Behavior based Reliable and Resilient System Development

Dr. Muhammad Taimoor Khan

Informatics System Institute

Alpen-Adria University, Klagenfurt, Austria

DeepSec 2017, Vienna

Recent developments (since $>$ 2014)

1. Socioeconomic

- Insider attacks
- Stealthy attacks (APT)
- Snowden revelations
- US policy for browsing
- Automotive vehicles insurance

2. Technological

- CompCert: Commercial certified C compiler
- DeepSpec: End-to-end certified software development (MIT, UPenn, Yale, Princeton, Intel, MS, FB, Google, Amazon, ...)

3. Legislative

- $>$ 42 US states has passed $>$ 240 cybersecurity related bills ($\geq$ 2017)
- No more blackbox computer systems in US Govt. sector

4. Business

- Rubica - first ever cyber crime insurance company offers upto 1 Million \$ cyber fraud coverage
- CompCert

Technological developments

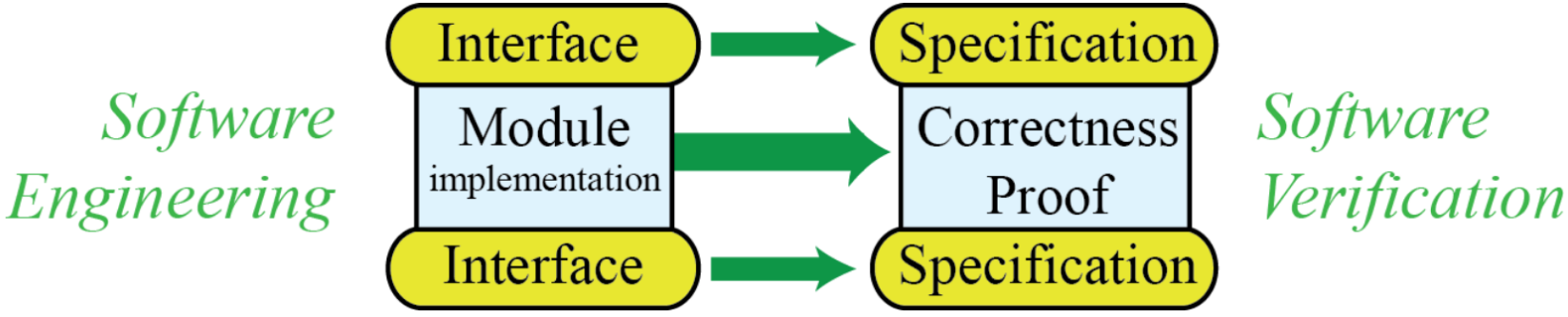
CompCert: Certified C Compiler — May 2015

is a first formally verified optimizing C commercial compiler

- uses machine-checked mathematical proofs (i.e. certified)
- preserves semantics of source language - no extra behavior
- absence of miscompilation issues

DeepSpec: NSF Project — 2016–2021

- academia - MIT, Penn, Yale, Princeton
- industry - Microsoft, Facebook, Amazon, Google, Intel, ...



DeepSpec - End Products

Legislative developments

NY and Colorado States CyberSecurity Regulations 2017

already effective since March and June of this year

- Penetration testing - Companies must provide annual proof of penetration tests, which use specialists to test weaknesses in company infrastructure
- Audit trails - The regulation requires covered entities to prove audit systems showing how they detect cybersecurity events
- Secure development - Regulated companies will have to prove that they use secure software development processes for in-house applications and they test the cybersecurity of external software
- Periodic risk assessments - Cybersecurity assessment is not a one-shot deal. February 2019 will also see companies submit their first reports, which demands periodic risk assessments to show that they are still compliant
- ...

Current system requirements

Certified computing systems that

- support audit against regulations/laws/policies ... among inter-disciplinary domains
- formally assure their reliability and security for operations
- are understandable by any machine or human
- can be verified by any machine
- are self-organized and resilient

Application of program analysis and verification is key to challenges

Challenges

Modern computing systems involve

- Computation + Physics + Chemistry + Biology + ...
- Algorithm + Logic + Control + ...
- Inter-disciplinary domains (e.g., Economics, Energy, Medicine, Law, ...)

