CISC 839
Software Engineering of Usable Computing Systems

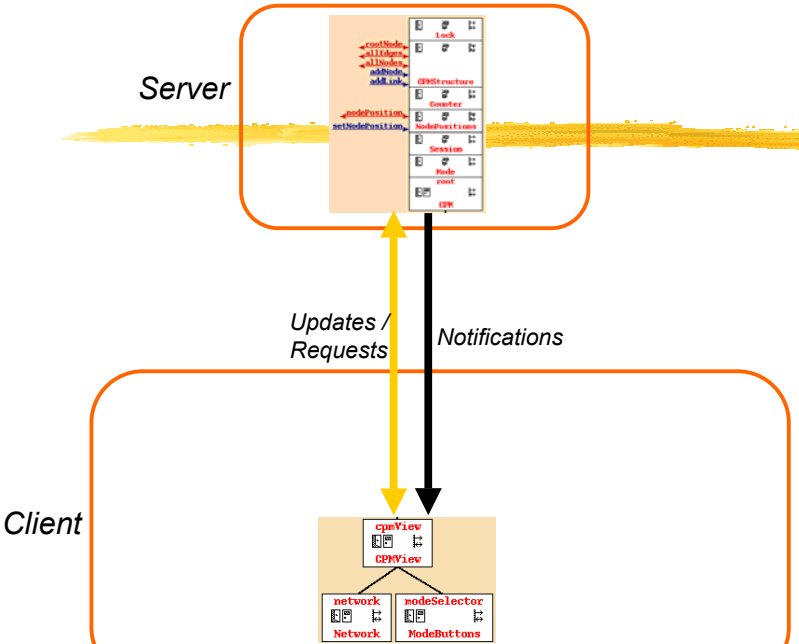
T.C. Nicholas Graham
http://www.cs.queensu.ca/~graham/cisc836

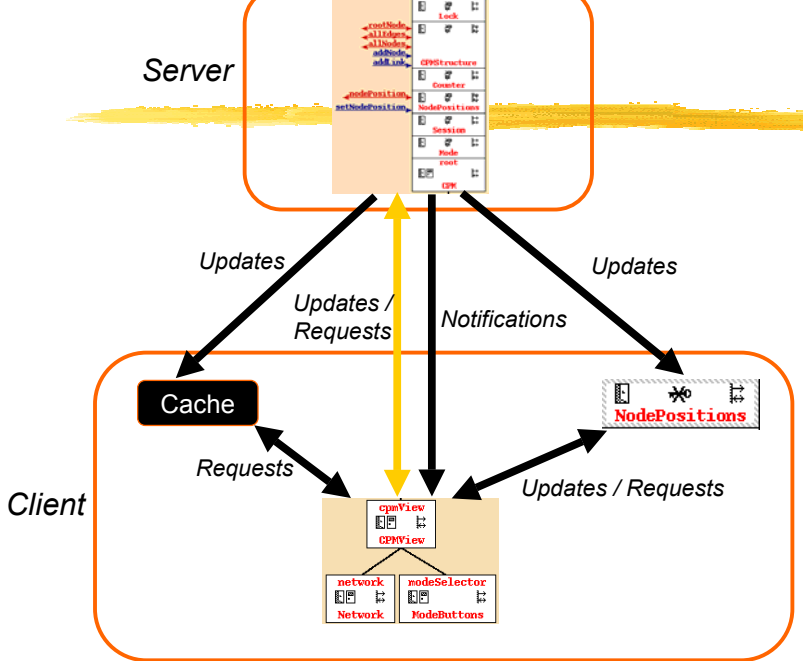
Outline

- Why is developing groupware hard?
- Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- Timings

Strategies for Cheating Latency

- Sacrifice group response time
- Reduce granularity of group actions
- Take turns
- Risk roll-backs
- Buffer
- Spend CPU, memory, bandwidth





