Infospaces

FH Offenburg

SS 2003

Parsen von HTML Informationen und Darstellung auf einer Webseite über einen ELVIN Server

Vorgelegt von:

Tobias Kiefer
Communication and Media Engineering
2. Semester
Matrikel Nr.: 164282

Andreas Brodmann
Communication and Media Engineering
2. Semester
Matrikel Nr.: 164267

Inhaltsverzeichnis

1. Vorwort 3

2. Grundidee 4

3. Allgemeine Informationen über ELVIN 7

4. Realisierung 8

4.1 Erzeugen der Informationen durch den "Producer" 8

4.2 Abrufen der Informationen durch den "Consumer" 9

5. Fazit 11

6. Flussdiagramme 12

7. Source Code 14

1. Vorwort

Da wir den Namen "Infospaces" noch nie zuvor in irgendeinem Zusammenhang gehört hatten, bot es sich für uns an, einmal in diese Vorlesung mit dem viel versprechenden Namen hineinzuhören.

Nachdem sich die ersten Zweifel gelegt hatten, ob sich ein Projekt mit einem so genannten ELVIN Server durchführen lässt, wurden die ersten Vorüberlegungen und die ersten Konzepte entworfen.
Dabei wurde festgelegt, dass die Umsetzung der Idee in der Programmiersprache "Python" durchgeführt werden sollte, damit somit auch die Möglichkeit bestand, eine weitere Programmiersprache kennen zu lernen.

Am Ende dieses Projekts kann man festhalten, dass es sehr viel Spaß gemacht hat, sich in ein neues Thema einzuarbeiten und das erlangte Wissen in Form von Source Code umzusetzen.

2. Grundidee

Das Projekt lässt sich grundsätzlich in drei Phasen unterteilen. Die erste Phase ist das Herausnehmen von Informationen aus einer Webseite, dann das Bereitstellen dieser Informationen im so genannten "Infospace", der von einem ELVIN Server zur Verfügung gestellt wird. Dieser Vorgang wird von einem "Producer", auf der "Producer Seite" durchgeführt.
In der dritten Phase entnimmt der "Consumer", auf der "Consumer Seite" angesiedelt, diese Informationen wieder aus dem Infospace. Dazu muss sich dieser Consumer auf eine ganz bestimmte Informationen in diesem Infospace abonnieren. Danach wird die entnommene Information in einem erzeugten HTML Dokument wieder dargestellt, welches von einer schon vorhandenen Webseite verlinkt ist.

Um nun ein wenig genauer auf die eigentliche Idee dieses Projekts einzugehen, muss man vorab noch kurz erwähnen, dass sich das Interesse dieses Projekts speziell auf Wetterdaten von finnischen Städten konzentriert. Im allgemeinen kann man die Art und Weise, wie solche Daten bereit gestellt werden und auch wieder abgerufen werden, auf jegliche Art von Webseiten anwenden.

Auf der "Consumer Seite" sollen die Wetterdaten wie z.B. die Temperatur und die Wetterbedingungen von verschiedenen finnischen Städten aus dem Infospace entnommen werden. Danach sollen die Wetterdaten in eine andere Webseite, die Informationen über Finnland enthält, eingebunden werden.

In der folgenden Abbildung 2.1 wird nun die System Topologie aufgezeigt, die diesem Projekt zu Grunde liegt.
Wie man erkennen kann, werden im Prinzip drei verschiedene Computer in Anspruch genommen. Computer 1 symbolisiert die Webseite, von der die Wetterdaten genommen werden. Auf dem 2. Computer wird der ELVIN Server und das "Producer Skript" betrieben. Die komplette "Producer Seite" wird durch einen 3. Computer dargestellt, auf dem ebenfalls ein Python Skript läuft und die Webseite abgelegt ist.

