Karlsruhe Institute of Technology

Quantifying Software Correctness By Combining Architecture Modeling and Formal Program Analysis

KeY Symposium 2023

Florian Lanzinger, Christian Martin, Frederik Reiche, Samuel Teuber, Robert Heinrich, and Alexander Weigl | 2023-08-10

Verification of Large Software Systems

Verification of Large Software Systems

- Many components...

Verification of Large Software Systems

- Many components…
- … with complex interactions…

Verification of Large Software Systems

- Many components...
- ... with complex interactions...
- ... in a complex usage environment

Verification of Large Software Systems

- Many components...
- ... with complex interactions...
- ... in a complex usage environment
- Complete verification becomes impractical

Verification of Large Software Systems

- Many components…
- … with complex interactions…
- … in a complex usage environment
- Complete verification becomes impractical
- Instead, we want to quantify correctness based on partial proofs

The QuAC approach:

Verification of Large Software Systems

- Many components...
- ... with complex interactions...
- ... in a complex usage environment
- Complete verification becomes impractical
- Instead, we want to quantify correctness based on partial proofs

The Quac approach:

→ Modular analysis

Verification of Large Software Systems

- Many components...
- ... with complex interactions...
- ... in a complex usage environment
- Complete verification becomes impractical
- Instead, we want to quantify correctness based on partial proofs

The QuAc approach:

→ Modular analysis

→ Partial analysis
(of software and usage scenarios)

Verification of Large Software Systems

- Many components…
- … with complex interactions…
- … in a complex usage environment
- Complete verification becomes impractical
- Instead, we want to quantify correctness based on partial proofs

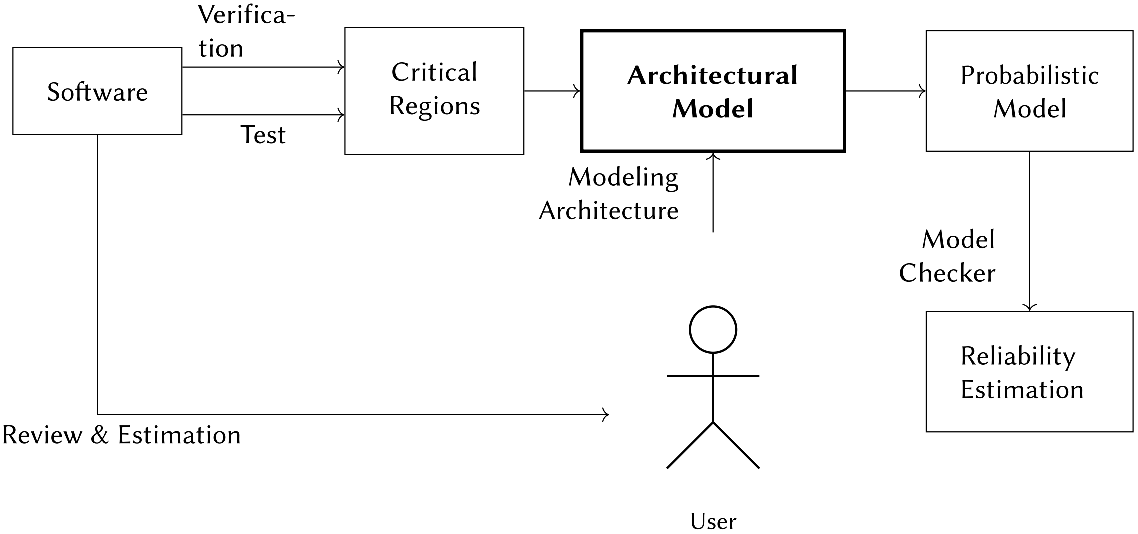
The QuAc approach:

→ Modular analysis

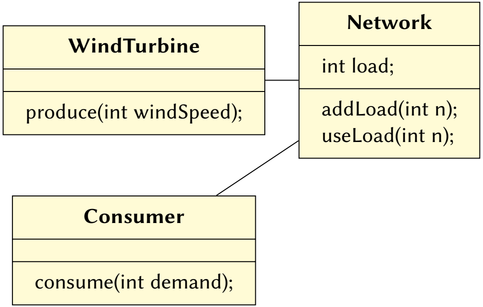
→ Partial analysis
(of software and usage scenarios)

