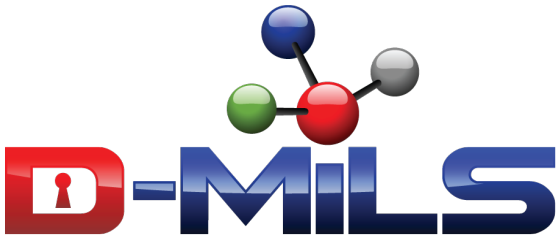


D-MILS and MILS-AADL

Thomas Noll

RWTH AACHEN UNIVERSITY

AADL Standards Meeting
IRIT University, Toulouse, France; January 27, 2016

Outline

The Starting Point: MILS

MILS

A component-based approach for the construction, assurance, and certification of dependable systems that encourages a commercial marketplace of off-the-shelf high-assurance components

The Starting Point: MILS

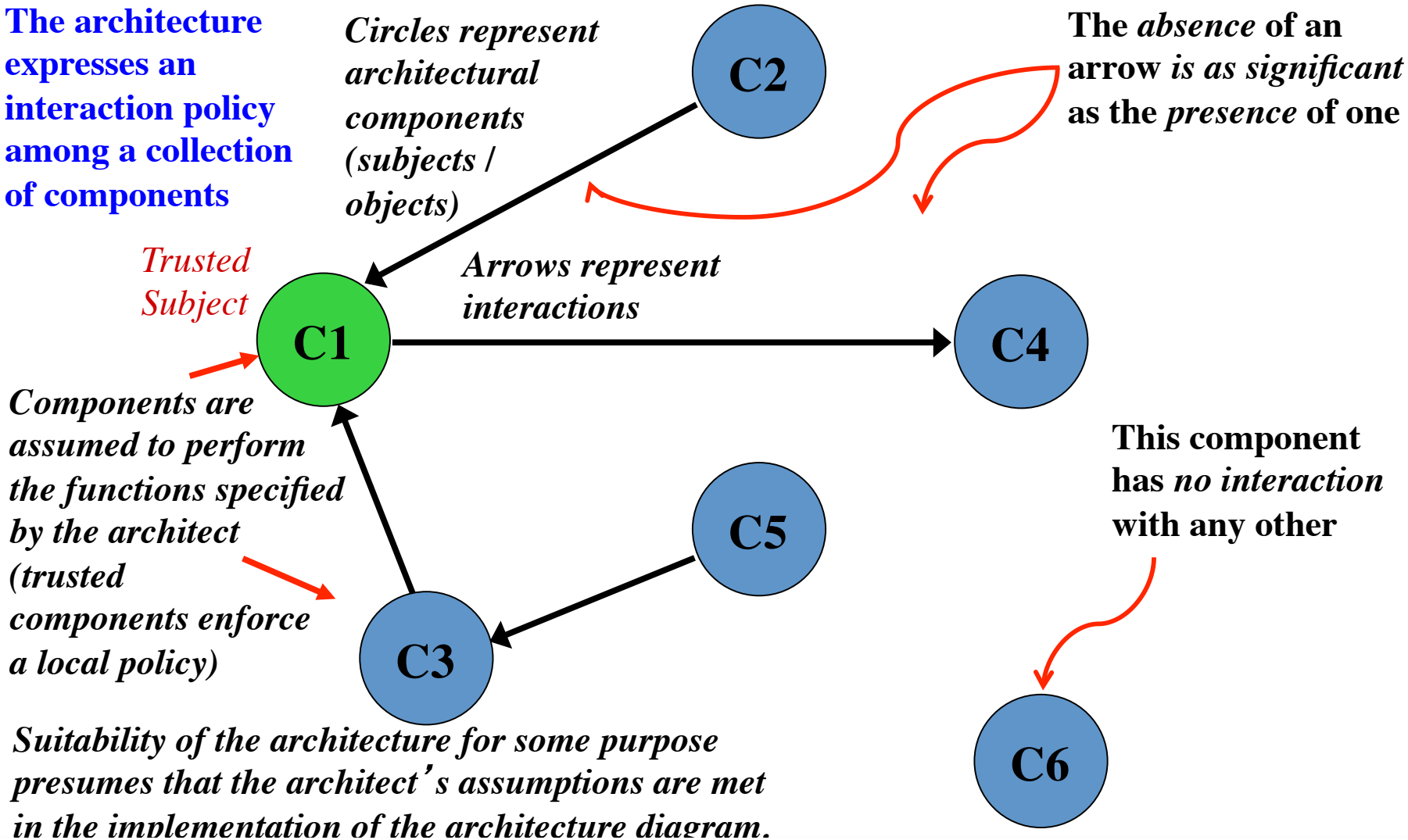
MILS

A component-based approach for the construction, assurance, and certification of dependable systems that encourages a commercial marketplace of off-the-shelf high-assurance components

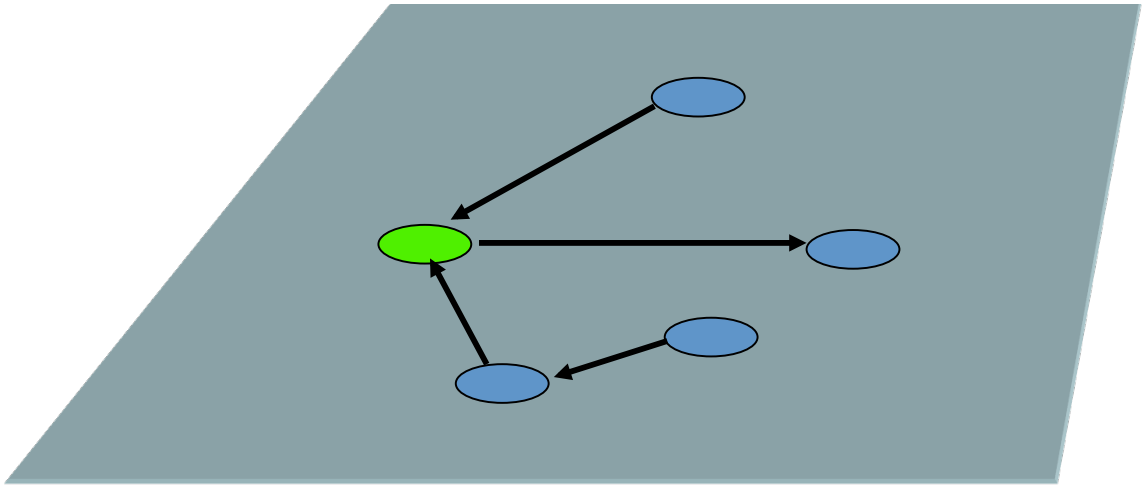
Two-phase approach

1. Architecture
   - Abstract policy architecture ("boxes", "arrows")
   - System dynamics by component behaviour and interactions
   - Assumption: architecture will be strictly enforced
2. Implementation
   - Robust resource-sharing platform
   - Composed of MILS foundational components
   - Creates strongly isolated "exported resources"

MILS Policy Architectures

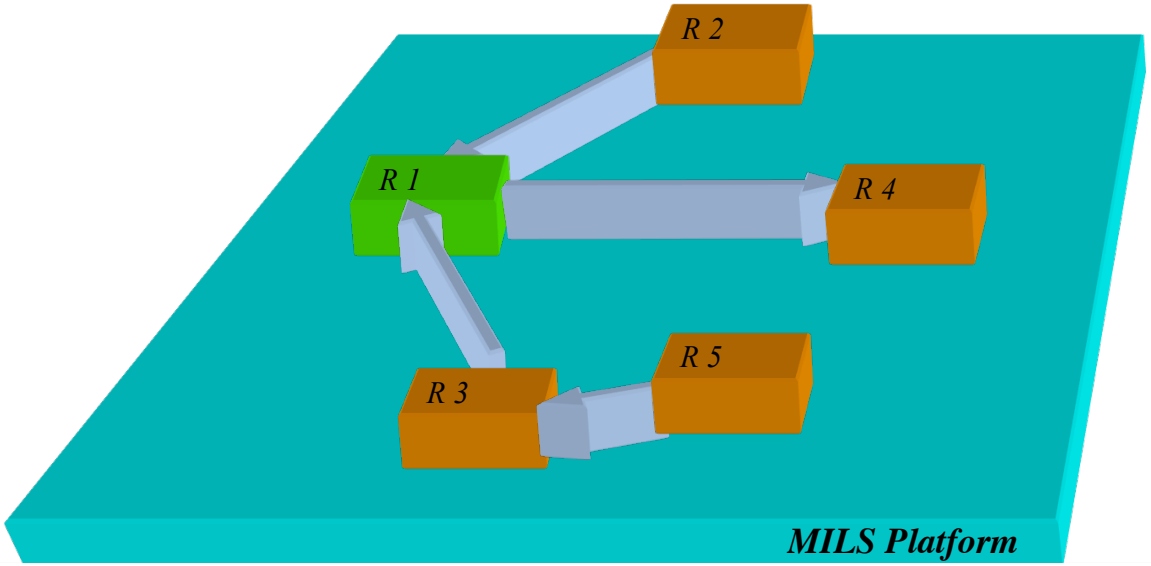


MILS Platform



Architecture

Validity of the architecture assumes that the *only* interactions of the circles (operational components) is through the arrows depicted in the diagram



Implementation

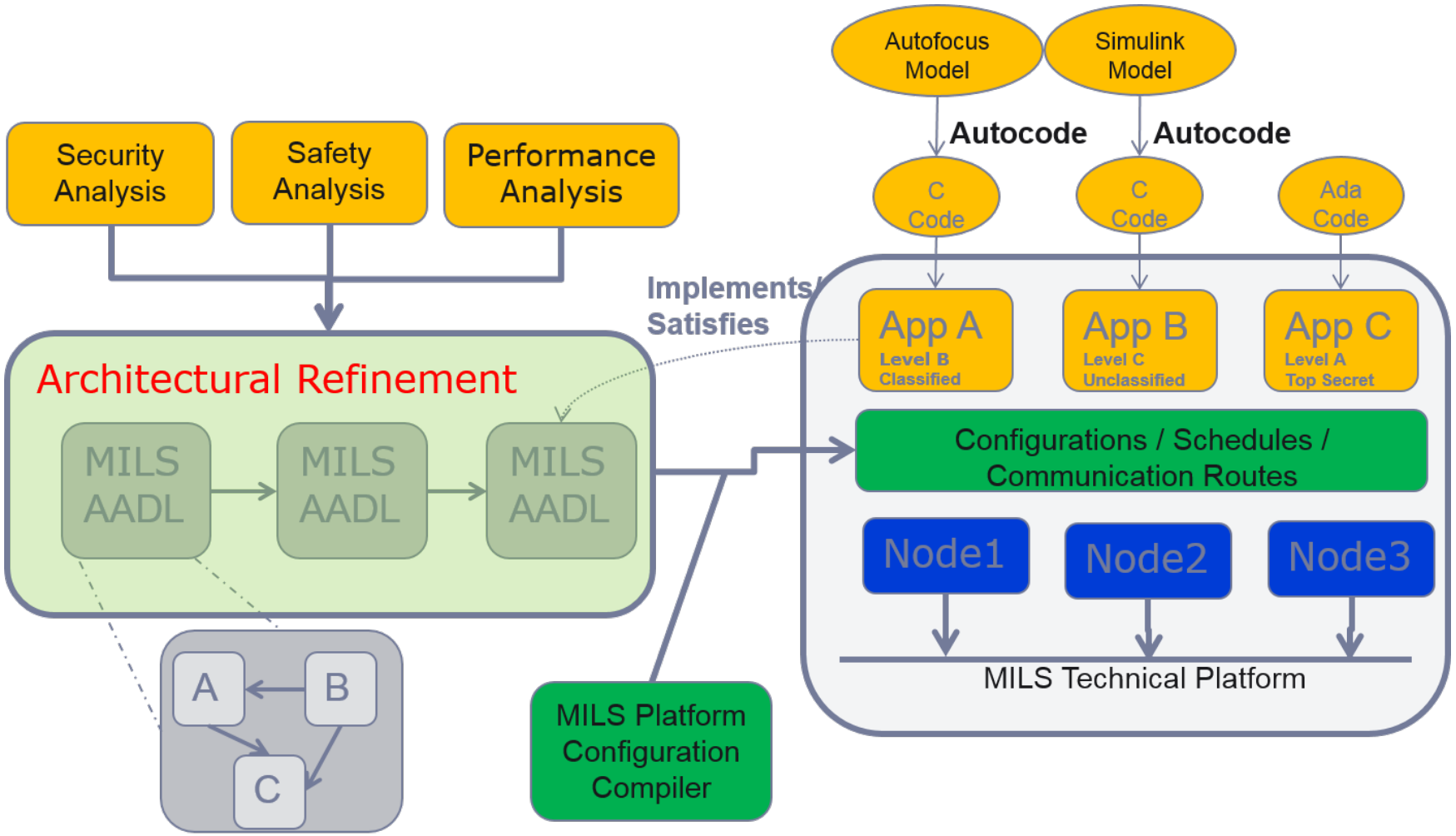
SK, with other MILS foundational components, form the *MILS Platform* allowing operational components to share physical resources while enforcing Isolation and Information Flow Control

Outline

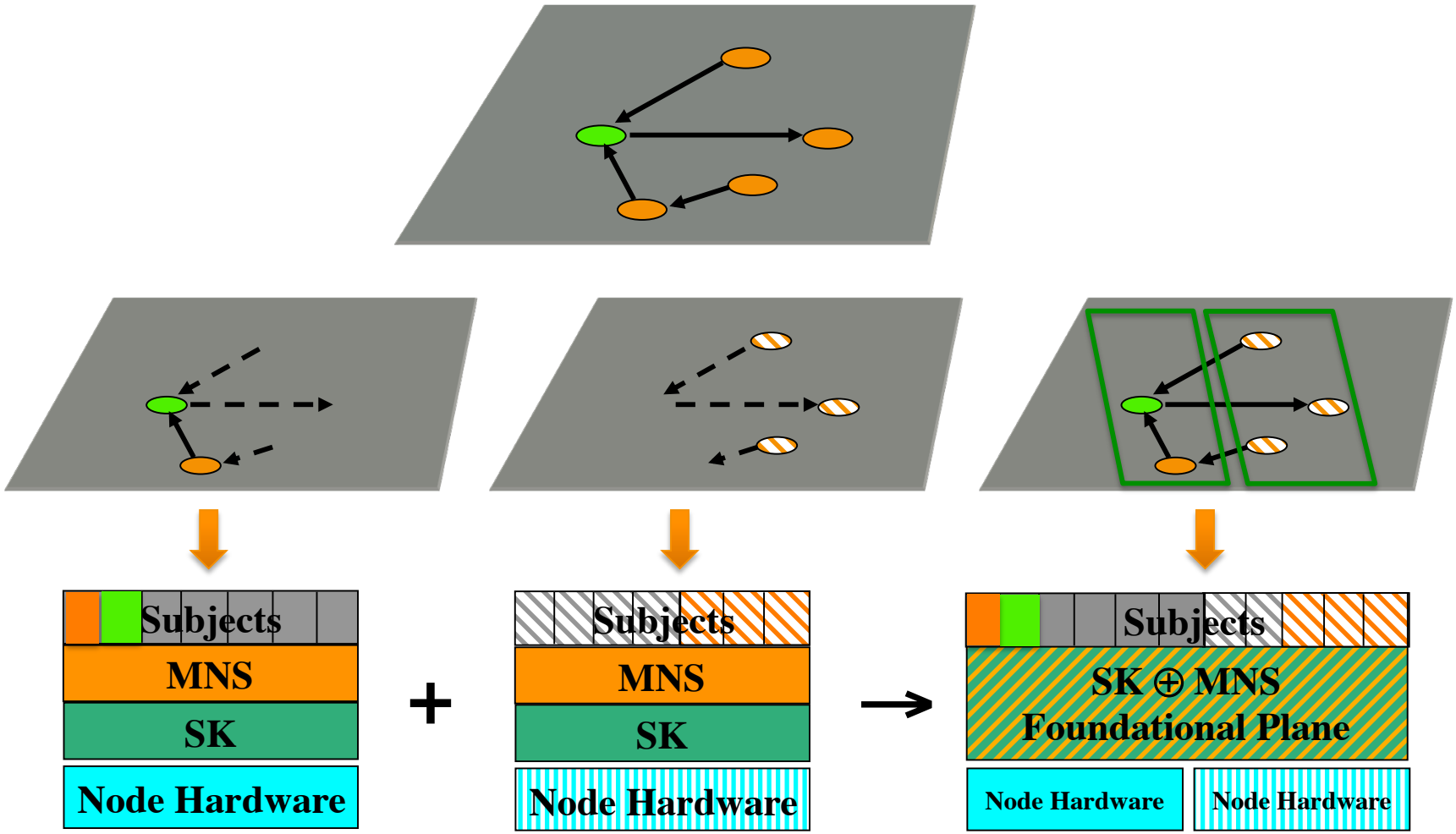
Distributed MILS (D-MILS) Overview

- A single policy architecture may span multiple D-MILS nodes
- Guarantees similar to a single MILS node: isolation, information flow control, determinism
- Determinism over network could be achieved in various ways - here we use Time-Triggered Ethernet (TTE)
- Must configure and schedule the network and the processors of the nodes coherently
- (Semi-)automated support for
  - verification of architectural properties
  - presentation of assurance case
  - generation of configuration

