Architecture and Protocol Verification and Attack Analysis

Ludovic Apvrille

Institut Telecom, Telecom ParisTech
ludovic.apvrille@telecom-paristech.fr

July, 1, 2010

Outline

Outline

On-board Vehicle Systems

On-board vehicle system

- ECUs (Electronic Control Units) = set of hardware components
  - Execution elements (CPUs, HWAs)
  - Communication elements (e.g., busses)
  - Storage elements (e.g., RAM, flash)
  - I/O devices, including sensors / actuators
- Software components
  - Executed on CPUs

One of EVITA's goals:

Proving security properties on those systems

Proving Security Properties: Overall Methodology

Methodology

1. Requirement identification
2. Architecture specification
3. Specification of security-related protocols
4. Verification of security properties on the overall system (Architecture + protocols)
    - **Performance analysis**
    - **Attack analysis**

Objective of this demonstration

- Focus on the last stage (verification)

Proving Security Properties: Overall Methodology (Cont'd)

Performance evaluation

- Impact of security mechanisms on system performance

Attack analysis

- **Magnified view approach**
  - Proof of security properties on a subpart of the EVITA architecture (e.g., a given protocol).
- **Global composition approach**
  - Reuse of proofs performed on sub-elements to validate requirements over the full system
  - Next presentation

Issues

(1) Performance properties

- Impact of the EVITA security architecture on system performance?
  - Cryptographic algorithms and protocols
- Partitioning issue
  - Shall algorithms be software or hardware implemented? Distributed among ECUs or centralized in a given ECU

(2) Security properties

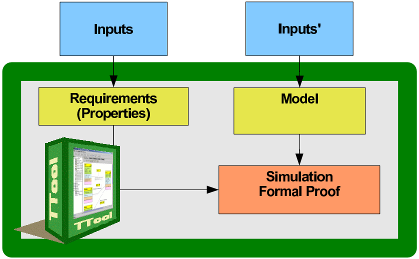
- Security requirements have been previously identified
- Derive attacks from requirements and ...
- Prove that those attacks are not possible in the EVITA infrastucture

Modeling and Verification Approach

Objective

- Performance evaluation, Attack analysis (magnified view approach)

- Consider inputs (e.g., EVITA deliverables)
- Make a model, using e.g. SysML and UML models
- Verify properties using simulation or formal verification techniques



Modeling and Verification Approach (Cont'd)

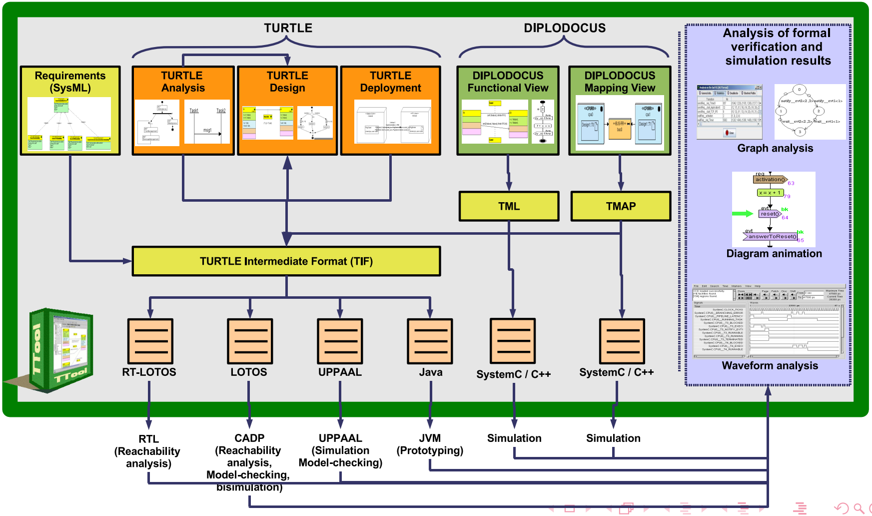
Analysis	(1) Performance analysis	(2) Attack analysis
Profile	DIPLODOCUS	TURTLE
Verification technique	Simulation	Formal verification (model-checking)
Focus of the model	Application complexity and architecture elements	Protocol description and basic architecture elements. Attacks modeling
Tools	TTool (edition, simulator)	TTool, CADP, UPPAAL

