

---

# Dynamic Components

## Defining Locality by Aggregation

Axel Wienberg  
Technical University Hamburg-Harburg, Germany

## Introduction and Overview

- ❑ context of the talk:
  - object-oriented analysis, design and implementation, mobile agents
- ❑ concrete, basis for actual system implementation
- ❑ may in turn enhance implementation platform for agent-oriented software
- ❑ general idea:
  - carry the analysis level concept of aggregation to design and implementation
- ❑ analysis, design: visual notation
- ❑ implementation:
  - exploit aggregation information for locality and mobility (migration)

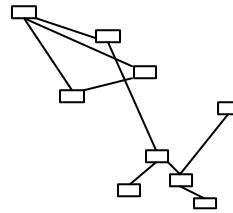
## Everything is an Object

There are many concepts for structuring an object-oriented model or program:

- ❑ packages (software components)
- ❑ classification, generalization (classes & inheritance)
- ❑ procedural abstraction (methods), nested state machines

No means for structuring a concrete system state

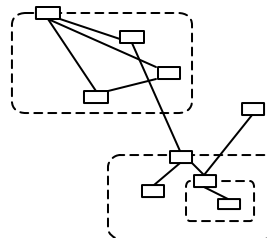
- ❑ "EVERYTHING IS AN OBJECT"
- ❑ objects and links: an arbitrary net
- ❑ advantage: very flexible
- ❑ disadvantage: not very expressive (few model-inherent constraints)



## Structuring the System State

Structuring the system state is important for large, persistent, reflective, distributed systems

- ❑ Which objects belong together (form a unit/aggregate)
  - for migration (locality)
  - for persistence
  - for browsing (user semantics)
  - for expected access (clustering)
  - for integrity checking
  - for access restriction (security, locking)
  - for accounting
- ❑ Distinct assignment of objects to nested groups



## Component Aggregation - Requirements

Requirements for component aggregation:

- Dynamic group membership (for migration)
- An object must not be in two places at the same time
- Objects can be nested inside further objects
- no further restrictions (flexibility of generic association, e.g. polymorphism)

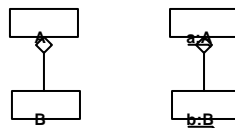
=> dynamic hierarchical structure

Can be exploited for various purposes!

## Aggregation in UML

aggregation

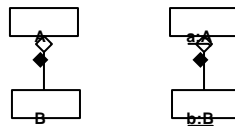
- weakly defined (entirely conceptual)
  - “part-whole relationship”
  - “organizational superiority”



- no cycles
- unrestricted multiplicity for whole, sharing possible, independent lifetimes

composition

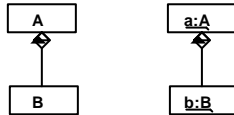
- no cycles
- no sharing
- multiplicity 1 for the whole
- cascading create and delete
- immutable relationship



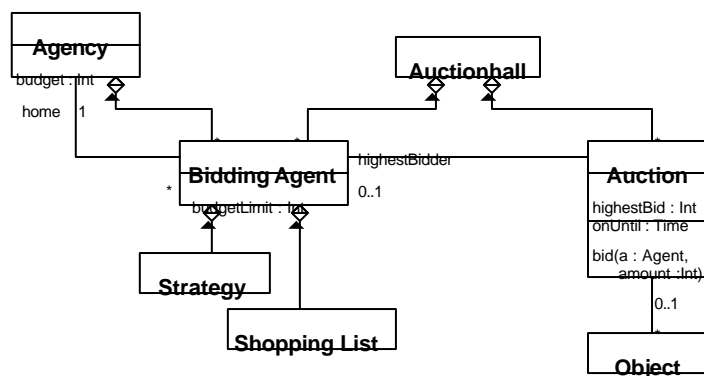
## Component Aggregation - Definition

Component aggregation:

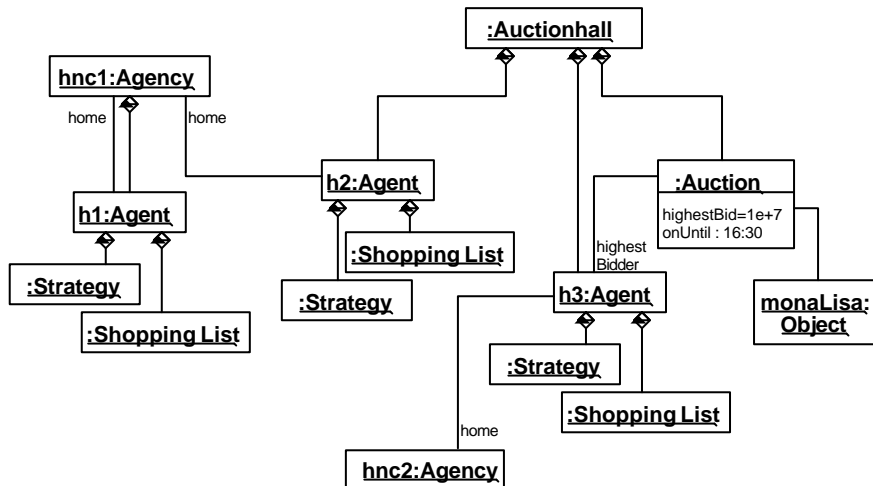
- ❑ a kind of aggregation (UML: a stereotype)
- ❑ instance: *component link* between two objects, directed from *supercomponent* to *subcomponent*.
- ❑ characteristic requirement: no sharing
  - i.e.: each object is immediate subcomponent of at most one object
  - The graph of objects and component links forms a *forest*
- ❑ component links may change over time; a component link implies no dependency of lifetimes; an aggregation does not restrict visibility
- ❑ notation:



## Component Aggregation - Class Diagram



## Component Aggregation - Object Diagram



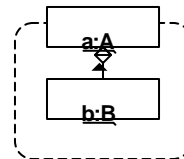
20-Sep-99-WienbergEUCANDynCo

9

## Component Boundaries

A *dynamic component* is the set of objects transitively aggregated by the dynamic component's *primary object*.

notational extension to UML: *component boundary*.



The functionality of a component

- is implemented by several objects making up the component
- expresses itself through interactions with the outside (messages)

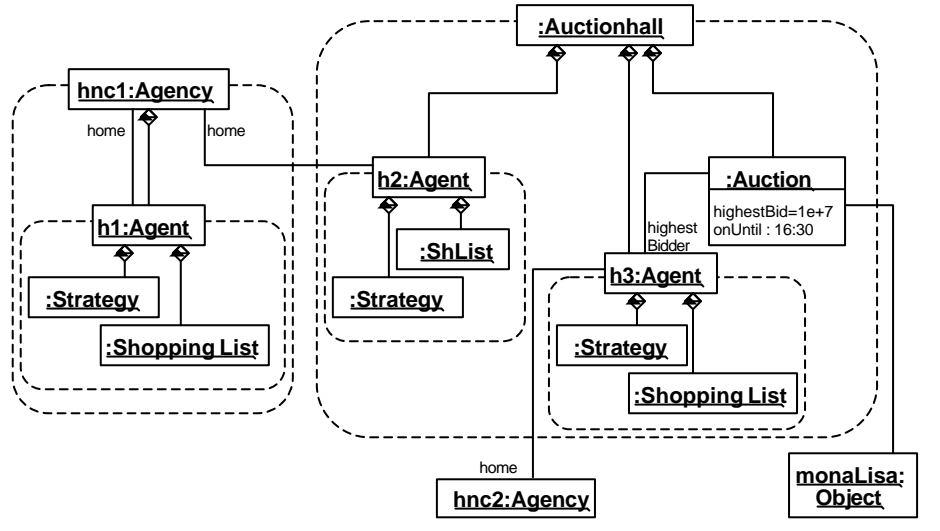
A component

- is an object (something that can be named)
- is a set of objects (the objects making up the component)