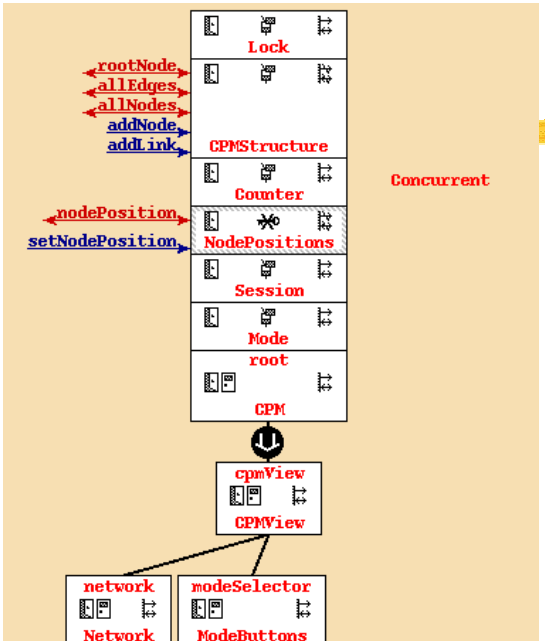
Space of Implementation Techniques

- Concurrency control: *none, optimistic, pessimistic*
- Topology: *tree, star, mesh, point*
- Replication: *centralized, partially replicated, replicated*
- Cache activity: *passive, active*
- Cache location: *standard, mirror*
- Network delivery: *lossless, lossy*
- Network order: *FIFO, any*

What Strategy is Best?

Depends on:

- Application characteristics
  - Symmetry, roles, synchronicity
- Hardware characteristics
  - How cheap is bandwidth, latency, CPU, memory?

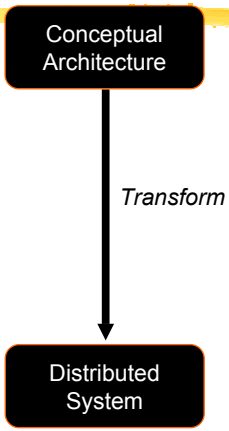


Annotations for Distributed Implementation

- Client / Server Split:
- Replication
- Concurrency Control

Concurrent

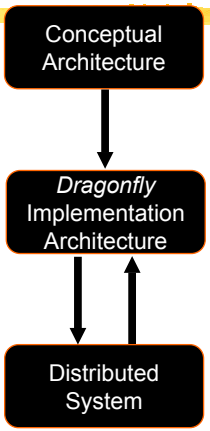
Serialized



Hide issues of:
- **Concurrency control**
- **Distribution**
- **Networking**

E.g. Clock, PAC*, C2, ALV

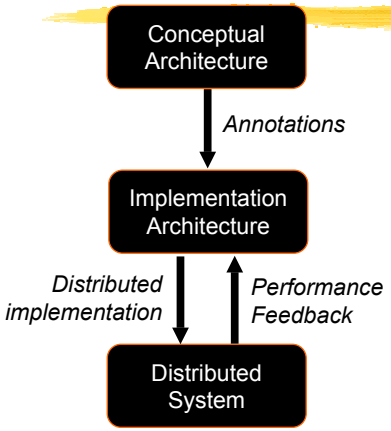
Choose:
- **Centralized or Replicated?**
- **Pessimistic or Optimistic?**
- **Caching?**
- **Hybrid?**



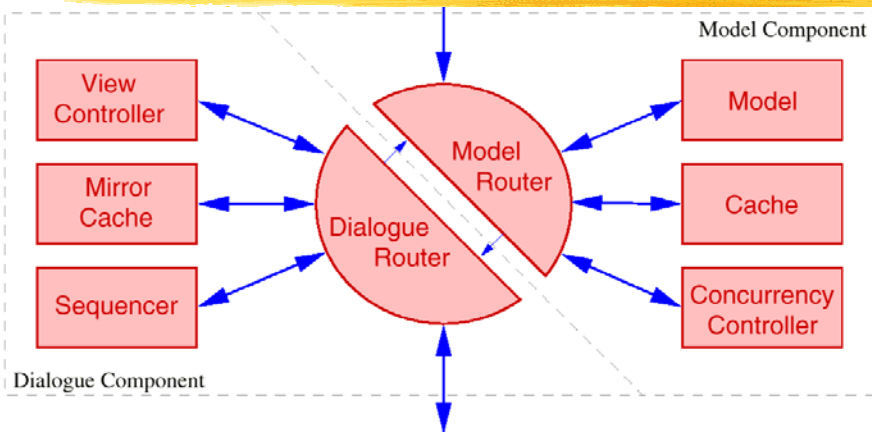
Developers concentrate of functionality of groupware application, not distributed implementation

Encapsulates design choices for:
- **Replication**
- **Caching**
- **Concurrency Control**
- **Client-Server Split**

