# Reusable formal models for secure software architectures

Thomas Heyman, Riccardo Scandariato, Wouter Joosen

IBBT-DistriNet, KU Leuven

# Problem

# Formal methods

## for secure software architectures

Enforce rigor

Enable reasoning

Provide assurance

$\longrightarrow$ interesting for SA

# Formal methods

## not widely used...

High overhead

Require **expertise**

Different **stakeholders**

# Solution

# Contribution – part I
## for the **security engineer**


# **Refined models**

of building blocks (e.g., security patterns)

**Created** and **used** by security engineer

→ Assess security

Results in better **documentation**

Verification results are **reusable**

# Contribution – part II

for the **software architect**

## **Abstract models**

of building blocks (e.g., security patterns)

Simple, behaves like refinement

**Created** by security engineer

**Used** by software architect

→ Uncover compositional issues

**(Re-)usable!**

# Outline

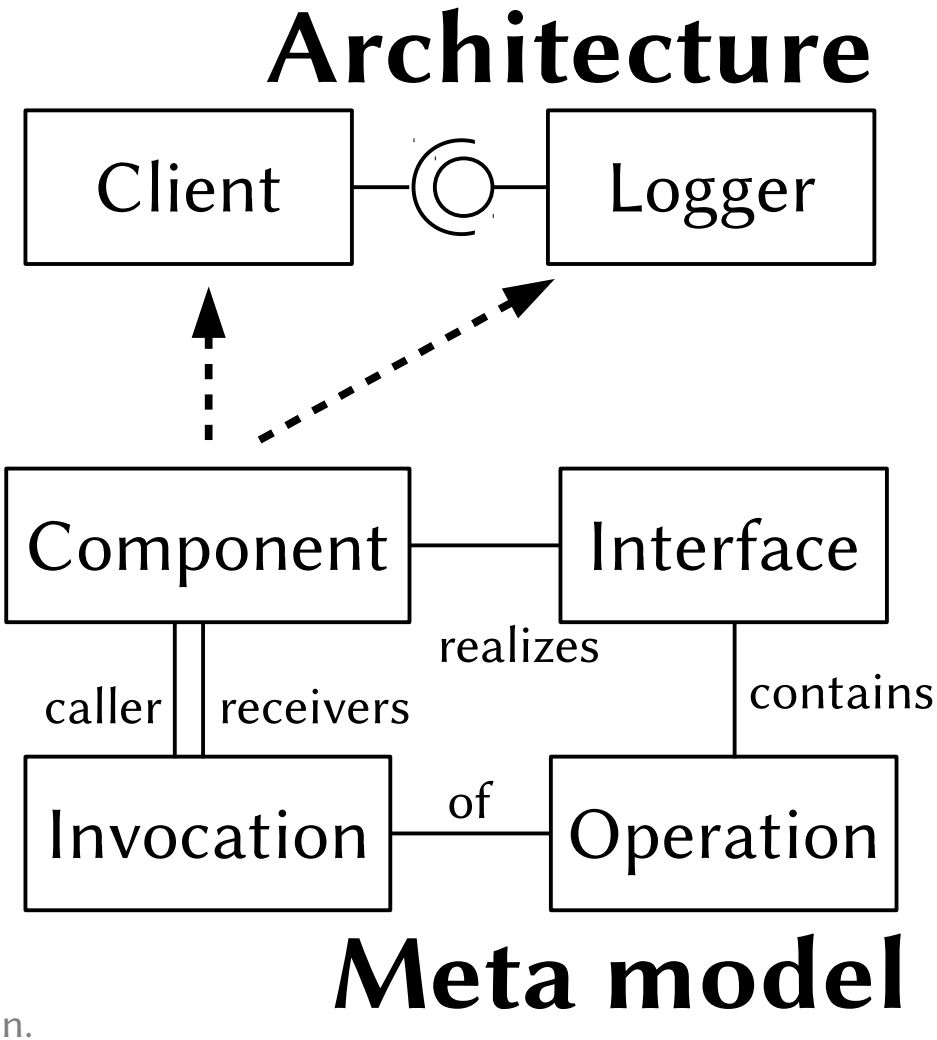
Background
Contribution I
Contribution II
Wrap-up

