Self-Organising Software Architectures for Distributed Systems

Ioannis Georgiadis, Jeff Magee and Jeff Kramer

Department of Computing
Imperial College London
180 Queen’s Gate, London SW7 2BZ, UK.

Self Organising Software Architecture

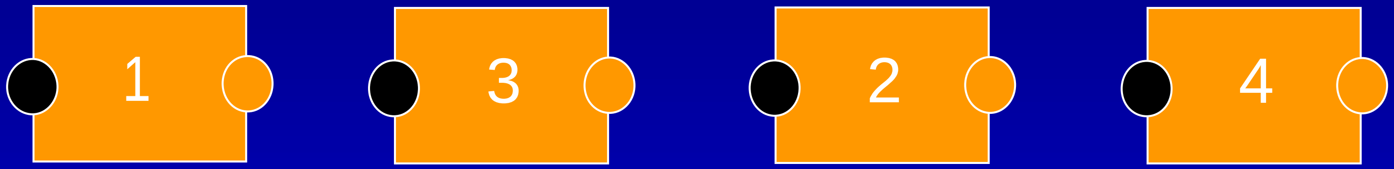
A **self-organising** architecture is both **self-assembling** and **self-healing.**

Self-assembling - initially, a set of component instances organise their interaction to satisfy architectural specification.

Self-healing - components collaborate to satisfy required architectural properties after failure/ change in the environment.

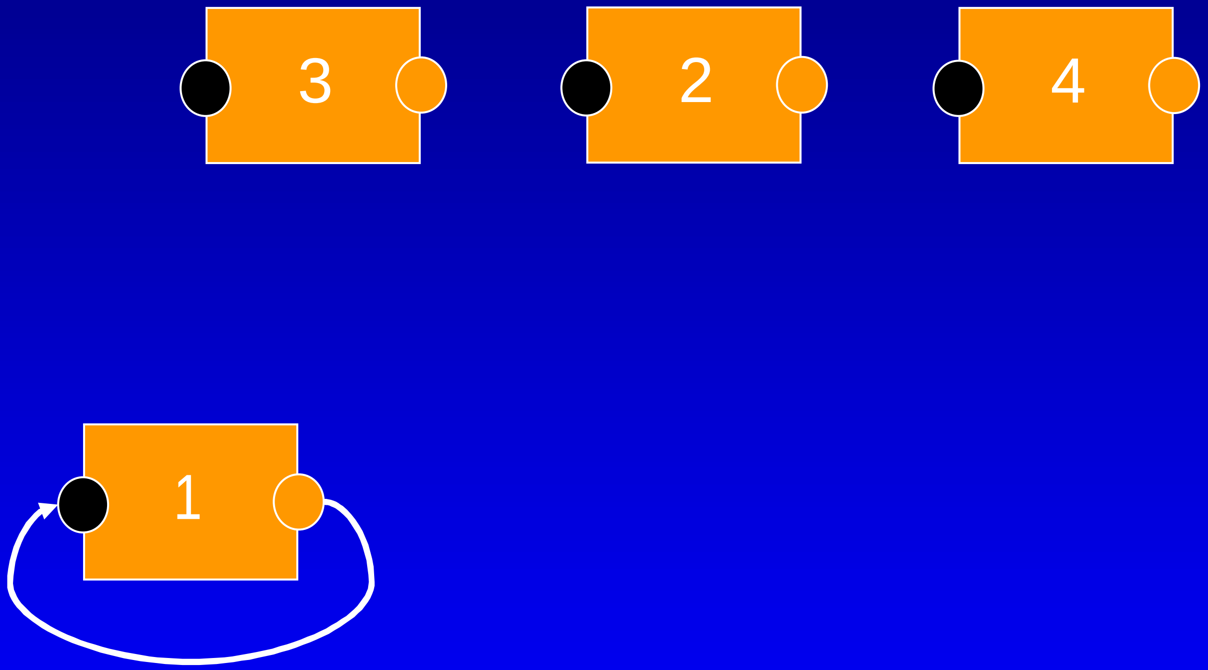
Objective is to minimise explicit management

