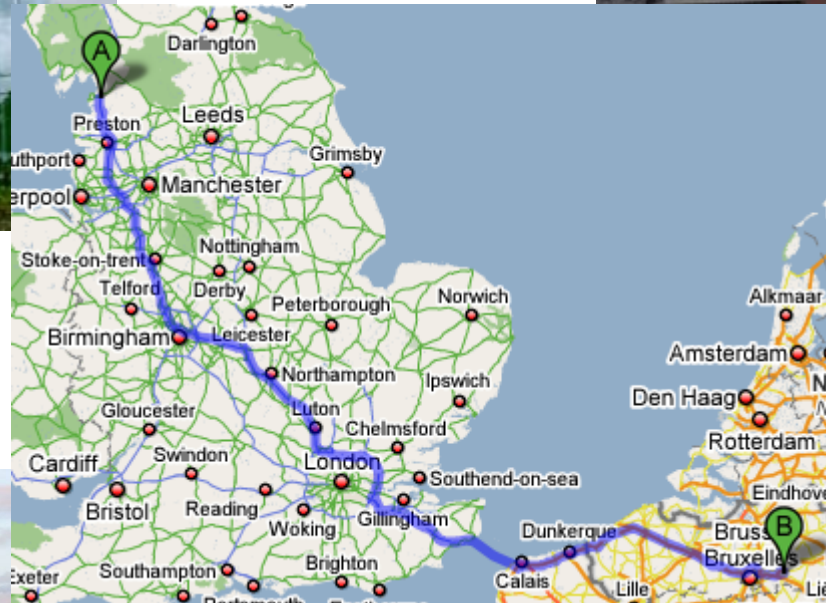


REFLECTIONS ON DEVELOPING ADAPTIVE SYSTEMS SOFTWARE

Paul Grace, Lancaster University

@Lancaster, @Leuven



Overview

- Part 1: Challenges of Systems-of-Systems
 - ▣ Need for complex adaptations
- Part 2: Developing adaptive systems software
 - ▣ Reflection, Aspects, Software architecture
- Part 3: Middleware solutions
 - ▣ Domain-engineering working middleware
 - ▣ Tackling usability and complexity with models

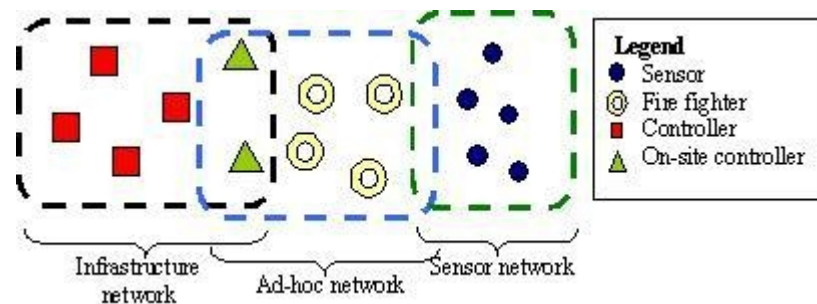
Challenges of Systems of Systems

Systems-of-Systems (SoS)

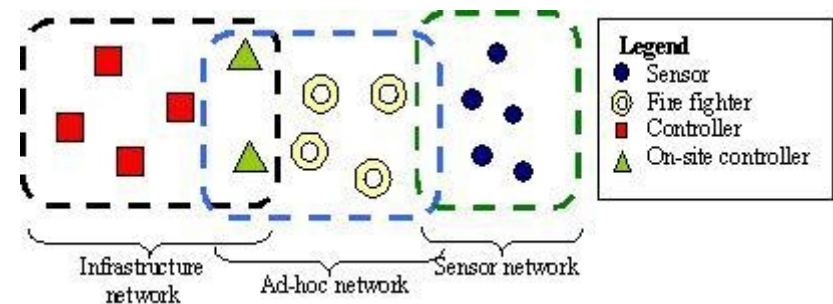
- As software systems become more ubiquitous a new class of large-scale distributed systems has emerged
 - ▣ *Systems of Systems*
 - Face extreme heterogeneity (*devices, platforms, networks*)
 - deliver services that cross device, platform and system administration boundaries.
 - Often exhibit emergent behaviour due to dynamic adaptation
 - ▣ This leads to a degree of *complexity* that is orders of magnitude greater than in traditional distributed systems.

Challenges

Emergency Response



Sensor Grids (Flood Prediction)



Challenges in Systems of Systems

- Functional:** Resource Discovery, Service Oriented, Group, Event Notification, ...
- Concerns:** Adaptation, Security, Dependability, ...
- Interoperability:** Domain, Platform, ...
- Usability:** Overcoming development complexity

Response

- Middleware is well placed to address these issues
 - Systems must be configurable
 - Systems must be reconfigurable
- Approaches from the adaptive middleware community are relevant here

Developing adaptive systems software

Part 2

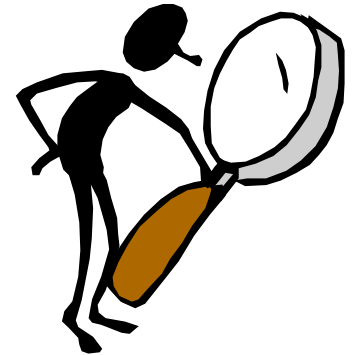
Reflection & Dynamic AOP

What is Reflective Middleware?

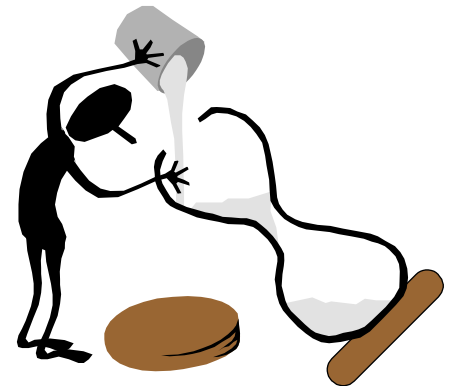
- ▶ Reflection refers to the capability of a system to ***reason about and act upon itself***
- ▶ A reflective system provides a representation of its own behaviour:
 - ▶ that is amenable to inspection and adaptation,
 - ▶ and is *causally connected* to the underlying behaviour it describes
- ▶ ***"Causally-connected"*** means that changes made to the self-representation are immediately mirrored in the underlying system's actual state and behavior, and vice-versa

Why Reflection in Middleware ?

- ▶ Support for introspection
 - ▶ The ability to inspect the structure and behaviour of the system
 - ▶ e.g. dynamic monitoring or accounting
 - ▶ Autonomic computing – self-healing, self-optimising
 - ...



- ▶ Support for adaptation
 - ▶ Short term dynamic re-configuration
 - ▶ e.g. changing protocol configuration
 - ▶ Longer term evolution
 - ▶ e.g. adding new multimedia service



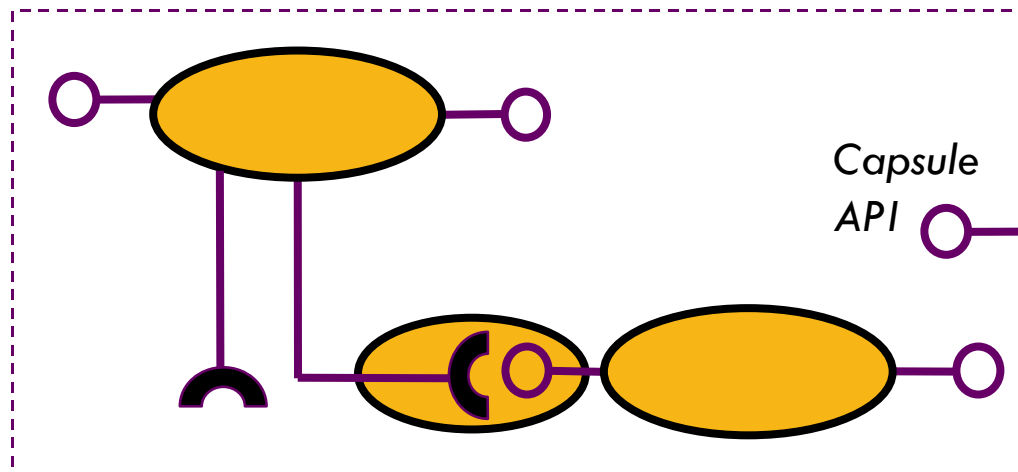
The Lancaster Approach: A Marriage of Three Technologies

