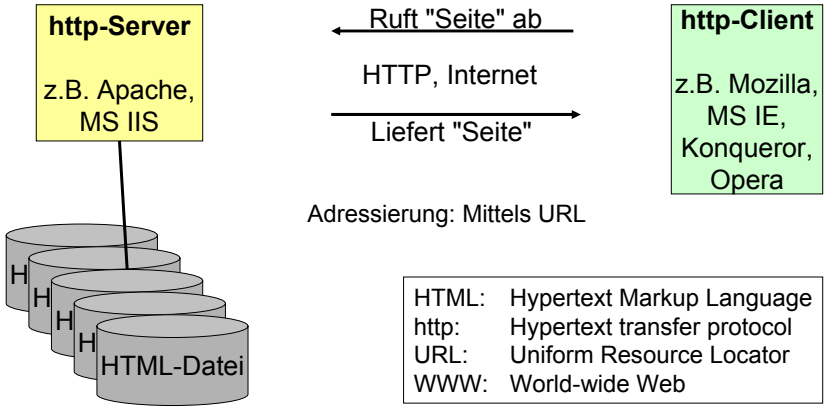
Teil 4:
Ruby-Extras & Ausblick

Ruby und das Internet

HTTP Client:	Net::HTTP
HTTP Server:	WEBrick, mod_ruby, FastCGI
CGI-Progr.:	CGI, CGI::Session
(HTML) templating:	erb, eruby
XML-Verarbeitung:	REXML
Web Services:	XMLRPC, SOAP
MVC-Framework:	Rails

Internet-Programmierung, Übersicht

Grundfunktion eines Zugriffs im WWW auf statische Inhalte:



HTML:	Hypertext Markup Language
http:	Hypertext transfer protocol
URL:	Uniform Resource Locator
WWW:	World-wide Web

Ruby? Programmierung generell? Hier (noch) nicht benötigt!

HTTP *client*: Programmatischer Zugriff auf Webseiten

```
#!/usr/bin/ruby
require 'net/http'

Net::HTTP.start('www.informatik.fh-wiesbaden.de') do |http|
  response = http.get('/~werntges/lv/ruby/pdf/ws2005/ruby-p-12.pdf')
  puts "Code = #{response.code}"
  puts "Message = #{response.message}"

  # HTTP Header-Felder ausgeben:
  response.each {|key, val| printf "%-14s = %-40.40s\n", key, val }

  # Inhalt der HTML-Seite (o.a.):

  case response['content-type']
  when 'application/pdf'
    puts "Dumping a PDF file..."
    File.open("./dump-it-here.pdf","w") do |file|
      file.binmode
      file.syswrite response.body
    end
  else
    p response.body[00, 55]
  end
end
```

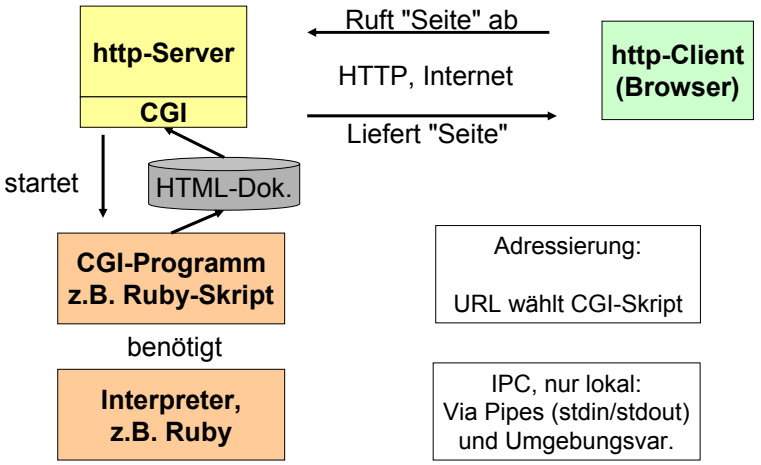
Demo, falls Netzwerkzugang

Internet-Programmierung: CGI

- **CGI**: Delegation an spezielle Programme
  - Schaffung des "Common Gateway Interface (CGI)"
  - http-Server startet einen separaten (lokalen) Prozess
  - IPC über stdin, stdout sowie mittels einer Reihe von Umgebungsvariablen.
  - Vollwertige HTML-Dokumente mit http-Headern unter Beachtung spezieller "escaping"-Regeln sind zu beachten.
- Vorteil: Flexibilität
- Nachteil: Prozess-Overhead
  - Jede Anfrage startet einen eigenen CGI-Prozess!
- Werkzeuge:
  - Mit CGI gewann Perl enorm an Bedeutung
  - Ruby: Übernahme + Weiterentwicklung von Perl's Modul "CGI"
  - Auswege bei Performance-Engpässen: Interpreter in http-Server integrieren (mod_perl, mod_ruby), oder speicherresident (fast-cgi).

