Hypertext transfer family of protocols (HTTP, HTTPS, SOAP)

CSE 870 Miniproject on Frameworks
Advanced Software Engineering
Contact: Dr. B. Cheng, chengb at cse dot msu dot edu

Matt Gerber
Adithya Krishnamurthy
Agenda

- Overview of Protocols
- Overview of Frameworks
- Class diagrams
- Design pattern used
- State diagrams
- Demo
- Conclusions
Overview of Protocols - HTTP

- Request/Response protocol: Client or User Agent (UA) initiates a request and a server responds.

- HTTP Message elements
  - Message Types: Request or Response
  - Message Headers: field-name "." [ field-value ]
    Example: Content-Type: text/html
  - Message Body: payload associated with request or response.
  - Message Length: Length of the message body.
  - General Header Fields: Cache-Control, Date, Pragma, Transfer-Encoding

- Request
  - Request Line: Method Request-URI HTTP-Version CRLF
    Example: GET http://www.msu.edu/index.html HTTP/1.1

- Response
  - Status Line: HTTP-Version Status-Code Reason-Phrase CRLF
    Example: HTTP/1.1 206 Partial content
Overview of Protocols - HTTPS

• Secure message-oriented protocol that works in conjunction with HTTP
• Secure wrapper around unsecure HTTP message - HTTP over SSL
• HTTPS Message elements
  – Same as those of HTTP except the range of headers is different and the bodies are cryptographically enhanced.
• Request
  – Request Line: Secure * Secure-HTTP/1.4
• Response
  – Status Line: HTTPS-Version  Status-Code Reason-Phrase CRLF
    Example: Secure-HTTP/1.4 200 OK
• Message Authenticity Code (MAC)
  – Used to verify message integrity and sender authenticity
  – MAC = hex(H(Message||[<time>]||<shared key>))
Overview of Protocols - SOAP

- SOAP stands for **Simple Object Access Protocol**
- XML based protocol to let applications exchange information over HTTP
- SOAP is platform and language independent
- SOAP Message elements:
  - Envelope element:
    `<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"`
  - Header (Optional): `<soap:Header>… </soap:Header>`
  - Body:
    `<soap:Body>
     <x>5</x>
    </soap:Body>`
• Overview of Protocols
• Overview of Frameworks
• Class diagrams
• Design pattern used
• State diagrams
• Demo
• Conclusions
Overview of Frameworks

• Inheritance

```java
class Fruit {
    //...
}
class Apple extends Fruit {
    //...
}
```

• Composition

```java
class Fruit {
    //...
}
class Apple {
    private Fruit fruit = new Fruit();
    //...
}
```
Overview of Frameworks

- **Whitebox Framework**
  - Only abstract classes in framework with no implementation
  - Lays down the control flow and structure of the system (Lowest Common Denominator approach)
  - Implementation of framework based on inheritance
  - Most flexible but hardest to understand and implement
Overview of Frameworks

- **Graybox Framework**
  - Combination of abstract and concrete classes in framework
  - Some common implementation moved back to framework classes
  - Implementation of framework based on inheritance and composition
  - Less flexible but easier to understand and implement than Whitebox
Overview of Frameworks

- **Blackbox Framework**
  - Mostly concrete classes in Framework
  - Most of common implementation moved back to framework classes
  - Implementation of framework based on composition
  - Least flexible but easiest for clients to understand and implement.
Agenda

• Overview of Protocols
• Overview of Frameworks
• Class diagrams
• Design pattern used
• State diagrams
• Demo
• Conclusions
Whitebox Class diagram - Framework
Blackbox Class diagram - Implementation

```
Class
  ➔ Form

Fields

Methods
  - addBtn_Click_1(object sender, EventArgs e) : void
  - Dispose(bool disposing) : void
  - dispResourceBtn_Click(object sender, EventArgs e) : void
  - divideBtn_Click(object sender, EventArgs e) : void
  - InitializeComponent() : void
  - multBtn_Click(object sender, EventArgs e) : void
  - subtractBtn_Click(object sender, EventArgs e) : void
  - testForm()
  - urlBox_TextChanged(object sender, EventArgs e) : void
```
Agenda

- Overview of Protocols
- Overview of Frameworks
- Class diagrams
- Design pattern used
- State diagrams
- Demo
- Conclusions
Design pattern used

• **Template Method**
  – Design pattern that defines the program skeleton of an algorithm.
  – Algorithm itself is concrete but the methods that are called from it need not be.
  – Subclasses can override the methods called from the algorithm but not the algorithm itself.
  – Therefore the subclasses provide the concrete behavior that can vary and the general flow is determined by the template method.
Design pattern used

- **Template Method**
Agenda

• Overview of Protocols
• Overview of Frameworks
• Class diagrams
• Design pattern used
• State diagrams
• Demo
• Conclusions
State Diagrams

State diagram for HTTPConnection:

- Closed
- Opening
- Idle
- Reading
- Writing
State Diagrams

State Diagram for HTTPRequest object

Idle → Validating → Building → Sending

Parsing Response → Receiving
State Diagrams
State Diagrams

State diagram for SOAPRequest

- Idle
- Constructing request
- Creating SOAP Envelope
- Validating URL
- Passing response
- Establishing Connection
- Sending request
- Reading response
State Diagrams
Agenda

• Overview of Protocols
• Overview of Frameworks
• Class diagrams
• Design pattern used
• State diagrams
• Demo
• Conclusions
Agenda

- Overview of Protocols
- Overview of Frameworks
- Class diagrams
- Design pattern used
- State diagrams
- Demo
- Conclusions
Conclusions

• There are 3 different kinds of frameworks!
• Client perspective must be considered while designing frameworks.
• Tradeoff is between ease of use and flexibility.
• Using design patterns in frameworks is a good thing!
Thank You!

Questions?