TELECOM ParisTech

TTool: Main Features

- Open-source UML toolkit
- Meant to support UML2 profiles
  - 8 UML profiles are currently supported
    - e.g., TURTLE, DIPLODOCUS
- Mostly programmed in Java
  - Editor, interfaces with external tools
  - Simulators are programmed in C++ or SystemC
- Formal verification and simulation features
  - Hides formal verification and simulation complexity to modelers
  - Relies on external tools
  - Press-button approach

TTool: TURTLE and DIPLODOCUS



TELECOM ParisTech

Outline

Introduction

Performance analysis

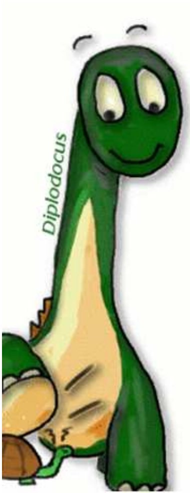
- DIPLODOCUS
- Case study: Active Brake

Attack analysis

Outlook

TELECOM ParisTech

DIPLODOCUS in a Nutshell



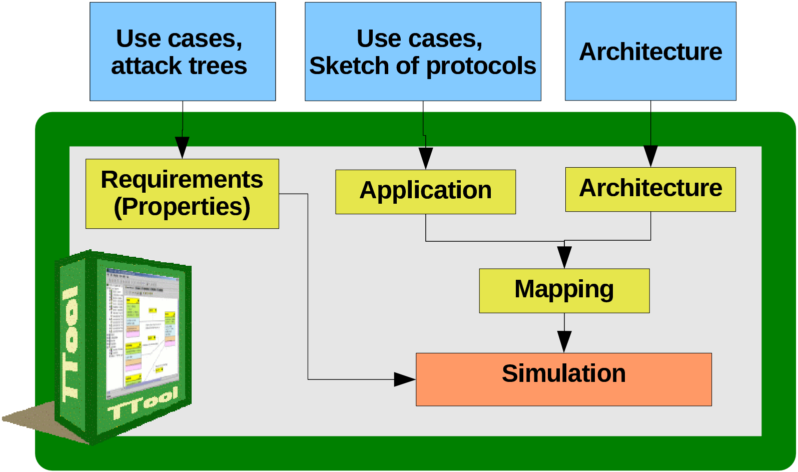
DIPLODOCUS = UML Profile

- System-level Design Space Exploration
- Y-Methodology
- MARTE compliant

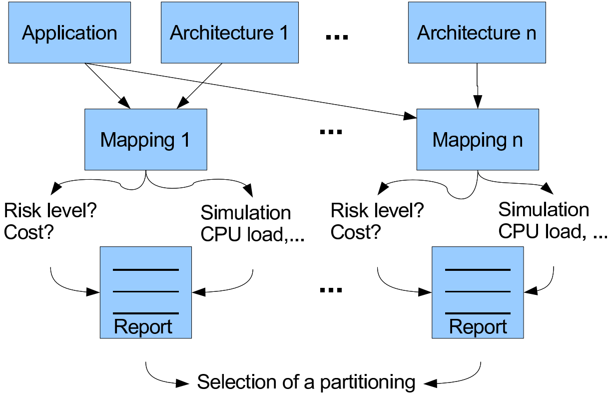
Main features

- Data are abstracted
- Formal semantics
- Very fast simulation support
- Fully supported by an open-source toolkit
  - TTool

DIPLODOCUS: Methodology for Performance Evaluation



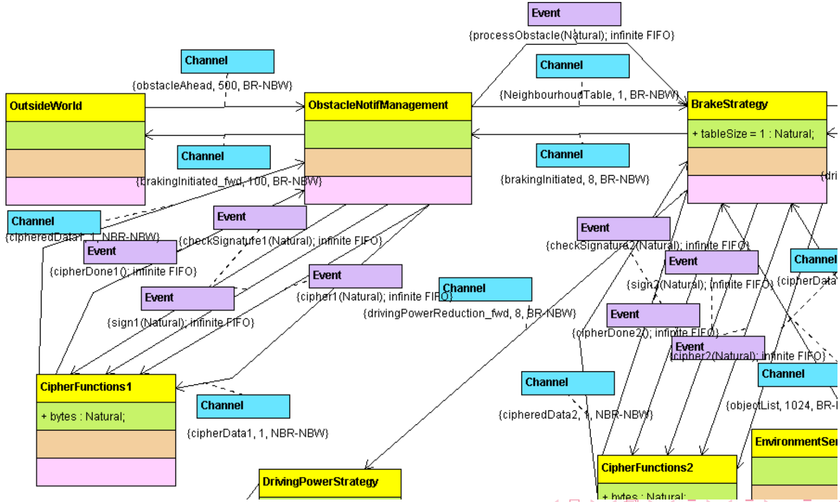
DIPLODOCUS: Methodology for Performance Evaluation (Cont'd)