System Topologie

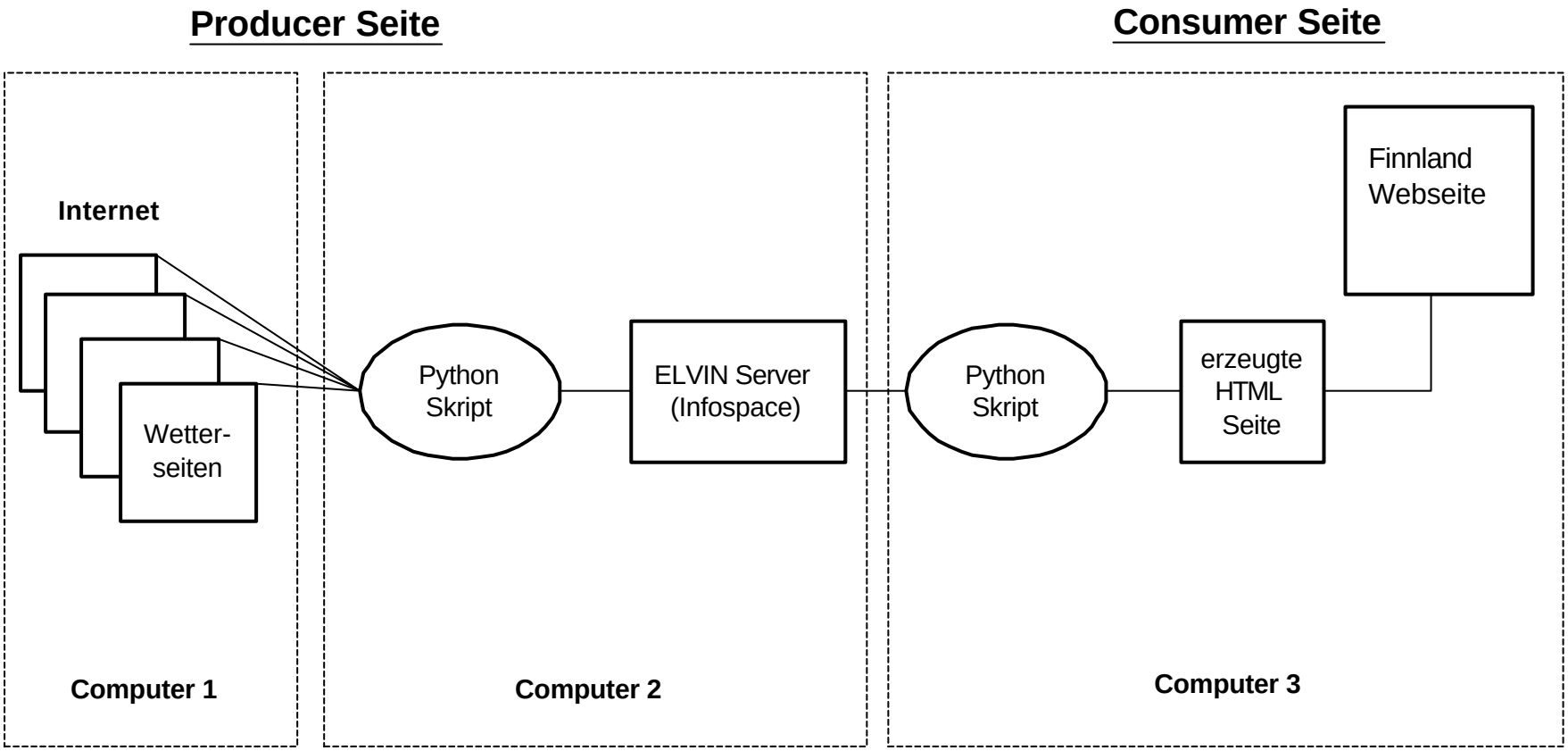


Abbildung 2.1: System Topologie

In der folgenden Liste werde die einzelnen finnischen Städte aufgezählt, von denen die Wetterdaten in das zu erstellende HTML Dokument übernommen werden sollen:

- Helsinki – Vanta
- Ivalo
- Joensuu
- Jyväskylä
- Lappeenranta
- Oulu
- Rovaniemi
- Savonlinna
- Tampere
- Turku

Folgende Informationen sollen von den einzelnen finnischen Städten übernommen werden:

- Name der Stadt
- Temperatur
- Luftfeuchtigkeit
- Sonnenaufgang
- Sonnenuntergang
- Datum des letzten Updates

3. Allgemeine Informationen über ELVIN

ELVIN ist ein Netzwerk Kommunikations-Produkt, um ELVIN Nachrichten an mehrere "Clients" zu verschicken, die vollständig auf den Inhalt einer jeden Nachricht basiert sind.

ELVIN bietet eine sehr einfache, flexible und sichere Kommunikations-Infrastruktur an. Es hat eine hohe Leistung, kurze Verzögerungen und ist sehr skalierbar. Es unterstützt dynamische Definitionen von Nachrichten und Subskriptionen und ist weltweit standardisiert.