# Background

## modelling software architectures

**Alloy**

```
sig Message {}

sig Logger in Component {
    contents: Message set → Time
}

sig Log in Operation {}

sig Logger.log(m:Message,t:Time) {
    m in this.contents.t
}

sig Client in Component {}

...
```




Thomas Heyman, Riccardo Scandariato, and Wouter Joosen.
*Security in context: analysis and refinement of software architectures.*
In Annual IEEE Computer Software and Applications Conference, July 2010.


# Modelling a pattern language for accountability

# A pattern language
## for accountability

## Contents

Secure Logger, Audit Interceptor, Authentic. and Authoriz. Enforcer, Secure Pipe

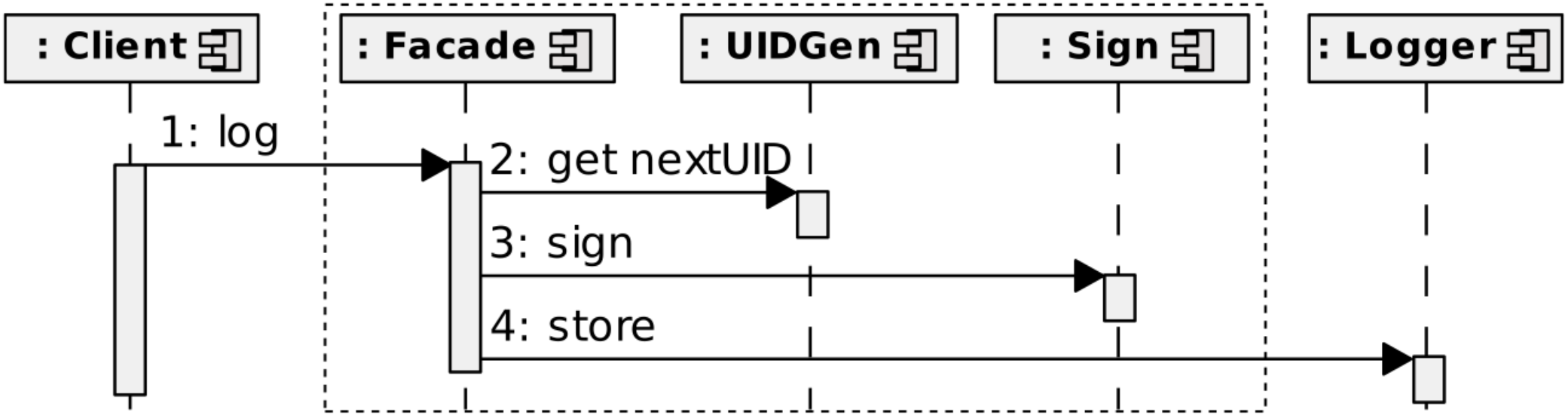
Christopher Steel, Ramesh Nagappan, and Ray Lai.
*Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management.*
Prentice Hall, 2005.

## Motivation

Self-contained set

Useful in practice (industrial projects)

# Modelling
## the Secure Logger pattern




Christopher Steel, Ramesh Nagappan, and Ray Lai.
*Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management.*
Prentice Hall, 2005.

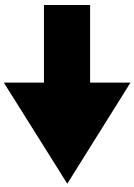
# Modelling
## the Secure Logger pattern


```
sig SignedMessage {
        content: ProcessedMessage one -> Time,
        signedContent: ProcessedMessage one -> Time,
        signedBy: Principal one -> Time
}

sig Logger in Component {
        contains: set SignedMessage -> Time,
        nextUID: Int one -> Time
}{
        all t:Time,c:Component,m:Message,i:Int {
                Execute[c,this,Log,m,t] => some t1:Time | this.log[m,t,t1]
                Execute[c,this,Read,m+i,t] => this.read[m,i,t]
                Execute[c,this,Verify,i,t] => this.verify[i,t]
        }
}

pred Logger.log(m:Message, t:Time) {
        some pm:ProcessedMessage, s:SignedMessage {
                pm.content.t = m
                0 <= pm.id.t
                pm.id.t < calculateNextUID[this,t]
                s.content.t = pm
                s.sign[LoggerEntity,t]
                s in this.contains.t
        }
```