→ Probabilistic model of usage scenarios

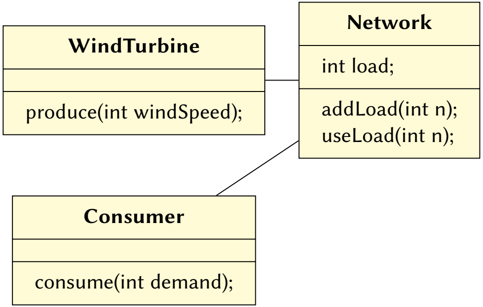
The QuAc Approach



Modeling Architecture and Behavior with Palladio

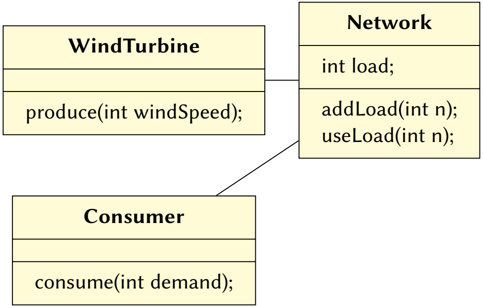


Modeling Architecture and Behavior with Palladio



```
windSpeed ~ 𝒰(5,9);
demand ~ 𝒰(0,4);
windTurbine.produce(windSpeed);
consumer.consume(demand);
```

Modeling Architecture and Behavior with Palladio



WindTurbine

produce(int windSpeed);

Network

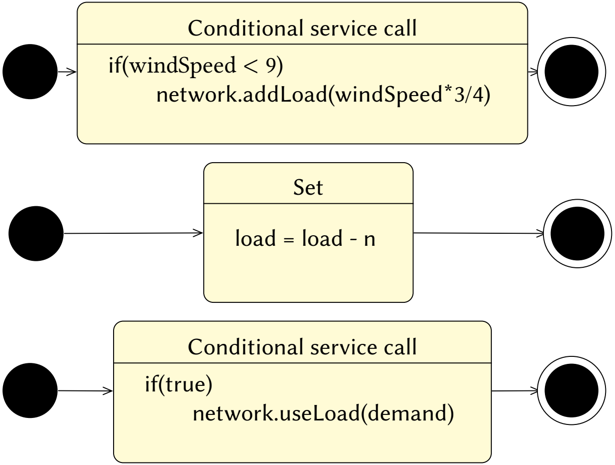
int load;

addLoad(int n);
useLoad(int n);

Consumer

consume(int demand);

```
windSpeed ~ 𝒰(5,9);
demand ~ 𝒰(0,4);
windTurbine.produce(windSpeed);
consumer.consume(demand);
```



Conditional service call

if(windSpeed < 9)
    network.addLoad(windSpeed*3/4)

Set

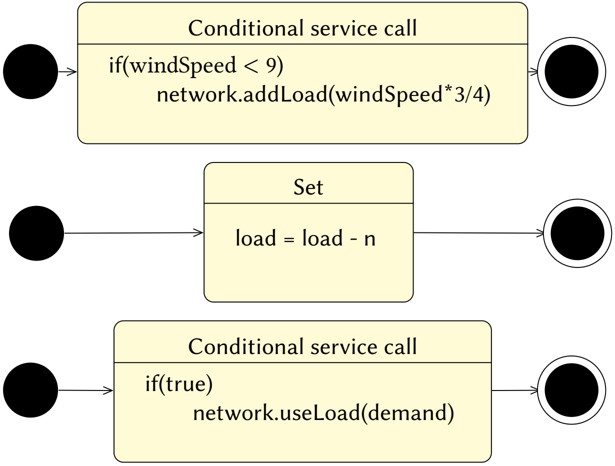
load = load - n

Conditional service call

if(true)
    network.useLoad(demand)

Implementing and Specifying Source Code

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```



Conditional service call

if(windSpeed < 9)
    network.addLoad(windSpeed*3/4)

Set

load = load - n

Conditional service call

if(true)
    network.useLoad(demand)

Analyzing Source Code with KeY

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```

Open goals:

$$\begin{aligned} \text{produce} &\mapsto \emptyset \\ \text{addLoad} &\mapsto \emptyset \\ \text{useLoad} &\mapsto \{\phi \implies \psi, 0 \leq n + \text{self.load}\} \\ \text{consume} &\mapsto \emptyset \end{aligned}$$

Analyzing Source Code with KeY

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```

Open goals:

$$\text{produce} \mapsto \text{false}$$

$$\text{addLoad} \mapsto \text{false}$$

$$\text{useLoad} \mapsto \bigvee_i \neg\phi_i \vee \bigvee_j \psi_j \vee 0 \leq n + \text{self.load}$$

$$\text{consume} \mapsto \text{false}$$

Analyzing Source Code with KeY

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```

Negated open goals:

$$\text{produce} \mapsto \text{false}$$

$$\text{addLoad} \mapsto \text{false}$$

$$\text{useLoad} \mapsto \bigwedge_i \phi_i \wedge \bigwedge_j \neg\psi_j \wedge \neg 0 \leq n + \text{self.load}$$

$$\text{consume} \mapsto \text{false}$$

Analyzing Source Code with KeY

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```

Projected negated open goals:

$$\begin{aligned}\text{produce} &\mapsto \text{false}\\ \text{addLoad} &\mapsto \text{false}\\ \text{useLoad} &\mapsto 0 > n + \text{self.load}\\ \text{consume} &\mapsto \text{false}\end{aligned}$$

