Distributed MILS (D-MILS) Specification, Analysis, Deployment, and Assurance of Distributed Critical Systems

Harald Rueß (fortiss, München)
Stefano Tonetta (FBK, Trento)
D-MiLS Consortium

CONSORTIUM PARTNERS:
Fondazione Bruno Kessler (IT)
fortiss (DE)
Frequentis (AT)
LynuxWorks (FR)
RWTH Aachen University (DE)
The Open Group (UK) Lead
TTTech (AT)
Université Joseph Fourier (FR)
University of York (UK)
Overview

Part 1: D-MILS project overview
- Overview of the consortium
- Objectives of the project and areas of work
- Overview of the approach and the D-MILS platform
- Specification language
- Verification framework
- Deployment on the D-MILS platform
- Assurance case

Part 2: Verification framework
- Overview of the compositional approach
- Target requirements
- Annotation language
- Verification algorithms
- Tool support
Scientific and Technical Objectives Summary

- High-level specification in declarative languages
- Comprehensive: “Top-to-bottom” and “End-to-end”
- Pervasive automation support
- Compositional verification of desired properties
- Integrated assurance case for certification support
- Distributed platform configuration compilation
- Strong analytical environment
  - Security and dependability attributes of system computed from the properties of the components and the architecture
Scientific and Technical Objectives

“Top-to-bottom” coverage:

♦ High-level, graphical architectural design in AADL
♦ Behavior specification with AADL behavioral annex
♦ Property specifications in AADL annotations
♦ Integrated verification represented via graphical Goal Structuring Notation (GSN)
♦ Architectural-level verification
♦ Automated inventory of hardware platform resources
♦ Synthesis of low-level component configurations
Scientific and Technical Objectives

“End-to-end” coverage:

- Implementation-independent architectural specification
- High-level specification of dependability attributes
- Seamless realization of distributed architectures
- Verify that component composition supports dependability attributes
- Modular and scalable deterministic platform
- Incremental binding of architecture, implementation, integration, and deployment parameters
Technical Results Expected

- Standardized, **component-based** high-assurance distributed platform
- **Compositional assurance** of systems from component assurance and composition analysis
- **Framework for certification** of systems built on the platform supported by **extensive automation**
- Enable application architectures to **seamlessly span multiple nodes**, for scalable determinism
- **Industrial D-MILS Pilots / Technology Evaluation**
  - Frequentis Voice Services
  - fortiss Smart Microgrid
A single policy architecture may span multiple D-MILS nodes expressed in declarative MILS-AADL

Guarantees similar to a single MILS node: isolation, information flow control, determinism

Determinism over network could be achieved in various ways – D-MILS uses Time-Triggered Ethernet

Configure and schedule the network and the processors of the nodes coherently

Verify architectural-based properties, develop GSN assurance case, synthesize platform configuration, using integrated tool chain leveraging existing verification technology (nuSMV, OCRA, BIP, AF3)
D-MiLS Research and Technology Development Areas

- **Architecture Analysis and Design Language MILS-AADL**
  - Behavior Annotation
  - Property Annotation
  - Goal Structuring Notation

- **Intermediate Languages**
  - Verification Framework
  - Assurance Framework

- **MILS Platform Configuration Compiler**
  - Target Configuration tools
  - D-MiLS Platform target

- **D-MiLS Platform**
  - Extended Separation Kernel
  - Ext. Time Triggered Ethernet

- **Integration GSN & AADL**

- **Configuration Synthesis**

- **Graphical & Declarative Languages**

- **Compositional Verification**

- **Compositional Assurance Case**

- **Representation Semantics and Transformations**

- **Pre-existing products**
Distributed MILS (D-MILS): Policy architecture deployment spanning nodes

MNS – MILS Networking System    SK – Separation Kernel
Distributed MILS Platform – MILS nodes with deterministic communication

A Distributed MILS Platform:

**Enables:**

- Realization of deterministic distributed MILS architectures

**Foundational Plane**

**SK ⊕ MNS**

Node Hardware  Node Hardware  Node Hardware  Node Hardware  Node Hardware

TTEthernet
The policy architecture:

...may be deployed on a *distributed MILS separation kernel* with two nodes, MNS and TTEthernet as follows:
Demonstrator: fortiss Smart Microgrid
Smart grid sends the current price of energy.

