D-MILS and MILS-AADL

Thomas Noll

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

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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"
The architecture expresses an interaction policy among a collection of components.

Circles represent architectural components (subjects / objects).

Arrows represent interactions.

The absence of an arrow is as significant as the presence of one.

This component has no interaction with any other.

Components are assumed to perform the functions specified by the architect (trusted components enforce a local policy).

Suitability of the architecture for some purpose presumes that the architect’s assumptions are met in the implementation of the architecture diagram.
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

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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

Security Analysis → Safety Analysis → Performance Analysis → Architectural Refinement

MILS AADL → MILS AADL → MILS AADL

App A
Level B
Classified

Autofocus Model
C Code

Autocode

Simulink Model
C Code

App B
Level C
Unclassified

App C
Level A
Top Secret

Ada Code

Configurations / Schedules / Communication Routes

Node1

Node2

Node3

MILS Platform Configuration Compiler

MILS Technical Platform
D-MILS Platform

Distributed MILS: Policy architecture deployment spanning nodes

Foundational Plane

Subjects

MNS

SK

Node Hardware

Subjects

MNS

SK

Node Hardware

SK ⊕ MNS

Foundational Plane

Node Hardware

Node Hardware

Thomas Noll, RWTH Aachen University
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

- Architecture Analysis and Design Language MILS-AADL
- Integration GSN & AADL
- Configuration Synthesis
- Graphical & Declarative Languages
  - Behavior Annotation
  - Property Annotation
  - Goal Structuring Notation
- MILS Platform
  - Configuration Compiler
  - Target Configuration tools
  - D-MILS Platform target
- Ext. Time Triggered Ethernet
  - Pre-existing products
- LSK

Compositional Assurance Case
- Verification Framework
- Assurance Framework
- Compositional Verification
- Representation Semantics and Transformations
- Pre-existing products

D-MILS and MILS-AADL
Outline

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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

crypto controller

Thomas Noll, RWTH Aachen University
Example: Cryptographic Controller II

crypto controller

split  bypass

crypto

merge
Example: Cryptographic Controller II

crypto controller

 split

 bypass

 crypto

 merge
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 $f : \text{function } t_1, \ldots, t_n \to t_0$ ($n \geq 1$, $t_i \in \text{Type}$)

- $[[\text{encrypt}]] : D_t \times D_{\text{key}} \to D_{\text{enc}(t)}$ and $[[\text{decrypt}]] : D_{\text{enc}(t)} \times D_{\text{key}} \to D_t$ such that

  \[ [[\text{decrypt}]]([[\text{encrypt}]](d, k_1), k_2) = d \]

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

- $[[\text{sign}]] : D_t \times D_{\text{key}} \to D_{\text{sgn}(t)}$ and $[[\text{verify}]] : D_{\text{sgn}(t)} \times D_{\text{key}} \to D_t$ such that

  \[ [[\text{verify}]]([[\text{sign}]](d, k_1), k_2) = d \]

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

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
Contract-Based Design

D⊨𝑃↓D , E⊨𝑃↓E /γ↓𝐵 (𝐷,𝐸)⊨γ↓𝐵 (𝑃↓D ,𝑃↓E ) ... Hardware
SK ⊕ MNS 
Foundational Plane
Node Hardware
Subjects
WP4: Compositional Verification D-MILS Final Review
Compositional Verification Tool Chain

COMPASS extended with contract-based analysis

New translation to OCRA

OCRA extended with EUF

nuXmv extended with EUF

MILS-AADL analysis

Contract refinement

Temporal logic entailment

Contract-based fault-tree analysis

FTA as param synthesis

New algorithm for temporal logics with infinite-state systems

New algorithm for invariants with infinite-state systems

Model checking of liveness

Model checking of invariant

COMPASS

OCRA

nuXmv

OCRA

nuXmv

xSAP

Thomas Noll, RWTH Aachen University

D-MiLS and MILS-AADL

23/61
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 \((p \text{ implies in the future } q)\)
- First-order
  - always \((\text{high}(\text{value}) \iff \text{high}(\text{cmd})) \text{ implies never } \text{high}(\text{output}))\)
- Real-time
  - always \((\text{corrupted}(\text{memory}) \text{ implies } \text{time\_until}(\text{alarm}) \leq \text{time\_bound})\)
Implementation in COMPASS Toolset I

COMPASS with contracts

8 D-MILS Final Review WP4: Compositional Verification

OCRA VIEW

3 TYPES OF ANALYSIS

CONTRACTS VIEWER
Implementation in COMPASS Toolset II
Implementation in COMPASS Toolset III

FAULT-TREE ANALYSIS
Outline

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
The MILS Platform Configuration Compiler

