A Self-Organizing Feedback Loop for Autonomic Computing

Bogdan Alexandru Caprarescu
bogdan@softinvent.org
http://bogdan.softinvent.org

Dana Petcu
petcu@info.uvt.ro
Agenda

- Autonomic Computing
- Engineering Autonomic Systems
- Towards Self-Organizing and Emergent Feedback Loops
- A Robust Architecture for Autonomic Systems
- The Multi-Agent Feedback Loop
- Evaluation
- Conclusion and Future Work
Autonomic Computing

- Autonomic systems manage themselves according to certain policies set by human administrators [Kephart and Chess, 2003]
- Named after the autonomic nervous system
- Four aspects of self-management:
  - Self-configuration
  - Self-optimization
  - Self-healing
  - Self-protection
- Autonomic systems are often referred as self-managing or self-adaptive systems
Top-down approaches
- Are usually centralized
- Employs one or several feedback loops to provide self-management
- MAPE-K (Monitor, Analyze, Plan, and Execute over a Knowledge base) is the reference model for autonomic feedback loops [Kephart and Chess, 2003]
Top-down approaches

- Advantages
  - Separate functionality from management => generic feedback loops
  - Already-built non-autonomic systems can be easily enhanced with autonomic capabilities
- Disadvantage: While managing the system, the feedback loop may itself encounter faults.

Where do we stop?
Bottom-up approaches
- Inspired from biological social systems: social insect colonies (e.g., ants, termites, bees), schools of fish, social human behavior, etc
- Are highly decentralized
- Are usually self-organizing
- Emergent self-management
Bottom-up approaches

- Advantages: high robustness and scalability
- Disadvantages:
  - It remains a challenge to completely understand how biological systems work.
  - Lack of mathematical models.
  - Limited applicability of biologically-inspired architectural styles.

Reality: most autonomic systems make a compromise between the top-down and bottom-up approaches.
Our Vision: Self-Organizing and Emergent Feedback Loops

- Inherit most advantages of both top-down and bottom-up approaches:
  - Generic and reusable feedback loops can be developed.
  - Can be used to manage already built non-autonomic systems.
  - Benefit from built-in fault tolerance
  - It is a more tractable approach than building fully self-organizing and emergent systems
A Robust Architecture for Autonomic Systems

- Managed System
  - A collection of interconnected components

- Policy Repository
  - Policies are used to specify the self-management behavior
  - Priorities are used to resolve policy conflicts

- Multi-Agent Feedback Loop (MAFL) heals the managed system and heals itself
The Multi-Agent Feedback Loop (MAFL)

- The management policies are assigned to a population of distributed agents.
- Each agent fulfills exactly one policy.
- The agents self-organize into groups.
  - All agents in a group use the same policy.
  - The number of agents in each group has to be between two thresholds: $L$ and $U$.
- Only one agent is allowed to execute changes at a time.
MAFL – Design Principles

- Design principles inspired by the behavior of ants:
  - Agents are anonymous and individually expendable
  - Agents communicate using messages as surrogates of pheromones
  - The communication is asynchronous and one way
  - All messages are broadcasted and individually expendable
  - Non-deterministic interractions: Agents query asynchronously the policy repository
    - Messages are unicast and grouped in request-reply pairs
    - These messages are also individually expendable
MAFL – Agent Architecture

- **Communicator**
  - Exchanges messages with other agents and policy repository
  - Performs state transitions
  - Allows or denies the autonomic loop to perform changes

- **Autonomic Loop**
  - Applies one management policy
MAFL – Agent States and Messages

- **Agent states**
  - **Searching** – without a policy; seeks to acquire one and elevate to working state
  - **Working** – monitors the managed system
  - **Candidate** – desires to execute changes
  - **Master** – is executing changes

- **Inter-agent message types**
  - **Working message** – sent by working, candidate and master agents to regulate the size of the sender’s group
  - **Candidate message** – sent by a candidate agent announcing its intention to become master
  - **Master message** – sent by the master to force all candidates to go back to the working mode and prohibit any working agent from becoming candidate
MAFL at Work

- Initially all agents are in the searching state (a)
- Agents acquire policies and form groups of working agents (b)
- Working agents (yellow) monitors the managed system (b,c,d)
- Some agents desires to execute changes (c)
- Only one agent is allowed to execute changes at a time (d)
MAFL – Self-Organizing Phase

- A searching agent:
  - Monitors the size of working agents groups.
  - Has two options to elevate to working state:
    - It joins a too small group of working agents (A)
    - It creates a new group of working agents (B)

- A working agent:
  - Monitors the size of its group
  - Demotes to searching state in one of the following cases:
    - Its group is too large (C)
    - Its policy was deleted from the repository (D)
MAFL – Management Phase

- A working agent
  - Monitor the managed system
  - Elevates to candidate when a need for change is detected (E)
- A candidate
  - Demotes to working when receiving either a higher priority candidate message (F) or a master message (G)
  - Elevates to master after surviving a given timeframe (H)
- Master demotes to working in one of the following cases:
  - Master received a higher priority master message (I)
  - Master finished executing changes (J)
A prototype of MAFL and policy repository was implemented in Java
- Datagrams were used as the only means of communication
- The group size thresholds were set to $L = 2$, $U = 3$
- Available for evaluation at http://bogdan.softinvent.org/research/feedbackloop

First test: configurations of up to 4 computers, 25 policies and 150 agents.
- The prototype always emerged to a stable state providing that at least $U \times N_{\text{policies}}$ healthy agents were running.
- The prototype remains stable for up to 5% message losses.
Second test: 5 policies, 30 agents, 3 computers.
- At low level the agents’ interactions are largely non-deterministic
- At high level the behavior tends to be deterministic: higher priority policies are more often executed.

<table>
<thead>
<tr>
<th>Policy Id</th>
<th>Priority</th>
<th>Activ. Prob.</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Low</td>
<td>70%</td>
<td>24%</td>
</tr>
<tr>
<td>1</td>
<td>Low</td>
<td>50%</td>
<td>21%</td>
</tr>
<tr>
<td>2</td>
<td>Medium</td>
<td>50%</td>
<td>43%</td>
</tr>
<tr>
<td>3</td>
<td>Medium</td>
<td>30%</td>
<td>44%</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
<td>10%</td>
<td>59%</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

- Our idea is to engineer self-organizing and emergent feedback loops for autonomic computing.
- We have developed a multi-agent feedback loop which is self-organizing and exhibits emergent qualities: it is very robust and stable.
- Our future research plans:
  - Allow multiple agents to simultaneously execute non-conflicting changes
  - Use the feedback loop to manage a real software system
  - Evaluate the scalability of the feedback loop
  - Locality in interaction