Introduction to PAGIS

Prototype distributed process network implementation
Metacomputer environment for GIS
Client-server based architecture
PN for computational model
Primitive processes from the GMT library
GIS
An architecture and subsystem for distributed process networks
Same model, same goals as for Jade
PN API for coordination language
AGIS/Jade and PAGIS/ProActive
Channel Location Techniques

Prototype system utilized Server-Side channels. Processes communicate through the server. Central storage enabled simple deadlock detection, but the required number of remote invocations is one. A central point of failure/potential bottleneck is present.

More recent versions utilize Producer-Side or Consumer-Side channels. Channel storage is placed local to the producer/consumer, reducing latency and decentralizing. Parker’s deadlock detection algorithm is less applicable, but one remote invocation is not hidden.
Active Half-Channels

Channels with activity of their own split the channel, with a thread responsible for communication.

Half-Channel thread absorbs latency neatly with reconfiguration.
**Distribution Issues**

**Network Latency**
- A process network hides some latency
- Remote invocations block computation

**Active Half-Channels**
- Provides required asynchronicity
- Reduce amount of communication by blocking transfers

**Managing Network Failure**
- Provided a new point of failure
- Implement fault-tolerance in Half-Channels
Realisation of PAGIS/ProActive

GIS sub-system and factories

Pluggable infrastructure

GIS/ProActive requires implementation of a couple of factories

WorkerToolkit to turn PNs active

ChannelFactory for ProActive-based queues

The rest of the architecture, including scheduling, is shared
Future Work

Scheduling (Placement)
Repository access node placed ‘close’ to the repository
Image visualization node placed on the user’s machine

Distributed Deadlock Detection
Simplified deadlock problem warrants simplified detection algorithm

Distributed Visualisation
Model of distributed concurrent state
MOP can intercept network behaviour for deadlock detection or visualisation
What we have done ...

I for specifying a PN
Minimal toolkit; localized view of reconfiguration
Consistent with semantics
Active realization
Active channels; where to use futures?

PAGIS
Readily adapted to API;
"half-channels": reconfiguration & latency-hiding

P

policies” for reified method calls
But more generally ...
“Adaptive” Reconfiguration

A “plug-in” new implementation of PN node tuned for changes in environment
- new resources available: reconfigure
- consistent with semantics: same history function

How to detect the need to change?
- Extend the MOP
  - Reify relevant aspects of implementation structures
  - Write an observer meta-program over these structures

The MOP changes the implementation
- And derives an equivalent revised abstract PN
  
e. reconfigure according to nature of resources