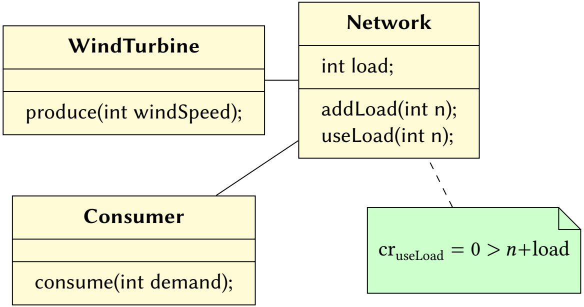
Analyzing Source Code with KeY

```
//@ invariant Network::load >= 0;

WindTurbine::produce(int windSpeed) {
  if (windSpeed < 9) {
    debuglog("producing");
    network.addLoad(windSpeed*3/4);
  }
}

void Network::addLoad(int n) { load += n; }
void Network::useLoad(int n) { load -= n; }

void Consumer::consume(int demand) {
  debuglog("consuming");
  network.useLoad(demand);
}
```

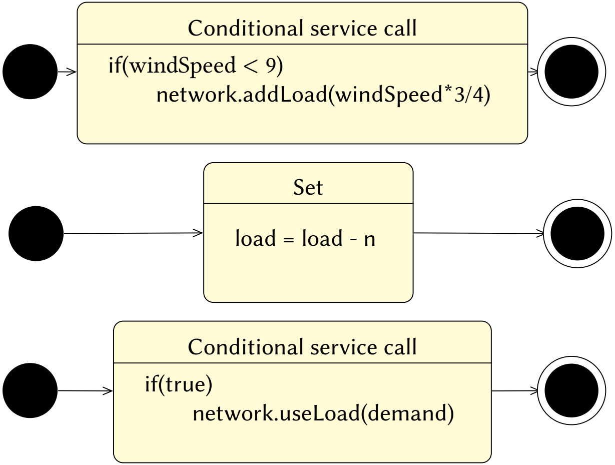
Critical regions:

$$\begin{aligned} \text{produce} &\mapsto \text{false} \\ \text{addLoad} &\mapsto \text{false} \\ \text{useLoad} &\mapsto 0 > n + \text{self.load} \\ \text{consume} &\mapsto \text{false} \end{aligned}$$

Extending Architecture Model with Analysis Results



```
windSpeed ~ 𝒰(5,9);
demand ~ 𝒰(0,4);
windTurbine.produce(windSpeed);
consumer.consume(demand);
```



Transform Extended Architecture Model into Probabilistic Error Model

```
int load;

fun usageProfile():
  load = 0;
  windSpeed ~ U(5,9);
  demand ~ U(0,4);
  produce(windSpeed));
  consume(demand);

fun addLoad(int n):
  if (false): critreg;
  load = load + n;

fun useLoad(int n):
  if (0 > n + load): critreg;
  load = load - n;
```

```
fun produce(int windSpeed):
  if (false): critreg;
  if (windSpeed < 9):
    addLoad(windSpeed * 3 / 4);

fun consume(int demand):
  if (false): critreg;
  if (true): useLoad(demand);
```

Failure probability:

$$\text{Prob(error)} = \frac{1}{5}$$

Demo

Evaluation

- Works for example on the slides
- For realistic programs, exact computation scales terribly
  - Number of code paths
  - More importantly: Number of random variables
  - Perhaps potential for optimization
- Approximate model counting is feasible (run times mostly under 10 min.)

Conclusion and Outlook

- Quantitative analysis of **Safety**
  - Find critical parameter regions with KeY
  - Transfer critical regions into Palladio
  - Compute probability of reachability
    - Depends on usage model

Conclusion and Outlook

- Quantitative analysis of **Safety**
  - Find critical parameter regions with KeY
  - Transfer critical regions into Palladio
  - Compute probability of reachability
    - Depends on usage model
- Outlook: Extension for **Security**
  - Attacker model:
    Attacker can manipulate service calls partially with certain probabilities/costs
    They use this to maximize the probability of entering a critical path