COMP 3700: Software Modeling and Design

(Component-based Software Modeling)

Outline

- Component-Based SE and Reuse
- What is a software component?
- Interfaces
- Software as a "metaproduct"

Software Components

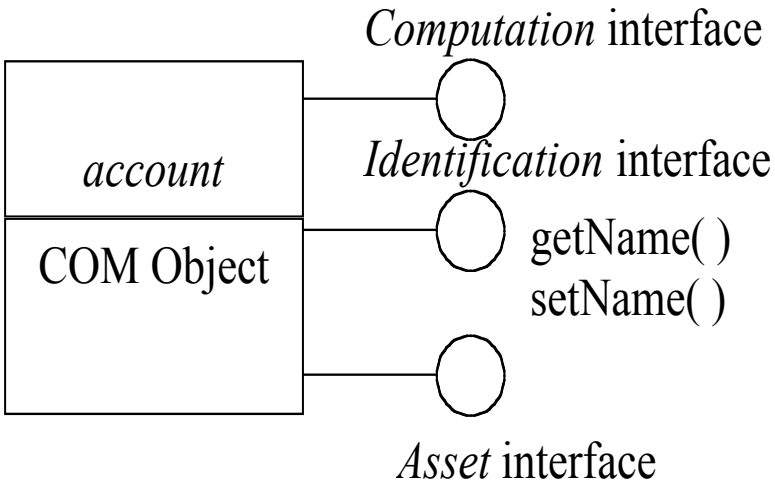
- **The main driver behind software components is *reuse*.**
- **That means we must *modularise* applications if they are to have potentially reusable parts.**
- **The expectation is that if the parts (often collections of classes) can be reused then costs will be reduced (in the long run…).**

So what's new?

- Modularisation of software is *not* new.
- What we want of a component is that
  - It may be used by other program elements (*clients*)
    - (encapsulation and low coupling – good strategies for any modular design)
  - The clients and their authors do not need to be known to the component's authors
    - This is a little bit new, and only works if all the respective authors work to a common standard

An Example Component

- A Windows executable
- Can be dynamically linked to any Windows application
- Can be composed with other COM objects



Do we get anything for free?

- Of course not!
- Components may be classes (or collections of classes), but they must satisfy *additional* guidelines:
  - So we really do understand what is provided and what is required at their interfaces
  - So that we know the framework or architecture in which they are to be used

Components as architecture

- Could view "independent components" as a category of software architectures
  - Pipes and filters
    - Unix
  - Parallel communicating processes
    - Java threads
  - Client-server
    - World-wide web;
    - CORBA – a middle layer that provides a common data bus
  - Event systems
    - Java event model and Java Beans

What is a Software Component?

- "Components are units of deployment"

  - Clemens Szyperski

Drivers for CBD

- The development of the WWW and Internet
  - Systems of loosely coordinated services
- Object-oriented design techniques and languages
- Move from Mainframe to client-server based computing
- Rapid pace of technological change
- Economic necessity of maximizing reuse

Are Components New?

- **Subroutines**
  - **Turing, 1949, *Checking a Large Routine***
- **Structured Programming**
  - **Dijkstra, 1968**
- **Libraries**
  - **NAG, 1971**
- **Information Hiding**
  - **Parnas, 1972**

Software Components

- Components are for composition
- (In principle) already existing "things" can be *reused* by rearranging them to make a new composite
- So components are about reuse
  - This drives many of the engineering requirements for software components

What is a component (2)?

