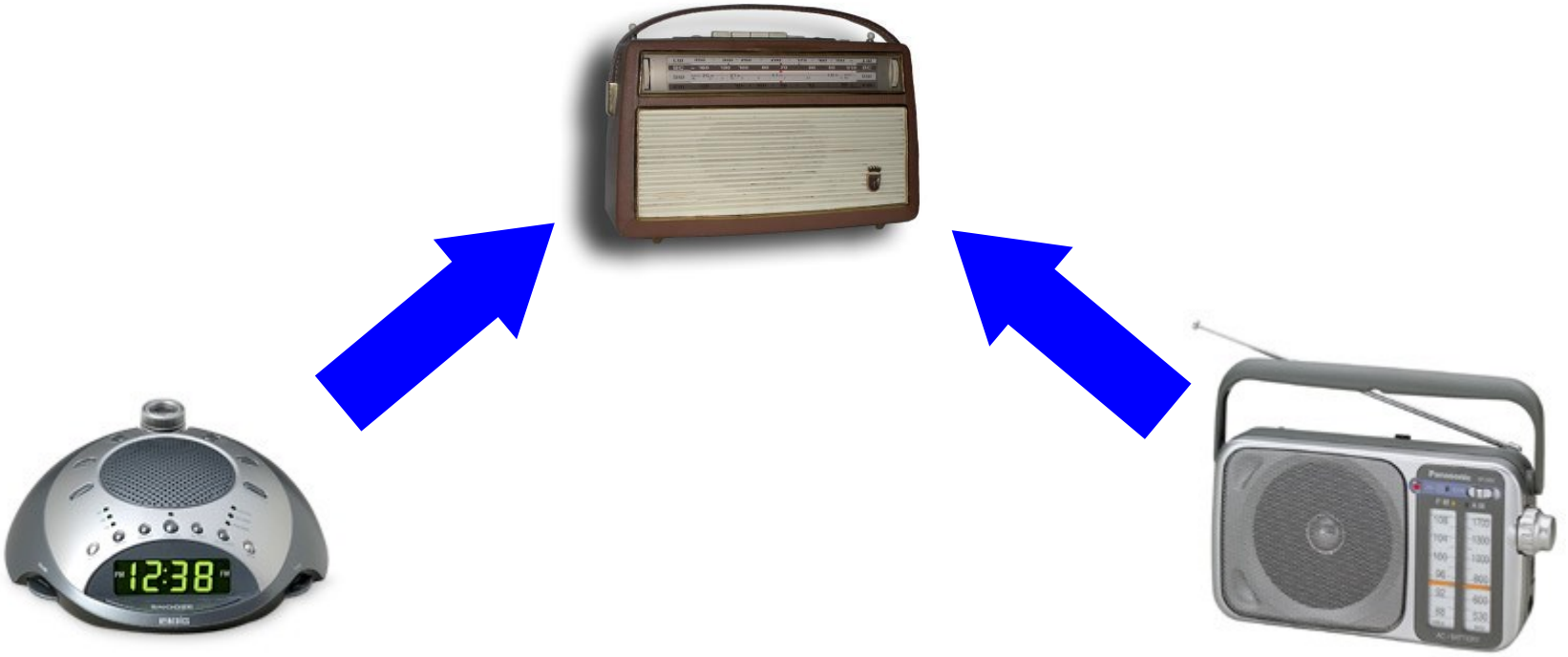
Using Resemblance to Support Component Reuse and Evolution



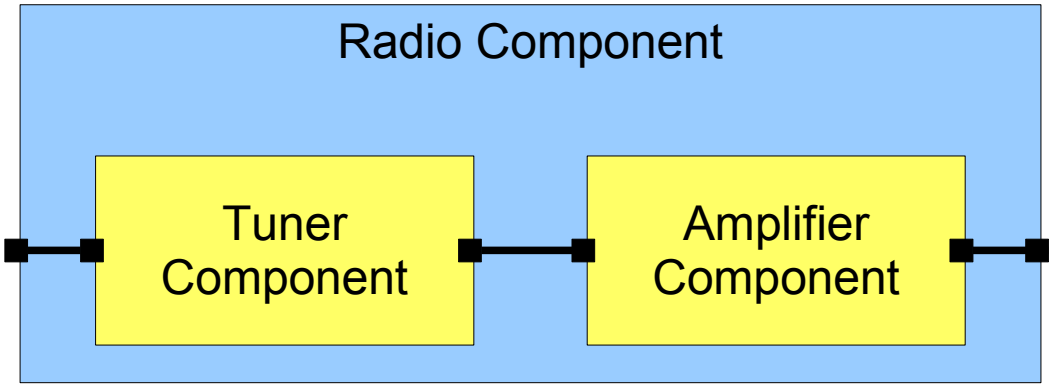
Andrew McVeigh, Jeff Kramer and Jeff Magee
Department of Computing
Imperial College, London