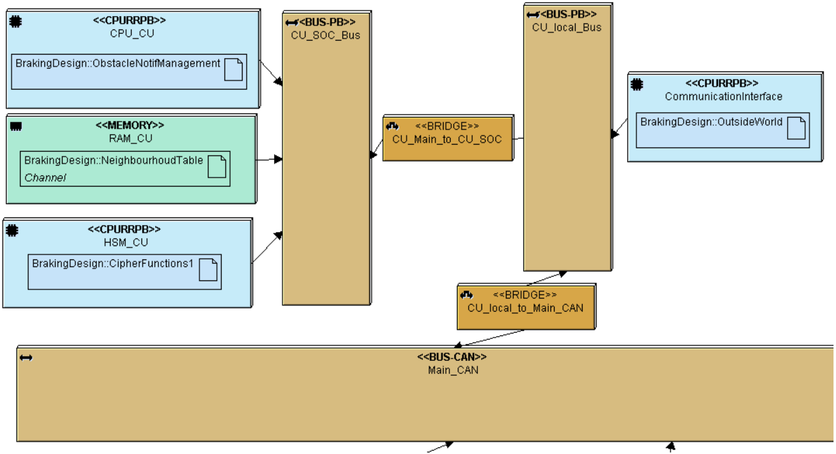
Description of the Active Brake Use Case

- Message sent from one car to another car (car2car)
  - Immediate danger of collision
  - Instant brake manoeuvre
- Message received and checked at Communication Unit level
- Plausibility check at Chassis Safety Controller level
  - If braking is the best solution, a brake order is sent to the brake control unit
    - Power Train Controller is also informed (to decelerate, etc.).
  - Braking information might be forwarded to other neighbour cars

Application Modeling



Architecture Modeling and Mapping



A Few Simulation Results

CPUs and Hardware Accelerators

CPU	Load	Contention delay
Load_Emulation	0.15711	29973
CPU_CU	0.11244	0
HSM_CU	0.11939	0
CPU_BCU	0.00010	6806
HSM_BCU	0.00004	0
CPU_PTC	0.00018	0
CPU_ChassisSensor	0.00035	200000
CPU_EnvSensor	0.01115	5818
HSM_CSC	0.11827	0
...	...	...

A Few Simulation Results (Cont'd)

Buses

Bus	Load
BCU_local_Bus	0.00017
CSC_local_Bus	0.56926
PTC_local_Bus	0.00026
CU_local_Bus	0.55783
CU_SOC_Bus	0.78811
Main_CAN	0.71469
CSC_SOC_bus	0.74216
...	...

Outline

TURTLE in a Nutshell

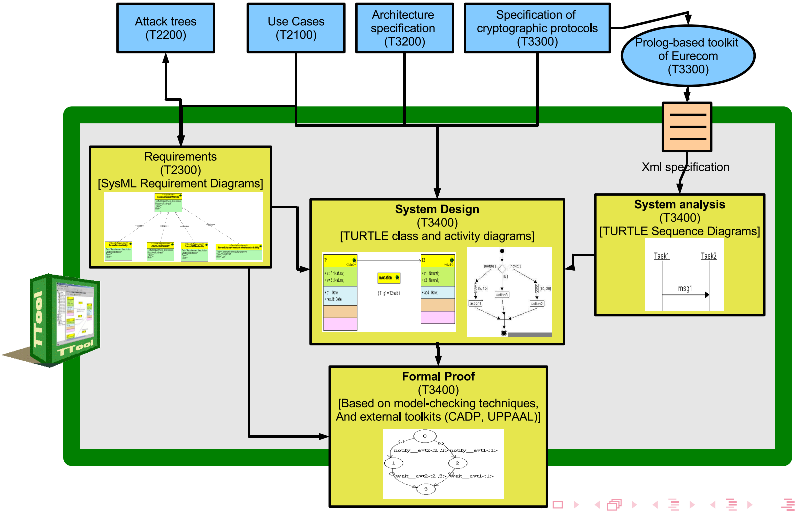
TURTLE = UML Profile

- Targets temporally constrained embedded systems
- Three sub-profiles: analysis, design, deployment
- Formal verification (and simulation)
- TURTLE Design = class diagram + a set activity diagrams

Main features

- Non deterministic operators
  - Choice, delays
- Fully supported by an open-source toolkit
  - TTool

TURTLE: Methodology for Attack Analysis



Model: Main Principles

Modeled elements

- Hardware elements in ECUs
  - HSM
  - Communication networks
- Software elements
  - Protocol stack at involved ECUs

Proving security properties

- Observer technique
- Model-checking is used to search for a given action

Description of the Case Study

Why this case study (not directly related to EVITA)?