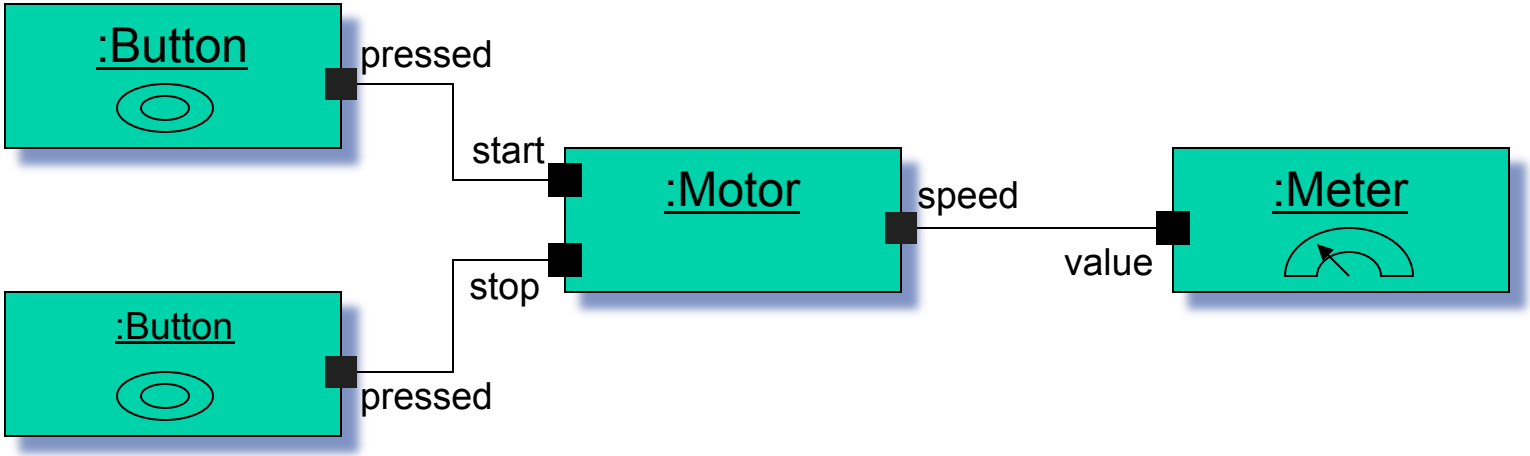
- A component makes its services available through interfaces
- And interfaces are of certain types or categories

Revised Definition

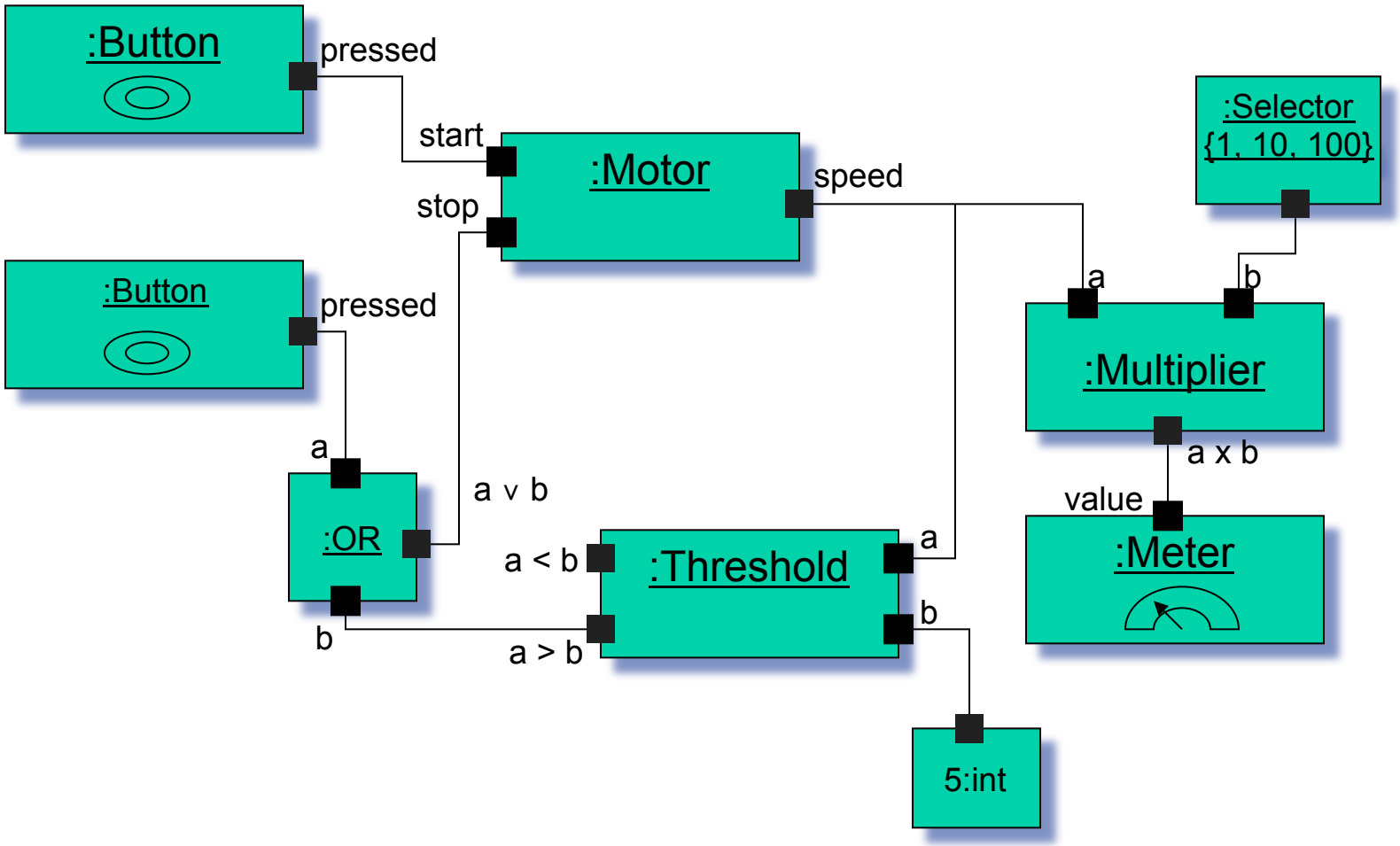
- **A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.**
- **A software component can be deployed independently and is subject to composition by third parties.**

