

## Organizational abstractions for adaptive systems

Alan Colman

Jun Han



### Motivation



- Software systems are becoming more open, distributed and pervasive
  - Software needs to operate in increasingly more dynamic operating and interactional environments
  - Rapid change in requirements, software goals shift
  - Humans in and above the loop
- Require more adaptable and adaptive software
- *How do we find architectural abstractions that will make software more amenable to adaptation?*

## Overview



- Ontogenic adaptation and organization
- Role-oriented adaptive design (ROAD)
- Management contracts
  - Built from Control & Communication Acts
- Implementation of contracts using Association Aspects

Conceptual underpinnings...

## Ontogenic adaptation (Maturana)



- Ability of a system to alter its parts and structure while maintaining its organizational integrity in the environment
- Example – the nervous system in animals
  - Nervous system provides structured pathways between cells
  - Cells die and are replaced
  - Pathways have the ability to form and restructure in response to environmental coupling and internal dynamics – e.g. the brain
- Example – bureaucratic organizations
  - Organizations have management layers - chains of command and reporting
  - Employees in organizations perform roles. Employees can change roles or be replaced
  - Organizations can be restructured to meet changing goals and environments

**Ontogenic adaptation = component interchange + structural plasticity + organizational regulation**

## Ontogenic adaptation and OO



- Ontogenic adaptation requires
  - Interchangeability of components
  - Plastic structure
  - Self-regulation
- Object-oriented methodologies provide interchange through ...
  - Separating interface from implementation – well-defined interfaces allow implementations to be exchanged
  - Polymorphic objects in an inheritance hierarchy – late binding
- Design patterns create flexible design *fragments* through indirection
- But have shortcomings of traditional static approaches...
  - Associations between objects are implicit in method invocations – scattered through code.
  - Object implementations are within a stiff structure of pre-existing associations
  - Structure of program frozen at compilation
  - Roles only a design concept
- *How to achieve structural plasticity and self-regulation in an object-oriented framework?*

## ROAD - Role-Oriented Adaptive Design

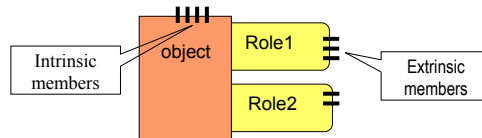


- Organizations are self-regulating configurations of roles
- Objects can adopt, drop and change roles within their organization
- Roles are dynamically associated with each other using contracts
- 3 types of role

# Roles and objects



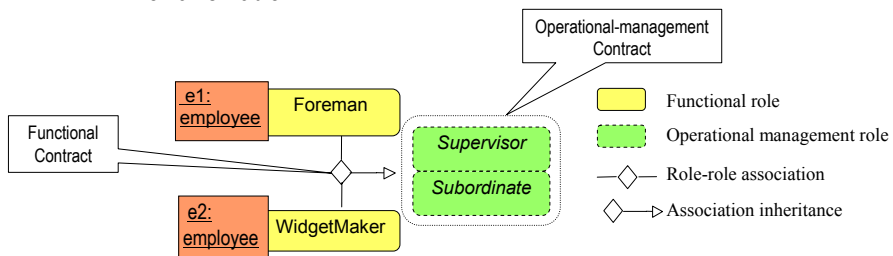
- A role is set of functions that is responsible to the system as a whole.
- In ROAD, roles are first-class design and *implementation* entities (Kristensen)
- Role-player objects provide situated computational and communication capabilities.



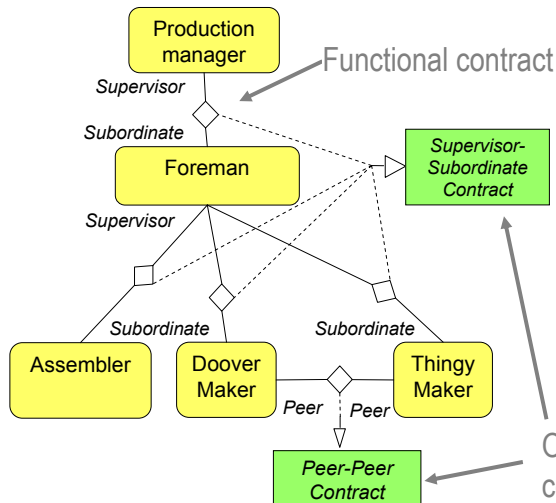
# 3 types of role in ROAD



1. *Functional* roles – achieve the first-order goals of the organization
2. *Operational-management* roles – are a control abstraction of functional roles – regulate interaction between functional roles
3. *Organizational-management* roles – have the ability to change the structure of the organization by changing contracts – they make sure the organization remains viable.



# An Organization built from Functional and Operational-management Roles and Contracts



- Operational-management contracts define protocols that regulate the interactions between functional roles
- Functional role contracts can be seen as specialisations of operational-management role contracts – “Association inheritance”
- Protocols need to be developed for different types of operational association - peer to peer, supervisor-subordinate etc.