Self Assembling



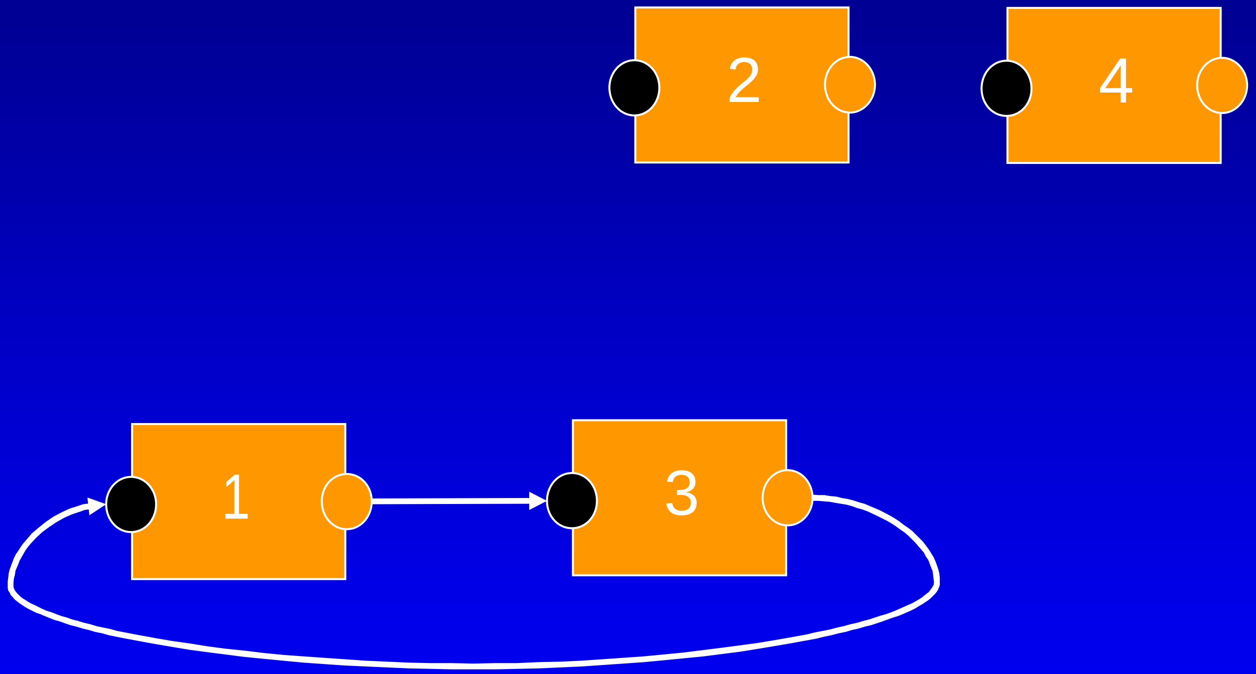
Ordered Ring Architecture

Self Assembling



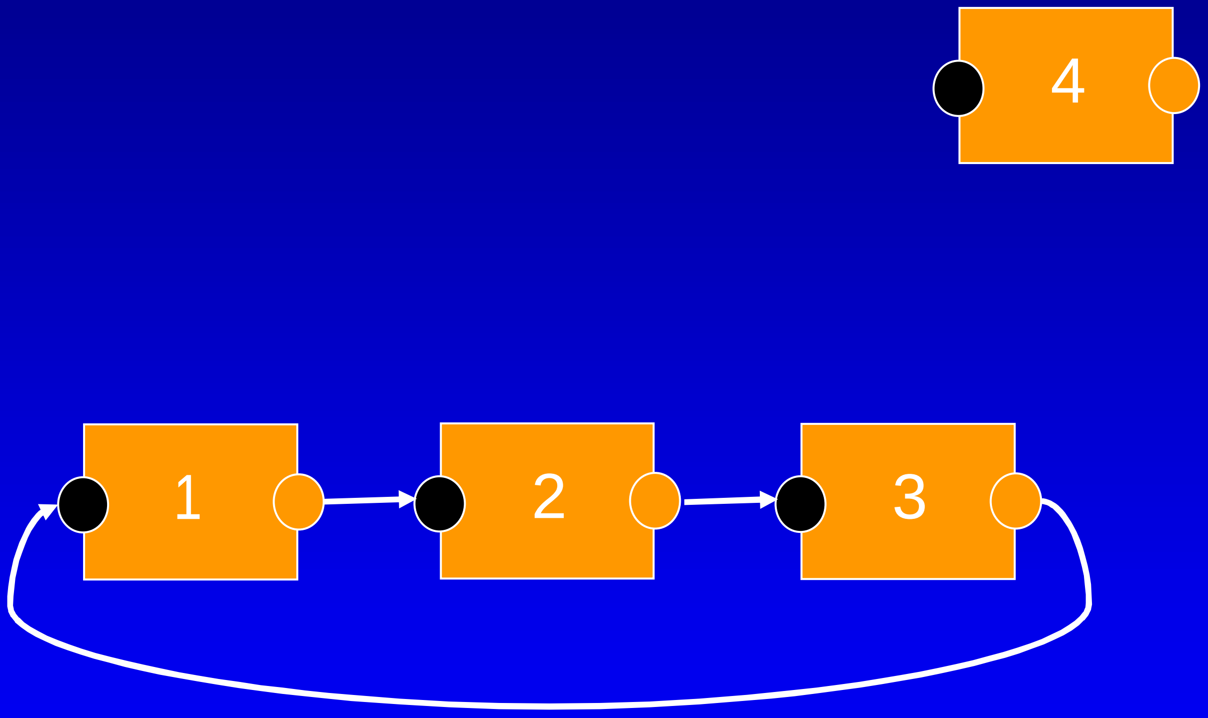
Ordered Ring Architecture

Self Assembling



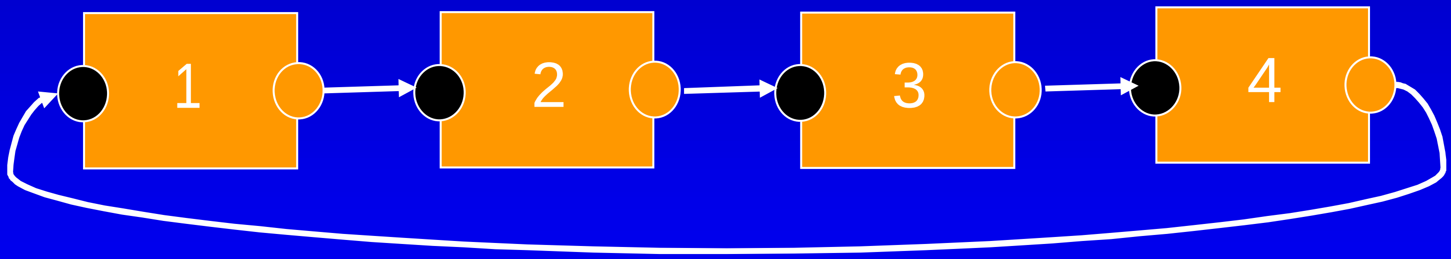
