Extending AADL for Security Design Assurance of the Internet of Things

Presented by Rick Kazman, PhD

Team: Carol Woody (PI),
Rick Kazman, Robert Ellison,
John Hudak, Allen Householder

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

DM-0002834

Agenda

Security Modeling and AADL

IoT Case Study

Conclusions

Agenda

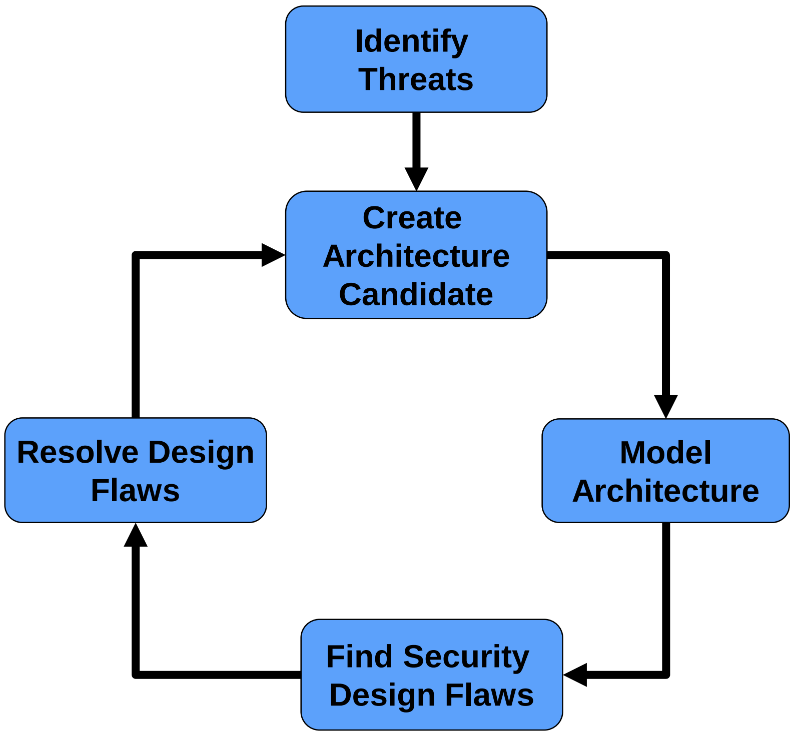
- ✓ **Security Modeling and AADL**
- **IoT Case Study**
- **Conclusions**

Value of Security Formal Modeling

Crucial security decisions to address threats are made in the architecture.

Analyzing an architecture for is a huge opportunity for improving security.

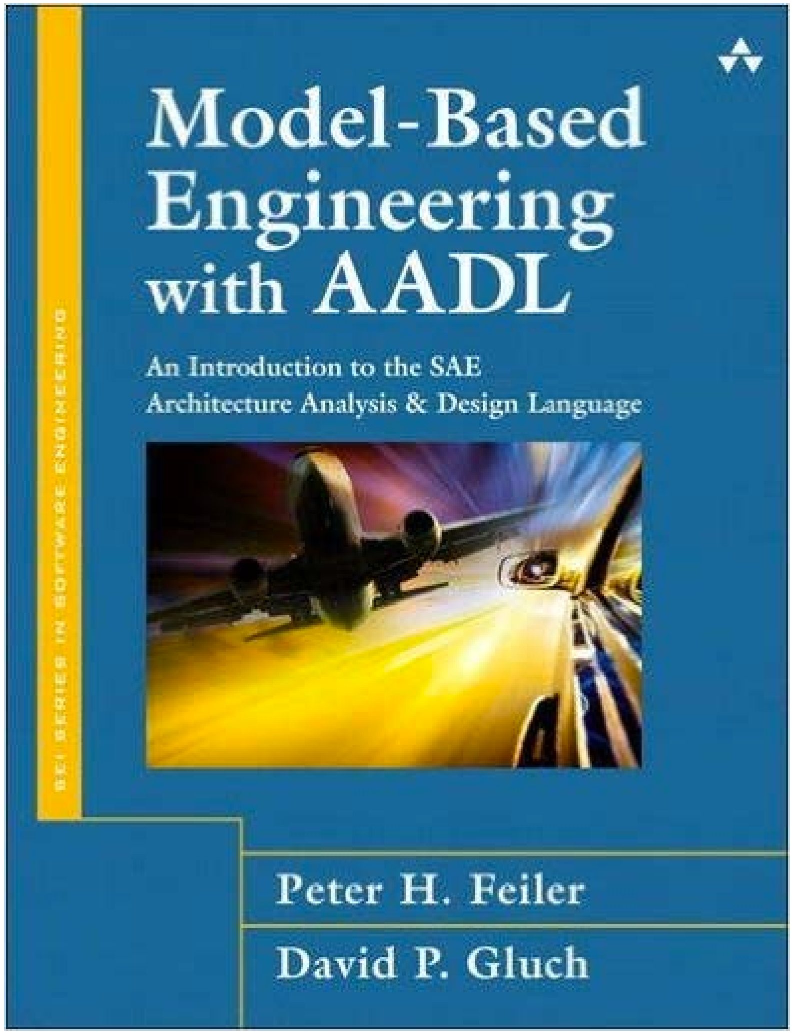
Formal modeling provides a means to verify a design.