SAVCBS 2006

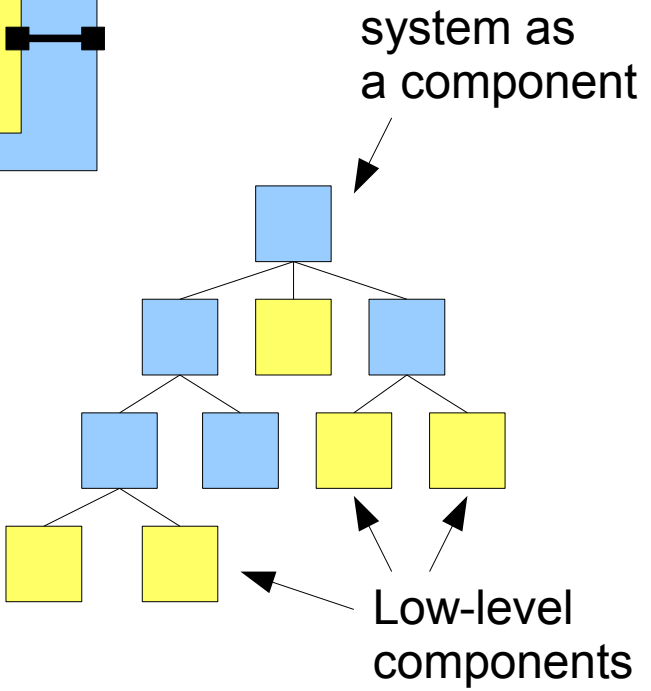
Introduction

The Vision of Software Components

- Composite components are constructed by composing existing components and connecting them together
  E.g. A radio...

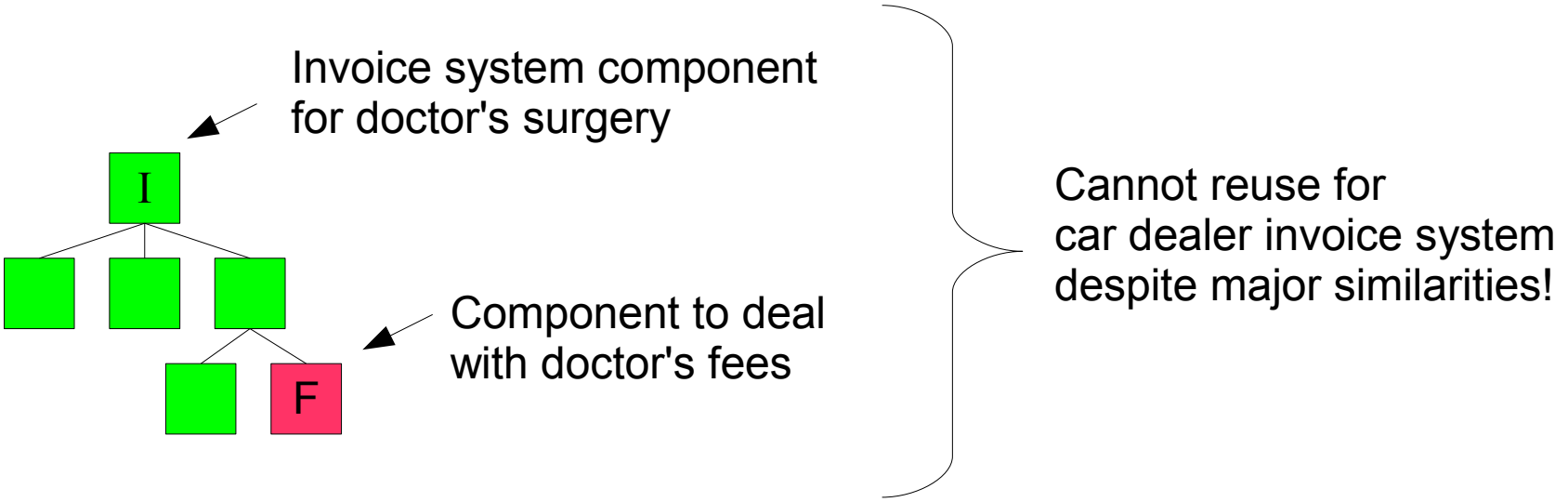


- This is a scalable concept...
  - Entire systems can be represented hierarchically in this way



But, Higher Abstraction = Less Reuse

- System construction should ideally be a case of connecting together increasingly higher-level components...
- BUT the higher the level of abstraction of a component
  - the more specific it generally is (buried abstractions)
  - the less reusable it becomes...



4 Requirements for a Reuse Solution

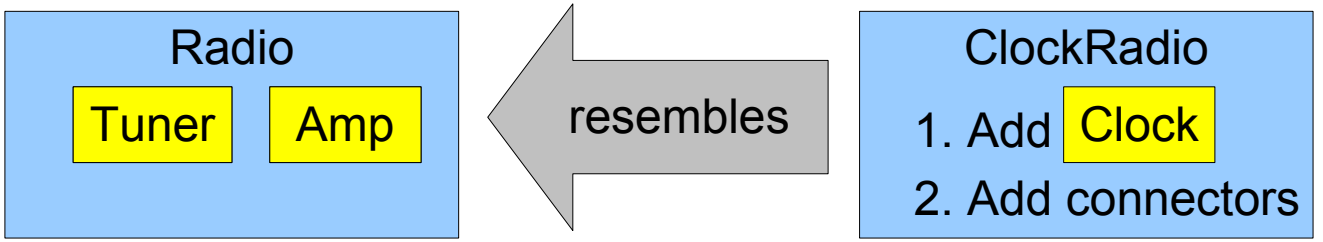
- Reuse implies (extensive) alterations — ***1. Alter***
- Can we just change existing component?
  - No! We can't break if it for existing users — ***2. NoImpact***
- **Can we copy and modify the source?**
  - No! Must be able to accept upgrades — ***3. Upgrades***
  - Copying leads to maintenance problems
  - We may not have the full source code — ***4. NoSource***

