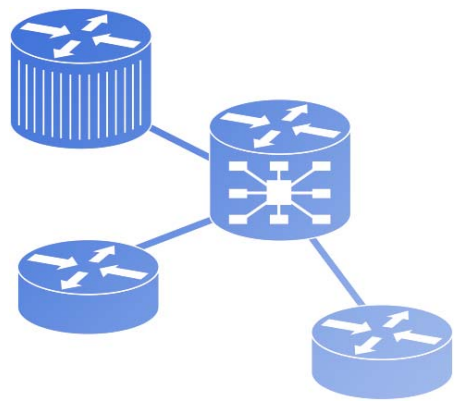
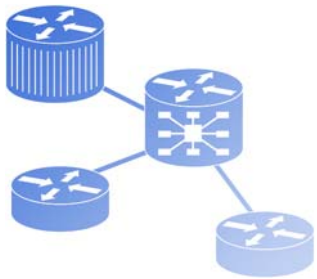
A Real-Time Pattern Based Approach to Autonomic Computing

Bogdan Solomon, Dan Ionescu
Network Computing and Control Technologies Research Laboratory, School of Information Technology and Engineering, University of Ottawa

Marin Litoiu, Mircea Mihaescu
IBM CAS Toronto

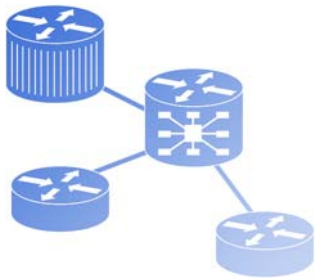
May 28, 2007





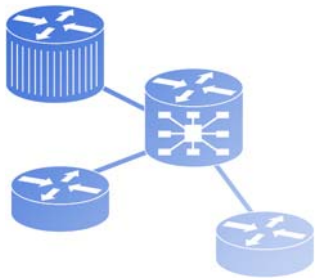
Content

- Project introduction
- Architecture
  - Requirements
  - High-level architecture
- Implementation
  - Component based
  - Web service based
- Current and future work



Project introduction

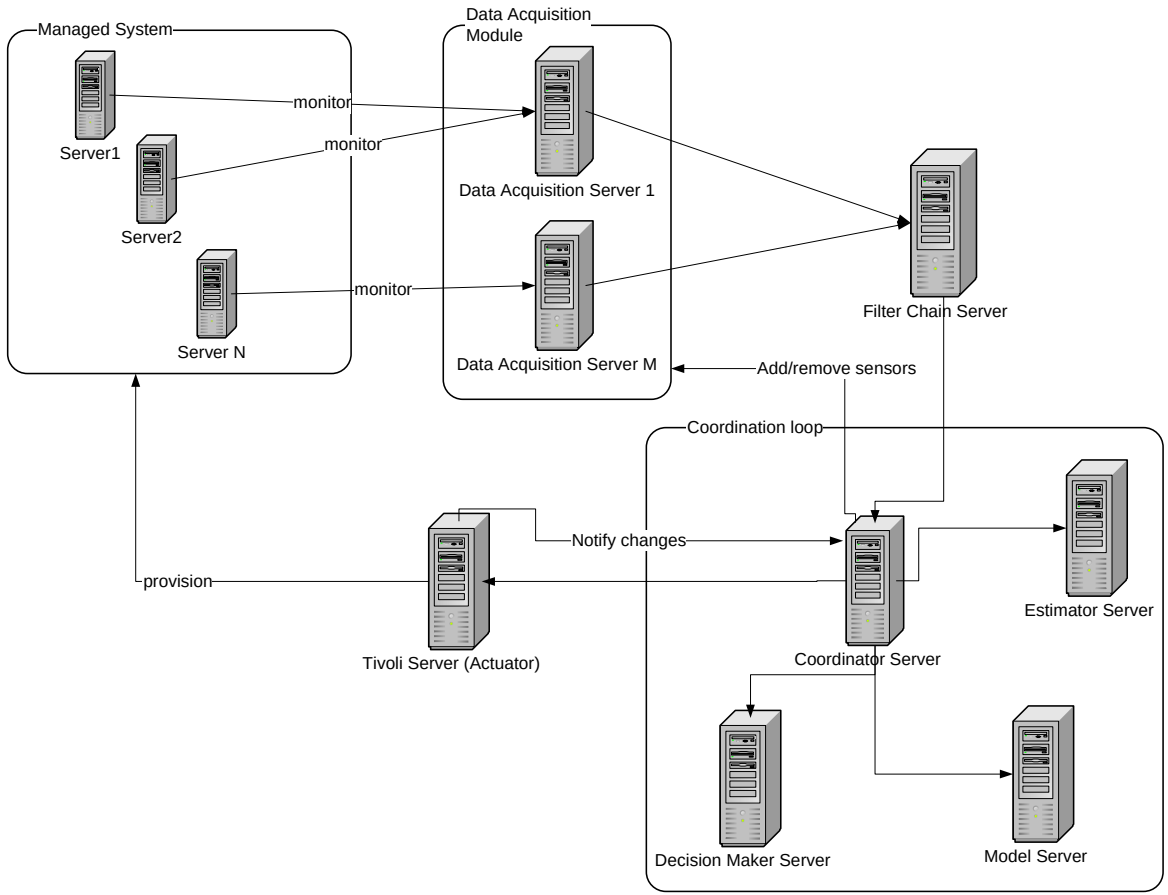
- Goal: Design and implement an agile platform for autonomic computing research and development
  - A plug and play architecture for:
    - Flexibly connecting together elements of self-management:
      - Transducers
      - Filters
      - Estimators
      - Controllers
      - Managers – for self–policies with plug and play control mechanisms
- Applications:
  - Identification of autonomic computing methodologies
    - Validate the layered queuing model for autonomic computing
    - Nonlinear model and its structure identification
  - Various control structures for autonomic computing
    - Discrete event system control
    - Classic control including robust, jumping parameters (stochastic), output optimal, etc
  - Adaptive control of autonomic systems