Internet-Programmierung: CGI

Seitenaufbau mit CGI



CGI-Programmierung (1)

- Ein elementares CGI-Script: „cgi_simple.rb“

```
#!/usr/bin/ruby
print "Content-type: text/html\r\n\r\n"
print "<html><body>Hello, World! It's
    #{Time.now}</body><html>"
```

  - Datei in …/cgi-bin kopieren, Ausführungsrechte beachten
  - URL z.B.: „http://localhost:8080/cgi-bin/cgi_simple.rb“

- Variante, falls der Web Server keinen *Header* erzeugt:

```
#!/usr/bin/ruby
print "HTTP/1.0 200 OK\r\n"
print "Content-type: text/html\r\n\r\n"
print "<html><body>Hello, World! It's
    #{Time.now}</body><html>"
```


CGI-Programmierung (2)

- Die Klasse „CGI“ des Moduls „cgi“: Formular erzeugen

```
#!/usr/bin/ruby
require 'cgi'
cgi = CGI.new("html3")
# Methoden zur Erzeugung von HTML-Elementen einfügen:
cgi.out {
  cgi.html {
    cgi.head { "\n"+cgi.title{"Kleiner Test"} } +
    cgi.body { "\n"+
      cgi.form("post","some_url") {"\n"+ cgi.hr +
        cgi.h1 { "Ein Formular: " } + "\n"+
        cgi.textarea("Texteingabe") +"\n"+
        cgi.br +
        cgi.submit
      }
    }
} }
```

CGI-Programmierung (3)

- Die Klasse „CGI“ des Moduls „cgi“: Formular auswerten

```
#!/usr/bin/ruby
require 'cgi'
cgi = CGI.new
# Zugriff auf Formularinhalte:
cgi.params    # {"Texteingabe" => ["meine Eingabe"]}
cgi.has_key?('Zahleneingabe')     # false
cgi.has_key?('Texteingabe')       # true
```

  - Zugriff über Methode „params“ – eine Art Hash von Arrays
  - Bem.: Arrays, weil Formularteile wie Checkbox-Gruppen auch mehrere Werte liefern können.

CGI-Programmierung (4)

- Mehr zur Klasse „CGI“
  - Die Klasse ist recht groß! Dokumentationsquelle:

```
http://www.ruby-doc.org/stdlib/libdoc/cgi/rdoc/
```

  - Weitere Highlights:

```
# Escaping:
puts CGI.escapeHTML("a < 100 && b > 200")
# a &lt; 100 &amp;&amp; b &gt; 200
# Ferner: unescapeHTML, (un)escapeElement (selektiv)
```

```
# Cookies (hier ohne Beispiel)
```

- *Session management* mit CGI::Session
  - Session: Eine Folge von Abrufen von HTML-Seiten einer *Website* durch denselben Anwender ist nicht per se erkennbar, denn HTTP arbeitet zustandslos.
  - Ruby: *Session key in cookie, session params in server-DB / file*

CGI-Programmierung (5): Session-Verwaltung

```
require 'cgi'
require 'cgi/session'

cgi = CGI.new("html3")
sess = CGI::Session.new( cgi, "session_key" => "rubyweb",
                              "prefix" => "web-session." )
if sess['lastaccess']
    msg = "You were last here #{sess['lastaccess']}."
else
    msg = "Looks like you haven't been here for a while"
end
count = (sess["accesscount"] || 0).to_i
count += 1
msg << "<p>Number of visits: #{count}</p>"
sess["accesscount"] = count
sess["lastaccess"] = Time.now.to_s
sess.close

cgi.out { cgi.html { cgi.body { msg } } }
```


Internet-Programmierung

- Active Server Pages (ASP) und Java Servlets:
  - Proprietäre Antworten auf CGI's Performanceproblem
  - ASP (nur MS IIS) bzw. Servlets erfüllen ähnliche Aufgaben wie CGI-Skripte, aber innerhalb des Prozessraums des http-Servers.
  - Der http-Server muss dazu (proprietär) erweitert werden, z.B. um einen VB-Interpreter oder eine Java VM aufzunehmen.
- Ruby:
  - Gleiches Konzept, proprietär mit reinen Ruby-Mitteln gelöst.
  - http-Server: WEBrick!