Using AADL to Model Architectures

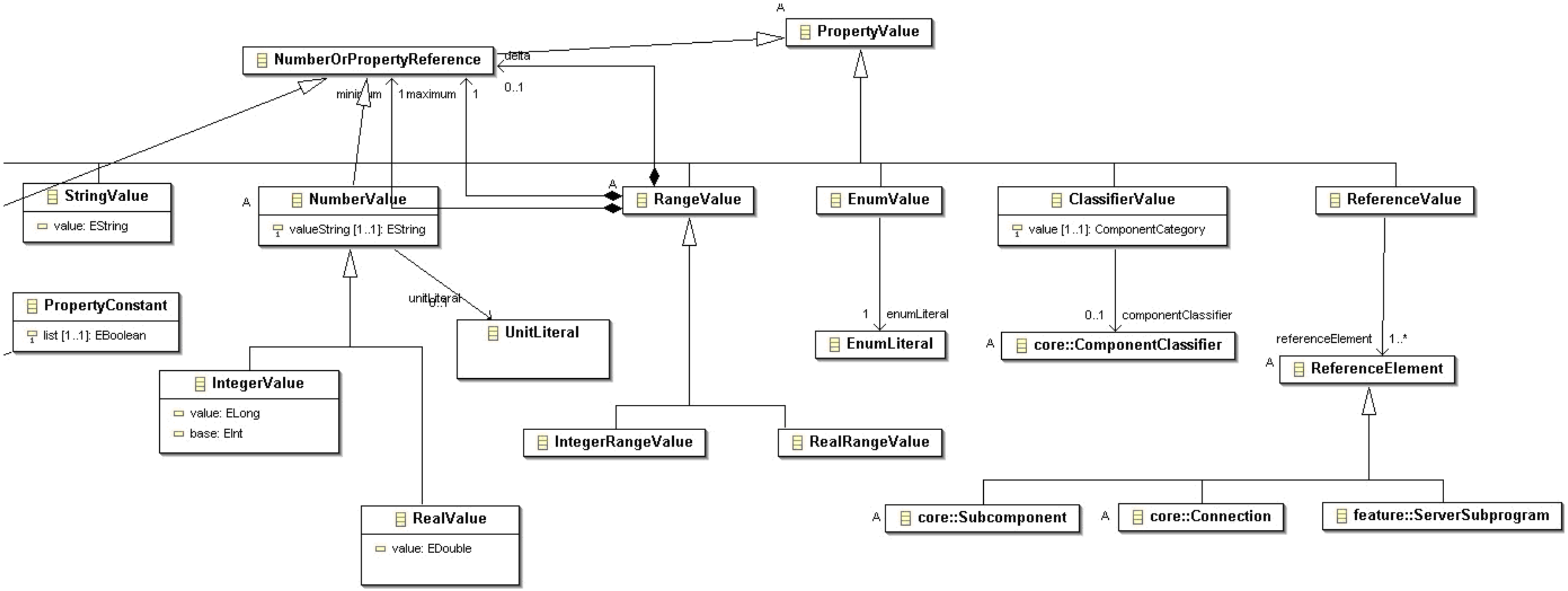
AADL (Architecture Analysis and Design Language):

- specifies a static representation of a system architecture
- can model logical flows, binding of software to hardware
- is strongly typed, allowing a consistency checks via a modeling tool set.