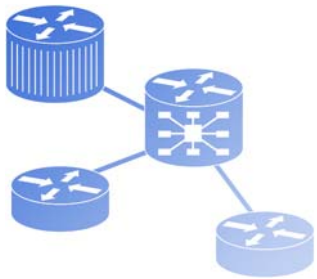


Project introduction

- Autonomic systems seen as control systems with sensors, controllers, actuators
- The state of the managed system is monitored
  - Response time, CPU utilization, etc.
- Controller implements various control rules
  - Kalman filtering, etc.
- Actuators take the appropriate action to provision, configure, heal the managed system

Project introduction

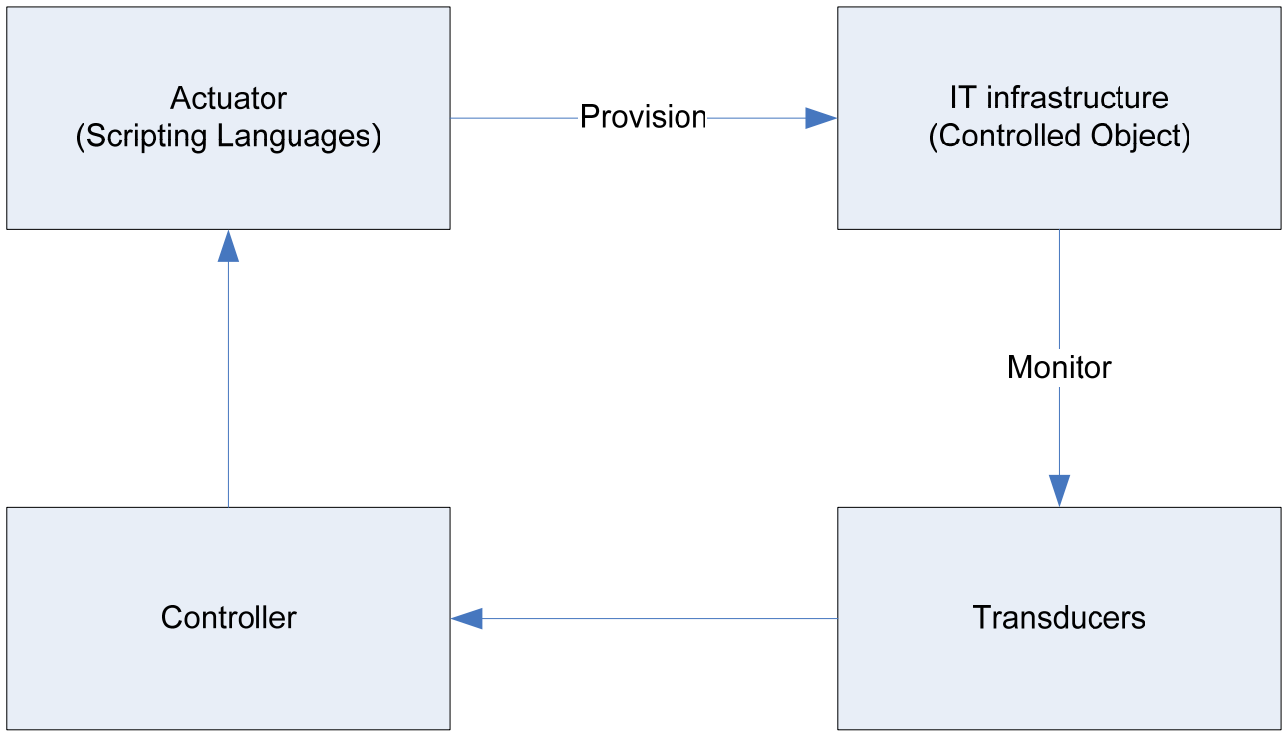




Architecture - Requirements