Der ELVIN Server ist in der Programmiersprache C geschrieben und läuft auf den meisten UNIX Plattformen, aber auch auf Windows NT/2000 Systemen. ELVIN Clients können in einer Vielzahl von verschiedenen Programmiersprachen geschrieben werden, wie z.B. in C, C++, Java, Python, Perl, Palm und Emacs Lisp.

ELVIN benutzt eine Client-Server Architektur um Nachrichten zu übermitteln. Die Clients bauen "Sessions" mit ELVIN Servern auf und sind danach in der Lage, eine Nachricht an den Server zu schicken, um sich dort zu registrieren und anschließend Nachrichten zu erhalten, die von anderen Clients gesendet wurden. Clients können sich sowohl als "Producer", aber auch als "Consumer" innerhalb einer "Session" verhalten.

Die Aufgaben eines ELVIN Servers sind es, die Client-Verbindungen zu überwachen und darüber hinaus die Nachrichten vom "Producer" zum "Consumer" zu routen. Der "Consumer" zeigt Interesse an einer Nachricht, in dem er sich beim Server subskribiert. Diese Subskriptionen beinhalten verschieden Kriterien, die sich auf den Inhalt einer Nachricht beziehen. Wenn der Server eine Nachricht erhält, überprüft er den Inhalt und vergleicht ihn mit den Subskriptionen und leitet die Nachricht an jeden angemeldeten "Client", auf den diese Nachricht zu trifft.

Der Hauptvorteil von ELVIN ist, dass die Nachrichten nicht nur an einen einzigen Empfänger adressiert sind, wie z.B. bei einem Brief oder bei einer speziellen Gruppe, wie in "News Groups". Dabei können die "Producer" nicht selbst entscheiden, wer im Endeffekt die Nachrichten erhält.

4. Realisierung

Bei der Realisierung dieses Projekts gab es zu Beginn verschiedene Schwierigkeiten. Zum einen war es die unbekannte Programmiersprache, zum anderen, sich in die Theorie der ELVIN Architektur einzuarbeiten.

Mit der ELVIN Architektur, genauer gesagt mit dem ELVIN Server, wurden erste Testläufe durchgeführt, um das grundsätzliche Prinzip zu verstehen. Dabei wurde der ELVIN Server lokal auf einem PC installiert, um dann auf dem gleichen PC und auf einem weiteren PC, ein Tickertape zu installieren. Dieses Tickertape wurde dazu benutzt, um erste Nachrichten von einem Rechner zum anderen zu schicken.

Mit diesen Vorkenntnissen wurde dann das eigentliche Projekt, wie in Kapitel 2 schon näher erklärt wurde, angegangen.

4.1 Erzeugen der Informationen durch den "Producer"

Der "Producer" ist verantwortlich für die Erzeugung der Nachrichten, die in den Infospace gestellt werden. Diese Nachrichten bestehen im vorliegenden Fall aus den Wetterinformationen für jeweils eine Stadt pro Nachricht. Die Webseite www.1uptravel.com bietet diese Daten für Städte weltweit an. Der "Producer" lädt zunächst die Seite mit den Wetterdaten einer Stadt. Die URL dazu befindet sich in einer Liste. Beim Parsen des Quellcodes werden die relevanten Daten extrahiert und anschließend als ELVIN Nachricht an einen ELVIN Server verschickt. Nun wird die nächste URL geladen und der Inhalt der Seite geparst und als ELVIN Nachricht versendet. Dieser Vorgang wiederholt sich nun bis die Liste mit den URLs leer ist. Nach einer Pause von 15 Minuten werden die URLs wieder verarbeitet.

Parsen

Das Parsen des HTML Codes vereinfacht sich enorm durch den Einsatz der SGML Parser Bibliothek.

Der SGML Parser verarbeitet ASCII Text nicht Zeichenweise oder Zeilenweise, sondern strukturiert d.h. es werden die Taginhalte, Kommentare usw. als Block verarbeitet.

Es wurden die Funktionen "handle_data" und "handle_comment" erweitert. Die Funktion "handle_data" wird immer dann ausgeführt, wenn der Parser auf eine Textstelle, d.h. der Text unmittelbar zwischen zwei eckigen Klammern <..>foobar<..> steht. Die Funktion "handle_comment" hingegen wird ausgeführt wenn der Parser auf einen HTML Kommentar trifft: <!-- foobar -->. In der Variable text steht jeweils der Text bzw. Kommentar und kann verarbeitet werden.