D-MILS Design Flow



D-MILS Platform



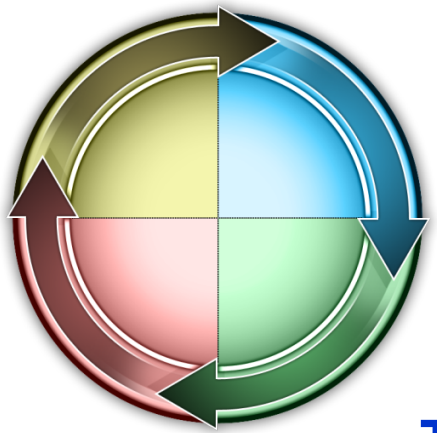
D-MILS Consortium

Use Cases, requirements validation

- fortiss (DE)
- Frequentis (AT)

Research/development

- FBK (IT)
- RWTH Aachen (DE)
- Univ J. Fourier Grenoble (FR)
- Univ of York (UK)
- INRIA (FR)



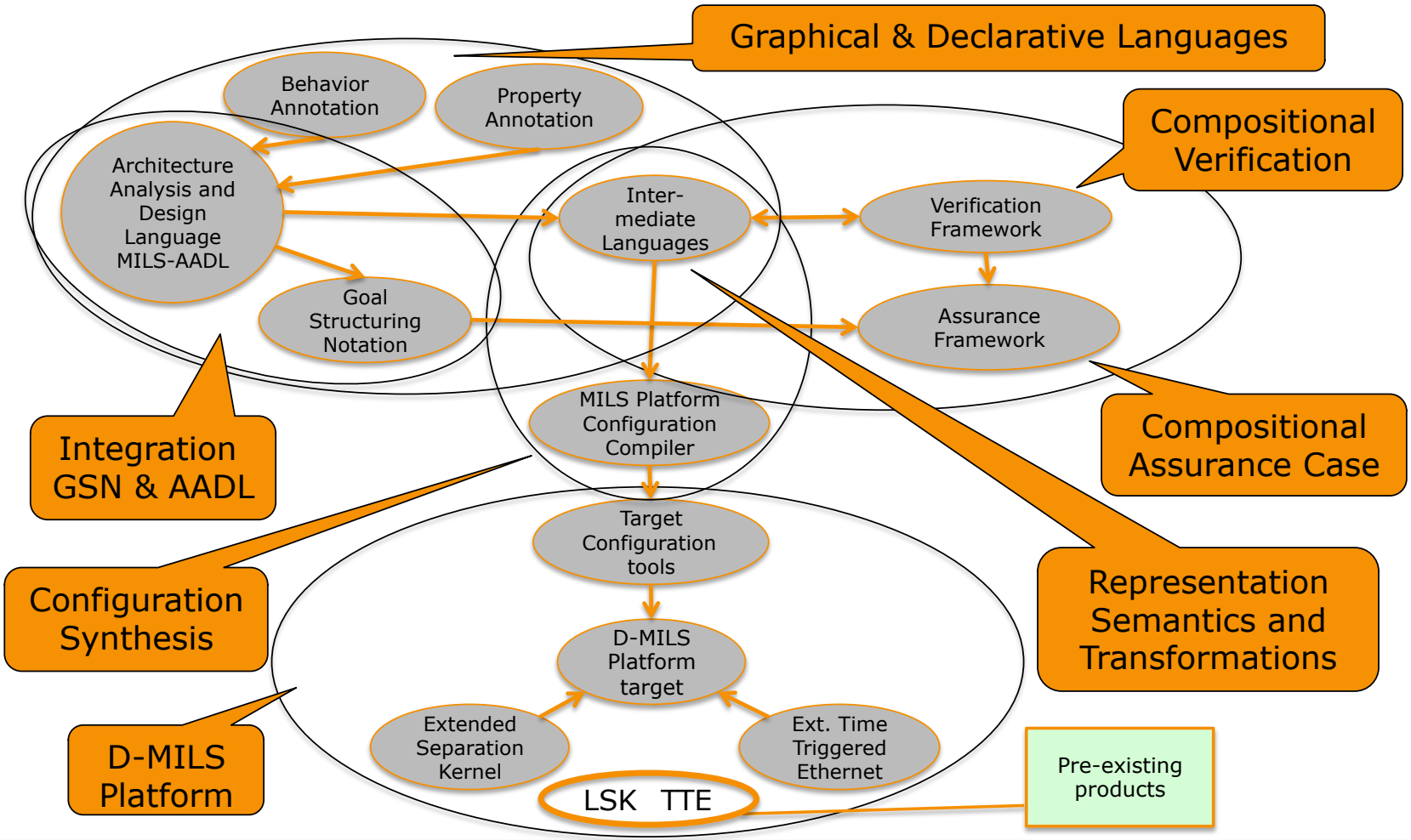
Dissemination and industry technologies

- Open Group (UK)

Technology supplier

- TTTech (AT)
- Lynx Secure Tech (FR)

D-MILS R&D Areas



Outline

Key Language Ingredients I

- **Approach:** define "extended subset" of Architecture Analysis and Design Language (AADL) and Error Model Annex (EMA)

Key Language Ingredients I

- **Approach:** define "extended subset" of Architecture Analysis and Design Language (AADL) and Error Model Annex (EMA)
- From AADL:
  - System = hierarchy of (hardware and software) components
  - Component defined by type (interface) and implementation
  - Component type features: ports
    - event ports: instantaneous communication (handshaking with message passing)
    - data ports: continuous communication
  - Component implementation:
    - internal structure given by subcomponents and SW/HW bindings
    - interaction via connections over ports
    - behaviour described by mode transition system and flows (mapping input to output data ports)
  - Activation of subcomponents and connections possibly mode dependent (to support modelling of redundancy)

Key Language Ingredients II

- From EMA:
  - Error model defined by type and implementation
  - Type = (visible) error propagations
  - Implementation = (invisible) error states, events, and transitions

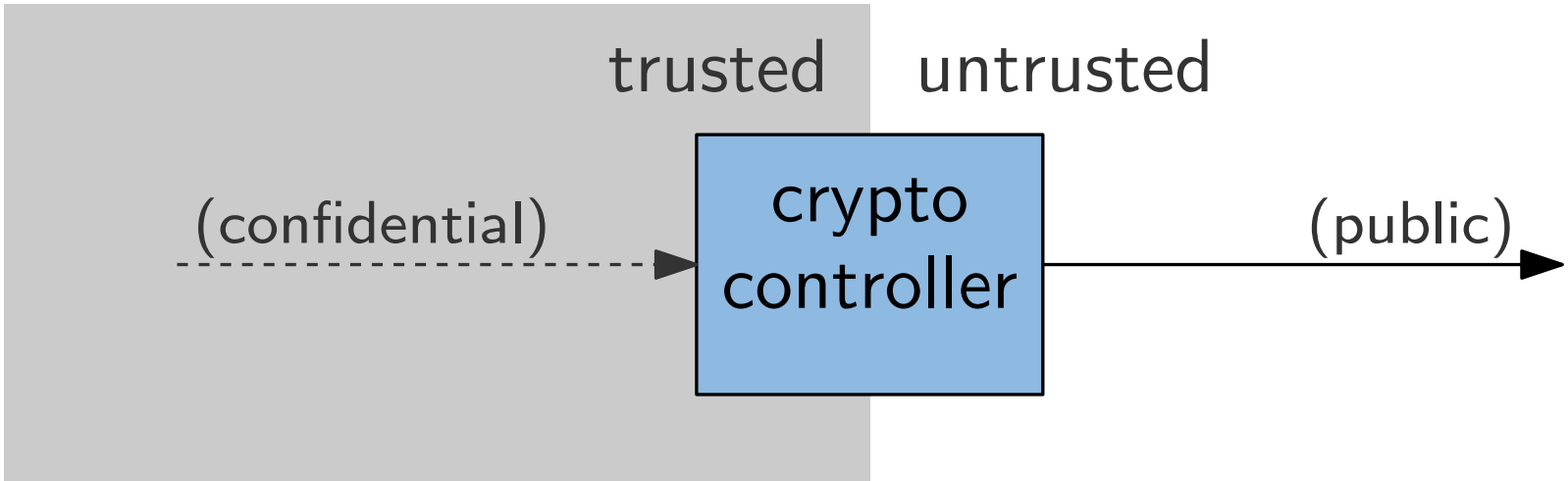
Key Language Ingredients II

- From EMA:
  - Error model defined by type and implementation
  - Type = (visible) error propagations
  - Implementation = (invisible) error states, events, and transitions
- From COMPASS extensions:
  - Hybrid modelling with linear value trajectories

D-MILS Extensions

- Extended type system (tuples, keys, ...)
- Symbolic names for type and value constants
- Time units
- Operations for asymmetric and symmetric encryption and authentication
- Uninterpreted functions
- Event data ports
- Contract annotations
- Configuration annotations