20-Sep-99-WienbergEUCANDynCo

10

## Component Boundaries - Object Diagram

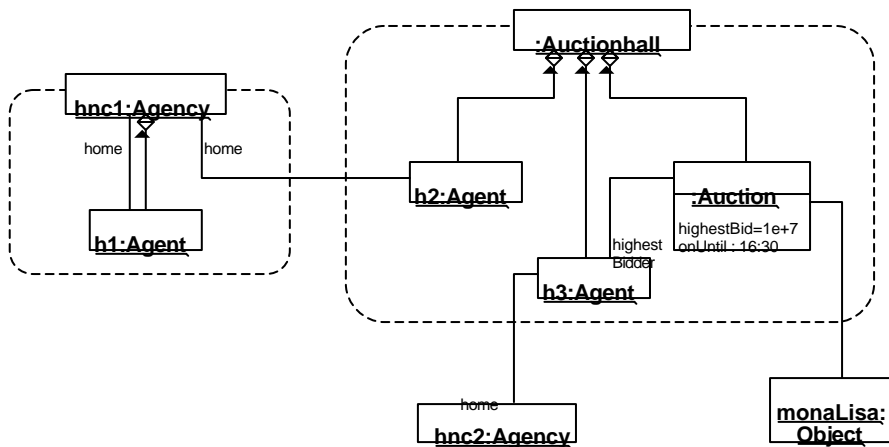


20-Sep-99-WienbergEUCANDynCo

11

## Coarsening

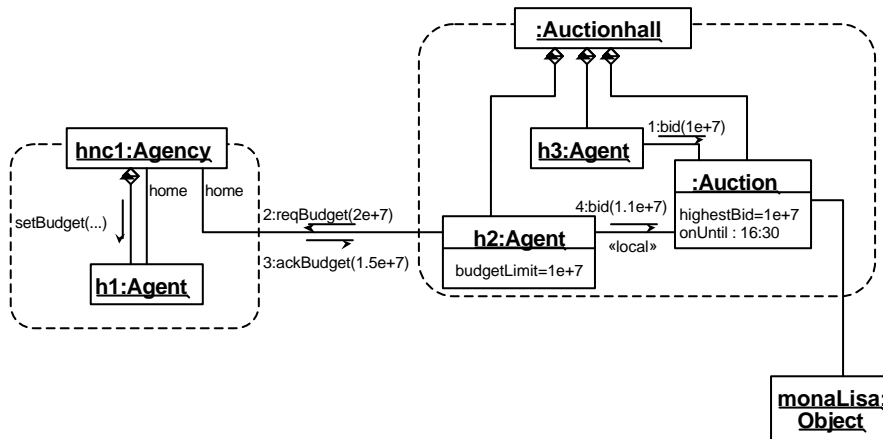
The system state can be described at different levels of detail by abstracting from the internals of a dynamic component