- ▶ **Components**
 - ▶ Middleware is built from components
 - ▶ A distributed component model would be built from components
- ▶ **Component Frameworks**
 - ▶ Domain-specific 'life-support environments' for plug-in components
- ▶ **Reflection**
 - ▶ Use reflection to access structure and behaviour of the underlying middleware platform



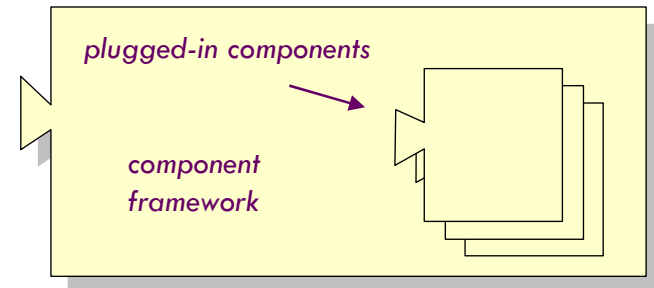
The OpenCOM v2 component model

- ◆ central concepts:
component | *capsule* | *interface* | *receptacle* |
binding
- ◆ use of IDL interfaces to give language independence



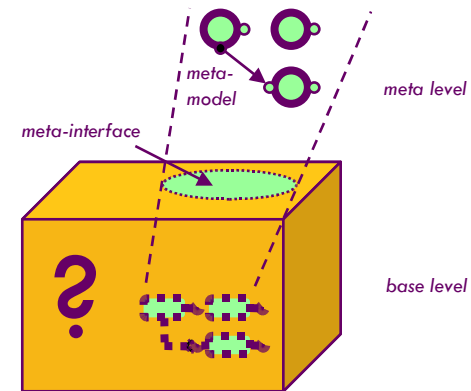
Component frameworks (CFs)

- ◆ middleware is built uniformly in terms of CFs
 - ▣ reusable run-time “life support environments” for plug-in components in a particular area of application
 - ▣ protocol stacking | routing | buffer management | job scheduling ...
- ◆ CFs offer
 - ▣ coarse-grained system structuring
 - ▣ system configurability
 - ▣ constraint on pluggability, reconfigurability
- ◆ have investigated various instantiations
 - ▣ ADL
 - ▣ MDSD approach



The reflective extensions

- ◆ reflection yields *openness* (inspection | adaptation | extension)
- ◆ use of multiple optional orthogonal *reflective meta-models*
 - ▣ architecture
 - represent the topology of a composition of components within a capsule
 - used for topological inspection, adaptation, extension
 - 'graph-oriented' meta-interface
 - ▣ interception
 - interpose interceptors in bindings
 - useful, e.g., for lightweight monitoring
 - ▣ interface
 - dynamically discover details of a component's interfaces/ receptacles
 - dynamically invoke dynamically-discovered interfaces



Other Related Work

- Systems based on Dynamic AOP
 - Invasive approaches
 - Typically use byte-code transformation
 - E.G. Midas/ Prose
 - Non-invasive approaches
 - Typically use behavioural reflection
 - E.G. Lasagne

Discussion point: What is the relationship between reflective and dynamic AOP based approaches?

Aspect-Oriented Programming

“There are some design decisions that are hard to cleanly capture because they *crosscut* the system’s basic functionality. We call the issues that these features address *aspects*...”

[Kiczales 1997]

Aspect

Top-level of encapsulation

Pointcut

Can identify static or dynamic points

Used to identify execution points ([joinpoint](#)) where aspect code should be executed (method calls, executions, etc.)

Advice

Contains the actual code to implement the concern

Conceptually similar to a method

Before, after, around execution points

Comparing AOP and reflection

Technology	Cross cutting	Adaptation	Self-Adaptation
Reflection	Poor	General	Yes
AOP	Strong	Aspects only	No

- Both technologies can benefit from the other;
 - ▣ Improving the management of crosscutting behaviour in reflective systems
 - ▣ Building self-adaptive aspect-based systems
- Multi-viewpoint adaptation

