Mobile Agents and Processes Using Communicating Process Architectures

Their Role in Pervasive Adaptation

Jon KERRIDGE, Jens-Oliver HASCHKE and Kevin CHALMERS
Overview

- Mobility
- Pervasive Adaptation – an overview
- Process Discovery
CSP based Parallelism - Processes

• A system comprises a network of processes
• The processes work together to solve a problem
• Processes undertake a sequence of operations
• All data is local to a process
• Processes can communicate data
• Processes communicate by means of channels

• A process **cannot** access properties of another process by means of a method call
Mobility - Process

• Process mobility means
  – The code and initial state of a process can be moved from one processor to another, where it can be connected into that processor’s infrastructure and executed
  – Classes referred to within the process can also be obtained from the processor from where the process definition was obtained
  – Once a Process has been transferred and instantiated it cannot be moved to another node and can only terminate if the Process itself terminates
Mobility - Agent

• A Mobile Agent is a specialisation of a mobile process
  – It can be transferred to another processor
  – It connects itself to processes already executing
  – It undertakes some action in conjunction with the host process, which modifies the state of the Agent
  – It can then disconnect itself from the host process
  – It can then cause itself to move to another processor, which could be the processor from which it started
  – An agent is a self contained capability that achieves some goal, which when complete, causes the Agent to terminate
Pervasive Adaptation

- According to the EU PerAda Action is manifest by:
  - Networked Societies of Artefacts
  - Evolvable Pervasive Systems
  - Adaptive Software Systems
  - Adaptive Security and Dependability
- Which require
  - Dynamicity of Trust
  - Tiny and Massively Networked devices
Process Discovery

• An exemplar system comprises:
  – A number of TCP/IP networked processors (nodes)
  – A Data Generator node
    • Keeps a record of connected nodes
    • Creates data of defined types, which is
    • Sent randomly to any Processing node that has registered with the network
  – A Processing node,
    • Processes data sent to it by the Data Generator
    • Processed data is then sent to a Gatherer node
  – The Gatherer node
    • Prints out the processed data objects
The Challenge

- The processes required to process a data object are guaranteed to be available on the network
- **BUT**
- Not every Processing node is initialised with an instance of all the required processes
- **AND**
- Processing Nodes can be added dynamically to the network
So

- When a Processing node receives a data object for which it does not have the required process
  - It sends a Discovery Agent around the network to find an instance of the required process at another node
  - The Discovery Agent returns to the originating node with a copy of the required process
  - The process is transferred to the node where it is instantiated and connected into the Processing node infrastructure
  - The Processing node can now process that type of data
Node Processing - Initialising

• Installs any data processes with which it was initialised

• Connects to Data Generator and Gather Nodes
  – By means of named net channels

• Registers itself with the Data Generator sending
  – Agent Visit Channel location
  – DataGenToNode channel location

• Initialises an instance of a Discovery Agent with this node’s Agent Return Channel

• Now in a position to accept inputs from the Data Generator process
Node Processing – Running:1

- Input from Data Generator
  - Notification of a new Node
    - Update the agent with the location of the new node’s visit channel
    - Remember this in a list of connected nodes
  - Data Object
    - If process available then process the object and send result to Gatherer
    - If process not available; update Discovery Agent and send it on a trip to find the required process
Node Processing – Running: 2

• Discovery Agent arrives at Agent Visit Channel
  – Connect Discovery Agent to this Node
  – Discovery Agent sends name of required process to Node
  – Node returns process, if available, or null
  – If process sent then Discovery Agent returns to home node
  – Otherwise Discovery Agent continues visiting nodes

• Discovery Agent arrives at Return Channel
  – Connect Discovery Agent to this (Home) Node
  – Discovery Agent transfer process to home node
  – Home node installs process and is now ready to process data of that type.
Discovery Agent Processing

