# Screen Scraping

## Integration heterogener Daten aus Webseiten in einen unternehmenseigenen Datenbestand

**Tobias Kiertscher**

### Inhalt

# Einleitung

Das Internet umfasst eine große Anzahl von Websites mit frei verfügbaren Daten, die aber in vielen Fällen nur für die Nutzung durch einen menschlichen Benutzer gestaltet sind. Erst in den letzten Jahren nimmt die zusätzliche Bereitstellung von Schnittstellen, die für die Maschine-zu-Maschine-Kommunikation optimiert sind, zu. Unter dem Stichwort *Open Data* entstehen vermehrt Bemühungen, Daten nicht nur frei zu Verfügung zu stellen, sondern auch über eine Schnittstelle, mit der sie sich leicht in neue oder bestehende Anwendungen integrieren lassen.

Diese Bemühungen werden jedoch auf absehbare Zeit nicht alle interessanten Websites erfassen. Es existieren weiterhin eine Vielzahl von Websites mit wertvollen Informationen, wo entweder die notwendigen Mittel fehlen, um neben der menschenfreundlichen auch noch eine maschinenfreundliche Schnittstelle anzubieten, oder wo seitens der Betreiber schlicht kein Interesse an einer solchen besteht.

In der Integration der Informationen aus diesen nicht maschinenfreundlichen Websites, mit den unternehmenseigenen Daten, steckt großes Potential. Um dieses Potential auszuschöpfen müssen die Informationen aus den Websites extrahiert und strukturiert gespeichert werden. Diesen Zweck erfüllt das Screen Scraping.

Das *Screen Scraping* ist eine Vorgehensweise, bei der automatisiert strukturierte Informationen aus Webseiten gewonnen werden. Dabei sind die Webseiten als grafische Schnittstelle für den Menschen gestaltet und wurden nicht dafür optimiert, dass Programme die enthaltenen Informationen automatisch extrahieren und weiterverarbeiten. Beim Screen Scraping werden Webseiten nicht wie üblich durch einen Browser abgerufen und grafisch dargestellt, sondern durch ein spezielles Programm abgerufen, das den HTML-Quelltext, i. d. R. ohne grafische Ausgabe, nach spezifischen Mustern durchsucht und Informationen extrahiert.

## Zielstellung

Beim Erfassen von Informationen durch *Screen Scraping* ist das Ziel, strukturierte Informationen in einer Datenbank abzulegen, um diese anschließend effizient durchsuchen und/oder weiterverarbeiten zu können. Insbesondere sollen tabellarische Informationen, potentiell mit verlinkten Detailseiten, in einer Website iden-

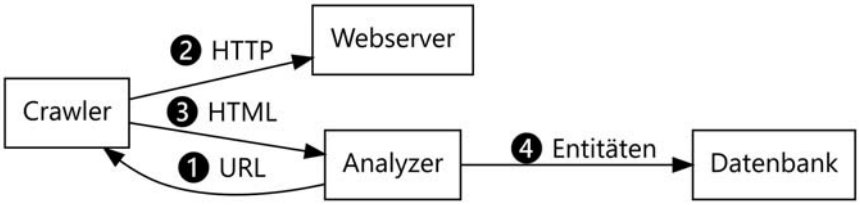
tifiziert, extrahiert, bereinigt, zusammengeführt und dann in einer Datenbank abgelegt werden. Diese Daten sollen anschließend mit unternehmenseigenen Daten kombiniert und in Folge ein Erkenntnisgewinn erreicht werden, der einen Wettbewerbsvorteil ermöglicht.

## Begriffe

Als *Website* wird eine Gruppe von *Webseiten* verstanden, welche über eine Navigationsstruktur, z. B. ein Menü, miteinander verknüpft sind. Die Webseiten einer Website sind in HTML kodiert und werden mittels eines *Webservers* über HTTP bzw. HTTPS bereitgestellt. Eine Website wird über einen *Domänennamen* und optional einen Pfad identifiziert. Der Domänenname und der Pfad werden in einer *URL* zusammengefasst.

Die Informationen, welche aus einer Website extrahiert werden sollen, sind in Entitäten gruppiert. Eine *Entität* enthält Informationen über eine identifizierbare Sache. Die Sache kann gegenständlich oder abstrakt sein. Eine Entität besitzt einen *Identifikator* mit dem sie von anderen Entitäten eindeutig unterschieden werden kann. Wenn Informationen über eine *Entität* auf mehrere Webseiten verteilt sind, werden diese partiellen Informationen *Facetten* genannt.

Ein Screen-Scraping-System besteht mindestens aus einem Crawler und einem Analyzer. Der *Crawler* ruft die HTML-kodierten Webseiten vom Webserver ab. Der *Analyzer* extrahiert die Informationen aus den Webseiten und speichert sie in einer *Datenbank*.




**Grundlegender Ablauf des Screen Scraping**

## Kriterien zur Optimierung

Ein Screen-Scraping-System kann nach unterschiedlichen Kriterien optimiert werden. Dabei bilden diese Kriterien oft Gegensätze, welche manchmal nur die Parametrisierung, aber manchmal auch die Architektur des Systems beeinflussen. Das hat zur Folge, dass u.U. bereits während der Entwicklung des Systems entschieden werden muss, welche Kriterien Priorität haben.

- Navigationsstrategie:
  Effiziente Erfassung ⇔ Unauffällige Erfassung
- Abfragestrategie:
  Schnelle Erfassung ⇔ Unauffällige Erfassung
- Analysestrategie:
  Einfache Implementierung ⇔ Robustheit gegen Änderungen des Seitenaufbaus

# Navigationsstrategie

Die Navigationsstrategie steuert die Erzeugung bzw. Ermittlung der URLs für den Seitenabruf. Folgen die URLs einem festen Schema und enthalten sie eindeutige Bezeichner oder IDs, die im Voraus bekannt sind, können sie oft direkt gebildet werden und müssen nicht aus einer Navigationsschaltfläche extrahiert werden. Dies wird als *Reverse Engineering von URLs* bezeichnet. Stehen die URLs jedoch nicht in einer direkten Beziehung zu den Informationen auf den Webseiten, müssen die entsprechenden Link-Elemente auf den Webseiten identifiziert und die im `href`-Attribut referenzierten URLs verwendet werden. Dies wird als *explorative Traversierung* bezeichnet.