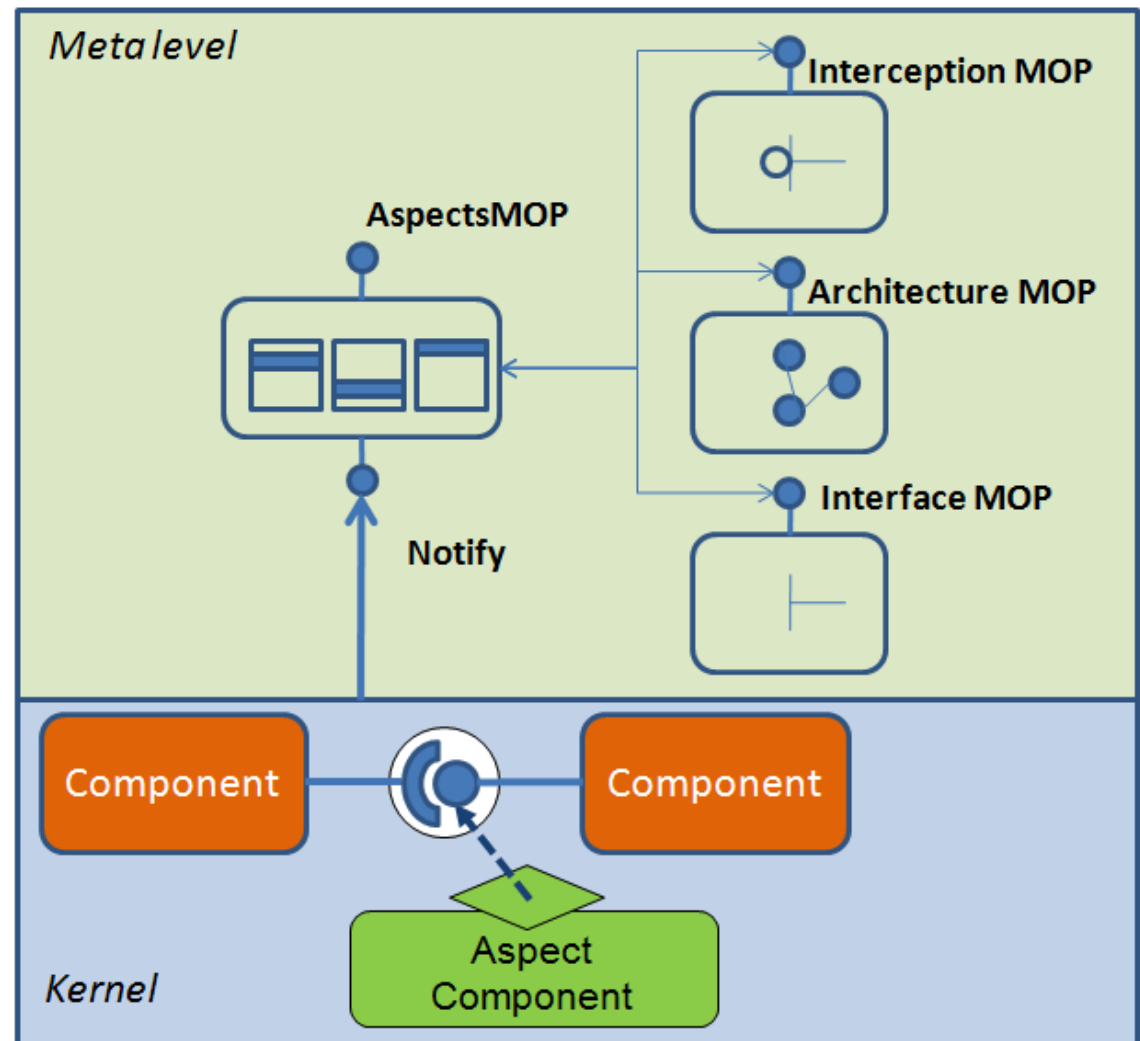
Aspect and Reflection Architecture

➤ With Bert Lagaisse, Eddy Truyen, Wouter Joosen

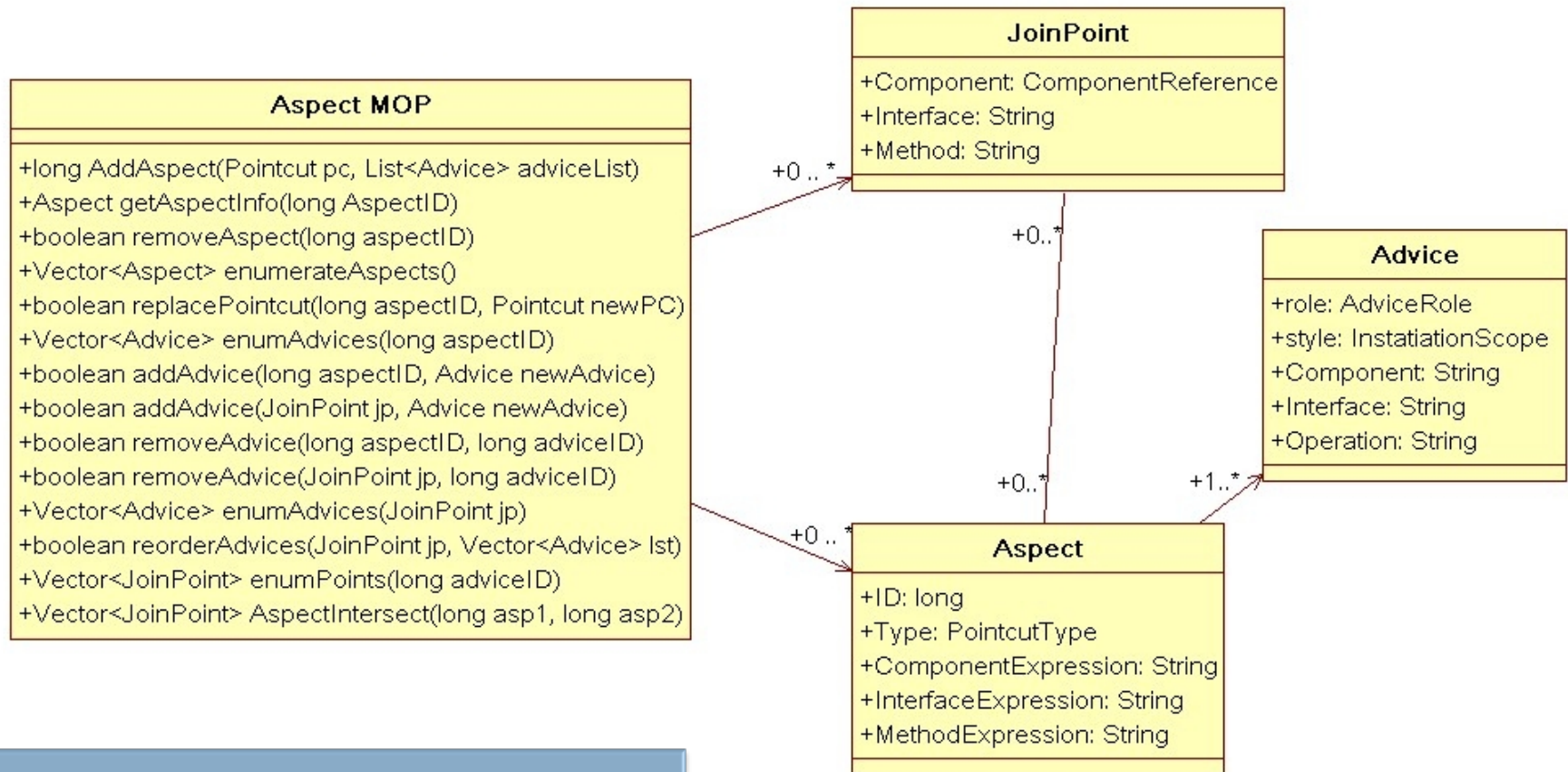
➤ Fine-grained introspection and adaptation of cross-cutting behaviour

➤ Aspects can be added to a system at run-time using the Aspect MOP as the implementation of this MOP is underpinned by existing reflective MOPs

➤ Reconfiguration aspects can be deployed using the Aspect MOP that abstract over reflective MOPs



The Aspect Meta-Object Protocol



Introspection and adaptation
operations

Meta-Representation

From Gordon's Minema Summer School talk

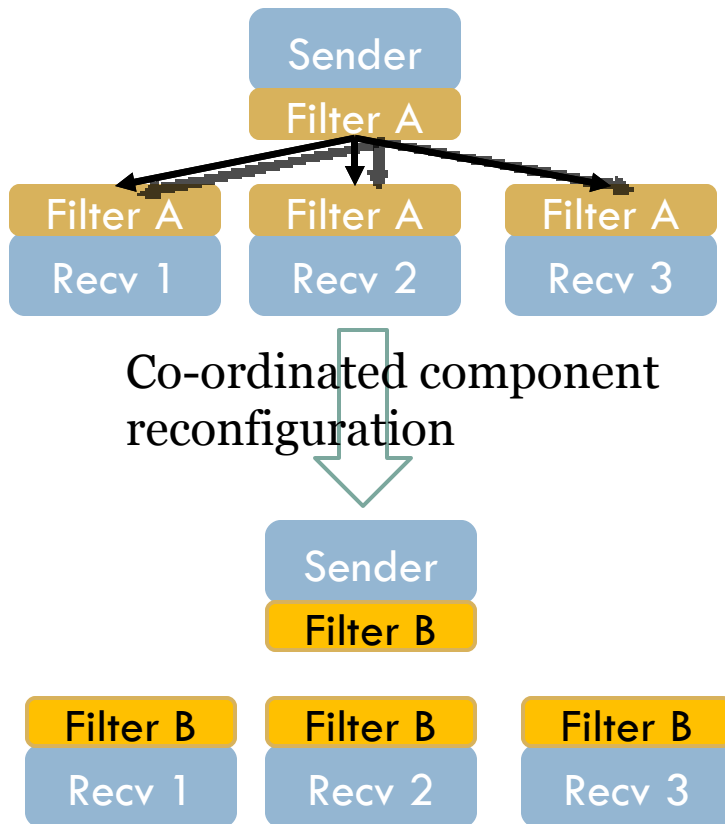
Future Research Challenges#1

- Co-ordinated adaptation
 - From local to co-ordinated adaptation
 - Across the layers
 - Between peers
- Autonomic computing
 - Implementing self-* properties
 - Exploring the relationship between autonomic pro and open architectures