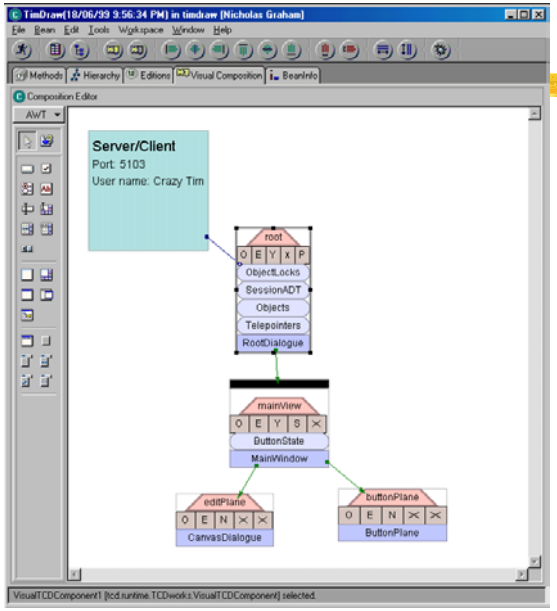
- **Hybrid implementation**
- **Dynamic reconfiguration**

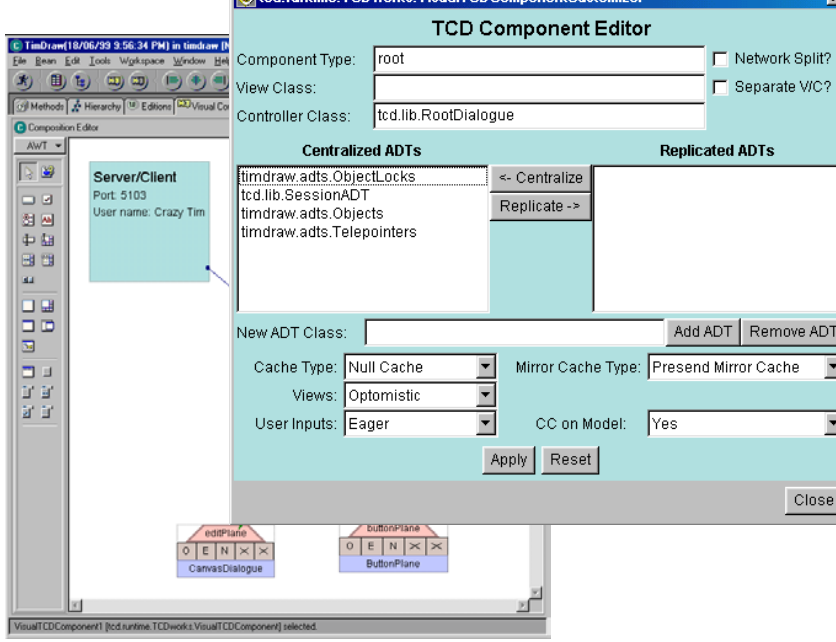


The Dragonfly Component



[Anderson et al 00]



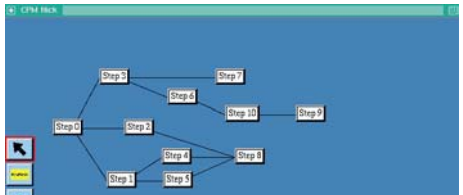


Outline

- Why is developing groupware hard?
- Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- Timings

Timing Clock and Dragonfly

	Local Area	Wide Area
Naïve Implementation	1,250 ms	24,000 ms
Presend Caching	130 ms	250 ms
Eager Concurrency Ctl	123 ms	190 ms
Replication	89 ms	86 ms



Response time where:

- *Server: LAN: UltraSparc 1, York U.; WAN: 100 MHz R4000 SGI Indy, Queen's University*
- *Client: UltraSparc 1, York U.*
- *Network Latency: LAN: 1 ms; WAN: 11 ms*

Conclusions

- Large gap between design-level architecture and distributed system
- Dragonfly acts as layer between, allowing experimentation with implementation strategies
- Programmers can give high-level guidance for implementation