# Verification

## encoding sec. requirements

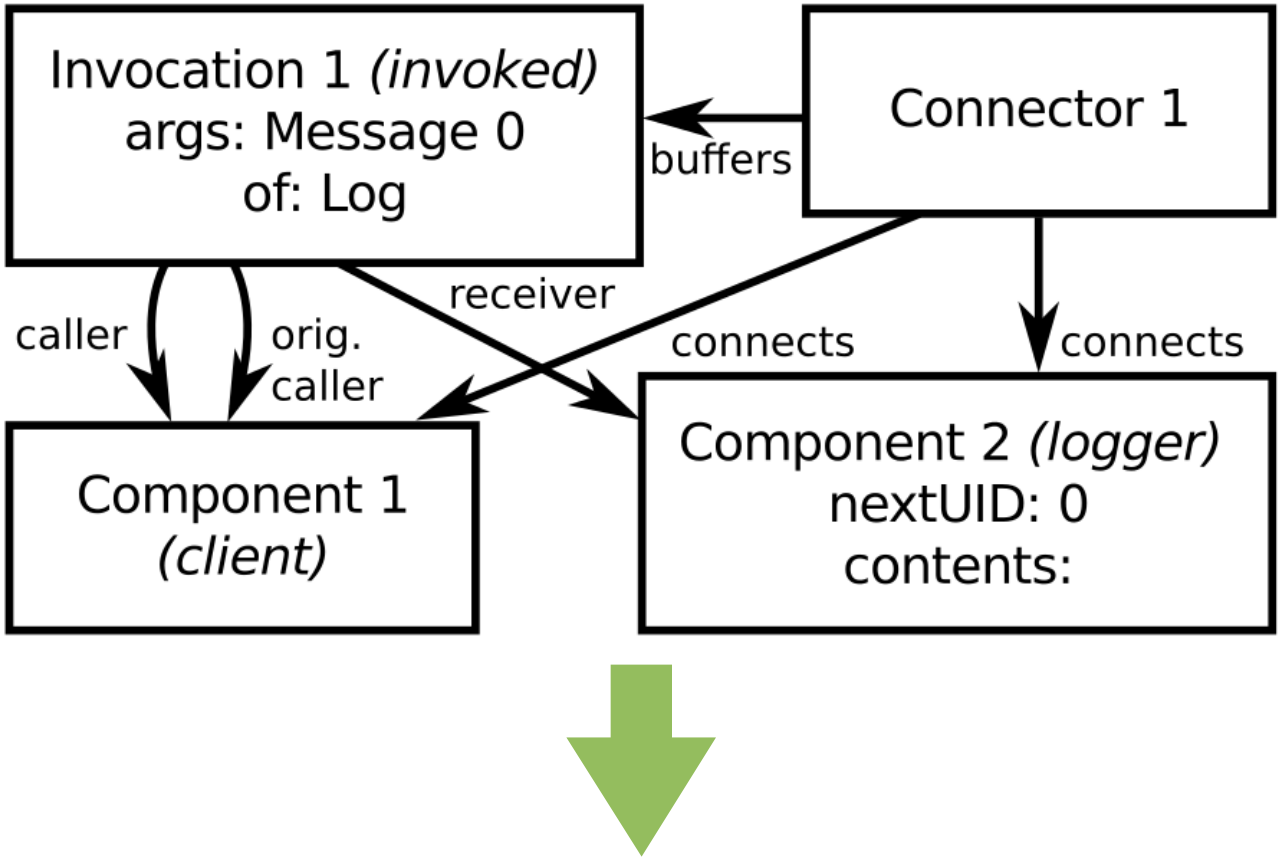

"Whenever a message is logged,
it can be read back later
or the verify method returns *false*."



```
assert NothingDeleted {
 all t:Time,m:Message,l:Logger,c:Component |
  Invoke[c,l,Log,m,t] implies (
   some t1:t.nexts+t {
```