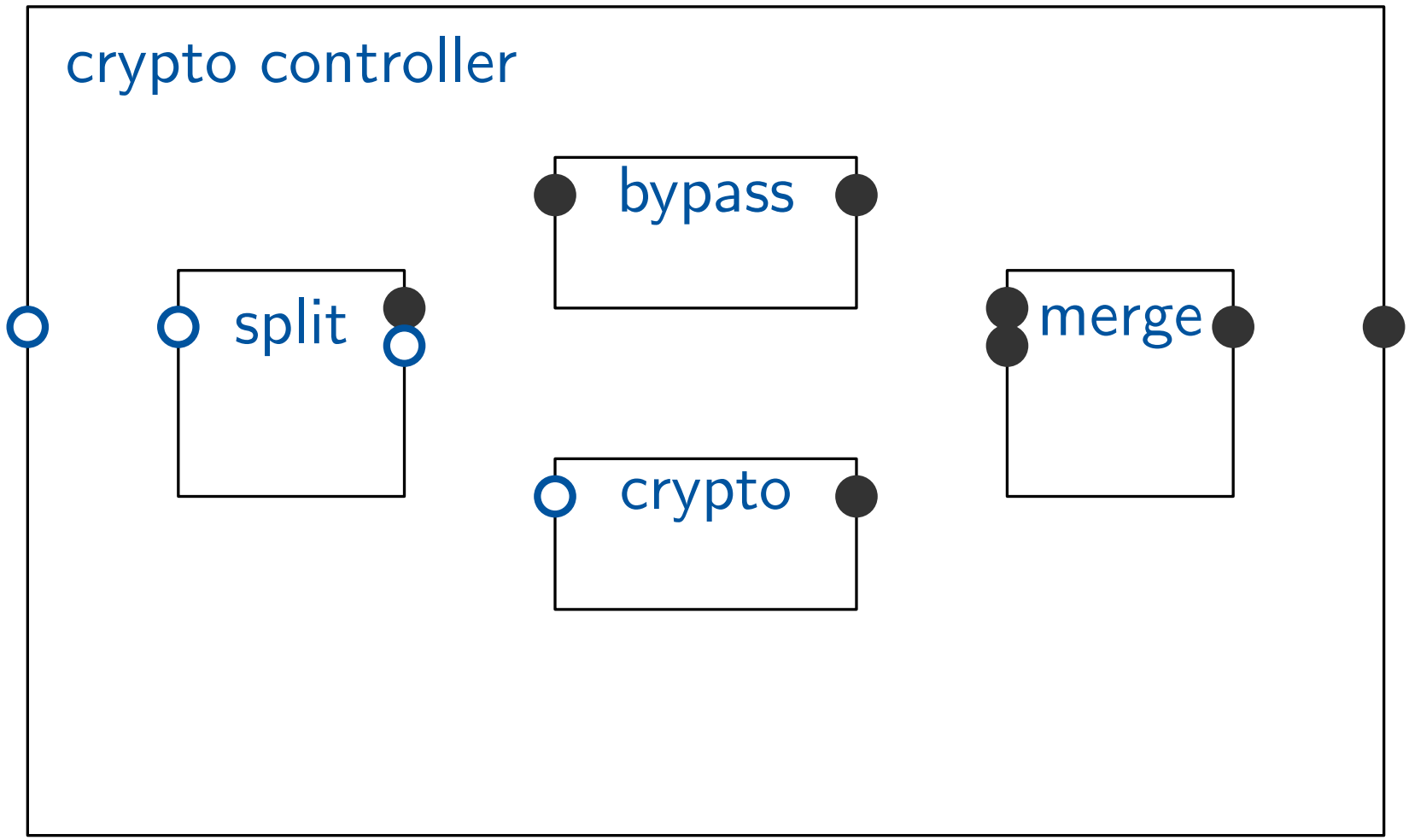
Example: Cryptographic Controller I



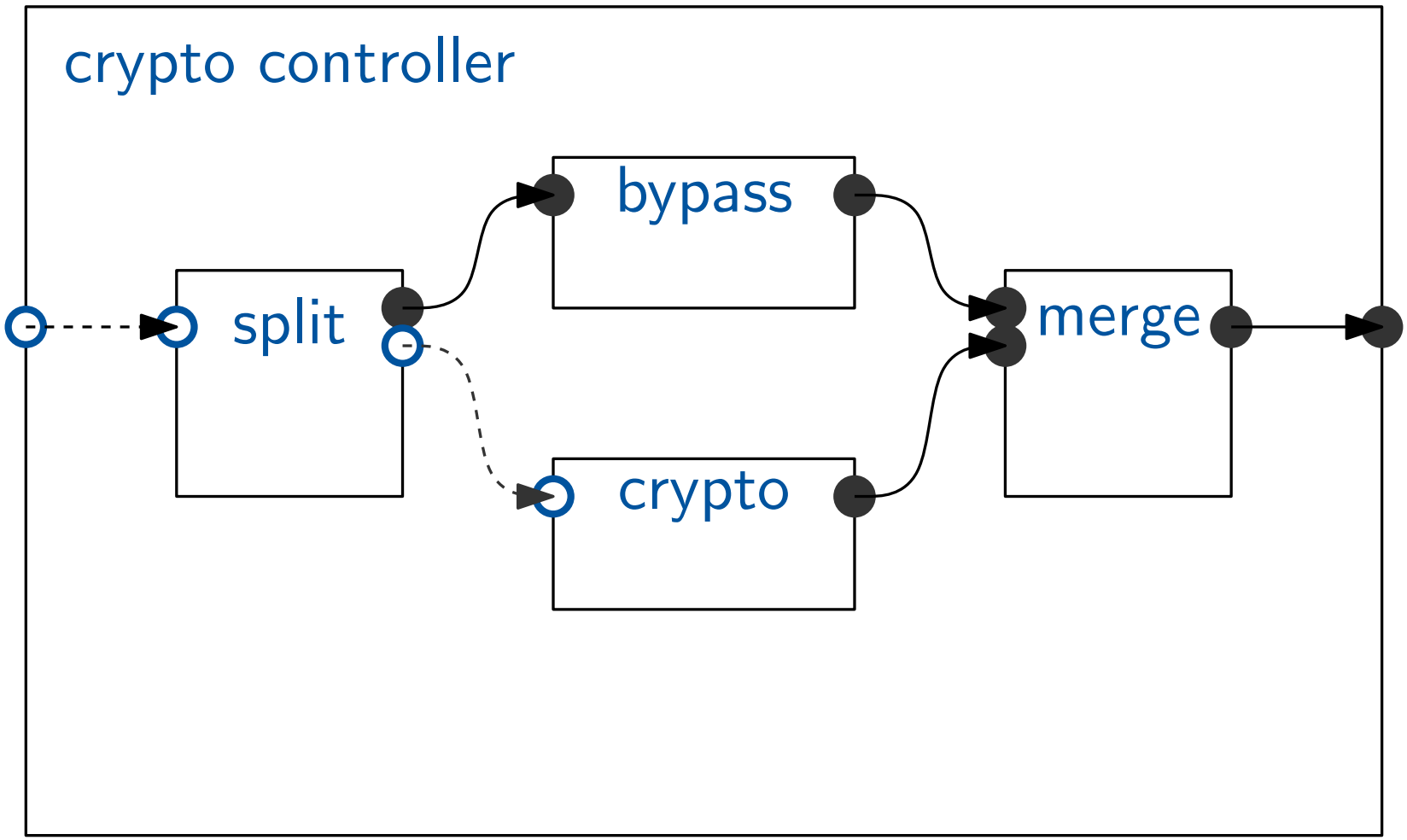
- Cryptographic controller
- Placed between trusted system and untrusted network
- Declassifies confidential information

Example: Cryptographic Controller II

Example: Cryptographic Controller II



Example: Cryptographic Controller II



Example: Cryptographic Controller III

```
constants
  Frame: type := (Header, Payload);
  Header: type := int;
  Payload: type := int;
  MyKeys: asymmetric key pair;

process Crypto
  features
    inpayload: in data port Payload;
    outpayload: out data port Payload;
end Crypto;

process implementation Crypto.Imp
  constants
    mykey: public key default pub(MyKeys);
  flows
    port encrypt(inpayload, mykey) -> outpayload;
end Crypto.Imp;
```

Sketch of the Semantics

- $[\![f]\!] : D_{t_1} \times \ldots \times D_{t_n} \to D_{t_0}$ for each function constant f: **function** $t_1, \ldots, t_n$ -> t_0 ($n \geq 1$, $t_i \in Type$)
- $[\![\mathbf{encrypt}]\!] : D_t \times D_{\mathbf{key}} \to D_{\mathbf{enc}(t)}$ and $[\![\mathbf{decrypt}]\!] : D_{\mathbf{enc}(t)} \times D_{\mathbf{key}} \to D_t$ such that

$$[\![\mathbf{decrypt}]\!]([\![\mathbf{encrypt}]\!](d, k_1), k_2) = d$$

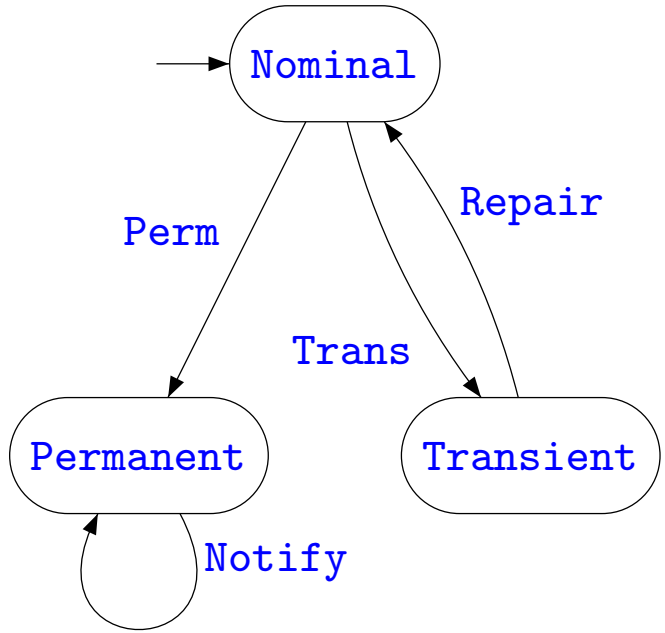
  if there exists a key $k \in D_{\mathbf{key}}$ such that $k_1 = \mathbf{pub}(k)$ and $k_2 = \mathbf{priv}(k)$
- $[\![\mathbf{sign}]\!] : D_t \times D_{\mathbf{key}} \to D_{\mathbf{sgn}(t)}$ and $[\![\mathbf{verify}]\!] : D_{\mathbf{sgn}(t)} \times D_{\mathbf{key}} \to D_t$ such that

$$[\![\mathbf{verify}]\!]([\![\mathbf{sign}]\!](d, k_1), k_2) = d$$

  if there exists a key $k \in D_{\mathbf{key}}$ such that $k_1 = \mathbf{priv}(k)$ and $k_2 = \mathbf{pub}(k)$,

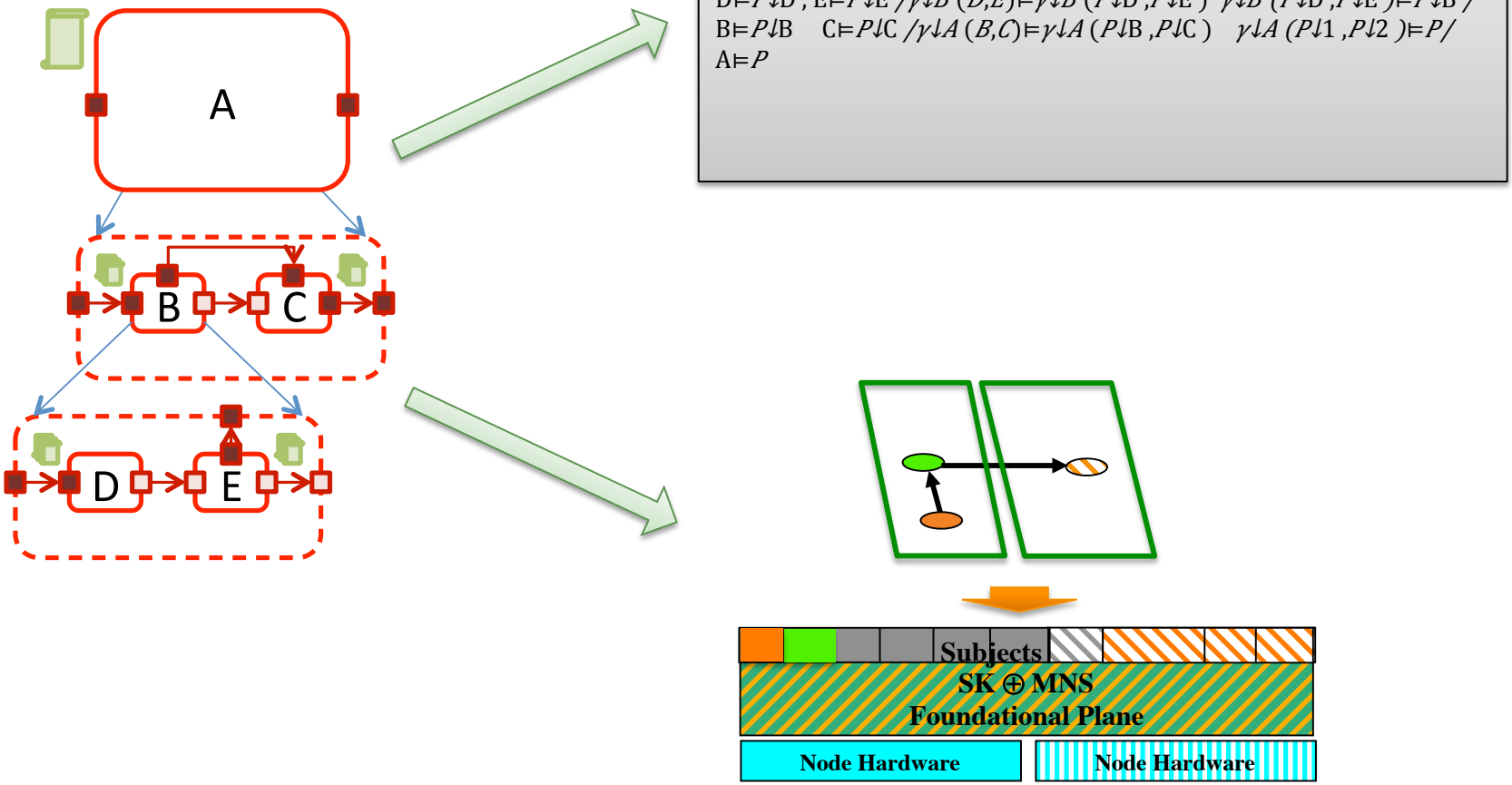
Error Modelling

```
error model TransPerm
  features
    Notify: out error propagation;
end TransPerm;
error model implementation TransPerm.Impl
  states
    Nominal: initial state;
    Permanent, Transient: state;
  events
    Perm: event rate 0.01 per hour;
    Trans: event rate 0.1 per hour;
    Repair: event rate 0.2 per min;
  transitions
    Nominal -[Perm]-> Permanent;
    Nominal -[Trans]-> Transient;
    Permanent -[Notify]-> Permanent;
    Transient -[Repair]-> Nominal;
end TransPerm.Impl;
```

