# Role Based Type-Checking

Robert Bram

Service discovery mechanisms through Role Based type checking, implemented with a proxy pattern and reflection.

# Discovery & Invocation Mechanisms

- Service discovery can be implemented via:
  - Naming mechanisms
    - CORBA, RMI, Web Services
  - Typing mechanisms
    - Jini
- Service invocation can be implemented via:
  - Naming mechanisms
    - Web Services
  - Typing mechanisms
    - Jini, RMI, CORBA
- Type safety VS catching naming errors

# Type Based Discovery - Knowledge Requirement

- High: component match
  - Service must match concrete type
- Medium: type match
  - Service must implement abstract type (interface)
- Low: role match
  - Service must implement set of methods

# Role Type

- A role is an abstract set of type members
  - Methods, types, properties
- Concrete type supports role type if
  - Members of role type form SUBSET of members of concrete type

# Role Type VS Natural Type

- Role types
  - Imply a specific relationship between objects of the role type and their context. Eg, actor (*role*) in a play (*context*).
  - Require object to have identity apart from role type. Eg, actor still a person (with identity) even when not acting.
- Natural types:
  - Do not imply a specific relationship with other types (except for whole-part where applicable).
  - Grant an object its identity. It cannot leave the type without losing its identity.

# Natural Type Supports Role Type

- Roles and classes both exist within hierarchies
- Roles and classes are interconnected by the supports relationship, specifying which classes support which roles
- Inheritance of implementation only occurs within class hierarchies.
- A role does not specify exactly how behaviour is to be achieved – this is left up to the classes that support it.

# Implementation of Role Types

- A role is an interface that does not have to be compiled into an object's type hierarchy.
- Implementation via proxy pattern and reflection.
- In Java, requires new implementation of *instanceof* and *casting*. Ideally we would like:
  - `if (concreteObject instanceof RoleType)`
    `RoleType object = (RoleType) concreteObject;`
- Cast returns proxy that maps call to member on `RoleType` to matching member on `concreteObject`.

# Role Based Jini Lookup

- `ServiceItem` includes `Object service`
  - Must implement all types in `ServiceTemplate`'s `Class [] serviceTypes`
  - Medium level knowledge requirement
- Instead, require `service` to support all Role types in `serviceTypes`
  - Low level knowledge requirement
  - Type safety still checked on members

# Relation to Ontologies

- An ontology is a description of a “world”
- Java is an ontology language (++)
- OWL for semantic web
- OWL-S for semantic web services
- Problem addressed here is *mapping between two ontologies* – how methods in one ontology are mapped into another