Dieser Abschnitt gliedert sich wie folgt:

- Reverse Engineering von URLs
- Explorative Traversierung
- Erkenntnisse

## Reverse Engineering von URLs

URLs für den Webseitenabruf besitzen üblicherweise den folgenden Aufbau:

*Protokoll* `://` *Server* [ `:` *Port*] `/` *Pfad* [ `?` *Abfrageparameter*] [ `#` *Anker*]

Der *Pfad* und die *Abfrageparameter* sind üblicherweise die variablen Teile beim Screen Scraping. Das *Protokoll*, der *Server* und, falls erforderlich, der *Port* stehen i. d. R. fest. Der *Anker* spielt für das Screen Scraping keine Rolle, da er angibt, welchen Teil einer Seite der Browser als erstes darstellen soll. Beim Screen Scraping wird die Seite jedoch normalerweise nicht gerendert, sondern lediglich der HTML-Quelltext der Seite analysiert.

Die Analyse einer URL kann unterschiedlich kompliziert sein und soll hier nicht im Detail erläutert werden. Es lassen sich jedoch häufig die folgenden Teile in Pfad oder Abfrageparametern identifizieren.

- Kategorie
- Seitenname
- Entitätenklasse
- Entitäten-ID
- Filterkriterium
- Paginationsparameter
- Anzeigeparameter
- Session-ID

An dieser Stelle ist das Ausprobieren von manuell veränderten URLs hilfreich, um die Möglichkeiten des Web-Servers zu erproben. Falls die Website ein Web-Framework oder ein Content-Management-System verwendet, dessen Dokumentation öffentlich zugänglich ist, kann es helfen diese nach Hintergrundinformationen und Systematiken zu durchsuchen.

### Vorteile

- Unabhängiger von konkreter Gestaltung der Navigationsschaltflächen
- Effizienter in der Abfrage da Seiten mit gesuchten Inhalten direkt abgefragt werden können

### Nachteile

- Abhängig vom konkreten Aufbau der URLs
- Auffälliger im Abfragemuster

Unter der Annahme, dass sich der technische Unterbau einer komplexen Website nicht so schnell ändert wie das Design bzw. die konkrete Beschriftung von Navigationsschaltflächen, ist das Reverse Engineering der URLs die robustere Variante.

Ob sich der technische Unterbau einer Website ändert, hängt natürlich von einer ganzen Reihe von Faktoren ab, die sich nicht immer sicher ermitteln bzw. voraussehen lassen. Dazu zählen z. B. die Reife der aktuellen Implementierung, die strukturelle Wandlung der präsentierten Informationen, die Kapazitäten des Website-Betreibers und zukünftige technische Entwicklungen.

## Explorative Traversierung

Bei der explorativen Traversierung werden, ausgehend von der Startseite, Navigationsschaltflächen gesucht, welche unmittelbar oder mittelbar zu der Seite mit den gesuchten Informationen führen. Aus diesen Schaltflächen wird dann die URL der Zielseite extrahiert. Normalerweise werden Navigationsschaltflächen im HTML durch `a`-Elemente repräsentiert. Das `href`-Attribut enthält dann die absolute oder relative URL der Zielseite.

Die Herausforderung bei dieser Strategie ist das algorithmische Beschreiben von Navigationspfaden. In einem einfachen Fall kann eine Liste von Identifikatoren ausreichen, um eine Zielseite zu beschreiben. Dabei beschreibt jeder Identifikator eindeutig eine Navigationsschaltfläche im HTML-Baum der aktuellen Seite. Als Identifikator kann z. B. eine ID, eine CSS-Klasse, ein XPath oder ein CSS-Selektor dienen. Die Liste wird ausgehend von der Startseite abgearbeitet, indem mit dem jeweiligen Identifikator die Navigationsschaltfläche und mit ihr die URL der nächsten Seite ermittelt wird. Diese URL wird aufgerufen und auf der geladenen Seite wird der nächste Identifikator verwendet. Ist die Liste zu ende, sollte die zuletzt geladene Seite die Zielseite sein. Zusätzliche Prüfkriterien, wie zu erwartende Elemente in der Seite oder Seitentitel können dazu dienen, frühzeitig zu erkennen, ob ein Navigationsschritt erfolgreich war oder nicht.

### Vorteile

- Unabhängiger von technischer Umsetzung (URLs)
- Entspricht mehr dem menschlichen Verhalten und ist damit unauffälliger

### Nachteile

- Komplexer in der Implementierung
- Abhängiger von konkreter Ausprägung des Designs
- Es müssen öfter Zwischenseiten aufgerufen werden, um zu den eigentlich gesuchten Informationen zu gelangen

## Erkenntnisse

Das Reverse Engineering von URLs eignet sich besser für das effiziente Erfassen von Inhalten. Da die URLs für die Webseiten mit den interessanten Inhalten u.U. direkt gebildet werden können, ist es nicht erforderlich, Zwischenseiten ausschließlich für den Zweck aufzurufen, über Navigationsschaltflächen die nächste URL zu erhalten. Des Weiteren können Abfragen häufig parallel getätigt werden, was den Durchsatz der Datenerfassung erhöht.

Die explorative Traversierung eignet sich eher für das unauffällige Erfassen von Inhalten. Denn das Traversieren der Navigationspfade stellt sicher, dass man Webseiten in einer Reihenfolge aufruft, die auch von einem menschlichen Benutzer gewählt werden könnte. Auch der Zwang eine Seite nach der anderen abzurufen, entspricht dem menschlichen Verhalten. Durch die serielle Natur der Traversierung ist jedoch die Geschwindigkeit und Effizienz der Datenerfassung begrenzt.

# Abfragestrategie

Die Art wie URLs mittels HTTP abgerufen werden, kann ein kritischer Punkt für den Erfolg für das Screen Scraping sein. Je nach Kooperationsbereitschaft des Website-Betreibers und Kapazitäten des Web-Servers kann die Abfrage auf Geschwindigkeit und Effizienz oder auf Unauffälligkeit optimiert werden.

