Reflective Middleware

ICS 243F Presentation
Sushil K Bajracharya
sbajrach@ics.uci.edu

Overview

- Dynamic Adaptive Middleware
- Reflective Middleware for dynamic adaptation
- Composability in Middleware
- CompOSE|Q as an example

Middleware

- Why do we need middleware
  - Interoperability, Heterogeneity, Abstraction
    - MOM, Event Based, OO middleware
  - Adaptability
    - Dynamically changing environment
    - QoS guarantees
      - Ubiquitous and pervasive computing
      - Mobile Multimedia Applications
      - Embedded/Real time systems
  - *Adaptive* Middleware
    - *Reflective* middleware

Adaptive Middleware

- Adaptive Middleware
  - Can change functional/non-functional constraints (e.g. QoS) either -
    - *Statically* – leverages the capabilities of specific platform or environment the middleware runs in
    - *Dynamically* – allows the application to run with optimum system responses where the requirements or environment is changing
  - A *reflective middleware* is an approach to build dynamically adaptive middleware

Reflective Middleware - basis

- Based on Computational Reflection concepts
  - Reification and Absorption
  - Meta-level architectures, meta-objects and meta-object protocol
  - Structural and behavioral reflection
- A reflective system…
  - Supports causally connected self representation (CCSR)
  - Can *reason about* and *act upon* itself
    - Is able to *inspect* and *change* itself during the course of its execution = Inspection + Adaptation

Reflective Middleware

- *Definition:* Reflective Middleware is simply a middleware system that provides inspection and adaptation of its behavior through an appropriate CCSR.
- Features
  - Open model, composed of group of collaborating components
  - Support for *dynamic customization of components*, accessible through meta-level interfaces
  - Gives a two or multi level view of the system
    - Base: application objects
    - Meta level: internal representation of middleware components

Safe Composability

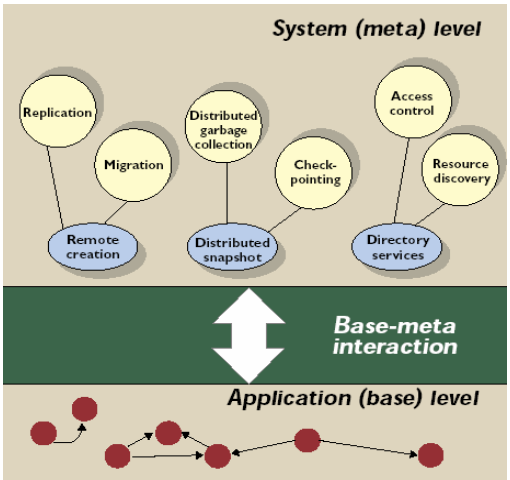
- Advantages of dynamism
  - Services can be dynamically and freely composed to achieve desired functionality
  - Customizability to satisfy application requirements in the presence of changing system conditions
- Issues …
  - validating compositions against *correctness*
    - manage change in large-scale distributed systems while ensuring application QoS requirements (reliability, availability, cost-effective utilization)
  - *semantics* of applications under such (new) compositions based in generic distributed management infrastructures
- A solution …
  - a well designed meta-architectural framework with a sound *formal basis* (e.g. TLAM)

TLAM

- *Two Level Actor Model* (TLAM) - a formal basis for safe composibility
  - provides a formal semantic basis for *specifying* and *reasoning* about the properties of and interactions among middleware components in CompOSE|Q.
  - based on the *Actor* model, a two level Actor system with actors distributed over the network.
    - *Base actors* are application objects
    - *Meta actors* are part of the runtime system.

…TLAM

- Provides a formal description of three basic core system services
  - Remote Creation
  - Distributed Snapshot
  - Directory Services
- Further higher level services can be defined from these three core services.
  - Each is accompanied by specific interface definitions and interaction constraints.



…TLAM

- Allows a formal specification and checking of non-interference properties such as *interference between*
  - Actors with a common acquaintance.
  - Base and meta level actors on the same node.
  - Meta actors implementing different services and thus modifying base level actors in possibly incompatible ways.