# Verification

## analyzing counterexamples


Automatically generated

"assume that invocations are eventually executed"

Manually interpreted

# Contribution 1

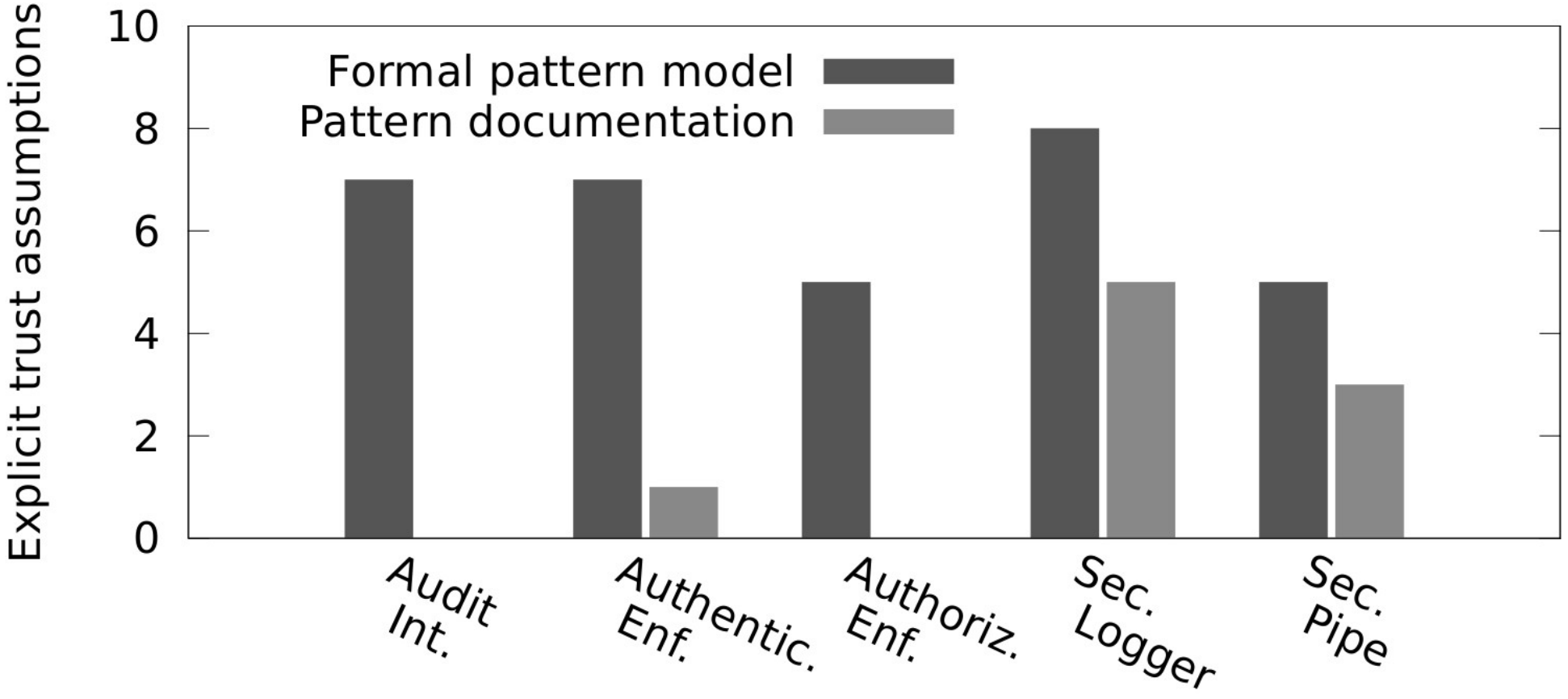

**Trust assumptions!**

- Usually left implicit
- Assurance requires explicit assumptions

**Modelling and verification...**

- Makes them explicit
- Finds extra assumptions

# Uncovered assumptions



# Composing abstract models

# Contribution 2

## Abstraction

```
pred Logger.log(m:Message,t:Time) {
      some c:Component,t1:t.prevs+t | Execute[c,this,Log,m,t1]
}
```

