Reactive Programming for Data-center Management: First Experiences and Prospects

Nicolas BERTHIER    Éric RUTTEN    Noël DE PALMA

November 18-22, 2013
Synchron’13
Outline

- Context & Motivations
- Reactive Programming...
- ... for n-tier Applications Management
- Validation
- Summary
Outline

- Context & Motivations
- Reactive Programming...
- ... for \( n \)-tier Applications Management
- Validation
- Summary
Autonomic Management Systems (AMSs)

Management Tasks

- Facilities: Energy, Cooling...
- Computation Infrastructure: Physical Machines...
- Application Management
  - Resource Allocation (Virtual Machines...), Configuration...
  - Example 3-tier Application:
    → Apache → Tomcat → Mysql
Autonomic Management Systems (AMSs)

Flow of Monitored Data
Sensors
Application, Data-center...

Flow of Execution Orders
Actuators

AMS Software
Autonomic Management Systems are Reactive Systems

Noticeable Properties

▶ Typical Feedback Loop
▶ Sensors & Actuators Spread Over the Managed System
▶ Most Often: Centralized Execution of the Software Taking Management Decisions
▶ Yet: Management Aspects Addressed Independently from Each Other
  ▶ Sizing / Repairing /…
Motivating Example: The *Sizing-repair* Case

### Sizing
- Over-load $\leadsto add$
- Under-load $\leadsto remove$

### Repair
- Failure $\leadsto repair$

![Diagram of Sizing and Repair processes with Tier T₁, load(s) & heartbeat(s), and requests connections.](image-url)
Motivating Example: The *Sizing-repair* Case

Tier-level Coordination

- Concurrency Management
  - Mutual Exclusions on “Accesses” to Shared (Actuators / Resources)
    - e.g., *add*, *repair*
  - Conflicting Behaviors
    - e.g., *remove*, *repair*
Motivating Example: The *Sizing-repair* Case

Sizing & Repair

load(s) & heartbeat(s) → add repair

requests → Tier T₁

Tier T₂ → add remove repair

load(s) & heartbeat(s)
Motivating Example: The *Sizing-repair* Case

**Inter-tier Coordination**

- Remote Impacts of Local Events, *e.g.*,  
  - Failure in Tier 1 $\leadsto$ *Transient* Under-load in Tier 2  
  - Failure in Tier 1 $\leadsto$ *Transient* (Under-load, then Over-load) in Tier 2  
  - Failure in Tier 2 $\leadsto$ *Transient* Load Variation in Tier 1

[Diagram showing inter-tier coordination with arrows indicating requests, add, remove, and repair actions between Tier 1 and Tier 2.]
Observation

- Systems’ Community: Realm of Object/Service/*-oriented Programming
  - Event-based: Handling One Event at a Time
  - Missing Global Knowledge
Coordinating AMSs’ Decisions

Observation

▶ Systems’ Community: Realm of Object/Service/*-oriented Programming
  ▶ Event-based: Handling One Event at a Time
  ▶ Missing Global Knowledge

Current Solutions...

▶ Heavy Use of Synchronization Primitives
  ▶ Tedious, Error-prone
▶ Very ad hoc...
  ▶ Hard to Generalize
Objective & Contributions

Objective

- Coordination-friendly Programming of AMSs

Proposal: Exploiting the Reactive Nature of AMSs

- New Design Methodology for AMSs
- Synchronous / Reactive Programming
- Discrete Controller Synthesis
- Prototype Implementation & Validation
Outline

- Context & Motivations
- Reactive Programming...
  - Synchronous Programming
  - Discrete Controller Synthesis
- ... for n-tier Applications Management
- Validation
- Summary
## Benefits

- **High-level Programming Languages**
  - *e.g.*, Automata
- **Efficient Verification Techniques**
  - Design-time Validation
- **Means for Static Property Enforcement**
  - Automatic Generation of *Minimally Restrictive Controllers*

