ELEC 372
Engineering Software Structures

Week 10: Software Architecture

Lecture 1

Examples [Shaw&Garlan]

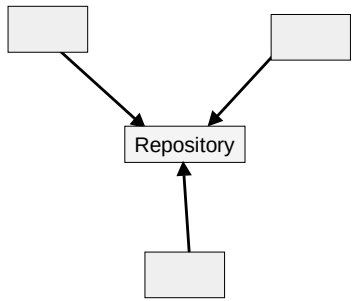
- Pipes and Filters
- Layers
- Event-Based
- Data-Centered
- Interpreter
- PAC

Data-Centered Architecture Style

- Repository
  - Allows data to be allowing shared amongst a set of components
- Blackboard
  - Active repository cooperative, distributed decision making

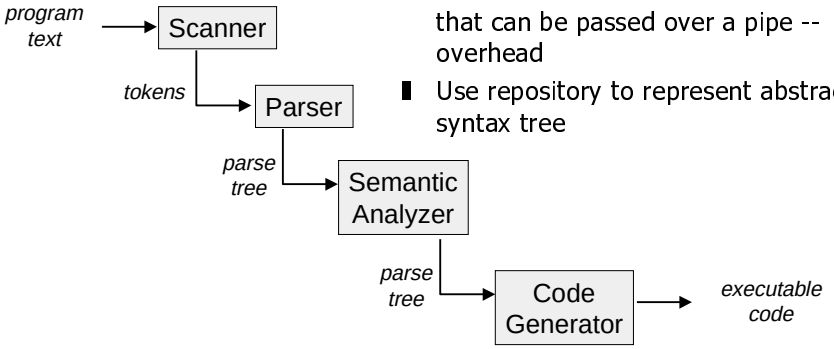
Repository

- Set of components share data
- E.g., database or data structure shared by multiple components
- Connectors permit components to read/write to repository

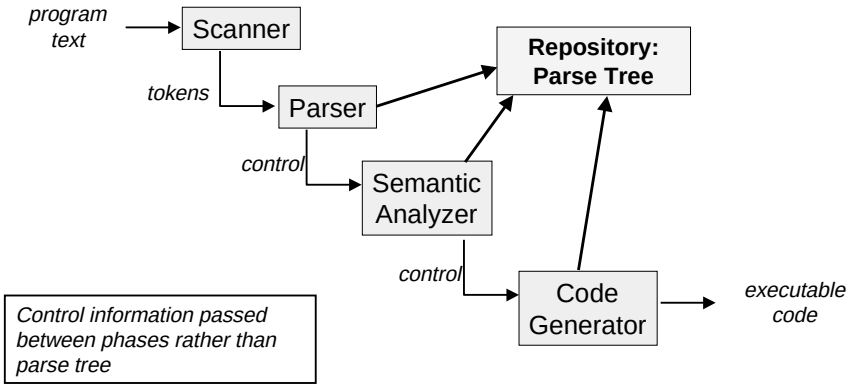


Example

- Compiler based on pipes and filter architecture style
- Parse tree is complex data structure passed between phases
- Needs to be converted into a stream that can be passed over a pipe -- overhead
- Use repository to represent abstract syntax tree

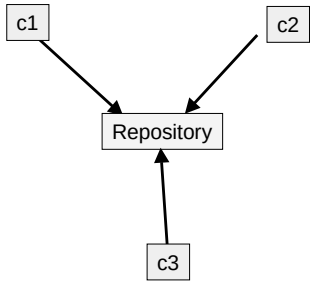


Compiling as pipe and filter architecture augmented with repository



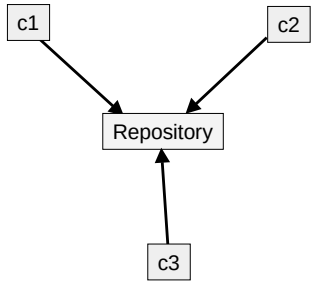
Properties of Repository Architecture Style

- Performance
  - For complex data structures, saves overhead of passing data structure from one component to another
  - If components on different machines, this overhead might be considerable
  - If components on different machines, cost of accessing repository might be high



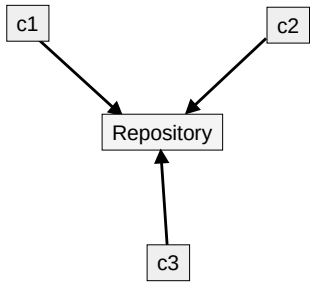
Properties of Repository Architecture Style

- Performance
  - E.g., if components and repositories on different machines, each call to repository goes over network
  - Need to balance this cost vs cost of moving data between phases using pipes and filters
  - E.g., if c2 makes 100 calls for information to repository, each call takes 1 ms, total time = 1 s. Compare to time to pass repository from c1 to c2



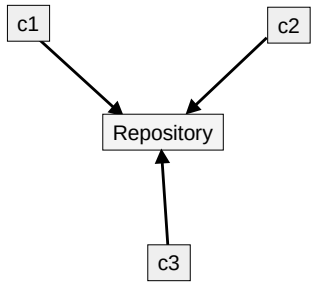
Properties of Repository Architecture Style

- Performance
  - Need to ensure repository can scale to load of components making requests of it
  - Ideally, do not want components waiting while repository operates on long queue of requests
  - This analysis is complex, out of scope of this course



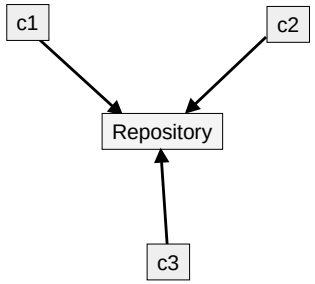
Properties of Repository Architecture Style

- Development Time
  - Numerous infrastructures for repositories already exist -- e.g.
    - Commercial databases
    - Java Remote Method Invocation (RMI)
  - Need to balance savings in development time vs need to learn infrastructures



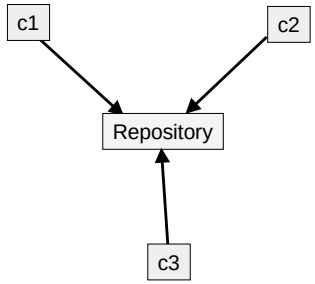
Properties of Repository Architecture Style

- Security
  - Repository potentially available to anyone who can connect to it
  - Particularly dangerous in distributed setting where repository available via Internet protocol
  - E.g., don't want medical patient information available to anyone who knows IP address of medical database