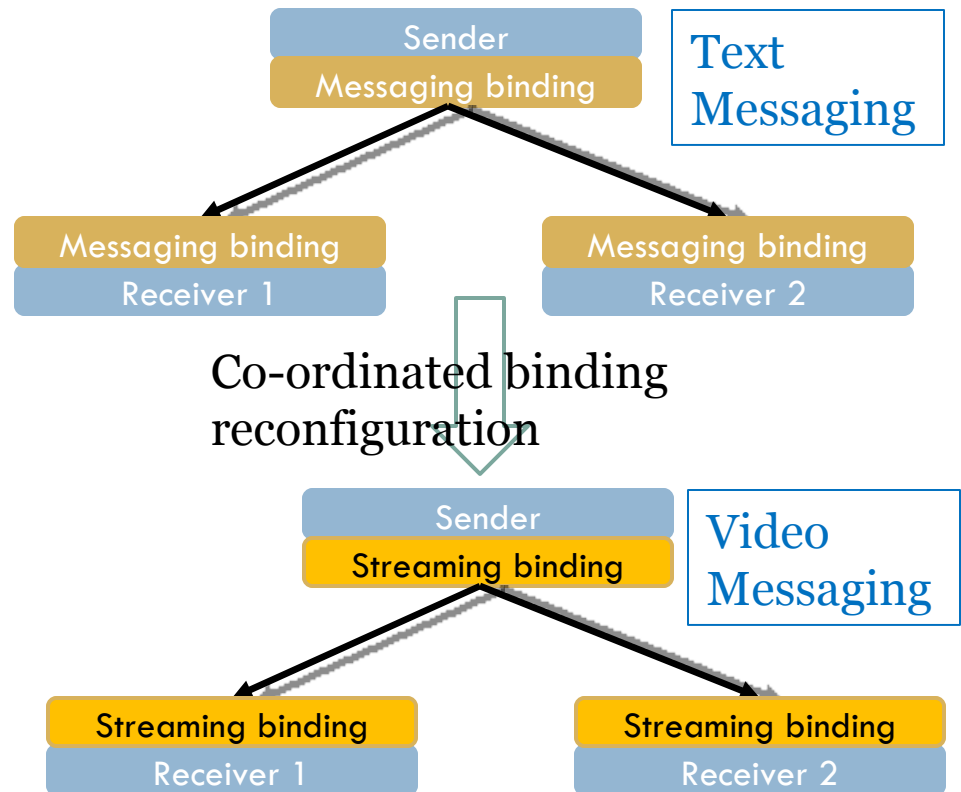


Example Middleware Reconfigurations

- Multimedia Stream (filter replace)



- Peer-to-peer messaging (binding style change)



Four Requirements for Co-ordinated Adaptation of Reflective Middleware

- Open Access
 - ▣ The ability to inspect and change the structure of software components across distributed nodes

- Consensus
 - ▣ Co-ordinating nodes must agree on the reconfiguration(s) to take

- Safe, valid reconfigurations
 - ▣ The distributed system must be in a safe state to reconfigure
 - ▣ Reconfigurations must be validated

- Flexibility
 - ▣ One reconfiguration mechanism doesn't fit all
 - ▣ Adaptation mechanisms should be flexible, often tailorable to the application type

Local Reconfiguration

□ Local OpenCOM component frameworks

□ Architecture MOP

- Local graph per framework

□ Validation

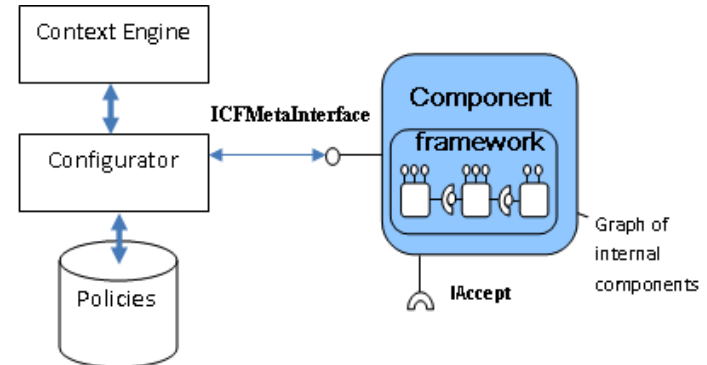
- Plug-in health check (IAccept)
- Roll-back for invalid changes

□ Quiescence

- Readers (method calls), writers (adaptation calls) access to framework

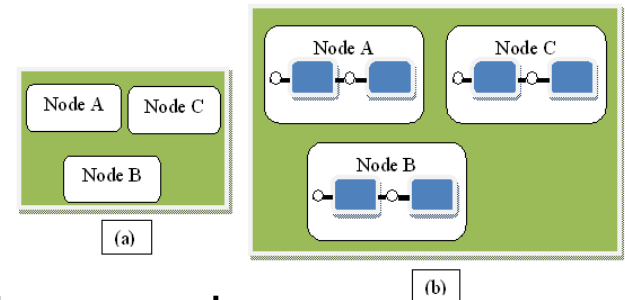
□ Configurators

- Context information selects policy (event-condition-action) to apply



Distributed Architecture MOP & Reification Strategies (Part 1)

- Distributed Component Framework
 - Set of local framework instances co-ordinating on a particular piece of middleware functionality
 - E.g. binding

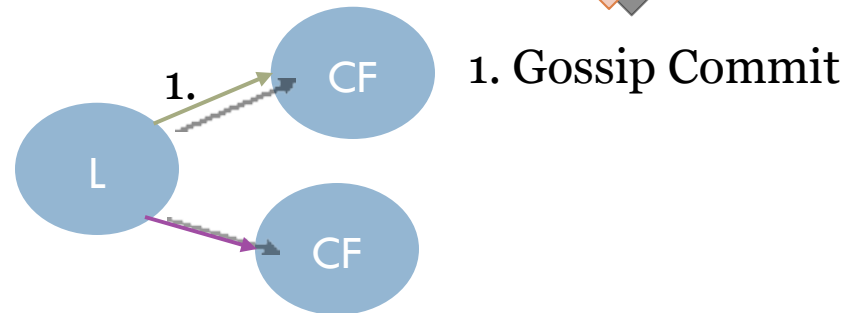
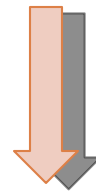
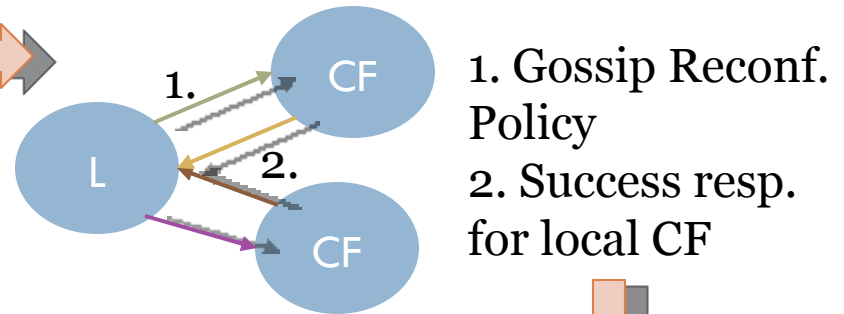
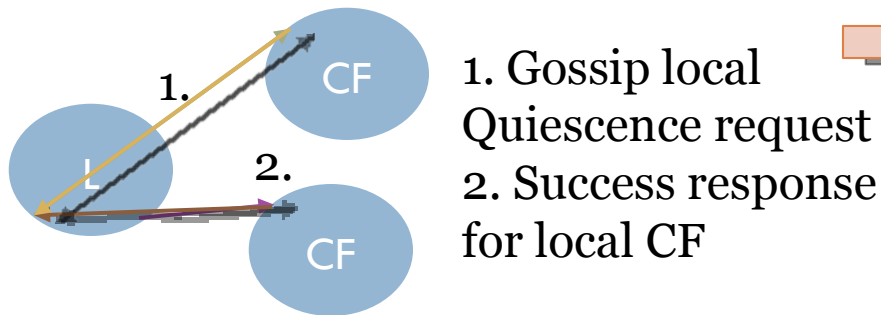


- Lightweight base protocol
 - Knowledge of nodes in the distributed framework
 - Implemented using lightweight group membership service
 - Plug-in appropriate protocol for environment
 - Initial implementation uses the SCAMP gossip protocol
- Build richer, flexible meta-architecture protocols on top
 - E.g. The set of connected individual component graphs
 - Push-model gossips local graph updates

Quiescence Management

- Centralised Quiescence [based upon Kramer & Magee]