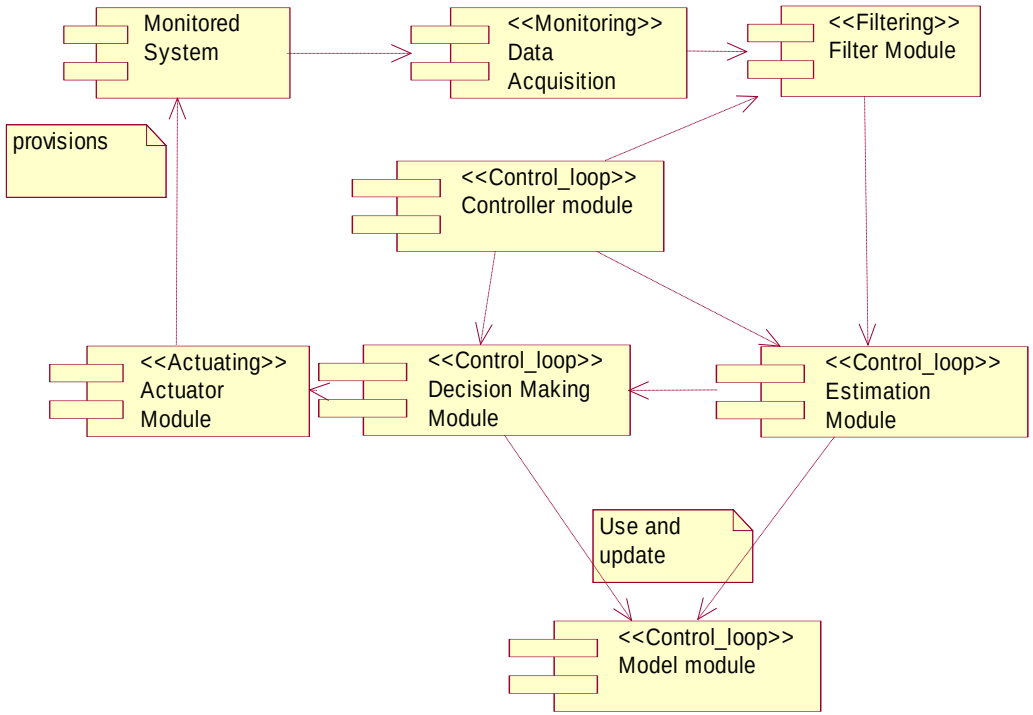
- Decomposing the autonomic system into its basic components
  - Each component has standard functionality and interfaces
  - Provides a modular architecture with reusable components
- Components can reside on various nodes
  - Distributed nature of the autonomic system
  - Real-time software patterns needed to ensure synchronization and reliability of data
- Components can be modified/added/removed at run time
  - Autonomic control loop becomes self-managed itself