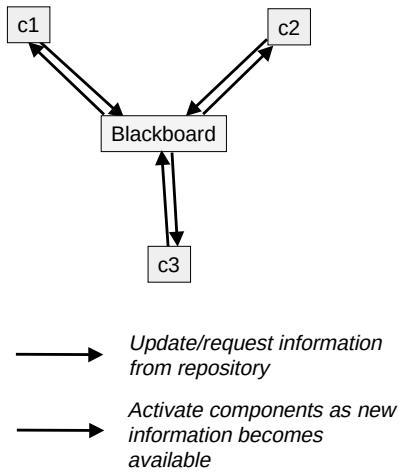
Properties of Repository Architecture Style

- Availability
  - Repository becomes critical component in system
  - Failure in repository causes entire system to fail



Blackboard Architecture Style

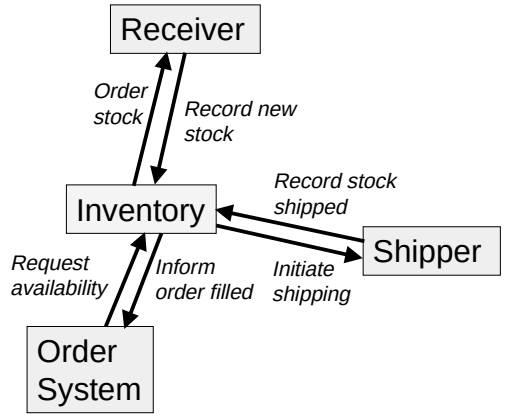
- Variant of repository
- Repository is active
  - Can activate components in response to changes in blackboard contents
- Typical application
  - Use blackboard to mediate cooperative activity amongst components
  - Tasks where desired outcome is imprecisely defined or error prone



Example: Inventory System using the Blackboard Architecture

- *Inventory* blackboard used to coordinate filling orders
- *Order* system initiates orders
- *Receiver* records new stock as it arrives
- *Shipper* sends order to be packaged
- All communication via inventory

Is this a good solution for your project? What are the advantages/disadvantages?

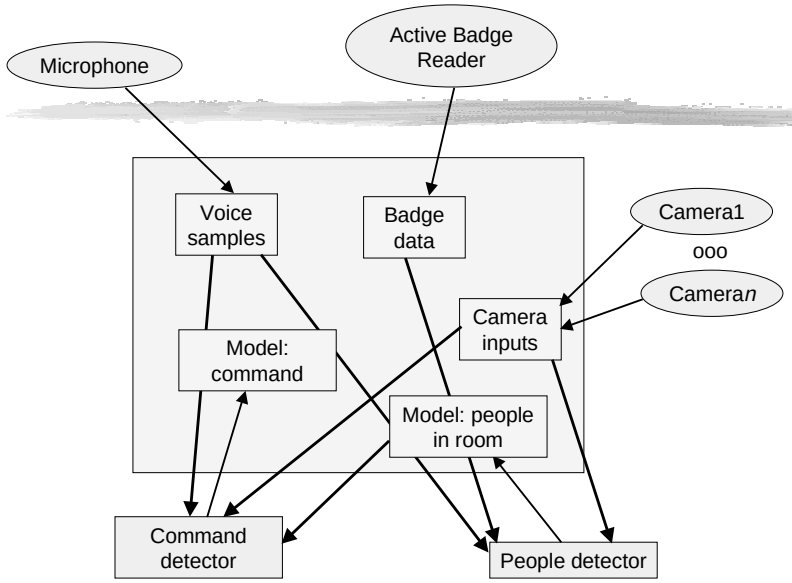


Example: Smart office

- Tracking who is in the office
  - Using: Active badges, voice recognition, cameras
  - Allows
    - Ease of finding colleagues
    - User authentication (no need to have door keys, logins at machines)
    - People's environments to follow them
      - Sitting down at someone else's PC can be just like sitting down at your own
- Smart commands
  - Use gestures/voice to
    - Turn on projector, connect to PC
    - Record contents of whiteboard save them under my username
  - Combine modalities
    - Say "Save *this* on my laptop", while pointing at whiteboard

Example: Smart office

- Application depends on combining a set of physical devices
  - Cameras, microphones, door switches
- and a set of services
  - Voice recognition speech recognition, gesture recognition
- and a set of models
  - Who is in room, what command are they execution



Interesting scenarios

- Person enters room, but their active badge is malfunctioning
- When they first speak, voice recognition detects their presence

ELEC 372
Engineering Software Structures

Week 10: Software Architecture

Lecture 2

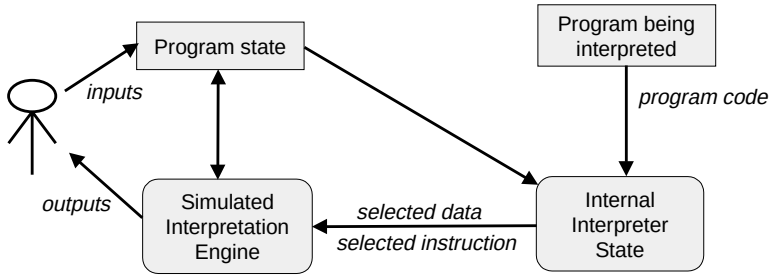
Examples [Shaw&Garlan]

- Pipes and Filters
- Layers
- Event-Based
- Data-Centered
- Interpreter
- PAC

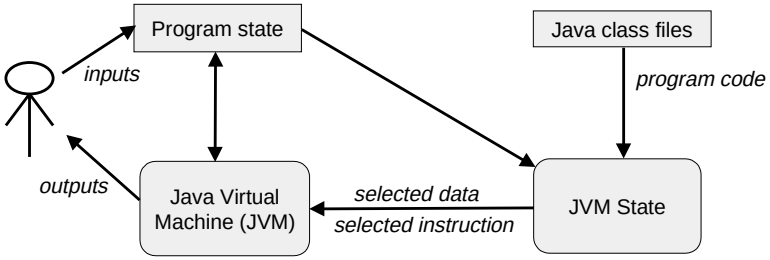
Interpreter Architecture Style

- Develop domain-specific language for problem area
- Express solution in terms of this language
- Include *interpreter* in system to evaluate the solution
- Motivation
  - Providing higher level language for problem domain simplifies programming problem
  - Examples
    - XML parsers
    - Extensible interpreters such as JVM, tcl
    - Rule-based/expert systems

Interpreter Architecture [Shaw&Garlan p. 27]

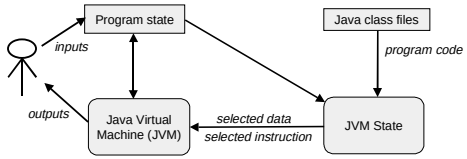


E.g., Interpreting Java Programs



Interpreting Java Programs

- Java programs are represented as a set of class files
  - Contain byte code describing operations on the Java Virtual Machine
- These programs are executed as if they were being run on a native machine
- Program state is actually represented in JVM