- Place all nodes in the framework in a safe state



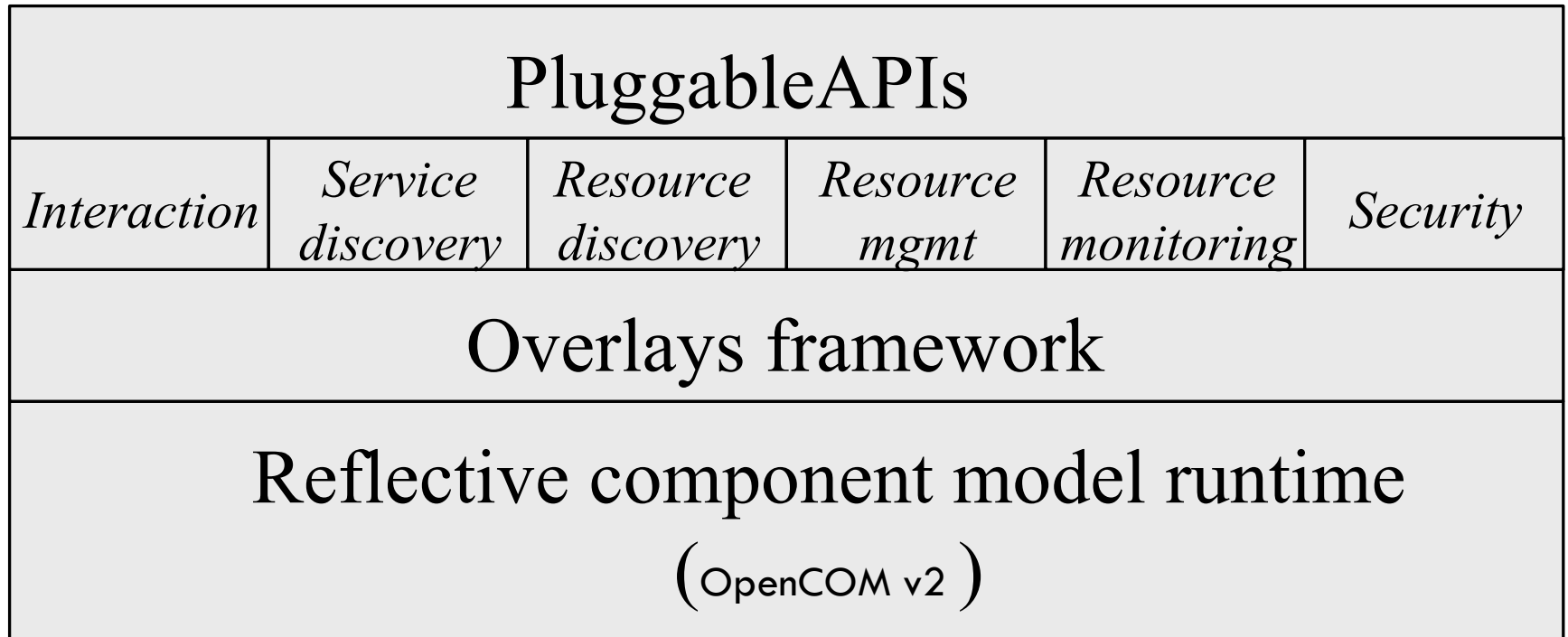
- Roll back at failed stages
- Use reflection to inspect and validate the committed changes across the nodes

Flexible Distributed Frameworks

- One size doesn't fit all
 - E.g. centralised
 - Resource Intensive
 - Not scalable to large distributed frameworks
 - Restrictive to place all nodes in safe-state concurrently
- Future Research
 - Investigate flexible decentralised approaches
 - Tranquillity (KUL), weaker quiescence
 - Distributed Configurators
 - Receive context events
 - Select reconfiguration policies
 - Distributed frameworks may have N configurators
 - Investigation of pluggable consensus protocols
 - Verification
 - Emergent behaviour
 - Adaptation cycles detection & resolution

Middleware solutions to SoS

The Overall Gridkit Architecture



Domain → Component Framework

Team

- Strategy
 - Gordon Blair, Geoff Coulson, Francois Taiani
- Architect, Developer, Integrator
 - Paul Grace
- Researchers (Domain Engineers)
 - Nelly Bencomo, Wei Cai, Carlos Flores Cortes, Phil Greenwood, Danny Hughes, Ackbar Joolia, Kevin Lee, Shen Lin, Lei Lui, Barry Porter, Rajiv Ramdhany, T. Sivaharan, Bhonalath Surajbali, Jo Ueyama, Nirmal Weerasinghe
- Students
 - Andrew Kirrage, Na Xu, Gareth Tyson, Sebastian Steinhauer

Domain Engineering

- “Practice of collecting past experience in a domain in the form of reusable assets”
 - ▣ Subsequently easy to reuse
- Specialised areas of middleware functionality
 - ▣ PubSub, Group, Service Discovery
- Domain specific configurations
 - ▣ Software architecture
- Domain specific reconfigurations
 - ▣ Behaviour transitions

Resource Discovery Framework (Carlos Flores Cortes)



Internet

Fixed-Wired

Wireless

Ad-hoc

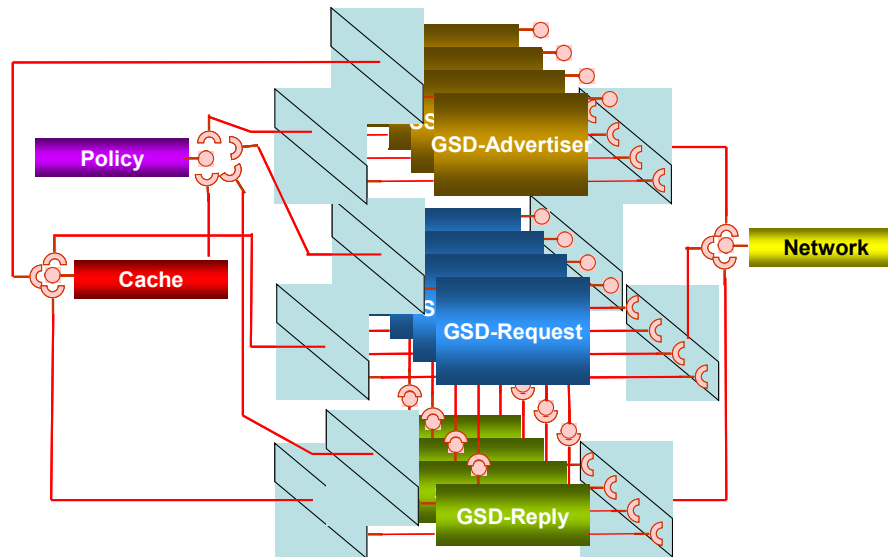
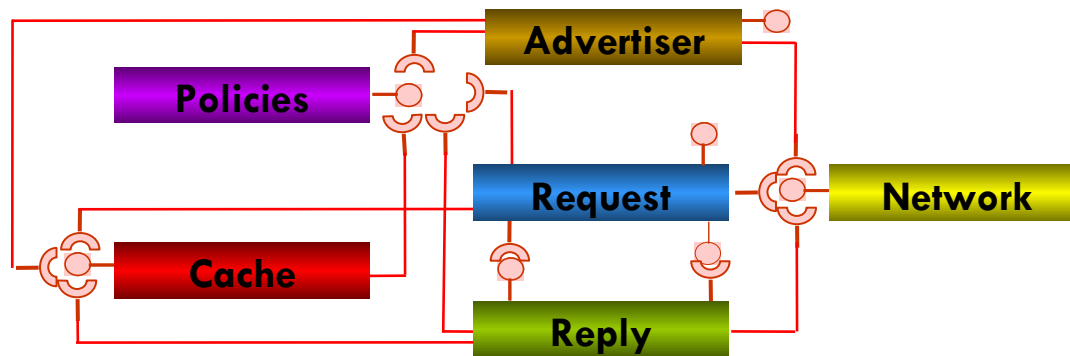
DISCOVERY PLATFORMS

