Ceylan: A framework for building extensible and dynamic autonomic managers

Yoann Maurel

JURY

Pr. Gaëlle CALVARY, Présidente, Grenoble INP
Dr. François CHARPILLET, Rapporteur, Inria
Dr. Roy STERRITT, Rapporteur, University of Ulster
Dr. Julie McCANN, Examinatrice, Imperial College London
Pr. Philippe LALANDA, Directeur, UJF Grenoble
Dr. Ada DIACONESCU, Co-directrice, Telecom Paris-Tech
A story of evolution

- A long time ago, in a galaxy not so far away
  - Software was easy to maintain
A story of evolution

- And then software became
  - Bigger and bigger
  - Interdependent

Administrator
A story of evolution

- And then software became
  - Bigger and bigger
  - Interdependent
  - Distributed, heterogeneous, dynamic
  - Complex and unmaintanable
Complexity is everywhere!
Solution? Autonomic Computing

"Lower CPU consumption"
**Problem:** How to produce such managers?

This is the purpose of this work

We propose a framework to build such systems
Outline

1. Autonomic Computing
2. Proposition
3. CEYLAN’s architecture
4. Validation
5. Conclusion
1. Autonomic Computing

1. Definition
2. Autonomic system’s architecture
Autonomic Computing

IBM 2001: “We should build self-managed systems!”

- Awaited benefits:
  - reduce maintenance costs
  - reduce operators’ errors
  - optimize resources’ allocation
  - personalized services to users

In short, assist human to cope with complexity
Multiple influences

Bio(socio)-inspired algorithms

Biology

Robotic and automatic systems

Software Engineering

Artificial Intelligence and Multi-agent system

AUTONOMIC COMPUTING

Ubiquitous Computing
Ambient Intelligence
Proactive Computing
Definition: Autonomic computing

- « The essence of autonomic computing systems is self-management, ... free system administrators from the details ... provide users with a machine that runs at peak performance 24/7 » [Kephart]
- «...realize computer and software systems and applications that can manage themselves in accordance with high-level guidance from humans » [Parashar]
- « The aim [is to] create the self-management of a substantial amount of computing function to relieve users of low level management activities ... The vision of autonomic computing is to create selfware through self-* properties» [Sterritt]

Autonomic computing is the field interested in all adaptive technologies to provide approaches and tools for building self-managed systems, called autonomic systems
An autonomic system has sufficient information on its environment and itself to be able to anticipate the situations they encounter and to adapt to it in order to deliver its services in the best possible way with respect to the high-level policies fixed by the administrator.

- Self-Configuration
- Self-Optimisation
- Self-Healing
- Self-Protection
- Self-...
Building an autonomic system is hard (1/2)

- Absorb complexity
- Deal with different goals
  - Resource management
  - Self-healing
  - Self-optimization
  - Business specific goals...
- Dynamic and fluctuating environments
  - Communication with other managers
  - Managed element modifications
Building an autonomic system is hard (2/2)

- They should be
  - Reliable
  - Maintainable
  - Administrable
  - Adaptable
  - Extensible

As most systems ... may be more than others!
Outline

1. Autonomic Computing
   1. Definition
   2. Autonomic system’s architecture
2. Proposition
3. CEYLAN’s architecture
4. Validation
5. Conclusion
The control loop

1. Monitoring
2. Decision
3. Action

“Lower CPU consumption”
An autonomic element consist in
- One or more managed elements
- Touchpoints
- An autonomic manager

Generally accepted structure of autonomic systems

- Autonomic Manager
- Goals
- Feedback
- Sensors
- Effectors
Organisation and choices

- Management architecture
  - One or many autonomic subsystems?
  - Centralized/Distributed/Hierarchical

- Manager architecture
  - Monolithic/modular

- Technologies
  - Rules, component, services...
Monolithic

- Black (red!) box: rules or code
Modular

IBM’s MAPE-K

Parashar, Hariri...

MAPE-K like
only general principles - a design pattern, insufficient in itself.
Chain of responsibilities

Mediation Framework, Monitoring frameworks...