Ordered Ring Architecture

Self Assembling



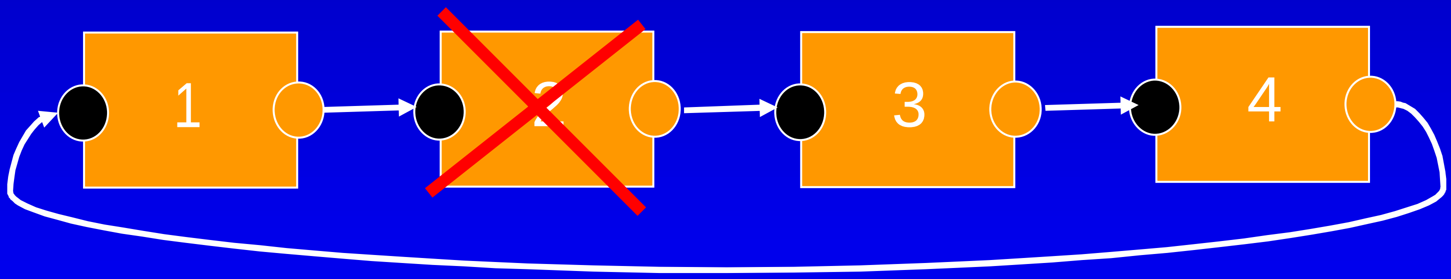
Ordered Ring Architecture

Self Assembling



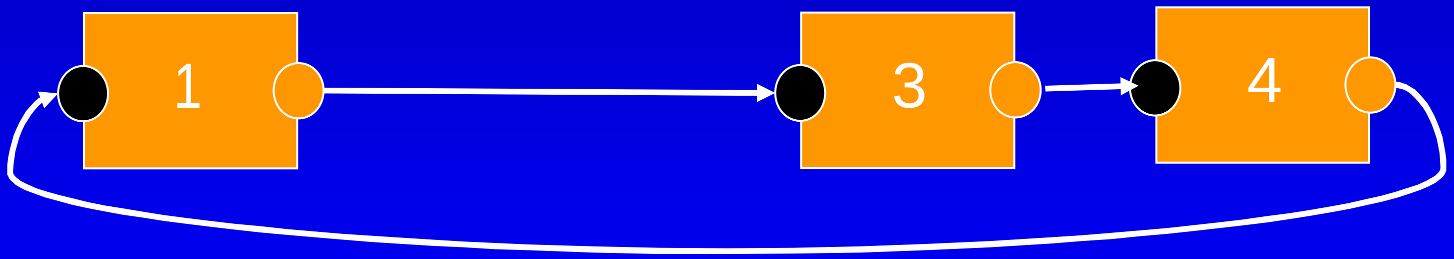
Ordered Ring Architecture

Self Healing