\(\Rightarrow\) Over-constrained System
Synchronous Programming of Reactive Systems

Benefits

- High-level Programming Languages
  - e.g., Automata
- Efficient Verification Techniques
  - Design-time Validation
- Means for Static Property Enforcement
  - Automatic Generation of *Minimally Restrictive Controllers* ✗ Over-constrained System

Usual Design Technique

- Model-driven Programming
  - System Model
  - Expressing Decisions Based on this Knowledge
Example Synchronous Program

Data-flow Representation

\[
\text{Even} \quad \text{Even}
\]

\[
E \quad O
\]

\[
a \quad e \quad q
\]

\[
\text{Even} \quad \text{Even'}
\]

\[
\text{Q}
\]

\[
\text{Even} \times \text{Even'}
\]

Example Synchronous Program

Data-flow Representation

As a Product of Communicating Boolean Mealy Automata

\[ Q = \text{Even} \times \text{Even}' \]
Executing a Synchronous Program

Typical Execution Pattern

- **Inputs**
  - Impulses (≈ Discrete Events)
  - Measures (≈ Always Relevant Data)

- **Outputs**
  - Actions
Executing a Synchronous Program

Typical Execution Pattern

- **Inputs**
  - Impulses (≈ Discrete Events)
  - Measures (≈ Always Relevant Data)

- **“Using” the tick**
  - Decide Relevant Instants (Not Necessarily Periodic!)
  - Build an Input Vector \( I \)
  - Execute the tick.step() Method
  - Interpret Output Vector \( O \), Send Commands to the Actuators

![Diagram of the execution pattern](image-url)
Discrete Controller Synthesis Problem

Principle

- Behaviors $A_1 \cdots A_n$ (e.g., Automata)
- Property $\Phi$ (e.g., Mutual Exclusions, Avoiding Bad States)

\[ A_1 \parallel \cdots \parallel A_n \quad \Phi \]
Discrete Controller Synthesis Problem

**Principle**

- Behaviors $A_1 \cdots A_n$ (e.g., Automata)
- Property $\Phi$ (e.g., Mutual Exclusions, Avoiding Bad States)
- Provided $A_1 \cdots A_n$ are (made) Controllable ($\leadsto A_1' \cdots A_n'$)

$$A_1' \parallel \cdots \parallel A_n' \quad \Phi$$
Discrete Controller Synthesis Problem

**Principle**

- Behaviors $A_1 \cdots A_n$ (e.g., Automata)
- Property $\Phi$ (e.g., Mutual Exclusions, Avoiding Bad States)
- Provided $A_1 \cdots A_n$ are (made) **Controllable** ($\sim A_1' \cdots A_n'$)

$\sim$ Controller $C$ s.t.

$$A_1' \parallel \cdots \parallel A_n' \parallel C \models \Phi$$
Discrete Controller Synthesis Problem

Principle

- Behaviors $A_1 \cdots A_n$ (e.g., Automata)
- Property $\Phi$ (e.g., Mutual Exclusions, Avoiding Bad States)
- Provided $A_1 \cdots A_n$ are (made) **Controllable** ($\leadsto A_1' \cdots A_n'$)

$\leadsto$ Controller $C$ s.t.

$$A_1' \parallel \cdots \parallel A_n' \parallel C \models \Phi$$

Available Tool

- In the Synchronous Languages Domain: Sigali (until now...
Outline

- Context & Motivations
- Reactive Programming...
- ... for \textit{n}-tier Applications Management
- Validation
- Summary
Principles

Driving the Managed System with *Resource Drivers*

- **Operating Code** Senses & Acts Upon the Resources
  - Example of Resources:
    - Tier of an Application, Sets of Virtual Machines...
    - Pieces of Sequential Code (*e.g.*, `add`)
  - Associated **Behavioral Model** Represents Operating States
    - *e.g.*, “Stopped”, “Running”, “Adding a node”...
Principles

Driving the Managed System with Resource Drivers

- **Operating Code** Senses & Acts Upon the Resources
  - Example of Resources:
    - Tier of an Application, Sets of Virtual Machines...
    - Pieces of Sequential Code (e.g., *add*)
  - Associated **Behavioral Model** Represents Operating States
    - e.g., “Stopped”, “Running”, “Adding a node”...