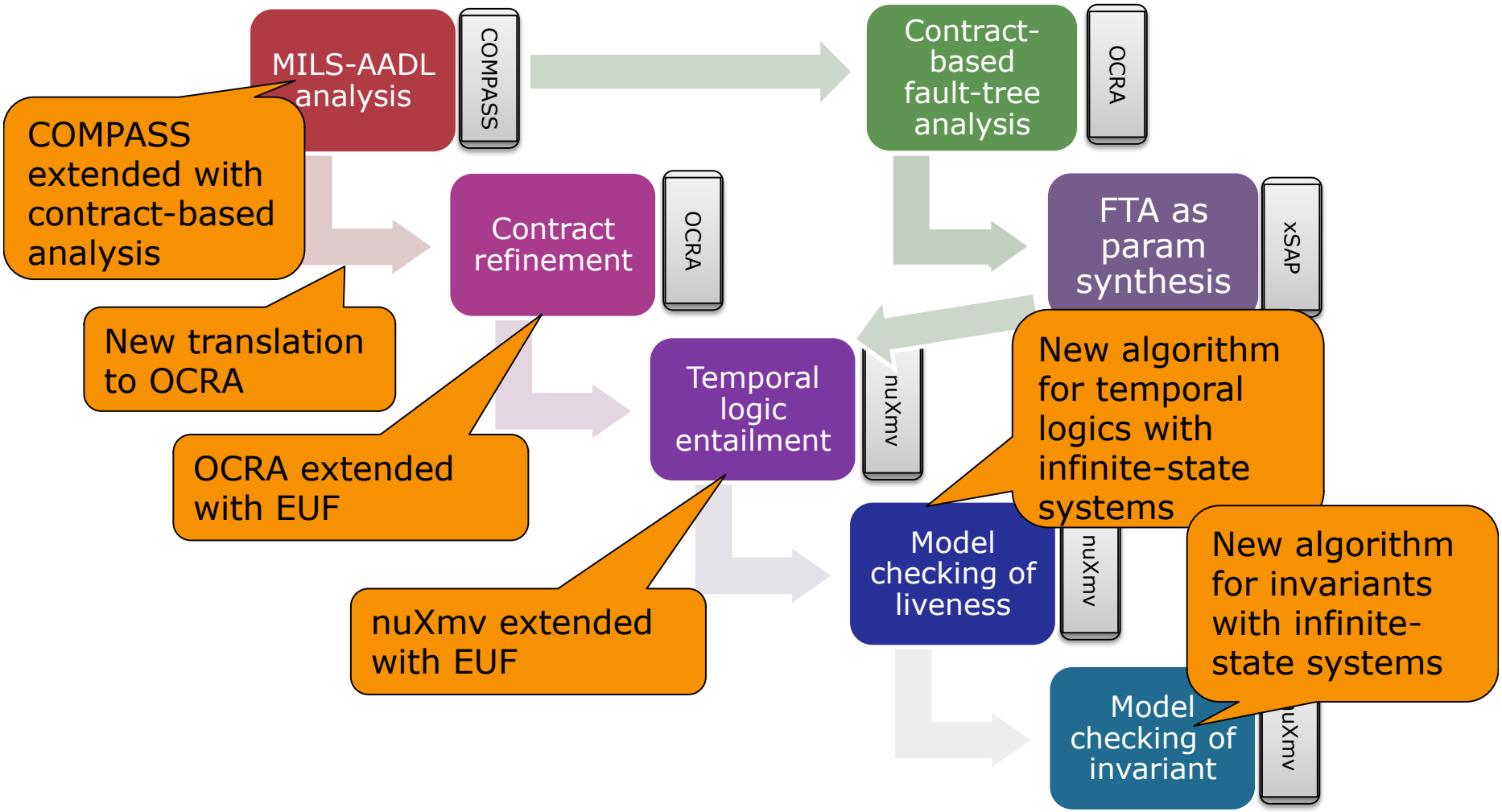


Outline

Contract-Based Design



Compositional Verification Tool Chain



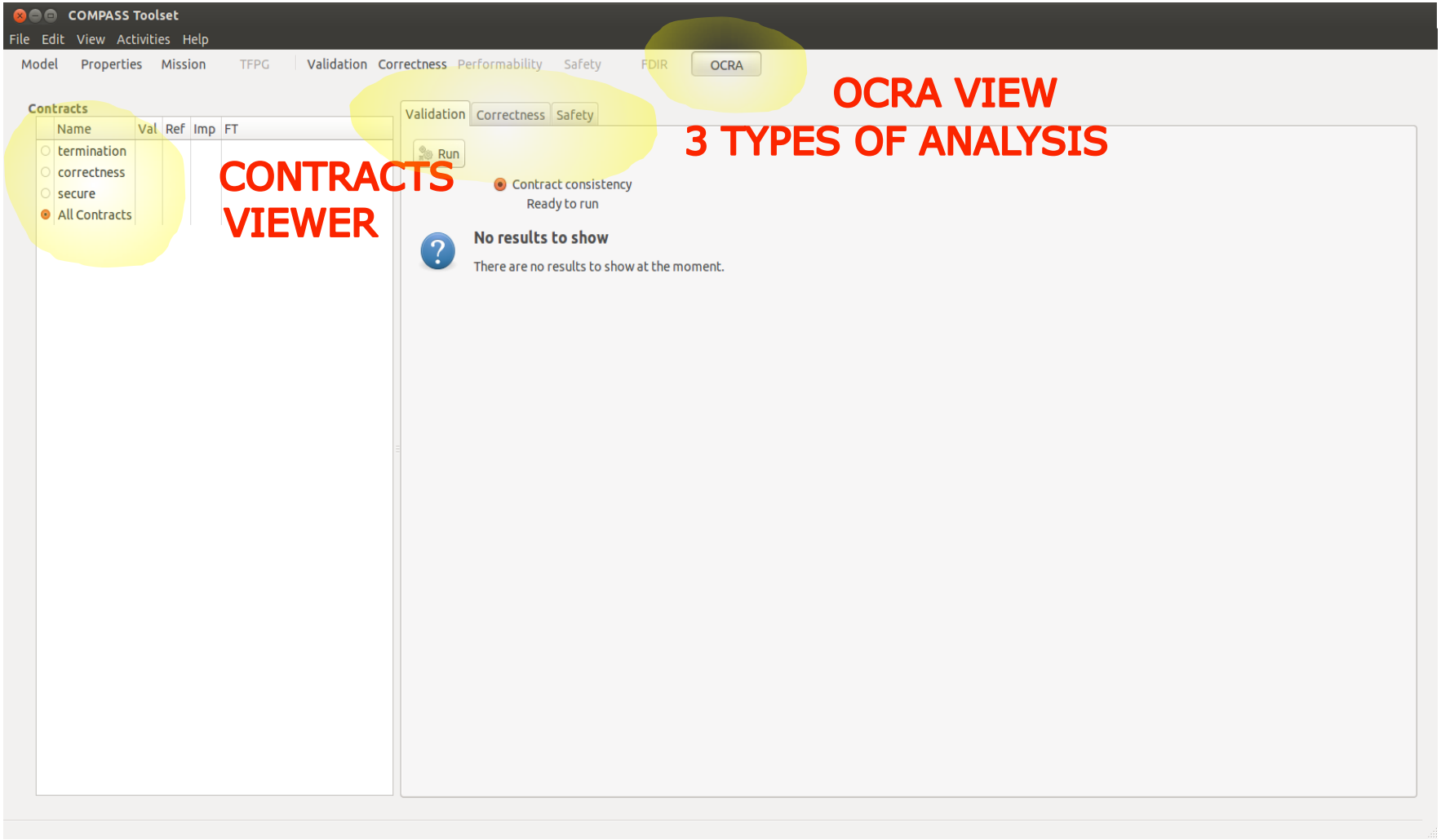
Contract Annotations

```
system Sys
  features
    cmd: in event data port int;
    switch_to_high: in event port;
    switch_to_low: in event port;
    return: out event data port int;
    outL: out data port int;
    { OCRA: CONTRACT secure
        assume: always (
            ({cmd} implies then ({return} releases (not ({cmd or switch_to_high or switch_to_low}))))
            and (((not {switch_to_high}) since {switch_to_low}) implies (not {is_high(last_data(cmd))}))
            and ({is_high(0)} = false) );
        guarantee: always ( ({is_high(outL)}=false));
    }
```

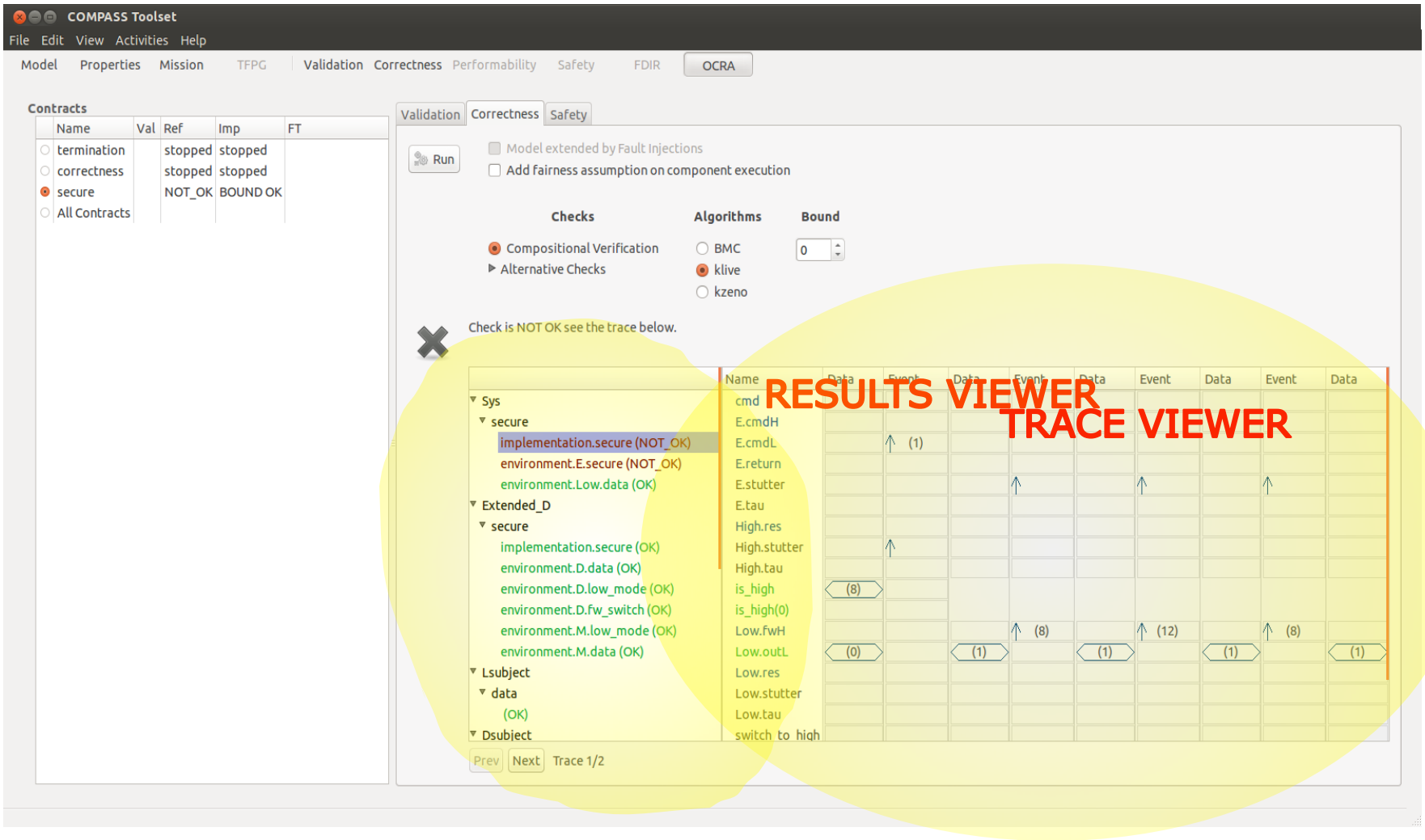
Property Specification Language

- LTL
  - always (p implies in the future q)
- First-order
  - always (high(value) iff high(cmd)) implies never (high(output))
- Real-time
  - always (corrupted(memory) implies time_until(alarm)<=time_bound)

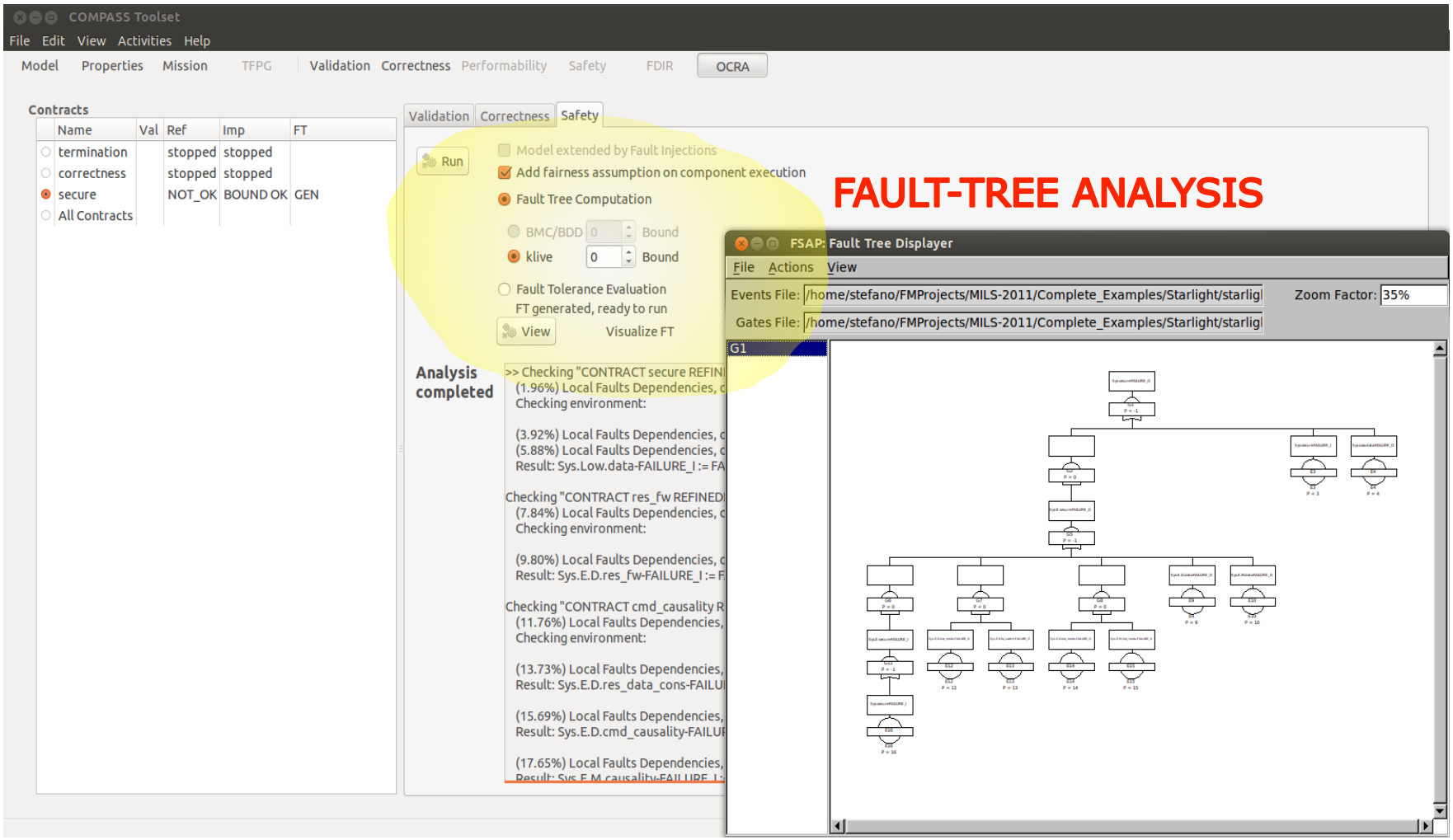
Implementation in COMPASS Toolset I



Implementation in COMPASS Toolset II



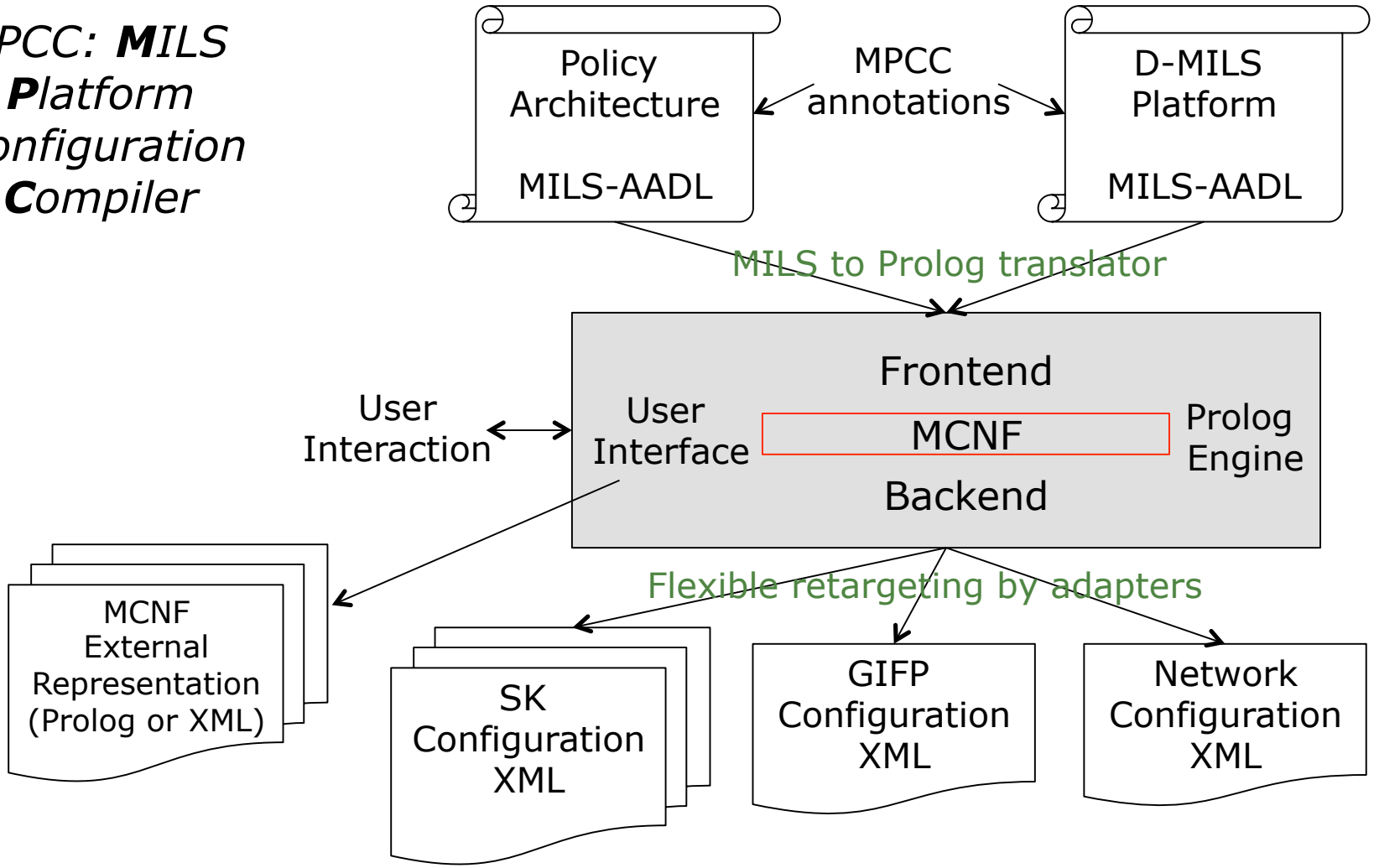
Implementation in COMPASS Toolset III



Outline

The MILS Platform Configuration Compiler

*MPCC: **M**ILS **P**latform **C**onfiguration **C**ompiler*

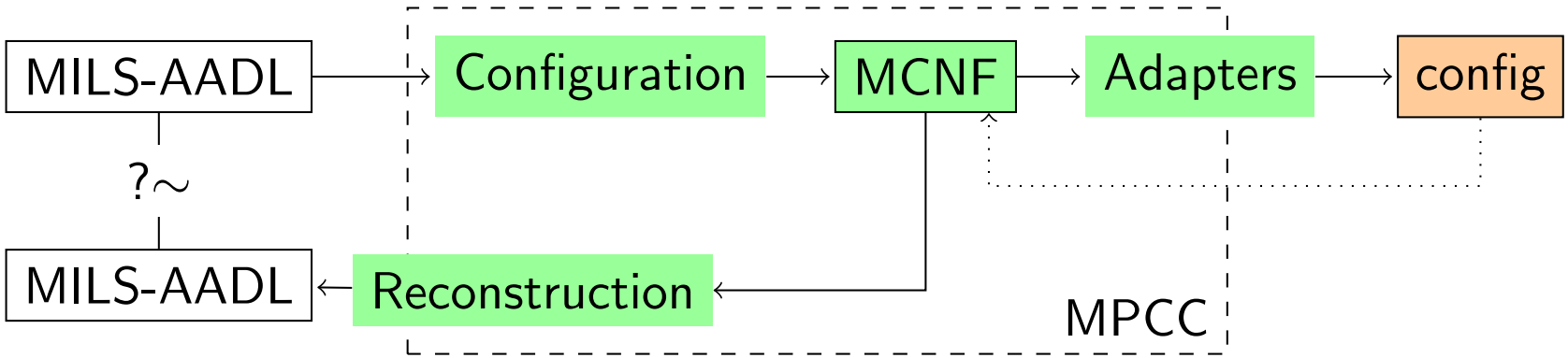


Configuration Annotations

```
system implementation main.i
  {MPCC: phylink( pr(tt1,es,p1), pr(sw0,p1),
                  [length('2m'),media(copper)] )}
  {MPCC: phylink( pr(tt2,es,p1), pr(sw0,p2),
                  [length('2m'),media(copper)] )}
  {MPCC: deployment(not_same([low,user])) }
  {MPCC: deployment(same([dispatch,user])) }

  subcomponents
    sw0: bus switch.i;
    tt1: node dell.i accesses sw0;
    tt2: node dell.i accesses sw0;
    high : subject Hsubject;
    low :  subject Lsubject;
    dispatch:  subject Dsubject;
    user:  subject Usubject;
 connections
    port user.cmd -> dispatch.cmd ;
    [...]
end main.i;
```