Example: Processing XML

- XML (eXtensible Markup Language) is a way of encoding structured data for
  - Archiving
  - Interchanging between applications
  - Sending over the Internet
    - (e.g. e-commerce business-to-business applications)
- XML processing is implemented via the interpreter architecture style

See e.g. http://www.xml101.com:8081/xml/

XML

- XML documents consist of tagged text
- Tags indicate semantics of data
- Tags may be arbitrary
  - Follow format of

```
<tagname attr1=value1,…,attrn=valuen>
    contents
</tagname>
```

```
<books>
  <book isbn="0201199300">
    <title>Software Architecture in Practice</title>
    <author>Len Bass</author>
    <author>Paul Clements</author>
    <author>Rick Kazman</author>
  </book>
</books>
```

XML DTD's

- A DTD (Document Type Definition) describes legal use of tags in a particular kind of XML document

```
<!DOCTYPE listOfBooks [
        <!ELEMENT books   (book*)>
        <!ELEMENT book    (title,author+)>
            <!ATTLIST book isbn CDATA "0">
        <!ELEMENT title   (#CDATA)>
        <!ELEMENT author  (#CDATA)>
    ]>
<listOfBooks>
```

Example document following this DTD

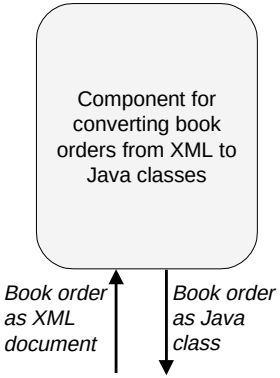
```
<books>
    <book isbn="0201199300">
      <title>Software Architecture in
          Practice</title>
      <author>Len Bass</author>
      <author>Paul Clements</author>
      <author>Rick Kazman</author>
    </book>
</books>
```

DTD's specify

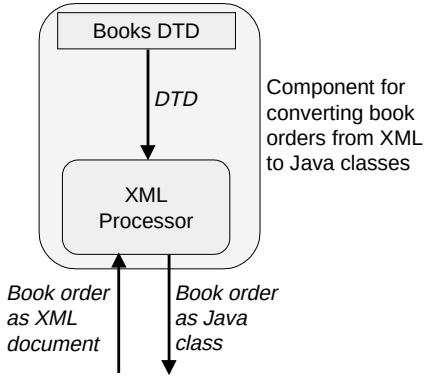
- What tags are legal
- How tags are combined
  - A book must have a title and 1 or more authors
  - A set of books consists of 0 or more books
  - A book has an ISBN attribute (unique number for books), which is character text (CDATA), with a default value of "0"


```
<books>
    <book isbn="0201199300">
      <title>Software Architecture in
          Practice</title>
      <author>Len Bass</author>
      <author>Paul Clements</author>
      <author>Rick Kazman</author>
    </book>
</books>
```

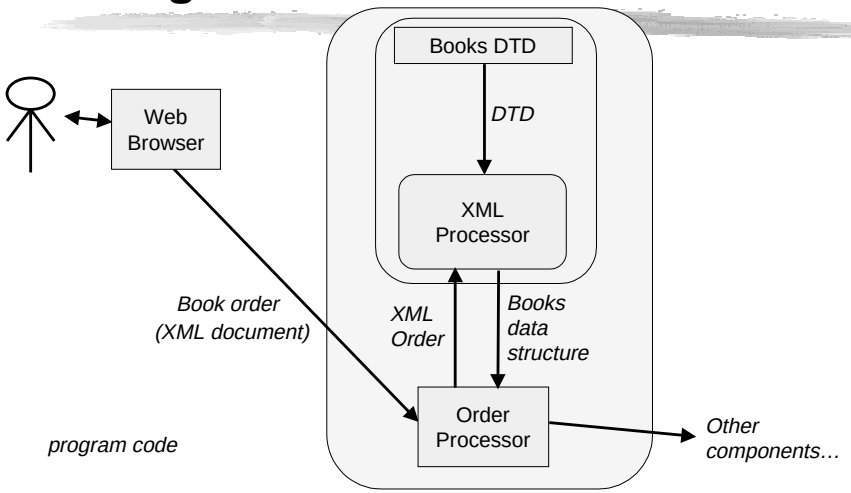
Using XML in a System for Ordering Books



Using XML in a System for Ordering Books



Using XML in a System for Ordering Books



XML Processor for Ordering Books

- XML Processor is an interpreter for DTD's
- When given the Books DTD, becomes a component capable of processing book orders
- Advantages
  - Can reuse XML processor in many contexts
  - If format of book orders change, just need to change DTD, not code

Attributes of Interpreter Style

- Development Time
  - Tremendous potential for reuse
- Performance
  - Interpretation may be slower than hand-coded solutions
- Modifiability
  - Can be improved by restricting modification to data being interpreted

ELEC 372
Engineering Software Structures

Week 10: Software Architecture

Lecture 2

Examples [Shaw&Garlan]

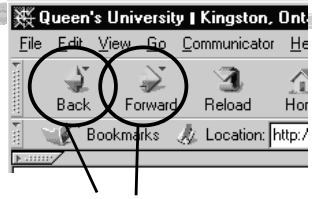
- Pipes and Filters
- Layers
- Event-Based
- Data-Centered
- Interpreter
- PAC

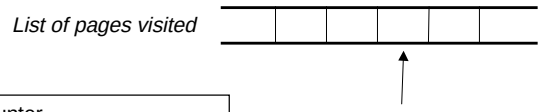
The PAC Architecture Style

- Supporting development of user interfaces
- *Conceptual* architecture that can be mapped to multiple *concrete* architectures
- Agent-based
  - Program decomposed into a set of agents, each responsible for some part of the task of interacting with the user

Example: PAC Architecture for a Set of Buttons

- Assume an interface has a set of buttons controlling navigation
  - "Back" means return to previous page in history, "Forward" means go forward to last page we moved back from (if there is one)





- "Back" decrements current page counter
- "Forward" increments current page counter
- If current page is front of list, "Back" button is disabled
- If current page is end of list, "Forward button is disabled
- Going to a new page enqueues that page after the current page, sets current page to that page

Starting with just a button…

- Buttons have
  - A *label* (e.g. "Back", "Forward")
  - A *state* (disabled, enabled)
- Buttons have local behaviour
  - "Rollover" - become highlighted when mouse moves over them
  - Become depressed following mouse down
  - Become released following mouse up