CompOSE|Q

(Composable Open Software Environment with QoS)

- QoS enabled customizable middleware framework for distributed computing, based on TLAM
- Consists of actors (base and meta) distributed over network, running parallel, communicating via asynchronous message passing
- Allows the concurrent execution of multiple resource management policies in a distributed system in a safe and correct manner
- Permits customization of resource management mechanisms such as placement, scheduling and synchronization

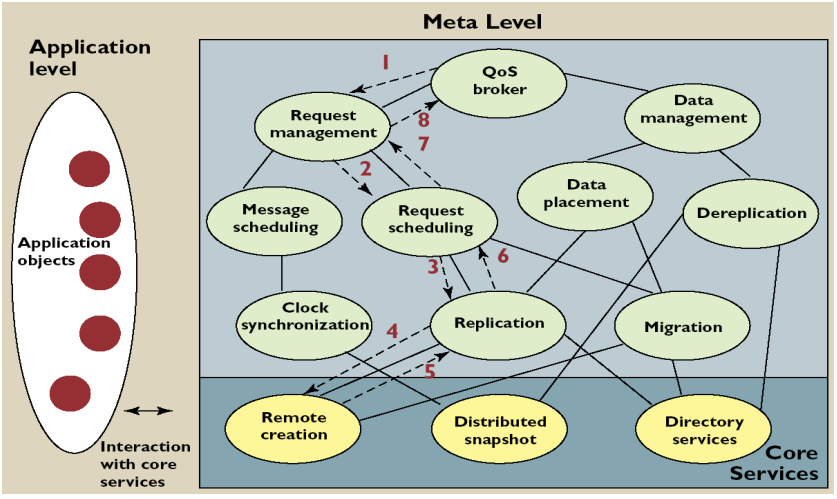
CompOSE|Q - Architecture

- Architecture
  - Basic composable Core Services
    - Remote Creation, Distributed Snapshot and Directory Services with interaction constraints
    - ensure their concurrent execution with each other and other meta level services
  - Services built using Core Services
    - Actor migration, replication of services and data, actor scheduling, distributed garbage collection, name services …
    - Each has its own interface definitions and interaction constraints
  - QoS enforcement mechanism

CompOSE|Q – core components

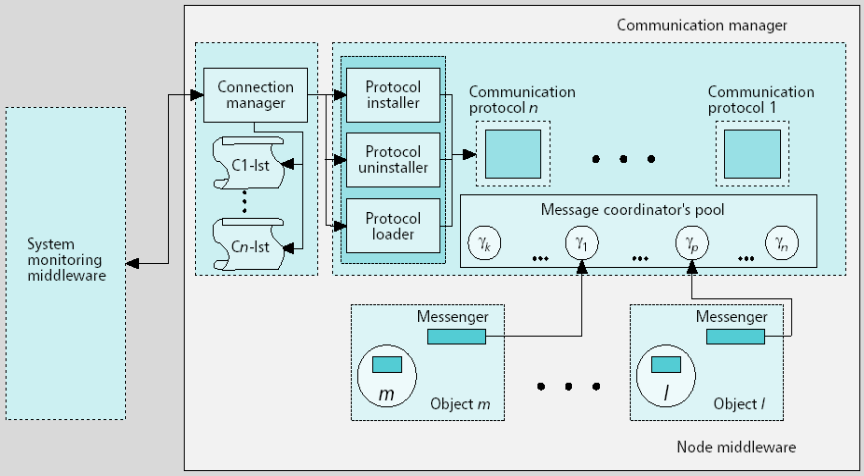
- Architecture is realized by these *Components*
  - TLAM
  - Resource Management Services
    - Core Services + Migration, QoS Brokerage
  - Reflective Communication Service Architecture
    - extends TLAM with a composable reflective communication framework (CRCF).
    - provides correct composition of communication services to QoS-based applications in a transparent and scalable fashion while ensuring correctness of basic middleware services

CompOSE|Q - Architecture



CompOSE|Q

Composable Reflective Communication Framework (CRCF)



CompOSE|Q – Implementation

- Runtime Architecture is implemented in Java, made up of three basic components
  - A *NodeManager* that manages and coordinates various components on a node
  - A *NodeInfoManager* that manages information needed by the local actors and interfaces with the directory service
  - A *communication sub-system* that handles messaging between actors

CompOSE|Q – Implementation