Dieser Abschnitt gliedert sich wie folgt:

- Verzögerung von HTTP-Anfragen
- Abruf von referenzierten Ressourcen
- Parallelisierung
- HTTP-Header
- Abfragezeitpunkt
- Erkenntnisse

Ein menschlicher Benutzer ruft i. d. R. im Browser nur einen Link einer Seite auf und wartet, bis diese aufgebaut ist. Der Browser ruft die Seite über HTTP ab und schickt dabei eine Reihe von HTTP-Header-Werten mit. HTML-Seiten referenzieren üblicherweise eine Vielzahl von zusätzlichen Ressourcen (CSS-Styles, JavaScript-Dateien, Bilder, …), welche von einem Browser abgerufen werden müssen. Erst dann

stellt der Browser die Seite grafisch dar und der Benutzer kann evtl. erneut auf einen Link klicken. Menschen besuchen Webseiten üblicherweise zu bestimmten Tageszeiten und selten mehrfach zum exakt gleichen Zeitpunkt.

Ein performantes Vorgehen beim Screen Scraping wäre hingegen das frühzeitige Erzeugen von mehreren URLs und deren parallele Abfrage. Es werden nur die HTTP-Header-Werte mitgeschickt, die unbedingt erforderlich sind, damit der Web-Server die gewünschte HTML-Seite ausliefert. Weitere Ressourcen werden nicht abgerufen, da dies die Implementierung verkompliziert und unnötige Übertragungskapazität verbraucht. Eine wesentliche Verzögerung von aufeinanderfolgenden Abrufen ist auch nicht notwendig. Sobald URLs erzeugt oder extrahiert wurden, die abgerufen werden sollen, kann dies unmittelbar geschehen, um die Übertragungskapazität optimal auszulasten. Sollen Informationen regelmäßig abgerufen werden, bietet es sich an, das Screen Scraping jede Nacht zu einem geplanten Zeitpunkt z. B. mit einem Cron-Job durchzuführen.

## Verzögerung von HTTP-Anfragen

Eine Verzögerung von HTTP-Anfragen senkt die Last auf dem Web-Server und kann das Zugriffsmuster dem eines menschlichen Benutzers angleichen. Als Nachteil senkt sie naturgemäß auch die Abfragegeschwindigkeit. Die Verzögerung kann einfach implementiert werden, indem vor jeder HTTP-Anfrage eine künstliche Pause eingeführt wird. Sollten im HTML referenzierte Ressourcen abgerufen werden, brauchen diese Abfragen nicht verzögert zu werden, da auch ein Browser referenzierte Ressourcen so schnell wie möglich und i. d. R. parallel abruft.

Jede Art von Pause senkt die Last auf dem Web-Server. Die Länge der Pause kann jedoch wichtig sein, wenn es darum geht einen menschlichen Benutzer nachzuahmen. Möglich sind z. B. eine feste Länge, eine zufällige Länge aus einem Intervall, oder eine Länge, die aus Eigenschaften der zuvor abgerufenen Web-Seite abgeleitet wird.

Eine feste Länge ist gut geeignet, um die Last auf dem Web-Server zuverlässig zu senken. Auch dadurch fallen die Zugriffe des Screen-Scraping-Programms schon weniger in den Statistiken des Web-Servers auf. Sie ist jedoch kaum geeignet, das Zugriffsmuster eines menschlichen Benutzers nachzuahmen.

Bei einer zufälligen Länge kann auch die statistische Verteilung eine Rolle spielen. Die üblichen Pseudozufallsgeneratoren erzeugen eine Gleichverteilung. Eine Normalverteilung entspricht jedoch eher einem natürlichen Prozess und ahmt damit vermutlich das menschliche Verhalten besser nach (vergl. [1]).

Soll die Länge der Pause aus der zuvor abgerufenen Web-Seite abgeleitet werden, kann z. B. die Größe der Web-Seite – Anzahl der Zeichen im HTML-Quelltext, Anzahl der HTML-Elemente innerhalb des `body`-Elements, o.ä. – als Multiplikator für eine minimale Länge genutzt werden. Dies spiegelt die Annahme wieder, dass ein menschlicher Benutzer bei großen Seiten länger braucht, um auf einen Folgelink zu klicken, als bei kleinen Seiten. Eine Berücksichtigung der zuvor abgerufenen Web-Seite ist nur dann sinnvoll, wenn die Navigationsstrategie der explorativen Traversierung eingesetzt wird.

Die letzten beiden Ansätze lassen sich auch gut kombinieren.

## Abruf von referenzierten Ressourcen

Für die Darstellung einer Web-Seite verwendet der Browser ein *Cascading Stylesheet* (CSS), welches für jedes HTML-Element vorgibt, wie es formatiert werden soll. Nahezu jede Web-Seite enthält eine oder mehrere Referenzen auf CSS-Dateien, welche das Aussehen der Web-Seite steuern. Zusätzlich referenzieren HTML-Seiten auch noch weitere Ressourcen, wie JavaScript-Code, Bilder und Web-Fonts. Der Browser muss diese mittels HTTP-Abrufen, bevor er die Web-Seite korrekt rendern kann. Um dem Benutzer möglichst schnell eine korrekte grafische Darstellung bieten zu können, lädt der Browser alle im HTML-Code referenzierten Ressourcen parallel herunter. Zusätzlich versucht der Browser bei dem Wechsel von einer Web-Seite zur nächsten, bereits heruntergeladene Ressourcen wieder zu verwenden. Dazu speichert er diese in einem lokalen Cache.

Folglich entsteht auf dem Web-Server ein typisches Muster für einen Website-Besuch: Bei Abruf der ersten Seite einer Website wird zunächst die HTML-Quelle und anschließend parallel alle referenzierten Ressourcen angefragt. Bei einem Seitenwechsel, z. B. durch das Anklicken eines internen Links, wird wieder zunächst die HTML-Quelle und anschließend parallel nur noch jene referenzierten Ressourcen angefragt, die noch nicht im lokalen Cache zwischengespeichert sind.

Für das Screen Scraping sind die referenzierten Ressourcen i. d. R. nicht von Bedeutung, da die gesuchten Informationen im HTML-Code enthalten sind. Deshalb müssen diese gar nicht abgerufen werden, wenn der Abruf effizient und schnell geschehen soll. Soll der Screen Scraper jedoch möglichst unauffällig sein, muss er derart implementiert werden, dass er das Verhalten eines Browsers nachahmt, der von einem Menschen bedient wird.