To address these we introduce two constructs:

Resemblance and Redefinition

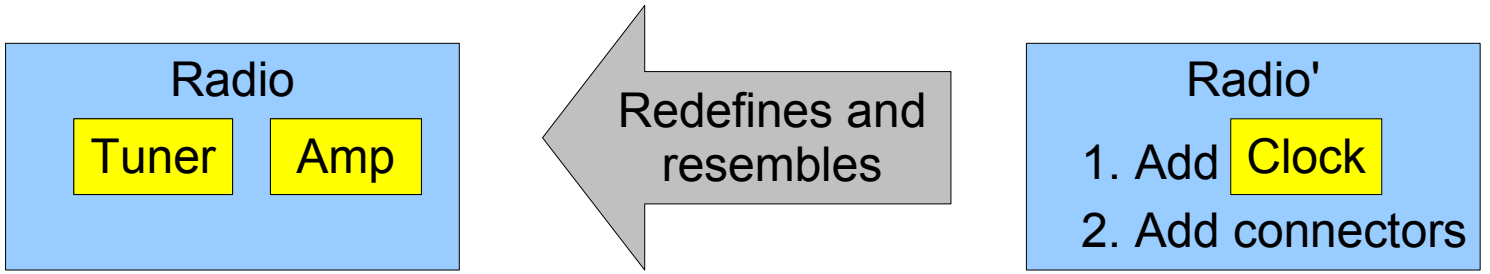
Resemblance: Enabling Reuse

- Defines a component in terms of similarity to another
  - An inheritance-like construct for components
  - The new component is specified as add / delete / replace changes to the architecture of a base component
- **We keep the changes as elements in the new component**
  - Lets us reason about combining changes, upgrades etc.
- Intuitively: ClockRadio resembles Radio, but adds a Clock



Redefinition: Modelling Evolution

- Used to model evolution of a component
- Replaces the existing definition of a component
  - The existing definition and the redefinition are kept separate
  - Changes will only be applied if redefinition is “loaded”
  - Can be combined with resemblance to evolve a component in terms of changes to the old definition
- Intuitively: Evolving a Radio to add a Clock



Using the Constructs

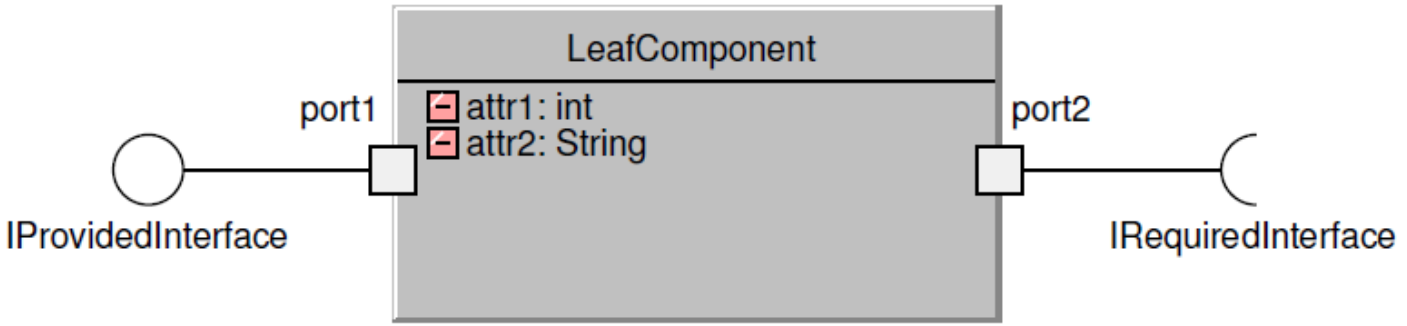
These can be used independently, or together:

- Resemblance
  - defines one component in terms of changes to another
- Redefinition
  - changes the definition of an existing component
- Resemblance + redefinition
  - allows evolution of an existing component in terms of changes to the previous definition