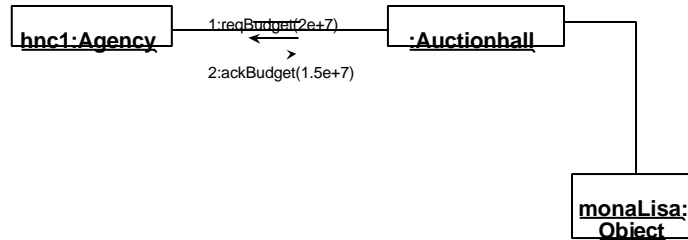
20-Sep-99-WienbergEUCANDynCo

12

## Communicating Dynamic Components



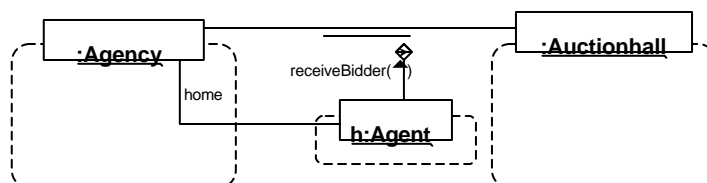
## Coarsening Communication



## Migrating Dynamic Components



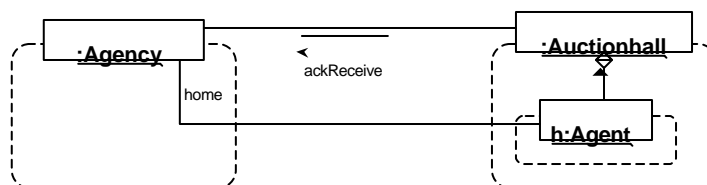
## Migrating Dynamic Components



## Migrating Dynamic Components



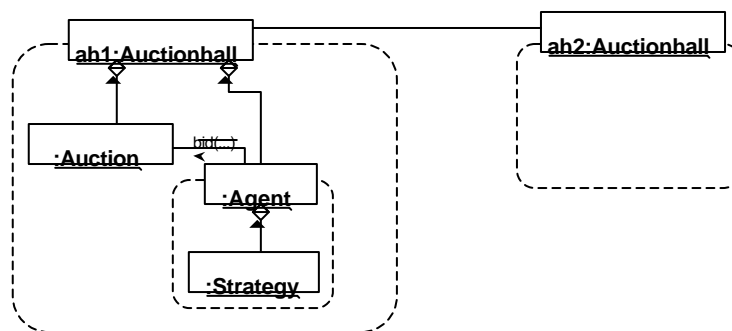
## Migrating Dynamic Components



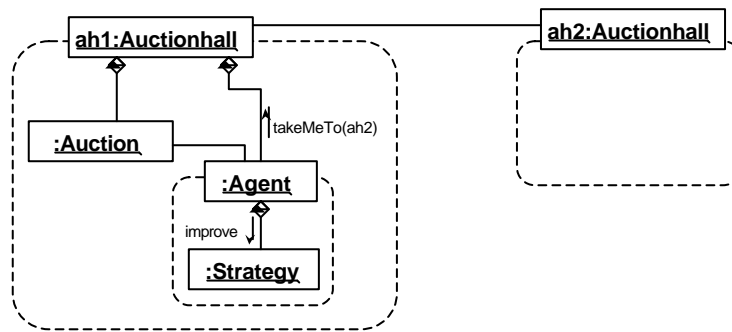
## Migrating Dynamic Components



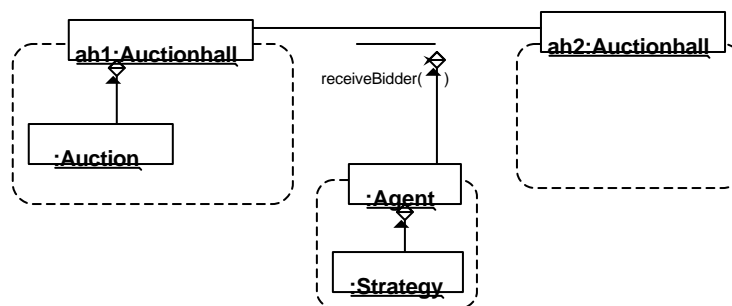
## Migrating Active Dynamic Components



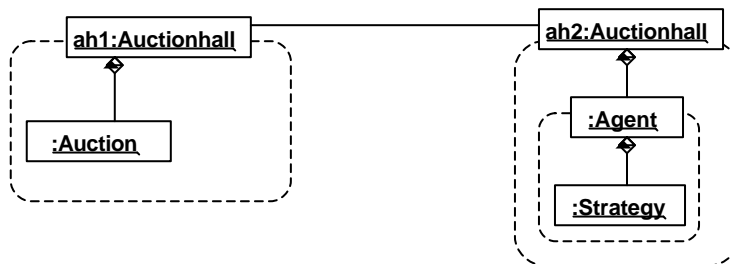
## Migrating Active Dynamic Components



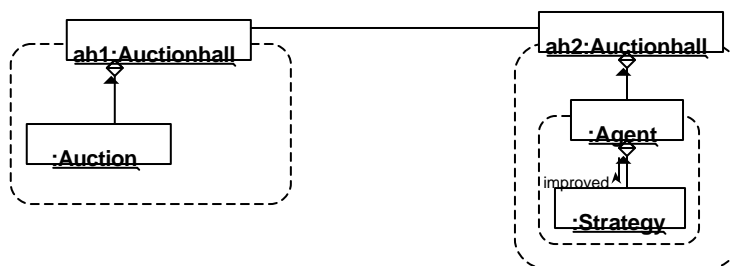
## Migrating Active Dynamic Components



## Migrating Active Dynamic Components



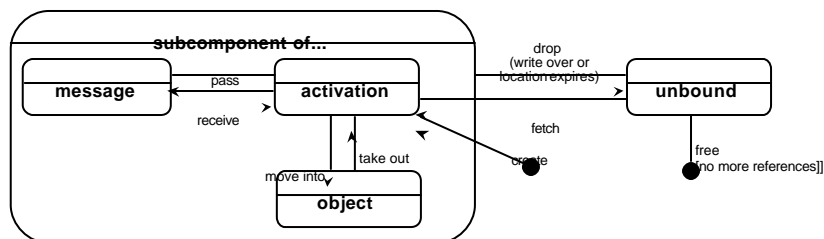
## Migrating Active Dynamic Components



## Migrating Activations

- ❑ messages originate at the communicating object
- ❑ messages are sent by activations
  - => activations are local to the object
  - => activations migrate along with the object
  
- ❑ migrating active objects is important (e.g. distributed workflow)
  - problems with current mobile object technology (e.g. *Voyager*)
  
- ❑ migrating threads <-> migrating activations

## Component Aggregation State



## Implementation in a Programming Language

---

**Distinguishes** between component and reference variables

Keeps component aggregation **polymorphic** and **mutable**

Provides operations on component links

- create **reference** to subcomponent
- destructive read: **take** component link **out** of component variable