Configuration Verification

Construct a fully mapped model from the MILS Configuration Normal Form



Reconstructed model structured according to the deployment

Configuration Correctness

Sanity check

Comparison of original and reconstructed MILS-AADL model

Configuration Correctness

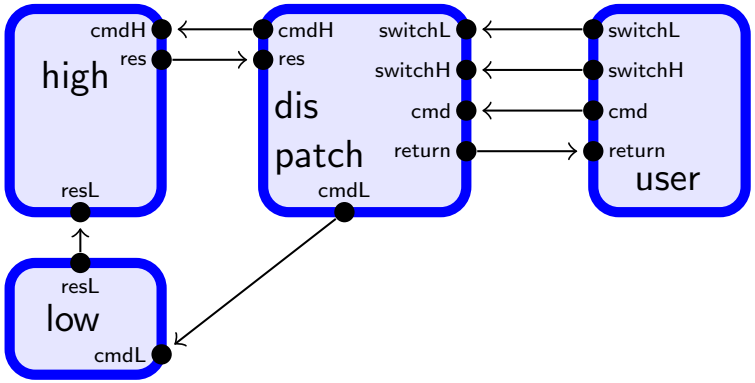
Sanity check

Comparison of original and reconstructed MILS-AADL model

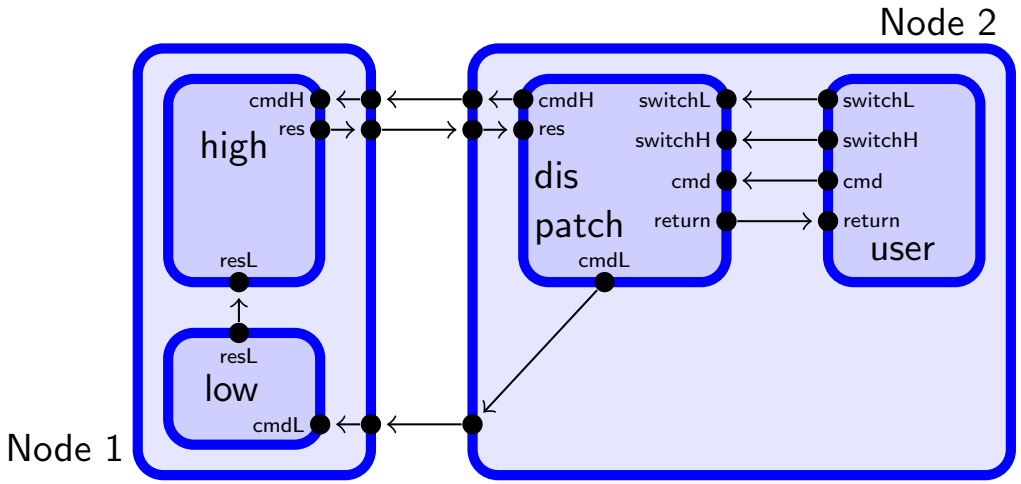
Equivalence requirements

- policy architecture: subjects and connections between these are "isomorphic" in both models
- resources: set of resources assigned to each subject in the reconstructed model is equivalent to set of resources requested in original model
- mapping constraints are satisfied

Example: Policy Architecture

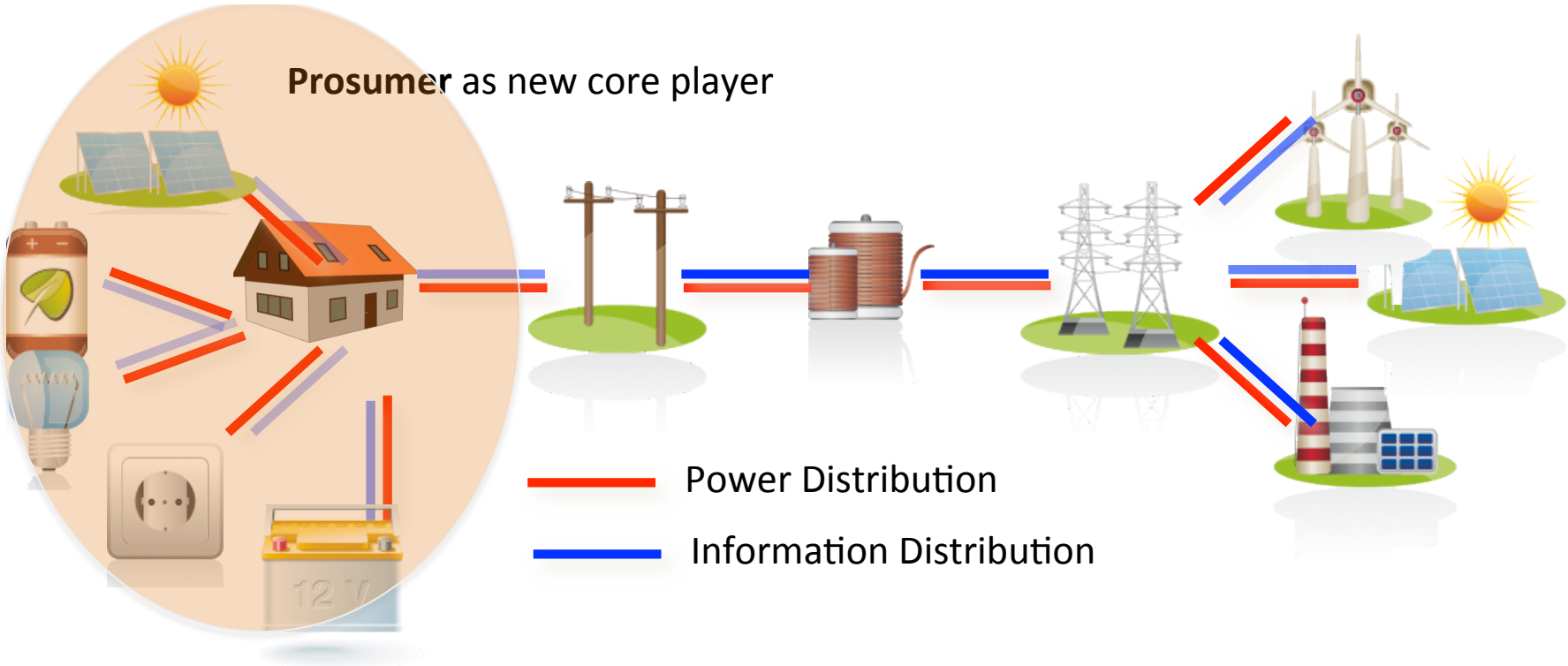


is equivalent to



Outline

fortiss Smart Microgrid



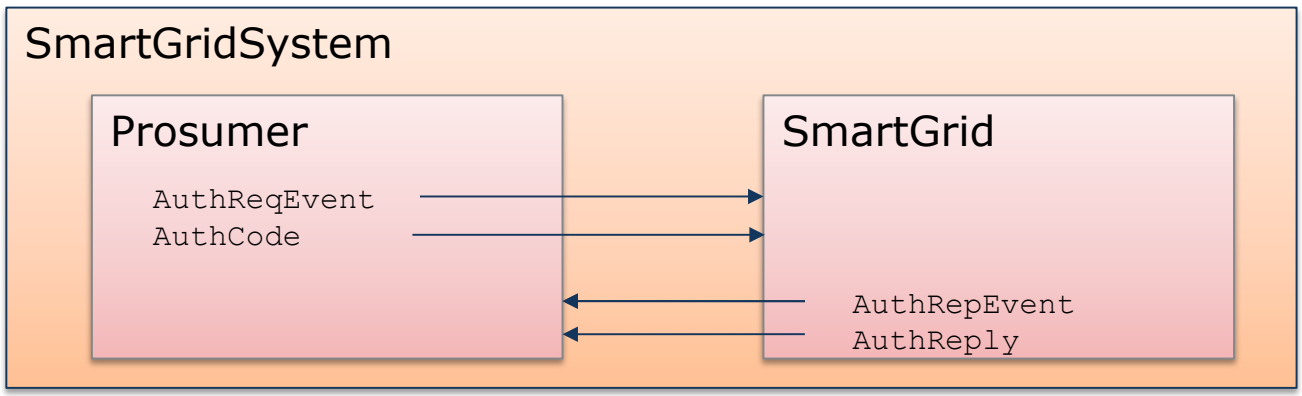
- Prosumers = Producers + Consumers
- Decentral but stable network
- Reliable usage/supply data and new mechanisms needed
- **Dependability /security not an overarching issue in initial designs**

Analysing Security Properties

Authentication: serving only authorized prosumers

Requirement (SMG_SO.7): *Shall*: Every prosumer system shall be authenticated to the micro grid.

Model (simplified):



Security property:

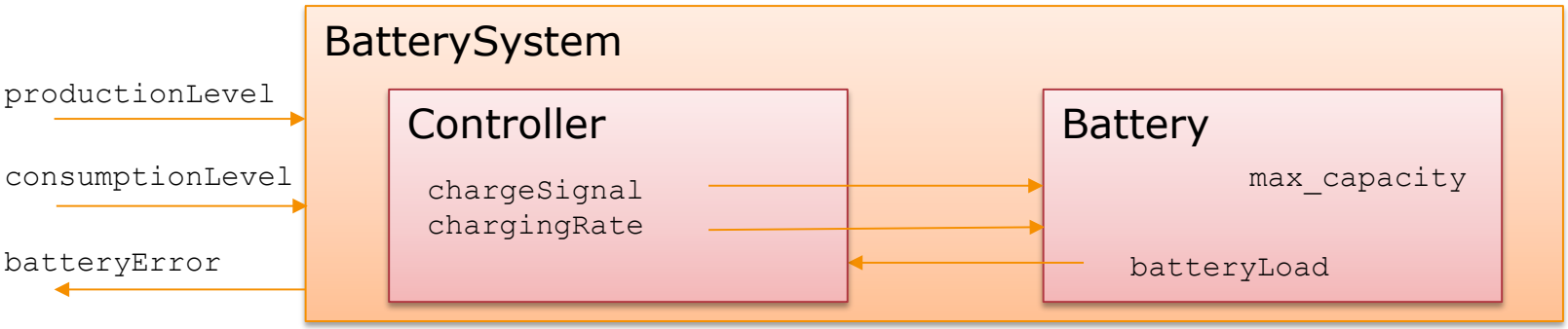
```
(G ({ProAuthCode} = secret -> (F (mode = mode_Authenticated))))
```

Analysing Safety Properties

Battery: prevention of overloading

Requirement (SMG_SA.8): *Shall*: Every battery component shall not be overloaded. This means that if the battery status is full, the control system shall not send any further loading signal.

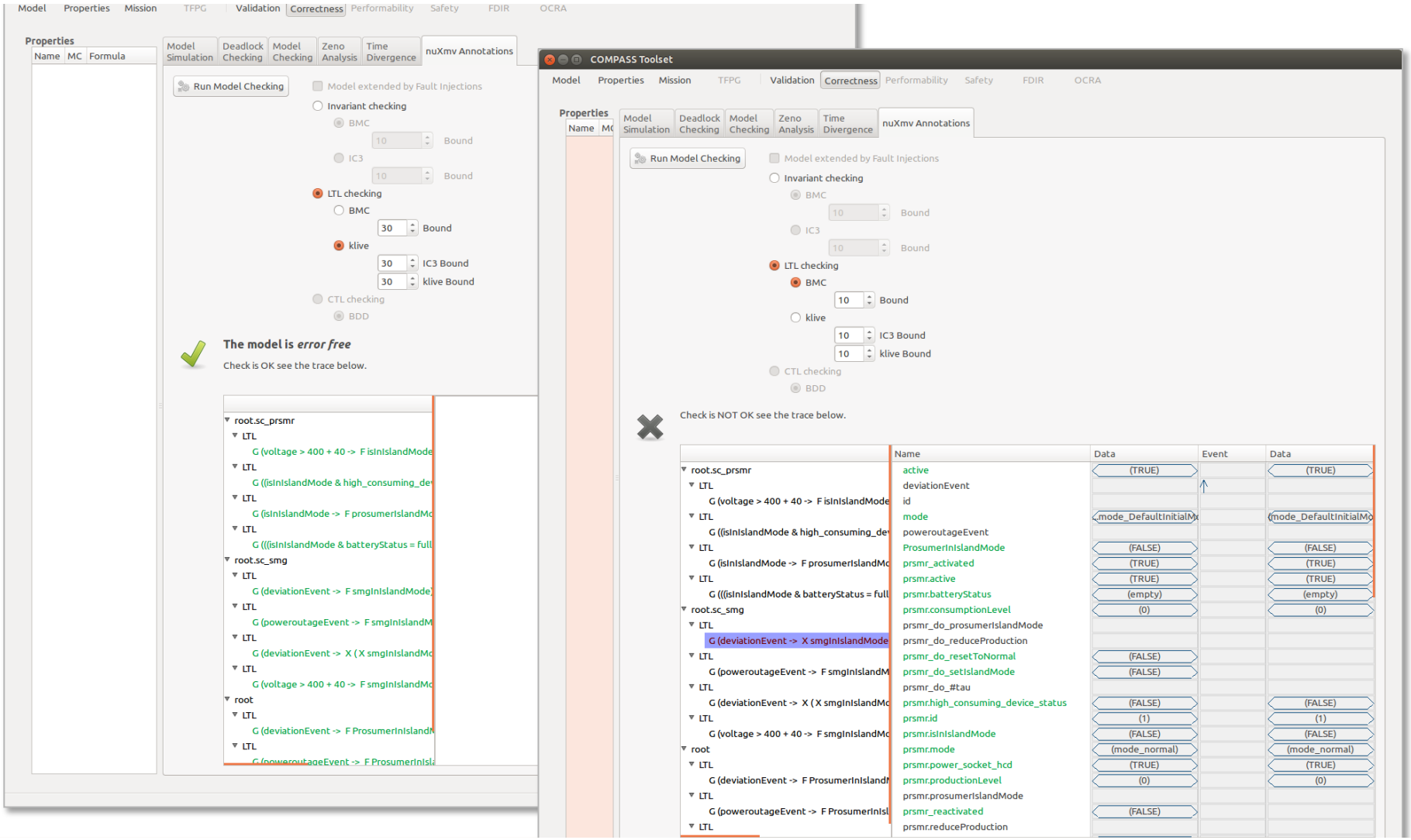
Model (simplified):