Extrahierung der Wetterdaten

Es werden Referenzmarken, d.h. bekannte Stellen im Quelltext gesucht. Diese Stellen sind statisch und ändern sich auch bei verschiedenen URLs nicht. Von diesen Positionen ab, wird in bekannten (statischen) Abständen gezählt und beim Erreichen bestimmter Positionen der Text ausgelesen. Es müssen mehrere Referenzmarken gesucht werden, da sich bestimmte Bereiche im HTML Code verändern können (z.B. Werbebanner).

4.2 Abrufen der Informationen durch den "Consumer"

Auf der "Consumer Seite" wurde ebenfalls wie auf der "Producer Seite" ein Python Skript geschrieben. Dieses Skript hat im Gegensatz zu dem Skript auf der Consumer Seite andere Funktionen. Im allgemeinen kann man sagen, dass die Aufgaben dieses Python Skripts folgende sind:

- Subskribieren am ELVIN Server auf das Schlüsselwort "Temperaturen"
- Entnehmen der einzelnen Informationen (Stadt, Temperatur, Luftfeuchtigkeit, Sonnenaufgang, Sonnenuntergang und Update Datum) aus der übertragenen Nachricht
- Erstellen eines HTML Dokuments mit integrierter Tabelle
- Schreiben der erhaltenen Informationen in die Zeilen und Spalten der Tabelle
- Speichern des HTML Dokuments auf der lokalen Festplatte

Verschiedene Probleme sind bei der Entwicklung des Source Codes entstanden, bzw. verschieden Möglichkeiten ergeben sich, wie man die Informationen in einer Tabelle in einem HTML Dokument darstellen kann.

1. Möglichkeit:

Bei der Generierung eines HTML Dokuments kann die Python Bibliothek "HTMLgen" verwendet werden, die speziell zur Erstellung von Webseiten in der Programmiersprache Python benutzt wird. Dabei sind jedoch verschiedene Probleme aufgetreten, die nicht ohne weiteres gelöst werden konnten. Das Hauptproblem bestand darin, dass man der Tabelle keine weiteren Zeilen hinzufügen konnte, und somit sämtliche Informationen in die gleiche Zeile geschrieben wurden. Das bedeutet, dass nur die letzt übertragenden Informationen aus dem Infospace in der Tabelle sichtbar waren, da die anderen zuvor überschrieben wurden.

2. Möglichkeit:

Eine weitere Möglichkeit wäre, die eintreffenden Nachrichten aus dem Infospace in einer Datenstruktur (eine Liste, ein Array, ...) abzuspeichern und im Anschluss daraus eine HTML-Seite zu erstellen. Aktuelle Nachrichten müssten dann bestehende "alte" Informationen überschreiben. Somit wäre es viel einfacher ein HTML-Seite zu erzeugen, da immer alle notwenigen Informationen vorhanden sind. Es entfällt der Aufwand dynamisch Zeilen in einer Tabelle anzufügen/löschen.

3. Möglichkeit:

Diese Möglichkeit wurde letztendlich in diesem Projekt umgesetzt. Dabei wurde im Voraus eine HTML Seite mit dem Programm "Macro Media Dreamweaver" erstellt, auf der ebenfalls von Beginn an eine Tabelle erstellt wurde. Anschließend wurde der HTML Code verwendet um über Strings im Python Code in die HTML Datei geschrieben. Die Informationen werden dabei zeilenweise in die Tabelle eingefügt.
Bei dieser Möglichkeit konnte auch sehr einfach das Navigationsmenü (Buttons) der vorhanden Finnland Webseite in den Python Code übernommen werden.