## Parallelisierung

Um einen möglichst hohen Durchsatz beim Abruf von HTML-Seiten zu erreichen, können die HTTP-Anfragen parallelisiert werden. Das bedeutet, dass z. B. mittels Threads, mehrere HTTP-Anfragen gleichzeitig an den Web-Server geschickt werden. Hat der Web-Server ausreichende Rechenkapazitäten, evtl. mehrere Prozessoren, oder werden mittels Load-Balancer mehrere Web-Server eingesetzt, kann die Abfragegeschwindigkeit damit erheblich gesteigert werden. Besonders wirksam kann die Parallelisierung bei der Navigationsstrategie mittels Reverse Engineering von URLs eingesetzt werden.

Man sollte sich jedoch bewusst sein, dass das Screen-Scraping-Programm bei parallelisiertem Abruf das Vielfache an Kapazitäten auf der Seite des Website-Betreibers nutzt, als ein einzelner menschlicher Benutzer. Es sollte sichergestellt werden, dass ausreichend Kapazitäten für gewöhnliche Besucher der Website verbleiben.

Soll das Abfragemuster dem eines menschlichen Benutzers angeglichen werden, kann die Parallelisierung jedoch nicht beliebig zur Durchsatzsteigerung eingesetzt werden. Vielmehr sollte sie nur zum Abruf von im HTML referenzierten zusätzlichen Ressourcen verwendet werden, um das Verhalten eines Web-Browsers nachzubilden.

## HTTP-Header

Bei jeder HTTP-Anfrage hat der Client, i. d. R. der Browser, die Möglichkeit neben der URL zusätzliche Informationen an den Web-Server zu übermitteln. Dies geschieht im HTTP-Header. Mit den Web-Entwickler-Werkzeugen von modernen Browsern kann man leicht ermitteln, welche Header-Felder der Browser bei einem typischen Web-Seiten-Abruf mitsendet. Dazu gehören u. a. `User-Agent`, `Accept`, `Accept-Encoding`, `Accept-Language` und evtl. `Cookie`.

Um sicherzugehen, dass der Abruf zuverlässig funktioniert, können einfach alle Header-Felder einer typischen Browser-Anfrage auch im Screen-Scraping-Programm verwendet werden. Dies ist auch erforderlich wenn das Screen Scraping unauffällig erfolgen soll. Zu beachten ist dabei, dass das `User-Agent`-Feld mit den üblichen Update-Zyklen der Browser aktualisiert werden muss.

Ist eine offene Absprache mit dem Website-Betreiber möglich und dieser mit dem Screen Scraping einverstanden, könnte das `User-Agent`-Feld jedoch auch genutzt werden, um das Screen-Scraping-Programm eindeutig als solches zu kennzeichnen. Dann hätte der Website-Betreiber die Möglichkeit, die Anfragen des Screen Scrapings mit entsprechenden Filtern nach seinen Erfordernissen zu priorisieren, oder zumindest die anteilige Last des Screen Scrapings in der Überwachung zu erkennen.

## Abfragezeitpunkt

Der Zeitpunkt zu dem ein Screen Scraping erfolgt, kann aus zweierlei Gründen wichtig sein. Wenn die Kooperation des Website-Betreibers vorausgesetzt werden kann, könnte ein Zeitpunkt gewählt werden, an dem üblicherweise wenige Besucher den Web-Server frequentieren. Dadurch werden Kapazitäten genutzt, die anderenfalls brachliegen würden und der normale Betrieb der Website wird nicht behindert. Soll der Abruf allerdings unauffällig erfolgen, muss das Screen Scraping eben genau dann durchgeführt werden, wenn viele Menschen die Website besuchen. Dann ist die Wahrscheinlichkeit höher, dass die Abfragen des Screen-Scraping-Programms nicht auffallen.

Bei der Wahl des Abfragezeitpunktes ist die Zeitzone der üblichen Besucher zu beachten. Hat die Website internationales Publikum, fällt die Zeitzone nicht so schwer ins Gewicht, wie wenn sich das Publikum z. B. auf ein Land beschränkt.

Wenn der Abruf regelmäßig aber unauffällig geschehen soll, muss der Zeitpunkt des Abrufs jeweils um eine zufällige und natürlich wirkende Zeitspanne verschoben werden (vergl. [1]).

## Erkenntnisse

Das Abrufen von Webseiten und anderen Ressourcen von einem Web-Server verbraucht Kapazitäten (Übertragungskapazität, Rechenkapazität) und hinterlässt

Spuren in Form von Log-Dateien. Ein Website-Betreiber mit entsprechenden Kenntnissen kann leicht erkennen, ob ein einfaches Screen-Scraping-Programm so schnell wie möglich Informationen sammelt oder ob ein menschlicher Benutzer mit einem Browser die Website abruft. Als Folge kann ein IP-Filter konfiguriert werden, der einen potentiell unerwünschten Client ausschließt. Es existieren auch Application-Level-Firewalls, welche die Muster von HTTP-Anfragen permanent überwacht und den Administrator warnt oder Anfragen sogar selbstständig blockiert, wenn unübliche Muster auftreten.

Je nach Szenario ist es notwendig, das Verhalten des Screen-Scraping-Programms dem eines menschlichen Benutzers anzugleichen. Die Motivation dafür kann sehr unterschiedlich aussehen, darf jedoch nicht darin bestehen die Interessen des Website-Betreibers zu verletzen. In vielen Fällen kann es technisch notwendig sein, da der Web-Server nur Browser-typische Anfragen zuverlässig verarbeiten kann.

Wird das Screen Scraping nur ein einziges Mal durchgeführt, rechtfertigt sich der zusätzliche Aufwand für das Nachahmen von menschlichem Verhalten nur in Maßen. Wird es jedoch regelmäßig durchgeführt oder sogar als Datenquelle für ein angebundenes System genutzt, hängt es von der Kooperationsbereitschaft des Website-Betreibers ab, wie weit die Abfrage auf Performance optimiert werden darf. Ein auf Performance optimiertes Screen-Scraping-Programm kann bei schwachen Web-Servern oder einer geringen Übertragungskapazität zu einem deutlichen Anstieg der Antwortzeit führen und kommt damit einem Denial-of-Service-Angriff nahe.

## Analysestrategie

Die Analysestrategie steuert das Auffinden von einzelnen Informationen in einer HTML-Seite. Um eine bestimmte Information in einer HTML-Seite zu finden, können verschiedene Techniken verwendet werden, die von festen IDs über CSS-Klassen und XPath bzw. CSS-Selektoren bis hin zu komplexer Mustererkennung reichen.