User Interface Plasticity

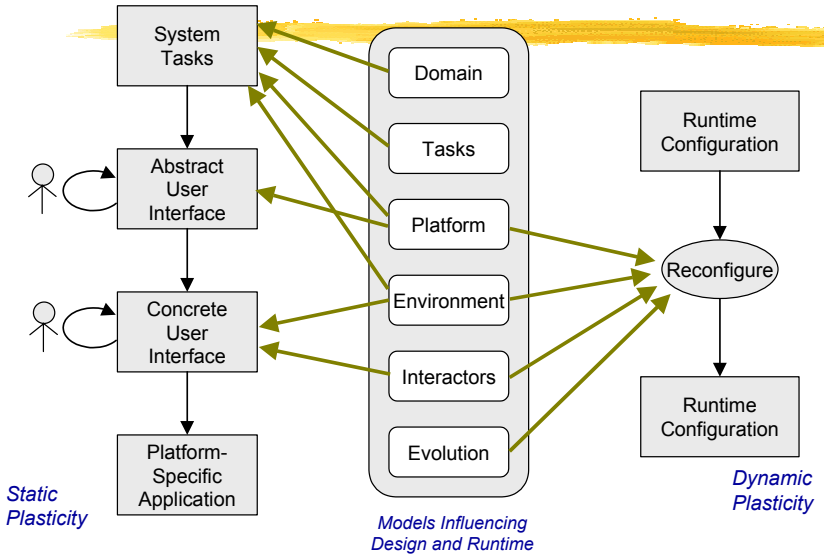
Plasticity in User Interfaces

- Proliferation of device types has led to difficulty in producing, maintaining different versions of products
- Platform versions differ in
  - User interface
  - Functionality
  - Development techniques

Version Problem

- Therefore, each version requires its own
  - system task model
  - UI design
  - Architecture
  - Evaluation
    - Usability testing, heuristic evaluation, etc.
- Maintaining consistency between versions
- Product branding

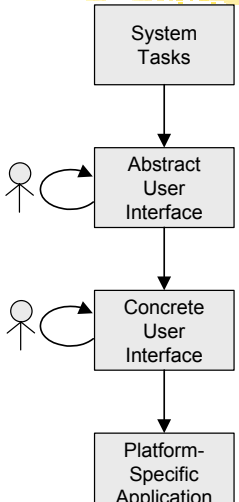
Plasticity in User Interfaces



Solutions

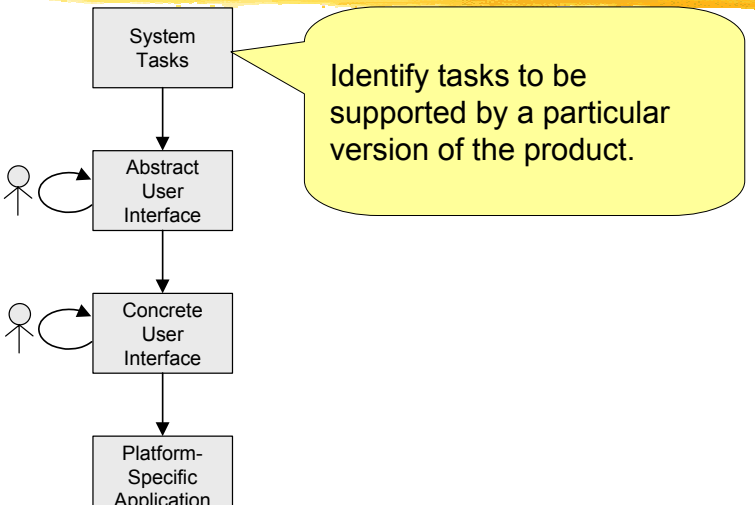
- Still a research area
- Static plasticity
  - Generate some portions of UI from higher-level models
  - E.g. from task model (Adept, Trident, Teallach)
  - E.g., from abstract UI description (XML/XSL, WML)
- Dynamic plasticity
  - System uses runtime models to adjust its behaviour