**vs.**

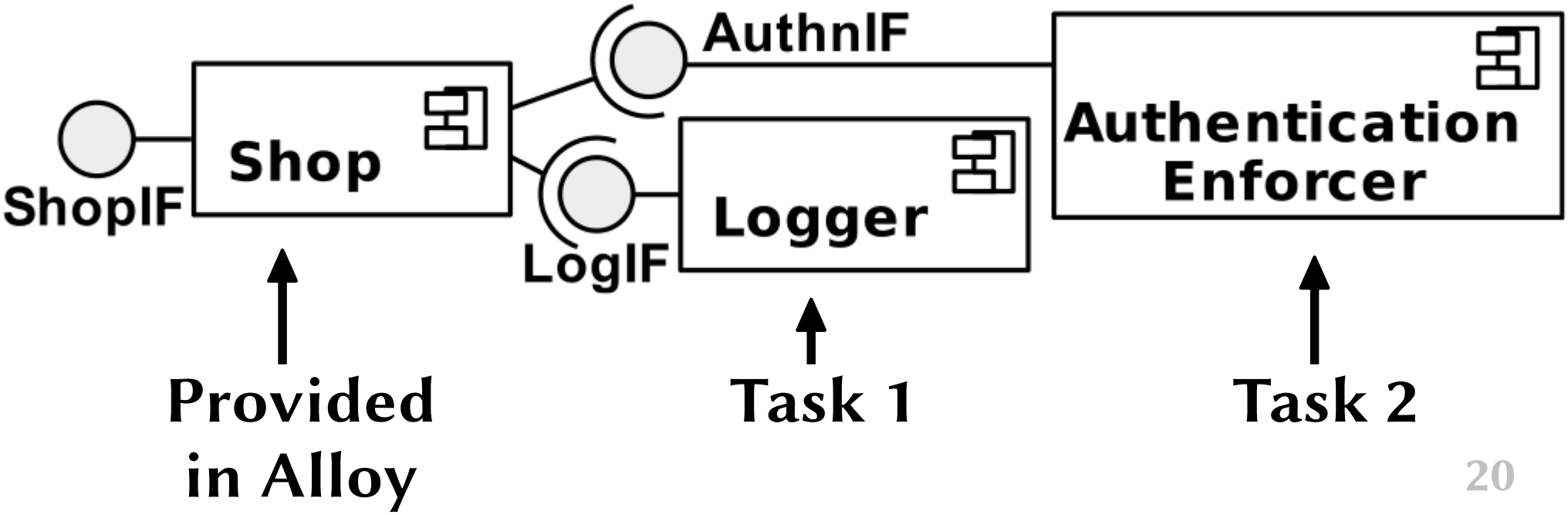

## Refinement

```
pred Logger.log(m:Message, t:Time) {
   some pm:ProcessedMessage, s:SignedMessage {
      pm.content.t = m
      0 <= pm.id.t and pm.id.t < calculateNextUID[this,t]
      s.content.t = pm and s.sign[LoggerEntity,t]
      s in this.contains.t
   }
}
```

# Case study

**Two subjects**
senior researchers

**Extend** basic architecture



# Case study

## results

Both candidates **successful**

- 1St 1 hour, 2nd 2 ½ hour
- + exit questionnaire = useful

Results

- both solutions **correct**
  - (in line with reference solution)
- ±7 assumptions each
- **1 flaw** in solution, results in assumption

# Summary

## what to take home...

Modelled pattern language
for accountability

Verify Once, Reuse Many

Provides insight in patterns

# Big picture

## this research in context

Larger **research track**

Formal methods in
secure software architecture

Under review: formal framework

In progress: DSL + tool support

*http://distrinet.cs.kuleuven.be/software/samodels/*
*thomas.heyman@cs.kuleuven.be*