The Notation

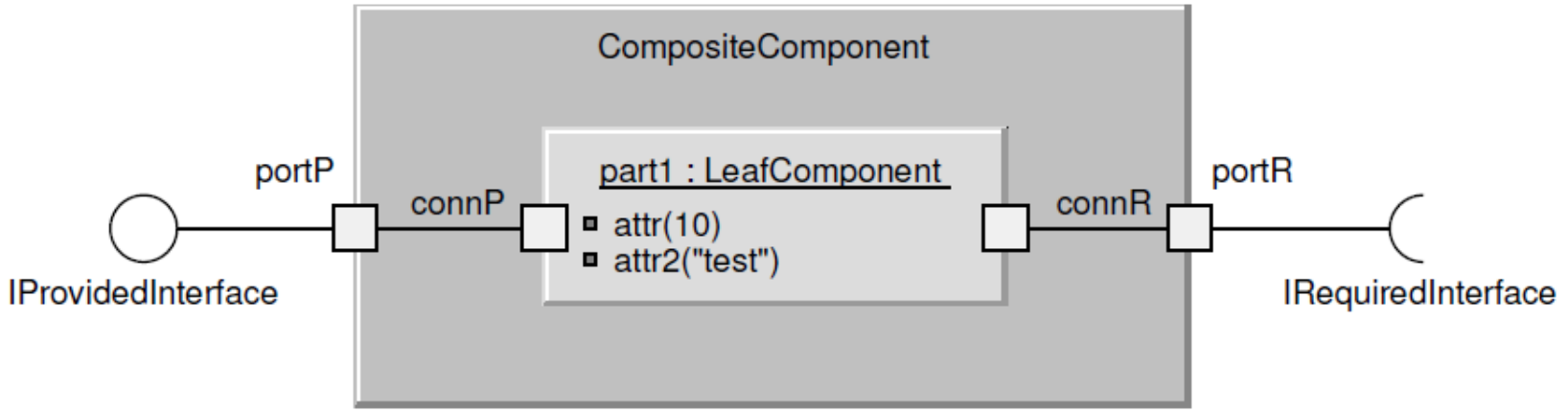
Notation for Leaf Components

- The graphical form is UML2 composite structure diagrams.
- The textual form is remarkably similar to Darwin.



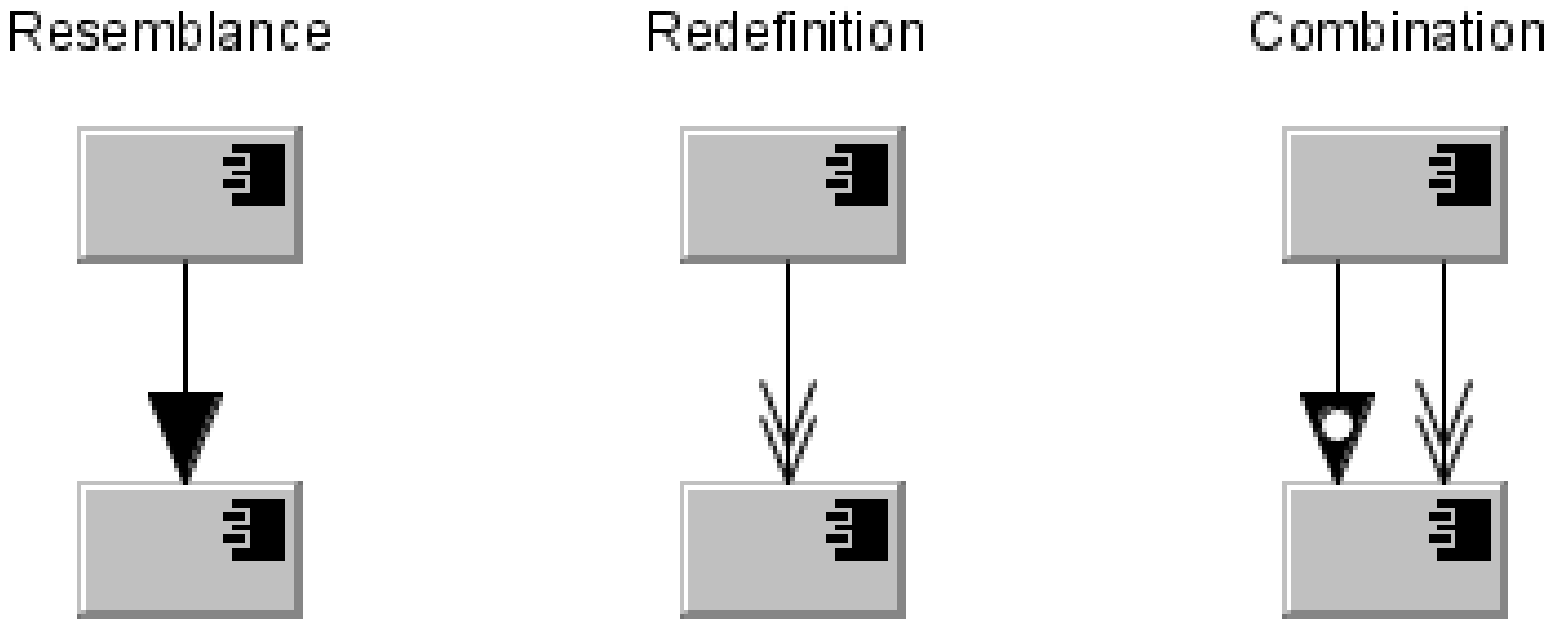

```
component LeafComponent
    describes com.example.JavaLeafComponent
{
    attributes:
        int attr1; String attr2;
    ports:
        port1 provides IProvidedInterface;
        port2 requires IRequiredInterface;
}
```

Notation for Composite Components




```
component CompositeComponent
{
   ports:
      portP provides IProvidedInterface;
      portR requires IRequiredInterface;
   parts:
      LeafComponent part1
         set attr1(10), attr2("test");
   connectors:
      connP joins portP to port1@part1;
      connR joins portR to port2@part1;
}
```