Zentrales Element in der Analyse einer Webseite ist der HTML-Quelltext, der beim Einlesen, mit einer darauf spezialisierten Programmierbibliothek, in eine baumartige Datenstruktur überführt wird. Diese Datenstruktur wird auch *DOM* (Document Object Model) genannt. Im DOM kann mit verschiedenen Techniken nach Elementen gesucht werden.

Dieser Abschnitt gliedert sich wie folgt:

- Elementtypen
- HTML-IDs
- CSS-Klassen
- XPath oder CSS-Selektoren
- Mustererkennung
- Erkenntnisse

## Elementtypen

HTML definiert eine ganze Reihe von Elementtypen, mit denen die Struktur einer Webseite gegliedert wird. Beispiele sind `div`, `a`, `table`, `footer`. In einigen Fällen reicht allein der Typ aus, um Elemente mit gesuchten Informationen oder Navigationsschaltflächen für die Traversierung eindeutig zu identifizieren.

## HTML-IDs

Ein Element im DOM anhand seiner ID zu finden, ist die einfachste und dabei eine recht robuste Technik. In HTML können einzelne Elemente mit einem Identifikator versehen werden, der innerhalb der Seite eindeutig ist. Dazu dient das Attribut `id` (vergl. [2]). HTML-IDs können u. a. dafür genutzt werden CSS-Styles anzuwenden, deshalb werden sie regelmäßig in Webseiten genutzt. Allerdings eignet sich die ID naturgemäß nur für Elemente die in jedem Fall nur ein einziges Mal in einer Webseite vorkommen.

## CSS-Klassen

Soll in einer Webseite eine Gruppe von Elementen in einer gleichen Weise formatiert werden, können CSS-Klassen genutzt werden um beliebige Untermengen von Elementen in einer Webseite mit CSS-Attributen zu verknüpfen (vergl. [3]). Diese CSS-Klassen können auch beim Screen Scraping zur Identifikation von Elementen im DOM genutzt werden.

## XPath oder CSS-Selektoren

Viele Programmierbibliotheken zur Analyse und Manipulation von HTML-Seiten stellen eine *DSL* (Domain Specific Language) zur Verfügung, um kontextabhängig

Elemente im DOM auszuwählen. Dazu erlauben sie die Kombination von allen bisher beschriebenen Identifikatoren in einem Pfadmuster. Typische Vertreter sind XPath (vergl. [4]), welches aus der Welt der XML-Techniken stammt und CSS-Selektoren (vergl. [5]), welche aus dem Sprachstandard für Web-Seiten-Formatierung stammt.

## Mustererkennung

Ist die Struktur einer Webseite vor dem Screen Scraping nicht detailliert bekannt, müssen andere Analysemethoden in Betracht gezogen werden. Das Navigationsmenü einer Webseite z. B. kann mit Hilfe von verschachtelten `div`-Elementen, durch ein `ul`-Element oder auch durch ein `table`-Element kodiert werden. Optional kann die Navigationsstruktur in ein HTML5 `nav`-Element eingebettet werden. Um auf einer unbekannten Website das Hauptmenü zu finden, reichen XPath- und CSS-Selektoren deshalb nicht aus.

Auf modernen Websites, die mit dem Ziel der Barrierefreiheit entworfen wurden, können evtl. WIA-ARIA-Attribute helfen, Strukturen in Webseiten zu erkennen (vergl. [6]). Diese Attribute haben u. a. das Ziel, Webseiten für Menschen mit Sehbehinderungen zugänglich zu machen. Für diese Zielgruppe bereiten spezielle Programme oder Browser-PlugIns eine Webseite derart auf, dass sie z. B. mit Hilfe einer Braille-Zeile navigiert werden kann. Eine ähnliche Analyse könnte auch beim Screen Scraping eingesetzt werden.

Die Informationsextraktion aus nahezu beliebigen Websites ist ein aktives Forschungsfeld. Es existieren eine ganze Reihe Veröffentlichungen in diesem Gebiet (vergl. [7–10]). Derartige Algorithmen versuchen typische Strukturen in Webseiten zu identifizieren, ohne ihre genaue Kodierung zu kennen. Auf eine detaillierte Betrachtung dieser Möglichkeiten wird an dieser Stelle verzichtet.

## Erkenntnisse

Programmierbibliotheken für die HTML-DOM-Analyse können helfen, die Implementierung einfach zu halten und dennoch einen hohen Grad an Robustheit zu erzielen.

Die folgende Tabelle enthält eine Auswahl an beliebten Programmierbibliotheken für das Parsen und Analysieren von HTML in verschiedenen Programmiersprachen.

| Name | Sprache | Website |
|---|---|---|
| Beautiful Soup | Python | https://www.crummy.com/software/BeautifulSoup/ |
| DOM | PHP | http://php.net/manual/en/book.dom.php |
| QueryPath | PHP | http://querypath.org/ |
| jsdom | JavaScript | https://www.npmjs.com/package/jsdom |
| TagSoup | Java | http://vrici.lojban.org/~cowan/XML/tagsoup/ |
| clj-tagsoup | Clojure | https://github.com/nathell/clj-tagsoup |
| Html Agility Pack | C# / VB.NET | http://html-agility-pack.net/ |
| CsQuery | C# / VB.NET | https://github.com/jamietre/CsQuery |
| Tidy | C++ | http://www.html-tidy.org/ |
| Xerces | C++ | http://xerces.apache.org/xerces-c/ |
| tagsoup | Haskell | https://hackage.haskell.org/package/tagsoup |

Bei CSS- und XPath-Selektoren sollte eine Balance zwischen spezifisch und generisch gefunden werden. Um ein Element in einer Webseite mit einem Selektor zu beschreiben, kann entweder der gesamte Pfad der Elternelemente evtl. mit Element-Typ und CSS-Klasse spezifiziert werden, was zu einem sehr spezifischen Selektor führt. Oder aber es werden nur die unbedingt notwendigen Pfadelemente im Selektor spezifiziert. In dem einen Fall führen schon kleine Änderungen in der Webseite dazu, dass der Selektor das gesuchte Element nicht mehr findet. In dem anderen Fall kann es schnell passieren, dass der Selektor unbeabsichtigt nicht gesuchte Elemente als Ergebnis liefert. Es muss also das richtige Mittelmaß bei der Formulierung der Selektoren gefunden werden.