Autonomic Manager

Goals

Managed Element

sensors

effectors

Yoann Maurel - PhD Defence
# Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Internal Architecture</th>
<th>Management Architecture</th>
<th>Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rainbow</td>
<td>Modular (MAPE-K like)</td>
<td>Centralized</td>
<td>Rules</td>
</tr>
<tr>
<td>Jade</td>
<td>Modular (MAPE-K)</td>
<td>Centralized</td>
<td>Code</td>
</tr>
<tr>
<td>Unity</td>
<td>Modular (MAPE-K)</td>
<td>Distributed</td>
<td>Utility function</td>
</tr>
<tr>
<td>Autonomia</td>
<td>Modular static (MAPE-K)</td>
<td>Distributed</td>
<td>Rules</td>
</tr>
<tr>
<td>Automate</td>
<td>Modular</td>
<td>Distributed</td>
<td>Component(Agent)/Rules</td>
</tr>
<tr>
<td>Auto-Home</td>
<td>Monolithic</td>
<td>Hierarchical</td>
<td>Code</td>
</tr>
</tbody>
</table>
## Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Internal Architecture</th>
<th>Management Architecture</th>
<th>Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rainbow</td>
<td>Modular (MAPE-K like)</td>
<td>Centralized</td>
<td>Rules</td>
</tr>
<tr>
<td>Jade</td>
<td>Modular (MAPE-K)</td>
<td>Centralized</td>
<td>Code</td>
</tr>
<tr>
<td>Unity</td>
<td>Modular (MAPE-K)</td>
<td>Distributed</td>
<td>Utility function</td>
</tr>
<tr>
<td>Autonomia</td>
<td>Modular static (MAPE-K)</td>
<td>Distributed</td>
<td>Rules</td>
</tr>
<tr>
<td>Automate</td>
<td>Modular</td>
<td>Distributed</td>
<td>Component(Agent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/Rules</td>
</tr>
<tr>
<td>Auto-Home</td>
<td>Monolithic</td>
<td>Hierarchical</td>
<td>Code</td>
</tr>
</tbody>
</table>
Conclusion: Current framework and architecture

- Excellent logical architecture (MAPE-K)

However

- Sometimes hard to implement
  - too constraining
  - do not separate different concerns/goals
  - coarse-grained architectural blocks
  - frequently implemented as rules or coarse-grained components with negative consequences on maintenance or poor dynamism/reusability

- Hard to extend
- Hard to reuse
Limitation

- **Dynamicity:**
  - The possibility of changing dynamically the internal architecture of managers is often ignored
  - MAPE-K does not provide guidance on how to manage dynamism between modules
  - Rules: hard to predict/understand/maintain (fine-grained)
Proposition
Goals

Ease the development, maintenance and evolution of autonomic managers

- Building a framework that
  - Supports **reusing** of common/redundant functionalities
  - Provides a **homogeneous** and **dynamic model** for the integration of autonomic functions
  - Promotes the **reuse** of autonomic functions
  - Allows different management concerns to be implemented in **isolation**
  - Facilitates the **evolution** and **extension** of manager’s behaviour at runtime
Proposition

- Decompose the manager behaviors into **elementary activities**: administration tasks
Proposition

- Decompose the manager behaviors into elementary activities: administration tasks
- Each path = one control loop
Example: CPU Management

- A: CPU monitoring
- B: Average CPU usage
- C: Threshold detection
- D: Decision (Switch to CPU friendly algorithm)
- E: Apply decision
Dynamic opportunistic cooperation

- Composition is **opportunistic** and depends on the **runtime** context
Dynamic opportunistic cooperation

- Composition is **opportunistic** and depends on the **runtime** context
Dynamic opportunistic cooperation

- Composition is **opportunistic** and depends on the **runtime context**
Dynamic opportunistic cooperation

- Composition is **opportunistic** and depends on the **runtime** context
Extensibility/Adaptation

- Task can be removed or deployed/added new task anytime
Non-functional aspects

- A task should be easy to develop
  - The framework should deal with non-functional concerns
- A task must be
  - Manageable
  - Discoverable and deployable at runtime
  - Dynamically activable/de-activable at runtime
  - Reusable
    - standard interfaces.
Framework - required functionalities

Integration of tasks:
- Coordination:
  - communication
  - synchronization of tasks: data, activation
- Conflict management:
  - avoid execution of competing tasks

Administration/Management
- lifecycle (discovery, installation, activations...)
- monitoring (number of executions, state, execution history)
- failure detection (one task is blocked)
- User interfaces

Dynamism (repository, deployment at runtime)

Construction
Control of the tasks

- A dedicated controller manages tasks lifecycle, conflicts, communication

**Management of the Task (controller)**

**Managed System**
Control of the tasks

- A dedicated controller manages tasks lifecycle, conflicts, communication
Construction and adaptation

- Administration
  - Creation
  - Modifications
  - Managed System
  - Runtime context

Management of the Task (controller)
Construction and adaptation

Administrator/expert

High-level goals

Self-Adaptation

Targeted manager (ADL)