Modern computing systems are integrating all operational domains

- systems have become hybrid and overly complex
- reliable and secure interaction among inter-disciplinary domains

The systems must be self-aware and self-organized, i.e. they should know what they are trying to do.

Context

Given a system implementation/design/model C
Show that C and its execution C_e is correct/reliable and secure

Existing solutions

- Testing and Simulation
  - random or highly random - statistics based
  - not rigorous
  - not exhaustive
  - no definition of a reliable test or a reliable simulation
  - shows the presence of bugs/threats and **not** the absence of bugs/threats. **Dijkstra**
  - no formal assurance/guarantee
- Standardization Organizations, e.g. ISO
  - manual based on testing and simulations
  - mostly emperical
  - no formal assurance

The solutions are quantitative and does not offer formal assurances

Our strategy and approach

Our strategy

- Build it right and continuously monitor

We **certify** that

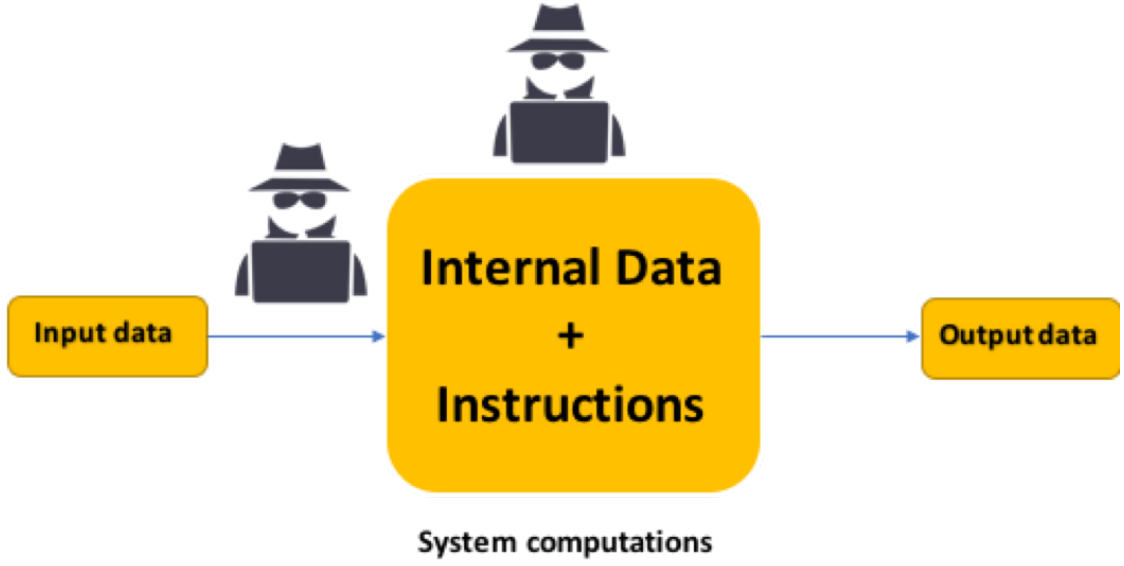
- the design of C is reliable and secure
- and its execution C_e is also reliable and secure

Our approach

- Specify the behavior of a system S as its
  1. functional properties, e.g. pre- and post-conditions, invariant
  2. non-functional properties, e.g. security, performance, energy
- Certify that the design C meets its specification S
  - through step-wise but sound refinements of the design
- Monitor the execution C_e through a middleware
  - ARMET: comparing the executions of specification S and implementation C_e

The system behavior

The behavior characterizes computations operating on input data



Can we assure that

- the computations are reliable and secure, e.g
  - behave as expected
  - terminate
  - are free of bugs/errors
  - cannot be compromised by any insider or external adversary
- the external-input data is reliable, e.g.
  - do we believe that we see is real?
  - data can be legal but not real - data integrity threat