Enabled button


Disabled button


"Rolled over" button

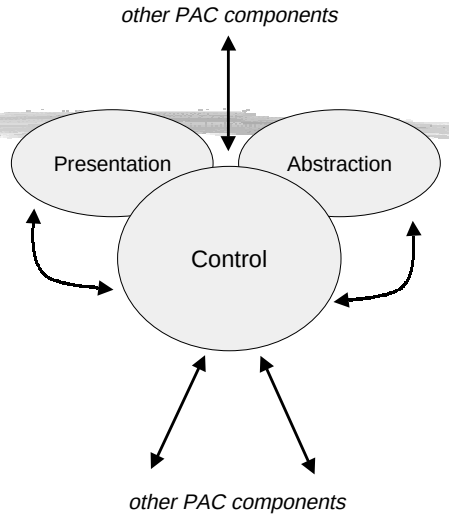

Depressed button
(Couldn't make accurate picture of this one)

Button Abstraction

- A button's state is therefore
  - Label (String value)
  - Is Enabled? (Boolean value)
  - Is "rolled over"? (Boolean value)
  - Is depressed? (Boolean value)
- Therefore, the state of a button is captured in three Boolean values and a string value
- In PAC, this state is called the button's *Abstraction*

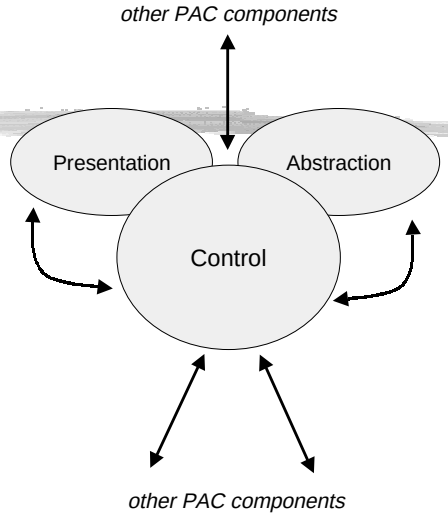
PAC Components

- PAC architectures constructed from a set of components
- Each component composed of three *facets*
  - *Presentation:* handles user input and redraws the display
  - *Abstraction:* underlying state of interactive entity
  - *Control:* mediates communication between Presentation, Abstraction and other PAC components



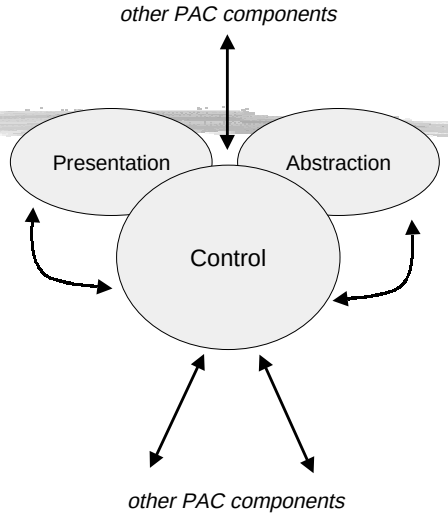
PAC Button

- Abstraction
  - Java object containing methods to set/get
    - Button label
    - Enabled/disabled state
    - Rolled-over state
    - Depressed/released state



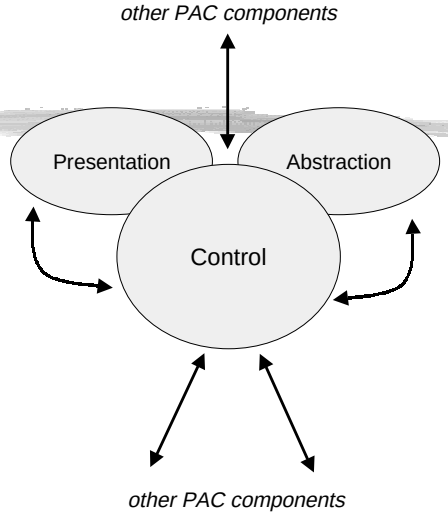
PAC Button

- Presentation
  - Accepts mouse input
    - mouse down
      - If button enabled, depresses button
    - mouse up
      - If button depressed, releases button
    - mouse enter
      - If active, puts button in rollover state
    - mouse leave
      - If rolled over, removes rollover state



PAC Button

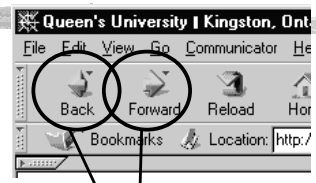
- Presentation
  - Responsible for redrawing button
    - When changes in enabling, depressed/released, rolled over, redraws with appropriate animation, GIF image

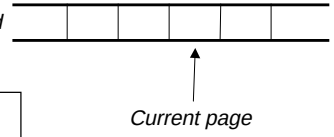


Question: Why doesn't input part of Presentation simply handle redrawing?

Building an Architecture with PAC Components

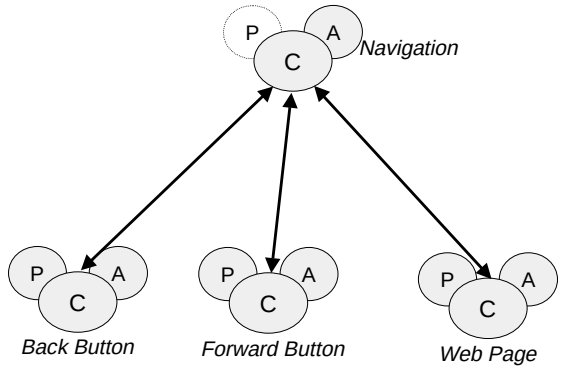
- Require a PAC component for each button
- Require a component for navigation
- Require components for the rest of the browser, such as the display of the web page





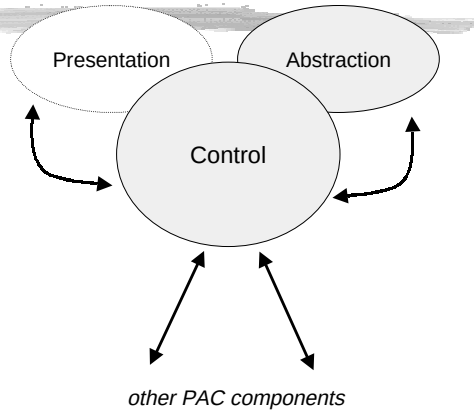
- "Back" decrements current page counter
- "Forward" increments current page counter
- If current page is front of list, "Back" button is disabled
- If current page is end of list, "Forward button is disabled
- Going to a new page enqueues that page after the current page, sets current page to that page