Safety properties: ("V" denotes "release")

- `(G (!({batteryLoad} > {max_capacity})))`
- `(G (({batteryLoad} = {max_capacity}) ->`
  `( ({batteryLoad} != {max_capacity})`
  `V (!({charge_signal} & ({chargingRate} > 0))))))`

Property Verification using COMPASS



Compositional Verification I

Safety property formulated as OCRA contract (abbreviated):

```
CONTRACT no_overload
assume:
  always ({productionLevel} >= 0 and {consumptionLevel} >= 0
    and {productionLevel}-{consumptionLevel} <= {max_charging_rate})
guarantee: (always ({batteryError} = false))
```

Refinement for Controller:

```
assume: true;
guarantee:
 (always ({batteryError} implies {batteryLoad > max_capacity})) and
 (always ({dischargingRate}>=0)) and
 (always ({charge_signal} implies
        (((not change({chargingRate}) and (not {done})) since
             ({batteryLoad + chargingRate <= max_capacity} and
              (not change({chargingRate}) and (not {done})))) and
         ((not change({chargingRate})) until {done}))));
```

Compositional Verification II

Refinement for Battery:

```
assume: true;
guarantee:
  {batteryLoad} = 0 and
  (always (change({batteryLoad}) implies {done})) and
  (always ({done} implies
     ({next(batteryLoad) = batteryLoad} or {next(batteryLoad) = 0} or
      {next(batteryLoad) = batteryLoad - dischargingRate} or
      ({next(batteryLoad) = batteryLoad + chargingRate} and
        previously ((not {done}) since ({charge_signal}))))))
```

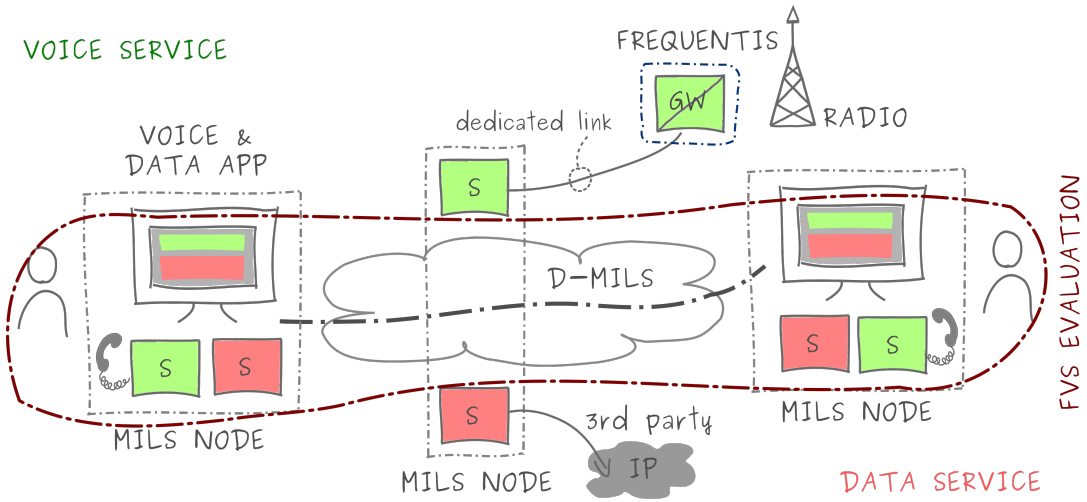
Observation:

- Contracts for the subcomponents are significantly more complex than the corresponding LTL formula used for the (monolithic) verification
- Reason: guarantees of the individual components have to capture a sufficiently strong abstraction of the behaviour of those components in order to establish the correct refinement of the overall property.

Outline

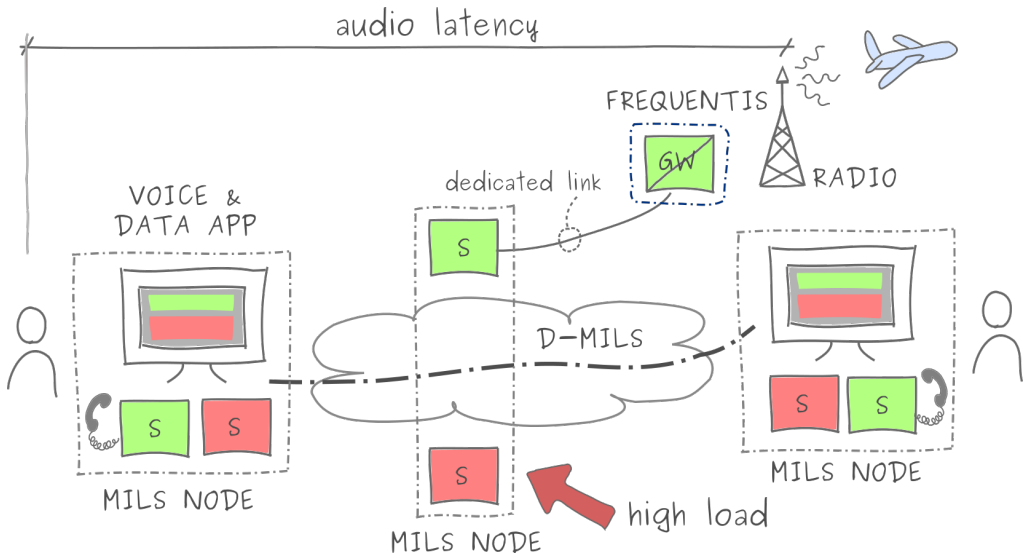
Frequentis Voice Service

- Integrated voice and data app connecting to different domains
- Integration of existing Frequentis services or appliances
- Controlled access to supporting IP services and 3rd party



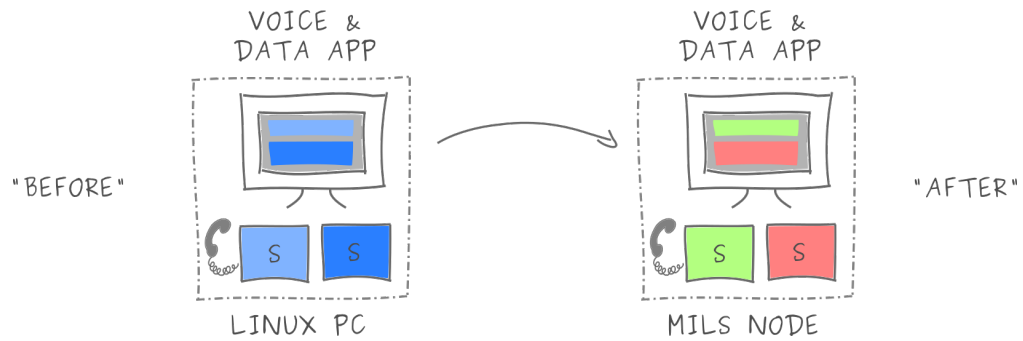
Key Requirements

- No operational impact on voice communication in case of loss or system degradation of any data service component
- End-to-end audio latency less than $100ms$

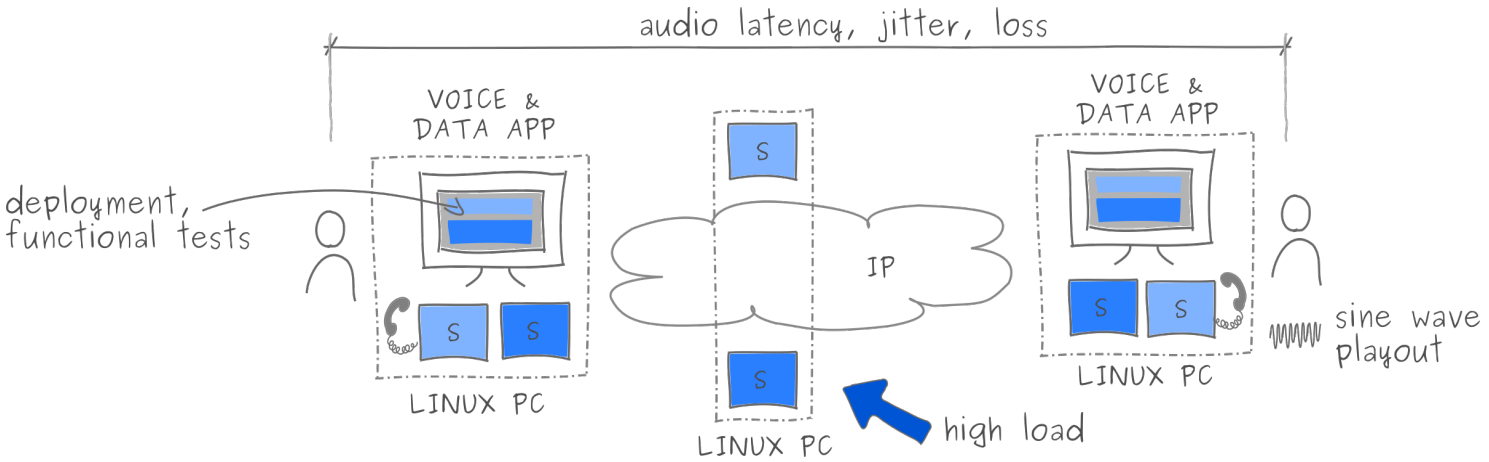


Industrial Evaluation

- Measures of Effectiveness
- Evaluation of the FVS demonstration includes
  - Development
  - Functionality
  - Reliability and Performance
  - Installation, Operation and Maintenance
- Comparing similar 'before' and 'after' measurements on the same HW platform

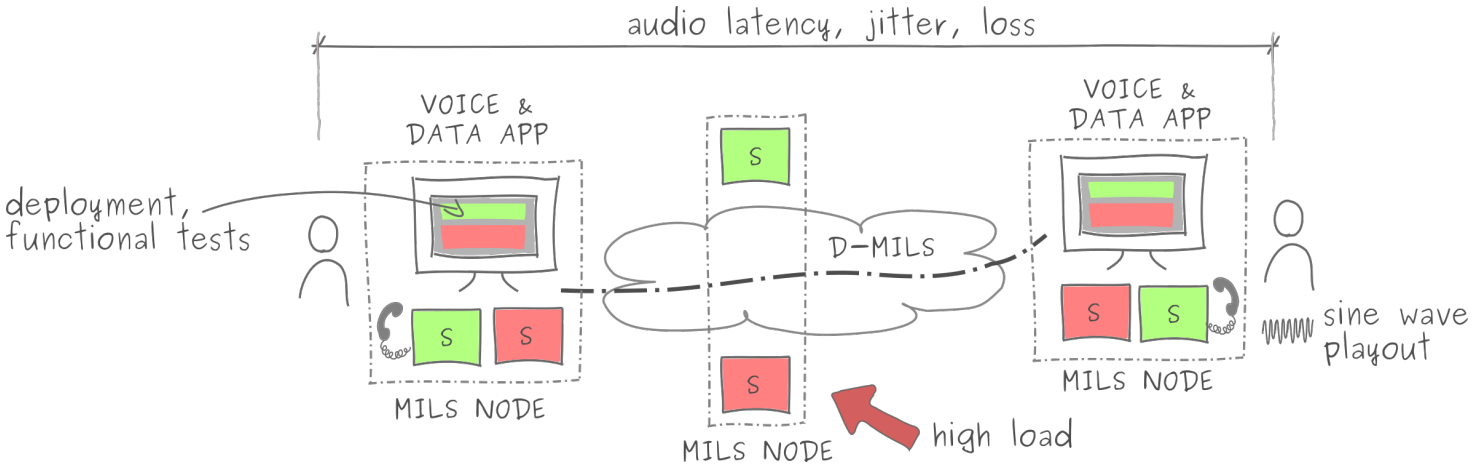


Linux PC Measurements



- Deployment process (installation, configuration ...)
- Media processing and audio latency
- Functional tests (connectivity, voice call, data transfer ...)

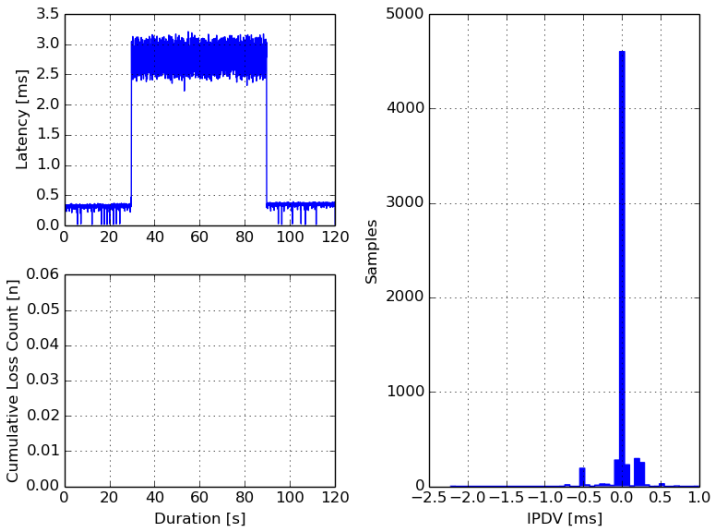
D-MILS Node Measurements



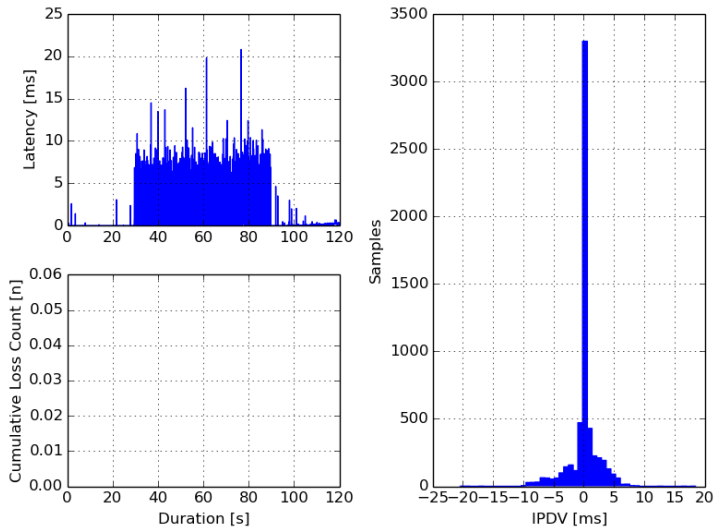
- Deployment process (installation, configuration ...)
- Media processing and audio latency
- Functional tests (connectivity, voice call, data transfer ...)
- Cross-domain effects on latency, jitter and packet loss

Performance Results

- Audio transmission interrupted by TCP date (peak rate)
- Plots show the impact of TCP data traffic on audio
- Tradeoff between performance and virtualization / separation



Reference: Linux PC



D-MILS Node

Outline

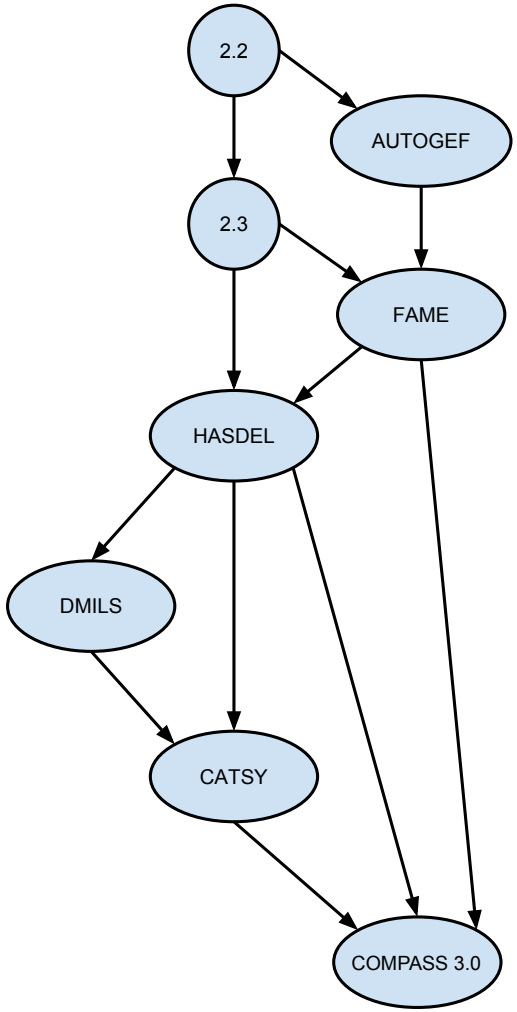
COMPASS Sub-projects and Extensions

On RWTH side:

1. COMPASS started in 2008
2. HASDEL in 2013
3. D-MILS in 2013
4. CATSY in 2015

Other projects (FBK):

1. AUTOGEF (2011)
2. FAME (2012)



Issues

Many previous projects, many ideas. This leads to some issues:

- Legacy code
- Outdated tools
- As many code repositories as there are projects

To address this, there is now:

Issues

Many previous projects, many ideas. This leads to some issues:

- Legacy code
- Outdated tools
- As many code repositories as there are projects

To address this, there is now: COMPASS 3.