Autonomic Management Strategies & Coordination Policies

- Expressed in a *Declarative Way*
  - e.g., Automata, Systems of Equations
  - Possibly Involving Costs
- Coordination Policies Enforced by using Discrete Controller Synthesis
Encoding the AMS Logic for an $n$-tier Application

Inter-tier Coordination

Tier-level Coordination

Independent Autonomic Managers

Behavioral Models

Operating Code

Tiers
Structure of a Resource Driver

Behavioral Model ($M_i$) (automata)

High-level Requests ↔ High-level Information
Low-level Notifications ↔ Low-level Commands

Operating Code (sensing & acting)

Monitoring Data $T_i$ Instructions

Command Part
Operative Part
Behavioral Model for a Tier

\[ \text{nodes-added/repair} \]

\[ \text{starting-up-node} \]

\[ \text{repairing-node} \]

\[ \text{#alive} \]

\[ \text{#failures} \]

\[ \text{load-avg} \]

\[ \text{max-nodes} \]

\[ \text{min-nodes} \]
Expressing Autonomic Management Strategies
Example Autonomic Management Strategies

**Repair Strategy**

\[
\text{repair} = (\#\text{failures} = 1 \land \text{repairing-node}) \lor \#\text{failures} > 1
\]

**Sizing Strategy**

\[
\text{up} = \text{max-nodes} \land \text{upper-threshold}(\text{load-avg}, \#\text{alive}) \\
\text{down} = \text{min-nodes} \land \text{lower-threshold}(\text{load-avg}, \#\text{alive})
\]

- Could be Less Trivial Synchronous Programs...
Specifying Coordination Policies

Inter-tier Coordination

Tier-level Coordination

Independent Autonomic Managers

Behavioral Models

Operating Code

Tiers
Example Tier-level Coordination Property

Allow Repair Operation (\textit{repair}) While Up-sizing (\textit{starting-up-node}), or Up-sizing Operation (\textit{add}) While Repairing (\textit{repairing-node})

In Case of Multiple "Recent" Failures Only (#\textit{failures} > 1)

Implementation

- Enforce both predicates:

  \[
  (\textit{starting-up-node} \land \textit{repair}) \Rightarrow \#\textit{failures} > 1 \\
  (\textit{repairing-node} \land \textit{add}) \Rightarrow \#\textit{failures} > 1
  \]
Example Inter-Tier Coordination Property

For Each Tier $T_i$,
Avoid Down-sizing Operations ($remove_i$)
if any of the “Preceding” Tiers $T_j$ both:

- “Recently” Underwent at Least One Failure ($\#failures_j > 0$)
- Is Currently Repairing One Node ($repairing-node_j$)

Implementation

Enforce the following predicate, $\forall i \in [2, n]$:

$$\left( \bigvee_{j=1}^{i-1} (repairing-node_j \lor \#failures_j > 0) \right) \Rightarrow remove_i$$
Filtering High-level Requests

\[\begin{align*}
\text{repair}_{\text{req}} &= \text{ok}_{\text{repair}} \land \text{repair} \\
\text{up}_{\text{req}} &= \text{ok}_{\text{sizing}} \land \text{up} \\
\text{down}_{\text{req}} &= \text{ok}_{\text{sizing}} \land \text{down}
\end{align*}\]

- High-level Requests: \( \{ \text{repair}, \text{up}, \text{down} \} \)
- Controllables: \( \{ \text{ok}_{\text{repair}}, \text{ok}_{\text{sizing}} \} \)
- Inputs of the Behavioral Model: \( \{ \text{repair}_{\text{req}}, \text{up}_{\text{req}}, \text{down}_{\text{req}} \} \)
Outline

