Using Publish-Subscribe Communication Genre for Mobile Agents

Amir Padovitz, Seng Wai Loke and Arkady Zaslavsky
School of Computer Science and Software Engineering
Monash University, Australia
padovitz@bigpond.com, swloke@csse.monash.edu.au, a.zaslavsky@monash.edu.au
Outline

- Objectives, background & motivation
- Implementation
- Experiments
- Results & interpretation
Objectives and motivation

- Find an efficient generic solution for communication between mobile agents of different toolkits
- Use lightweight content-based messaging in multi-agent systems
- Integrate Elvin’s publish-subscribe communication model into mobile agents
- Consider the use of such a solution for communication between context aware mobile agents.
Benefits of Event-Based Mechanisms

➢ Event is a natural abstraction for the activities within the environment and the interactions among places and agents

➢ Event-based models provide implicit invocation

➢ Elvin paradigm is independent of agent toolkit
Example scenario – context awareness

- Wired host
- Mobile agent migration
- Adjust subscriptions for web server notifications
- Wireless device
- Notify change in context
- Send warnings
- Web server
- Receive agent new status
- Interacting agent
Context based messaging

- Send messages to agents based on their current context, rather than their identity.
- Agent request messages according to its context.
- Use Elvin’s subscription language. E.g.:
  - Subscription:
    \[(CREDITS == "low" && DISKSPACE == "< 5GB")\]
  - Notification:
    CREDITS: “low” DISKSPACE: “< 5GB”
    COMMAND: “come_home”
Collections and sub-collections

- Notification:
  - CREDITS: “low” DISKSPACE: “< 5GB”
  - GROUP: “Tom” COMMAND: “come_home”

- tune the subscription (optional)
  
  (CREDITS == “low” && DISKSPACE == “< 5GB” &&
  GROUP = “Jane”)

3/08/2005
Community of context aware mobile agents
Elvin Focus

- Content based messaging.
- Strong subscription language for specifying content with logical expressions.
- Robust and efficient for dissemination of large quantities of messages.
- Support multiple programming languages.
- Affiliation with Elvin group (development and support)
Background - Constructs

- Elvin provides content-based, event driven notification mechanism, using centralized server and exposes API for clients
- Agent toolkits
  - IKV++'s Grasshopper, IBM's Aglets
Implementation – the problem

- Elvin is designed for traditional static applications. Not for mobile ones...
- Mobile agents often change physical locations and are not traceable by Elvin Server
- Elvin classes are not serializable so cannot migrate with the mobile agent
Implementation – the problem

stage 1:

- Host 1
  - Elvin Client Library
  - Subscribe for events
- Host 2
  - Elvin Client Library

stage 2:

- Host 1
  - Elvin Client Library
  - Migration of mobile agent
  - Send notifications via connection
- Host 2
  - Elvin Client Library
Implementation – the solution

- Introduce new functionality represented by MobileConsumer class
- Implements functionality that support Elvin communication
- MobileConsumer - Serializable, so can migrate with the agent as a member variable
- Situated in between the client and Elvin, conceptually part of Elvin-Client layer.
- (Can be used by all toolkits)
Implementation – the solution

MobileConsumer

App. Client

Elvin Client

Elvin Server
Implementation – the solution

- **MobileConsumer Functionality** -
  - Removes all existing subscriptions before migration
  - Re-subscribes to the same events after migration
  - Keeps Elvin related information such as subscriptions in Serializable form
  - Encapsulates implementation details, generic and offers easy usage of its functionality
Implementation - solution

- **MobileConsumer API:**
  - NotifyPreMigration()
  - NotifyNewLocation(…)

- Client activates the functions just before and after migration.

- Ideally, they are implemented in callback functions, designed for that purpose by the agent toolkits (common feature)
MobileConsumer mc = new MobileConsumer(ElsvinServerURL);

public void onCreation(Object itin) / init(Object[] creationArgs) {
    Subscription sub = new Subscription(strSubscription);
    //NotificationHandler class to handle notification events
    NotificationHandler nh = new NotificationHandler( );
    sub.addNotificationListener(nh);
    mc.addSubscription(sub);
}

public void onDispatching(MobilityEvent m) / beforeMove( ) {
    mc.notifyPreMigration( );
}

public void onArrival(MobilityEvent e) / afterMove( ) {
    NotificationHandler nh = new NotificationHandler( );
    mc.notifyNewLocation(nh, strSubscription);
}
Experiment I - Objectives

- Communication via Elvin between different agent toolkits.
- Performing communication with migrating agents.
- Proof of concept.
Agents send Hello, Location and Received messages

Agents subscribe to each other Hello

Monitor subscribes to Hello, Location and Received
Experiment (cont.)

Monitor Simulation

Host 1
Grasshopper
subscribes to: Hello from Aglet

Elvin Server

Host 5
Aglets
subscribes to: Hello from Grasshopper

Host 1
Migrate
NotifyPreMigration( )
remove all subscriptions

Elvin Server

Host 5
Migrate
NotifyPreMigration( )
remove all subscriptions

Experiment (cont.)
Experiment (cont.)

Host 2

NotifyNewLocation(...)

Grasshopper

renew all subscriptions

Elvin Server

send notifications:
Hello from Grasshopper
Arrival at Location

Host 6

NotifyNewLocation(...)

Aglets

renew all subscriptions

Elvin Server

send notifications:
Hello from Aglet
Arrival at Location

Host 2

Grasshopper

Host 6

Aglets
Experiment (cont.)

Host 2 received notification:
Hello from Aglet

Elvin Server

Monitor Simulation

received notification:
Arrival at Location

Grasshopper

received notification:
Hello from Grasshopper

Host 6

received notifications:
Received Hello (Grasshopper)
Received Hello (Aglet)

Aglets

Monitor Simulation

Elvin Server

send notification:
Received Hello

Host 2

Grasshopper

send notification:
Received Hello

Host 6

Aglets

Experiment (cont.)
Potential problem

- Performance benchmarks suggest short Elvin procedures and relatively long migration times.

- Mobile agents are not responsive during migration.

- Implication: potentially, high amount of lost messages.
Experiment II - measuring message delivery

- Generate close-to-reality agents behavior
- Focus on parameters that affect the % of lost messages:
  - Duration of agent delay in a location.
  - Bandwidth - effect on migration and communication duration.
Experiment II - Architecture

- Creator
- Elvin Server
- Monitor
- Host 1
- Host 2
- Host 3
- Host 4
- Host 5
- Host 6
- machine 1
- machine 2
- machine 3
- machine 4
Experiment II

Total random behavior of agents:
- Random migration paths.
- Arbitrary number of messages to random recipients.
Agents internally count received messages and send results to *Monitor*

*Monitor* counts all messages sent and compares with agents results
Results & Interpretation

Avg. lost messages per location delay

distribution of lost messages per delay

% lost messages

Range of lost messages

Delay per location

3/08/2005
Results & Interpretation

bandwidth delay of 4 seconds

Effects of high and low bandwidth

![](image1)

![](image2)
Results & Interpretation

comparison of different bandwidth delays

% lost messages vs. Delay per location for different delays:
- 0 sec delay
- 2 sec delay
- 4 sec delay
- 6 sec delay
- 8 sec delay
Future work

- Consider mechanisms for reliable messaging
  - Centralised approach with Monitor service passive and agents active
  - Centralised approach with Monitor service active
  - Decentralised architecture with peer-to-peer agent message exchange
- Evaluate scalability of Elvin in multi agent system
ContextExplorers - HP PhD endowment
QUESTIONS?

The End