1996 European Conference on Object-Oriented Programming

Connector Design



Connector Design



Lessons from electronics kit

Families of products from kits of components

- **Design of a component infrastucture**
  - **Basic technology - e.g. do components interact via procedure calls or remote method invocations?**
- **Component design**
  - **Components must conform to the component infrastructure**
- **Product building**

Infrastructure:

- Do pluggable connectors mean common data types across all components?
- No!
  - Local usage may not fit a common type
- Answer: Encapsulation
  - No direct access to the data of any component from outside
  - All communication should be a request defined in an interface

Revised Definition

- **A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.**
- **A software component can be deployed independently and is subject to composition by third parties.**

1996 European Conference on Object-Oriented Programming

Independent Deployment

- **Encapsulation**
- **Cannot be partially deployed**
- **Must have clear specifications of what it *requires*, as well as what it provides**
- **Must have well-defined interfaces**

Interfaces

- Interfaces allow the clients of a component to access the services provided by a component
- Different interfaces will normally provide access to different services
- Each interface specification could be viewed as a *contract* between the component and a client

Explicit context dependencies

- Components must also specify their needs
  - i.e. the context of composition and deployment
- This means both
  - The component's requires interfaces, and
  - The component world it is prepared for
    - (CORBA, COM, Java…)

Component Specification

- **Provides Interfaces**
  - The services a component can offer to a client
- **Requires Interfaces**
  - The services required by a component to help it deliver its promises
- **Context of Use**
  - The "world" the component lives in

Interfaces as contracts

- Can view an interface specification as a "contract" between the client and a provider
- So the interface specification must state:
  - What the client needs to do
  - What a provider can rely on
  - What a provider must promise in return
  - What the client can rely on

Pre- and Post-Conditions

- **Pre-conditions:**
  - **What the client must establish before calling the operation**
  - **The provider can rely on this condition being true whenever the operation is called**
- **Post-conditions:**
  - **What the provider must establish before returning to the client**
  - **The client can rely on this condition being true whenever the call to the operation returns**

Example

```
public interface Directory {
    public void addEntry(String name, File file);
    // pre name != "" and file != null
    // post File file = map.get(name)
}
```

Associate pre- and post-conditions to every method in the interface

The nature of software

- "Delivery of software means delivering the blueprints for products"
  - Clemens Szyperski
- When software is installed on a computer, an instance of the product is instantiated
- The computer can instantiate the product one or more times
- Better to view software as a "metaproduct"

Summary

We have:

- Seen some of the drivers behind the introduction of component-based software engineering
- Explored some definitions of software components
- Identified the importance of specifying requires and provides interfaces