Our assurance

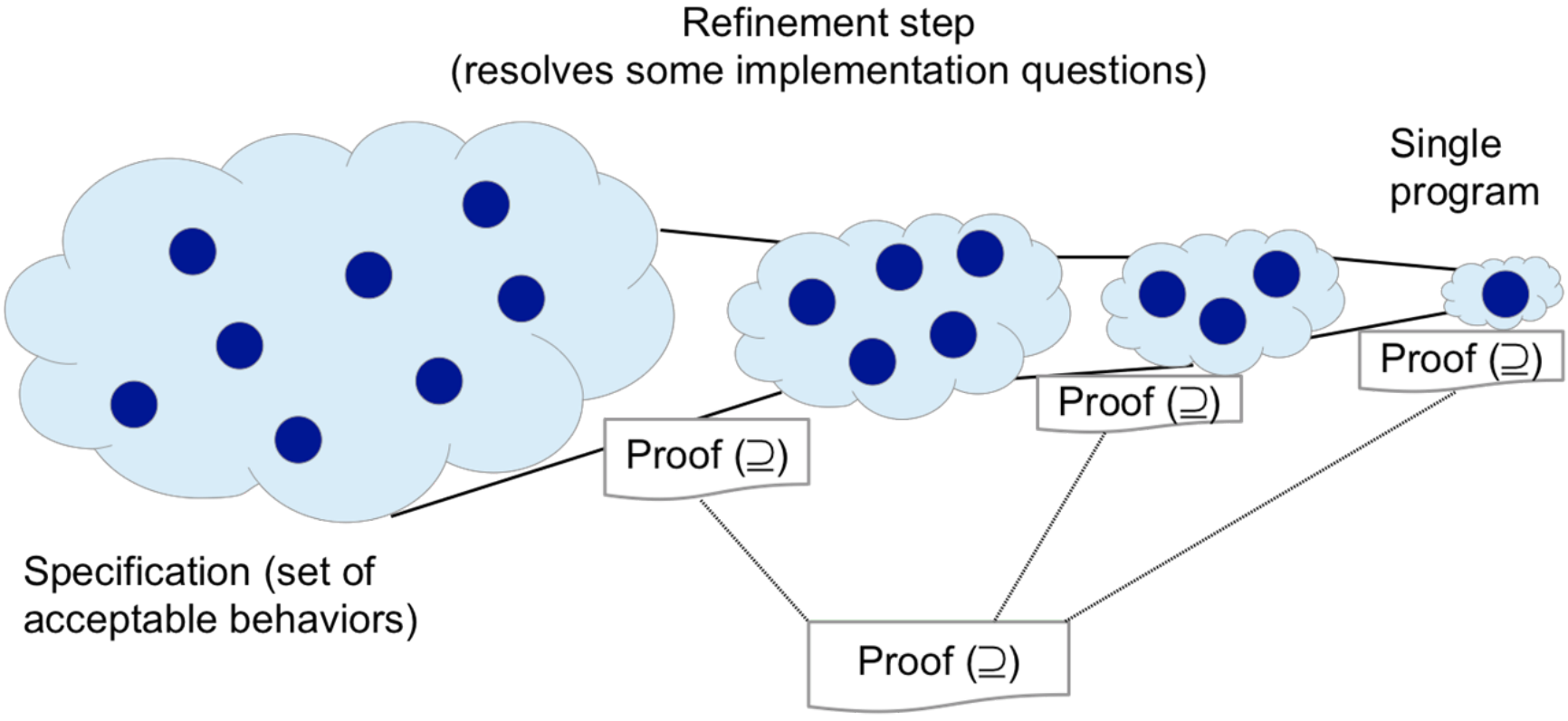
We **certify** that

- the computations design is reliable and secure
  - through stepwise refinement of the system specification
- the external-input data is free of integrity threats
  - through non-linear verification based vulnerability analysis of the system specification
- the computations execution is reliable and secure
  - through monitoring the consistency between expected and execution behaviors
  - optionally, through monitoring the identified integrity vulnerabilities

We design the computations in known environment
and execute the computations in unknown environment

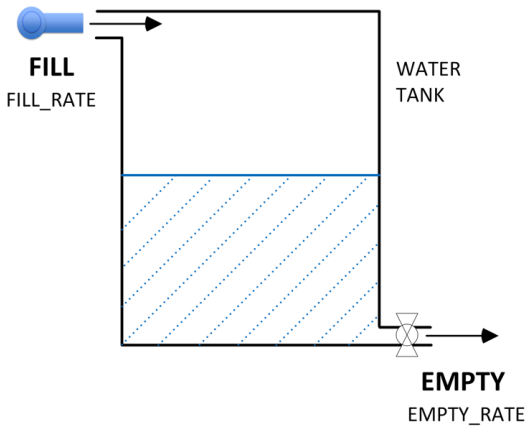
Program derivation through stepwise refinement