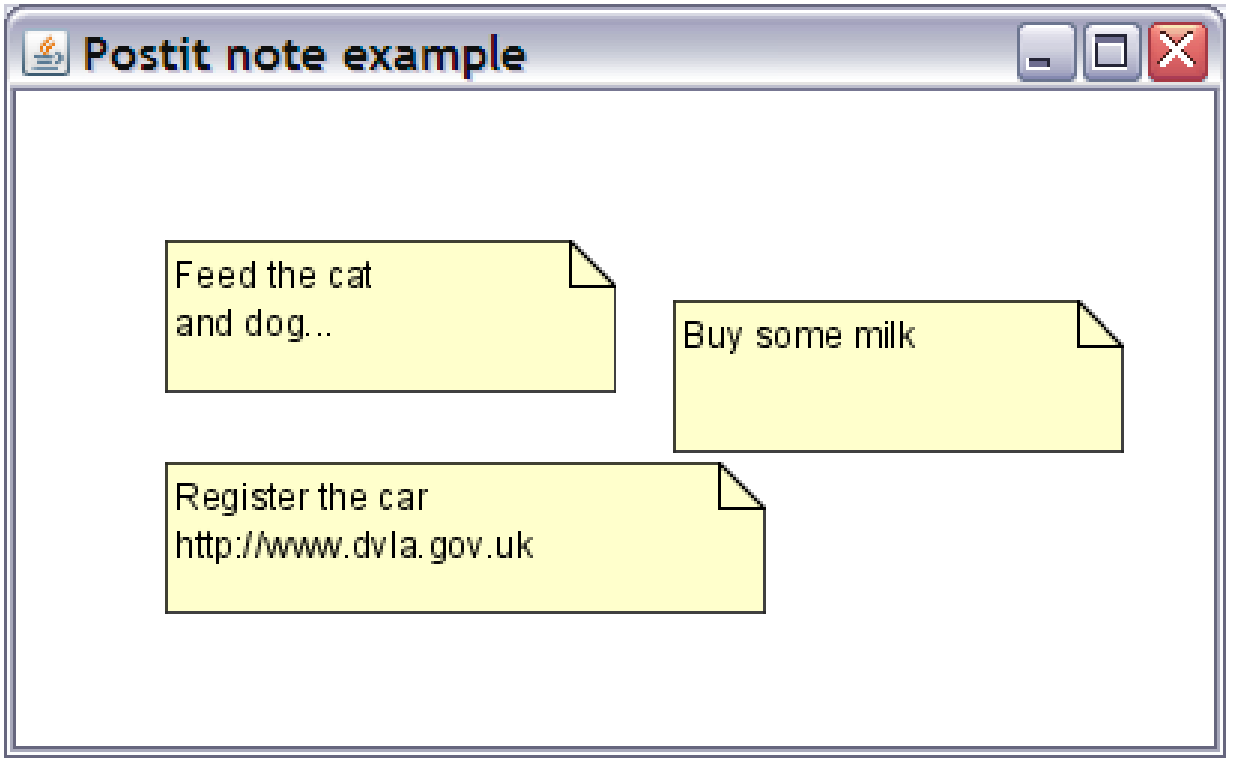
Notation for the Constructs



Applies to both composite and leaf components

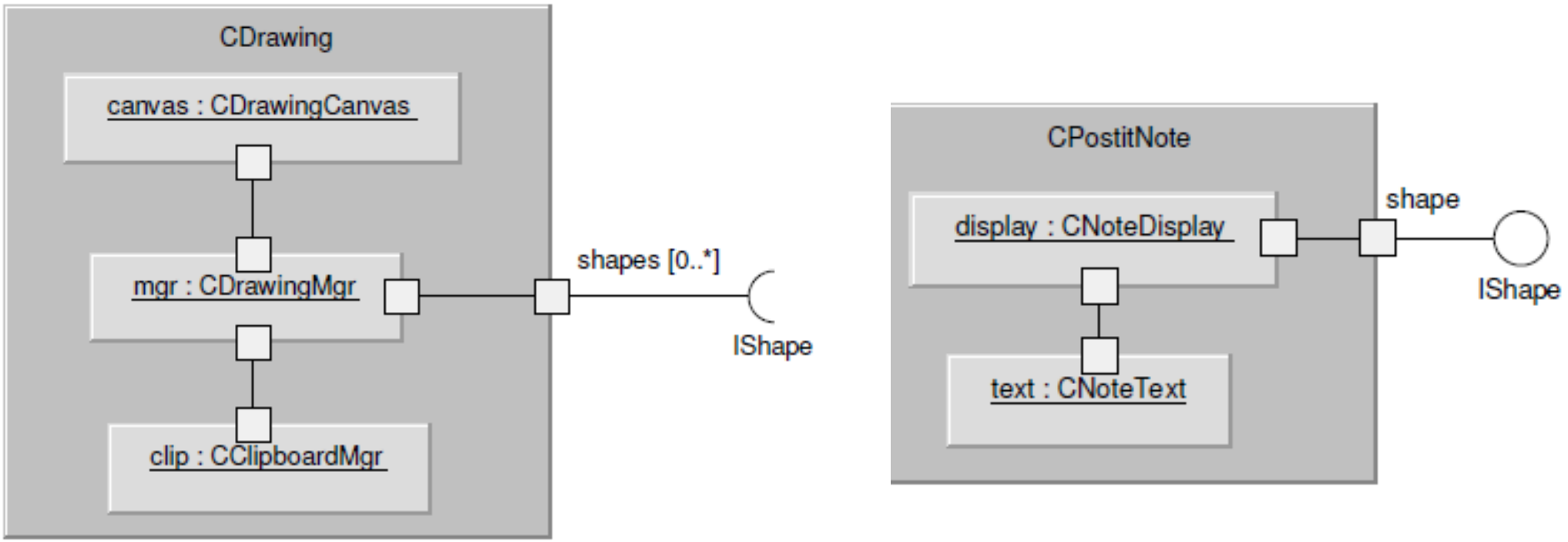
Example

A note taking application



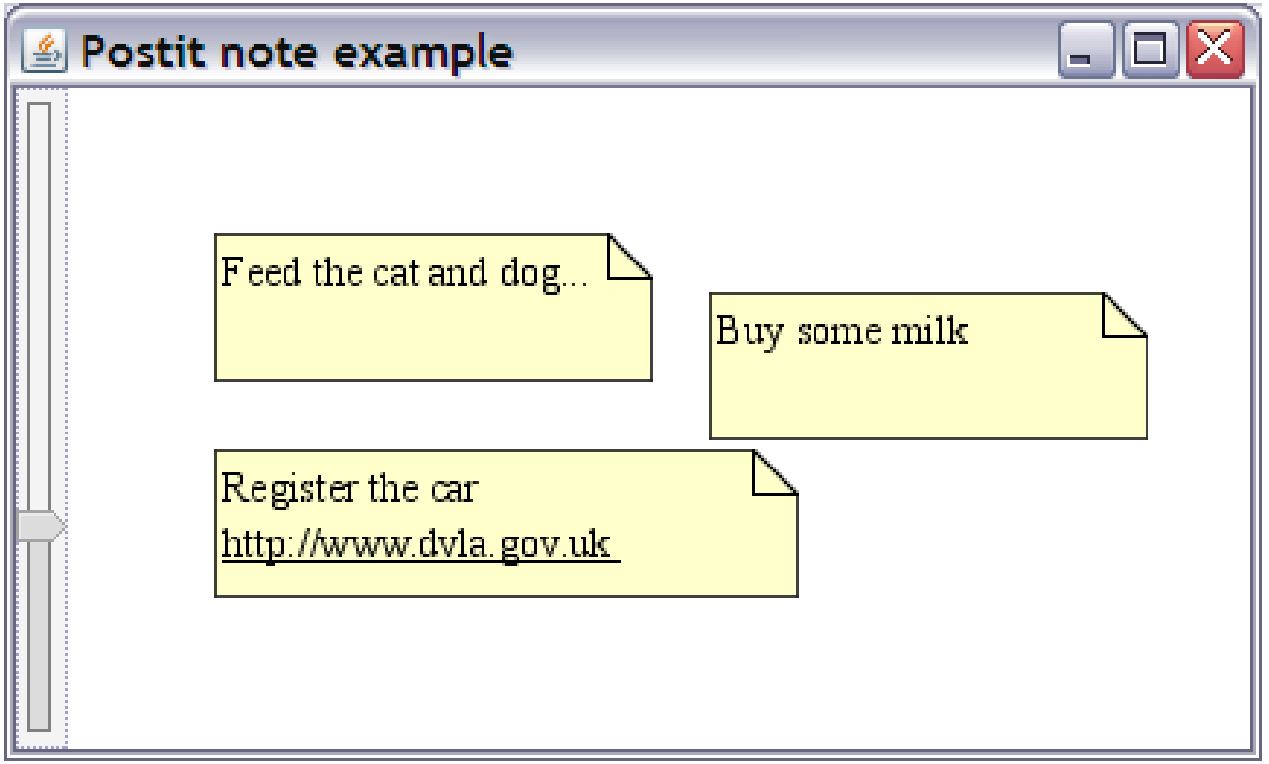
The Base Application