WEBrick – oder: Es muss nicht immer Apache sein

- WEBrick
  - Ein „*pure* Ruby“ HTTP-Server

```
#!/usr/bin/ruby
require 'webrick'
include WEBrick

s = HTTPServer.new(
  :Port          => 2000,
  :DocumentRoot => File.join(Dir.pwd, "/html")
)

trap("INT") { s.shutdown }

s.start
```

Demo!

WEBrick – oder: Wozu noch Java Servlets?

- WEBrick als Ruby Servlet-Container à la Apache Tomcat u.a.

```
#!/usr/bin/ruby
require 'webrick'; include WEBrick

s = HTTPServer.new( :Port => 2000 )

class HelloServlet < HTTPServlet::AbstractServlet
  def do_GET(req, res)
    res['Content-Type'] = "text/html"
    res.body = %{
      <html><body>
        <h2>Hello</h2>
        <p>You're calling from a #{req['User-Agent']}</p>
        <p>I see parameters: #{req.query.keys.join(', ')}</p>
      </body></html>
    }
  end
end
s.mount("/hello", HelloServlet)

trap("INT"){ s.shutdown }
s.start
```

Demo!

Internet-Programmierung

- Weitere Auswege aus CGI'S Performance-Engpässen:
  - Interpreter in http-Server integrieren (mod_perl, mod_ruby), oder
  - speicherresident arbeiten (fast-cgi).

- Hinweise
  - FastCGI wird in "Programming Ruby" nicht behandelt, wohl aber von "Ruby Developer's Guide"
  - FastCGI ist generell eine sehr interessante Alternative, auch wenn sie eher selten erwähnt wird.
  - Besonders interessant z.B. bei zeitaufwändigen, aber nur einmal notwendigen Schritten wie dem Anmelden bei einer Datenbank.
  - Weitere Informationen unter http://www.fastcgi.com

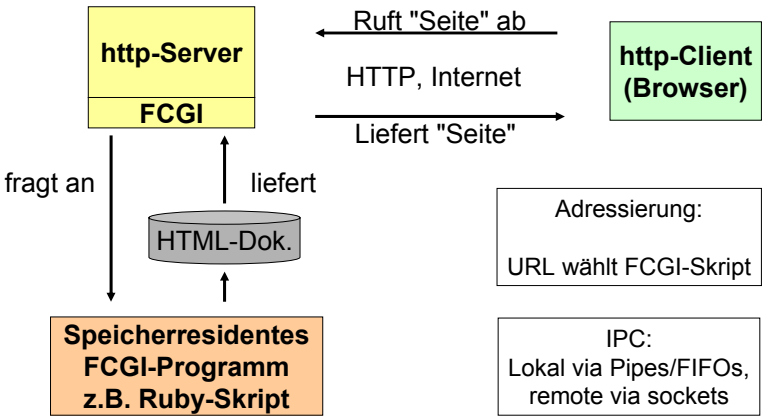
Beschleunigungstechniken auf Serverseite

- **mod_perl? mod_ruby!**
  - Der Ruby-Interpreter wird Teil des Webserver-Prozesses (insb. Apache, „httpd“)
  - Wirkung: Ruby-Interpreter muss nicht jedes Mal geladen werden, CGI-Scripte laufen viel schneller
  - Nachteile: Ein Prozess für viele Anwendungen – Konflikte bei schlecht programmierten Modulen oder memory leaks. U.U. mehrere httpd-Prozesse für eine session → Datenkonsistenz??

- **FastCGI**
  - Auch Ruby-Scripte können mittels FastCGI beschleunigt werden.
  - Konzept: CGI-Proxy in httpd, speicherresidente fcgi-Prozesse als „Server im Server“
  - Wirkung: Vermeidung vieler Nachteile von mod_xxx