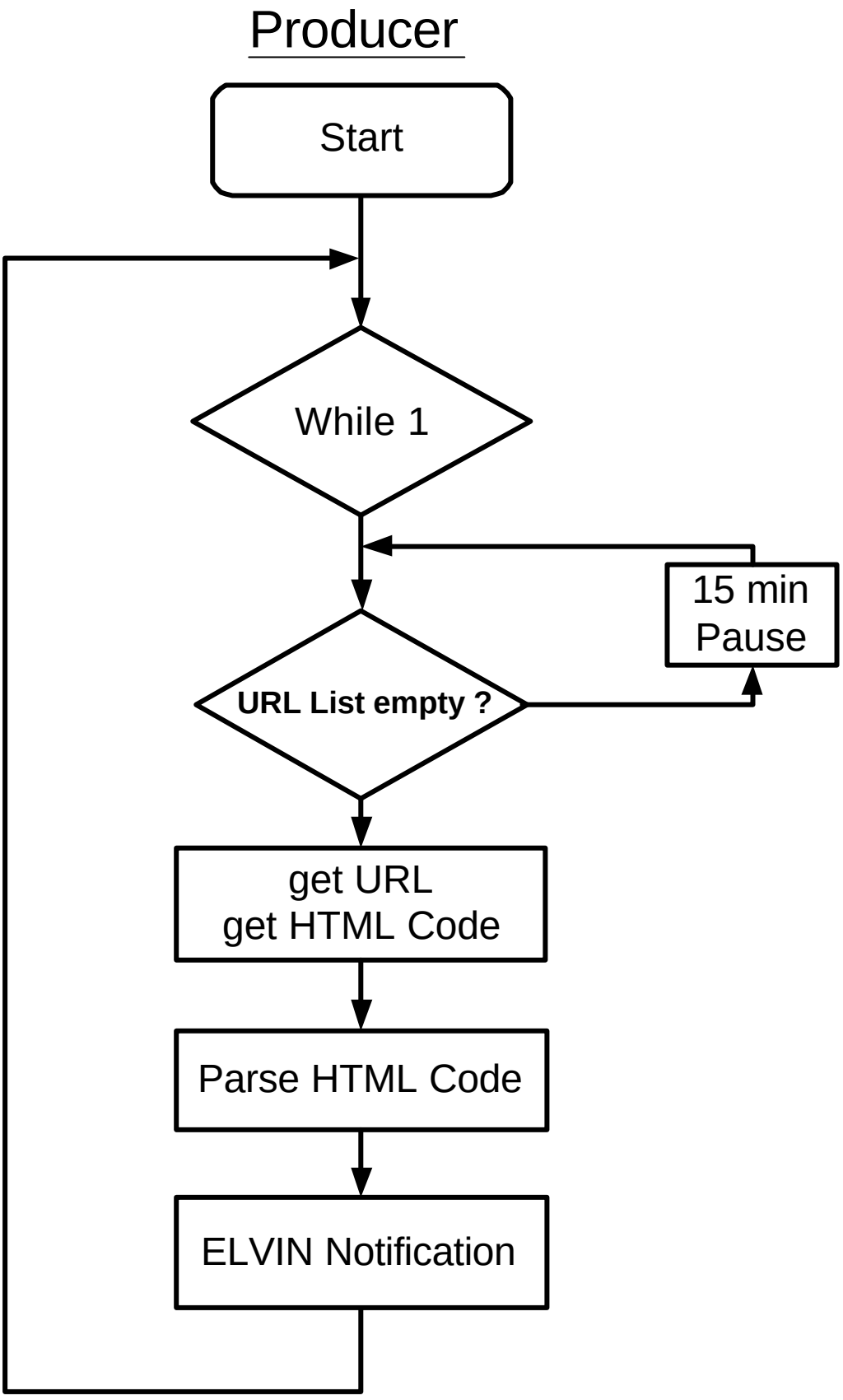
5. Fazit

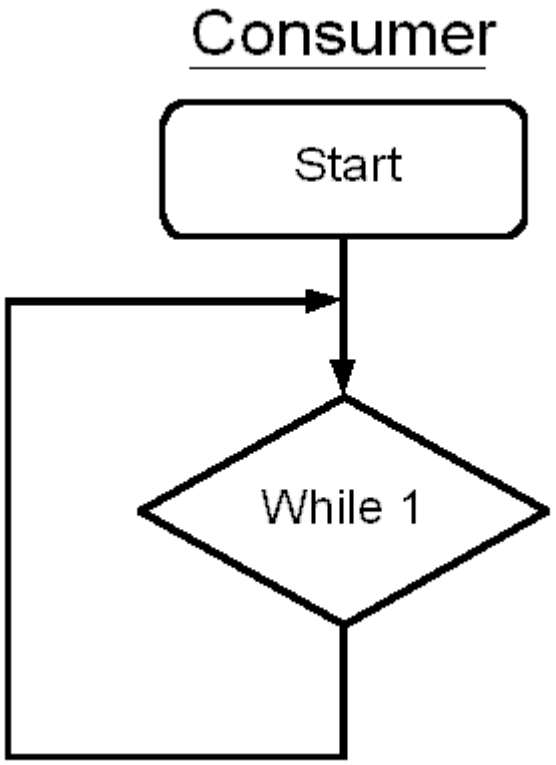
Die Grundfunktionalität des Systems ist gewährleistet, dennoch gibt es zahlreiche Ansätze, um das System zu verbessern.