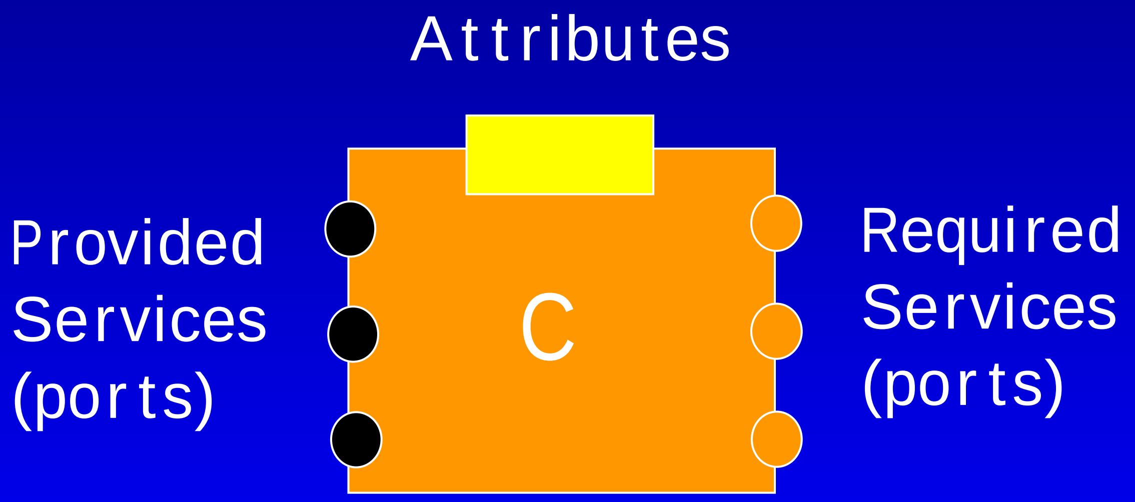
Ordered Ring Architecture

Self Healing



Ordered Ring Architecture

Component Model



Architecture Specification

Architecture is specified by a set of constraints on structure and attribute values.