• Three states
  – Initialising
    • Discovery Agent is pre-loaded with all the Agent Visit Channel locations
    • Discovery Agent is told the name of the required process
    • Discovery Agent then visits nodes to find required process
  – Visiting
    • Discovery Agent connects to the visited node
    • Discovery Agent sends name of required process to node
    • Node sends Discovery Agent either the process or null
    • Discovery Agent continues journey if returned null or home if process loaded
  – Returning
    • Discovery Agent returns to home node and connects to it
    • Transfers process to home node
    • Home node can then install the process
class AdaptiveAgent implements MobileAgent, Serializable {

    def ChannelInput fromInitialNode
    def ChannelInput fromVisitedNode
    def ChannelOutput toVisitedNode
    def ChannelOutput toReturnedNode

    def initial = true
    def visiting = false
    def returned = false

    def availableNodes = []
    def requiredProcess = null
    def returnLocation
    def processDefinition = null
    def homeNode
Agent Processing – Connect and Disconnect

```java
def connect (List c) {
    if (initial) {
        fromInitialNode = c[0]
        returnLocation = c[1]
        homeNode = c[2]
    }
    if (visiting) {
        fromVisitedNode = c[0]
        toVisitedNode = c[1]
    }
    if (returned) {
        toReturnedNode = c[0]
    }
}
def disconnect() {
    fromInitialNode = null
    fromVisitedNode = null
    toVisitedNode = null
    toReturnedNode = null
}
```
Agent Processing - Initialise

```java
if (initial) {
    def awaitingTypeName = true
    while (awaitingTypeName) {
        def d = fromInitialNode.read()
        if (d instanceof List) {
            for (i in 0..<d.size) { availableNodes << d[i] }
        }
        if (d instanceof String) {
            requiredProcess = d
            awaitingTypeName = false
            initial = false
            visiting = true
            disconnect()
            def nextNodeLocation = availableNodes.pop()
            def nextNodeChannel = NetChannelEnd.createOne2Net(nextNodeLocation)
            nextNodeChannel.write(this)
        }
    }
}
```

- Update the list of registered nodes
- Initialise Discovery Agent
- Send Agent on a trip round nodes
**Agent Processing - Visiting**

```java
if (visiting) {
    toVisitedNode.write(requiredProcess)
    processDefinition = fromVisitedNode.read()

    if (processDefinition != null) {
        toVisitedNode.write(homeNode)
        visiting = false
        returned = true
        def nextNodeLocation = returnLocation
        def nextNodeChannel = NetChannelEnd.createOne2Net(nextNodeLocation)
        disconnect()
        nextNodeChannel.write(this)
    } else {
        disconnect()
        def nextNodeLocation = availableNodes.pop()
        def nextNodeChannel = NetChannelEnd.createOne2Net(nextNodeLocation)
        nextNodeChannel.write(this)
    }
}
```

- **Send name of required Process to Node**
- **If available place copy of process in Agent and return to home node**
- **Otherwise, disconnect from node and go to next node on the list of available nodes**
Agent Processing - Returning

if (returned) {
    toReturnedNode.write([processDefinition, requiredProcess])
}
Node Process Internal Architecture

- Node Process provides internal channels which it uses to connect to visiting agents
  - The channels are specific to the type of visiting agent
def visitingAgent = agentVisitChannel.read()
visitingAgent.connect([NodeToVisitingAgentInEnd, NodeFromVisitingAgentOutEnd])

def visitPM = new ProcessManager(visitingAgent)
visitPM.start()

def typeRequired = NodeFromVisitingAgent.in().read()
if (vanillaOrder.contains(typeRequired)) {
    def i = 0
    def notFound = true
    while (notFound) {
        if (vanillaOrder[i] == typeRequired) {
            notFound = false
        } else {
            i = i + 1
        }
    }
    NodeToVisitingAgent.out().write(vanillaList[i])
}
else {
    // do not have process for this data type
    NodeToVisitingAgent.out().write(null)
}

visitPM.join()
Mobile Social Network

- People dynamically join a wireless network using a mobile device
- A service is provided that allows ‘friends’ to exchange diary information for the immediate future so they can meet face-to-face.
- An agent is sent from a new arrival to network, with their list of friends and their free times
- Agent finds out which friends are also registered
- Agent finds times when friends are free at same time
  - It then arranges a meeting
continued

• Agent can determine the diary system used by each friend
• Ensures it has correct diary interrogation system bound in
• The Agent can adapt its behaviour as people change their mobile devices and their software infrastructure.
Conclusions

• Parallelism enables construction of Agent systems at the Application Layer
• Implemented the Itinerary Agent pattern
• Can move processes from one node to another
• Nodes can adapt their processing as the needs arise
Relationship to Pervasive Adaptation

- Networked Societies of Artefacts
- Evolvable Pervasive Systems
- Adaptive Software Systems
- Adaptive Security and Dependability