Lehrbeauftragte:
DI Peter Meerwald MSc
DI Robert Praxmarer

Team:
Angerer Theresa
Czernik Thomas
Frick Matthias
Havranek Ivo

Web Spider Dokumentation

In der Lehrveranstaltung

Erweiterte Konzepte der Programmierung

MultiMediaTechnology
Bachelorstudiengang
mmt

Salzburg im SS2010, am 18.04.2010

Inhaltsverzeichnis

Unser Ziel war es, einen Web Crawler zu entwickeln, der Websites nach sogenannten „broken links“ absucht. Unser Programm kann mit solchen Links umgehen und ermittelt innerhalb einer Domain alle kaputten oder aber funktionsfähigen Links. Unsere Implementierung des Web Crawlers wurde vollständig in C++ umgesetzt und arbeitet kommandozeilenbasiert.
Das Repository kann unter git@github.com:the-resa/WebSpider.git geclont werden.

1. Broken Links Liste und Statistik

Für das Untersuchen der Links haben wir eine Website entwickelt. Diese befindet sich unter:

http://83.169.37.29/frick/Demosite/ bzw.
http://www.wrel.de/frick/Demosite/.

Die Seite enthält Angaben zur Anzahl an korrekten und broken Links.

2. Programmarchitektur

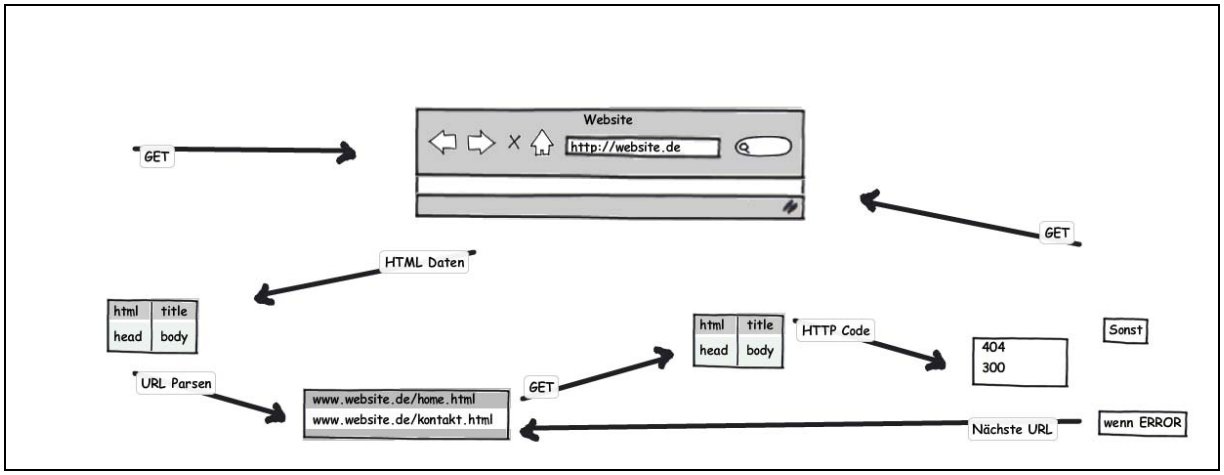

Abb. 1: WebCrawler - skizzierter Aufbau

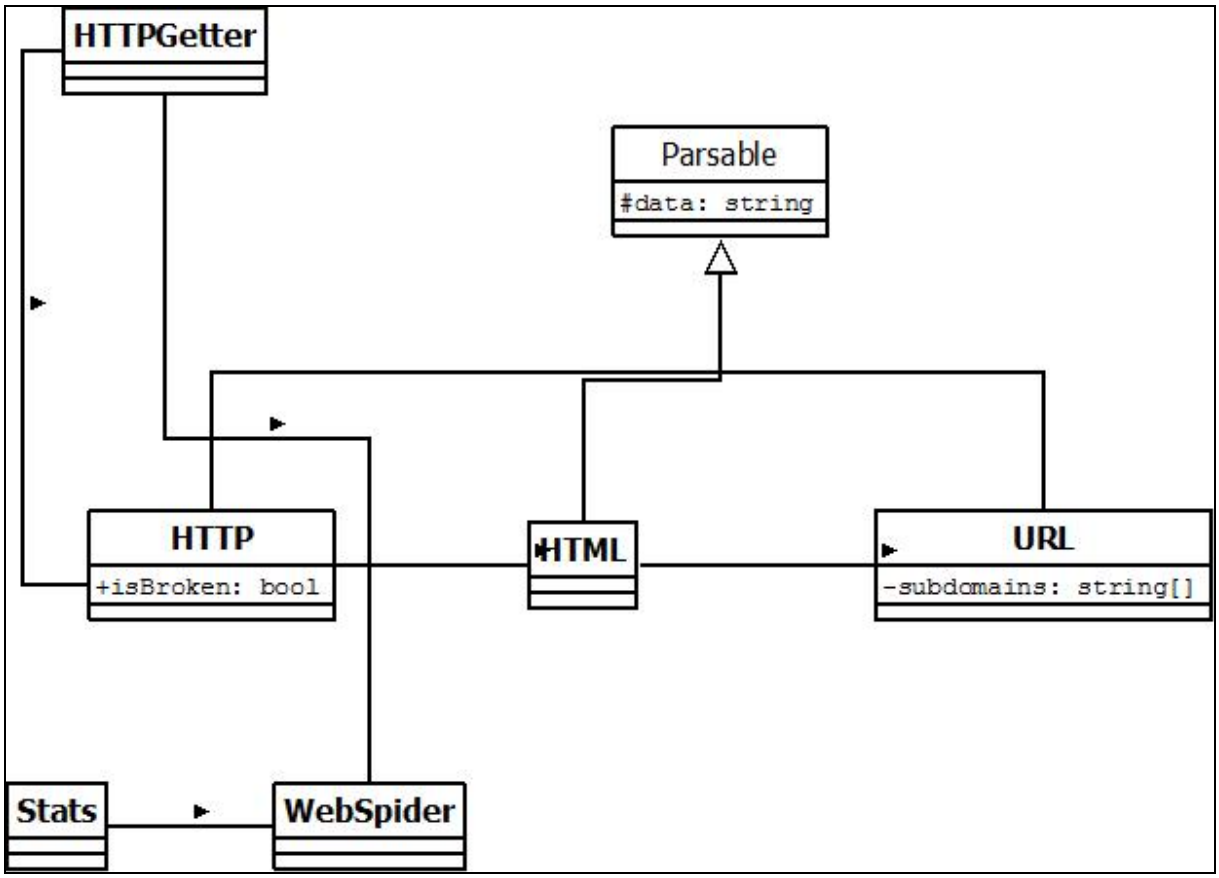

Abb. 2: UML Diagramm - eine erste Spezifikation

Zur Umsetzung haben wir zunächst folgenden Pseudocode entwickelt. Das fertige Programm orientiert sich in seinem Grundaufbau an diesem Entwurf:

```
void crawlURL (url) {
    crawledURLs.add (url); // using vec to save data

    if (status_code != 200) {
        brokenURLs.add (url);
    }
    else {
        URLs = HTMLParser.getURLs (url);
        for (int i = 0; i < crawledURLs.size (); i++) {
            for (int j = 0; j < URLs.size (); j++) {
                if (crawledURLs[i] != URLs[j]) {
                    crawlURL (URLs[j]);
                }
            }
        }
    }
}
```

Abb. 3: Pseudocode

Später haben wir die Programmarchitektur wieder umstrukturiert (vgl. Abb. 2 bzw. Abb. 4). Der Web Crawler hat nun diesen Aufbau:

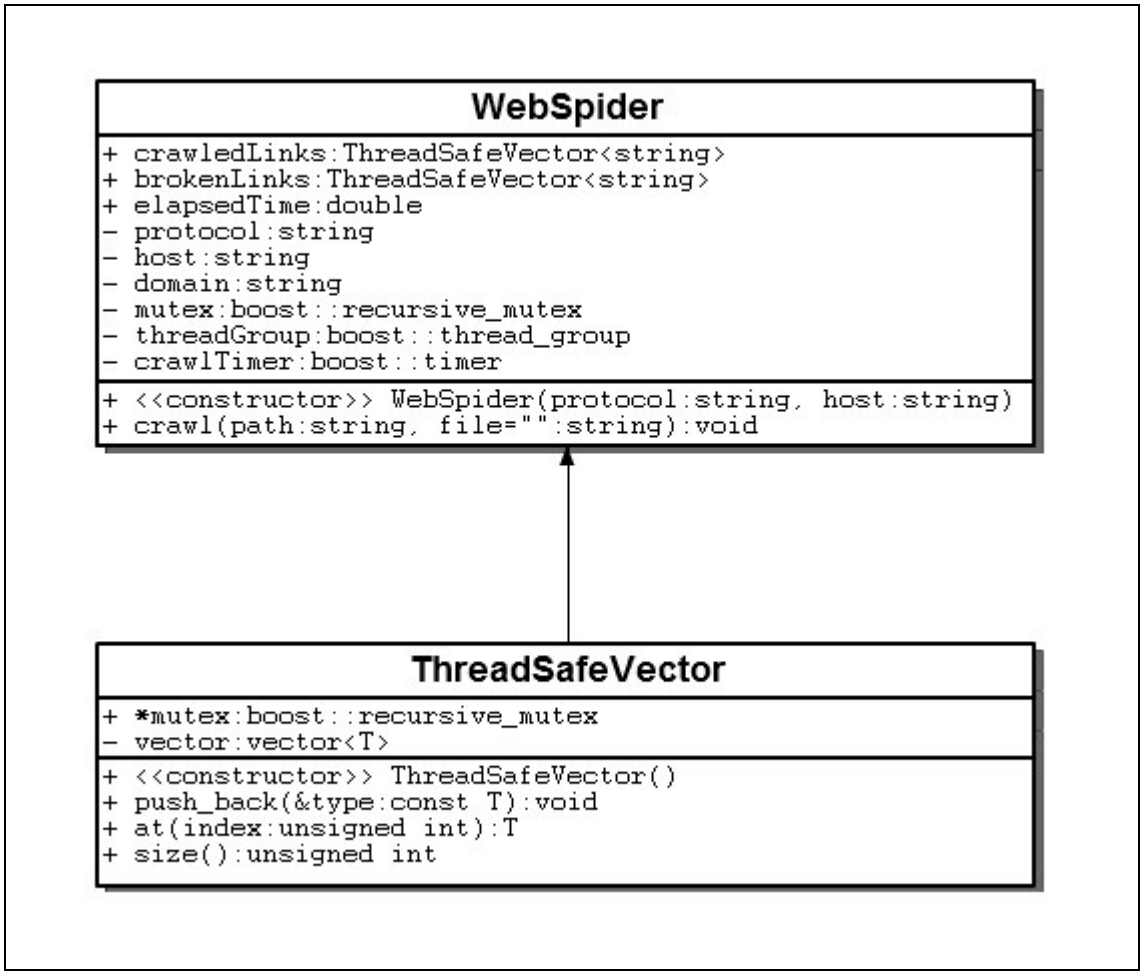

Abb. 4: UML Diagramm - finale Fassung

3. Automatisierter Programmaufbau

Anhand der ***makefile*** (zu finden unter *WebSpider/WebSpiderVS*) werden die Abhängigkeiten und Bedingungen unseres Web Crawlers miteinander verbunden. Dadurch ergibt sich dann später ein ausführbares Programm.

Mithilfe von ***CXX=*** geben wir den Compiler an. In unsrem Fall ist dies der g++ Compiler.

Unser Target heißt ***webcrawler***, und kann dann mit ***make webcrawler*** einfach aufgerufen werden. Mit ***$(CXX)*** können wir ein Target erzeugen. Dazu wird bei unserem Web Crawler der Objektcode von ***Webspider.o*** und ***main.o*** mit der Library ***boost Thread***, ***boost regex*** und ***lbooxt_system*** miteinander verknüpft.

boost Thread ist hierbei für das Multithreading verantwortlich. ***boost regex*** managt Regular Expressions und das ***lbooxt_system*** arbeitet mit dem http-Request.

4. Threading

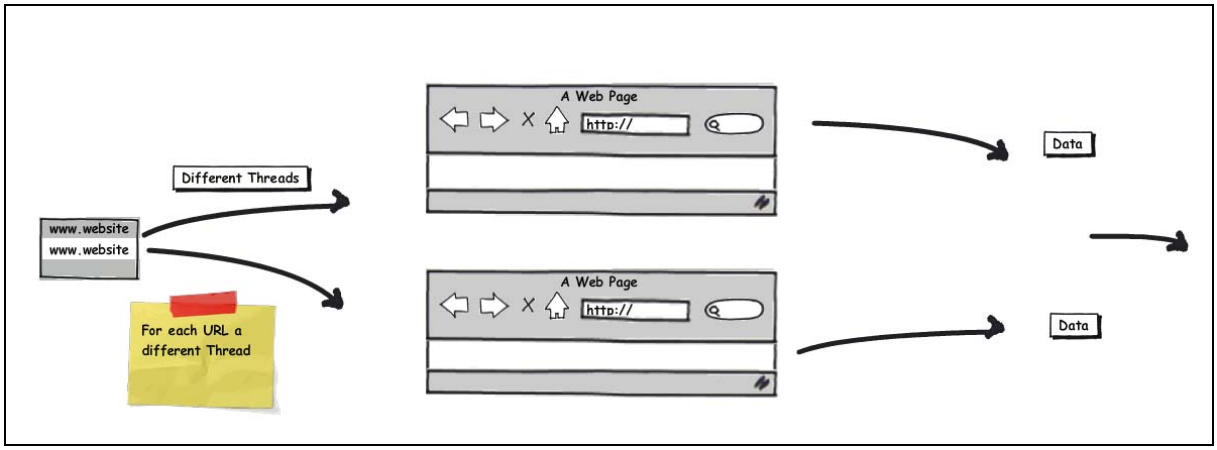

Abb. 4: Threading - Grobfassung

Unser Programm enthält einen Thread (siehe auch Punkt 5). Dieser ist folgendermaßen aufgebaut:

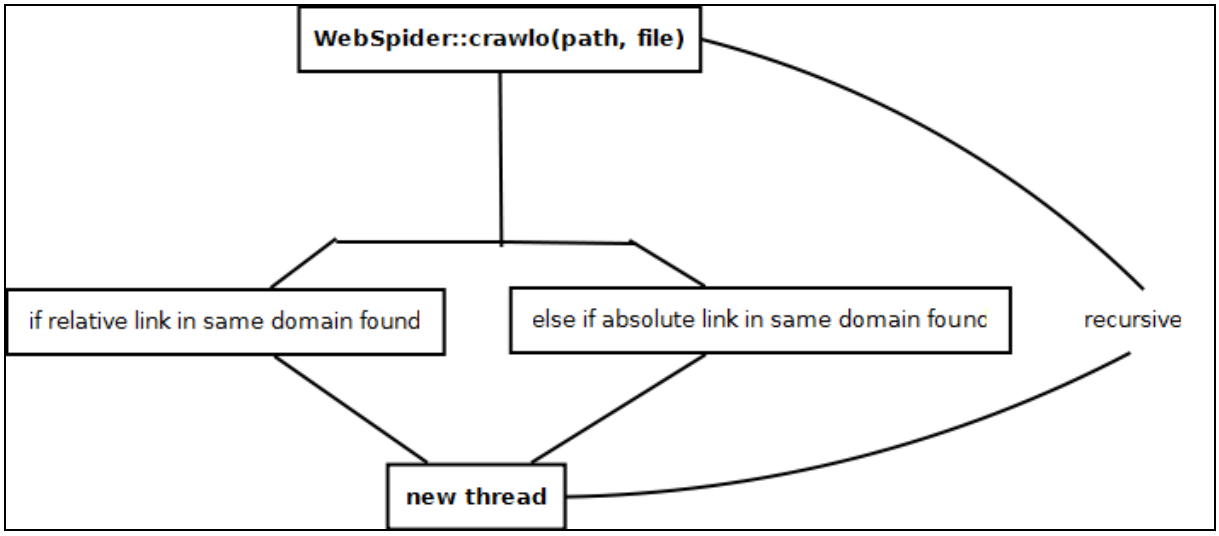

Abb. 5: Threading innerhalb unseres Programmes

5. Multithreading

Da unser Crawler rekursiv arbeitet, gelang es uns leider nicht, ein echtes Multithreading zu generieren. Bei Testläufen arbeitete unser Single-Thread Programm genauso schnell, wie die Multithreading Implementierung.

6. Programmeinschränkungen

Unser Web Crawler kann nicht mit Redirects umgehen, diese erkennt er einfach als broken link an.
Weiters ist unser Web Crawler nicht HTTPS oder FTP fähig. Er versteht praktisch nur HTTP und kann damit umgehen und dies behandeln.
Der Web Crawler sucht in der eigenen Domain nach broken links, er geht dabei jedoch nicht aus diesem Domain-Bereich hinaus. Das bedeutet, dass ein Link vom Domain-Bereich "wrel.de" auf "google.at" nicht weiter verfolgt wird. Er ist somit auf den einen Bereich konzentriert und eingeschränkt. Unser Web Crawler beachtet keine robots.txt und kann nicht mit Cookies umgehen.

7. Programmerweiterungen

Wie bereits unter Punkt 6. erwähnt, kann unser Web Crawler nur innerhalb einer Domain suchen. Dies jedoch ist auch als positiv zu vermerken, da so das Progamm nicht abstürzen kann. Der IP-Range ist eingeschränkt, damit wir die Funktionalität unseres Crawlers besser untersuchen können.
Ein weiteres Feature ist, dass das Programm eine Seite nur einmal untersucht. Es kommt so nicht zu verfälschten Tests des Crawlers. Links werden dadurch auch zahlenmäßig richtig ermittelt.

8. Verwendete Bibliotheken

Für die Realisierung unseres Web Crawler Projektes haben wir lediglich die Boost Library verwendet. Genauer gesagt die ***boost_1_42*** Library von Beman Dawes (2008).
Die genaue Installation dieser Library haben wir anhand einer README Datei beschrieben (zu finden im Root Verzeichnis des Repositorys).
Die Bibliothek unterliegt folgender Lizenz:
Boost Software License - Version 1.0 - August 17th, 2003.

Diese ist komplett also Open Source erhältlich und unterliegt keinerlei Beschränkungen in ihrer Verwendung oder aber sonstiger Rechte.