Towards Building Adaptable Autonomic Managers
MAASC 2011

Yoann Maurel – University of Grenoble
Challenge: building managers

Admin

Goals

Manager

Managed System

Yoann Maurel – LIG/ADELE
Building manager is hard!

MULTIPLE AND/OR CHANGING:

Goals and Concerns (Self-*, Business goals, ...)

Manager

Managed System
Building manager is hard!

MULTIPLE AND/OR CHANGING:

Goals and Concerns (Self-*, Business goals, ...)

Manager

Admin

Managed System

Management Situations
Building manager is hard!

MULTIPLE AND/OR CHANGING:

Goals and Concerns (Self-*, Business goals, ...)

Activities and management actions (Monitoring, Learning, finding patterns, inferring, ...)

Management Situations

Managed System

Yoann Maurel – LIG/ADELE
Building manager is hard!

MULTIPLE AND/OR CHANGING:

Goals and Concerns (Self-*, Business goals, ...)

Activities and management actions (Monitoring, Learning, finding patterns, inferring, ...)

Management Situations

Environment (update, new server, service, ...)

Yoann Maurel – LIG/ADELE
Building an autonomic managers is hard

- Flexibility/adaptability is required:
  - multitude of possible conditions and corresponding management actions
  - System’s evolution

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

As most systems ... may be more than others!
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)
CEYLAN : A framework for building adaptable and extensible autonomic managers
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

Managed System

disable
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: Problem detection (Threshold)
- D: Decision (Switch to CPU friendly algorithm)
- E: Apply decision

Managed System
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)
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

Management of the Task (controller)

Runtime context

Managed System

Yoann Maurel - LIG/ADELE
Construction and adaptation

Administrator/expert

High-level goals

Self-Adaptation

HMI / ADL

Targeted manager (ADL)

Creation Modifications

Runtime context

Management of the Task (controller)

CONTEXT
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 (Input/Output Ports)
  - Activation (Scheduler)
  - Conflict Management (Coordinator)
  - Statistic
  - Functionality specification (type) / implementation (Processor)
  - Administration
Task Lifecycle

Task states:
- Configured
- Valid
  - Active
  - Waiting
- Invalid
  - One data type provider missing
  - One provider for each required data type
- Destroyed

Transitions:
- Configured → Started
- Configured → Stopped
- Invalid → Valid
  - Task is processing
- Valid → Waiting
  - request has been processed
- Valid → Active
- Valid → Blocked
  - task's activity is considered suspect
- Valid → Destroyed
Outline

1. Proposition
2. CEYLAN’s architecture
   1. Task architecture
   2. Control
   3. Construction/Adaptation
3. Validation
4. 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: 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
Implementation can be changed dynamically
Implementation can be changed dynamically
Conflict Management: By arbitration

- Implementation can be changed dynamically
Outline

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

Model@ Runtime

Create Destroy Reconfigure

Construction

Type and data models

Discover Implementation

Compare Implementation

C

Implementation Repository

Controller

Managed System

T1 T2 T3 T4 T5

Becomes invalid

Yoann Maurel - LIG/ADELE
AdaptaMon

Yoann Maurel - LIG/ADELE
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

- **CEYLAN**: Component model
- **CILIA**: Dependencies management
- **IPOJO**: Service Architecture
- **OSGi**: Multi threading Management
- **JAVA**: Autonomic Managers
XML based ADL

Architecture
Implementation

<task type="Plan. Camera implementation-name="CameraTasks">
  <input-port ref="ceylan-direct-input">[...]</input-port>
  <scheduler ref="ceylan-scheduler-base">[...]</scheduler>
  <coordinator ref="ceylan-direct-output">[...]</coordinator>
  <processor ref="hello-world">[...]</processor>
  <output-port ref="ceylan-direct-output">[...]</output-port>
</task>
Administration HMI

Graphical interface for Keyton

Yoann Maurel - LIG/ADELE
Administration HMI

Graphical Interface for Coyote

Task tree

Scheduler

Scheduler Debug

Information

Create an RMI connector client and connect it to the RMI connector server.

Get an RMIClientConnection

Add a notification listener...

GetEvent

Organize Task Trees with drag & drop.

Yoann Maurel - LIG/ADELE
Administration HMI

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

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

OSGi Gateway

Storage

Image capture

Movement detection

Communication

Alarm

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 is not managed
accurate detector = CPU

Yoann Maurel - LIG/ADELE
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
CPU+Disk Management

![Graph showing CPU and Disk usage over time with thresholds marked on the y-axis and time in seconds on the x-axis. The graph includes lines for CPU, Disk, Alarm, Erase, and Detector with specific points marked for monitoring.]
Problem: blocked tasks

Solution:
- Replace blocked tasks
Auto-replaced blocked tasks

Disk/CPU Usage

Faulty implementation

New Implementation

CPU

Disk

Erase

CPU

Disk1

Timeout = blocked

substitution

Time (s)
Replace malfunctioning tasks

- Solution:
  - Replace malfunctioning tasks
Replace malfunctioning tasks

- Too many consecutive executions
- The task is efficient
- Replacement by the administrator
- Replacement by the system
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

Disk/CPU Usage

Time (s)
More complex behavior

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

Diagram showing various components and their interconnections labeled as 'A', 'M', 'P', 'E', with 'Intrusion Monitoring' at the bottom.
Combination

![Graph showing disk and CPU usage over time with labels for different activities like CPU, Alarm, Selection, Erase, Compress zip, Camera, Detector.]
Synthesis

- Complex and extensible behavior via the combination of tasks
- Implementation of the different concerns in isolation
- Conflict management via arbitration
We introduce a new level of abstraction: task
In Short

- **Flexibility**

Manager

Composite Task (activities centered)

Task (function centered)

Manager

Code/rules

System.out.println()

If (Foo) then BAR

Yoann Maurel - LIG/ADELE
In Short

- **Flexibility**

Manager

Composite Task (activities centered)

Task (function centered)

Code/rules

Manager

System.out.println()

If (Foo) then BAR

Yoann Maurel - LIG/ADELE
In Short

- Flexibility + Adaptation + Dynamicity + Opportunism

Manager

Composite Task (activities centered)

Task (function centered)

Manager

Code/rules

System.out.println()

If (Foo) then BAR

Yoann Maurel - LIG/ADELE
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
Questions?