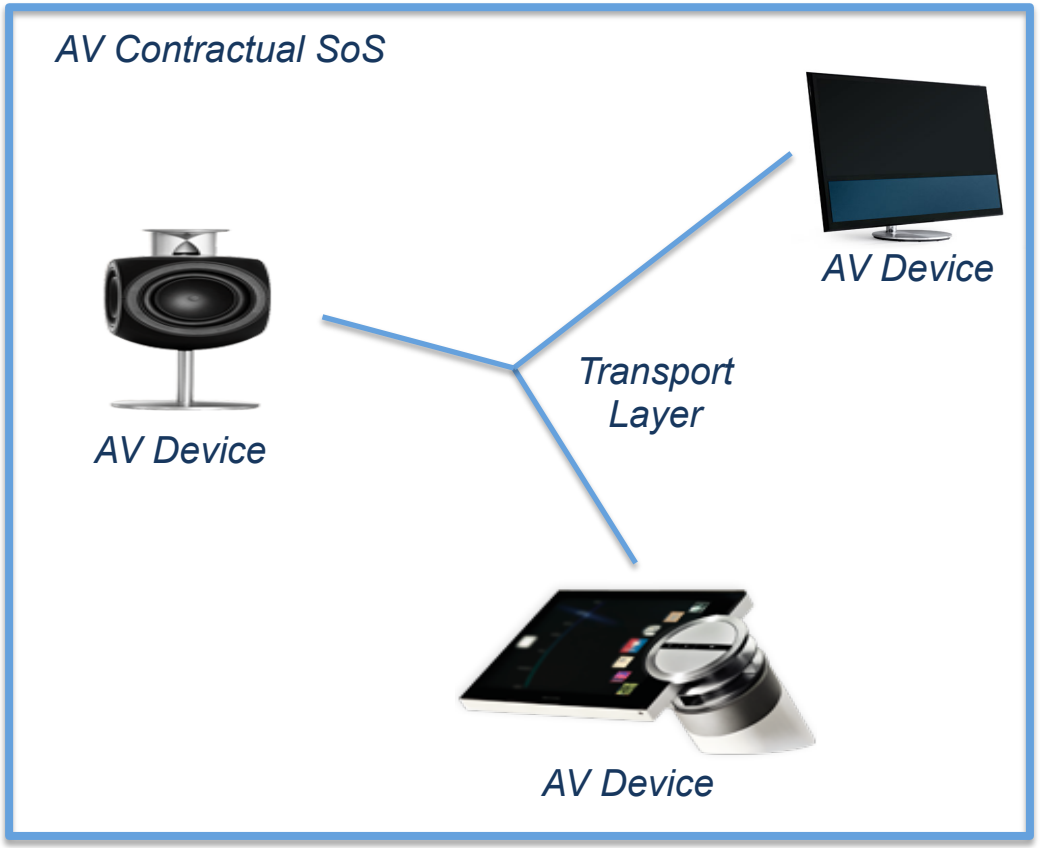
- To provide the B&O user experience, need a global '*leader*' to maintain global clock, SoS architecture, streaming details, …
- The ability to elect a leader may be considered an *emergent behaviour of SoS*
- Require that all AV devices in the SoS *conform to* several contracts
  - Use contract pattern to model the SoS, its CSs and the contracts of the SoS


