A Generic Framework for the Engineering of Self-Adaptive and Self-Organising Systems

Giovanna Di Marzo Serugendo
dimarzo@dcs.bbk.ac.uk

Birkbeck College, University of London
http://www.dcs.bbk.ac.uk/~dimarzo

In Collaboration with:

University of Newcastle, University of Luxembourg

1

Outline

- Introduction
- Requirements Engineering
- Generic Framework
  - Design-time
  - Run-time
  - Control
- Applications
  - Self-Adaptive
  - Self-Organising
- Proofs of concepts

2

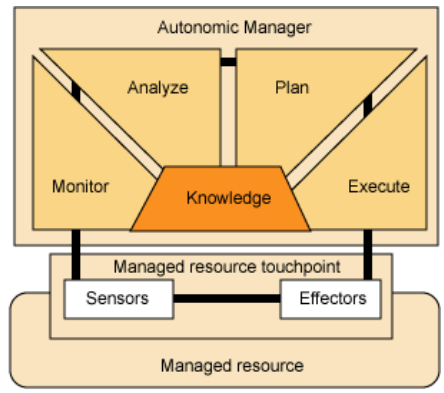
Self-* Systems

- Self-Adaptive
  - Top-down
  - Self-evaluation + Reconfigurable
  - Autonomic Computing
- Self-Organising
  - Bottom-up
  - Emergent properties

3

Self-* Systems

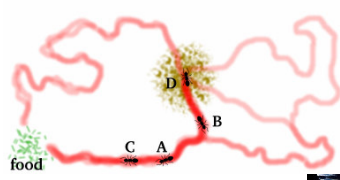
- Self-Adaptive (Autonomic Computing)



4

Self-* Systems

- Self-Organising



5

Goal

- Answer the question:

How to build trustworthy and controllable self-* systems?

- Dependability properties
- Evidence of dependability
- Human administrator controls system
  - Despite self-* capabilities of system

6

Requirements Engineering

- Autonomous Components
  - R: Decoupling of components
- Interoperability
  - R: Decoupling of descriptions, QoS, constraints, policies, supporting infrastructure

7

Requirements Engineering

- Self-awareness
  - R: Sensing/monitoring capabilities – Reasoning/Acting (different levels)
- Behaviour guiding/bounding
  - R: Individual and global goals/policies Environmental constraints policies

8

Requirements Engineering

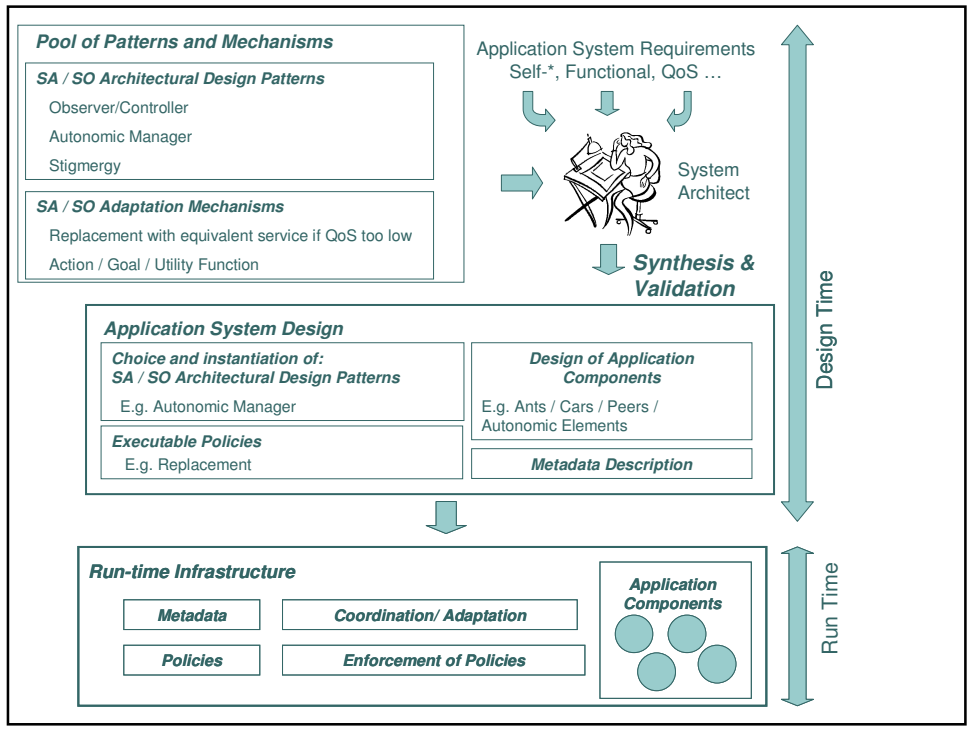
- Development process
  - R: Design-time descriptions
    - system's / components'propertie
    - behaviour patterns
    - policies
  - R: Run-time enforcement of policies

