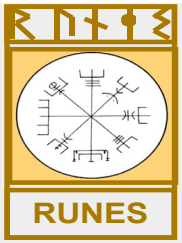


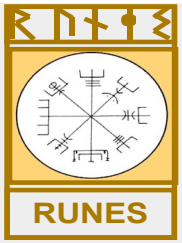
The RUNES Middleware System

The RUNES EU Project

Paolo Costa, Luca Mottola, Gian Pietro Picco
Dip. Di Elettronica ed Informazione
Politecnico di Milano

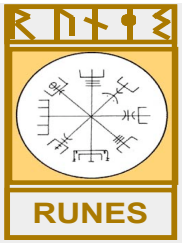
Geoff Coulson
Department of Computing
Lancaster University

Cecilia Mascolo, *Stefanos Zachariadis*
Department of Computer Science
University College London



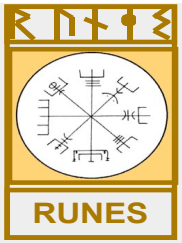
Outline

- Background and Motivation
- RUNES Component Metamodel
- RUNES Middleware Services
- Implementation
- Related Work
- Conclusions



Fire in the Road Tunnel!

- A Road Tunnel
  - Lots of sensors deployed
- Cars/Trucks Going Through
  - Vehicles/People have portable devices
- Crash!
  - Road Tunnel Closes
- Emergency Team Goes In
  - Equipped with life-saving PDAs
- How To Make Everything Work Together?

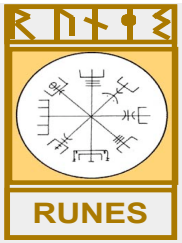


RUNES?

- Reconfigurable Ubiquitous Network Embedded Systems
  - inherently *heterogeneous* and *dynamic*
- the software of such systems tends to be ad-hoc
  - little provision for generalisable and reusable abstractions and services
- it's hard to develop for such systems
  - especially targeting multiple systems!

→ **need for a *generic programming platform***

  - need abstractions and services that can span the full range of networked embedded systems
  - need consistent mechanisms for *configuring*, *deploying*, and *reconfiguring* systems
  - must be small, simple, efficient and highly tailorable

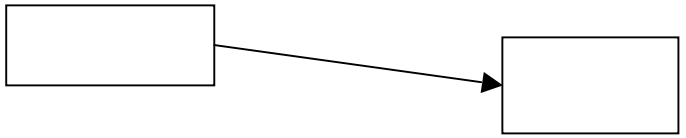


How Heterogenous?

- Sensor
  - IEEE 802.15.4 (250kbps), 10kB RAM, 48kB flash, 1MB storage, msp430 8MHz 16bit CPU
- PDA
  - IEEE 802.11b (11MBps), 128MB flash, 128MB RAM, 4GB storage, 400MHz StrongARM CPU

The Grand Scheme of Things

```
#include <runes.h>

void hello();
```



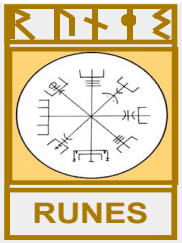
MDA/ADL Meta-Programming

IDL / Component Metamodel

Middleware | Middleware | Middleware | Middleware | Middleware | Middleware

C/Linux | Win32 | J2ME | TinyOS | Contiki | FreeRTOS

Automated



The Awful Truth

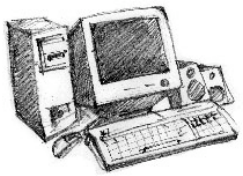
```
#include <runes.h>

void hello();
```

MDA/ADL Meta-Programming

IDL / Component Metamodel

Middleware | Middleware | Middleware | Middleware | Middleware | Middleware

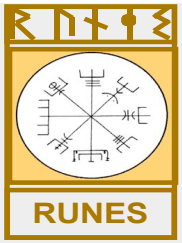
C/Linux | Win32 | J2ME | TinyOS | Contiki | FreeRTOS



Automated

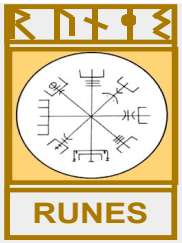
Runes Middleware

- Component-Based System Aiming To:
- Tackle Heterogeneity
  - Hardware Platforms
  - OS
  - Programming Languages
- Offer Reconfigurability
  - Software
  - Network
    - Allows Networking of Heterogeneous nodes