PAC Architecture

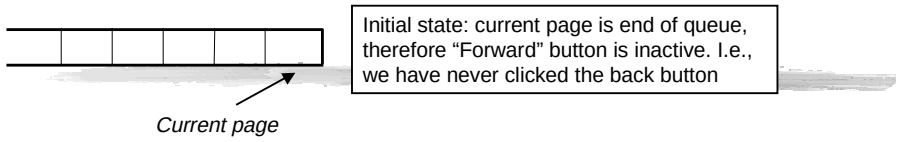


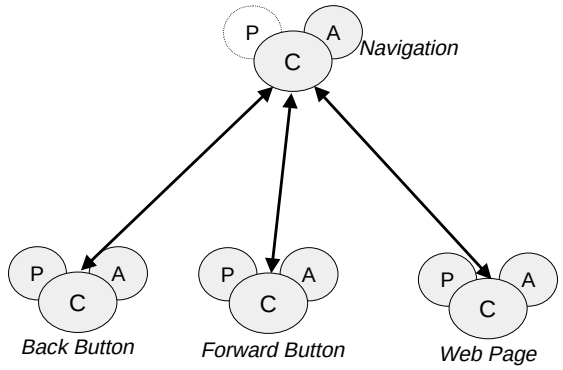
Navigation Component

- Abstraction contains queue of pages, current page pointer
- If pages before current page, enables back button
- If pages before current page, enables forward button
- Actions from buttons manipulate current page
- No Presentation facet required

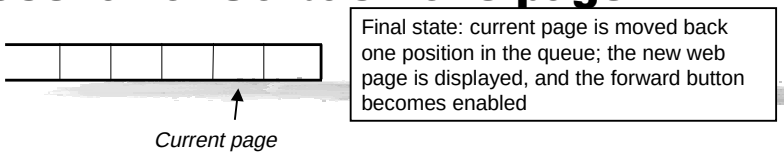


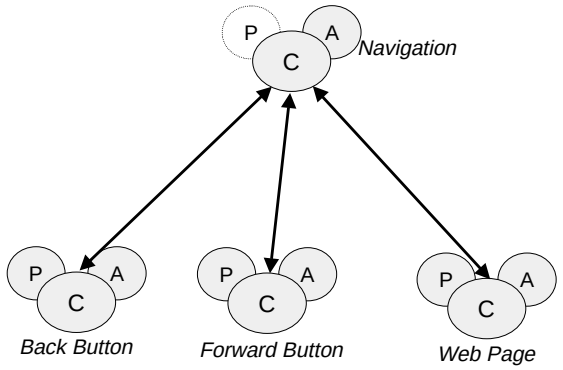
Scenario: Go back one page



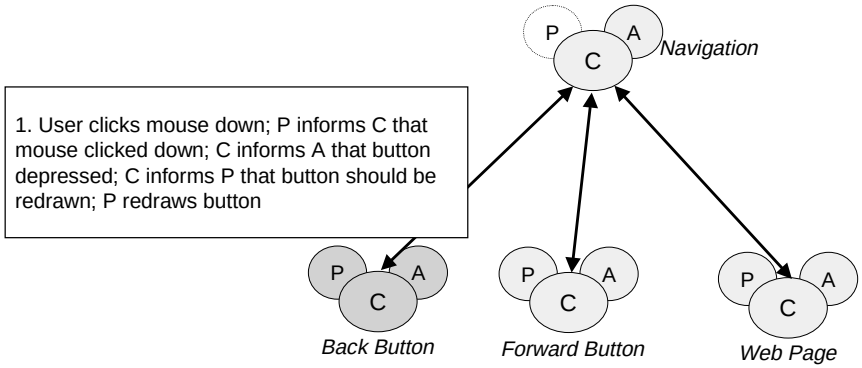


Scenario: Go back one page

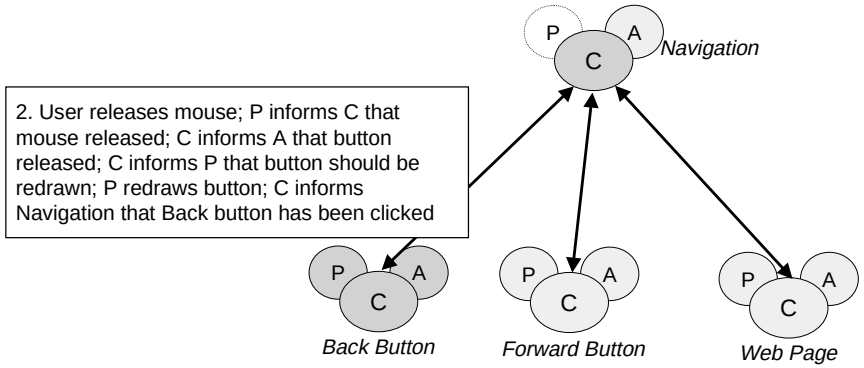




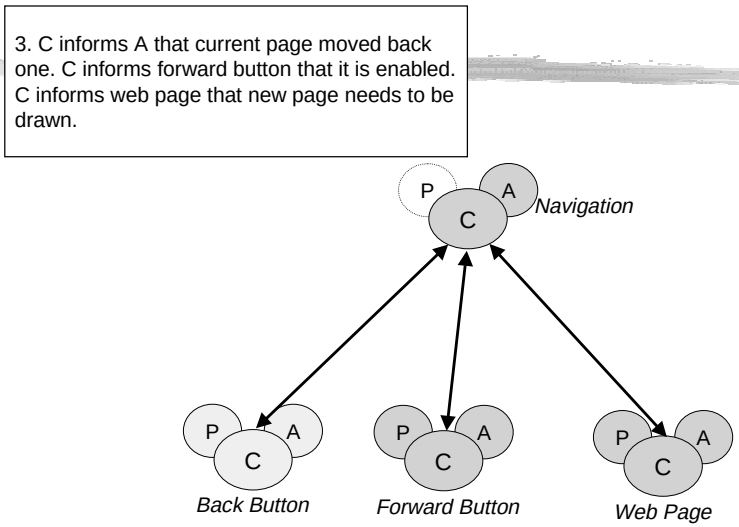
Scenario: Go back one page



Scenario: Go back one page



Scenario: Go back one page



ELEC 372
Engineering Software Structures

Week 10: Software Architecture

Lecture 3

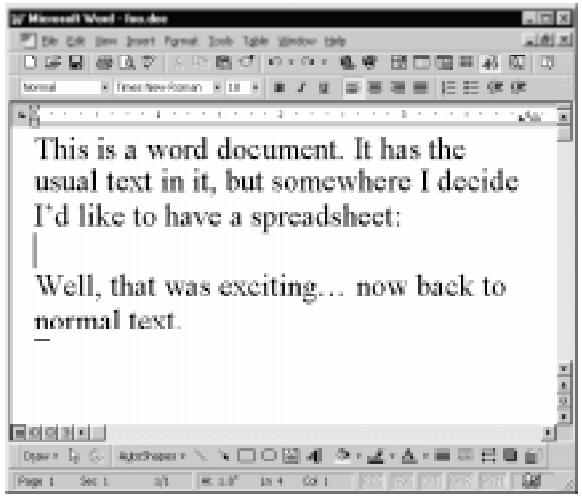
Case Study

- Microsoft's COM architecture
  - Component Object Model
  - Used for building Microsoft's large applications
    - Word, Excel, PowerPoint, …
  - Variants
    - DCOM (distributed version)
    - OLE (Object Linking and Embedding)
      - Built on top of COM to allow document types to be combined
    - ActiveX (Web version)

