An Infrastructure for Adaptable Middleware

Pierre-Charles David & Thomas Ledoux

{pcdavid,ledoux}@emn.fr

Distributed Objects and Applications 2002
University of California, Irvine

October 30, 2002

Motivation

- Applications must adapt to fit their changing environment
  - Increasing *diversity* of platforms (PDAs servers)
  - Increasing *dynamicity* in execution conditions (e.g. mobile computing)
- How to make software able to *be adapted*, or better to *adapt itself* to changing execution conditions?
  - Software must be *reconfigurable*
  - Both the software and its environment must be *observable*
  - To adapt itself, the software must contain *adaptation logic* [Dowling01] (decides which reconfigurations are appropriate given the changes it observes)

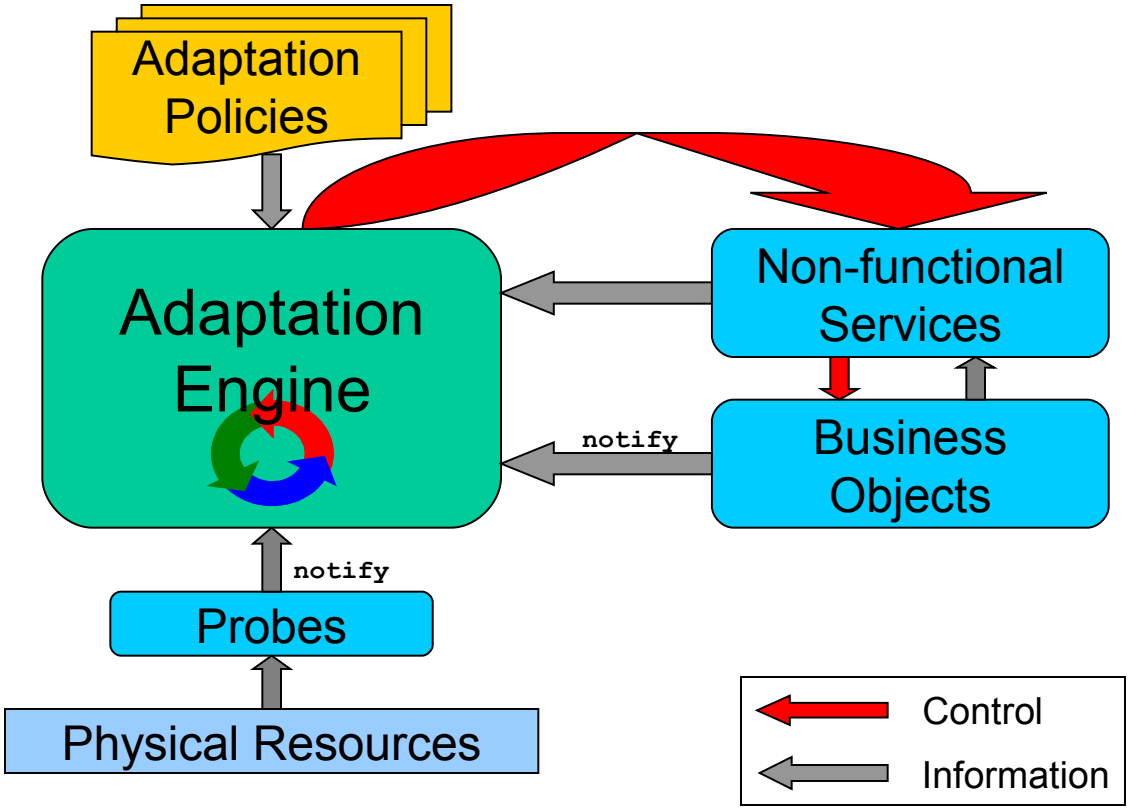
Our Approach to the Adaptation Problem

- Infrastructure point of view
  - Try to find a *generic* but *configurable* solution
  - Embed this abstract solution in an infrastructure layer
  - Concentrate on *non-functional adaptations*
- Build on the *middleware* approach
  - Isolates applications from underlying platforms (CORBA)
  - Separates business logic from non-functional services (EJB)
  - Previous works on reflexive & reconfigurable middleware (dynamicTAO, OpenORB, OpenCORBA…)

Other approaches are possible, with different trade-offs

Overview of the Proposed Solution

Run-time adaptation of non-functional services according to adaptation policies provided independently of the functional code.