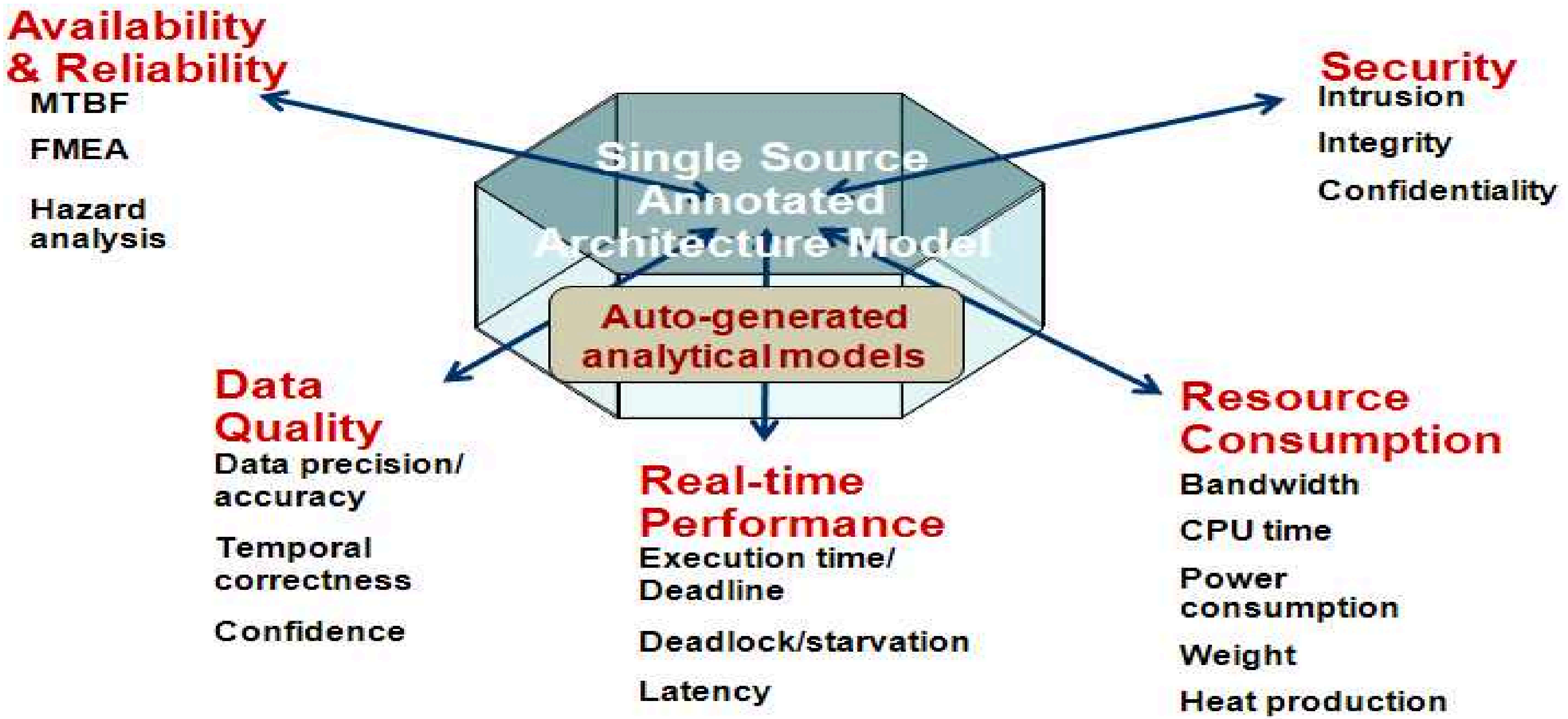
AADL for Modeling Security Properties

AADL has *properties* to capture component characteristics.



Example: a component that performs encryption can have one property for latency (e.g. 3 ms.), and another for type of encryption, e.g. RSA.

AADL for Modeling Security Properties



AADL supports the addition of modeling formalisms beyond the capabilities of the core language.