**MPCC: MILS Platform Configuration Compiler**

- **Policy Architecture**
- **MILS-AADL**
- **D-MILS Platform**
- **MILS-AADL**
- **SK Configuration XML**
- **GIFP Configuration XML**
- **Network Configuration XML**
- **MCNF**
- **External Representation** (Prolog or XML)
- **User Interface**
- **User Interaction**
- **Flexible retargeting by adapters**
- **MILS to Prolog translator**
- **Prolog Engine**
- **Frontend**
- **MCNF**
- **Backend**

Thomas Noll, RWTH Aachen University

D-MILS and MILS-AADL
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

Node 1

Node 2
Outline

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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
Authentication: serving only authorized prosumers

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

**Model (simplified):**

```
SmartGridSystem

Prosumer
AuthReqEvent
AuthCode

SmartGrid
AuthRepEvent
AuthReply
```

**Security property:**

\[
(G (\{\text{ProAuthCode}\} = \text{secret} \rightarrow (F (\text{mode} = \text{mode\_Authenticated}))))
\]
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):**

```
BatterySystem

Controller
  chargeSignal
  chargingRate

Battery
  max_capacity
  batteryLoad

productionLevel
consumptionLevel
batteryError
```

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

- \((G (\neg (\{\text{batteryLoad}\} > \{\text{max\_capacity}\})))\)
- \((G ((\{\text{batteryLoad}\} = \{\text{max\_capacity}\}) \rightarrow ( (\{\text{batteryLoad}\} \neq \{\text{max\_capacity}\}) \lor V (\neg (\{\text{charge\_signal}\} \land (\{\text{chargingRate}\} > 0)))))))\)
Property Verification using COMPASS

The model is error free:
Check is OK see the trace below.

The model is NOT error free:
Check is NOT OK see the trace below.
Safety property formulated as OCRA contract (abbreviated):

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

**Refinement for Controller:**

```plaintext
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}))));
```
Refinement for Battery:

```plaintext
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

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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
- 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

Thomas Noll, RWTH Aachen University
D-MILS and MILS-AADL
Outline

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
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)
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:
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)
Project goals:

- Update SLIM language
- Update tools
- Improve examples and tutorials
- Roadmap (also for ESA)
Main goal of updated SLIM: re-integration with AADL, including:

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

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

Also useful for (nested) data types defined by the user.
Currently, contracts in D-MILS/COMPASS stored as SLIM annotations.

Change: introduce AADL properties system into SLIM