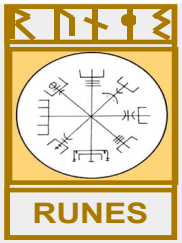
Reconfigurable

- Allows the static and dynamic reconfiguration of the nodes
  - Important because of the nature of the environment
- Static: different configurations installed
- Dynamic:
  - upload and offload of components and code dynamically
    - Optional
  - Dynamic component rebinding



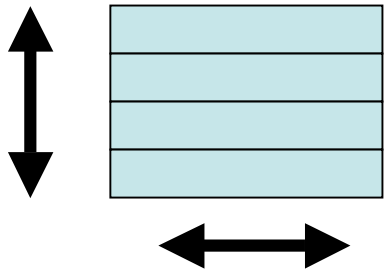
The RUNES Component Meta Model

- RUNES is a Component-Based Middleware
- Component-Based Implies Adherence To Particular Component Metamodel
  - The RUNES Component Metamodel
- Developed for devices with scarce resources in mind

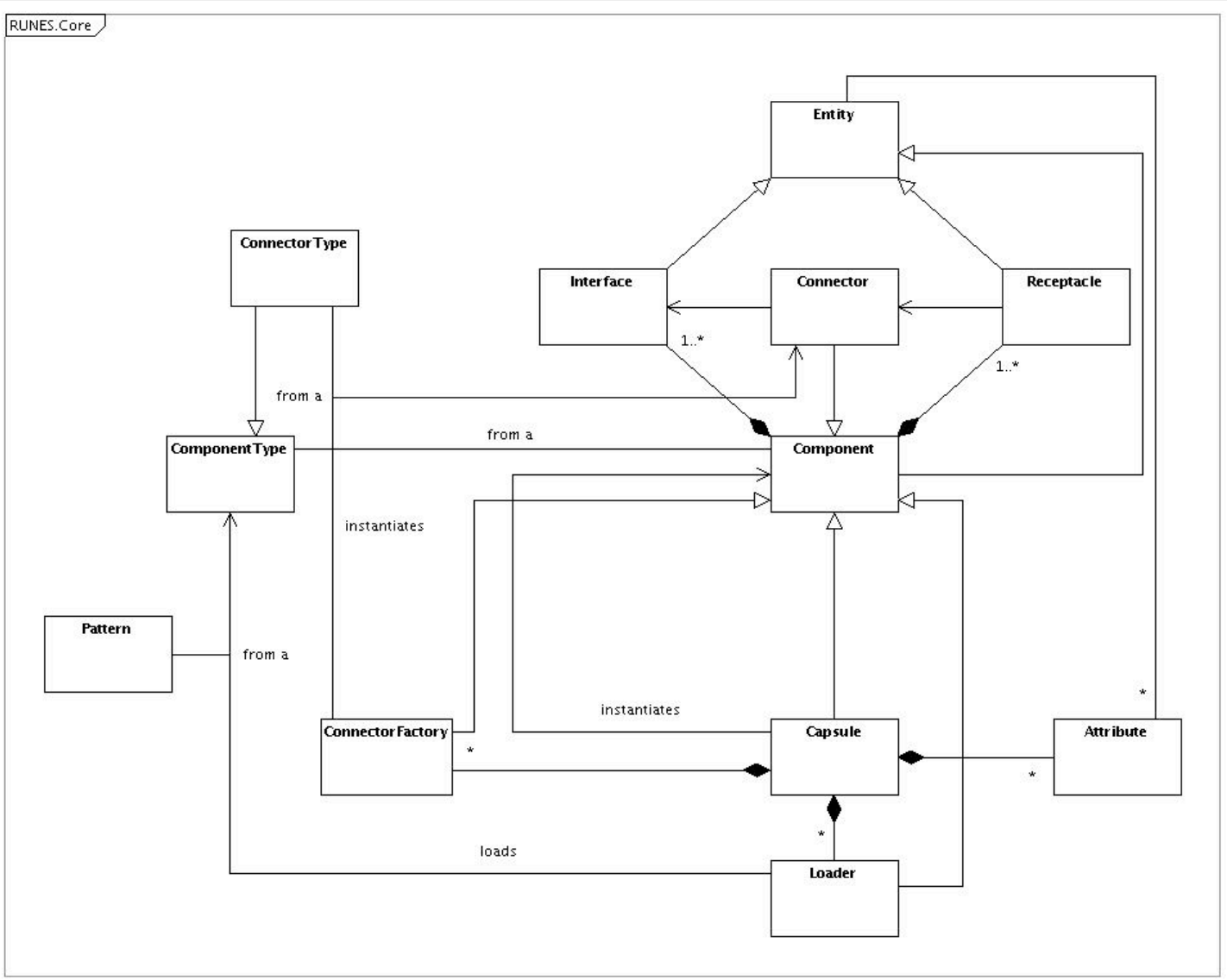


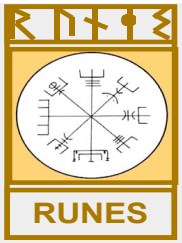
Aims

- a generic component-based programming model
  - Inspectable and adaptable at runtime
  - 'low level' and efficient; can employ different implementations on different hardware
- applied uniformly throughout the stack
  - network, OS, middleware, applications
  - *all above uniformly realised as reconfigurable compositions of components*