Agenda

Security Modeling and AADL

✓ IoT Case Study

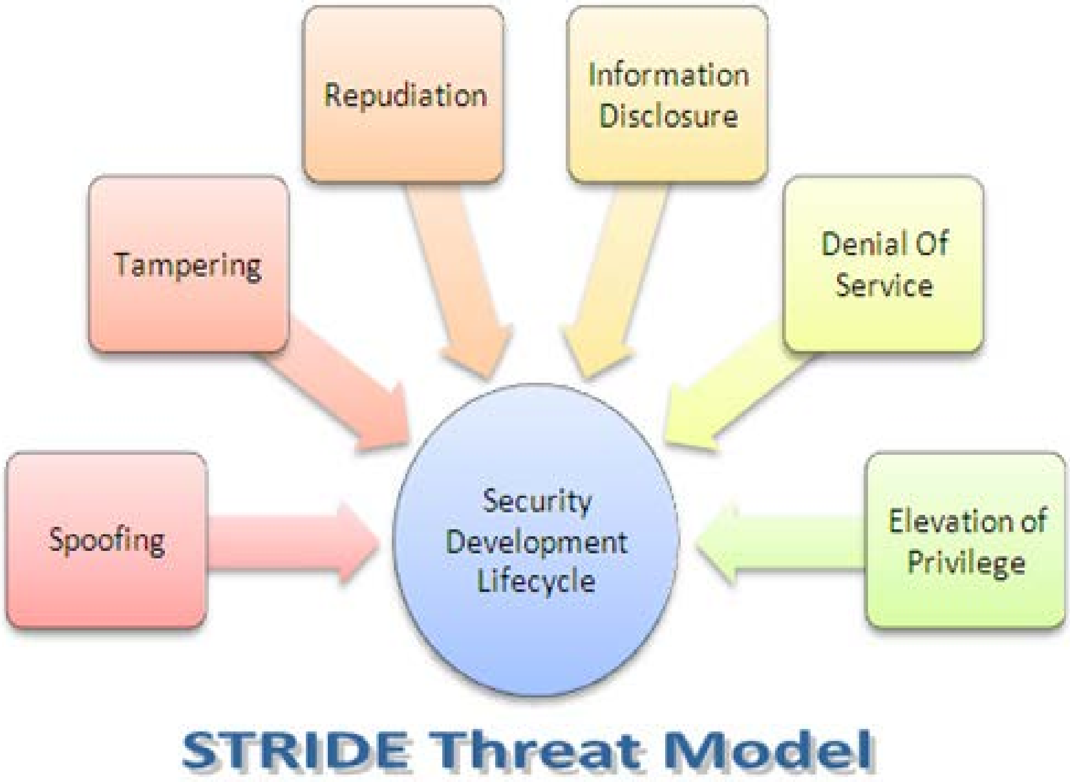
Conclusions

Extending AADL for Security

Our goal was to extend AADL to better address security in the architecture.

To drive this analysis we began by specifying threats in the context of a real-world IoT (internet of things) problem: *automotive electronics*.

Using STRIDE to Determine Threats



We employed the STRIDE model.

STRIDE is a way to walk through categories of threats and determine their potential consequences.