Static Plasticity

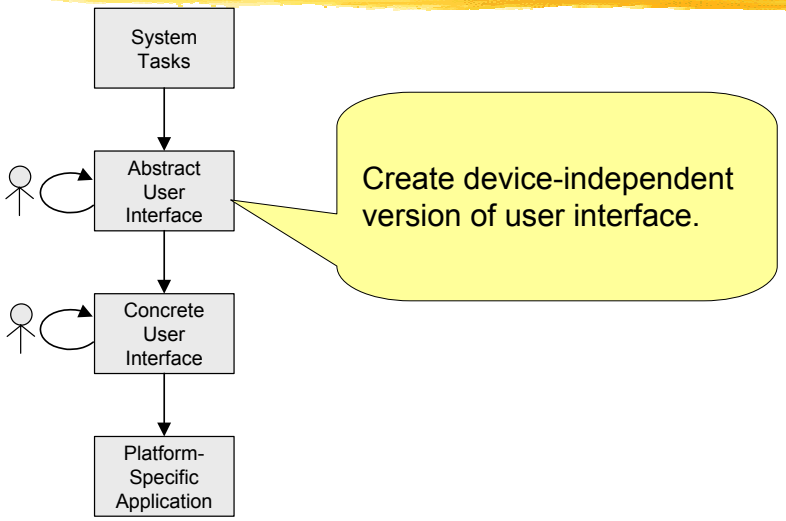


- Process for creating device-specific versions of application
- Trade off
  - Development time
  - Version consistency
  - Quality of individual versions
- Technologies
  - WML (E.g., Sympatico/Lycos)
  - XML/XSL/UIML (E.g., 724 Solutions)
  - ATS (Design Recovery Inc)

