A Non-Intrusive Component-Based Approach for Deploying Unanticipated Self-Management Behaviour

Sandro Santos Andrade
Raimundo José de Araújo Macêdo
{sandros, macedo}@ufba.br

Federal University of Bahia - Brazil
Department of Computer Science (DCC)
Distributed Systems Laboratory (LaSiD)
Multiinstitutional Doctorate in Computer Science (DMCC)
Motivation

- Architecture-based approaches have proven to be useful for early rationale of non-functional requirements
- Increasing complexity demands the use of fully automatic self-managed software platforms
- Underlying runtime support environment can provide self-management behaviour to already deployed systems (non-intrusivenessness)
- Demand for the support of unanticipated changes
Motivation

- Aspects of dynamic software architectures:
  - Underlying component model
  - Degree of autonomicity
  - Planning and high-level reasoning
  - Runtime support environment
  - Consistency and system integrity
Motivation

- Aspects of dynamic software architectures:
  - Underlying component model
  - Degree of autonomicity
  - Planning and high-level reasoning
  - Runtime support environment
  - Consistency and system integrity

No assumptions apart from connectable ports and configurable attributes
Motivation

- Aspects of dynamic software architectures:
  - Underlying component model
  - **Degree of autonomicity**
  - Planning and high-level reasoning
  - Runtime support environment
  - Consistency and system integrity

Fully automatic dynamic adaptation by means of component and architecture reconfiguration
Motivation

- Aspects of dynamic software architectures:
  - Underlying component model
  - Degree of autonomicity
  - Planning and high-level reasoning
  - Runtime support environment
  - Consistency and system integrity

We assume the use of a dynamic reconfigurable component-oriented middleware
Motivation

- Aspects of dynamic software architectures:
  - Underlying component model
  - Degree of autonomicity
  - Planning and high-level reasoning
  - Runtime support environment
  - Consistency and system integrity

We rely on basic middleware guarantees
Motivation

Aspects of dynamic software architectures:

- Underlying component model
- Degree of autonomicity
- Planning and high-level reasoning
- Runtime support environment
- Consistency and system integrity

Not addressed at all
The Proposed Framework

Our goal:

- To provide self-management behaviour in already deployed component-based systems, in a non-intrusive approach

Characteristics:

- Reusable core of component-based services for autonomic control loops
- Building blocks for definition of adaptation policies and architectural changes
- Hooks (hot-spots) for the connection of developer-supplied components for environment monitoring
- The adaptation model itself is designed and deployed as a component configuration, which in turn can undergo adaptation
Our approach:

1. periodic evaluation of the i-th adaptation policy
2. gathering of data from environment monitoring devices
3. if policy evaluation results true submit changes to reconfiguration service
4. redeploy updated configuration

Architecture Reconfiguration Service
Common Services
Runtime Support Environment
Network Communication Sub-System
The Proposed Framework

Framework architecture:
The Proposed Framework

1. Adaptation module:
The Proposed Framework

1. Adaptation module:
The Proposed Framework

1 Adaptation module:
The Proposed Framework

1 Adaptation module:

[Diagram showing the proposed framework with classes and interfaces: Redeployment Module, Adaptation Manager, Environment Module, Adaptation Policy 1, Adaptation Policy 2, Adaptation Policy n, IRedeploymentManager, IEnvironmentManager, Deployment plan UUID, IAdaptationPolicy (multiple).]
Adaptation module (*policies*):
1. Adaptation module (*rules*):
The Proposed Framework

1. Adaptation module (*rules*):

```
Adaptation Policy 1

Evaluation rate

IAdaptationAction (multiple)

Adaptation Rule

Device name (null)
Relational operator (null)

Value (null)
Logical Operators (or)

Packet Loss > 0.8 null

cpu Usage > 0.9 null
MemoryUsage > 0.5 null

Tree-based component configuration for adaptive rule: (PacketLoss > 0.8 or (CpuUsage > 0.9 and MemoryUsage > 0.5))
```
The Proposed Framework

1. Adaptation module (*actions*):
Environment module:

- Environment monitoring components implement a specific interface (*Idevice*)
- Device name should match those used by adaptation policies
The Proposed Framework

Redeployment module:

- Provides the glue code for a specific runtime support environment
- *IRedeploymentService* define methods for handling the adaptation actions
Implementation Issues

- Fully implemented on top of CIAO (*Component-Integrated ACE ORB*)
- DAnCE
  - XML deployment descriptors
  - Remote deployment, remote connections, component repository
- ReDaC
  - Dynamic redeployment of architectural changes
  - Basic services for consistency maintenance
Deploying Self-Management

- Required steps:
  - Identify self-management requirements (adaptation rules and actions)
  - Identify required environment monitoring devices
  - Implement or reuse required environment monitoring devices
  - Create observer component configuration by composing the provided building blocks
  - Deploy observer system
ARCOS platform provides reusable services for interoperable industrial data acquisition, closed control loops, and supervisory activities.
ARCOS platform provides reusable services for interoperable industrial data acquisition, closed control loops, and supervisory activities.
Evaluation Experiments

- ARCOS platform provides reusable services for interoperable industrial data acquisition, closed control loops, and supervisory activities
Evaluation Experiments

1. Self-optimization
Evaluation Experiments

Self-healing

Fault Detection Device

Control Manager

PID Controller Replica 1
kp ki kd

PID Controller Replica 2

PID Controller Replica 3

Passive redundancy implementation in PID controller
Unanticipated Changes

- Unanticipated = unforeseen when creating the observer component configuration
- As the adaptation model itself is deployed as a component configuration it is able to undergo adaptation
- This can be done:
  - Manually: by requiring runtime changes in the observer component configuration
  - Automatically: by deploying another running instance of our framework
Unanticipated Changes

- Automatically: by deploying another running instance of our framework
Unanticipated Changes

- Automatically: by deploying another running instance of our framework
Unanticipated Changes

- Automatically: by deploying another running instance of our framework
Conclusion and Future Work

- Component-based representation of adaptation policies and adaptation changes
- Implemented framework for self-management of CCM-based applications

Future work:

- Experiments on unanticipated changes support
- Mechanism for handling conflicting adaptation policies
- Planning and high-level reasoning
- Self-management with real-time constraints
- Performance evaluation experiments
Questions?