Using STRIDE to Determine Threats

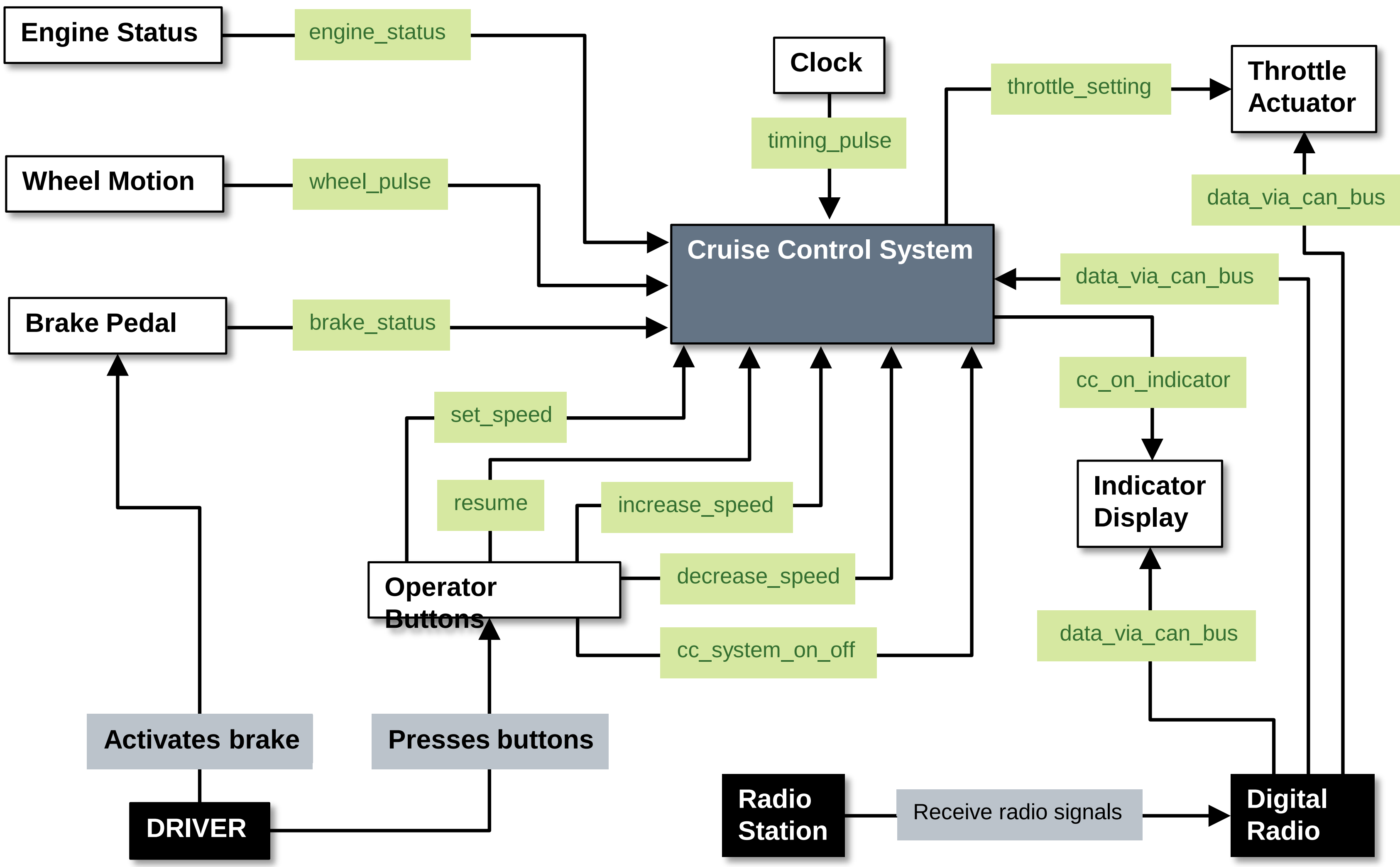
Element	Interactors	Data flows	Data stores	Processes-Software
Spoofing identify	●			●
Tampering with data		●	●	●
Repudiation	●			●
Information disclosure		●	●	●
Denial of service		●	●	●
Elevation of privilege				●

Using STRIDE to Determine Threats

We instantiated STRIDE in the context of the automotive electronics example. For each category we described:

1. how the threat might be realized
2. the implications of this threat on the automotive example
3. the risks that arose from this threat analysis
4. questions about the architecture that drive the design
5. the security property violated

Modeling the Automotive Electronics System



Modeling STRIDE Attack Types in AADL

We did an in-depth AADL-based analysis of several attack types from the STRIDE model.

Consider *Information Disclosure*.

To guard against inadvertent information disclosure you need to authenticate and authorize components.

This establishes who can access what data, with what rights.

Modeling Information Disclosure in AADL

We made the following architectural design decisions:

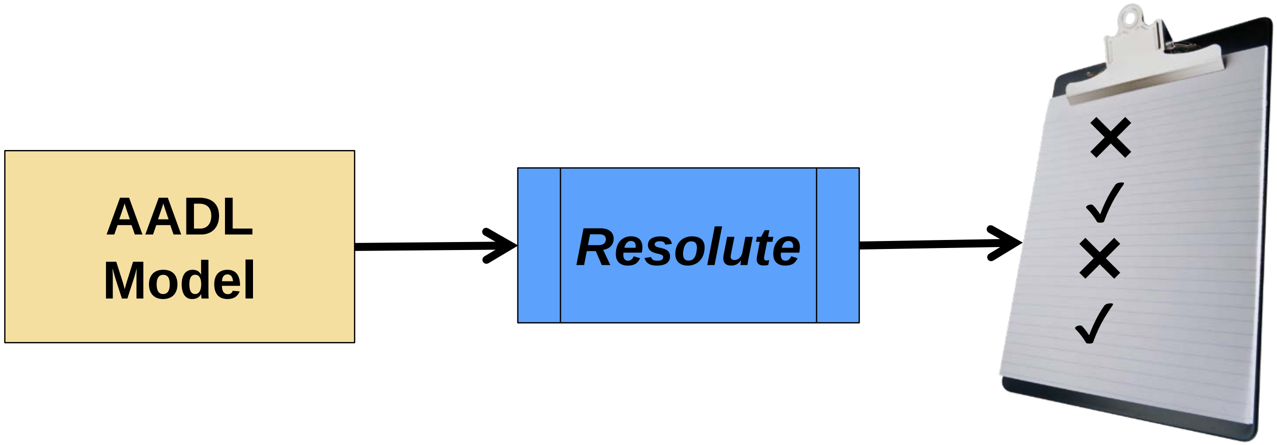
- For component A to communicate data to component B, they must be of the same *group*. To support this, we define a property ***AccessGroup***: a list of groups.
- In addition we ensure that A and B have the proper *access rights* for the data being communicated.
- We created a user-defined AADL property, ***AccessMode,*** that has one the following values: **r** for read access, **w** for write access, **rw** for read-write access, and **x** for execute.
- Finally we create a compound rule that checks ***AccessGroup*** and ***AccessMode*** properties of each component to ensure they are consistent.

Analyzing Security Properties

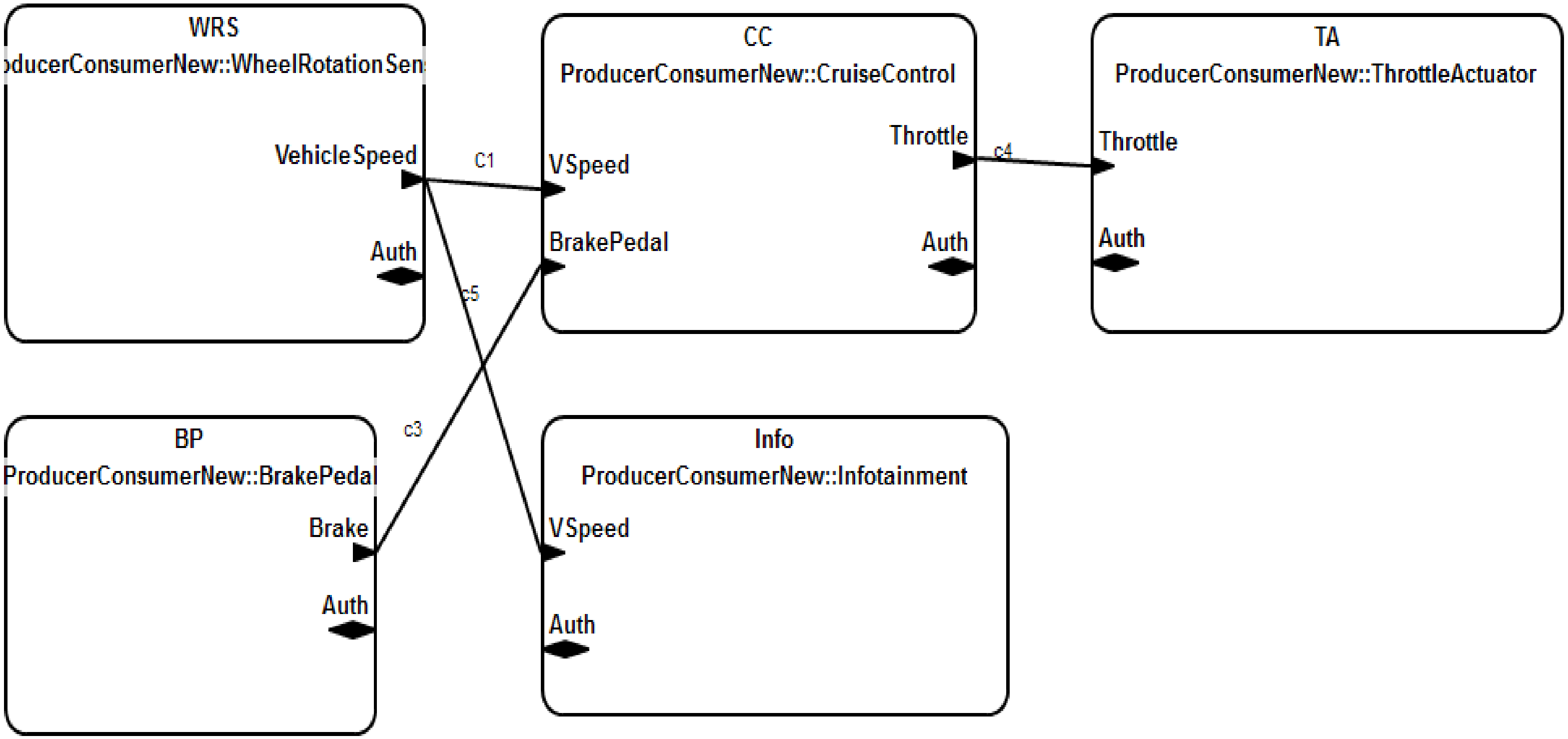
To reason about the satisfaction of security properties in a system architecture we need to:

1. Specify the properties, and the architectural elements they are associated with in AADL; and then
2. Analyze claims over those properties.

To analyze claims we use the *Resolute* model-checker.



Model Problem Specified in AADL



A subset of automotive electronics includes sensors (WheelRotation, BrakePedal), actuators (ThrottleActuator), and control application software (Cruise Control and Infotainment).

Semantics of Access Privileges

For example, a vehicle speed sensor can *write* to a global data area from which multiple control systems of a vehicle can *read* the vehicle speed.

The data would be annotated with an *AccessMode* of **r** and a list of subsystems that are allowed to read that data, for example, the cruise control system.

A malicious application could attempt to write to the vehicle speed and overwrite the true speed value with a lower value, causing the cruise control to rapidly accelerate the vehicle.

Semantics of Access Privileges

Thus we need to specify that only certain applications can write speed data, and that only certain applications can read speed data.

To accomplish this, we check the *AccessGroup* that contains the list of components that can access the data.

Semantics of Access Privileges

We formally specify this in AADL as follows:

```
property set Security_Trust is
    AccessMode: enumeration (r, w, rw, x) applies to (all);
    AccessGroup: enumeration (CC, ABS) applies to (all);
end Security_Trust;
```

Semantics of Access Privileges

An architect would annotate a data type with properties as follows:

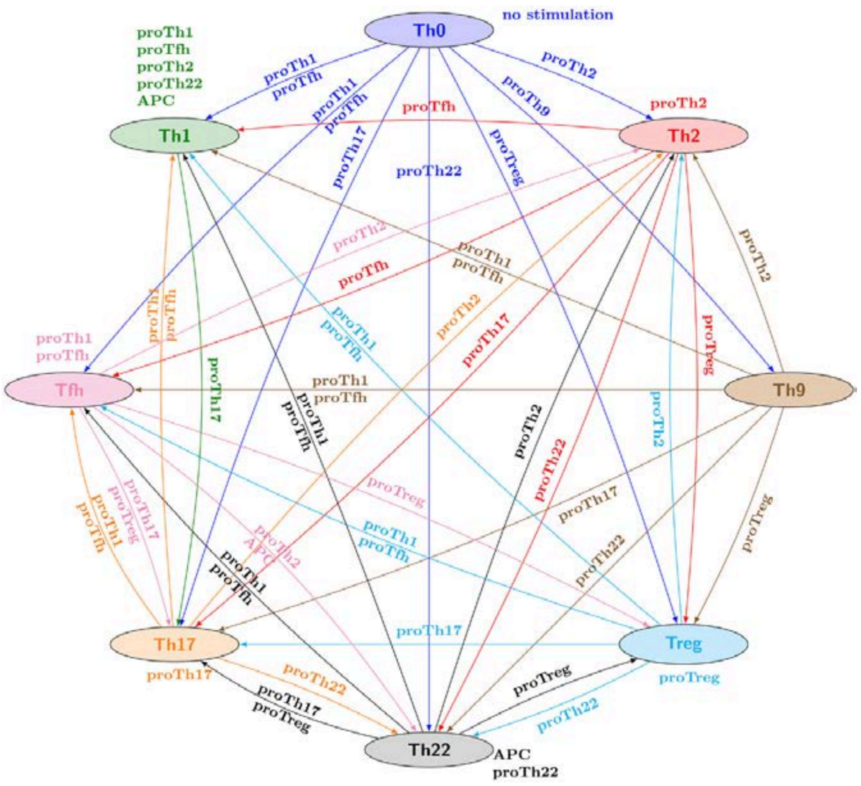
```
data VehicleSpeed
properties
    Security_Trust::AccessMode => r;
    Security_Trust::AccessGroup => CC;
end VehicleSpeed;

data ThrottlePosition
properties
    Security_Trust::AccessMode => w;
    Security_Trust::AccessGroup => CC;
end ThrottlePosition;
```