- Illustrate proofs of security requirements with TURTLE
- A small yet representative system
- Contains all interesting concepts:
  - Entities, network elements, crypto functions and protocols, attacks

Description

- Alice and Bob, who want to exchange a confidential data
- Use the Needham-Schroeder protocol to setup a session key K, using a trusted server
- Then, Bob sends the data to Alice using K

The Needham-Schroeder Protocol

Description

A represents Alice, B Bob, S the Server; R_X is a random number generated by X, and K_{XY} a key used by X and Y to cipher / decipher information with a symmetric encryption algorithm

1. $A \rightarrow S : A, B, R_A$
2. $S \rightarrow A : \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
4. $B \rightarrow A : \{R_B\}_{K_{AB}}$
5. $A \rightarrow B : \{R_B - 1\}_{K_{AB}}$

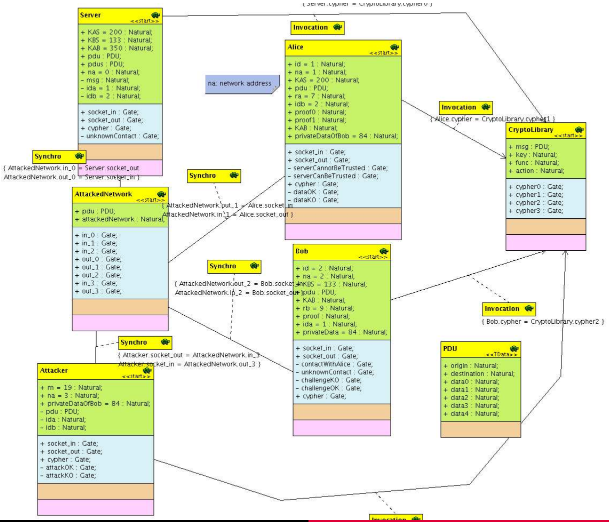
Requirement *req1*

The data sent by Bob to Alice shall be confidential.

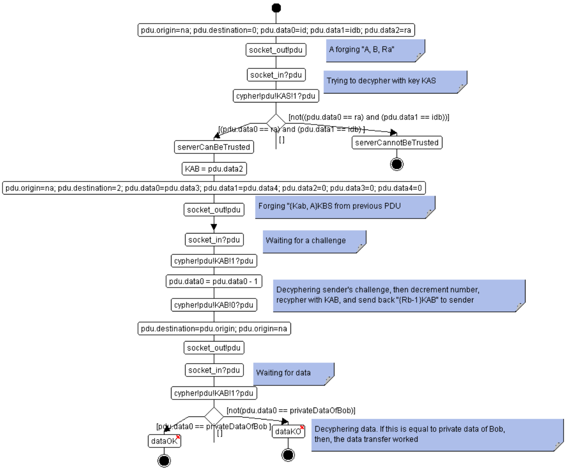
Attacks on the Needham-Schroeder Protocol

- Several known attacks against Needham-Schroeder
- Considered attack: *S. Gurgens et al., "Role based specification and security analysis of cryptographic protocols using asynchronous product automata", Database and Expert Systems Applications, Sept. 2002.*
  (Cx denotes an attacker pretending to be an entity x):
  1. $A \rightarrow C_S : A, B, R_A$
  2. $C_B \rightarrow S : B, A, R_C$
  3. $S \rightarrow C_B : \{R_C, A, K_{BA}, \{K_{BA}, B\}_{K_{AS}}\}_{K_{BS}}$
  4. $C_A \rightarrow B : \{R_C, A, K_{BA}, \{K_{BA}, B\}_{K_{AS}}\}_{K_{BS}}$
  5. $B \rightarrow C_A : \{R_B\}_{R_C}$
  6. $C_A \rightarrow B : \{R_B - 1\}_{R_C}$
- From that attack, *req1* can be proved as non-satisfied.