UDDI, INS

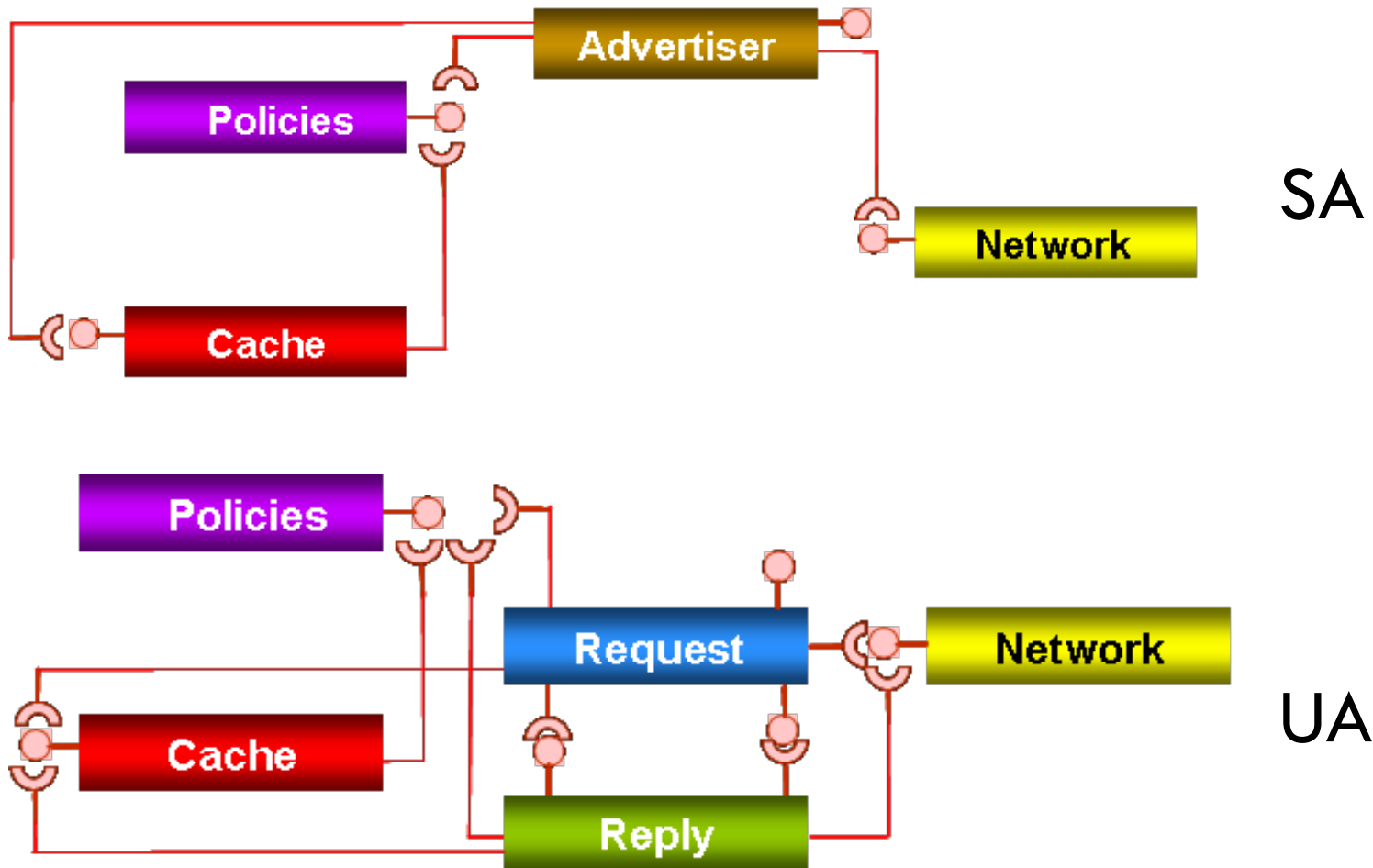
SLP, UPnP, Jini, Salutation,
Bonjour

SSD, GSD, Allia,
SLP-B, Bluetooth,
Lanes, Service
Rings, Splendor,
Ariadne

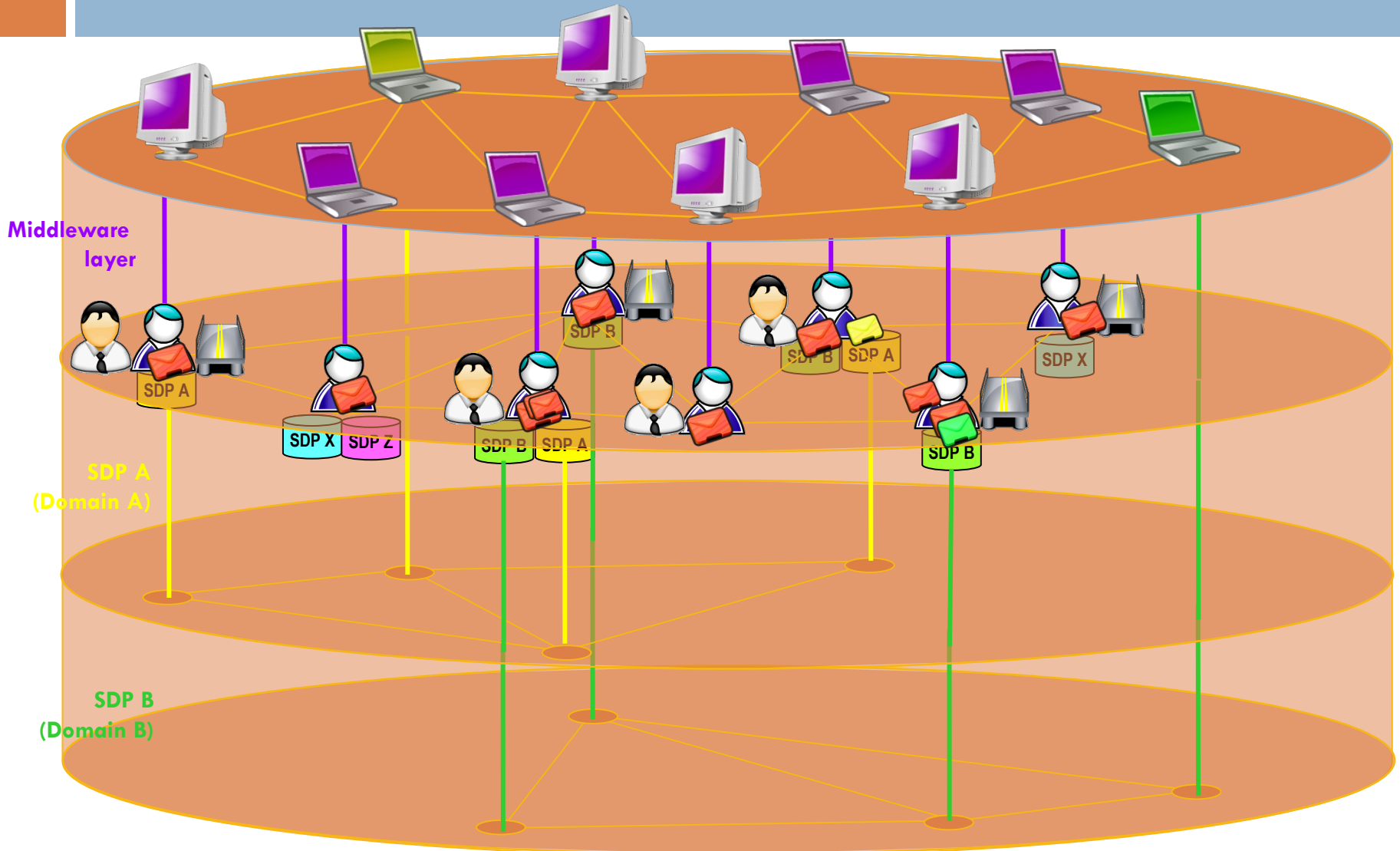
A Multi-protocol Framework for Service Discovery