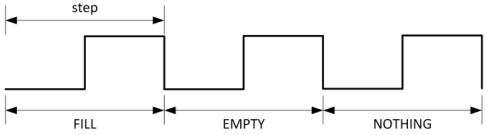
Abstract data type based declarative specification



Proof constructed and checked with Coq  , a general-purpose logic platform

Example specification

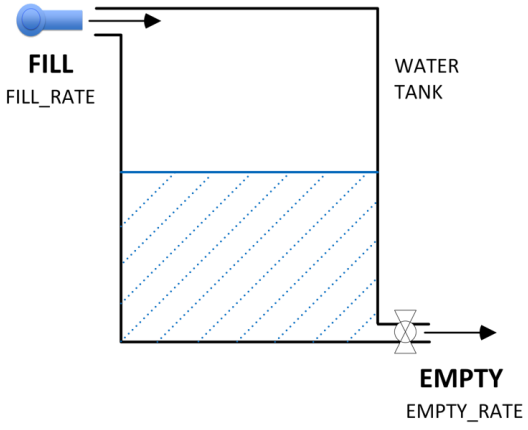


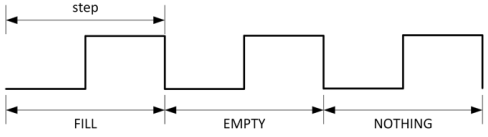



```
public enum Action { NOTHING, FILL, EMPTY }

class WaterTankSpec {
    private int water_level = 0;

    public void newSensorReading(int reading) {
        if (abs(reading - water_level) > SENSOR_ACCURACY)
            water_level = {n | True};
    }

    public Action timestep(int target_level) {
        Action act = {a | (a = FILL → water_level + FILL_RATE ≤ TANK_MAX)
                        ∧ (a = EMPTY → water_level - EMPTY_RATE ≥ 0)};
        if (act == FILL)
            water_level += FILL_RATE;
        else if (act == EMPTY)
            water_level -= EMPTY_RATE;
        return act;
    }
}
```