HMI / ADL

Runtime context

Creation Modifications

Management of the Task (controller)
CEYLAN’s Architecture

1. Task architecture
2. Control
3. Construction/Adaptation
Administration Tasks in detail

- Elementary behavior (specialized algorithm)
  - monitoring some parameter
  - detecting crossing of a threshold
  - inferring a value
Tasks modular architecture

- Separation of concerns
  - Communication
  - Activation
  - Conflict Management
  - Statistic
  - Functionality specification (type) / implementation

![Diagram of modular architecture]
Ports: Communication

- Collects and produces data:
  - Data are described in a data model
  - Dictionary of values with a unique name
Collects and produces data:
- Data are described in a data model
  - Dictionary of values with a unique name
Scheduler: tasks triggering

- Triggering conditions depend on context, including:
  - Nature/quantity/values of collected data
  - Time interval/certain date/last activation
  - Shared information with other tasks

![Diagram of Scheduler and Tasks]

Task is waiting
Scheduler: tasks triggering

- Triggering conditions depend on context, including:
  - Nature/quantity/values of collected data
  - Time interval/certain date/last activation
  - Shared information with other tasks
Coordinator: Conflict Management

- Obtain the necessary authorization to handle data
  - Part of the conflict management mechanism
  - Only used for potentially conflicting data types
  - E.g. Compression task vs Deletion task

Task is ready

Diagram:
- Input Port
- Scheduler
- Coordinator
- Processor
- Output Port
- Statistics
- Selection Mechanism
Coordinator: Conflict Management

- Obtain the necessary authorization to handle data
  - Part of the conflict management mechanism
  - Only used for potentially conflicting data types
  - E.g. Compression task vs Deletion task

Task is **activated**
Processor: Business code

- Specialized (business) algorithm
Statistics

- Statistics collection for task evaluation:
  - Number of task executions
  - Quality and type of data processed/produced
  - Average task execution task
  - Lifecycle information (blocked task...)

Yoann Maurel - PhD Defence
Administration

- Manage task state and lifecycle information
Task Lifecycle

- Started
- Stopped
- Configured
- Invalid
- Destroyed
- One provider for each required data type
- One data type provider missing
- Valid
- Blocked
- Active
- Waiting
- Task is processing
- Request has been processed
- This activity is considered suspect

Yoann Maurel - PhD Defence
Outline

1. Autonomic Computing
2. Proposition
3. CEYLAN’s architecture
   1. Task architecture
   2. Control
   3. Construction/Adaptation
4. Validation
5. Conclusion
Control of the tasks

- A dedicated controller manages tasks lifecycle, conflicts, communication
Conflict Management

- Several approaches
  - By synthesis
  - By filtering
  - By arbitration
  - By negotiation

- To bring flexibility
Conflict Management

- Several approaches
  - By synthesis
  - By filtering
  - By arbitration
  - By negotiation

- To bring flexibility

After execution
Before execution
Use of specialized task to aggregate/select data
Conflict Management: By arbitration

- A special component decides on the task to be activated on which data
Conflict Management: By arbitration

- A specialized components decides which task can activate on which data
A specialized components decides which task can activate on which data
A specialized components decides which task can activate on which data
Conflict Management: By arbitration

- Implementation can be changed dynamically

Token based

Controller

Managed System

T1

T2

T3

Election

ARBITER

Task

Managed System
Conflict Management: By arbitration

- Implementation can be changed dynamically

Priority based

Managed System

T1
T2
T3
Conflict Management: By arbitration

- Implementation can be changed dynamically
Outline

1. Autonomic Computing
2. Proposition
3. CEYLAN’s architecture
   1. Task architecture
   2. Control
   3. Construction/Adaptation
4. Validation
5. Conclusion
Construction

- **At design time** the manager is described in terms of task types
  - Describing task functionality
  - Fully independent from implementation

- **At runtime**
  - Administrator modifies targeted manager model if necessary
  - Construction module is responsible for instantiating this model
    - Implementation discovery
Construction

Targeted Manager (ADL)

Construction

Model@ Runtime

Create Destroy Reconfigure

Controller

T1 → T2 → T3 → T4 → T5

Becomes invalid

Managed System

Type and data models

Discover Implementation

Compare Implementation

Implementation Repository

Yoann Maurel - PhD Defence
Adaptation

Controller Managed System

becomes invalid

Construction
Self-adaptation GUI/HMI administrator

Managed System
Validation

1. CEYLAN’s implementation
2. Application
In short

- Fully functional implementation
  - 10000 lines of java code + 6000 for HMI
- Extensible implementation of each modules
  - Scheduler, port, arbiter...
  - Basis for new implementation
Service-Oriented Component Model