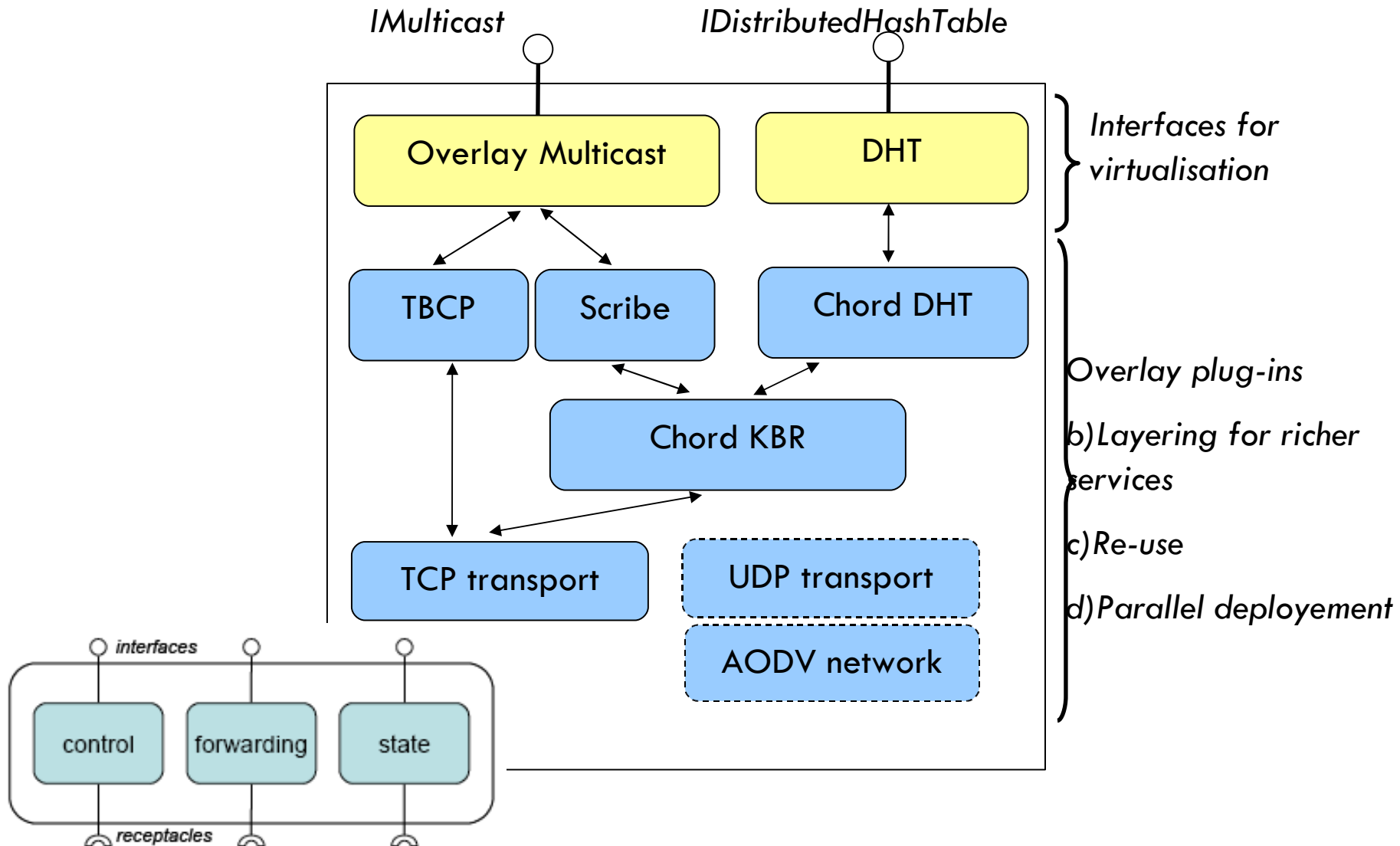
Roles



Cross Domain Service Discovery



Overlay Networks (EuroSys: Wed 4pm)



More Domains

- Ad-hoc routing
 - ▣ Rajiv Randhamy
- Gossip
 - ▣ Shen Lin
- Sensor Networks
 - ▣ Danny Hughes
- Publish Subscribe
 - ▣ T. Sivaharan
- Dependability
 - ▣ Barry Porter
- Resource Management
 - ▣ Wei Cai

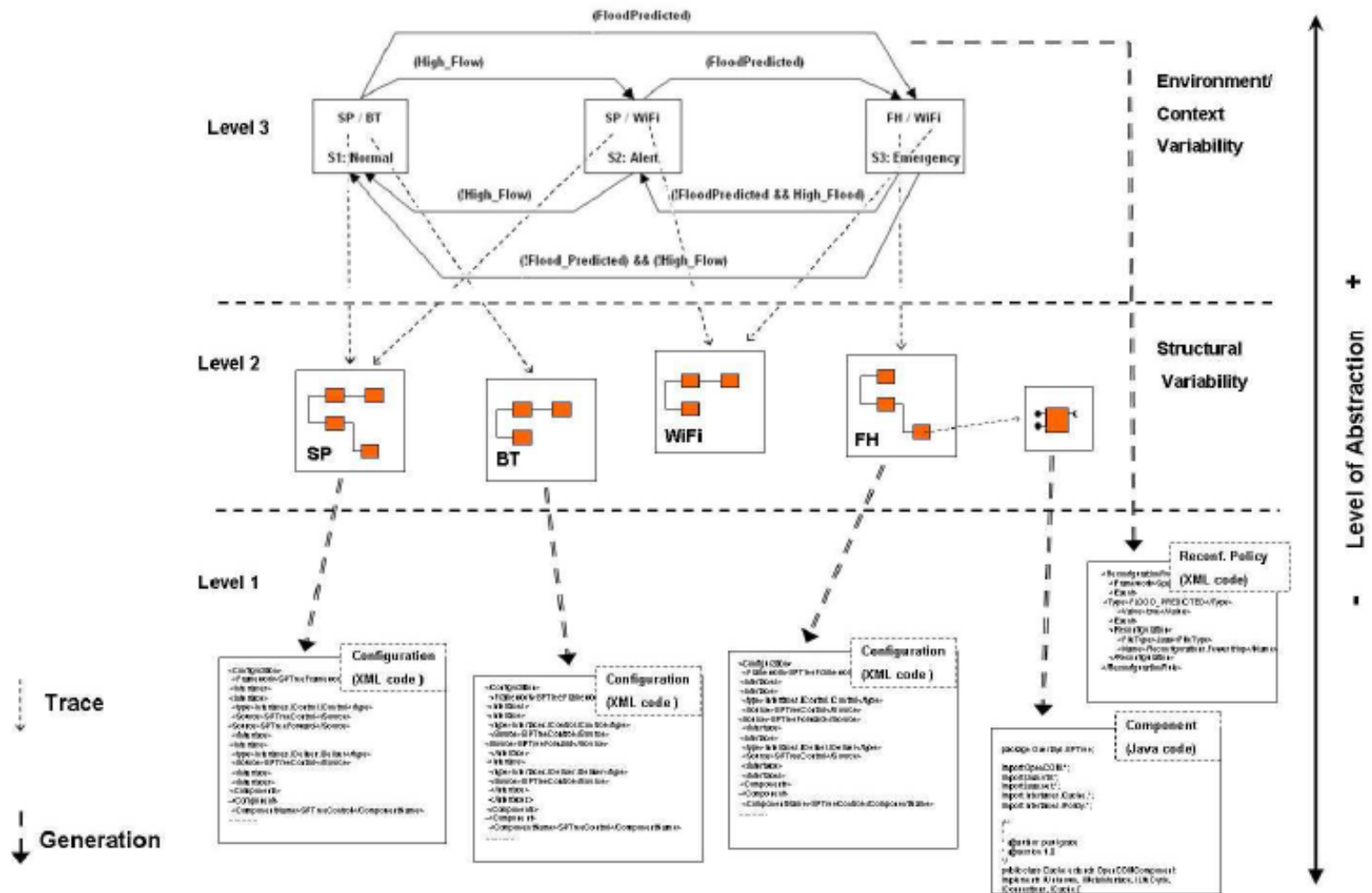
Some Observations

- **We have created highly complex software**
 - **Increasingly difficult to maintain and use!**
 - Need to tame the beast and provide principled software engineer approaches to handling configurability and reconfigurability
- **Propositions**
 - Variability in underlying system model, e.g. loading, binding, support for reflection
 - Variability in network infrastructure, interaction types, service discovery security, dependability
 - Added dimension of systems of systems (and mapping between them)
- **Potentially a key role for models in managing this variability**

The Role of Models

- Nelly Bencomo
- Models@design.time
 - Recognising the concept of middleware families
 - Generating instances of the family from higher level model descriptions
- Models@runtime
 - Maintaining a representation of the associated model during the execution of the system
 - A causally connected self-representation driving re-configuration or autonomic management

