Software Engineering for Cyber-Physical Systems

Erik Fredericks
Oakland University
http://efredericks.net // fredericks@oakland.edu
November 2nd, 2018
Graduate students

- **Reihaneh Hariri** (Ph.D.)
  Uncertainty, big data, natural language processing

- **Katherine Bowers** (Ph.D.)
  Self-adaptive systems, search-based techniques, cyber-physical systems

- **Seth Jones** (M.S. ➔ Ph.D.)
  Cyber-physical systems, pushing towards search :)

- **Nicholas Kerns** (B.S.)
The need for software assurance

Assurance
System satisfies its requirements
Provides trust in system

Safety-critical applications
Failure is not simply a violation
Loss of money or life

Increasingly self-adaptive and multi-agent
New explosion of concerns
Focus areas

- Search-based software engineering (SBSE)
- Cyber-physical systems (CPS)
- Self-adaptive systems (SAS)
- Uncertainty
Search-based software engineering (SBSE)

Apply search-based heuristics to software engineering problems

Search techniques:
- Genetic algorithms / genetic programming
- Ant colony optimization / particle swarm optimization
- Hill climbing
- Multi-objective optimization

SE problems:
- For us, mainly requirements engineering / software testing concerns
- Moving towards formalisms as well
Application to cyber-physical systems (CPS)

CPS introduce a new set of concerns in the cyber-physical boundary

Are hardware failures causing software timing conflicts?

Will a software fault cause a robot effector to be damaged?

Can a CPS be optimized via software engineering techniques?
Uncertainty

We focus on known unknowns [Esfahani.2011] and emergent behaviors / feature interactions [Esfahani.2011a]

Or, contextual uncertainty ➔
   Combination of unanticipated system parameters and environmental states
   Scale is enormous, especially at cyber-physical boundary

Impacts
   Software violations
   Hardware faults
   Poor performance
Self-adaptive systems (SAS)

Reconfigure at run time in response to uncertainty
Long-running systems
MAPE-K loop [Kephart.2003]

Self-adaptation enabled by feedback loop
Multi-agent systems

Multi-agent systems (MAS)
Autonomous agents working towards common goal
Each agent’s goals may be unique!

Generally communicate via message passing
Ensure that system continuously provides **acceptable** behavior in the face of uncertainty
Key challenges (general)

Ensure system continuously satisfies its key objectives, both functional and non-functional at run time

**Functional:** measurable requirements

“The system shall operate within 5.5V and 7.5V”

**Non-functional:** qualitative / hard to measure

“The application should have an intuitive user interface”
Measuring objectives at *run time*

Run-time requirements monitoring
Enabled via **utility functions** [deGrandis.2009]

\[
\text{util}(\text{goal}_v) = \begin{cases} 
1.0 & \text{if Num Passive Data Mirrors} \equiv 0 \\
 x & \text{if } 0 < \text{Num Passive Data Mirrors} < 20\% \text{ of total nodes} \\
0.0 & \text{if Num Passive Data Mirrors} \geq 20\% \text{ of total nodes}
\end{cases}
\]

0.0 \rightarrow \text{violation}, 1.0 \rightarrow \text{satisfied}, (0.0, 1.0) \rightarrow \text{satisficed}
Current and recent research projects
Cognitive Assisted Living (CAL)

Support Alzheimer’s patients at home to detect cognitive decline

Quantify patient responses as software engineering metrics via NLP techniques

Provide early-warning indicators of decline

[Fredericks.2018]
Alzheimer’s Disease

Cognitive state diagnosis occurs during doctor’s visits

- To which stage of Alzheimer’s a patient belongs
- Three/five stages common, but these are not discrete
CAL Environment

Doctor’s visits occur yearly/quarterly

1. Can you tell me something that happened in the news recently?

2. Please draw a circle and mark in the numbers to indicate the hours of a clock.

CAL provides daily reports via medical surveys
CAL Environment

*Extensible* SmartHome environment

Initial concept:
- Tablet
- Server
- Cloud services
- Natural language processing

Extensions planned for incorporating wearables
- Heart rate, blood pressure, etc.
- Neural monitoring
Software Engineering Implications

System is **multi-agent** and **self-adaptive**

- Each separate agent must be performing correctly
- System as a whole must be performing correctly
- Run-time adaptations must be verified

**Non-functional** concerns as well!
- Security implications
- Power optimization
- Usability
  - Any “ility” really

An Extraordinary Implication from Fallen London / Sunless Sea
Working!
Currently at an object!