- Company X makes a drawing application, which has a postit-note component

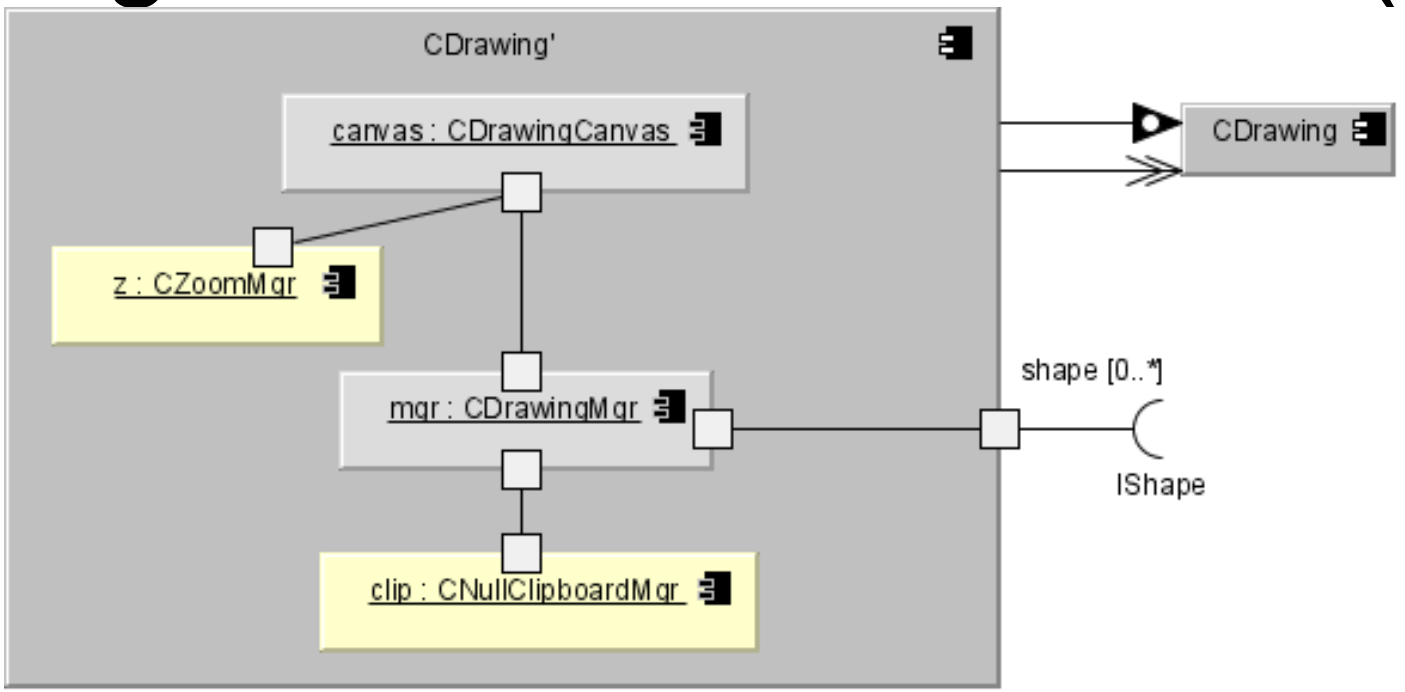


Reusing and Altering

- Company Y wishes to reuse and customise
  - Add a zoom facility
  - Remove the clipboard
  - Add hyperlinked text

