“Self-Healing”: Softening Precision to Avoid Brittleness

author: Mary Shaw
presenter: Heather Goldsby

Overview

- Practical Systems
- Gathering Realistic Requirements pertaining to “health”
- Developing a self-healing system in light of the realistic system properties and requirements

Programs vs. Practical Systems

<table>
<thead>
<tr>
<th>The Fantasy...</th>
<th>The Reality...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs: complete knowledge</td>
<td>Systems: Approximate Knowledge</td>
</tr>
<tr>
<td>Often execute and stop</td>
<td>Run Indefinitely</td>
</tr>
<tr>
<td>Stable knowledge configuration</td>
<td>Dynamic components and configuration</td>
</tr>
<tr>
<td>Component well specified</td>
<td>Components poorly understood</td>
</tr>
<tr>
<td>Prevent failures</td>
<td>Remediate problems</td>
</tr>
<tr>
<td>Causal domain*</td>
<td>Biddable domain*</td>
</tr>
<tr>
<td>Goal of correctness or perfection*</td>
<td>Goal of sufficient correctness*</td>
</tr>
</tbody>
</table>

* as explained on upcoming slides
Programs vs. Practical Systems: Domains

- System operates
- Assumptions about the domain impact our view of adaptation
- Causal Domain
  - Predictable causal relationships, which allow the effect of machine behavior to be calculated at the interface with the domain.
  - E.g., elevator
- Biddable Domain
  - Lack predictable causality because of uncontrolled external influences

Programs vs. Practical Systems: Correctness

- Goal of program quality is correctness - a demonstration that the program will compute what its (formal) specification claims it will compute.
- In practice, however...
  - Requirements are imprecise and hard to obtain because of constraints such as time, cost and limited knowledge
  - We obtain useful results without correctness - Windows (enough said)
- The goal of practical systems should be sufficient correctness

Sufficient Correctness

- Sufficient correctness is the degree to which the system developer aspires to establish that the system meets its specifications, given constraints of...
Practical Systems

- Maintaining the health of practical systems is complex
- Health itself is imprecisely defined by incomplete and dynamic requirements
- Because health is not fully understood repair mechanisms must operate independently of the detection of unacceptable behavior

What is health?

- To achieve sufficient correctness and ensure a piece of software meets its specifications we need specifications.
- Specifically, what is "health"?
- Classic models try to capture full static requirements, but run into problems because:
- The distinction between "healthy" and "broken" is indistinct
- Requirements change over time
- Knowledge of properties is incomplete because the list is open ended, and the cost of specification is high
- User's preference among properties is very complex
- Shaw's practical systems reason from partial knowledge
- Specifically, credentials, which are incremental, evolving specifications.
- Some of credentials are codified, some are in the public domain.

What is illness?

- Traditional programs have two states - normal and broken
- Transition from normal to broken
- Traditional self-healing programs have the same two states
- Transition from normal to broken
- AND from broken to normal
- Depends on precise requirements for system "health"
Real Systems

Explicit distinctions between normal and non-normal states are artificial.
Instead, there are gradual transitions between desirable and undesirable conditions.

Real Self-healing Systems

Gradient between acceptable and non-acceptable behavior.
Mechanisms move system from broken to normal.
Mechanisms also improve system within range of normal, or within the range of broken to make things less broken.

What is illness?

Health corresponds to sufficient correctness.
This definition recognizes imprecise requirements and different uses of systems.

Realistic Self-Healing Systems

How to self-heal

Traditional Self-Healing Approaches

Explicit specification of conditions that trigger healing responses.
Feedback mechanisms compare system conditions to setpoints and try to drive the system towards a setpoint.
Problems with traditional approach.
Distinction between health and illness is gradual.
Must be able to predict the effect of...
Homeostasis

- *def:* “the mechanism through which a system acts to maintain a stable internal environment despite external variations.”
- *mechanisms which react to change without distinguishing between “good” and “bad” states*
- *e.g., predator/pray relationship*

Software Homeostasis

- *def:* “as a software system property refers to the capacity for the system to maintain its normal operating state, or the best approximation to that state, as a result of normal operation.”
- *e.g., background garbage collection, internet packet routing*

Advantages to Software Homeostasis

- *Does not require precise specification of “health”*
- *Mechanisms improve performance independent of health status*
- *Thus, they do not rely on the detection of unacceptable behavior*
Does the gradient between desirable and undesirable system conditions appropriately capture the notion of system health?

Do you think it is possible to implement homeostasis for a system that has conflicting goals such as conserving battery and providing high quality audio transmissions (through methods such as using FEC)?

How does Shaw claim softening precision avoids brittleness?

Questions???

Additional References