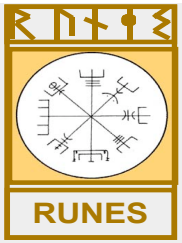
Component Metamodel





Metamodel Elements

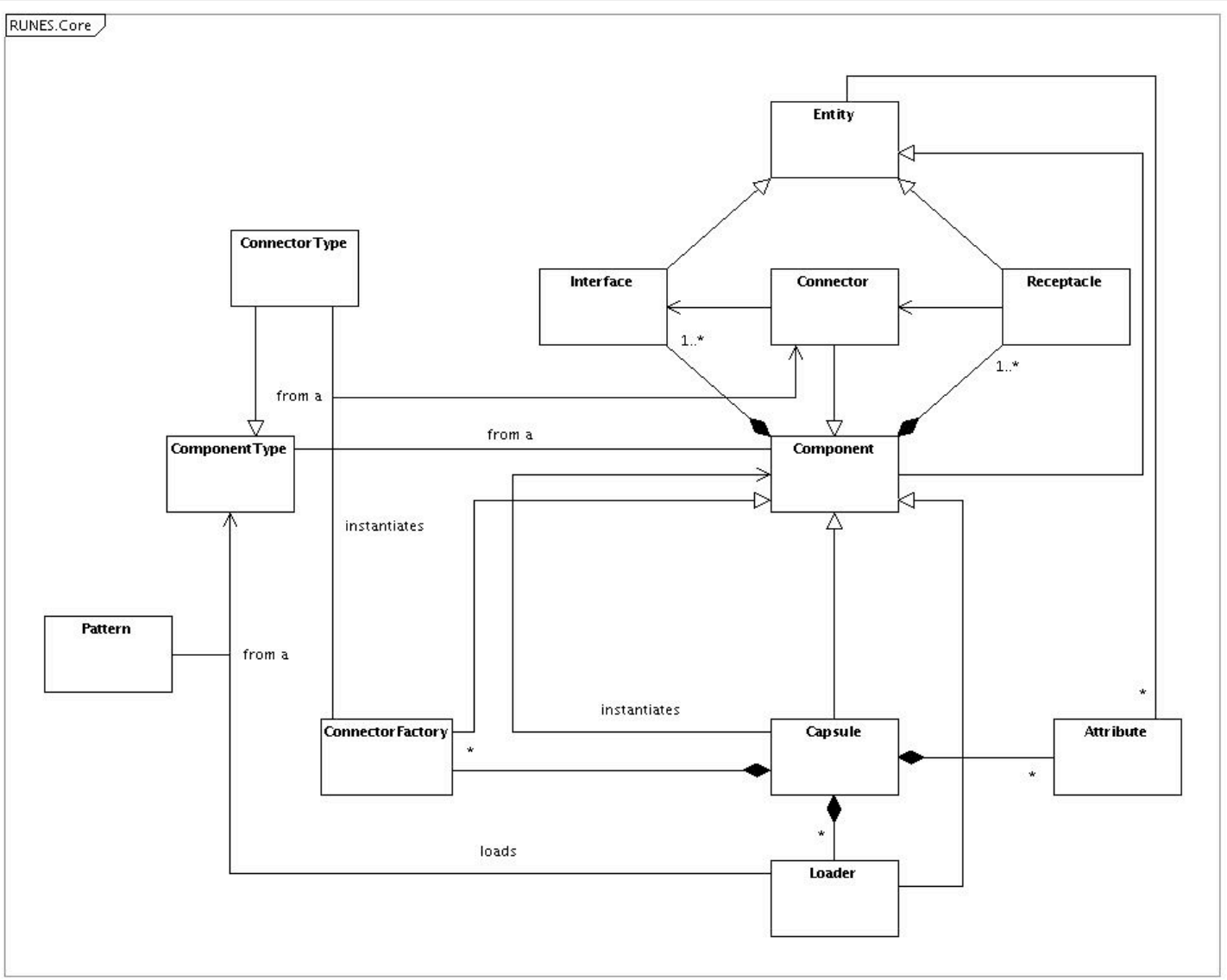
- central concepts:
  - *capsule*
  - *component*
  - *interface*
  - *receptacle*
  - *connector*
  - *connector factory*
  - *loader*
  - *registry*
  - *component framework*
- uses IDL for prog. language independence