Running the Resolute Checker

Running the *Resolute* model checker requires that the model be instantiated.

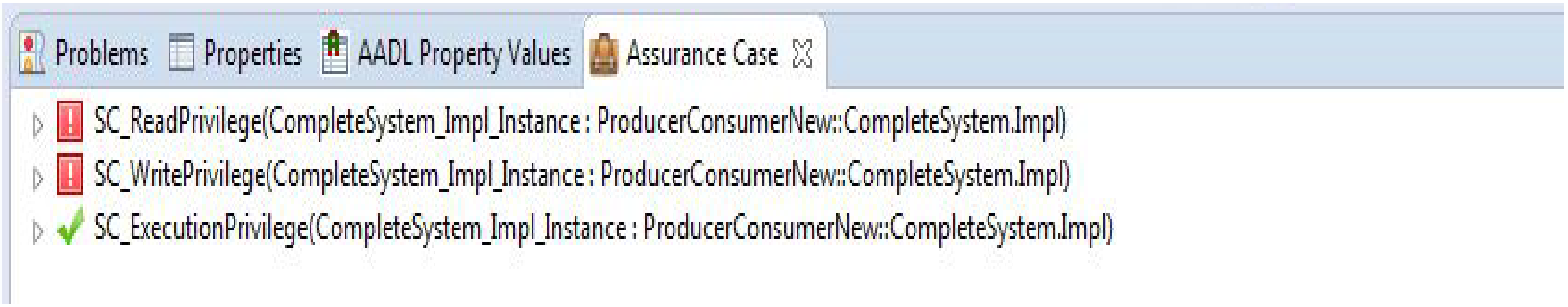
Resolute walks the instantiated model hierarchy looking for components specified in its claims and checks those components according to the logic encoded in the claim.



Running the Resolute Checker

For example we want to ensure that data being read by the cruise control has AccessMode properties properly specified.

We also run the command execution check on incoming data to see if data is properly specified as executable.



The read and write privilege checks both failed but the execution check passed.

The *VehicleSpeed* data *AccessMode* property was set to **w** on the incoming port and for the *ThrottlePosition AccessMode* property was set to **r** causing both checks to fail.

Agenda

Security Modeling and AADL

IoT Case Study

✓ Conclusions

Limitations

An AADL model does not necessarily support security *validation*. Is the system sufficiently secure for the planned usage?

- A weakness *external to the model* enables an attacker to operate outside the model. Example: A buffer overflow vulnerability or successful phishing can enable an attacker to obtain privileges without authentication.
- Specifications may be correct but not provide the desired level of security assurance. Example: An architect claims that a component sufficiently verifies input data to eliminate the risk of a SQL-injection. But CWE 89 recommends alternative mitigations with a higher level of assurance than input verification, such as component output verification.

Conclusions

Architectural security modeling has several benefits:

- guides an architect to reason and design in terms of system-wide security properties
- provides a framework for checking that such properties are maintained during system maintenance and evolution

But it has limitations; it should be considered *part of* a complete approach to security.

This research has also resulted in many collaborations:

- Prof. Mel Rosso-Llopart (CMU/IRS) / Malware Analysis
- Prof. Jungwoo Ryoo (Penn State) / Architecture Analysis for Security (AAFS) method
- Prof. Yuanfang Cai (Drexel U.) / Titan architecture analysis tool


Thank You