TASKS

1. Turn off the stove
2. Close the fridge
3. Comfort the patient
Recent projects

**Providentia** [Bowers.2018 - SSBSE]

Automated optimization of non-functional requirements (NFR)

- Provide metrics for quantifying NFRs in a goal-oriented framework (KAOS)
- Optimize system performance via genetic algorithm (GA)
  - GA goal: optimize NFRs
Providentia - Initial Goal Model

(A) Maintain [DataAvailable]

(B) Maintain [Operational Costs \leq Budget]

(D) Achieve [Measure Network Properties]

(E) Achieve [Minimum Num Links Active]

(F) Achieve [Network Partitions == 0]

(J) Achieve [Cost Measured]

(K) Achieve [Activity Measured]

(L) Achieve [LossRate Measured]

(M) Achieve [Workload Measured]

(N) Achieve [Capacity Measured]

(O) Achieve [Link Deactivated]

(P) Achieve [Link Activated]

Sensor

RDM Sensor

Network Actuator
Providentia - NFR Aggregation

Key

Goal | NFR | Requirement / Expectation

Minimize [Power]

Goal (A): Maintain [DataAvailable] (0.3)
Goal (E): Achieve [Minimum Num Links Active] (0.1)
Goal (I): Achieve [Adaptation Costs == 0] (0.1)
Goal (V): Achieve [Num Passive Data Mirrors == 0] (0.3)
Goal (W): Achieve [Num Quiescent Data Mirrors == 0] (0.2)

Quantification: weighted sum of functional goal utility function
Providentia - Genome

<table>
<thead>
<tr>
<th>Goal:</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>I</th>
<th>K</th>
<th>M</th>
<th>O</th>
<th>P</th>
<th>V</th>
<th>W</th>
<th>...NFR_{n}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome:</td>
<td>0.3</td>
<td>0.0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.3</td>
<td>0.2</td>
<td>...NFR_{n}</td>
</tr>
</tbody>
</table>

**NFR7**

**Fig. 4: Providentia sample genome.**

\[
\text{util}(\text{goal}_V) = \begin{cases} 
1.0 & \text{if } \text{Num Passive Data Mirrors} == 0 \\
 x & \text{if } 0 < \text{Num Passive Data Mirrors} < 20\% \text{ of total nodes} \\
0.0 & \text{if } \text{Num Passive Data Mirrors} \geq 20\% \text{ of total nodes}
\end{cases}
\]
 Providentia - Results

Significantly reduced both invariant and non-invariant goal violations (utility = 0.0)

Demonstrated tradeoffs when satisfying NFRs (competing global objectives)

(a) Invariant violation comparison.  (b) Noninvariant violation comparison.

Fig. 6: FR violation experimental results.
Ragnarok / Valkyrie

First we **break the system (Ragnarok)**
Then we **fix the system (Valkyrie)**

Two separate genetic algorithm-based techniques
- Ragnarok: generate environmental conditions to induce violations
- Valkyrie: generate system configurations to combat discovered environments

[Fredericks.2016 - SEAMS]
Figure 4: Data flow diagram of Ragnarok and Valkyrie techniques.
Figure 9: Comparison of RDM non-invariant goal violations between Ragnarok and random search.

Figure 10: Comparison of average fitness values between Valkyrie and random search.

Test cases may lose relevance as their environment changes
  Camera sensor lens scratched as SVS drags against brick wall
  Reduced visibility
  Disabled sensor
  Desire to continue *safe* operation

E.g., sensing radius still valid but reduced
  **Ideal** = 0.5m
  **Radius** = 0.4m
  **Minimum** = 0.3m
Run-time testing techniques

Proteus
Adapting **test plans** at run time
Monitor state of adaptive system and update test case relevance

Veritas
Evolving **test cases** at run time
If test case relevance drifting from optimal...
(1+1)-ONLINE EA searches for new test case parameter values
...within reason
Proteus results (RDM)

Fig. 3. Cumulative number of irrelevant test cases for each experiment.

Fig. 4. Cumulative number of false negative test cases for each experiment.
Veritas results (RDM)

Fig. 6. Comparison of average relevance between Veritas and Control test cases.
AutoRELAX - Requirements-based Uncertainty

RELAX operators

<table>
<thead>
<tr>
<th>AS EARLY AS POSSIBLE</th>
<th>AS LATE AS POSSIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS MANY AS POSSIBLE</td>
<td>AS FEW AS POSSIBLE</td>
</tr>
<tr>
<td>AS CLOSE AS POSSIBLE TO [f]</td>
<td>AS CLOSE AS POSSIBLE TO [q]</td>
</tr>
</tbody>
</table>

RELAX operators [Whittle2009]

SVS Goal A

ACHIEVE [50% Clean]

RELAXed SVS Goal A

ACHIEVE

[AS CLOSE AS POSSIBLE TO 50% Clean]

RELAX example
AutoRELAX

Genetic algorithm optimizes application of RELAX operators
  ● Minimize number of RELAXed goals
  ● Minimize adaptations