- Autonomic Managers
- CEYLAN
- CILIA
- IPOJO
- OSGi
- JAVA

- Component model
- Dependencies management
- Service Architecture
- Multithreading Management
XML based ADL

ATOMIC TASK DECLARATION (PLATFORM INDEPENDANT)
<task type="Plan.Camera" implementation-name="CameraTasks">
    ... Port
Handler
scheduler 
Handler
Input Port 
Handler
Coordinator 
Handler
statistics 
Handler
admin
</task>

Architecture
Implementation

Yoann Maurel - PhD Defence
Administration HMI
Administration HMI
Administration HMI

- Assisted creation of tasks, task types and data types
Outline

- Autonomic Computing
- Proposition
- CEYLAN’s architecture
- Validation
  - CEYLAN’s implementation
  - Application
- Conclusion
Intrusion Monitoring Application

![Diagram of Intrusion Monitoring Application]

- **OSGI Gateway**
  - Storage
  - Communication
  - Image capture
  - Movement detection
  - Alarm

- Areas:
  - Garden
  - Hall
  - Living Room
Various objectives

- Goals
  - Disk Management
  - CPU Management
  - Battery Management
  - Calibration,…

- Progressive construction of the manager by integration of the different concerns
  - Implementation in isolation
Non Managed Application

Disk/CPU Usage vs Time (s)

- Disk is not managed
- Accurate detector = CPU

ALARM
CPU+Disk Management

- Goal:
  - Use less CPU if possible
    - CPU friendly detector
  - Delete old images from storage
Goal:
- Use less CPU if possible
  - CPU friendly detector
- Delete old images from storage
Problem: blocked tasks

- Solution:
  - Replace blocked tasks
Auto-replaced blocked tasks

Faulty implementation

New Implementation

CPU

Disk

Erase

timeout = blocked

substitution

Disk1

Thresholds

Time (s)

Disk/CPU Usage

0 50 100 150 200 250 300

0 10 20 30 40 50 60

90% 70% 50% 30% 10%
We try to use compression

- **Goal:**
  - Use *compression* when effective, *erase* otherwise

- **Solution**
  - Token based arbiter
Arbiter

Intrusion Monitoring

tokens consumption

CPU

Disk

Erase

Compress

Time (s)
More complex behavior

- Goal:
  - Battery management,
  - Alarm management,
  - CPU management,
  - Cameras...
Combination

![Graph showing disk and CPU usage over time with various thresholds and events marked.]
Synthesis

- Complex and extensible behavior via the combination of tasks
- Implementation of the different concerns in isolation
- Conflict management via arbitration

- 20-30% slower than ad hoc MAPE-K
  - But dynamic, extensible at runtime
Conclusion
In Short

- We introduce a new level of abstraction: task

Manager

MAPE like (activities centered)

Monitor

Analyze

Plan

Execute

Task

(function centered)

Code/rules

System.out.println()

If (Foo) then BAR
In Short

- **Flexibility**

Manager

Composite Task (activities centered)

Task (function centered)

Code/rules

System.out.println()

If (Foo) then BAR

Yoann Maurel - PhD Defence
In Short

- Flexibility

Manager

Composite Task (activities centered)

Task (function centered)

Code/rules

Manager

System.out.println()

If (Foo) then BAR
In Short

- Flexibility + Adaptation + Dynamicity + Opportunism

Manager

Composite Task (activities centered)

Task (function centered)

Code/rules

System.out.println()

If (Foo) then BAR

Yoann Maurel - PhD Defence
Conclusion

- Supports **reusing** of common/redundant functionalities
  - Reusable non-functional tasks modules (scheduler, coordinator, port)

- Provides a **homogeneous and dynamic model** for the integration of autonomic functions
  - Tasks have standard interfaces
  - Separation Specification/Implementation

- Promotes the **reuse** of autonomic functions
  - Specialized algorithm
  - Adequate granularity
Conclusion

- Allows different management concerns to be implemented in isolation
  - One concern = one set of tasks
  - Composite tasks

- Facilitates the *evolution* and *extension* of manager’s behaviour at runtime
  - Task can be discovered/installed/removed dynamically at runtime
  - Layered architecture to ease administration
Perspective: IDE

Administrator/expert

High-level goals

Creation

Modifications

Self-Adaptation

HMI / ADL

Runtime context

Targeted manager (ADL)

Management of the Task (controller)
Perspective: Self Adaptation

Administrator/expert

High-level goals

Targeted manager (ADL)

Self-Adaptation

HMI / ADL

Creation Modifications

Runtime context

Management of the Task (controller)
Management of the Task (controller)

Managed System
Questions?