A Policy-based Mobile Agent Infrastructure

Rebecca Montanari, Gianluca Tonti
D.E.I.S. Department
University of Bologna
Italy
{rmontanari,gtonti@deis.unibo.it}

Cesare Stefanelli
Department of Engineering
University of Ferrara
Italy
{cstefanelli@ing.unife.it}

Outline

- Motivations for a policy-based approach to mobility
- A policy-controlled Mobile Agent Framework
- Case Study
- Conclusions and Open Issues
Motivations

Mobility adds complexity to the design and development of applications

- Programmers have to explicitly deal with the allocation of agents
- Poor programming abstractions and structures for controlling mobility
- Mobility specifications embedded into the agent code are not adequate to cope with the frequent changes of the environment state
- Need to dynamically change the mobility behaviour of agents without re-engineering the agent code

Need of Flexibility

Separation of concerns: mobility and application logic

Policy-based Approach to Mobility

What are policies?

“Policies are rules governing choices in the behavior of a system, and are separated from the components in charge of their interpretation”

- Policies are used to simplify the management of complex systems and networks

IDEA:

Policies for governing the mobility behaviour of Mobile Agents

Mobility Policies specify:

WHEN, WHERE and WHICH UNIT of mobility must migrate
**SOMA Design Guidelines**

The adoption of a policy-based approach to mobility requires

- The choice of a policy language
- The design of a set of services for policy management

- Ponder language developed by the Imperial College of London
- Specification Service
- Distribution Service
- Enforcement Service
- ... 

**Mobility Policies Examples**

- Ponder for controlled mobility
- Event-triggered policies
- Declarative policies

**Examples**

```
inst oblig MobPol1 {
    on CPULoad(90);
    subject s = agents/Manager;
    do s.go(G1.toString(), "run");
    when MonitoringSystem.isReachable(G1);
}
```

```
inst oblig MobPol2 {
    on CPULoad(90);
    subject s = System/Relocator;
    target t = agents/Manager;
    do s.relocate(t, G1.toString(), "run");
    when MonitoringSystem.isReachable(G1);
}
```
Policy-based Architecture

Policy Specification

Ponder environment

http://www-dse.doc.ic.ac.uk/Research/policies/index.shtml

Ponder Viewer, Editor, Syntactic and Semantic Verifier, Compiler

JAVA Policy Object
### Policy Distribution

**Policy Distribution Service**
- Currently implemented as a stationary agent, one for each SOMA domain
- Constantly informed about changes in the set of policies
- Whenever a policy change occurs it locates the interested subjects and update them
- At any initialization it forces the agent registration to the Event Service

```java
inst oblign MobilityPolicy{
  on NewConnection (PlaceID);
  subject s = agents/Manager;
  do itineraryPath.addLast(PlaceID); }
```

### Policy Installation

**Policy Repository Service**
- A distributed directory service that persistently stores policies
- Each directory entry records the Java object representing the Ponder policy
- Each directory entry has a unique distinguished name

- Agents retrieve their policies from the Policy Repository
- To avoid frequent accesses to the Policy Repository, a copy of the policy can be also cached within the agent
Policy Taxonomy

SOMA provides different strategies for policy installation, depending on application requirements.

Obligation Policy

Agent Policy

System Policy

Infrequently Used Policy
Frequently Used Policy
Critical Policy

by reference
by copy
by copy

Policy Enforcement

SOMA Place

Policy Repository

Policy Enforcer

Policy Interpreter

Monitoring and Event Service

Monitors data collected by exploiting JVMPI and JNI

JEDI technology

Events are objects that encapsulate information related to changes

Monitoring data collected by exploiting JVMPI and JNI

Event dispatching exploits the JEDI technology

Monitoring and Event Service

inst oblig MobilityPolicy{
  on NewConnection (PlaceID);
  subject s = agents/Manager;
  do itineraryPath.addLast(PlaceID); }

Event reaction

Policy Enforcement Service
A SOMA agent is composed of three parts: state, code, and mobility policies.

The Agent Worker Thread executes the agent code.

The Policy Worker Thread enforces the mobility policies.

Case Study

Balancer agent

Load balancing problem....
Case Study: Policy-based MA

public class BalancerAgent2 extends PolicyAgent{
    LinkedList ItineraryPath;
    Hashtable load = new Hashtable();
    PlaceID currentPlace; int average;
    int currentLoad;
    void run() { // Initialisation method
        .... // ItineraryPath initialisation
        startMethod(); }
    void startMethod() {
        currentPlace = super.getPlaceID();
        retrieveLocalLoad();
        average = dataElaboration();
        balancingActions(); }
    void retrieveLocalLoad() {
        currentLoad = checkLocalLoad();
        load.put(currentPlace, currentLoad); }
    int balancingActions() { ... }
    int dataElaboration() { ... }
    int checkLocalLoad() { ... }
    // BalancerAgent2 End

inst oblig MobPol1
on NewConnection( node );
subject s = res/agents/BalancerAgent2;
do s.ItineraryPath.addFirst(node);
when node.QoSLevel = “High”;

inst oblig MobPol2
on TaskCompletion ( BalancerAgent2);
subject s = res/agents/BalancerAgent2;
do s.ItineraryPath.addLast(s.currentNode)
  -> node = s.ItineraryPath.removeFirst()
  -> s.go (node, “startMethod”);
when ItineraryPath.size()>0 and s.currentLoad < 100

inst oblig MobPol3
on SecurityAlarm( node );
subject s = res/agents/BalancerAgent2;
do s.ItineraryPath.remove(node);

Conclusions and Open Issues

- Policy-based approach to control mobility can simplify application design and maintenance
- Dynamic modification of agent mobility behaviour via changes in policy specifications
- Policy-based approach to rule and control other aspects of mobile applications (security, fault tolerance, communication…)
- Policy conflict analysis and resolution