Specification with a set of behaviors

Example code

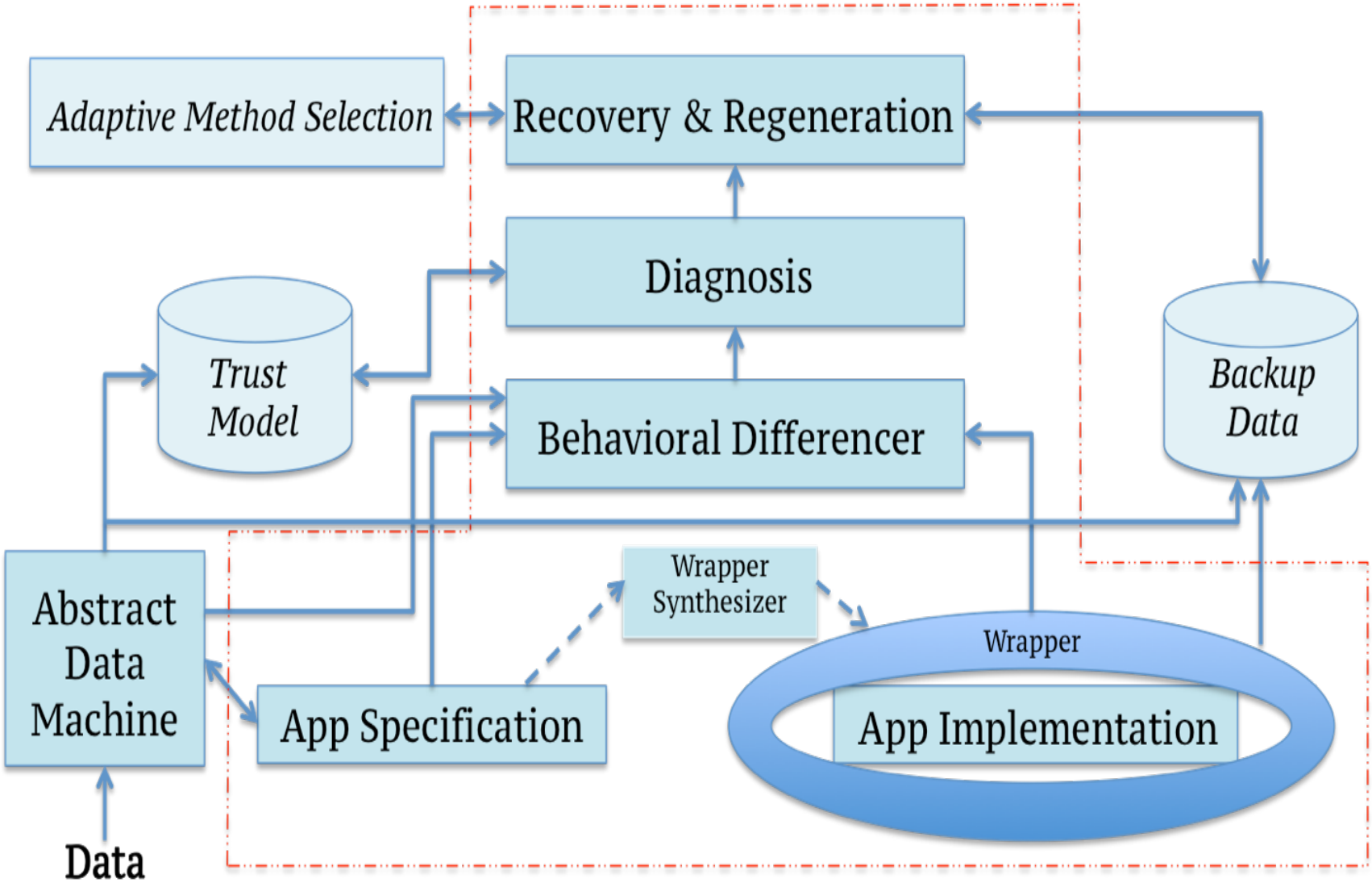




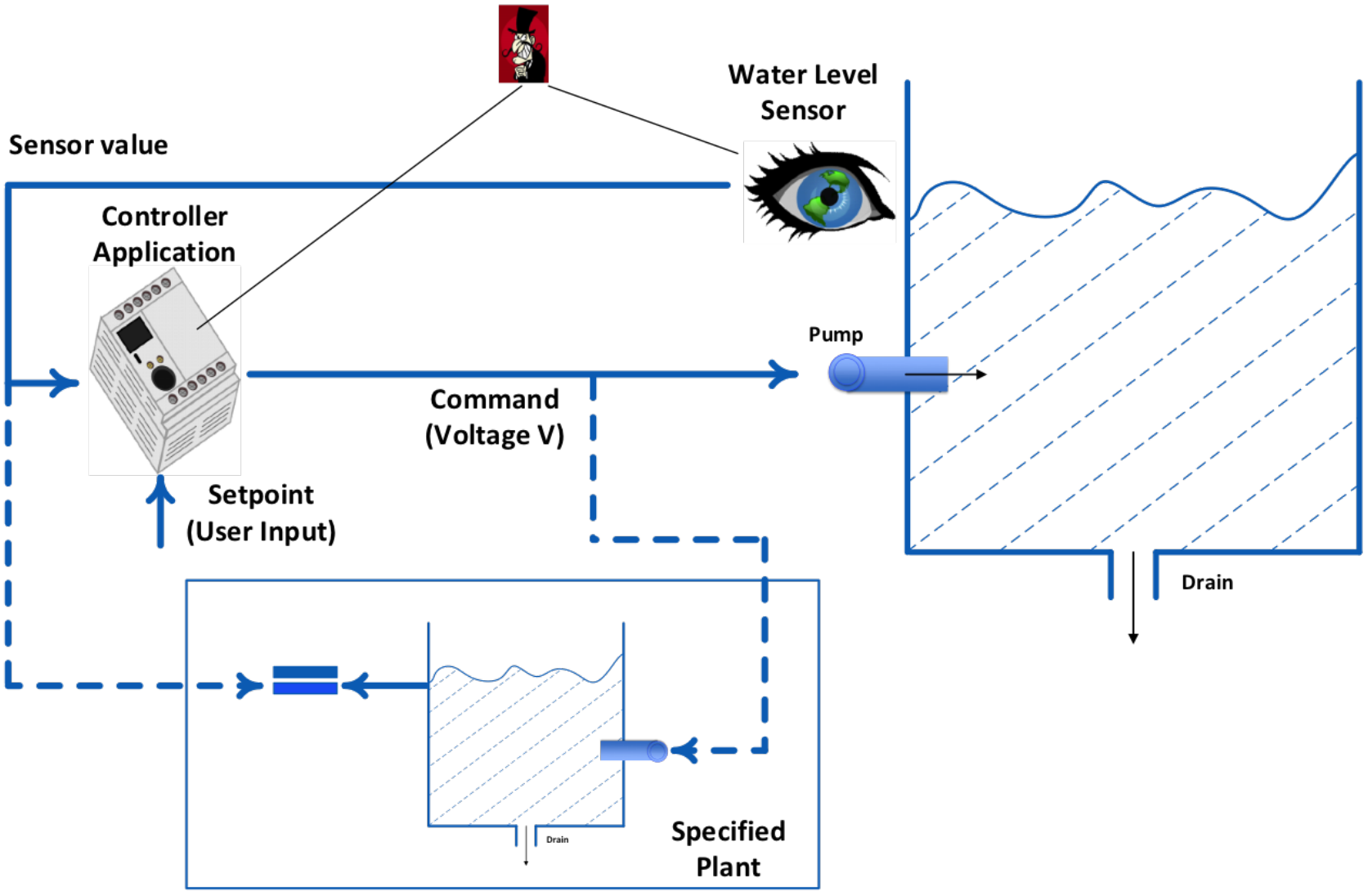

```
public enum Action { NOTHING, FILL, EMPTY }

class WaterTank {
    private int water_estimate = 0;

    public void newSensorReading(int reading) {
        water_estimate = reading;
    }

    public Action timestep(int target_level) {
        if (water_estimate < target_level
        && water_estimate + SENSOR_ACCURACY + FILL_RATE < TANK_MAX)
            water_estimate += FILL_RATE; return FILL;
        } else if (water_estimate > target_level
            && water_estimate - SENSOR_ACCURACY - EMPTY_RATE >= 0)
            water_estimate -= EMPTY_RATE; return EMPTY;
        } else
            return NOTHING;
    }
}
```