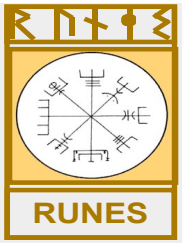


Elements (1)

- **Component** Components are encapsulated units of functionality and deployment that interact with other components exclusively through "*interfaces*" and "*receptacles*".
- **Interfaces** are expressed in terms of sets of operation signatures and associated data types (i.e. therefore each component must expose its public operations in a user defined interface).
- **Receptacles** A component must declare receptacles if it 'requires interfaces' from other components. Receptacles are used to make explicit the dependencies of a component on other components. A component can host zero or more receptacles depending on its needs.

Component Metamodel

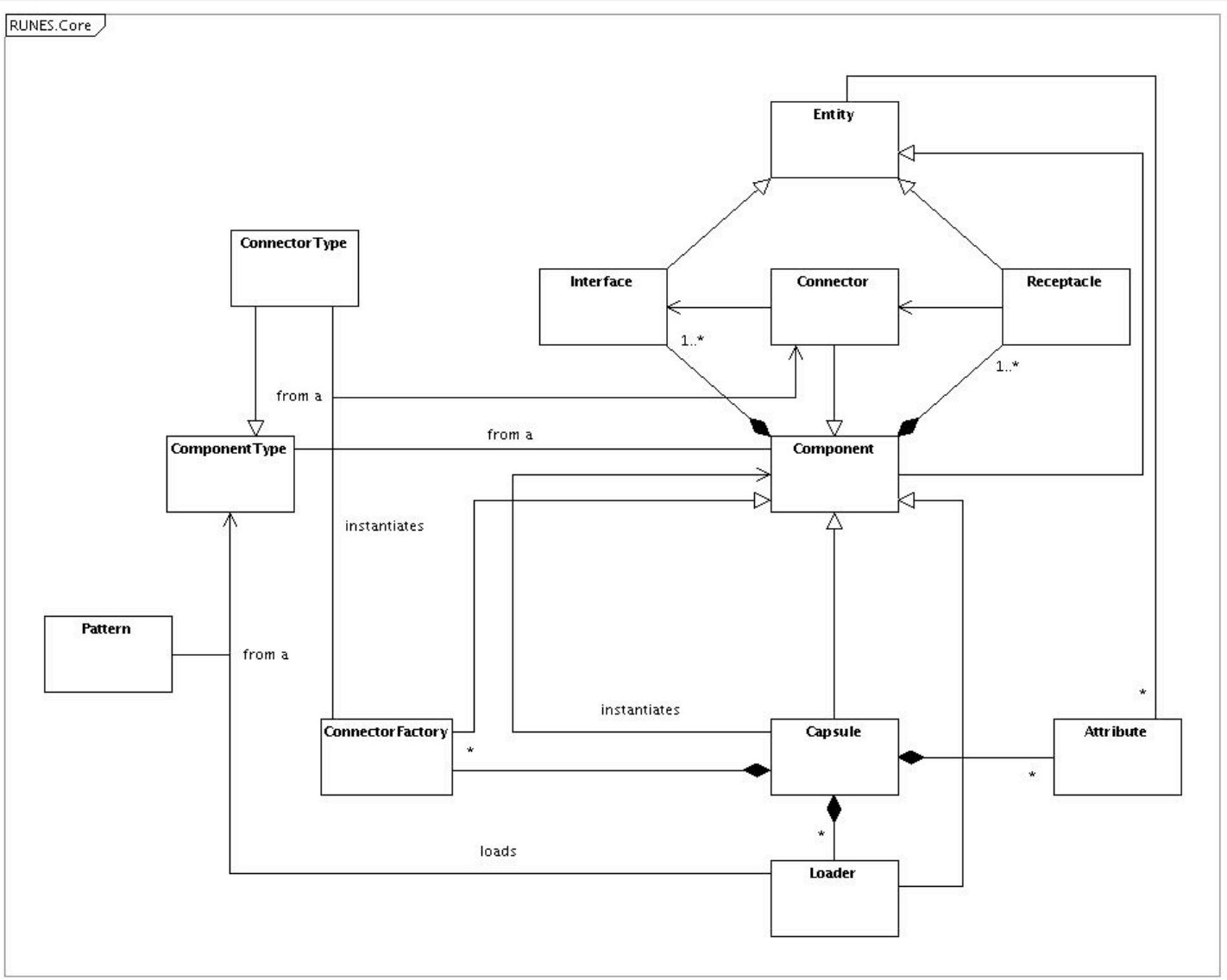


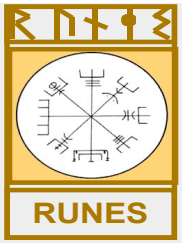


Elements (2)

