Introduction

Problem: security and reliability

Purpose: design and implementation of safe/secure systems
- Help system designers to describe their requirements
- Ensure safety and security policies enforcement
- Make it simple, quickly and easily usable

Involved subjects in this presentation
- Security and safety in real-time embedded systems
- Architectural description, AADL
- Partitioned architectures
Contents

- Safety and security requirements: what and why?
- Existing approaches: benefits and limits
- AADL: how to model and verify safety/security concerns
Contents

- Safety and security requirements: what and why?
- Existing approaches: benefits and limits
- AADL: how to model and verify safety/security concerns
Security: what and why?

- Avoid data sharing at several security levels
  - E.g: top-secret data not accessible to an unevaluated entity

- Main actors: objects and subjects
  - Objects represent data evaluated at a security level (secret, …)
  - Subjects perform operations (read/write) on objects

- Security policy: set of rules
  - Allowed operations according to a security clearance
  - E.g: Bell-Lapadula, Biba, Chinese Wall, …
Safety: what and why?

Safety properties and requirements
- Timing, communication, data types, …
- Depend on your system (domain, constraints, …)
- Enforced across the system

Enforce safety
- Fault handling (e.g: exceptions)
- Fault-containment (e.g: limit fault propagations)
- Identify and handle failures
- Reduce their impact locally, avoid their propagation globally
Contents

- Safety and security requirements: what and why?
- Existing approaches: benefits and limits
- AADL: how to model and verify safety/security concerns
Technics for safe and secure systems

- ARINC653 (avionics standard)
  - Safety goal, reduce faults propagation
  - No statement about security

- MILS (methodology)
  - Separated software components, depend on their security level

- Formal verification (design-time)
  - Model checking, arithmetic languages
  - Need to be related with implementation

- Programming languages
  - Verification at compile-time (e.g: SPARK/Ada, Eiffel)
  - Specific to a language
Interesting facts

Software isolation, fault-containment (ARINC653 & MILS)
- Avoid fault propagation across partitions
- Isolate software as more as possible

Classification of components (MILS)
- Single Level Security (SLS)
- Multiple Single Level Security (MSLS), isolation between flows
- Multiple Level Security (MLS), no isolation

Security policy enforcement (MILS)
- Reduce information leakage, security policy breakage
Security and safety in AADLv1

- Security levels and safety requirements as properties
  - Added to each component
  - New semantic tokens needed

- No dedicated component for partition
  - Process component with some properties, no specific runtime
  - Isolation concept not present

- Error model annex
  - Failures and fault-propagation modeling
What to expect from AADL version 2?

- **Partitioned architectures modeling**
  - Isolation across partition
  - Dedicated runtime for each partition

- **Provide facilities to model security & safety requirements**
  - Security and safety level(s) of each component
  - Make their representation more consistent

- **Security and safety policies enforcement**
  - Assembly of components don’t break requirements
  - Communication enforces safety/security policies
Contents

- Safety and security requirements: what and why?
- Existing approaches: benefits and limits
- AADL: how to model and verify safety/security concerns
New components in AADLv2

- **Virtual bus component**
  - Modeling protocol stacks, communication channels …

- **Virtual processor component**
  - Modeling processor cores, runtime part, …

- **Not “physical” components**
  - Describe components properties and requirements
  - Specify requirements of hardware or software entities
  - Affect related components
Virtual processor for partition handling

- Dedicated runtime for each partition
  - Subset of functionalities accessible only for one partition
  - Properties and requirements for each partition
  - Fault-containment for partition’s faults

- **Virtual processor + Process = partition!**
  - Space isolation modeled with process components
  - Partition’s runtime aspects modeled with virtual processor
  - Time isolation modeled with the main processor component

Partition’s address space
Partitioned/separation kernel
Partition’s runtime
Virtual bus as safety/security class

Virtual bus as a security layer
- Describe the requirements of a component/connection/port
- Use a component instead a bunch of properties

Common layer for safety & security
- Contains requirements and properties
- E.g: security timing requirements, ...
- Improve model's consistency

Hierarchical layers
- One layer can extend another

```
virtual bus secure
  properties
    Security_Level => Secret;
  end secure;

virtual bus implementation secure.classA
  properties
    Data_Domain => A;
  end secure.classA;

virtual processor partition
  properties
    Provided_Virtual_Bus_Class => (classifier (secure.classA));
  end partition;
```
Safety and security requirements analysis

- **Component’s classification**
  - Use components classification (MLS, SLS, …)
  - Detect illegal declarations and potential optimization

- **Fault-containment analysis**
  - Fault propagation across partitions

- **Security and safety policies enforcement**
  - Compatibility between security/safety classes
  - Check components aggregation, communications’ legality
  - Model consistency against partitioned architectures
Voice communication
- Pilot talks to passengers and crew
- Isolate communication flows

Several security layers
- Critical and unclassified

Partitioned architectures
- Separate security levels
- Runtime’s internals modeling

Partition (process + virtual processor)
Verification tools

- **OSATE plugins**
  - Check model against Bell-Lapadula security policy
  - Verify communication timing requirements
  - Verification patterns compliant with AADLv1

- **Ocarina and POK verification tools**
  - Ocarina, AADL toolsuite (see [http://aadl.enst.fr](http://aadl.enst.fr))
  - POK, partitioned kernel and middleware
  - AADLv2 compliance
  - Check models against Bell-Lapadula and Biba security policies
  - Analyze fault-containment across partitions
Towards code generation for partitioned systems

- **Code generation patterns for AADLv1**
  - Generate distributed, real-time and embedded-compliant code

- **New patterns for AADLv2**
  - Generate automatically safe and secure code!

- **Compliance with ARINC653/MILS**
  - Generate partition, configuration items, failure handler
  - Enforce model’s semantics
  - Reduce impact of application-code errors
Conclusion

- Objectives: ensure security and safety with AADL models
  - Take advantage of virtual processors and buses
    - Combine safety and security in a same class
    - Model partitioned systems
  - Verification of security/safety requirements
    - Error early detected, fixed at low cost
  - Design, verify and implement safe and secure system
    - Map AADL concepts to partitioned architectures