A component must satisfy these constraints before joining a system.

Using Alloy

An input port is connected to exactly one output port:

```
RingComp.ringInp.bind in RingComp.ringOutp
all c:RingComp | one c.ringInp.bind
```

All ring components form a single chain:

```
some c:RingComp | c.*ringConn = RingComp
```

Design approach

Self-configuration: *A sequence of* ***internal*** *actions to create an architecture that conforms to its specification (style)*

$$G_{start} \xrightarrow{A_e} G \xrightarrow{A_i} G_{end}$$

External Actions

$$a_e = \begin{cases} attrib(r,v) \\ join(c) \\ leave(c)/\ fail(c) \end{cases}$$

Internal Actions

$$a_i = \begin{cases} bind(p_i, p_j) \\ unbind(p_i, p_j) \end{cases}$$

Selector function

Divide Component Integration Process Into Port Integration
A required port is bound to **at most one** provision

Selector Function (Selector)

$$selector(p) : G \xrightarrow{a_i^p} G'$$

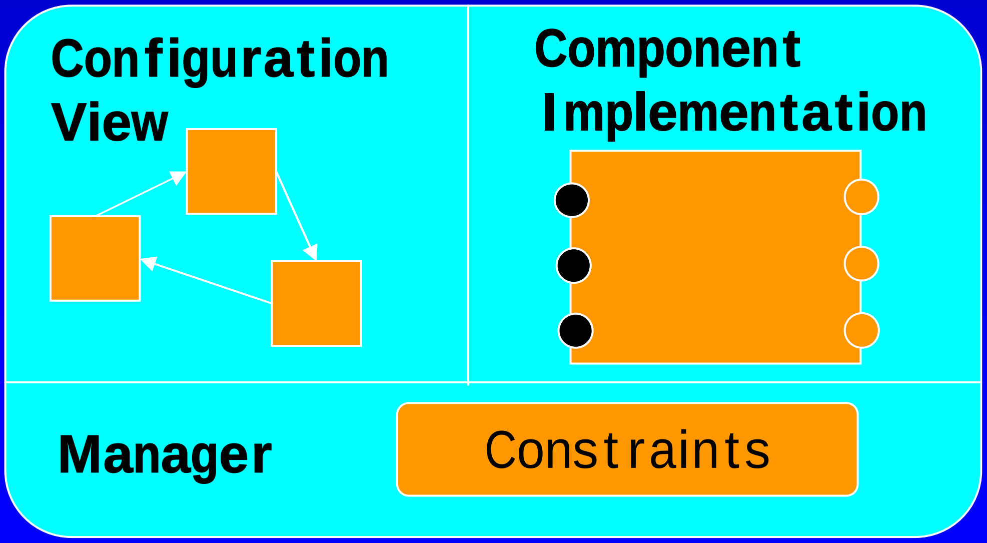
Configuration: A sequence of selector invocations

$$G \xrightarrow{a^{p1}} G_1 \xrightarrow{a^{p2}} \Lambda \xrightarrow{a^{pn}} G_{end} \; , \quad \begin{array}{l} required\ ports\ \ p_1, \mathrm{K}\ , p_n \\ internal\ actions\ \ a^{p1}, \mathrm{K}\ , a^{pn} \end{array}$$