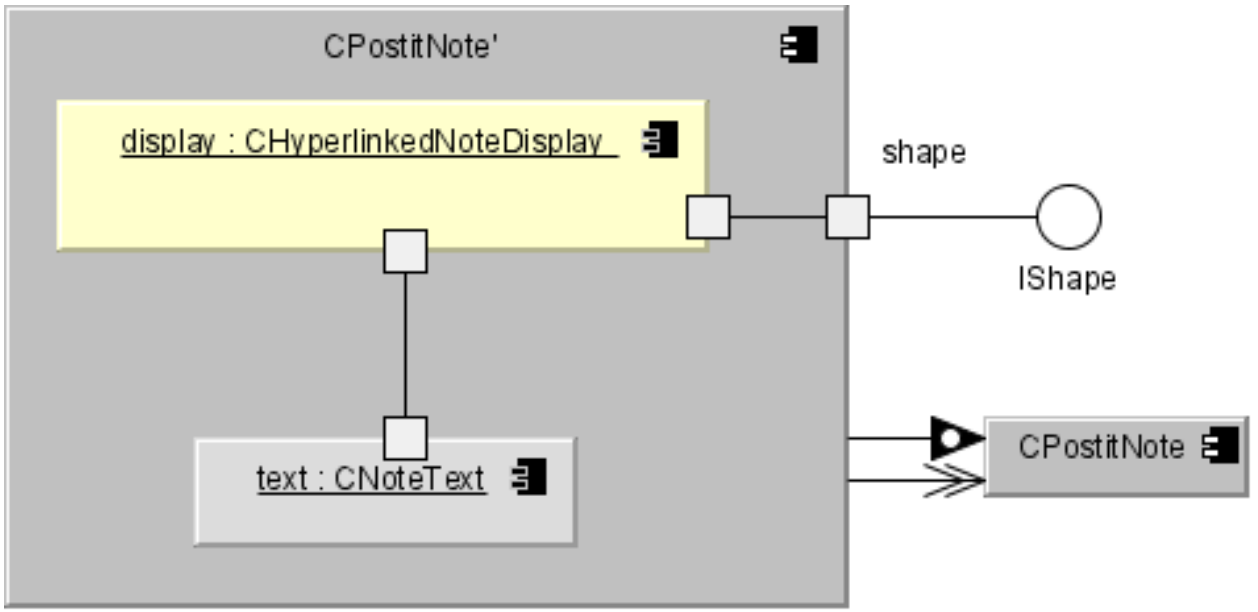


Using Resemblance to Alter (1)




```
redefine-component CDrawing
   resembles [previous] CDrawing
{
   replace-parts:
      CNullClipboardMgr clip;
   parts:
      CZoomMgr z;
   connectors:
      zoom joins zoom@z to surface@canvas; }
```