Autonomic computing architecture - a control system view



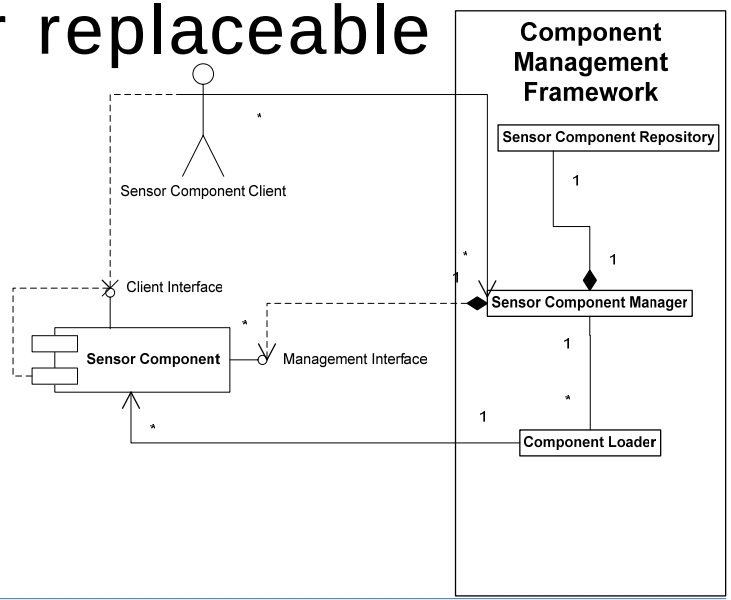
High-level architecture

- Seven standard components identified



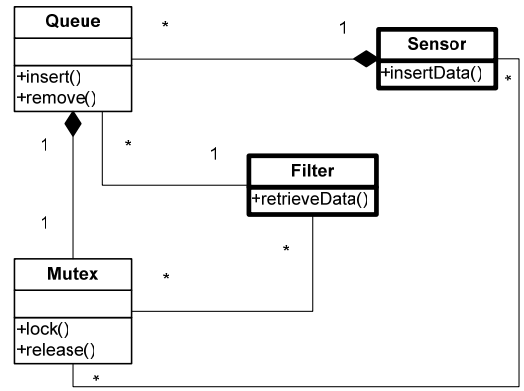
High-level architecture

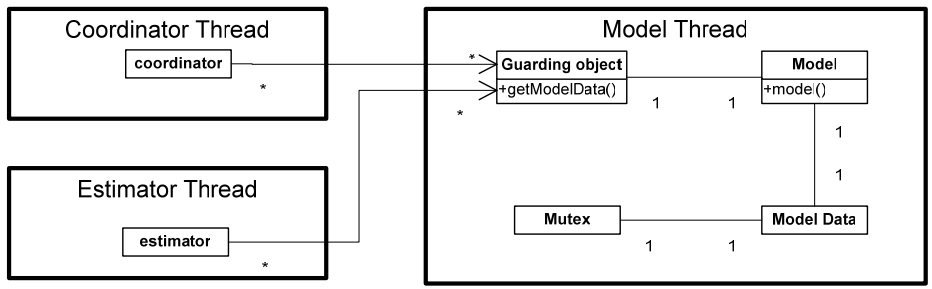
- Each component uses the component-based architecture pattern which provides:
  - The infrastructure for replaceable components
  - Isolation of defects
  - Ease of reusability
  - Strong encapsulation



High-level architecture (cont.)

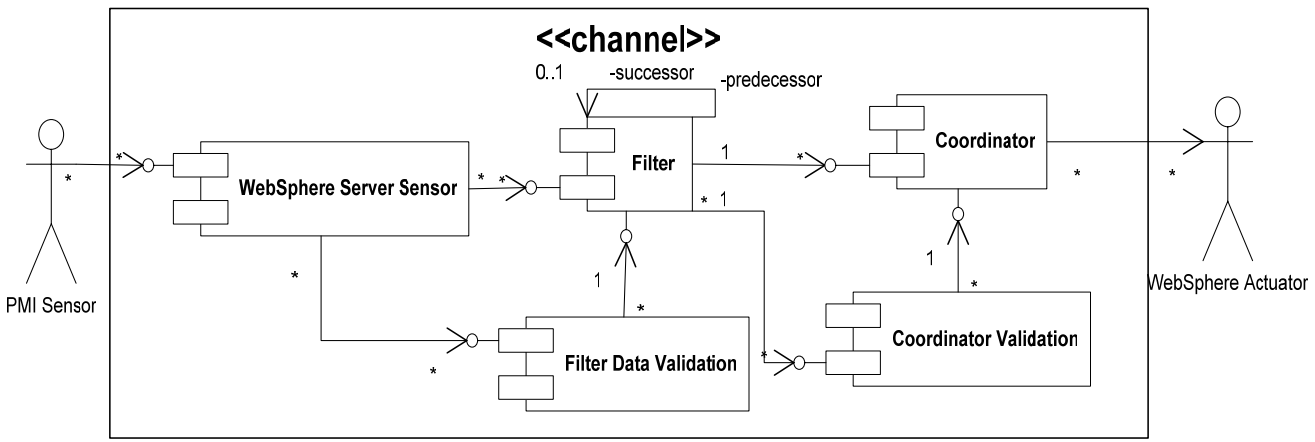
- Concurrency patterns used to ensure the synchronization of messages between components
  - Message-queuing pattern for asynchronous messaging
  - Guarded-call for synchronous messaging

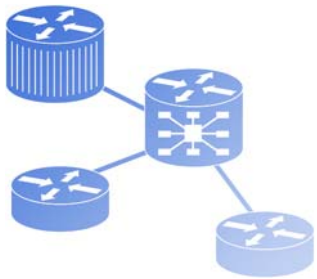




High-level architecture (cont.)