COMPASS 3

Project goals:

- Update SLIM language
- Update tools
- Improve examples and tutorials
- Roadmap (also for ESA)

COMPASS 3

Project goals:

- Update SLIM language
- Update tools
- Improve examples and tutorials
- Roadmap (also for ESA)

SLIM Language Changes

Main goal of updated SLIM: re-integration with AADL, including:

- Core AADL V2/V3 language
- Error model annex
- Behavioural annex

SLIM Language Changes

Main goal of updated SLIM: re-integration with AADL, including:

- Core AADL V2/V3 language
- Error model annex
- Behavioural annex

Multiple benefits:

- Easier to get industry acceptance: AADL is a known language
- Possible to use existing AADL tooling
  (e.g., no editor specifically for SLIM)
  - OSATE2 development environment can be used for AADL V2

SLIM Data Types

SLIM predefines a few data types:

- `bool`
- `int`
- `real`
- `clock`
- `continuous`

AADL Data Types

AADL has a data type system that allows the definition of (custom) data types. For example:

Data type package

```
package SLIMdatatypes
public
  data int32
  end int32;
  ...
end SLIMdatatypes;
```

Also useful for (nested) data types defined by the user.

Storing Contracts

Currently, contracts in D-MILS/COMPASS stored as SLIM annotations.

Change: introduce AADL properties system into SLIM

AADL properties

```
system sys
  features
    myPort: in event port {blocking = true;};
  properties
    contracts => [
      {assumption => "true";
       guarantee => "eventually myPort";}];
end sys;
```

AADL Property Sets

Possible properties are defined in property sets.

SLIM properties

```
property set SLIMpropset is
  blocking: aadlboolean => true;
  contract: type record (
    assumption: aadlstring;
    guarantee: aadlstring;);
  contracts: list of SLIMpropset::Contract;
  contractRefinement: record (
    Contract: aadlstring;
    SubContracts: list of aadlstring;);
end SLIMpropset;
```

To be discussed: integration with AADL Constraint Annex

New Pattern System

COMPASS property patterns only available from a limited predefined set.

Patterns

- The system shall have a behaviour where ϕ globally holds.
- The system shall have a behaviour where with probability higher than p it is the case that ψ holds continously within time bound $[t_1, t_2]$.

New Pattern System

COMPASS property patterns only available from a limited predefined set.

Instantiated patterns

- The system shall have a behaviour where $80 \leq \text{voltage} \leq 90$ globally holds.
- The system shall have a behaviour where with probability higher than 0.98 it is the case that $\text{voltage} \geq 80$ holds continously within time bound $[0, 10]$.

New Pattern System

COMPASS property patterns only available from a limited predefined set.

Implemented pattern systems

Formalism	Intended use	Authors
CTL, LTL	functional properties	[Dwyer et al., 1999]
MTL, TCTL	real-time properties	[Konrad & Cheng, 2005]
PCTL, CSL	probabilistic properties	[Grunske, 2008]

New Pattern System

COMPASS property patterns only available from a limited predefined set.

Implemented pattern systems

Formalism	Intended use	Authors
CTL, LTL	functional properties	[Dwyer et al., 1999]
MTL, TCTL	real-time properties	[Konrad & Cheng, 2005]
PCTL, CSL	probabilistic properties	[Grunske, 2008]

A new set of patterns has been defined by Matti, Grunske et al. which is more complete, and can be built from a grammar.

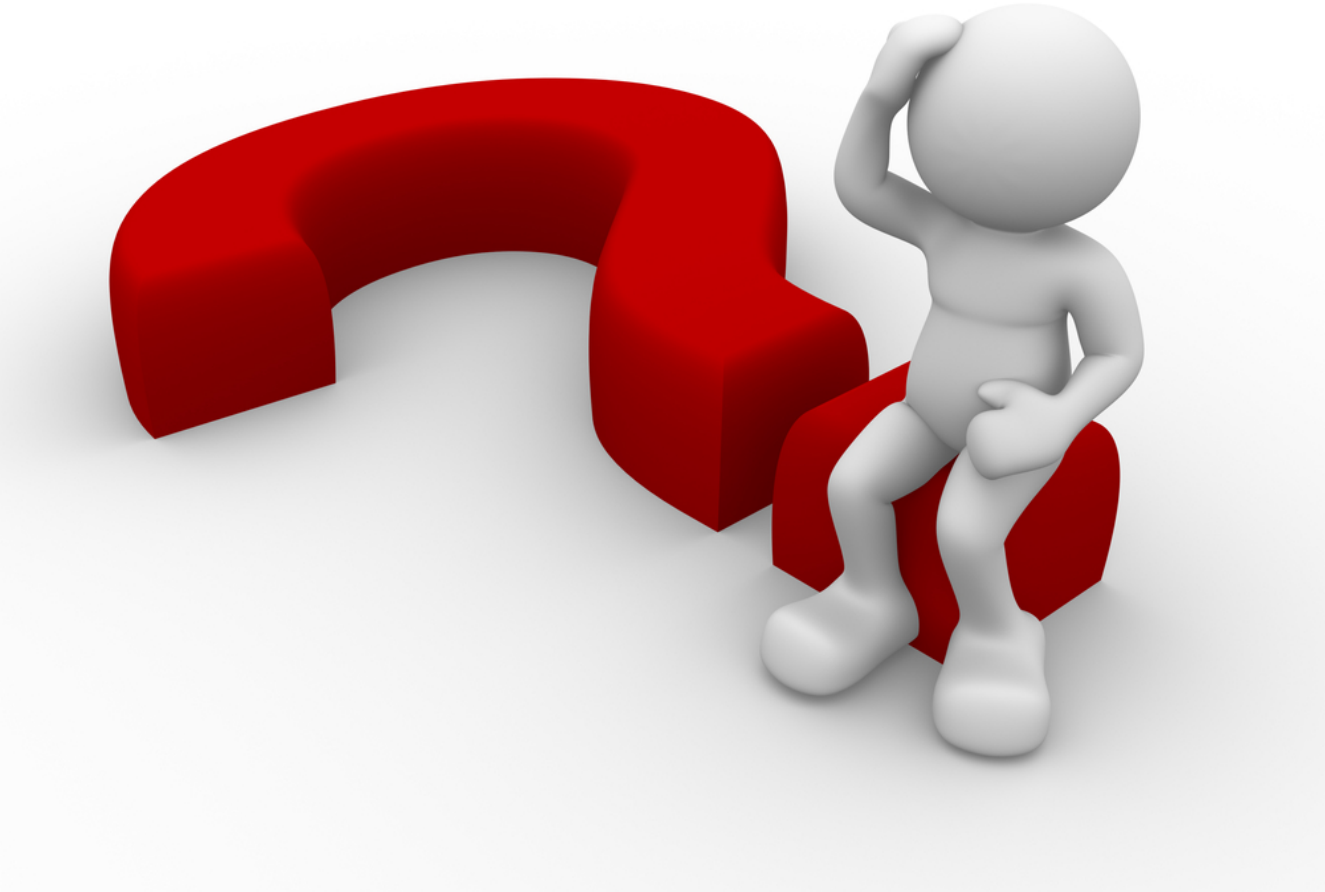
Pattern Grammar I

```
Property ::=
  Scope "," Class

Scope ::=
  "Globally" | "Before" Proposition | "After" Proposition |
  "Between" Proposition "and" Proposition |
  "After" Proposition "until" Proposition |
  ("During" | "While") Proposition

Class ::=
  Occurrence | Order
Occurrence ::=
  Existence | Absence | Universality | Recurrence
Order ::=
  Precedence | Precedence-Invariance | Response |
  Chain-Precedence | Chain-Response | Until

Existence ::=
  {P} "holds eventually" [Time] [Probability]
Absence ::=
  {P} "never holds" [Time] [Probability]
Universality ::=
  {P} "continuously holds" [Time] [Probability]
Recurrence ::=
  {P} "is true repeatedly" ["every" {Time}] [Probability]
```

Pattern Grammar II

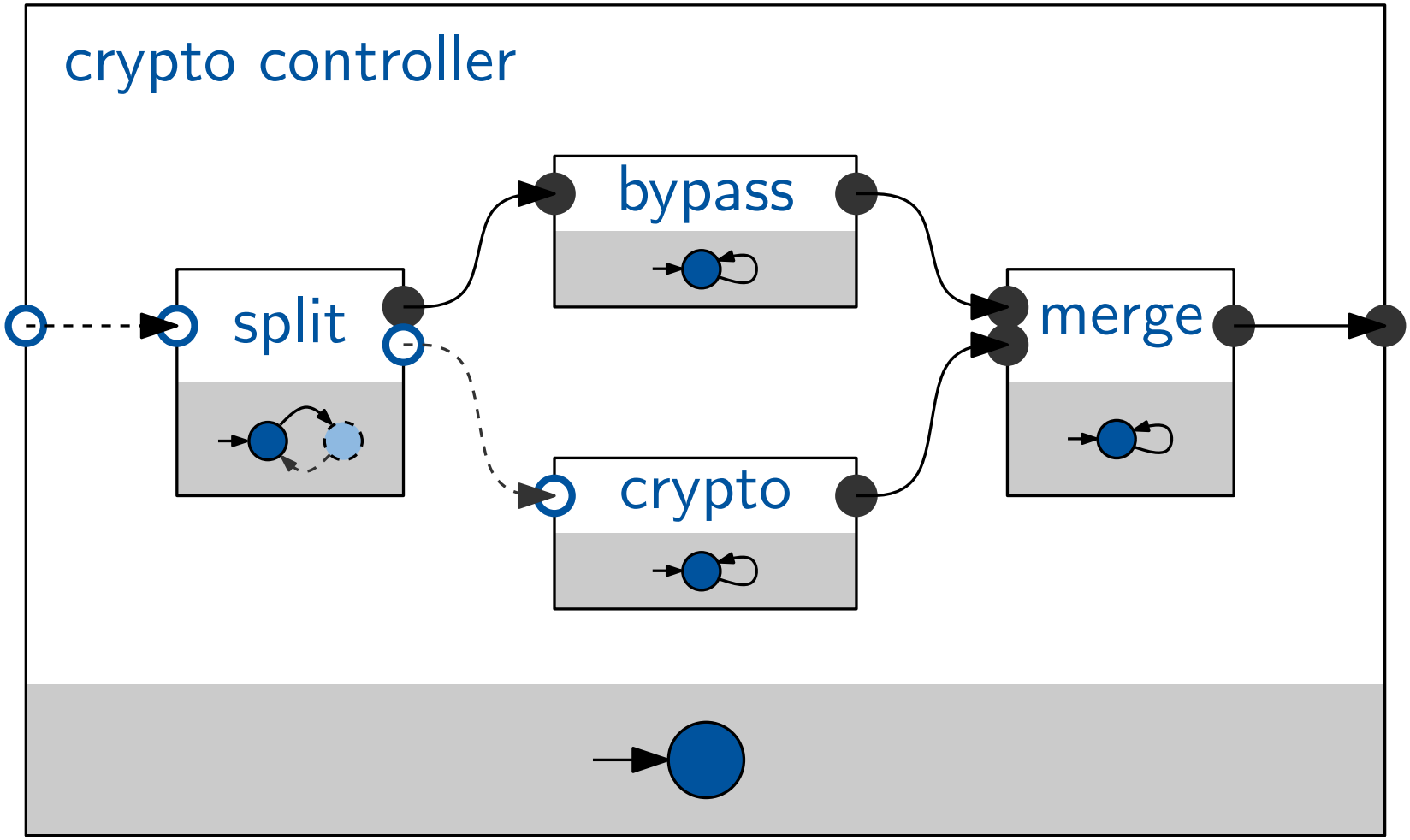
```
Precedence  ::=
  "If" {P} "holds then" {S} "previously held true"
  ["between" {Time} "and" {Time}] [Probability]
Precedence-Invariance ::=
  "If" {P} "holds then" {S} "continuously held true before"
  ["between" {Time} "and" {Time}] [Probability]
Response ::=
  {P} "is followed by" {S} [Time] [Probability]
Chain-Precedence ::=
  "If" {P} "holds then" {S} "previously held true"
       (", before which" {T} "held true")*
Chain-Response ::=
  {P} "is followed by" {S} (", which is followed by" {T})*
Until ::=
  {P} "holds continuously until" {S} "holds" [Time] [Probability]

Time ::=
  "within" {Time} | "after" {Time} |
  "between" {Time} "and" {Time}
Probability ::=
  "with probability" (< | <= | > | >=) {Probability}
```

Questions?