- **Connector** is used to bind a pairs of receptacle and interface thus enabling the creation of a configuration of components.
- **ConnectorFactory** a component that creates connectors.
- **Capsule** the namespace where everything lives - a "container" for components. Also offers the main API
- **Registry** a memory manager - repository for metadata (attributes)
- **Loader** a component that Loads other Components
  - Components are loaded from **Patterns**

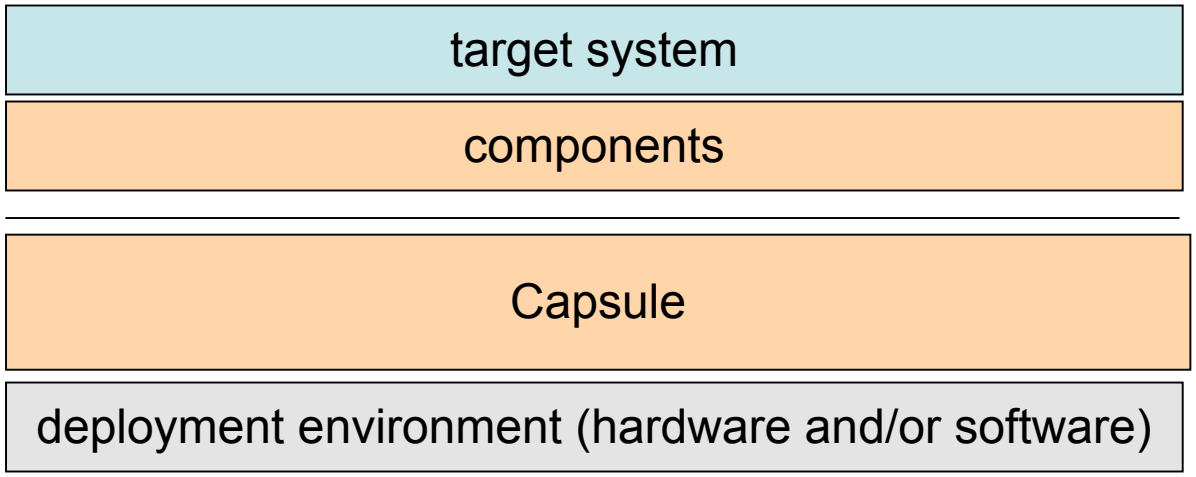
Component Metamodel



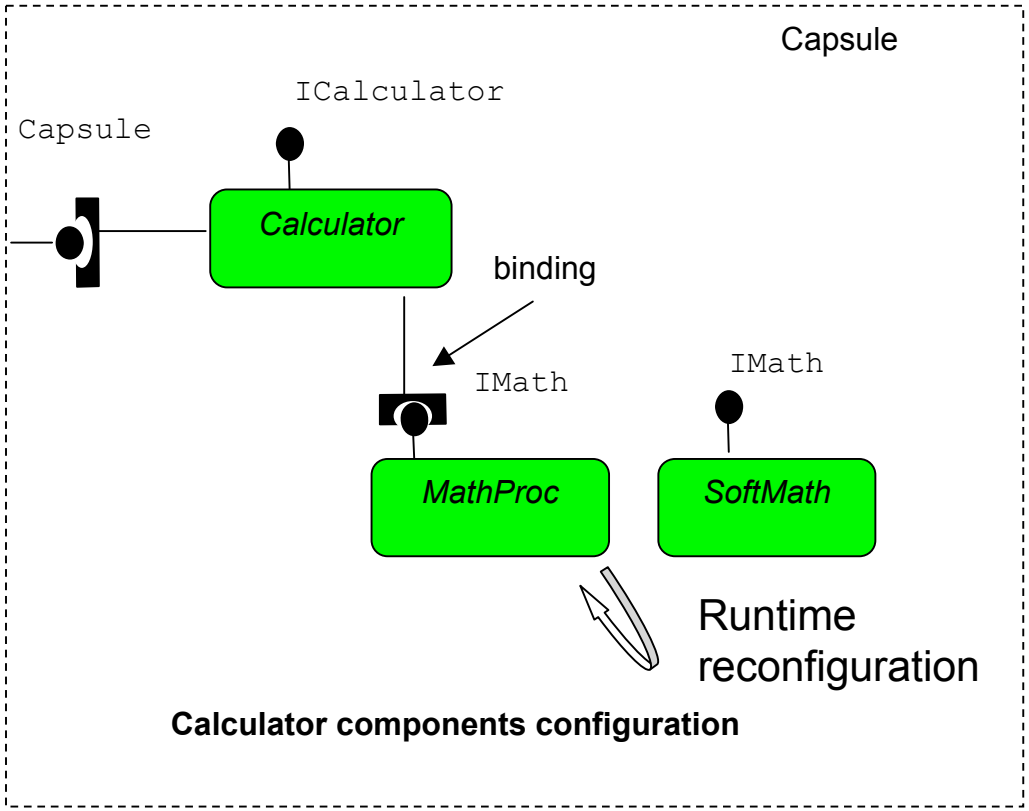


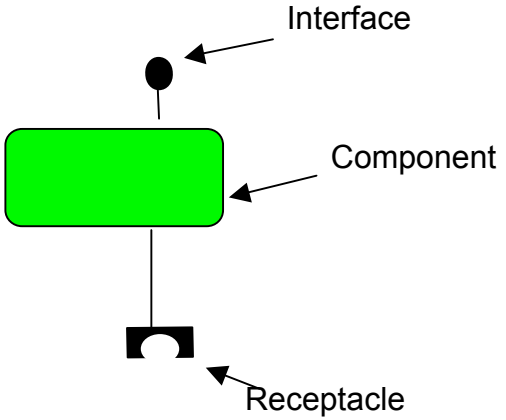
Main API (Pseudo OMG IDL)

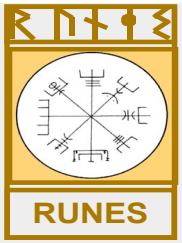
```
comp_type load(pattern name)
status    unload(comp_type t);
comp_inst instantiate(comp_type t);
status    destroy(comp_inst comp);
connector bind(int_inst i, recep_inst r,
                  connectorfactory b);
status    putattr(ID entity, ID key, any value);
any       getattr(ID entity, ID key);
```



Example

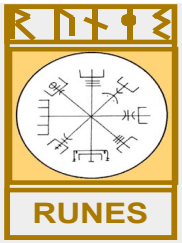

Capsule
ICalculator
Capsule
Calculator
binding
IMath
IMath
MathProc
SoftMath
Runtime
reconfiguration
Calculator components configuration


Interface
Component
Receptacle



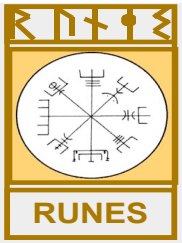
Component Frameworks

- re-usable, dynamically-deployable, software architectures
  - give structure, tailorability and constraint
  - built as compositions of components and/or other CFs
- provide "life support environments" for plug-in components in a particular area of concern
  - example: a protocol stacking CF that takes plug-in protocols
- embody constraints on pluggability
  - example: disallow stacking of IP plug-in above TCP plug-in
  - constraint specification may be ad-hoc
  - or may employ generic constraint languages such as OCL (with automatically generated run-time machinery)
  - or logic based languages (Prolog?)



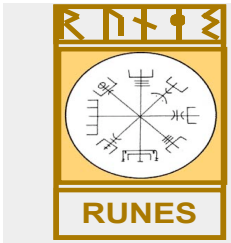
Component Frameworks (2)

- Current thinking is that CFs are "design patterns"
  - i.e. we can't abstract common requirements
- I disagree
  - plug(), unplug()
  - Constraints expressed/validated with OCL or Prolog