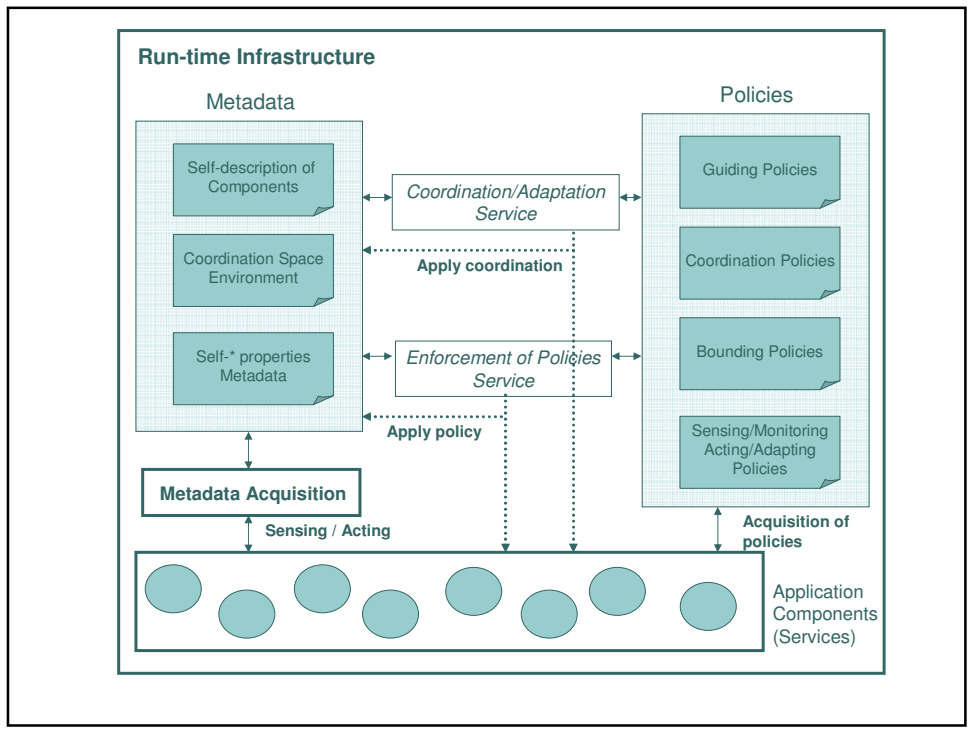
9

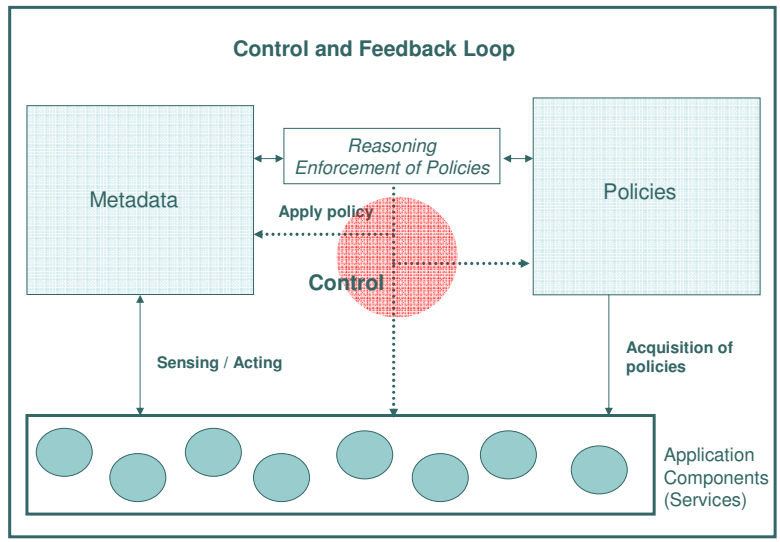
Generic Framework

- Service-oriented architecture
  - R: Autonomous components
- Self-describing Components/Services
  - R: Interoperability
- Run-time usage of Meta-Data
  - R: interoperability + self-awareness
- Adaptation Mechanism
  - R: design-time specification of adapting policies
- Executable Policies
  - R: run-time implementation of adaptation mechanism