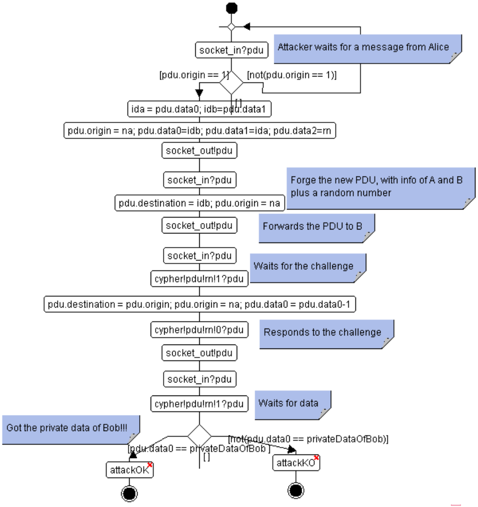
Class Diagram



Activity Diagram of Alice



Activity Diagram of Attacker

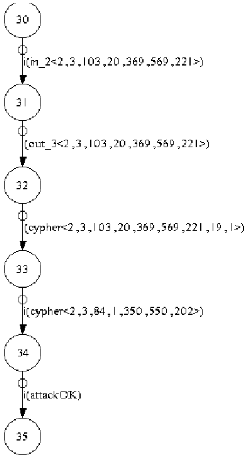


Formal Verification with CADP

Verification approach

- Generate a Reachability Graph using CADP
- Minimize of the reachability graph
- Search for traces containing the *attackOK* and *attackKO* actions

Reachability graph



Formal Verification with UPPAAL

Verification approach

- Select actions of interest on the UML model
- Automatically invoke UPPAAL
- Search the accessibility and liveness of selected actions

Network can be probed

Reachability of: Action state (attackKO)
-> property is NOT satisfied

Reachability of: Action state (attackOK)
-> property is satisfied

Reachability of: Action state (dataKO)
-> property is NOT satisfied

Reachability of: Action state (dataOK)
-> property is satisfied

Liveness of: Action state (attackKO)
-> property is NOT satisfied

Liveness of: Action state (attackOK)
-> property is NOT satisfied

Liveness of: Action state (dataKO)
-> property is NOT satisfied

Liveness of: Action state (dataOK)
-> property is NOT satisfied

Formal Verification with UPPAAL (Cont.)

Network cannot be probed

Reachability of: Action state (attackKO)
-> property is NOT satisfied

Reachability of: Action state (attackOK)
-> property is NOT satisfied

Reachability of: Action state (dataKO)
-> property is NOT satisfied

Reachability of: Action state (dataOK)
-> property is satisfied

Liveness of: Action state (attackKO)
-> property is NOT satisfied

Liveness of: Action state (attackOK)
-> property is NOT satisfied

Liveness of: Action state (dataKO)
-> property is NOT satisfied

Liveness of: Action state (dataOK)
-> property is satisfied

Network is always probed

Reachability of: Action state (attackKO)
-> property is NOT satisfied

Reachability of: Action state (attackOK)
-> property is satisfied

Reachability of: Action state (dataKO)
-> property is NOT satisfied

Reachability of: Action state (dataOK)
-> property is NOT satisfied

Liveness of: Action state (attackKO)
-> property is NOT satisfied

Liveness of: Action state (attackOK)
-> property is satisfied

Liveness of: Action state (dataKO)
-> property is NOT satisfied

Liveness of: Action state (dataOK)
-> property is NOT satisfied

Outline

Results

Fully integrated environment for the design and verification of embedded systems

- Based on UML / SysML, open-source toolkit (TTool)
- Formal proof can address
  - Safety and security properties
    - Proofs achieved on authenticity, confidentiality, freshness
  - Functional and non functional properties

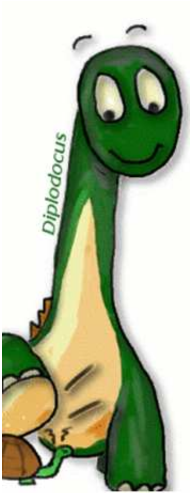
Recall on methodological stages

- Requirement capture (SysML, DIPLODOCUS)
  - Attack trees, definition and organization of requirements
- Performance analysis (DIPLODOCUS)
- Attack analysis, magnified view approach (TURTLE)

A Few Industrial Case Studies with TTool

- MPEG coders and decoders (Texas Instruments)
- LTE (Freescale)
- Partitioning in vehicle embedded systems, formal proof of security properties (EVITA project)
- Many other systems!

To Go Further ...



TTool

- Type *TTool UML* under google
- And click on the *I am lucky* button!

DIPLODOCUS, TURTLE

- *DIPLODOCUS UML*
- *TURTLE UML*