Internet-Programmierung: FastCGI

Seitenaufbau mit FastCGI

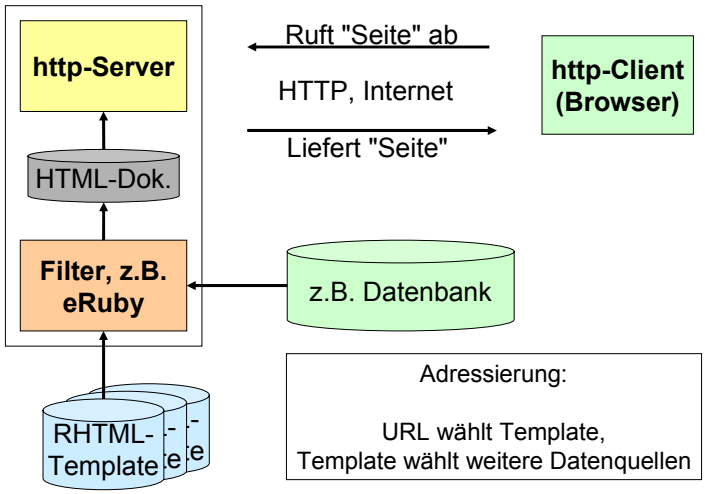


Internet-Programmierung

- SSI (Server-Side Includes)
  - Dynamische Erzeugung spezieller Seitenteile direkt durch den http-Server. Beispiel: Datum, Copyright-Vermerke
  - Ansatz: Spezielle HTML-Kommentare zur Steuerung
  - Nachteil: Proprietär (typisch für Apache) und wenig flexibel
- Der Template-Ansatz
  - HTML-Dateien mit "Platzhaltern", die im Moment des Abrufs zu ersetzen sind mit aktuellen Inhalten, z.B. aus Datenbanken
  - Naheliegend z.B. für tabellarische Ergebnisse
  - Verallgemeinerung hin zur dynamischen Generierung ganzer Seiten ist fließend
  - Typisch für PHP!
  - Ruby-Alternative: eRuby (vgl. "Programming Ruby", ch. 14, p. 150f)
  - Kommerzielles Beispiel: SAP ITS, mit "AGate" und "WGate"

Internet-Programmierung: *Templates*

Prinzip der Seitengestaltung mit *templates*



Embedded Ruby – oder: Wozu PHP, ASP, JSP?

Template-Technik mit **erb** oder **eruby**

- Situation
  - Sie möchten eine i.w. statische HTML-Seite an einigen Stellen dynamisch mit Inhalten befüllen, etwa aus einer DB.
- Lösungsansatz
  - Ruby-Code eingebettet in statische HTML-Texte → *.rhtml-Seiten
  - Präprozessor übersetzt/expandiert: *.rhtml → *.html
  - Web-Server: Ruft Präprozessor auf, reicht dessen Output weiter
- Einbettung von Ruby-Code: Regeln

```
<% ruby code %>      Code zw. <% … %> ausführen
<%= ruby expr %>     Auswertung des Ruby-Ausdrucks,
                     Ergebnis ersetzt <%= … %>
<%# ruby expr %>     Auskommentiert (gut zum Testen)
% ruby code          Eine ganze Zeile Ruby-Code
```

- Tools:
  - erb Standalone-Tools, ein Filter
  - eruby Performantere Alternative, wird in Apache integriert

Embedded Ruby – oder: Wozu PHP, ASP, JSP?

- Template-Technik mit **erb** oder **eruby** und DBI**:** booklist.rhtml

```
% require "dbi"
% dbh = DBI.connect("dbi:Mysql:rubydemo:localhost",
    "werntges", "rubypw")
% sth = dbh.execute("SELECT title, author1, edition, pubyear
    FROM morebooks")
<html><body><h1>Bücherliste</h1>
<p><table border="1">
<tr><th>Titel</th><th>Autor</th><th>Ed</th><th>Jahr</th></tr>
% sth.each do |row|
  <tr>
%   row.each do |field|
    <td><%= field.to_s%></td>
%   end
 </tr>
% end
</table></p>
</body></html>
% sth.finish
% dbh.disconnect if dbh
```


Ruby und XML? REXML!

- **XML im Web-Kontext?**
  - XHTML 1.1, XForms, …
  - SVG
  - MathML
  - SMIL
  - Stylesheets, XSLT

  Mehr dazu in der LV „XML-Technologie“, Liste I, 5.Semester

- **Ruby und XML?**
  - Es gibt verschiedene XML-Module für Ruby
  - Inzwischen mit Ruby verteilt: REXML
  - Beispiel: Siehe XML-Tech, Kap. „XPath“ und „APIs“

Ruby und Web Services?

- **Remote Procedure Calls mit XML-RPC**
  - In Ruby recht einfach mit modul „XMLRPC“
  - Sowohl Client- als auch Server-Teile
  - WEBrick kommt dabei recht gelegen…
- **SOAP und WSDL**
  - Ruby Modul „SOAP“
  - SOAP-Unterstützung im Framework „Rails“

Mehr dazu in meiner Version (!) der LV „Web-Basierte Anwendungen“, Liste V, 7. Semester

Größeres Projekt, wahlweise mit Ruby implementierbar!