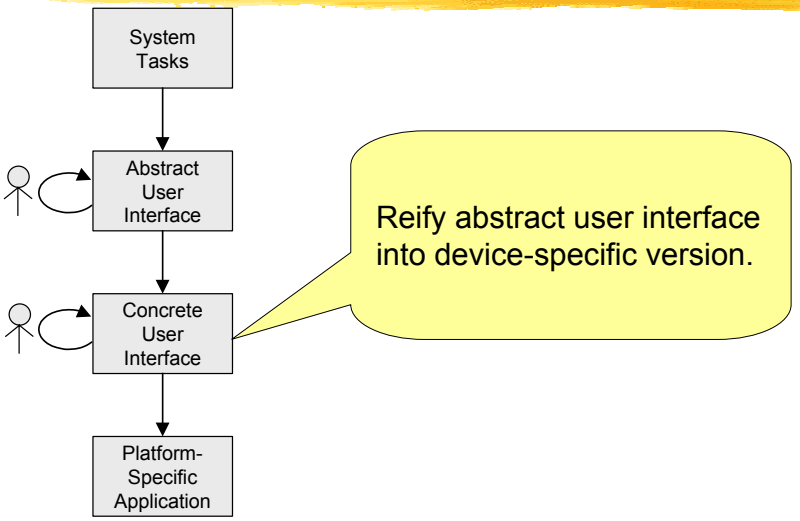
Static Plasticity



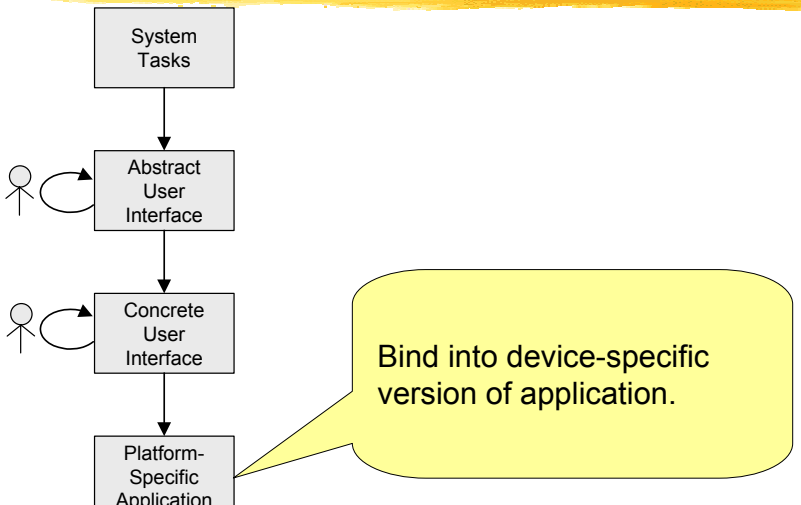
Static Plasticity



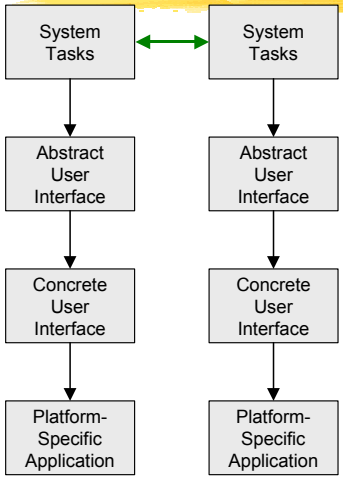
Static Plasticity



Static Plasticity

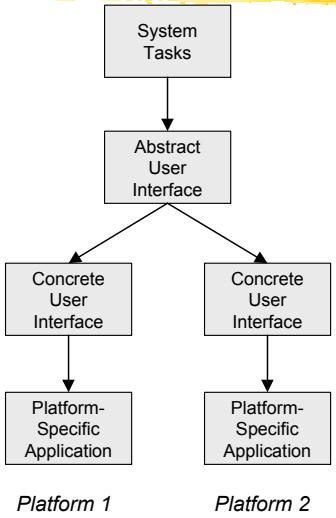


Static Plasticity



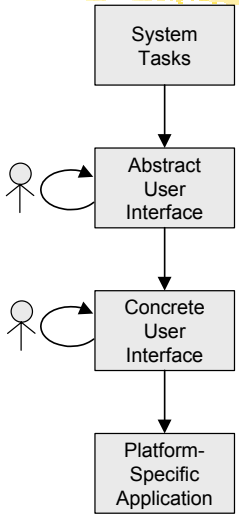
- Choice of version split point influences
  - Development cost
  - Impact of changes
  - Consistency
  - Usability of different versions

Static Plasticity



- Choice of version split point influences
  - Development cost
  - Impact of changes
  - Consistency
  - Usability of different versions

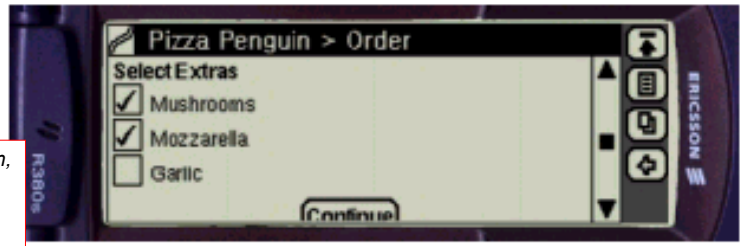
Example: WML



- Wireless Markup Language (WML) used to abstract differences between small devices
  - Usually Mobile Phones
- Tagged language similar to HTML
- Abstract differences in display size, input mechanisms

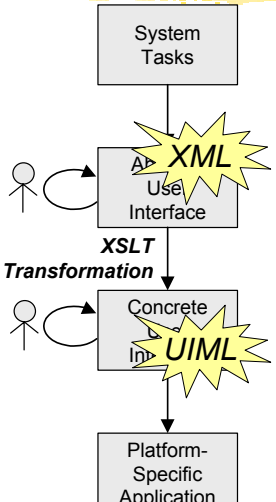
AUI's in WML

```
<p>
   <b>Select Extras</b>
   <select multiple="true">
      <option>Mushrooms</option>
      <option>Mozzarella</option>
      <option>Garlic</option>
   </select>
</p>
```



Source: Ericsson, Mobile Phone R380 Design Guidelines for

Example: XML/XSL/UIML



- XML used to specify abstract version of user interface
- Automated translation to concrete user interface
- Translation rules expressed using XSLT
- Concrete user interfaces expressed in (e.g.) UIML

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

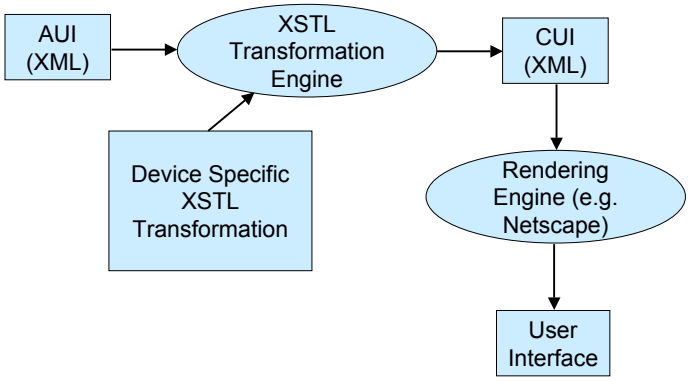
- What tags are legal
- How tags are combined
  - A book must have a title and 1 or more authors
  - A set of books consists of 0 or more books
  - A book has an ISBN attribute (unique number for books), which is character text (CDATA), with a default value of "0"


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


XML/XSL

- Specify content (AUI) using XML
- Use XSLT to transform to CUI
  - eXtensible Stylesheet Language Transform
  - Similar idea to TXL