# Contractual SoS Definition View

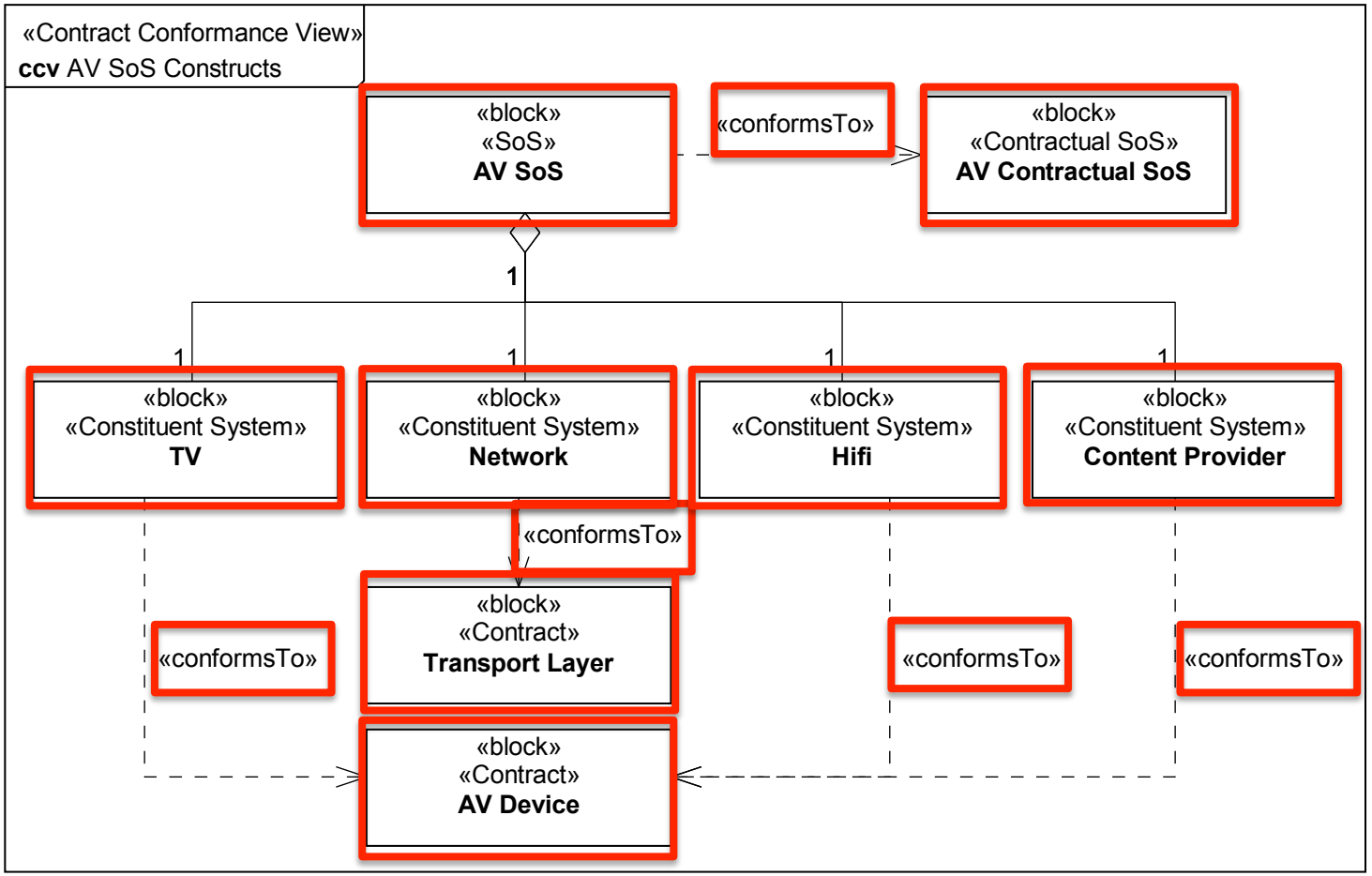




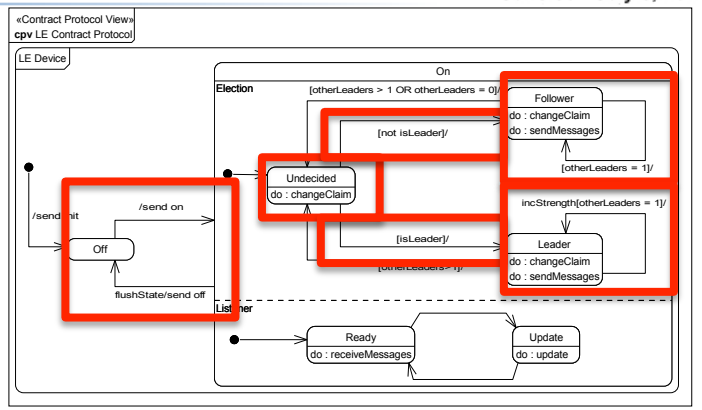
# Contract Conformance View





# LE Device



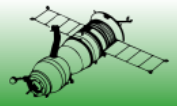



```
process LE_Device = i : nat @
begin
 …
  actions
  Off = on!id -> (Undecided /_\
          off!id -> flushState();Off)
  Undecided = changeClaim(<undecided>);Listener;
              ([isleader]& Leader
               []
               [ not isleader] & Follower)
Follower = …
Leader = …
```