10


Pool of Patterns and Mechanisms
SA / SO Architectural Design Patterns
Observer/Controller
Autonomic Manager
Stigmergy
SA / SO Adaptation Mechanisms
Replacement with equivalent service if QoS too low
Action / Goal / Utility Function
Application System Requirements
Self-*, Functional, QoS …
System Architect
Synthesis & Validation
Design Time
Application System Design
Choice and instantiation of:
SA / SO Architectural Design Patterns
E.g. Autonomic Manager
Design of Application Components
E.g. Ants / Cars / Peers / Autonomic Elements
Executable Policies
E.g. Replacement
Metadata Description
Run-time Infrastructure
Metadata
Coordination/ Adaptation
Policies
Enforcement of Policies
Application Components
Run Time


Run-time Infrastructure
Metadata
Policies
Self-description of Components
Coordination Space Environment
Self-* properties Metadata
Coordination/Adaptation Service
Apply coordination
Enforcement of Policies Service
Apply policy
Guiding Policies
Coordination Policies
Bounding Policies
Sensing/Monitoring Acting/Adapting Policies
Metadata Acquisition
Sensing / Acting
Acquisition of policies
Application Components (Services)



Control

- Three different ways:
  - Direct reconfiguration of components
    - Remove / change
  - Dynamic modification of metadata
    - Indirect: changed metadata implies changes in component behaviour
  - Dynamic modification of policies
    - Indirect: changed policy implies changes in component behaviour

14

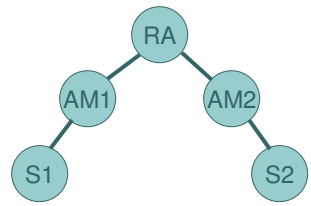
Applications

- Self-Adaptive
  - Data resource centre (Kephart et al.)
  - Two applications providing a service
    - Demand for service varies over time
    - Performance of application depends on demand and on resources allocated to the application
    - Service level agreement (QoS provided to consumer)
    - Goal: optimise overall system performance

15

Applications

- Self-Adaptive
  - Two applications managers handling resources (servers)
    - Resources are dynamically allocated on basis of policies
    - If application manager cannot apply its policy, asks a Resource Arbiter for additional resources



16

Applications

- Self-Adaptive (within framework)
  - Components:
    - Two Application Managers (AM1, AM2)
    - Resource Arbiter (RA)
    - Two Servers (S1, S2)
  - Meta-data
    - Servers transaction time
    - Servers CPU availability

17

Applications

- Self-Adaptive (within framework)
  - Policies (Action policies)
    - AM-Policy1:
      - *increase CPU by 5% if response time is above 100 ms*
    - AM-Policy2:
      - *if transaction time > 100 ms and CPU availability > 98% ask RA for more CPU*
    - RA-Policy:
      - *if request for CPU, grant and give priority to AM1*

18

Applications

- Self-Adaptive: design-time validation
  - Individual applications:
    - response time < 100ms
  - System
    - Two services with response time < 100 ms
  - If AM1 overloaded, then AM2 fails

19

Applications

- Self-Organising System
  - Stigmergy / Car traffic flow
    - Goal: maximise traffic throughput
  - Components:
    - Cars
      - origin and destination, initial route
    - Traffic light controllers
      - insert traffic flow information at 8 branches of road intersection

20

Applications

- Self-Organising System
  - Metadata
    - Traffic flow information
  - Policies
    - Car: *if traffic flow down the path is above threshold change direction and recalculate route*
    - TrafficLight policy: *modify traffic lights duration according to traffic flow observed*

21

Concrete Case Study

- Self-Assembly
  - Manufacturing cell
  - Agentification of elements
  - Framework supporting:
    - dynamic re-configuration
    - selection
    - determination of best element
  - Long term goal:
    - Give high-level description
    - System finds solution to actually build the product

22

Proofs of concept

- Acquisition and use of metadata
  - Mediator system (Newcastle)
  - Web services monitored by SubMediators
    - Continuous observation of Web Services
      - Availability, functionality, performance, faults, exceptions
      - Production of metadata
    - SubMediators take dynamic decisions
      - Most reliable Web service, quickest response, etc.
  - Policies for QoS expressed through a policy language

23

Proofs of concept

- Run-time infrastructure
  - Service-oriented middleware supporting
    - Self-description of services (functionality)
    - Registration and requests of services on the basis of specifications
  - Allows
    - Seamless binding of components
      - No API (only based on descriptions)
    - Dynamic reconfiguration (while code is executing)

24

Predictability

- Enforcement of policies provides predictability
  - Matching of specifications / metadata
  - Components Replacement
  - Automated Reasoning
- Meta-policy
- Hierarchical policies

25

Summary

- Self-Organising Systems
  - Traditionally use metadata (stigmergy)
  - + Policies
  - ➢ Allows to have overall control
- Self-Adaptive Systems
  - Traditionally use policies
  - + Metadata shared among components
  - ➢ Inserts decentralised control and self-organisation

26