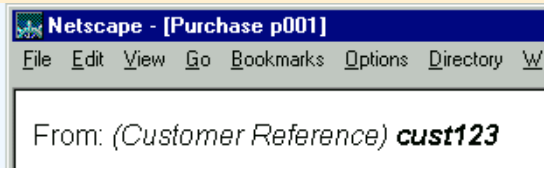
XML/XSLT



Example

```
01  <?xml version="1.0"?>
02  <purchase id="p001">
03    <customer db="cust123"/>
04    <product db="prod345">
05      <amount>23.45</amount>
06    </product>
07  </purchase>
```

XML Description of a customer record



Source: http://www.xml.com/pub/2000/08/holman/s1.html

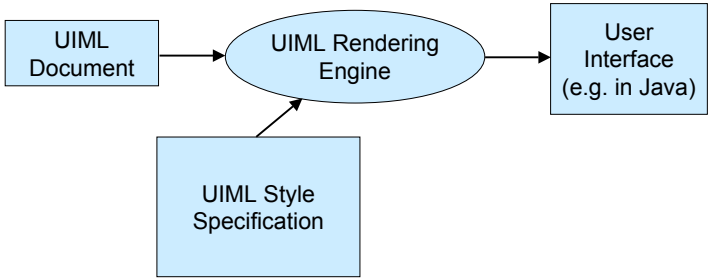
XSLT Transformation

```
01  <xsl:template match="customer">
02    <fo:block space-before.optimum="20pt" font-size="20pt">
03      <xsl:text>From: </xsl:text>
04      <fo:inline-sequence font-style="italic">
05        <xsl:text>(Customer Reference) </xsl:text>
06        <fo:inline-sequence font-weight="bold">
07          <xsl:value-of select="@db"/>
08        </fo:inline-sequence></fo:inline-sequence></fo:block>
09  </xsl:template>
```

UIML

- User Interface Markup Language
  - Specific XML dialect to express user interfaces
  - Normally combined with style-sheets to provide AUI⇒CUI transformation

UIML




```
<UIML>
<HEAD>
  <AUTHOR>Stephen Williams</AUTHOR>
  <DATE>December 3, 1997</DATE>
  <VERSION>1.0</VERSION>
</HEAD>
<APP CLASS="App" NAME="DialogApp">
  <GROUP CLASS="Dialog" NAME="PrintFinishedDialog">
    <ELEM CLASS="DialogMessage" NAME="PrintFinishedMsg"/>
    <ELEM CLASS="DialogButton" NAME="OKButton"/>
  </GROUP>
</APP>
<DEFINE NAME="OKButton">
  <PROPERTIES>
  <ACTION
     VALUE="DialogApp.EXISTS=false"
     TRIGGER="Selected"
  />
  </PROPERTIES>
</DEFINE>
</UIML>
```

Example: A dialogue box


```
APP.App{
   +TOOLKIT:          jfc;
   +RENDERING-PREFIX:   java.awt;
}
GROUP.Dialog {
   +RENDERING: Frame;  /* This is an AWT frame */
   +LAYOUT:    BorderLayout;
   +FONT_NAME: "sanserif";
   +FONT_SIZE: "11";
   SIZE:       "150,105";
   +CONTENT:   "Error: No Label";
   +BACKGROUND:"lightgray";
}

ELEM.DialogMessage{
   RENDERING:   "Label";
   FONT-STYLE:  "bold";
   ALIGNMENT:   "Center";
}

ELEM.DialogButton{
   RENDERING:   "Button";
   FONT-STYLE:  "Plain";
   ALIGNMENT:   "South";
   SIZE:        "80,25";
}
```

Example: A dialogue box


```
APP.App{
   +TOOLKIT:          jfc;
   +RENDERING-PREFIX:   java.awt;
}
GROUP.Dialog {
   +RENDERING: Frame;  /* This is an AWT frame */
   +LAYOUT:    BorderLayout;
   +FONT_NAME: "sanserif";
   +FONT_SIZE: "11";
   SIZE:       "150,105";
   +CONTENT:   "Error: No Label";
   +BACKGROUND:"lightgray";
}

ELEM.DialogMessage{
   RENDERING:   "Label";
   FONT-STYLE:  "bold";
   ALIGNMENT:   "Center";
}

ELEM.DialogButton{
   RENDERING:   "Button";
   FONT-STYLE:  "Plain";
   ALIGNMENT:   "South";
   SIZE:        "80,25";
}
```