```plaintext
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**

```plaintext
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 $\phi$ globally holds.
- The system shall have a behaviour where with probability higher than $p$ it is the case that $\psi$ holds continuously 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 continuously within time bound $[0, 10]$. 
New Pattern System

COMPASS property patterns only available from a limited predefined set.

### Implemented pattern systems

<table>
<thead>
<tr>
<th>Formalism</th>
<th>Intended use</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL, LTL</td>
<td>functional properties</td>
<td>[Dwyer et al., 1999]</td>
</tr>
<tr>
<td>MTL, TCTL</td>
<td>real-time properties</td>
<td>[Konrad &amp; Cheng, 2005]</td>
</tr>
<tr>
<td>PCTL, CSL</td>
<td>probabilistic properties</td>
<td>[Grunske, 2008]</td>
</tr>
</tbody>
</table>
New Pattern System

COMPASS property patterns only available from a limited predefined set.

<table>
<thead>
<tr>
<th>Implemented pattern systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Formalism</strong></td>
</tr>
<tr>
<td>CTL, LTL</td>
</tr>
<tr>
<td>MTL, TCTL</td>
</tr>
<tr>
<td>PCTL, CSL</td>
</tr>
</tbody>
</table>

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

1. The MILS Approach
2. The D-MILS Project
3. The MILS-AADL Language
4. System Analysis
5. System Deployment
6. Evaluation on Smart Microgrid
7. Evaluation on Voice Service
8. The Future
9. Example Analysis: Security Type Checking
Abstract Syntax I

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

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 $L$ (low/public) and $H$ (high/confidential/secret/private)
  - partial order $L \sqsubseteq H$ ("can flow to")
  - extension to multi-level security by generalisation to lattice
Some Security Concepts

- Here: two security levels $L$ (low/public) and $H$ (high/confidential/secret/private)
  - partial order $L \sqsubseteq H$ (“can flow to”)
  - extension to multi-level security by generalisation to lattice
- Analysis (can be) based on event traces in $E^*$
  - security assignment $\sigma : E \rightarrow \{L, H\}$
  - 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 $L$ (low/public) and $H$ (high/confidential/secret/private)
  - partial order $L \sqsubseteq H$ ("can flow to")
  - extension to multi-level security by generalisation to lattice

- Analysis (can be) based on event traces in $E^*$
  - security assignment $\sigma : E \rightarrow \{L, H\}$
  - 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 $E = \text{In} \uplus \text{Out}$ and $T \subseteq E^*$. Security assignment $\sigma$ ensures (event) non-interference if, for all $t_1, t_2 \in T$,

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

**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)**

- $In = \{pwd_1_H, pwd_2_H\}$, $Out = \{enc_1_L, enc_2_L\}$
- $t_1 = pwd_1 \cdot enc_1$, $t_2 = pwd_2 \cdot enc_2$
- $t_1 |_{In \cap s^{-1}(L)} = \varepsilon = t_2 |_{In \cap s^{-1}(L)}$, but
  - $t_1 |_{Out \cap s^{-1}(L)} = enc_1 \neq enc_2 = t_2 |_{Out \cap s^{-1}(L)}$
- $\Rightarrow$ Interference
Cryptographically-Masked Information Flow

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

**Example (Password encryption)**

- \( In = \{pwd_1^H, pwd_2^H\} \), \( Out = \{enc_1^L, enc_2^L\} \)
- \( t_1 = pwd_1 \cdot enc_1, t_2 = pwd_2 \cdot enc_2 \)
- \( t_1|_{In \cap s^{-1}(L)} = \varepsilon = t_2|_{In \cap s^{-1}(L)}, \) but
  - \( t_1|_{Out \cap s^{-1}(L)} = enc_1 \neq enc_2 = t_2|_{Out \cap s^{-1}(L)} \)
  - \( \Rightarrow \) 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.
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)

\[
\begin{align*}
m0 & \rightarrow [\text{then } \text{low1} := \text{encrypt}(\text{val}, \text{key})] \rightarrow m1; \\
m1 & \rightarrow [\text{when high then } \text{low2} := \text{encrypt}(\text{val}, \text{key})] \rightarrow m2; \\
m1 & \rightarrow [\text{when not high then } \text{low2} := \text{low1}] \rightarrow m2; \\
\end{align*}
\]

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

Definition

\[ L \text{ is a low-equivalence relation on ciphertexts if} \]

\[ \forall v_1, v_2, k_1, k_2: \]

1. **safe usage:**
   \[ \forall u_1 \in \text{encrypt}(v_1, k_1). \exists u_2 \in \text{encrypt}(v_2, k_2): u_1 \sim_L u_2 \]

2. **prevent occlusion:**
   \[ \exists u_1 \in \text{encrypt}(v_1, k_1), u_2 \in \text{encrypt}(v_2, k_2): u_1 \not\sim_L u_2 \]

Lifted to low-equivalence relation \( \sim_L \) 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 [McCullough 1988]

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