Implementation Experiment

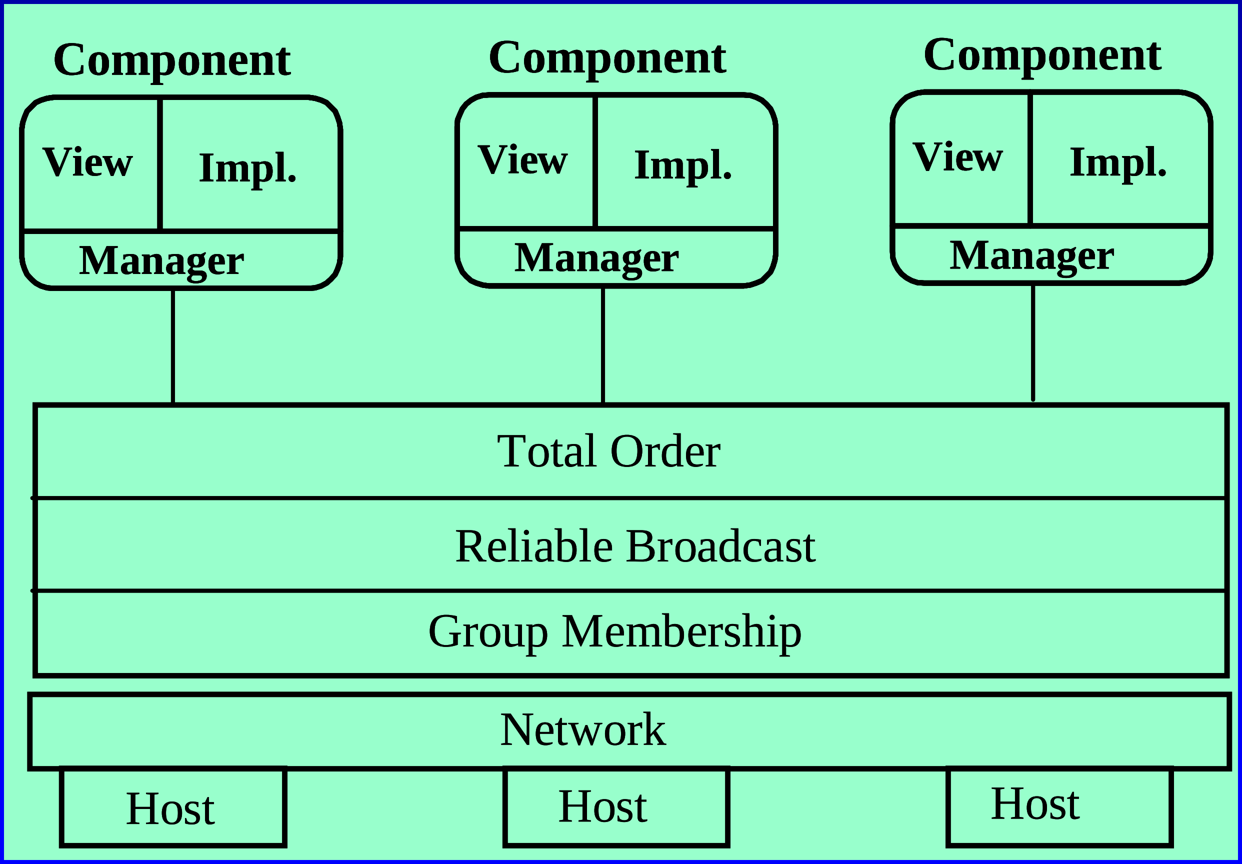
Fully distributed implementation with no centralised control.

Each component is created with the set of system constraints and maintains a view of the system.



Implementation approach

Total order atomic broadcast required to maintain view consistency.



Results so far

- + Alloy permits consistency checks on architecture specification.
- + Decomposing constraint satisfaction into per port selector functions permits “Style composition”.
- + Attributes are good generalising abstraction for internal component state change.
- - Need to relax consistency of architectural view for scalability.
- - Design of “Selector function” using graph grammars not satisfactory.

Related work

- Graph Grammars/ Structural Constraints
  - Metayer, Hirch-Inverardi-Montanari
- Chemical Abstract Machine
  - Inverardi-Wolf, Wermelinger
- Raven - reconfiguration & constraints
  - Coatta-Neufeld
- Self-adaptive C2
  - Oriezy-Gorlick-Johnson-Taylor-Medvidovic
- Armani & Self-repairing systems
  - Schmerl-Garlan