Rails: Ein MVC-Framework für WBA auf Ruby-Basis

- **Einige Highlights von Rails**
  - Model – View – Controller- Pattern, beispielhaft konsequent umgesetzt
  - Hoch-effiziente Arbeitsweise dank vieler sinnvoller Konventionen
  - Einbindung unterschiedlicher Datenbanken mittels ORM
  - Drei Datenbanken: Entwicklung, Konsolidierung, Produktion
  - Skalierbare Lösungen: Session-Integrität auch bei Lastverteilung auf mehrere Applikations-Server
  - Unterstützung „agiler Prozesse“ (!)
- Zitate:
  - Siehe **www.rubyonrails.org/quotes**

Ruby: Ausblick

Weiterführendes Material
Die Zukunft von Ruby
Zum Schluss: Stilfragen

Weiterführendes Material

- RAA - Ruby Application Archive
  - Analog zu Perl's "CPAN"
  - Zentrale Sammelstelle für Ruby-Module
    http://raa.ruby-lang.org/
- RubyGarden
  - Diskussionsforen, Einführungsmaterial und vieles mehr rund um Ruby:
    www.rubygarden.org,
    www.rubygarden.org/ruby?HomePage
  - Besonders empfohlen: "Coding in Ruby
    www.rubygarden.org/ruby?CodingInRuby
  - Das wöchentliche Ruby-Programmierquiz: www.rubyquiz.com

Die Zukunft von Ruby

- Aktuell: Ruby **1.8.4**
  - Erschienen 24.12.2005
  - Nur noch 1.8 *maintenance updates* erwartet
- Zwischenschritt: Ruby 1.9.x
  - Experimentelle Version, nur für Tester & Entwickler
  - M17N (Unterstützung zahlreicher nationaler Zeichensätze), neue Regexp-Engine "Oniguruma", "generational" GC
- Großes Ziel: Ruby 2.0 "Rite"
  - Noch keine Zeitaussage, Hinweise unter http://www.rubyist.net/~matz/slides/rc2003/
  - Einschließlich Bytecode-Engine & noch schnellerem GC
    - **yarv**: V 0.3.3 (2005-12-22) http://www.atdot.net/yarv/
  - Nicht vollständig abwärtskompatibel !

Schließlich...

- Tipps zum Wiederholen
  - Kapitel "Ruby Crystallized"!
  - Welche Kap. von "Programming Ruby" haben wir behandelt?
  - "Ruby in a Nutshell"
  - Betonung der Praktikumsübungen!
- Fragen, Anregungen, Kritik
  - Vorlesung
  - Praktikum

Stilfragen

Agile Softwareentwicklung

The Ruby Way ...
(Sammlung, wächst noch)

Agile Manifesto (und warum Ruby so gut dazu passt)

- Agile Prozesse und Ruby
  - *Very High Level Language* → Ermöglicht Konzentration auf das Problem, nicht auf die Technik/Sprache
  - „Simplicity--the art of maximizing the amount of work not done--is essential."
    → In Ruby nahezu ideal realisiert, Teil der Ruby-Kultur!
  - „Continuous attention to technical excellence and good design enhances agility."
    → Gutes Design verfügbarer Ruby-Klassen war stets wesentlich für den Erfolg von Ruby
  - „The best architectures, requirements, and designs emerge from self-organizing teams."
    → Grundlage der Open Source-Bewegung, insb. von Ruby erfolgreich demonstriert.

Agile Manifesto

- Große Unternehmen (Bsp: Siemens, BT) sind z.Z. dabei, Agile Prozesse für die Software-Entwicklung zu nutzen!
- Quelle: Siehe http://www.agilemanifesto.org
  - (Demo der Seite)

Stilfragen

- Entschuldigen Sie, dass der Brief so lang geworden ist, ich hatte keine Zeit für einen kürzeren.
  J. W. v. Goethe
- Vollkommenheit entsteht offenbar nicht dann, wenn man nichts mehr hinzuzufügen hat, sondern wenn man nichts mehr wegnehmen kann.
  Antoine de St. Exupéry
- In der Kürze liegt die Würze.
  (Volksmund)

Stilfragen

- Mache die Dinge so einfach wie möglich, aber nicht einfacher.

  A. Einstein

- *When you say something in a small language, it comes out big. When you say something in a big language, it comes out small*.

  L. Wall, über Perl

- *Don't write 200 lines of code when 10 will do*.

  H. Fulton

Stilfragen

- *A smart servant should do a complex task with a few short commands*.

  Y. Matsumoto ("Matz"),
  über Computer und deren Sprachen

- *A program should follow the "Law of Least Astonishment"*.

  G. James, in
  "The Tao of Programming"

- Die dümmsten Informatiker schreiben die dicksten Programme.

  (Volksmund :-))

Stilfragen

- *Ruby arose from the urge to create things that are useful and beautiful. Programs written in Ruby should spring from that same God-given source. That, to me, is the essence of the Ruby Way*.

  H. Fulton, in "The Ruby Way"