Es hat sich bewährt die CSS- oder XPath-Selektoren, die bei der Analyse eingesetzt werden, mit aussagekräftigen Namen zu versehen und in eine Konfigurationsdatei auszulagern. Dies hat den Vorteil, dass im Falle einer kleinen Veränderung in der Webseitenstruktur, lediglich die Konfigurationsdatei angepasst werden muss, und nicht der Programm-Code.

Wurde eine Webseite erkennbar durch ein Server-seitiges Framework generiert, kann es helfen, sich deren Komponenten und die üblicherweise produzierten HTML-Strukturen genauer anzusehen. Es kann nützlich sein, spezialisierte Funktionen zu implementieren, welche sich die durch das Framework produzierten Muster

in der Webseitenstruktur zunutze machen. Z. B. könnte eine Funktion implementiert werden, welche eine immer wieder verwendete Struktur für Pagination-Links erkennt und die URLs der Einzelseiten zurückgibt (vergl. Abschnitt Pagination).

# Herausforderungen

Beim Screen Scraping begegnet man einer Reihe von Herausforderungen, die dadurch entstehen, dass man eine Schnittstelle "missbraucht". Konkret verwendet man ein Datenaustauschformat (HTML) und eine Informationsstruktur (Aufbau und Navigationsstruktur der Website), die für die grafische Präsentation von Inhalten für einen menschlichen Konsumenten und die interaktive Exploration optimiert sind, aber nicht unbedingt für den Austausch von strukturierten Informationen zwischen maschinellen Systemen.

Viele Aspekte des Datenaustauschs bei Websites sind daher suboptimal für die automatisierte maschinelle Weiterverarbeitung der enthaltenen Informationen. Diese fangen mit der Kommunikation von Fehlersituationen an und erstrecken sich bis hin zur Einbettung eigentlich strukturierter Informationen in unstrukturiertem Fließtext.

- De-normalisierte Präsentation
- Pagination
- Abfragebegrenzungen
- Passwortgeschützte Bereiche
- Fehlende Benachrichtigung über Datenänderungen
- Aktive Inhalte
- Identifikatoren

## De-normalisierte Präsentation

Eine Darstellung, bei der jede Entität mit all ihren Attributen auf mindestens einer Seite vollständig präsentiert wird, soll als *normalisierte Darstellung* bezeichnet werden. Die Erfassung von Informationen auf einer Website, welche eine normalisierte Darstellung verwendet, ist verhältnismäßig einfach, weil für die Rekonstruktion einer Entität immer nur genau eine Webseite erforderlich ist.

Bei der Präsentation von Daten in einer Website, kommt es jedoch häufig vor, dass eine Entität auf mehreren Seiten in unterschiedlichem Detailgrad oder mit einer variierenden Teilmenge von Attributen dargestellt wird. Die Ursache dafür sind üblicherweise wahrnehmungspsychologische Aspekte und Erkenntnisse im Bereich der Gestaltung von Mensch-Maschine-Schnittstellen, die den Aufbau der Website beeinflussen. Auch Optimierungen um Ressourcen zu sparen, können zu einer Aufteilung von Entitäten führen. Wenn z. B. das Ausliefern einer Detailseite wesentlich mehr Server-Leistung in Anspruch nimmt als das einer Übersichtsseite. Solch eine Darstellung soll als *de-normalisierte Präsentation* bezeichnet werden.

Bei der Erfassung von Entitäten, die de-normalisiert präsentiert werden, kann man unterschiedliche Strategien verfolgen.

1. **Erfassung von Teilsichten der Entitäten, die der Darstellung in den Webseiten entsprechen**
   Hierbei wird jede Webseite individuell betrachtet und die enthaltenen Informationen direkt in einer Datenbank gespeichert. Ist die Datenbank relational, wird für jede *Facette* (Teilsicht einer Entität) eine eigene Tabelle als Vorstufe verwendet. Erst wenn alle Facetten einer Entität vollständig zusammengetragen wurden, wird die Entität in ihrer Zieltabelle zusammenhängend gespeichert.
2. **Erfassung von vollständigen Entitäten durch die Kombination der Inhalte mehrerer Webseiten**
   Bei dieser Strategie werden alle Webseiten, die für die Rekonstruktion einer Entität notwendig sind, direkt hintereinander abgerufen. Die extrahierten Informationen werden im Arbeitsspeicher gehalten und zu einer Entität zusammengefügt. Ist eine Entität vollständig zusammengetragen, wird sie in einer Datenbanktabelle gespeichert.

Der Vorteil der ersten Strategie ist, dass der Prozess jeder Zeit abgebrochen und später wieder fortgeführt werden kann. Sie ist dementsprechend robust gegen Ausfälle der Netzwerkverbindungen oder Störungen auf dem Server oder dem Client. Eine Website muss i. d. R. nur ein einziges Mal abgerufen werden, da alle Informationen die extrahiert werden können sofort in der Datenbank gesichert werden. Der Nachteil sind die zusätzlichen Tabellen, die u.U. redundante Informationen enthalten.

Der Vorteil der zweiten Strategie ist, dass für jeden Typ von Entität nur eine Tabelle benötigt wird. Nachteilig ist jedoch, dass Webseiten mehrfach aufgerufen werden

müssen, sollte die Erfassung durch eine Störung unterbrochen werden. Denn bei einer Unterbrechung der Erfassung, die einen Neustart des erfassenden Programms notwendig macht, gehen die im Arbeitsspeicher vorgehaltenen Informationen von unvollständigen Entitäten verloren.

Bei einer geringen Menge von zu erfassenden Informationen, ist die zweite Strategie wahrscheinlich die effizientere Wahl. Sollen jedoch größere Mengen an Informationen erfasst werden, und das womöglich in regelmäßigen Abständen über einen längeren Zeitraum, ist die erste Strategie empfehlenswert.

### Beispiel

Eine Liste von Unternehmen wird in drei Darstellungsformen auf einer Website präsentiert. Als Einstieg dient eine tabellarische Übersicht aller Unternehmen. Wobei jedoch nur der Name und das Bundesland aufgeführt sind. Ein Link je Unternehmen, dessen Zieladresse eine eindeutige ID enthält, führt zu einer Detailseite. Die Detailseiten enthalten die meisten interessanten Attribute wie *PLZ*, *Ort*, *Anschrift*, *Telefon*. Allerdings führt von jeder Detailseite ein Link auf eine weitere Seite mit dem Unternehmensprofil, das u. a. eine Kontaktperson für das Unternehmen aufführt. Eine Entität (ein Unternehmen) wird also in drei unterschiedlichen Teilsichten de-normalisiert präsentiert.