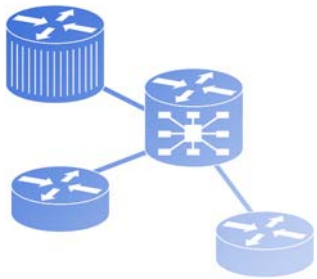
- Reliability pattern used to ensure the validity of the data
  - Protected single channel pattern used to ensure that the data being passed through the control loop is valid





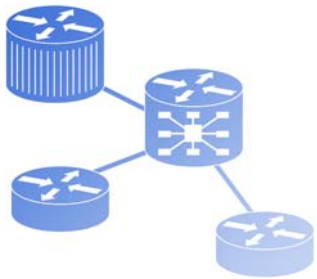
Interoperability

- Interoperability is achieved through the use of web services for communication between components
- Each component manager provides a web service endpoint
- Use of an UDDI registry to discover component locations



Implementation

- Goal of the application
  - Provisioning of a cluster of WebSphere Application Servers based on the servlet response time
- Components – J2EE web services deployed on WebSphere Application Server 6.1
- Sensors – PMI based sensors either wrapped as a web service or a WSDM metric (Apache Muse)
- Controller – Kalman filter based controller
- Actuator – Tivoli Intelligent Orchestrator

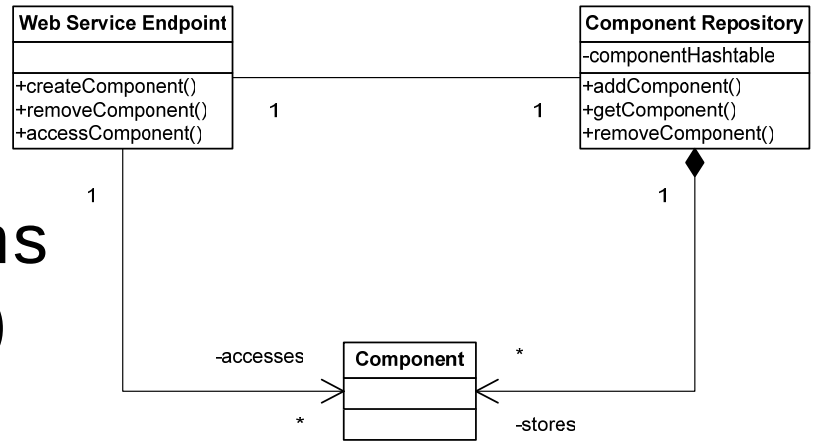


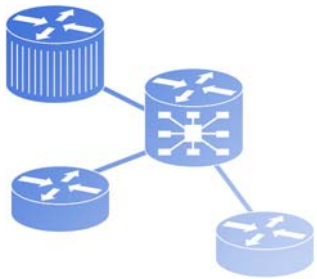
Implementation (cont.)

- Component framework
  - Class loader information used to obtain name of available components
  - Java reflections used to dynamically create components
  - Static hashtable used to store the components based on a unique ID across the whole control loop
  - Component clients make calls through the component manager using the unique ID

Implementation (cont.)

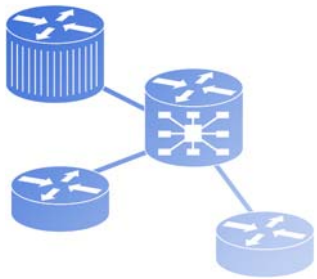
- Component manager developed as a web service
- Provides access to
  - Management functions (create/remove/stop)
  - Access functions (specific to each component)





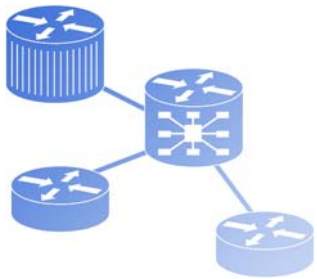
Implementation (cont.)

- Data between components passed via objects using name – value pairs
- Coordinator responsible for the execution of the control loop by calling the other components



Current and future work

- Using WSDM technology for autonomic computing
- Developing and implementing a versatile test bed for autonomic computing
- Developing an adaptive control loop that will manage the autonomic control loop
- Developing a library of components
- Developing an application that would allow the user to select the components to be deployed



Questions and answers

Thank you!