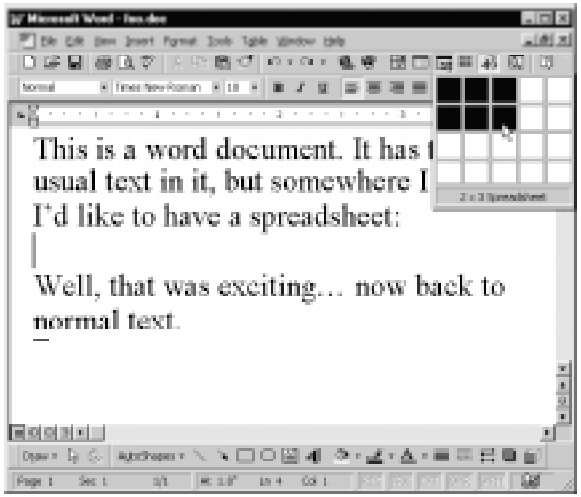
Microsoft OLE

- Combines
  - Object-Oriented Architecture Style
  - Interpreter Architecture Style
  - Layer Architecture Style
  - Event-Driven Architecture Style

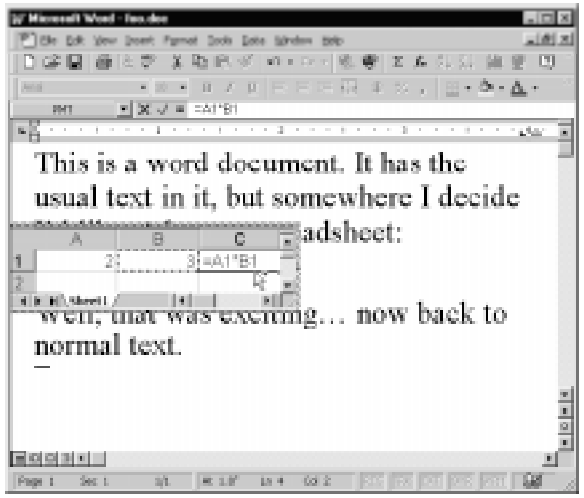
Motivation: Object Embedding



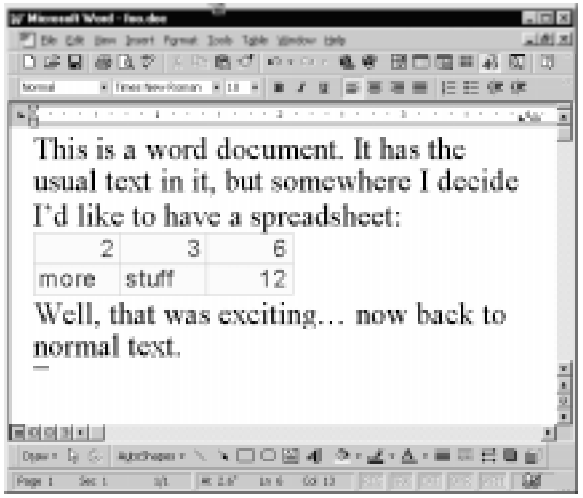
Motivation: Object Embedding



Motivation: Object Embedding



Motivation: Object Embedding



What's Actually Going On…

- Microsoft applications are made up of a set of components
- E.g., in Word
  - *Document* component holds the value of one document
  - *Window* component implements UI functionality of one window
  - *List* component holds formatting information for a list in a document
  - Etc…

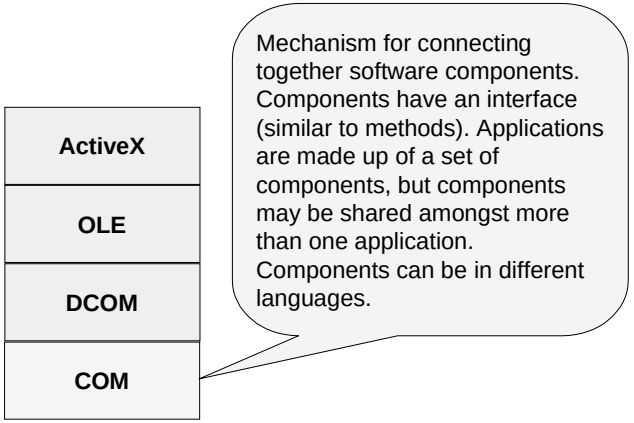
What's Really Going On…

- Similarly, Excel spreadsheet documents are made up of a number of components
- COM (Component Object Model) allows our Word document to embed elements of spreadsheet documents
- Therefore create a compound document involving both Word and Excel components

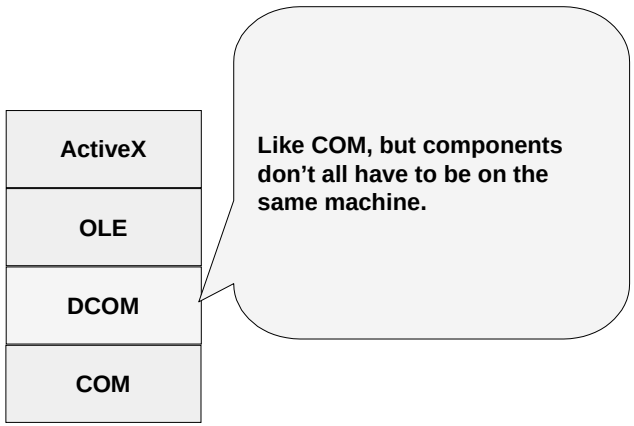
COM as a Layered System
(Idealized View)



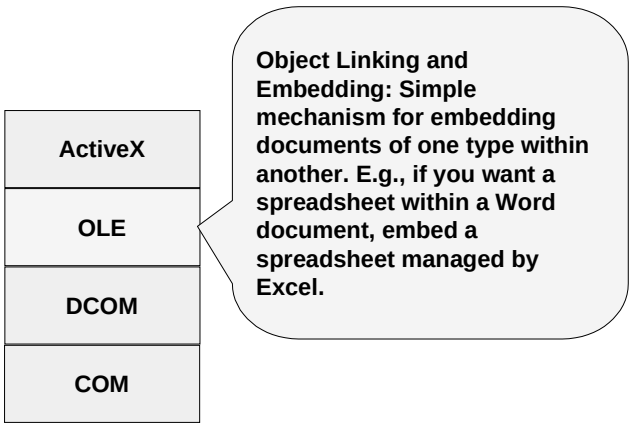
COM as a Layered System
(Idealized View)