Die Zieltabelle für Unternehmen könnte wie folgt aussehen:

- `unternehmen`
  - `id`
  - `zeitstempel`
  - `name`
  - `plz`
  - `ort`
  - `anschrift`
  - `bundesland`
  - `telefon`
  - `kontaktperson`

Mit der ersten Strategie werden drei zusätzliche Datenbanktabellen für die Teilsichten benötigt:

- `denorm_unternehmen_info`
  - `id`

    - url
    - zeitstempel
    - name
    - bundesland
- denorm_unternehmen_detail
    - id
    - url
    - zeitstempel
    - plz
    - ort
    - anschrift
    - telefon
- denorm_unternehmen_kontakt
    - id
    - url
    - zeitstempel
    - kontaktperson

Zunächst wird die Webseite mit der Übersichtstabelle abgerufen und jedes gelistete Unternehmen in `denorm_unternehmen_info` gespeichert. Dabei wird mit einem Zeitstempel der Zeitpunkt festgehalten, wann die Informationen abgerufen wurden. Für die Spalte `id` wird eine neue ID gebildet. In der Spalte `url` wird dokumentiert, mit welcher URL die Informationen abgerufen wurden. Anschließend wird die Tabelle `denorm_unternehmen_info` nach allen bekannten Unternehmen befragt. Für jedes Unternehmen, für das noch kein Eintrag in `denorm_unternehmen_detail` existiert, wird die Detailseite abgerufen. Die daraus extrahierten Informationen werden in der Tabelle `denorm_unternehmen_detail` gespeichert. Dabei wird für das Feld `id` die generierte ID aus der Tabelle `denorm_unternehmen_info` verwendet. Auch hier wird URL und Zeitpunkt in `url` und `zeitstempel` dokumentiert. Für die dritte Teilsicht `denorm_unternehmen_kontakt` wird genauso verfahren. Als letzten Schritt werden alle drei Tabellen zusammengeführt und in die Tabelle `unternehmen` übertragen. Dabei wird mit einem Zeitstempel der Zeitpunkt festgehalten, wann die Informationen zusammengeführt wurden.

Bei dieser Strategie kann der Prozess im Prinzip jederzeit unterbrochen und neugestartet werden, ohne dass Webseiten mehrfach abgerufen werden müssen. Der Ver-

lauf des Seitenabrufs kann über die einzelnen Zeitstempel genau nachverfolgt werden. Und die Tabelle `unternehmen` beinhaltet zu jedem Zeitpunkt nur vollständig erfasste Unternehmen.

Es stellt sich die Frage, warum für die erste Strategie zusätzliche Tabellen benötigt werden. Die Tabelle `unternehmen` könnte ja auch schrittweise vervollständigt werden. Das klingt verlockend bringt jedoch Probleme mit sich. I. d. R. gibt es keine Garantie dafür, dass für alle Entitäten jede Teilsicht verfügbar ist. So könnte z. B. ein Unternehmen eine Profilseite ohne Kontaktperson besitzen. Würde man nur die Tabelle `unternehmen` verwenden, müsste man aus einem leeren Feld `kontaktperson` schließen, dass die Profilseite noch nicht abgerufen wurde. Der Abruf der Profilseite steht also noch aus. Enthält die Profilseite aber keine Kontaktperson, gibt es keine elegante Möglichkeit, zu dokumentieren, dass die Profilseite sehr wohl schon abgerufen wurde, jedoch kein Ergebnis lieferte. Durch die zusätzliche Tabelle `denorm_unternehmen_kontakt` ist es möglich einen Eintrag mit `web_id` und `zeitstempel` zu speichern, der ein leeres Feld `kontaktperson` besitzt. Damit ist eindeutig dokumentiert, dass die Profilseite abgerufen wurde, es aber keine Kontaktperson gibt. Ein weiterer Nachteil würde darin bestehen, dass für einen Konsumenten der Tabelle `unternehmen` nicht ersichtlich ist, welche Unternehmen vollständig erfasst wurden und welche nicht.

Mit der zweiten Strategie wird zunächst die Webseite mit der Übersichtstabelle abgerufen und die Liste der Unternehmen im Arbeitsspeicher gehalten. Für jedes Unternehmen werden sowohl die Detailseite als auch das Unternehmensprofil abgerufen und der Datensatz für das Unternehmen anschließend mit einer neuen eindeutigen ID in die Tabelle `unternehmen` gespeichert. Dabei wird mit einem Zeitstempel der Zeitpunkt festgehalten, wann die Informationen zusammengeführt wurden.

Wird bei dieser Strategie der Prozess unterbrochen und neugestartet, muss in jedem Fall die Übersichtsseite erneut abgerufen werden und evtl. auch die Detailseite für ein Unternehmen. Der genaue Verlauf des Seitenabrufs kann nicht nachverfolgt werden, da nur der Zeitpunkt für das Zusammenführen der Informationen gespeichert wird. Auch bei dieser Strategie beinhaltet die Tabelle `unternehmen` zu jedem Zeitpunkt nur vollständig erfasste Unternehmen.

# Pagination

Pagination ist eine Technik, bei der eine Menge von Elementen auf mehrere Seiten verteilt wird. Das hat den Zweck, dass z. B. bei der Beantwortung einer Suchanfrage nicht sofort alle Ergebnisse gefunden, übertragen und dargestellt werden müssen, sondern erst wenn der Benutzer tatsächlich an einer größeren Anzahl von Ergebnissen interessiert ist. In vielen Szenarien wird auch eine Sortierung der Ergebnisse vorgenommen, welche die Wahrscheinlichkeit erhöht, dass sich die Elemente, für die sich der Benutzer interessiert, unter den zuerst präsentierten Ergebnissen befinden.