# LE Device – CML model



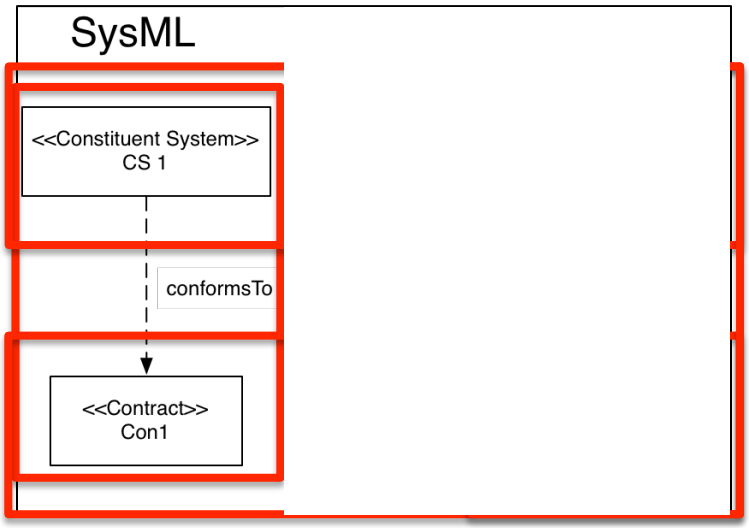
```
process LE_Device = i : nat @
begin
  state
    id : NODE_ID := i
    mem: map NODE_ID to CS :=
     {cid |-> mk_CS(<off>, 0) | cid in set node_ids \ {id}}
    inv dom mem = node_ids  \ {id} and dom mem <> {}

    highest_strength : STRENGTH := 0
    highest_strength_id : NODE_ID := 0
    inv highest_strength_id in set (dom mem union {id})

    leaders : nat := -1
    inv leaders <= card dom mem

    myCS : CS := mk_CS(<off>, 0)

    myNeighbours:seq of NODE_ID := [i|i in set dom mem@i <>id]

    isleader : bool := false

operations

    Init: () ==> ()
    Init() ==
    (
      id := i;
      flushState()
    )

    flushState: () ==> ()
    flushState() ==
    (
      mem := {cid|->mk_CS(<off>,0)|cid in set node_ids\{id}};
      highest_strength := 0;
      highest_strength_id := if id=0 then 1 else 0;
      leaders := -1;
      myCS := mk_CS(<off>, 0)
    )

    write: NODE_ID * DATA ==> ()
    write(n,dat) ==
    (
      if is_TL_MSG(dat) then mem(n) := mk_CS(<off>,0) else
          mem(n) := dat
    )
    pre i in set dom mem
    post mem(i) = dat or mem(i).c = <off>

    update:()==>()
    update() ==
    (
      leaders:= card{n|n in set dom mem @ mem(n).c = <leader>};
      highest_strength := maxStrength();
      highest_strength_id := maxStrengthID();
      isleader := amILeader()
    )
    post leaders>0=>mem(highest_strength_id).s=highest_strength

    maxStrength:() ==> nat
    maxStrength() ==
    (
      dcl strs : set of nat := {cs.s|cs in set rng mem @
          cs.c = <leader>} @  return maxSet(strs,0)
    )

       maxStrengthID : () ==> NODE_ID
    maxStrengthID() ==
    (
      dcl minId : NODE_ID, maxStrIds : set of NODE_ID @
        (
         if id = 0
         then minId := 1
         else minId := 0;
         maxStrIds := {n | n in set dom mem @ mem(n).s =
           highest_strength and mem(n).c = <leader>};
         return maxSet(maxStrIds,minId)
        )
    )

    changeClaim: CLAIM ==> ()
    changeClaim(newc) ==
    (
      dcl currStr : STRENGTH := myCS.s @
      myCS := mk_CS(newc, currStr)
      )
      pre myCS.c = <off> => newc = <undecided> and
          myCS.c = <undecided> => newc = <leader> or
          newc = <follower> and myCS.c = <leader> =>
           newc = <undecided> and
          myCS.c = <follower> => newc = <undecided>

      incStrength:()==>()
      incStrength() ==
      (
        if myCS.s < ulp
        then myCS := mk_CS(myCS.c, myCS.s+1)
      )
      pre myCS.s < ulp
      post myCS.s = myCS~.s + 1

    amILeader: () ==> bool
    amILeader() ==
    (
      return (leaders = 0) or highest_strength < myCS.s
    )

    amILeader2: () ==> bool
    amILeader2() ==
    (
      return (leaders = 0)  or highest_strength < myCS.s

    )

    whoIsHighest: () ==> NODE_ID
    whoIsHighest()==
    (
     return highest_strength_id
    )

actions

    Off = on!id -> (Undecided /_\ off!id -> flushState();Off)

    Listener = ReceiveData;update();OutputData

    ReceiveData = (n_rec!id?s?dat ->write(s,dat);ReceiveData)
                        [_ n_timeout _> Skip

    OutputData = leaderClaim!id!isleader -> Skip

    SendCS =  (||| t in set dom mem @ [{}] n_send!id!t!myCS ->
              Skip)

    Undecided = changeClaim(<undecided>);Listener;
                ([isleader]& Leader
                 []
                 [ not isleader] & Follower)

    Leader = changeClaim(<leader>);SendCS;Listener;
             ([leaders > 0] & Undecided
              []
              [leaders = 0] & incStrength();Leader)

    Follower = changeClaim(<follower>)/*;SendCS*/;flushState();
               Listener; ([leaders >1 or leaders=0 ] & Undecided
                          []
                          [leaders = 1] &)

  @ init -> Init();(Off/_\deInit->Skip)

end
```


# Verifying Contract Conformance



- *Contract Conformance Viewpoint*: **informal**
- How to verify this conformance?
- SysML may be translated to the formal notation CML
- CML refinement could be used as means of checking conformance.



# Verifying Contract Conformance



- Results may be reported to the engineer, and recorded at the SysML level.
  - Success: included on a *Contract Conformance Viewpoint* as an *evidence* model element
  - Failure: a trace of the CS not permitted by the contract (or vice versa) translated into a SysML sequence diagram