Each prosumer sends a plan indicating how much energy it intends to consume and provide during the day.

Smart grid checks whether the grid can support the resulting consumption or production.

If the overall plan is not feasible, the prosumers need to modify their plans and resend them.

The negotiation continues until the plans are accepted.
Smart Microgrid Prosumers

D-MiLS Project Overview © 2015 D-MiLS Project 15
Demonstrator: Frequentis Voice Services

cwp... controller working position
rce... radio control equipment
r-rce... remote rce
c-rce... center rce
swim... system wide information management
Summary of Accomplishments to Date

- Defined syntax and formal semantics of **MILS-AADL dialect**
- Parser for MILS-AADL
- Transformations of MILS-AADL for verification and configuration
- Compositional verification framework for MILS-AADL models
- Foundations and tool support for compositional GSN assurance cases
- Synthesis of MILS component configuration data for target components
- **Operational D-MILS Platform** (distributed LynxSecure separation kernel running over TTEthernet)
- **MILS Platform Configuration Compiler** providing synthesis of configuration data for target platform components
- Two industrial demonstrators in progress: fortiss smart micro grid and Frequentis Voice Services
The framework consists of a collection of tools integrated to support modeling, validation and verification.

Modeling language: MILS-AADL
- With a formal semantics

Validation with
- Simulation
- Deadlock checking
- Timelock checking
- Reachability and other queries in temporal logic

Verification of
- Functional requirements
- Real-time requirements
- Security requirements
- Safety requirements
Compositional approach

- Framework based on a compositional approach
- System properties are inferred by component properties

Advantages:
- Efficient reasoning
- Delegate proof of application components to the provider
- Focus on the verification of the architecture

Formalized assumptions: components’ expectations on their environment
- Assumptions must be satisfied by the environment
Starlight example (architecture)
Starlight example (verification)

- The system provides some service to the user
  - The user issues commands that are processed by H or L
- Functional requirement: the system returns the correct result
- Commands labeled with high and low security levels
  - The user must switch the system to high before issuing a high command
- Security requirement: the low component must not receive high commands
- Safety requirement: the system satisfy functional and security requirements even if some subcomponents fail
- System requirements guaranteed by the properties of the subcomponents
Requirements and properties

- Functional requirements:
  - Invariants
  - Temporal logic

- Real-time and hybrid requirements
  - Functional requirements with timing constraints and taking into account models of physical components

- Security requirements
  - Requirements implementing security functions
  - Non-interference

- Safety requirements
  - Requirements related to safety
  - Modeled and verified taking into account failures
Annotation language

- Used to formalize requirements and specify verification tasks

- Annotations are interpreted by the specific tool
  - Tool’s specification syntax with references to the MILS-AADL model
  - Example:
    
    ```
    {OCRA: CONTRACT st
        assume: always ({secret(cmd)} implies
            ((not {switch_to_low} since{switch_to_high})));
        guarantee: never {secret(low_cmd)};
    }
    ```

- Possibility to connect to other tools (e.g., crypto protocol verification)
Verification issues

- MILS-AADL models have infinite-domain data variables, continuous-time semantics, with safety and security concerns
- Model checking of reachability for infinite-state systems is a hard problem
- Temporal logic even harder
- Safety and security properties harder and harder
- Major problem of model checking in general: scalability
Semantics of MILS-AADL models is a transition system

States given by component modes and assignment to data variables

Data types include integer and real

Parameters may include undefined functions (e.g., “computation(data)” or “is_secret(data)”)
New technique (Bradley 2012) to prove invariants automatically finding a suitable inductive invariant.

Currently recognized as the most effective model checking algorithm.