Beim Client wird momentan nicht darauf geachtet, falls die Verbindung zum ELVIN Server einmal abbrechen sollte. Dazu könnte man ein zusätzliches Verbindungs-management verwenden.
Im Moment gibt es ebenfalls kein "Exception-Handling", d.h. bei Time Outs der Wetterseiten befindet sich der "Producer" in einem undefinierten Zustand.

Eine GUI könnte die Benutzung des Producer-Skripts vereinfachen und angenehmer gestalten, wie z.B. das Auswählen von verschiedenen ELVIN Servern, Verwaltung verschiedener Städtelisten, usw..

6. Flussdiagramme




Consumer
Start
While 1

7. Source Code

Producer:

```
import urllib
import elvin
import string
import time
from sgmllib import SGMLParser

class myParser(SGMLParser):

   def reset(self):
      SGMLParser.reset(self)
self.flg_1 = 0
self.flg_2 = 0
self.flg_3 = 0
self.flg_4 = 0
self.count1 = 0
self.townname = ""
self.temperature_c = ""
self.sunset = ""
self.sunrise = ""
      self.updated = ""
      self.skycond = ""
      self.humidity = ""
      return

   def handle_data(self, text):                                    # function executed when text is found
      if self.flg_1 == 1:                                          # flg_1 set?
         if self.count1 == 3:                                      # towname is reached
            self.townname = text[0:(len(text)-14)]                 # remove unneeded part of string
            self.flg_1 = 0
            self.count1 = 0
         else:
            self.count1 = self.count1 + 1
```

```
    if self.flg_2 == 1:                                     # flg_2 set?

        if self.count1 == 9:                                # skcondition is  reached
            self.skycond = text

        if self.count1 == 15:                               # temperature is reached
            self.temperature_c = text[:-2]                  # remove unneeded part of string

        if self.count1 == 26:                               # humidity reached
            self.humidity = text
            self.count1 = 0
            self.flg_2 = 0
        else:self.count1 = self.count1 + 1


    if self.flg_3 == 1:
        if self.count1 == 16:                               # sunrise reached
            text =  string.replace(text, "\r", "")          # remove crlf
            text =  string.replace(text, "\n", "")
            self.sunrise = text

        if self.count1 == 21:                               # sunset reached
            text =  string.replace(text, "\r", "")          # remove crlf
            text =  string.replace(text, "\n", "")
            self.sunset = text
            self.count1 = 0
            self.flg_3 = 0
        else:self.count1 = self.count1 + 1


    if text[:8] == "Updated:":                              # update reached
        text = string.replace(text, "\r", "")               # remove crlf
        text = string.replace(text, "\n", "")
        self.updated = text[8:len(text)]                    # remove unneeded part of string

    return

def handle_comment(self, text):                             # function executed when a comment is found
```

```
        if text == "   BEGIN LOCAL HEADER   file://":                    # known position reached
            self.flg_1 = 1                                               # set flg_1
        if text == " ----------BEGIN CURRENT CONDITIONS---------- file://":  # known position reached
            self.flg_2 = 1                                               # set flg_2
        if text == " BEGIN ASTRO INFO file://":                          # known position reached
            self.flg_3 = 1                                               # set flg_3

        return

### 10 towns in finland http://www.1uptravel.com/weather-forecast/finland.html
URL_list =["http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFHK&Submit=GO",     # 10 towns in
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFIV&Submit=GO",     # finnland
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFJO&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFJY&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFLP&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFOU&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFRO&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFSA&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFTP&Submit=GO",
           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EFTU&Submit=GO"]

### 7 towns in germany http://www.1uptravel.com/weather-forecast/germany.html
##URL_list_2 =["http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDAC&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDDH&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDDI&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDDK&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDDM&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=EDTL&Submit=GO",
##           "http://www.1uptravel.com/cgi-bin/weather-forecast/hw3.cgi?config=&forecast=zandh&pands=ETSE&Submit=GO"]


while 1:
    for s in URL_list:                                                   # process all URLs in list
        f = urllib.urlopen(s)
        s = f.read()
        parser = myParser()
        parser.feed(s)                                                   # parse code
        parser.close()
```

```
    print "Town: " + parser.townname
    print "Temperature: " + parser.temperature_c + " C"
    print "Humidity: " + parser.humidity
    print "Sunrise: " + parser.sunrise
    print "Sunset: " + parser.sunset
    print parser.updated
    print " "
    f.close()

    connection = elvin.connect("elvin://localhost")
    connection.notify(Temperaturen="Finland", Town=parser.townname, C=parser.temperature_c,H=parser.humidity,
            SR=parser.sunrise, SS=parser.sunset, U= parser.updated)
    connection.close()
    time.sleep(5)                                                    # short pause
  time.sleep(900)                                                    # 15 minutes pause
```