Development Process

- **Coding**: standard Java code (with some constraints)
- **Compilation**: provide a configuration file telling which parts of the application will be adaptable at run-time
- **Deployment**: write *adaptation policies* and develop/reuse the required *services*
- **Runtime**: let the engine adapt the system, or add new adaptation policies

Structure of the Applications

(as seen by the engine)

- Each adaptable business object is augmented with dynamic *attributes* and *roles*
- Attributes: describe objects & distinguish them
  - some predefined, managed by the system (`className`)
  - some reflect the business object's state (instance variables specified at compilation-time)
- Roles: implement non-functional services
  - dynamically attached, detached and configured
  - modify the object's behavior (using reflection)

The Adaptation Policies

- Purpose: configure the generic engine for a particular application
- *System Policies*:
  - Associate non-functional services configurations with execution conditions
  - Independant of a specific application, and hence incomplete
- *Application Policies*:
  - Define groups of business objects based on their non-functional needs
  - Associate system policies to these groups

System Policies: Concepts

- A policy = a named set of *rules*
- Adaptation rule: *condition* □ *action*
- *Conditions*: logical expressions
  - expressed in a Scheme-like syntax
  - special variables represent the execution environment properties (eg. CPU usage, bandwidth, packet drops stats…)
- *Actions*: attachment of a role
  - optional configuration parameters

System Policies: Syntax

```
(define-policy "distribution.server"
  (when network.available
    (attached "distribution.rmi" "server"
              '((name.server . "localhost")))))

(define-policy "caching"
  (when (< network.bandwidth.free_kbs 300)
    (attached "smart-proxy" "main")))
```

System Policies: Semantic

- A policy load-time:
  - Analysis of the condition expressions
  - Deployment of the required system *probes*
- At run-time:
  - Probes invoked periodically
  - Depending expressions are evaluated
  - If their value changes, the engine applies the corresponding action (or its opposite)

Applications Policies: Concepts

- Goal: allow different parts of the application to be adapted differently
- An application policy:
  - Defines a (dynamic) group of business objects
  - Binds system policies to it (members will be affected by the corresponding rules)
- Groups organized hierarchically
  - A root group, `all`, contains all the business objects
  - A group *G* is defined by a parent *P* and a predicate *p*
  - $G = \{ c \in P \mid p(c) \}$
  - Predicates in Scheme-like syntax; can access objects attributes

Application Policies: Syntax

```
(define-group "remote-accessible" :parent "all"
  (select (member (attribute "className")
                  '("Bookstore" "Catalog" "Book")))
  (bind "distribution.server"))


(define-group "secure" :parent "remote-accessible"
  (select (= (attribute "security-level") "sensitive"))
  (bind "distribution.ssl"))


(define-group "best-buyers" :parent "all"
  (select (and (= (attribute "className") "Customer"))
               (> (attribute "invoiceTotal") 500)))
  (bind "logging"))
```

Application Policies: Semantic

- At policy load-time:
  - Group creation
  - Binding of the system policies
  - Initial members computation
- At runtime:
  - When an object's attributes change
  - or an object is created/destroyed:
    - re-evaluation of the predicates
    - move from/to groups if necessary
    - attachment/detachment/reconfiguration of roles

Summary of Contributions

- *Separation* of the adaptation logic from the business logic
- *Integration* of all the required parts into a unified framework (monitoring + decision + reconfiguration)
- Fine-grained, dynamic adaptation
- Supports *unanticipated adaptation* through run-time modification of policies

Future Works

- *Investigate* adaptation of object migration policies
- Replace monitoring & reconfiguration parts with *State of the Art* solutions
  - guarantees of correctness on reconfigurations
- *Enhance* the decision process
  - more declarative policies