Higher Levels of Abstraction



Genie Tool

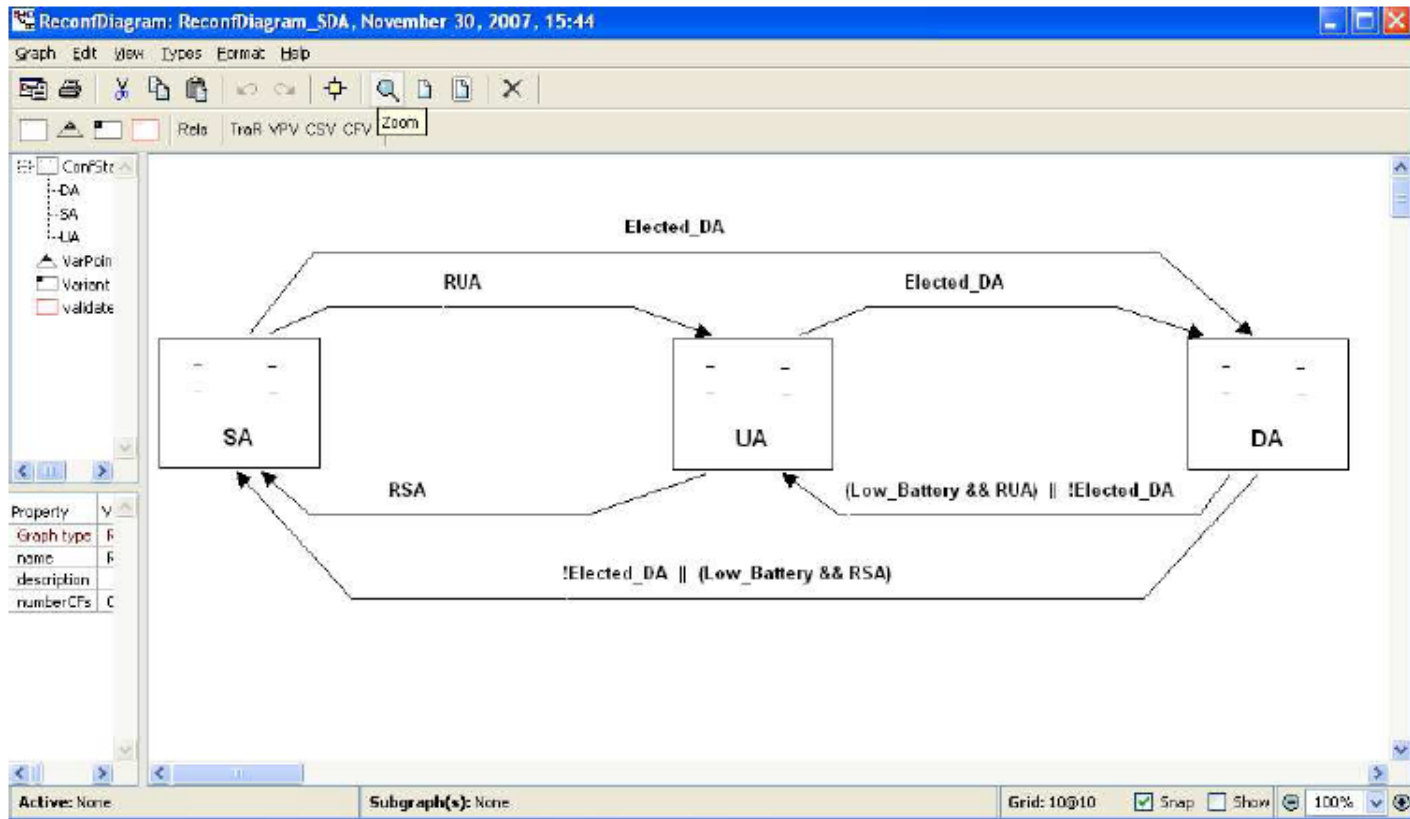
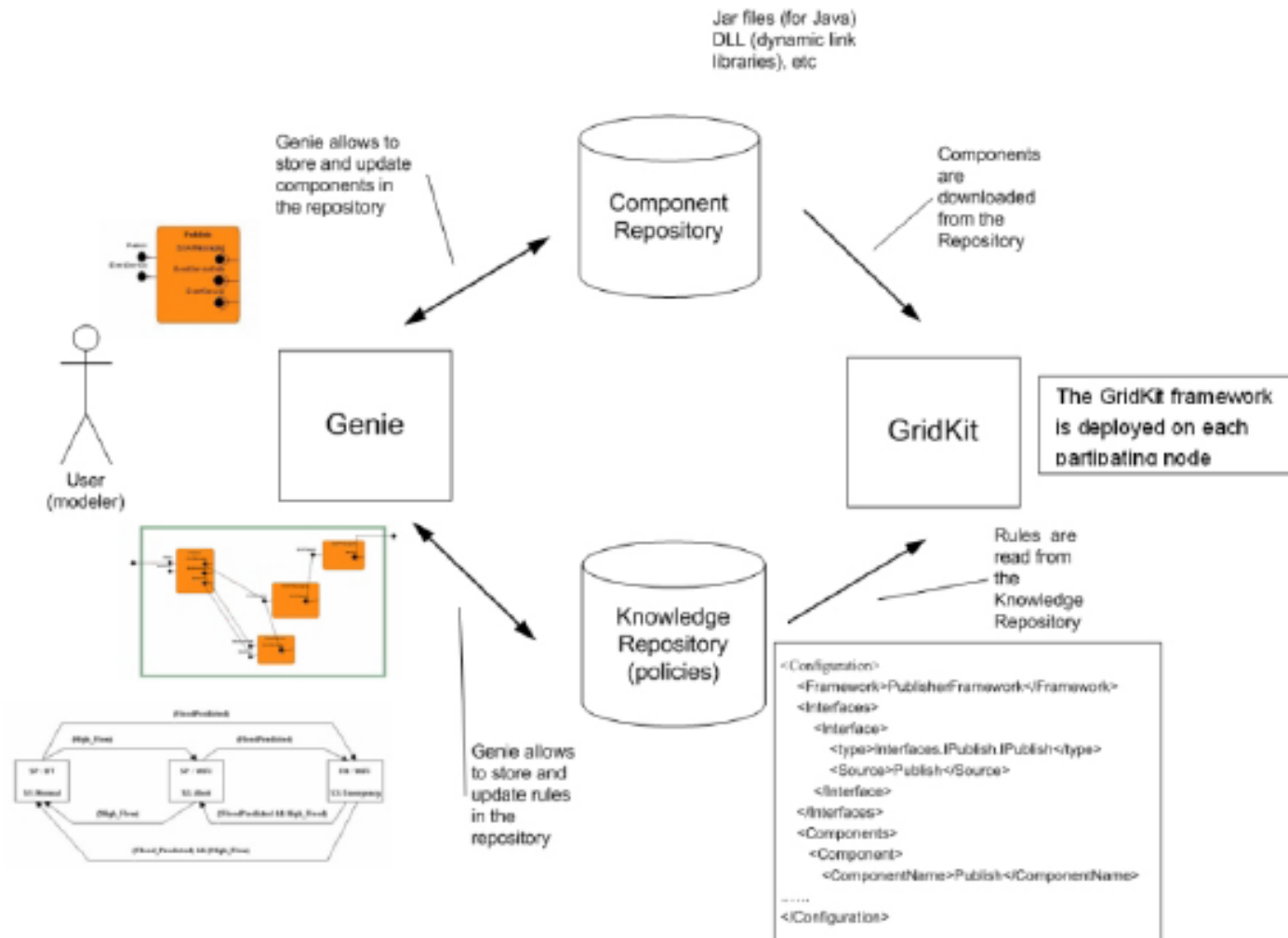


Figure 10: Transition diagram model for the Service Discovery Application.

Integration with Gridkit



Conclusions

- Systems of Systems pose many challenges
 - Adaptive middleware solutions demonstrate potential in this area
 - Increasing complexity
 - Embrace software engineering solutions
 - Modelling can raise the level of abstraction for developing complex systems software
 - **<http://gridkit.sourceforge.net>**