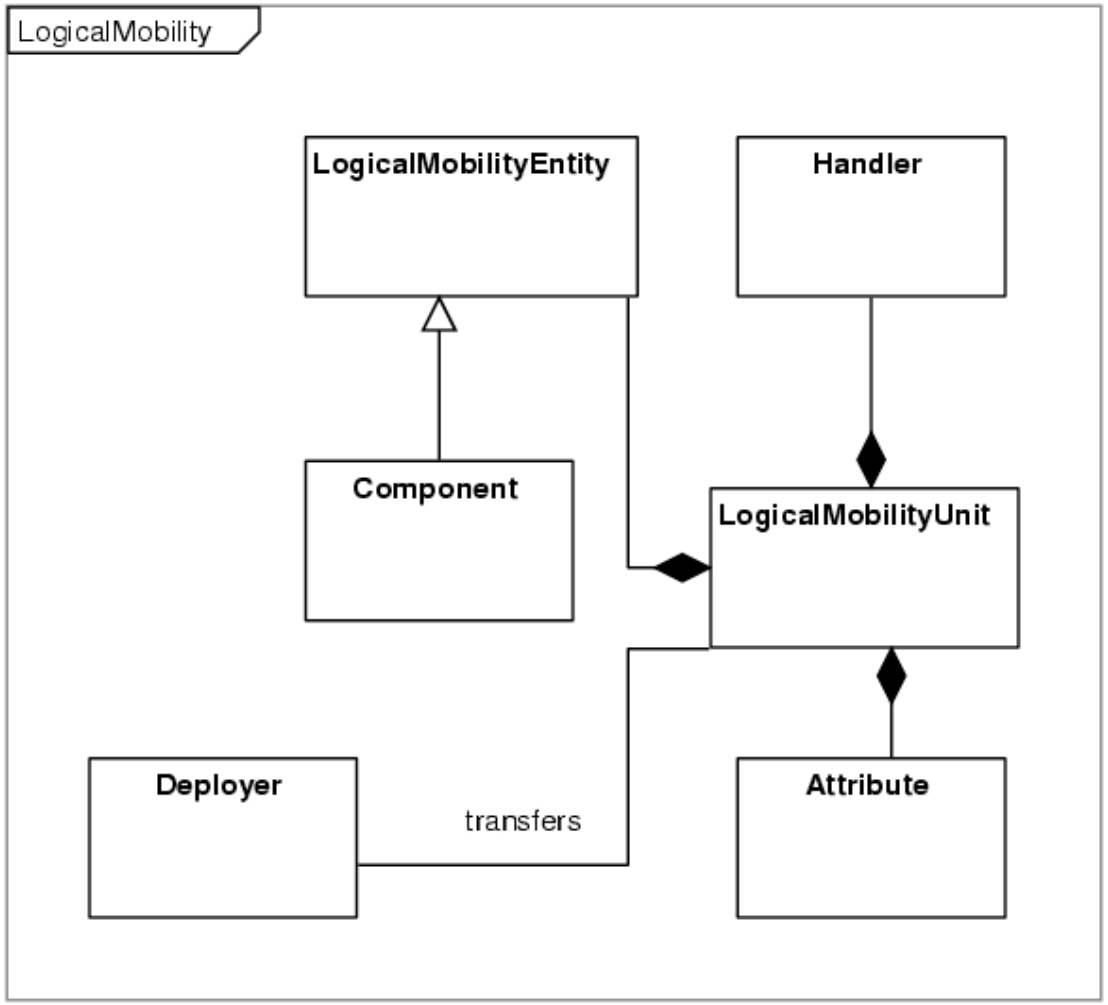


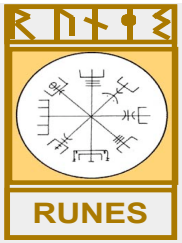
Some example CFs

- network services
- distributed reconfiguration service
- location/context services
- advertising and discovery services
- protocol stacks