Single program with one behavior

ARMET - monitoring and resilience



Demo


Water Level
Sensor
Sensor value
Controller
Application
Pump
Command
(Voltage V)
Setpoint
(User Input)
Drain
Drain
Specified
Plant

ARMET - reliable, secure and resilient software

- Self-aware system
  - through specification and dependency-directed reasoning
- System is allowed to only behave legally
  - specify legal behavior of the system (predictions)
  - observe runtime behavior of the system (observations)
  - continuous monitoring of prediction/observation consistency
  - IF inconsistency, THEN diagnosis
  - recovery (safe state from alternate, reliable resources)
- Detection of known and unknown errors and attacks
  - as inconsistency between observations and predictions
- System adaptability to evolving constraints, standards
  - specify policies as legal behavior and monitor behavioral consistency
- ARMET is sound and complete
  - whenever there is an attack or error, it alarms
  - whenever it alarms, there is an attack or error

Automated vulnerability analysis for data integrity

Vulnerability analysis in non-linear domain is NP hard

- requires solving logical combination of non-linear constraints
- non-convex and non-smooth optimization
  - multiple feasible regions
  - multiple locally optimal points within each region

δ-complete decision procedure solves the problem, e.g. dReal

Given: a model of system (may include non-linear constraints) and a set of properties with an error δ

Ask: are there any values that satisfy the model but are not real?

1. **No** (unsat)
   - there are no such values
   - you are safe
2. **Yes** (δ-sat)
   - there are such values (returns set of values)
   - values are vulnerabilities/attack vectors for the model
   - remove such vulnerabilities from the design
     - refinine/strengthen the model/constraints
     - until unsat

Team

- Muhammad Taimoor Khan
  - Informatics System Institute, Alepn-Adria University, Austria
- Dimitrios Serpanos
  - Director, Industrial Systems Institute, Uni. of Patras, Greece
- Howard Shrobe
  - Director CyberSecurityInitiative, MIT CSAIL, USA
- and all collaborators

Thanks

DEEPSEC
IN-DEPTH SECURITY