Operational -management contracts

# Operational-management contracts

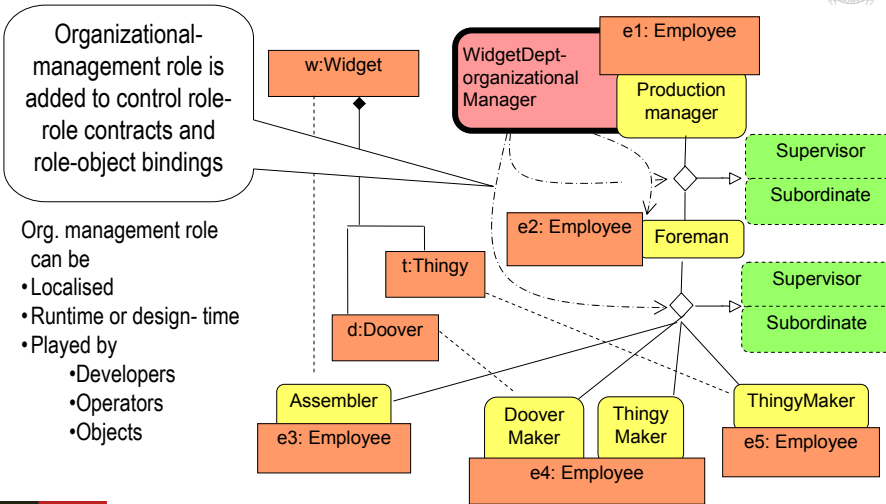


- Contracts specify the respective obligations of *particular* parties – more than DBC
- Management contracts built from Control-Communication-Act primitives and protocols
- Specify what management roles can say to each other

Type of communication	Communicative control acts
Direct Control	DO, SET_GOAL
Indirect Control	RESOURCE_ALLOCATE(r), RESOURCE_REQUEST(r)
Information	ACCEPT, REJECT, INFORM, QUERY, ACKNOWLEDGE

- In aspect-oriented implementation CCAs pattern-matched to method signatures
- Contracts monitor communication. Enable clause-violation and breach-of-contract to be detected

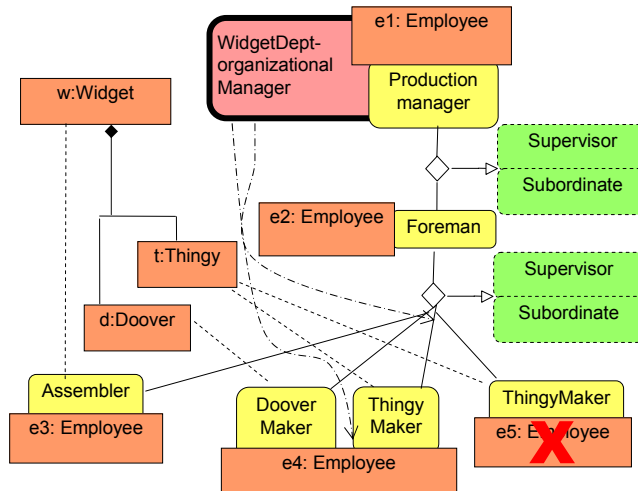
## 3<sup>rd</sup> type of role- Organizational management



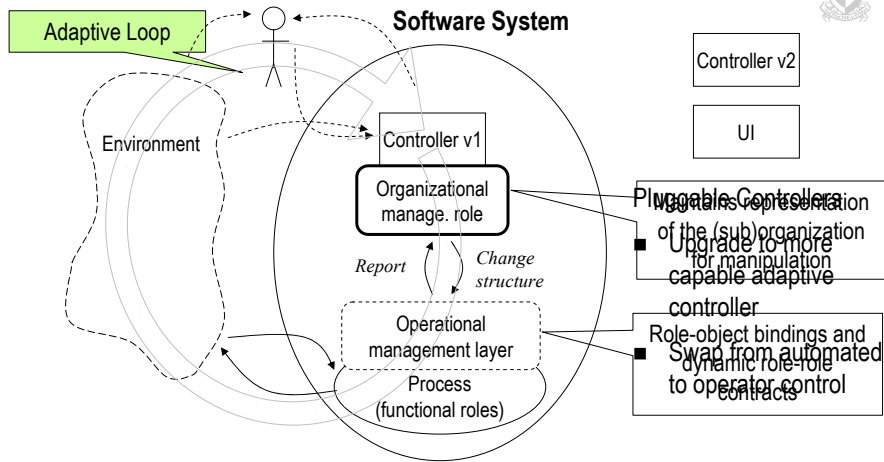
## Adaptive behaviour



1. e5 becomes unavailable
2. Contract revoked  
Associations removed  
Binding broken
3. New contract created.  
ThingyMaker role transfers from e3 to e4  
Association created  
ThingyMaker role bound to e4



## Summary - ROAD schema – taking the environment into account



## Recent Work



- Implementation of Contracts using Association Aspects
  - Management layer **independent** from functional classes. Contracts and organizational management implemented as a 'separate concern' – similar to a nervous/management system
  - Implemented using "Association Aspects" extension to the AspectJ compiler (Sakurai)
  - Permits *perobjects* aspect instances to be created. These are associations between groups of objects that maintain state
  - Contract enforcement using *before* and *after* advice
  - CCA pointcuts pattern matched to method signatures
- Structure of the management/nervous system
  - Recursive structure of self-managed composites
  - NFRs flow down, contract performance flows up

## CCA pointcuts and clauses



- Party pointcuts that match the parties to the contract and the direction of communication.

```
pointcut aToB(MRole a, Mrole b): associated(a,b) && this(a) && target(b);
```

- CCA pointcuts that define the types of message.

```
pointcut resAlloc( ) : call(* ra_*(Resource));
```

- Instances of these two above basic types of pointcut are then composed into pointcuts that represent particular *clauses* in the management contract. These composite pointcuts define *who* can say *what*.

```
pointcut a1(Supervisor sup, Subordinate sub): aToB(sup, sub) && resAlloc();
```

- Code at [www.it.swin.edu.au/personal/acolman](http://www.it.swin.edu.au/personal/acolman)

## Further work and open questions...



- How to represent and reason about organizational structure
- Rules for organizational integrity
- Role object bindings
- Definition of protocols for other types of operational management contract
- Formalize CCAs?
- Indirect control / resource ownership
- QoS adaptability - is *functional* adaptation possible?
- Implementation using load-time weaving?
- ...

# Discussion / Questions?

---