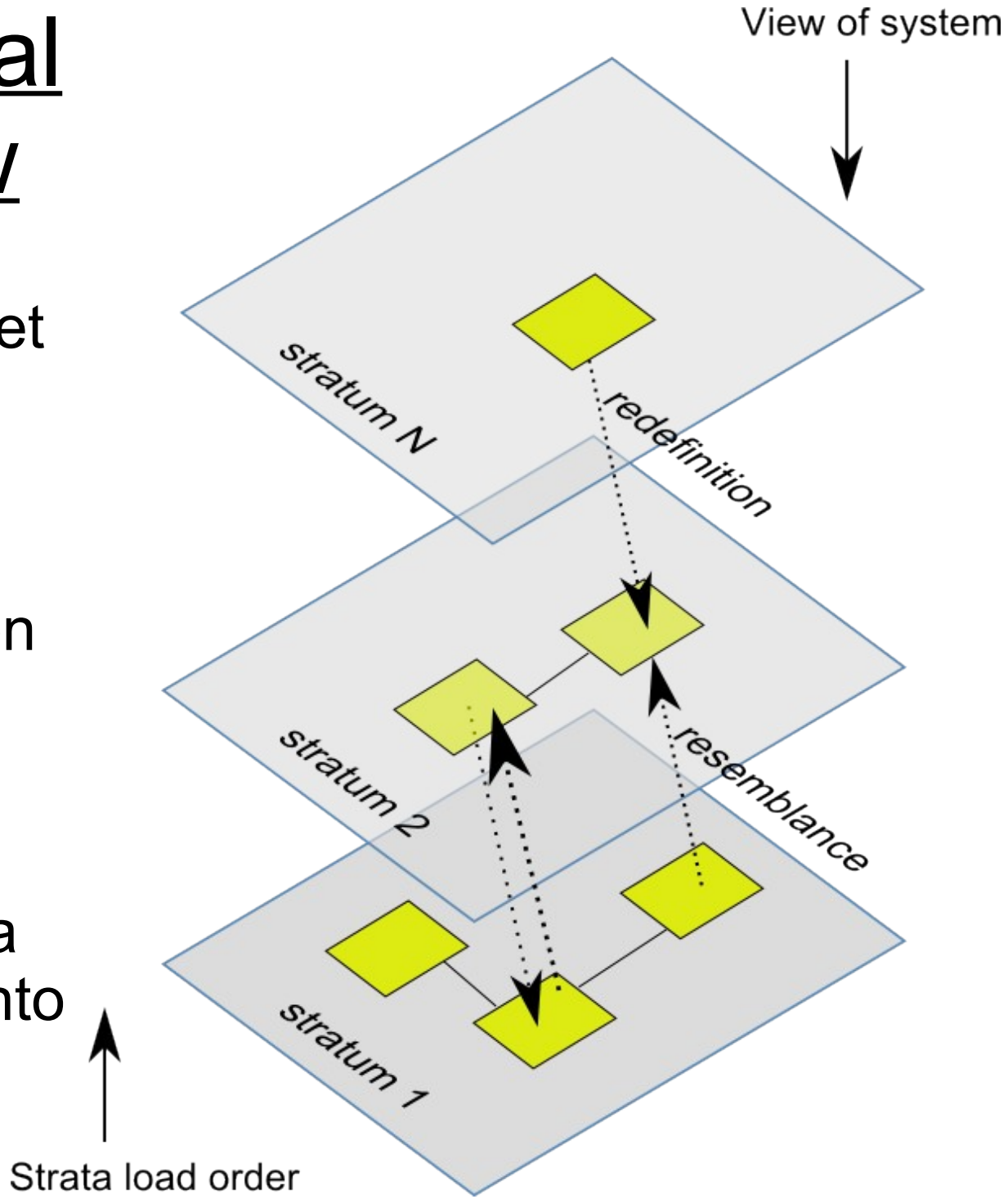
Using Resemblance to Alter (2)




```
redefine-component CPostitNote
   resembles [previous] CPostitNote
{
   replace-parts:
      CHyperlinkNoteDisplay display;
}
```

Conceptual Overview

- A *stratum* groups a set of related definitions
- *Resemblance* copies an existing component's definition into the current definition, and allows changes
- Redefinition pushes a new definition back into an existing name



Issues

- Most issues occur when combining multiple redefinitions of the same component
  - This occurs when combining independently developed changes. This related to a merge conflict in a CM system.
- How do we reason about the soundness of combined redefinitions?
  - What is the resultant system behaviour?
  - Does the combination accomplish the goals of each redefinition, or do they conflict?
- Currently only for non-distributed architectures...

Related Work

Related Work

- MAE
  - Architectural configuration management system
- ADLS
  - Darwin, ROOM, C2SADEL etc.
- Koala & product line architectural approachs
  - Parametrization for reuse
  - Variation points
- COM and other component standards
  - mechanisms versus design approach

Conclusions and Further Work

Summary

- The constructs satisfy many of the requirements:
  - **Alter**: Parts, attributes, connections can be added, deleted, replaced. Extensive changes possible.
  - **NoImpact**: Only see the changes if redefinition is applied
  - **Upgrades**: Can be phrased as another redefinition
  - **NoSource**: Most changes can be performed with just the architectural description.
    i.e. No implementation code

- Major issue is how to reason about combined redefinitions that are independently developed
  - What properties are we trying to preserve?
  - How do these relate to engineering specifications?

Further Work

- Graphical support for modelling with changes
- Expressing the properties we want preserved
  - Protocol compliance of component compositions
  - Reachability of a specified goal
- Resolving conflict between redefinitions
  - Structural
  - Behavioural
- Further work on formal models
  - Alloy model for showing structural conflict exists
  - FSP translation for protocols
  - Semantic model