Outline

The Cryptographic Controller Revisited



Abstract Syntax I

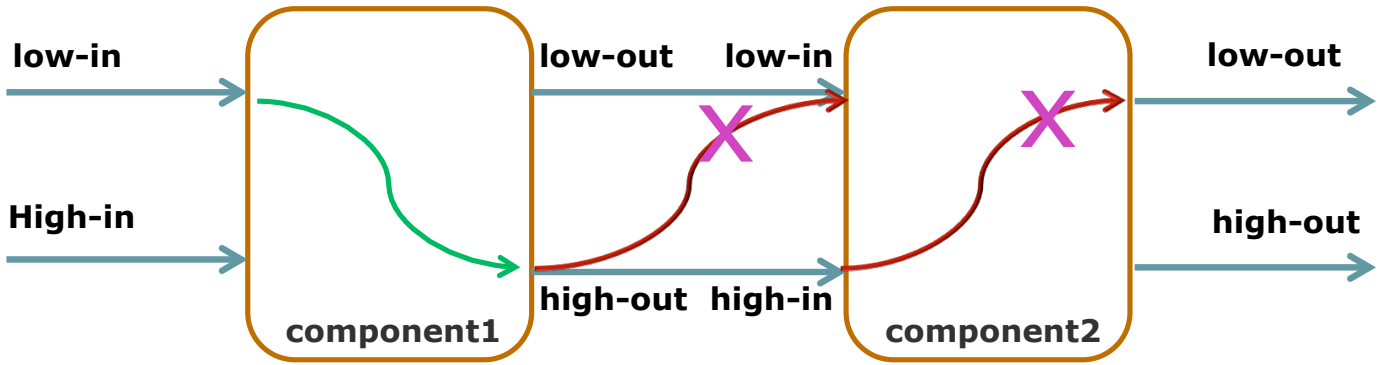
Case	Grammar
Level	$\sigma ::= \texttt{H} \mid \texttt{L}$
Basic type	$t ::= \texttt{int} \mid \texttt{bool} \mid \texttt{enc}\ \tau$
Security type	$\tau ::= t\ \sigma \mid \texttt{key}\ \sigma$
Expression	$e ::= n \mid x \mid e \oplus e$
System	$S ::= \texttt{system}\ s(S^*\ P^*\ C^*\ V^*\ M^*\ T^*)$
Port	$P ::= p : (\texttt{in} \mid \texttt{out})(\texttt{event}\ \sigma \mid \texttt{data}\ \tau\ e)$
Connection	$C ::= ([s.]p, [s.]p)$
Variable	$V ::= x : \tau\ e$
Mode	$M ::= m : [\texttt{initial}]\ \texttt{mode}\ \sigma$
Transition	$T ::= m\ \texttt{-[}\ [p]\ [\texttt{when}\ e]\ [\texttt{then}\ x := e]\ \texttt{]->}\ m'$

Abstract Syntax II

```
system cryptocontroller(
   inframe: in (int L, int H) (0,0)
   outframe: out (int L, enc int H L) (0, encrypt(0, k0))
   system split(...)
   system bypass(...)
   system merge(...)
   system crypto(
      inpayload: in int H 0
      outpayload: out enc int H L encrypt(0, k0)
      k: key L k0
      m: initial mode L
      m-[then outpayload:=encrypt(inpayload,k)]->m
   )
   connection (split.payload, crypto.inpayload)
   connection (crypto.outpayload, merge.payload)
   ⋮
)
```

Information Flow Security

Non-interference:"High-security inputs have no effects on low-security outputs"



- Non-interference property includes:
  - Confidentiality (secrets kept)
  - Integrity (data not corrupted)

Some Security Concepts

- Here: two security levels $\mathtt{L}$ (low/public) and $\mathtt{H}$ (high/confidential/secret/private)
  - partial order $\mathtt{L} \sqsubseteq \mathtt{H}$ ("can flow to")
  - extension to multi-level security by generalisation to lattice

Some Security Concepts

- Here: two security levels $\mathtt{L}$ (low/public) and $\mathtt{H}$ (high/confidential/secret/private)
  - partial order $\mathtt{L} \sqsubseteq \mathtt{H}$ ("can flow to")
  - extension to multi-level security by generalisation to lattice
- Analysis (can be) based on event traces in E^*
  - security assignment $\sigma : E \to \{\mathtt{L}, \mathtt{H}\}$
  - projection $t|_E$ for $t \in E^*$, $E \subseteq E$
  - $t_1, t_2 \in E^*$ called E-equivalent ($t_1 \sim_E t_2$) iff $t_1|_E = t_2|_E$

Some Security Concepts

- Here: two security levels $\mathtt{L}$ (low/public) and $\mathtt{H}$ (high/confidential/secret/private)
  - partial order $\mathtt{L} \sqsubseteq \mathtt{H}$ ("can flow to")
  - extension to multi-level security by generalisation to lattice
- Analysis (can be) based on event traces in E^*
  - security assignment $\sigma : E \to \{\mathtt{L}, \mathtt{H}\}$
  - projection $t|_E$ for $t \in E^*$, $E \subseteq E$
  - $t_1, t_2 \in E^*$ called E-equivalent ($t_1 \sim_E t_2$) iff $t_1|_E = t_2|_E$

Definition (Non-interference [Goguen/Meseguer 1982])

Let $E = In \uplus Out$ and $T \subseteq E^*$. Security assignment σ ensures (event) non-interference if, for all $t_1, t_2 \in T$,

$$t_1 \sim_{In \cap \sigma^{-1}(\mathtt{L})} t_2 \implies t_1 \sim_{Out \cap \sigma^{-1}(\mathtt{L})} t_2$$

Interpretation: behaviour seen by "low" observer unaffected by changes in "high" behaviour

Cryptographically-Masked Information Flow

- **Observation:** encryption breaks traditional non-interference
- Public ciphertexts *do* depend on confidential contents!

Cryptographically-Masked Information Flow

- **Observation:** encryption breaks traditional non-interference
- Public ciphertexts *do* depend on confidential contents!

Example (Password encryption)

- $In = \{pwd1_{\mathtt{H}}, pwd2_{\mathtt{H}}\}$, $Out = \{enc1_{\mathtt{L}}, enc2_{\mathtt{L}}\}$
- $t_1 = pwd1 \cdot enc1$, $t_2 = pwd2 \cdot enc2$
- $t_1|_{In \cap s^{-1}(\mathtt{L})} = \varepsilon = t_2|_{In \cap s^{-1}(\mathtt{L})}$, but
  $t_1|_{Out \cap s^{-1}(\mathtt{L})} = enc1 \neq enc2 = t_2|_{Out \cap s^{-1}(\mathtt{L})}$

$\Rightarrow$ Interference

Cryptographically-Masked Information Flow

- **Observation:** encryption breaks traditional non-interference
- Public ciphertexts *do* depend on confidential contents!

Example (Password encryption)

- $In = \{pwd1_{\mathtt{H}}, pwd2_{\mathtt{H}}\}$, $Out = \{enc1_{\mathtt{L}}, enc2_{\mathtt{L}}\}$
- $t_1 = pwd1 \cdot enc1$, $t_2 = pwd2 \cdot enc2$
- $t_1|_{In \cap s^{-1}(\mathtt{L})} = \varepsilon = t_2|_{In \cap s^{-1}(\mathtt{L})}$, but
  $t_1|_{Out \cap s^{-1}(\mathtt{L})} = enc1 \neq enc2 = t_2|_{Out \cap s^{-1}(\mathtt{L})}$

$\Rightarrow$ Interference

Common approach: declassification

- Allows security level of incoming information to be lowered (here: password)
- Categorisation according to *where/who/when/what* [Sabelfeld/Sands 2005]
- Problems:
  - exceptions to security policy might introduce unforeseen information release
  - systematic handling of re-classification unclear

Adapting Non-Interference

- Non-interference: if a program is run in two low-equivalent environments, the resulting environments are low-equivalent
- Confidentiality thus requires: attacker may not distinguish between ciphertexts
- Naive approach: all ciphertexts are indistinguishable
- But: enables occlusion (security leaks by implicit data flow)

Adapting Non-Interference

- Non-interference: if a program is run in two low-equivalent environments, the resulting environments are low-equivalent
- Confidentiality thus requires: attacker may not distinguish between ciphertexts
- Naive approach: all ciphertexts are indistinguishable
- But: enables occlusion (security leaks by implicit data flow)

Example (Occlusion)

```
m0 -[then low1 := encrypt(val, key)]-> m1;
m1 -[when high then low2 := encrypt(val, key)]-> m2;
m1 -[when not high then low2 := low1]-> m2;
```

Cannot distinguish between `low1` and `low2` even though (in-)equality reflects `high`

Adapting Non-Interference

- Non-interference: if a program is run in two low-equivalent environments, the resulting environments are low-equivalent
- Confidentiality thus requires: attacker may not distinguish between ciphertexts
- Naive approach: all ciphertexts are indistinguishable
- But: enables occlusion (security leaks by implicit data flow)

Example (Occlusion)

```
m0 -[then low1 := encrypt(val, key)]-> m1;
m1 -[when high then low2 := encrypt(val, key)]-> m2;
m1 -[when not high then low2 := low1]-> m2;
```

Cannot distinguish between `low1` and `low2` even though (in-)equality reflects `high`

Wanted: notion of low-equivalence that semantically rejects occlusion without preventing intuitively secure uses

Possibilistic Non-Interference [McCullough 1988]

- Encryption non-deterministically calculates a ciphertext out of a set
- Encrypted values low-equivalent if sets of possible results coincide

Possibilistic Non-Interference [McCullough 1988]

- Encryption non-deterministically calculates a ciphertext out of a set
- Encrypted values low-equivalent if sets of possible results coincide

Definition

$\sim_{\mathrm{L}}$ is a low-equivalence relation on ciphertexts if $\forall v_1, v_2, k_1, k_2$:

1. safe usage:
   $\forall u_1 \in \texttt{encrypt}(v_1, k_1).\exists u_2 \in \texttt{encrypt}(v_2, k_2) : u_1 \sim_{\mathrm{L}} u_2$
2. prevent occlusion:
   $\exists u_1 \in \texttt{encrypt}(v_1, k_1), u_2 \in \texttt{encrypt}(v_2, k_2) : u_1 \not\sim_{\mathrm{L}} u_2$

Possibilistic Non-Interference [McCullough 1988]

- Encryption non-deterministically calculates a ciphertext out of a set
- Encrypted values low-equivalent if sets of possible results coincide

Definition

$\sim_{\mathrm{L}}$ is a low-equivalence relation on ciphertexts if $\forall v_1, v_2, k_1, k_2$:

1. safe usage:
   $\forall u_1 \in \texttt{encrypt}(v_1, k_1).\exists u_2 \in \texttt{encrypt}(v_2, k_2) : u_1 \sim_{\mathrm{L}} u_2$
2. prevent occlusion:
   $\exists u_1 \in \texttt{encrypt}(v_1, k_1), u_2 \in \texttt{encrypt}(v_2, k_2) : u_1 \not\sim_{\mathrm{L}} u_2$

- Lifted to low-equivalence relation $\sim_{\mathrm{L}}$ on values and environments

Definition (Possibilistic non-interference (informal))

If a program is run in two low-equivalent environments, there exists a possibility that each environment produced from the first environment is low-equivalent to some that can be produced from the second environment

Possibilistic Non-Interference and Safe Usage of Encryption

Example (Safe usage of encryption)

```
m0 -[then low := encrypt(high, key)]-> m1;
```

- Let $\sigma(\texttt{high}) = \texttt{H}$ and $\sigma(\texttt{key}) = \sigma(\texttt{low}) = \texttt{L}$
- Let environments η_1, η_2 with $\eta_1 \sim_{\texttt{L}} \eta_2$ such that
  1. $\eta_1(\texttt{high}) = v_1$, $\eta_1(\texttt{key}) = k$
  2. $\eta_2(\texttt{high}) = v_2$, $\eta_2(\texttt{key}) = k$
- Execution respectively yields
  1. $E_1' = \{\eta_1[\texttt{low} \mapsto u_1] \mid u_1 \in \texttt{encrypt}(v_1, k)\}$
  2. $E_2' = \{\eta_2[\texttt{low} \mapsto u_2] \mid u_2 \in \texttt{encrypt}(v_2, k)\}$
- Now

  $\forall u_1 \in \texttt{encrypt}(v_1, k_1).\exists u_2 \in \texttt{encrypt}(v_2, k_2) : u_1 \sim_{\texttt{L}} u_2$

  implies that $\forall \eta_1' \in E_1'.\exists \eta_2' \in E_2' : \eta_1' \sim_{\texttt{L}} \eta_2'$

$\Rightarrow$ Possibilistic non-interference

Possibilistic Non-Interference and Occlusion

Example (Occlusion)

```
m0 -[then low1 := encrypt(val, key)]-> m1;
m1 -[when high then low2 := encrypt(val, key)]-> m2;
m1 -[when not high then low2 := low1]-> m2;
```

- Let $\sigma(\texttt{high}) = \sigma(\texttt{val}) = \texttt{H}$ and $\sigma(\texttt{key}) = \sigma(\texttt{low1}) = \sigma(\texttt{low2}) = \texttt{L}$
- Let environments η_1, η_2 with $\eta_1 \sim_{\texttt{L}} \eta_2$ such that
  1. $\eta_1(\texttt{high}) = \mathit{true}$, $\eta_1(\texttt{val}) = v_1$, $\eta_1(\texttt{key}) = k$
  2. $\eta_2(\texttt{high}) = \mathit{false}$, $\eta_2(\texttt{val}) = v_2$, $\eta_2(\texttt{key}) = k$
- Execution respectively yields
  1. $E_1' = \{\eta_1[\texttt{low1} \mapsto u_1, \texttt{low2} \mapsto u_2] \mid u_1 \in \texttt{encrypt}(v_1, k), u_2 \in \texttt{encrypt}(v_2, k)\}$
  2. $E_2' = \{\eta_2[\texttt{low1} \mapsto u, \texttt{low2} \mapsto u] \mid u \in \texttt{encrypt}(v_1, k)\}$
- Now $\exists u_1 \in \texttt{encrypt}(v_1, k), u_2 \in \texttt{encrypt}(v_2, k) : u_1 \not\sim_{\texttt{L}} u_2$ implies that $\exists \eta_1' \in E_1' : \eta_1'(\texttt{low1}) \not\sim_{\texttt{L}} \eta_1'(\texttt{low2})$
- On the other hand, $\forall \eta_2' \in E_2' : \eta_2'(\texttt{low1}) \sim_{\texttt{L}} \eta_2'(\texttt{low2})$
- Thus $\exists \eta_1' \in E_1'. \forall \eta_2' \in E_2' : \eta_1' \not\sim_{\texttt{L}} \eta_2'$

$\Rightarrow$ Possibilistic interference

The Type Checking Approach

- Introduce typing environment T
  - local variables and data ports $\rightarrow$ security type τ (data type t + security level σ)
  - modes and event ports $\rightarrow$ security level σ

The Type Checking Approach

- Introduce typing environment T
  - local variables and data ports $\rightarrow$ security type τ (data type t + security level σ)
  - modes and event ports $\rightarrow$ security level σ
- Specify typing rules
  - parametrised by T
  - derive types of connections and transitions
- Example: encryption and decryption

$$\frac{T \vdash e_1 : \tau \quad T \vdash e_2 : \texttt{key L}}{T \vdash \texttt{encrypt}(e_1, e_2) : \texttt{enc}\ \tau\ \texttt{L}}$$

$$\frac{T \vdash e_1 : \texttt{enc}\ \tau\ \sigma \quad T \vdash e_2 : \texttt{key H}}{T \vdash \texttt{decrypt}(e_1, e_2) : \tau^{\sigma}}$$

The Type Checking Approach

- Introduce typing environment T
  - local variables and data ports $\rightarrow$ security type τ (data type t + security level σ)
  - modes and event ports $\rightarrow$ security level σ
- Specify typing rules
  - parametrised by T
  - derive types of connections and transitions
- Example: encryption and decryption

$$\frac{T \vdash e_1 : \tau \quad T \vdash e_2 : \texttt{key L}}{T \vdash \texttt{encrypt}(e_1, e_2) : \texttt{enc}\ \tau\ \texttt{L}}$$

$$\frac{T \vdash e_1 : \texttt{enc}\ \tau\ \sigma \quad T \vdash e_2 : \texttt{key H}}{T \vdash \texttt{decrypt}(e_1, e_2) : \tau^\sigma}$$

Theorem ([MILS Workshop 2015])

If the system is typeable, it is possibilistically non-interfering.