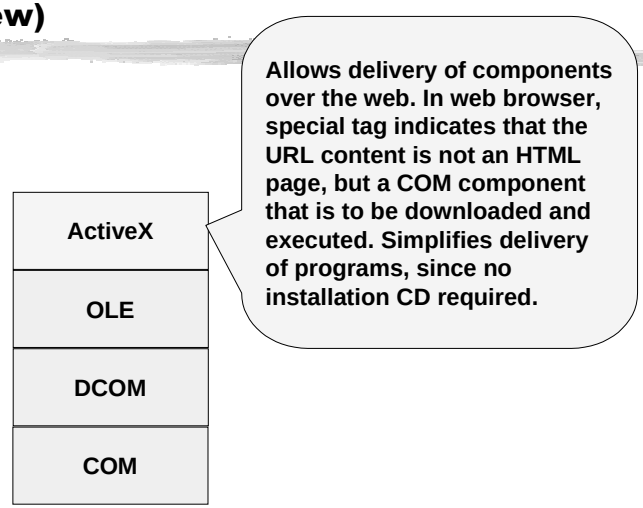
COM as a Layered System
(Idealized View)



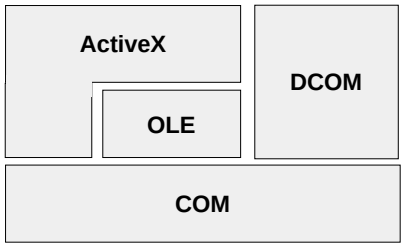
COM as a Layered System
(Idealized View)



COM as a Layered System
(Idealized View)



COM as a Layered System
(Slightly more accurate view)

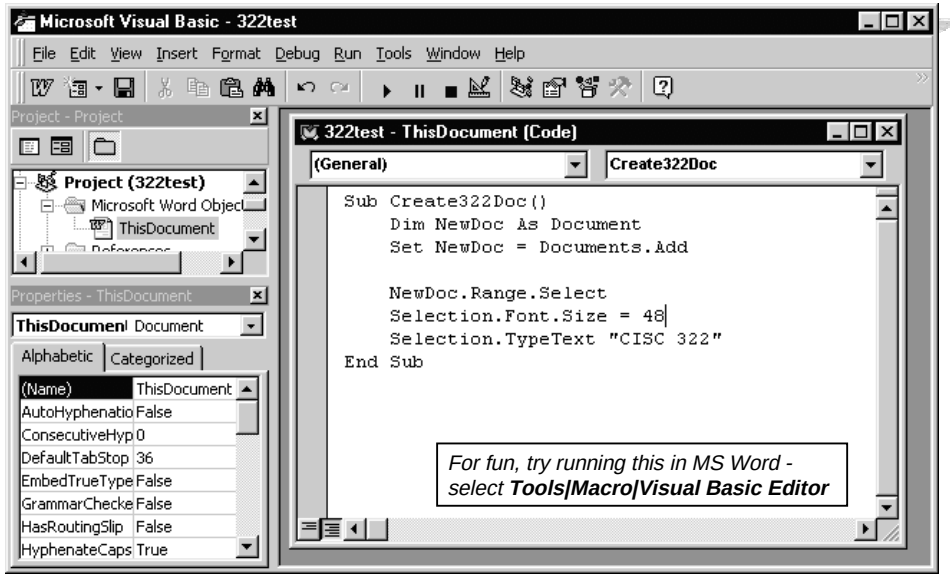


COM as an OO System

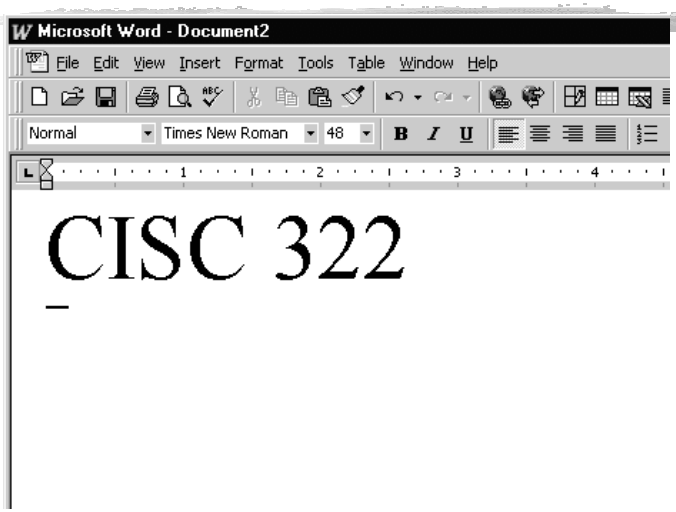
- Applications are built as a set of COM components
- These components are constructed using standard OO techniques
  - Inheritance
  - Aggregation
  - Polymorphism

References:
- *D. Mackenzie, Word 2000 VBA, WROX Press, 1999*
- *D. Chappell, Understanding ActiveX and OLE, Microsoft Press, 1996*