**Definition**

\( \sim_L \) is a low-equivalence relation on ciphertexts if \( \forall v_1, v_2, k_1, k_2: \)

1. safe usage:
   \( \forall u_1 \in \text{encrypt}(v_1, k_1). \exists u_2 \in \text{encrypt}(v_2, k_2): u_1 \sim_L u_2 \)
2. prevent occlusion:
   \( \exists u_1 \in \text{encrypt}(v_1, k_1), u_2 \in \text{encrypt}(v_2, k_2): u_1 \not\sim_L u_2 \)
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**

\(\sim_L\) is a low-equivalence relation on ciphertexts if \(\forall v_1, v_2, k_1, k_2:\)

1. safe usage:
   \[\forall u_1 \in \text{encrypt}(v_1, k_1). \exists u_2 \in \text{encrypt}(v_2, k_2) : u_1 \sim_L u_2\]
2. prevent occlusion:
   \[\exists u_1 \in \text{encrypt}(v_1, k_1), u_2 \in \text{encrypt}(v_2, k_2) : u_1 \not\sim_L u_2\]

- Lifted to low-equivalence relation \(\sim_L\) 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.
Example (Safe usage of encryption)

\[ m_0 \rightarrow \text{then low := encrypt(high, key)} \rightarrow m_1; \]

- Let \( \sigma(\text{high}) = H \) and \( \sigma(\text{key}) = \sigma(\text{low}) = L \)
- Let environments \( \eta_1, \eta_2 \) with \( \eta_1 \sim_L \eta_2 \) such that
  1. \( \eta_1(\text{high}) = v_1, \eta_1(\text{key}) = k \)
  2. \( \eta_2(\text{high}) = v_2, \eta_2(\text{key}) = k \)
- Execution respectively yields
  1. \( E'_1 = \{ \eta_1[\text{low} \mapsto u_1] \mid u_1 \in \text{encrypt}(v_1, k) \} \)
  2. \( E'_2 = \{ \eta_2[\text{low} \mapsto u_2] \mid u_2 \in \text{encrypt}(v_2, k) \} \)
- Now
  \( \forall u_1 \in \text{encrypt}(v_1, k_1). \exists u_2 \in \text{encrypt}(v_2, k_2) : u_1 \sim_L u_2 \)
  implies that \( \forall \eta'_1 \in E'_1. \exists \eta'_2 \in E'_2 : \eta'_1 \sim_L \eta'_2 \)

\( \Rightarrow \) Possibilistic non-interference
Possibilistic Non-Interference and Occlusion

Example (Occlusion)

\[ m_0 -\{\text{then } \text{low1} := \text{encrypt}(\text{val}, \text{key})\} \rightarrow m_1; \]
\[ m_1 -\{\text{when high then } \text{low2} := \text{encrypt}(\text{val}, \text{key})\} \rightarrow m_2; \]
\[ m_1 -\{\text{when not high then } \text{low2} := \text{low1}\} \rightarrow m_2; \]

Let \( \sigma(\text{high}) = \sigma(\text{val}) = H \) and \( \sigma(\text{key}) = \sigma(\text{low1}) = \sigma(\text{low2}) = L \)

Let environments \( \eta_1, \eta_2 \) with \( \eta_1 \sim_L \eta_2 \) such that

1. \( \eta_1(\text{high}) = \text{true}, \eta_1(\text{val}) = v_1, \eta_1(\text{key}) = k \)
2. \( \eta_2(\text{high}) = \text{false}, \eta_2(\text{val}) = v_2, \eta_2(\text{key}) = k \)

Execution respectively yields

1. \( E'_1 = \{\eta_1[\text{low1} \mapsto u_1, \text{low2} \mapsto u_2] \mid u_1 \in \text{encrypt}(v_1, k), u_2 \in \text{encrypt}(v_2, k)\} \)
2. \( E'_2 = \{\eta_2[\text{low1} \mapsto u, \text{low2} \mapsto u] \mid u \in \text{encrypt}(v_1, k)\} \)

Now \( \exists u_1 \in \text{encrypt}(v_1, k), u_2 \in \text{encrypt}(v_2, k) : u_1 \not\sim_L u_2 \)
implies that \( \exists \eta'_1 \in E'_1 : \eta'_1(\text{low1}) \not\sim_L \eta'_1(\text{low2}) \)

On the other hand, \( \forall \eta'_2 \in E'_2 : \eta'_2(\text{low1}) \sim_L \eta'_2(\text{low2}) \)

Thus \( \exists \eta'_1 \in E'_1 : \forall \eta'_2 \in E'_2 : \eta'_1(\text{low1}) \not\sim_L \eta'_2(\text{low2}) \)

\[ \Rightarrow \text{Possibilistic interference} \]
The Type Checking Approach

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

**Example:** encryption and decryption

\[
T \vdash e_1 : \tau \\
T \vdash e_2 : \text{key} \ L \\
T \vdash \text{encrypt}(e_1, e_2) : \text{enc} \tau L \\
T \vdash e_1 : \text{enc} \tau \sigma T \\
T \vdash e_2 : \text{key} \ H \\
T \vdash \text{decrypt}(e_1, e_2) : \tau \sigma
\]

**Theorem ([MILS Workshop 2015])**

If the system is typeable, it is possibilistically non-interfering.
The Type Checking Approach

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

$$
egin{align*}
T \vdash e_1 : \tau & \quad T \vdash e_2 : \text{key} \ L \\
T \vdash \text{encrypt}(e_1, e_2) : \text{enc} \ \tau \ L \\
T \vdash e_1 : \text{enc} \ \tau \ \sigma & \quad T \vdash e_2 : \text{key} \ H \\
T \vdash \text{decrypt}(e_1, e_2) : \tau^\sigma
\end{align*}
$$

Theorem ([MILS Workshop 2015])

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

Thomas Noll, RWTH Aachen University
D-MiLS and MILS-AADL
The Type Checking Approach

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

\[
\begin{align*}
T \vdash e_1 : \tau & \quad T \vdash e_2 : \text{key } L \\
T \vdash \text{encrypt}(e_1, e_2) : \text{enc } \tau L \\
T \vdash e_1 : \text{enc } \tau \sigma & \quad T \vdash e_2 : \text{key } H \\
T \vdash \text{decrypt}(e_1, e_2) : \tau^\sigma
\end{align*}
\]

**Theorem** ([MILS Workshop 2015]):

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