AutoRELAX-SAW [Fredericks.2013]
  ● Add hyper-heuristic to optimize fitness function
    ○ Stepwise adaptation of weights (SAW)
Upcoming research

Modeling a medically-oriented smart home (CAL) as a self-adaptive system
Including patient studies to quantify verbal/spatial responses

Leveraging wearable devices and cameras to detect patient/user state

Uncertainty in big data analytics
References


References


And thanks to..
Application domains

Remote data mirroring (RDM) network
  Provided by industrial collaborators

Smart vacuum system (SVS)
  Transitioning work towards cyber-physical domain
RDM network application

Network of physically-remote servers

Uses:
  - Uptime / availability
  - Data protection

Reconfigurations:
  - Topology changes
  - Data propagation
  - Server state

Uncertainty:
  - Dropped/delayed messages
  - Severed links
SVS

Robotic vacuum that’s definitely NOT modeled on the Roomba
Must clean most of a simulated environment

Adaptations:
  Switch power mode
  Change path algorithm
  Obstacle avoidance

Uncertainties:
  Steps
  Dirt location/distribution
  Unsafe conditions
Basics of search algorithms
GA

Generate initial population

Evaluate population and calculate fitnesses

Perform evolutionary operations (selection, mutation, crossover)

Retain elite individuals and regenerate the rest of the population

Done?

Evaluate one final time

Output best solution(s)
Software Engineering for Cyber-Physical Systems Course
Lab Setup

Every student was assigned Raspberry Pi 3B
Class Materials

Class mainly focused on cutting-edge research in software engineering
● Generally motivated in the context of cyber-physical systems (CPS)

Topics included:
● Goal modeling
● Uncertainty analysis
● Self-adaptive / Multi-agent systems
● Search-based software engineering
● Etc.

And of course
● The Cloud
Lab Materials

- REST server/client
- Multiprocessing (OpenMPI)
- Encryption/decryption
- Requirements monitoring

- Offloading processing to another entity
  - The Pi isn’t the best at processing
Integrating Google Cloud

Leverage Google Cloud to offload heavy processing tasks

Teach students how tiny, low-power nodes can still be useful!

Each student received $50 in free credits from Google
(If you don’t know about this, sign up! https://cloud.google.com/edu/)
Integrating Google Cloud

Labs focused on mini-projects using:

- **Vision API**
  - (Near) real-time image recognition

- **BigQuery API**
  - (Near) real-time big dataset parsing

Based modules on Codelabs + random tutorials
Vision API

Classify various images using Google Cloud

- Logos
- Label devices (recognize object)
- Facial expressions
  - One student turned this into a term project for recognizing his face out of other faces
BigQuery API

Student interact with the open-source datasets indexed by Google

E.g., Github, Reddit, etc.

We focused on Github, specifically commit statements

- Searched for the number of times that TBD, TODO, or FIXME appears
- Interestingly, there was a paper at AAAI (NL4SE Workshop) mining similar information
```sql
SELECT SUM(copies) FROM [bigquery-public-data:github_repos.sample_contents] WHERE NOT binary AND (content CONTAINS 'FIXME' OR content CONTAINS 'TODO' OR content CONTAINS 'TBD')
```

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9034065</td>
</tr>
</tbody>
</table>
Extra Fitness Functions
AutoRELAX fitness functions

- Fitness determination
  - Minimize number of RELAXed goals: $$FF_{mrg} = 1.0 - \frac{|\text{relaxed}|}{|\text{Goals}_{\text{non-invariant}}|}$$

- Minimize number of adaptations: $$FF_{na} = 1.0 - \frac{|\text{adaptations}|}{|\text{faults}|}$$

- Overall fitness: linear weighted sum

$$Fitness = \begin{cases} 
\alpha_{mrg} \cdot FF_{mrg} + \alpha_{na} \cdot FF_{na} & \text{iff invariants true} \\
0.0 & \text{otherwise} 
\end{cases}$$

$$\alpha_{mrg} = 0.3 \quad \alpha_{na} = 0.7$$
Veritas relevance calculation

- To perform evolution, some measure of fitness must be defined for each test case
  - Test case fitness (relevance) is defined as:
    \[
    \text{distance}_{\text{normalized}} = 1.0 - \frac{|\text{value}_{\text{measured}} - \text{value}_{\text{expected}}|}{\text{value}_{\text{expected}}}
    \]

- For example:
  - Test case expected value = 0.50m
  - Test case measured value = 0.45m
  - Fitness = 0.90
  - High relevance to environment
  - Test case expected value = 0.50m
  - Test case measured value = 0.01m
  - Fitness = 0.02
  - Low relevance to environment