- move** component link **into** a component variable

Operations maintain the properties of component aggregation

- no sharing** (checked statically)
- no cycles** (optionally checked at runtime)

Is upwards compatible: Language **extension**

Offers **reflective access** to the component structure

Allows **efficient** execution and inspection

---

20-Sep-99-WienbergEUCANDynCo

27

## Code example

---

```
class Auctionhall {
void takeMeTo(Agent aref, Auctionhall dest) {
    Agent a@;           // declare component variable a
    a = agents.takeOut@(aref); // call method returning component link
                        // and move the link into a
    log("Agent leaving: "+a.getName());// reference a and send message
    ah2.receiveBidder(a@); // take out component link from a
                        // and pass it to a message
}

void receiveBidder(Agent a@) {
    log("Agent arrivng: "+a.name);
    agents.add(a@)
}

ComponentVector<Agent> agents;

...
}
```

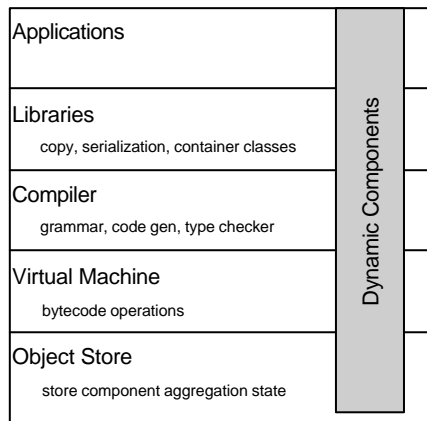
---

20-Sep-99-WienbergEUCANDynCo

28

## Implementation in a Programming Env.

---



## Summary

---

Dynamic components are

- nested
- communicating
- migrating
- active

groups of objects

- visualized using component boundaries (a UML extension)
- defined by the component links in the current system state
- specified using component aggregation (during design)
- expressible in a programming language (during implementation)
- able to represent most kinds of aggregation (during analysis)
- observable at runtime, structuring the system state