- Context & Motivations
- Reactive Programming...
- ... for n-tier Applications Management

- Validation
  - Prototype Implementation
  - Validation

- Summary
Implementation Choices

Implementing the tick

- Using BZR
  - Encapsulating Sigali

Distributed Execution Platform: A³

- Message-oriented Middleware (Agent-based)
- Fault-tolerant
- Atomic Executions
- Dynamic Deployments
  - Allows Adding & Removing Nodes (e.g., Virtual Machines)
Experimental Setup

<table>
<thead>
<tr>
<th>Tier</th>
<th>Repair</th>
<th>Sizing</th>
<th>available VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{apache}}$</td>
<td>✓</td>
<td>✗</td>
<td>1</td>
</tr>
<tr>
<td>$T_{\text{tomcat}}$</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
</tr>
<tr>
<td>$T_{\text{mysql-proxy}}$</td>
<td>✓</td>
<td>✗</td>
<td>1</td>
</tr>
<tr>
<td>$T_{\text{mysql}}$</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
</tr>
</tbody>
</table>

- Synthetic Workload (based on Rubis’ Auction Site)
- 9 VMWare Virtual Machines
- Workload Injection from a Separate Physical Machine
- Collect Exponentially-Weighted-Moving-Average (EWMA) of CPU Utilization Percentages
## Validation

### Experimental Setup

<table>
<thead>
<tr>
<th>Tier</th>
<th>Repair</th>
<th>Sizing</th>
<th>available VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{apache}}$</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
</tr>
<tr>
<td>$T_{\text{tomcat}}$</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
</tr>
<tr>
<td>$T_{\text{mysql-proxy}}$</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
</tr>
<tr>
<td>$T_{\text{mysql}}$</td>
<td>✓</td>
<td>✓</td>
<td>3</td>
</tr>
</tbody>
</table>

- Synthetic Workload (based on Rubis’ Auction Site)
- 9 VMWare Virtual Machines
- Workload Injection from a Separate Physical Machine
- Collect Exponentially-Weighted-Moving-Average (EWMA) of CPU Utilization Percentages

Comparing Non-coordinated & Coordinated Executions
Non-coordinated Execution

CPU utilization (EWMA %) / time (s)

- apache
- tomcat-1
- tomcat-2
- tomcat-3
- mysql-proxy
- mysql
- mysql-2
- mysql-3
Coordinated Execution

CPU utilization (EWMA %) / time (s)

- apache
- tomcat-1
- tomcat-2
- tomcat-3
- mysql-proxy
- mysql-1
- mysql-2
- mysql-3

failure
Outline

- Context & Motivations
- Reactive Programming...
- ... for $n$-tier Applications Management
- Validation
- Summary
Summary of the 1st Part

Objective

- Coordination-friendly Programming of AMSs

Proposal: Exploiting the Reactive Nature of AMSs

- New Design Methodology for AMSs
- Synchronous / Reactive Programming
- Discrete Controller Synthesis
- Prototype Implementation & Validation
Future Works

Quantitative Aspects

- Extending Behavioral Models with Relevant Quantitative Data
  - Enforce Quantitative Properties
    - *cf.* Next Part...

Distributed Execution of the AMS / Controller

- Centralized Design (Scalability Issue)
  \[ \sim \text{Synchronous Languages Implementation Problem} \]
- Decentralized Design (Modeling Issue)
  \[ \sim \text{Decentralized Control Problem} \]
Thanks!

Questions?
Outline

- Context & Motivations

- Reactive Programming...
  - Synchronous Programming
  - Discrete Controller Synthesis

- ... for n-tier Applications Management

- Validation
  - Prototype Implementation
  - Validation

- Summary
Outline

- Architecture of the Prototype
Architecture of the Prototype

center

M1 M2
T1 T2

S1 A1
A2 S2
A3 S3

tomcat
tomcat

apache