Consumer:

```
import elvin                                                    # Importieren der Elvin Biblio

list_of_cities = []

def deliver(sub, msg, insec, rock):

   town = msg.get("Town")                                       # Stadtname aus der Message
   c = msg.get("C")                                             # Temperatur aus der Message
   h = msg.get("H")                                             # Luftfeuchtigkeit aus der Message
   sr = msg.get("SR")                                           # Sonnenaufgang aus der Message
   ss = msg.get("SS")                                           # Sonnenuntergang aus der Message
   u = msg.get("U")                                             # akutuelles Datum aus der Message

   list_of_towns = [town,c,h,sr,ss,u]                           # Zusammenfügen der einzelnen Infos in eine Liste
   list_of_cities.append(list_of_towns)                         #

   if len(city) == 7 :                                          # Liste voll?
      file = open('temperaturen.htm', 'w')
      file.write('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><html><head><title>Untitled Document</title><meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1"></head><body><p> <br>  <a href="Fakten.htm" target="_self"><img src="Web/Submenu1_1.gif" width="75" height="25"
border="0"></a><a href="Bildergalerie.htm" target="_self"><img src="Web/Submenu1_2.gif" width="100" height="25" hspace="10" border="0"></a><a
href="%DCbermich_1.htm" target="_self"><img src="Web/Submenu1_3.gif" width="100" height="25" hspace="7" border="0"></a><img src="Web/Submenu1_4_1.gif"
width="200" height="25"></p><p><font face="Verdana, Arial, Helvetica, sans-serif">Hier finden Sie die aktuellen Temperaturen von verschiedenen finnischen </font><br>
<font face="Verdana, Arial, Helvetica, sans-serif">St&auml;dten.</font></p></p><table width="500" border="1" cellpadding="2" cellspacing="0" bordercolor="#CCCCCC">
<tr>      <td><font face="Verdana, Arial, Helvetica, sans-serif">Stadt</font></td>      <td><font face="Verdana, Arial, Helvetica, sans-serif">Temperatur</font></td>
<td><font face="Verdana, Arial, Helvetica, sans-serif">Luftfeuchtigkeit</font></td><td><p><font face="Verdana, Arial, Helvetica, sans-serif">Sonnen-</font><br><font
face="Verdana, Arial, Helvetica, sans-serif">aufgang</font><br></p></td>    <td><p><font face="Verdana, Arial, Helvetica, sans-serif">Sonnen-</font><br><font
face="Verdana, Arial, Helvetica, sans-serif">untergang</font></p></td><td><font face="Verdana, Arial, Helvetica, sans-serif">Update</font></td></tr>')
      for temp in city:
         file.write('<tr><td>')
         file.write(str(temp[0]))
         file.write('</td><td align="right">')
         file.write(str(temp[1]))
         file.write('         </td><td align="right">')
         file.write(str(temp[2]))
         file.write('              </td><td align="right">')
```

```
            file.write(str(temp[3]))
            file.write('          </td><td align="right">')
            file.write(str(temp[4]))
            file.write('             </td><td>')
            file.write(str(temp[5]))
            file.write(' </td></tr>')

        file.write('</table></body></html>')
        file.close()

        del list_of_cities[0:len(list_of_cities)]                    # Loesche Elemente aus Liste
        print "HTML file written"

    return

#************Aufbau einer Verbindung zum ELVIN Server**********************

connection = elvin.connect("elvin://localhost")
sub = connection.subscribe("require(Temperaturen)")
sub.add_listener(deliver)
sub.register()
connection.run()
```