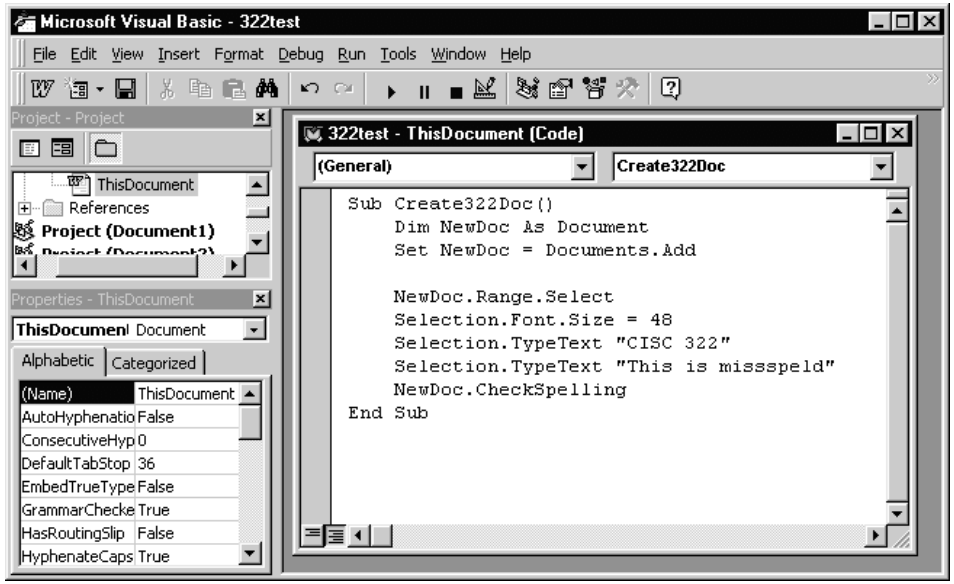
A Visual Basic COM (OLE) Program



Result of Running the Program within MS Word



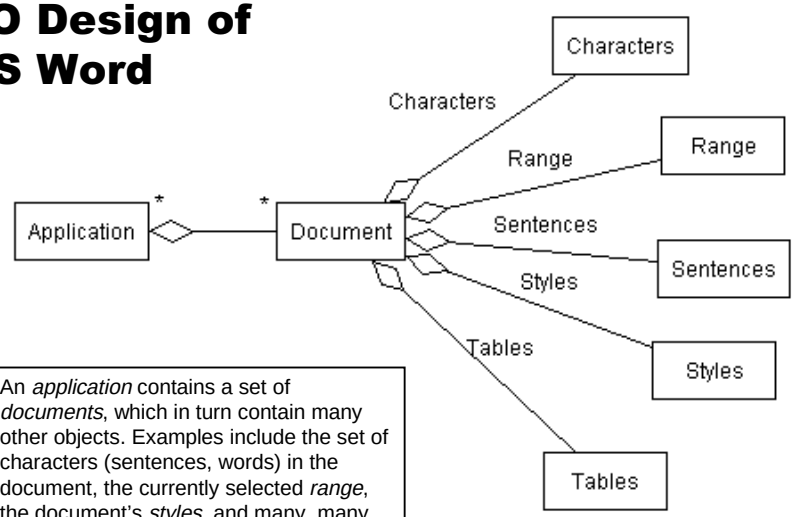
A Visual Basic COM (OLE) Program



Result of Running the Program within MS Word

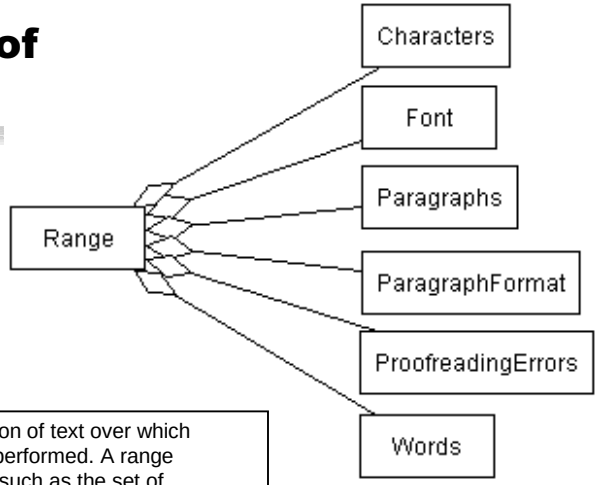


OO Design of MS Word



An *application* contains a set of *documents*, which in turn contain many other objects. Examples include the set of characters (sentences, words) in the document, the currently selected *range*, the document's *styles*, and many, many others.

OO Design of MS Word



A *range* represents a section of text over which editing operations can be performed. A range contains many properties, such as the set of *characters/sentences/words* in the range, the set of *fonts* used, the *paragraph format,* the set of errors returned by checking tools, and many more.

Example Operations

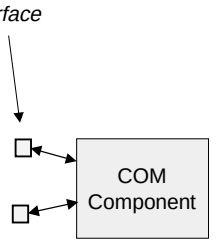
- Range
  - *void insertBefore (String)* -- inserts given string before the range
  - *void cut ()*-- copies the range to the clipboard, deletes the range
  - *void sort (…)* -- sorts the range according to given parameters
  - *SpellingSuggestions getSpellingSuggestions (…)* -- returns spelling suggestions for any misspelled words in the range
  - *long delete (Unit, Count)* -- deletes *count* number of *unit* items
  - …and dozens more…

Connectors in COM

- COM components can be created in many different languages
  - E.g., C++, Java, C#, Visual Basic, …
- Components can reside in different applications, in the operating system and (with DCOM) on different machines
- Connectors therefore are not simple method call connectors
  - What object do you call?
  - What format are parameters in? E.g., when passing a Visual Basic object as a parameter to a C++ COM object?

Connecting COM Components

- Components provide *interfaces*
  - Language-independent points of service
  - Interfaces described in *Interface Definition Language* (IDL)
  - Language-dependent IDL translators transform method calls from native language to IDL format
  - IDL is very close to C++, so C++ translation is trivial

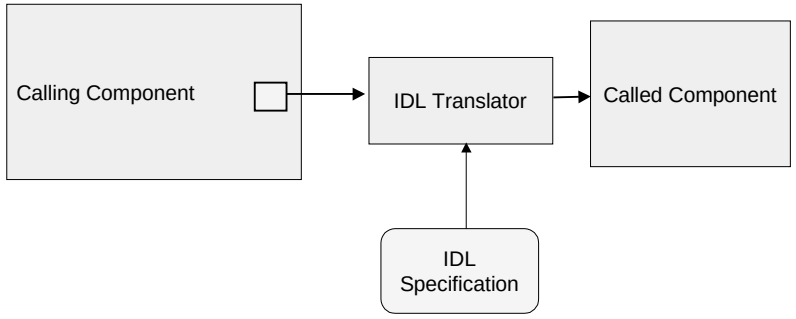


Example IDL Specification

```
[object, uuid(E7CD0D00-1827-11CF-9946-444553540000) ]

interface ISpellChecker : IUnknown {
   import "unknwn.idl";
   HRESULT LookupWord([in] OLECHAR word[31],
                      [out] boolean *found);
   HRESULT AddToDictionary([in] OLECHAR word[31]);
   HRESULT RemoveFromDictionary([in] OLECHAR word[31]);
}
```

Connecting COM Components



The Object-Oriented Architectural Style

- Object-orientation can be seen as an architectural style
  - Components = objects
  - Connectors = call
- We have already seen the fundamental elements of object-orientation
  - Encapsulation
  - Inheritance
  - Polymorphism

The Object-Oriented Architectural Style

- Unified Modeling Language (UML) provides building blocks to describe OO architectures
  - Class diagrams
    - Define types of components, relationships between components
  - Object diagrams
    - Define runtime structure of architecture, showing how classes are instantiated into objects

The Object-Oriented Architectural Style

- Sequence diagrams
  - Shows timeline of how components communicate with each other in real scenarios
- Use-Case diagrams
  - Show how users interact with system

Relation between OO Design and Software Architecture

- Software architecture describes the high-level structure of a system in terms of components and connectors
- OO Design (e.g. using UML) can express software architectures
  - Can express not just structure but also dynamic information
- OO Design can also express information at a much lower level than architecture
  - E.g. data structure information