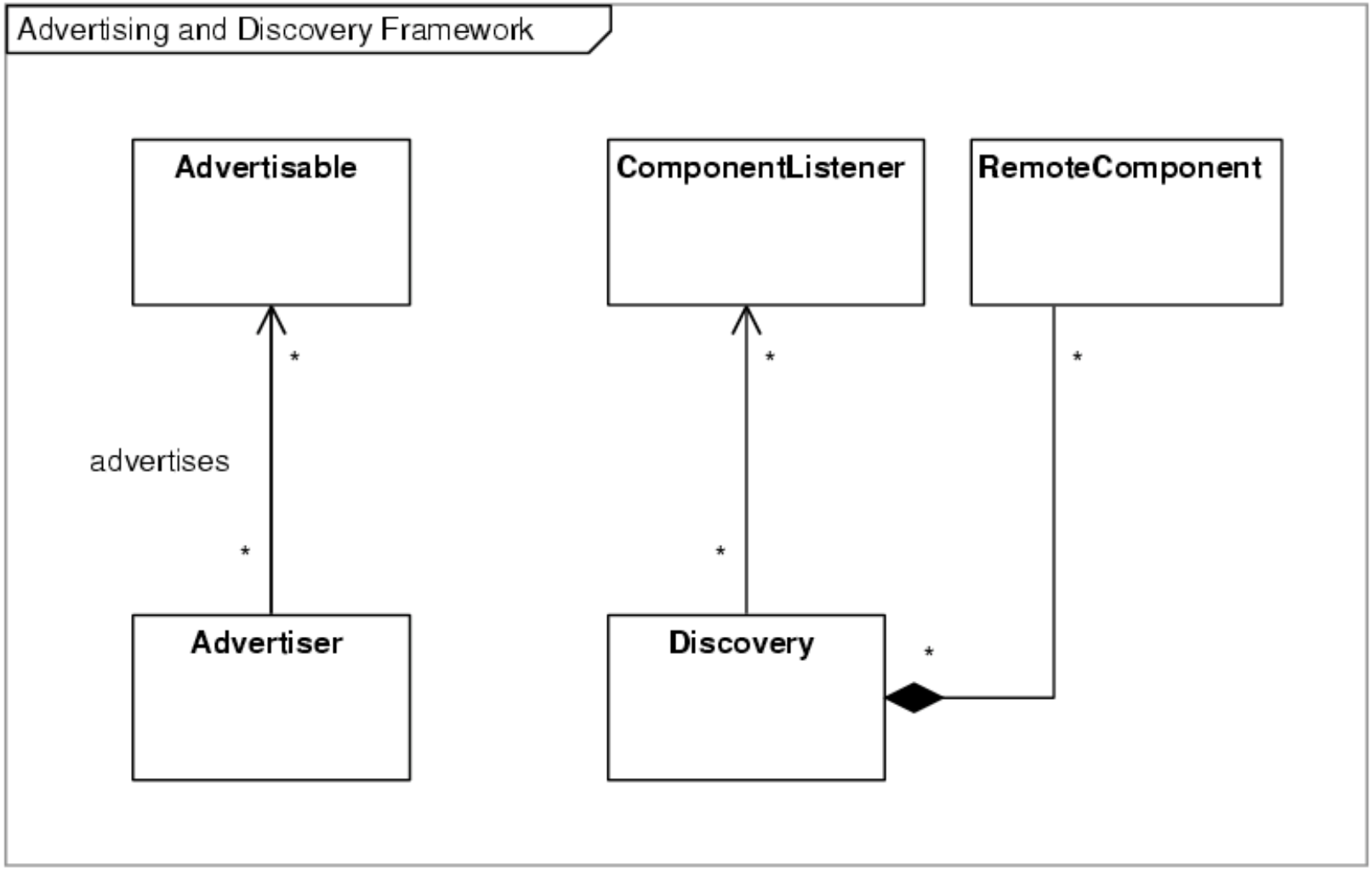


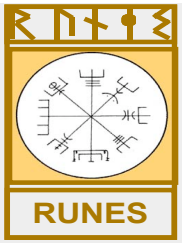
Reconfiguration





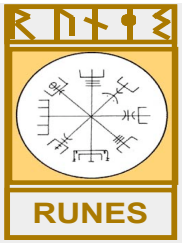
Advertising and Discovery





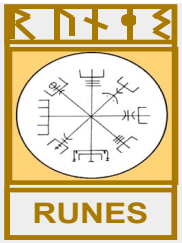
Where We Are

- Java Implementation
  - J2ME
  - Released and operational
  - CFs Already Exist
- C/Linux Implementation
  - Released and operational
  - CFs “being ported”
- C/Win32 Implementation
  - Being worked on



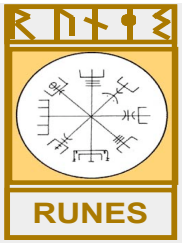
Where We Are (2)

- C/Contiki implementation
  - "In progress"
  - Design Decisions
    - No component instances
    - No customisable connectors
    - Components are pre-`load()`ed



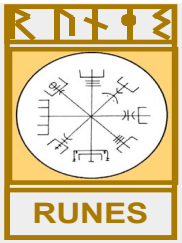
Related Work

- CORBA CM
- OSGi
- Gravity
- one.world
- THINK
- DPRS
- …



Conclusions

- RUNES Middleware is a component based middleware
  - Aiming at heterogeneous environments
  - Allowing for static and dynamic reconfiguration
  - Cross-layered approach



Any Questions?

- *A Reconfigurable Component-based Middleware for Networked Embedded Systems*. Paolo Costa, Geoff Coulson, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco and Stefanos Zachariadis Submitted for Journal Publication. December 2005.

- *The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems*. Paolo Costa, Geoff Coulson, Cecilia Mascolo, Gian Pietro Picco and Stefanos Zachariadis In 16th IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC05). IEEE Press. Berlin, Germany. September 2005.