Build an inductive invariant $F$ such that $F \models P$

Trace of formulas $F\downarrow 0 = I, F\downarrow 1, ..., F\downarrow k$ such that:
- $F\downarrow i+1 \subseteq F\downarrow i$ ($F\downarrow i \models F\downarrow i+1$)
- $F\downarrow i \wedge T \models F\downarrow i+1$
- $F\downarrow i \models P$

Eventually either counterexample is found or $F\downarrow i \equiv F\downarrow i + 1$ proving $P$

Mixture of inductive reasoning and search-based techniques
IC3 + implicit abstraction

- Integrated with predicate abstraction
- Only the evolution of a set of predicates is tracked in the abstraction, the rest is abstracted away
- Implicit abstraction does not compute the abstract state space
- Definition of predicates embedded in the transition relation
- Abstraction refinement is fully incremental
  - Can keep previous trace $F\downarrow 1, ... , F\downarrow k$
  - Abstract transition relation strengthened by additional predicates
- Implemented in nuXmv
Temporal logic

- Many requirements formalized into temporal logic (e.g. LTL)
- No effective procedure to verify LTL over infinite-state systems
- Standard automata-based approach to $M \vDash \phi$:
  - Reduction to check that a certain condition $f$ can be visited finitely many times
- K-Liveness (Classen & Sorenssson 2012):
  - Key idea: check if $f$ can be visited at most $k$ times for increasing value of $k$
  - Reduced to invariant checking
  - Very efficient for finite-state systems
  - Integrated with IC3 for an incremental check of different $k$
- Implemented in nuXmv
  - Combined with IC3IA for verification of infinite-state systems
K-liveness for timed/hybrid models

- Problem for parametric and real-time/hybrid systems
  - The number of visits of $f$ can depend on parameters
  - $f$ can be visited an arbitrary number of times in a finite amount of time (related to Zeno paths)

- K-Zeno: check if there is a bound on the number of times the fairness is visited along a diverging sequence of time points

- Essential point: use an additional transition system $Z \downarrow \beta$ to force a minimum distance $\beta$ between two fair time points

- Note: $\beta$ is a symbolic expression over parameters and variables.

- Key contribution: define $\beta$ so that, if $M \models \phi$, then there exists $k$ such that $f$ can be visited at most $k$ times.

- Implemented in nuXmv and integrated in HyCOMP for the verification of hybrid systems
Contract-based reasoning

- Assumptions and guarantees expressed in temporal logic
- Refinement proved generating a set of proof obligations in temporal logic
- Proof obligations discharged with k-liveness/k-zeno
- Implemented in OCRA
Automatic generation of invariants

- Previous method requires a manual definition of the decomposition
- Other methods generate components’ properties automatically
- Application for timed systems and timed properties
- Observation:
  - invariant generation methods ignore time synchronization
  - invariants generated on timed models are too weak
- New approach
  - strengthening the invariants by exploiting time properties
  - augment atomic components with additional history clocks
  - generate local invariants for extended components
  - infer additional history clock constraints from interactions
- Method implemented and experimented on classical benchmarks
  - D-Finder prototype for Real-Time BIP
  - additional heuristics to improve scalability
Secure-BIP

- An extension of the BIP component framework with Information Flow Security

- Secure-BIP = BIP + security annotations
  - security labels on ports and variables
  - track information flow of interactions and data

- Two notions of non-interference studied:
  - event non-interference wrt interaction flow
  - data non-interference wrt data flow

- Static verification of non-interference
  - based on sufficient syntactic conditions
  - implemented in the Secure-BIP tool
Tool support for algorithms

- OCRA/nuXmv covers:
  - Invariants
  - LTL
  - LTL with real-time constraints
  - LTL for hybrid systems

- BIP covers
  - Deadlock
  - Transitive Non-interference

- Intransitive non-interference will be structurally guaranteed by the MILS-AADL model.

- Safety addressed with
  - COMPASS by model extension and applying above compositional methods on the extended models
  - XSAP for fault tree analysis
Conclusions

- Verification framework based on formal methods
- Focused on analysis of architecture
- Main concerns: automation, efficiency, representation of requirements
- Compositional approach formalizing assumptions and guarantees of components
- Model-based approach, i.e. same model for analysis, for platform configuration, for assurance case
- Evidence of architecture correctness combined with arguments on the platform in the assurance case