Üblicherweise wird eine fest vorgegebene oder vom Benutzer wählbare maximale Anzahl von Elementen verwendet, um die Elemente in Untermengen aufzuteilen. Der Benutzer bekommt, wenn er zu einer Ansicht mit mehreren Elementen navigiert oder eine Suchanfrage abschickt, nur die erste Ergebnisseite, bzw. Untermenge von Elementen, präsentiert und muss Navigationsschaltflächen wie z. B. *Weiter* oder *Nächste Seite* verwenden, um zur nächsten Untermenge von Elementen zu navigieren.

In vielen Fällen werden neben *Zurück* und *Weiter* auch nummerierte Navigationsschaltflächen angezeigt, die es dem Benutzer erlauben, direkt zur N-ten Untermenge zu springen. Dabei sind jedoch wiederum nicht immer alle Untermengen mit einer nummerierten Navigationsschaltfläche vertreten, sondern oft nur eine begrenzte Anzahl vor und nach der aktuell angezeigten Untermenge.

Um alle Elemente einer paginierten Menge mit Screen Scraping zu erfassen, gibt es zwei mögliche Strategien: Den Kettenabruf und den Direktabruf.

## Kettenabruf

Der Kettenabruf folgt der Navigationsstrategie der explorativen Traversierung. Auf jeder aktuell vorliegenden Seite, beginnend mit der ersten, wird nach der Navigationsschaltfläche gesucht, welche zu einer Seite mit der nachfolgenden Untermenge von Elementen führt. Ist die Schaltfläche vorhanden und aktiv, kann davon ausgegangen werden, dass noch weitere Elemente existieren. Es wird demzufolge die verlinkte Seite abgerufen und gleichsam untersucht.

### Vorteile

- Einfacher Algorithmus
- Unabhängigkeit vom konkreten Aufbau der URL

### Nachteile

- Abhängigkeit von Navigationsschaltflächen

Der Kettenabruf ist i. d. R. besser geeignet als der Direktabruf, wenn die Navigationsschaltfläche leicht identifiziert werden kann, die Gesamtanzahl der Elemente nicht bekannt ist und es nicht auf eine hohe Erfassungsgeschwindigkeit ankommt.

## Direktabruf

Der Direktabruf folgt der Navigationsstrategie des Reverse Engineering von URLs. Die URL der paginierten Ergebnismenge wird auf Parameter für die Pagination untersucht und es werden direkt abgeleitete URLs für die verschiedenen Untermengen gebildet. Sollte ein Parameter für die Anzahl der Elemente je Untermenge existieren, bietet es sich an, diesen Parameter zu maximieren, um die Anzahl der notwendigen Aufrufe, für das Erfassen der gesamten Menge von Elementen, zu minimieren.

### Vorteile

- Parallelisierung möglich
- Mehr Kontrolle über die Abfrage-Parameter (z. B. mehr Elemente pro Seite)

### Nachteile

- Abhängigkeit vom konkreten Aufbau der URL
- Ohne Informationen über Gesamtanzahl von Elementen ineffizient

Der Direktabruf kann sowohl verwendet werden, wenn die Gesamtanzahl der gesuchten Elemente bekannt ist, als auch wenn diese unbekannt ist. Wenn die Gesamtanzahl der Elemente bekannt ist, kann, unter Zuhilfenahme der maximalen Anzahl von Elementen je Seite, direkt die benötigte Menge von URLs gebildet werden. Ist die Gesamtanzahl jedoch nicht bekannt, müssen die URLs inkrementell gebildet und iterativ abgerufen werden. Als Abbruchkriterium kann der Erhalt einer Fehlerseite, der HTTP-Status-Code 404 oder der Erhalt einer Seite ohne Elemente dienen.

Der Direktabruf ist i. d. R. besser geeignet als der Kettenabruf, wenn die URL einfach aufgebaut ist, sich über die URL-Parameter mehr Möglichkeiten als die Navigationsschaltflächen bieten und wenn eine besonders effiziente Abfrage wichtig ist.

Wenn das Screen-Scraping-System auf Effizienz und Erfassungsgeschwindigkeit optimiert werden soll, ist es wichtig, die Abfrage mit dem Betreiber der Webseite abzustimmen, da z. B. durch eine parallelisierte und hochfrequente Abfrage der Betrieb bzw. die Erreichbarkeit der Webseite beeinträchtigt werden kann.

## Abfragebegrenzungen

Auf einigen Websites steht eine Suchfunktion zur Verfügung, welche jedoch nur eine begrenzte Anzahl von Ergebnissen liefert. Erfolgt die Suche nur mit einem einzigen Kriterium, kann man gegen diese Begrenzung nichts unternehmen. Stehen jedoch mehrere Kriterien zur Einschränkung der Suche zur Verfügung, ist es prinzipiell möglich, den Suchraum durch geschickte Wahl von zusätzlichen Kriterien derart zu partitionieren, dass mit mehreren Suchanfragen doch alle Ergebnisse zurückgegeben werden.

Mathematisch betrachtet, wird dabei ein mehrdimensionaler Raum derart in Partitionen aufgeteilt, dass die Vereinigung aller Partitionen den gesamten Raum abdeckt. Die Ergebnismenge für jede Partition muss dabei kleiner als die Abfragegrenze sein. Werden die Partitionen jedoch zu klein gewählt, werden unnötig viele Suchanfragen erzeugt.

Bei einigen Arten von Kriterien sind überlappende Partitionen notwendig, um den gesamten Suchraum abzudecken. Um solche Abfragen effizient zu gestalten, ist es erstrebenswert, die Überlappung der Partitionen zu minimieren. Des Weiteren müssen bei Ergebnissen aus überlappenden Partitionen mögliche Duplikate entfernt werden.

### Beispiel

Das folgende Beispiel soll den Einsatz von Partitionen veranschaulichen. Ein Suchformular für Veranstaltungen in Deutschland besitzt die folgenden Auswahlfelder:

- *Name*
- Zeitraum mit *Datum von* und *Datum bis*
- *PLZ* und *Umkreis* in Kilometern

Zielstellung ist es, alle Veranstaltungen zu erfassen, die im Juli 2017 stattgefunden haben. Wird als Zeitraum 01.07.2017 bis 31.07.2017 eingegeben, ergibt die Suche genau 200 Treffer. Das Datum der ersten 50 Veranstaltungen ist jedoch einheitlich der 01.01.2017. Das legt die Vermutung nahe, dass nicht alle Suchergebnisse ausgegeben werden, sondern die Trefferanzahl immer auf 200 begrenzt ist.

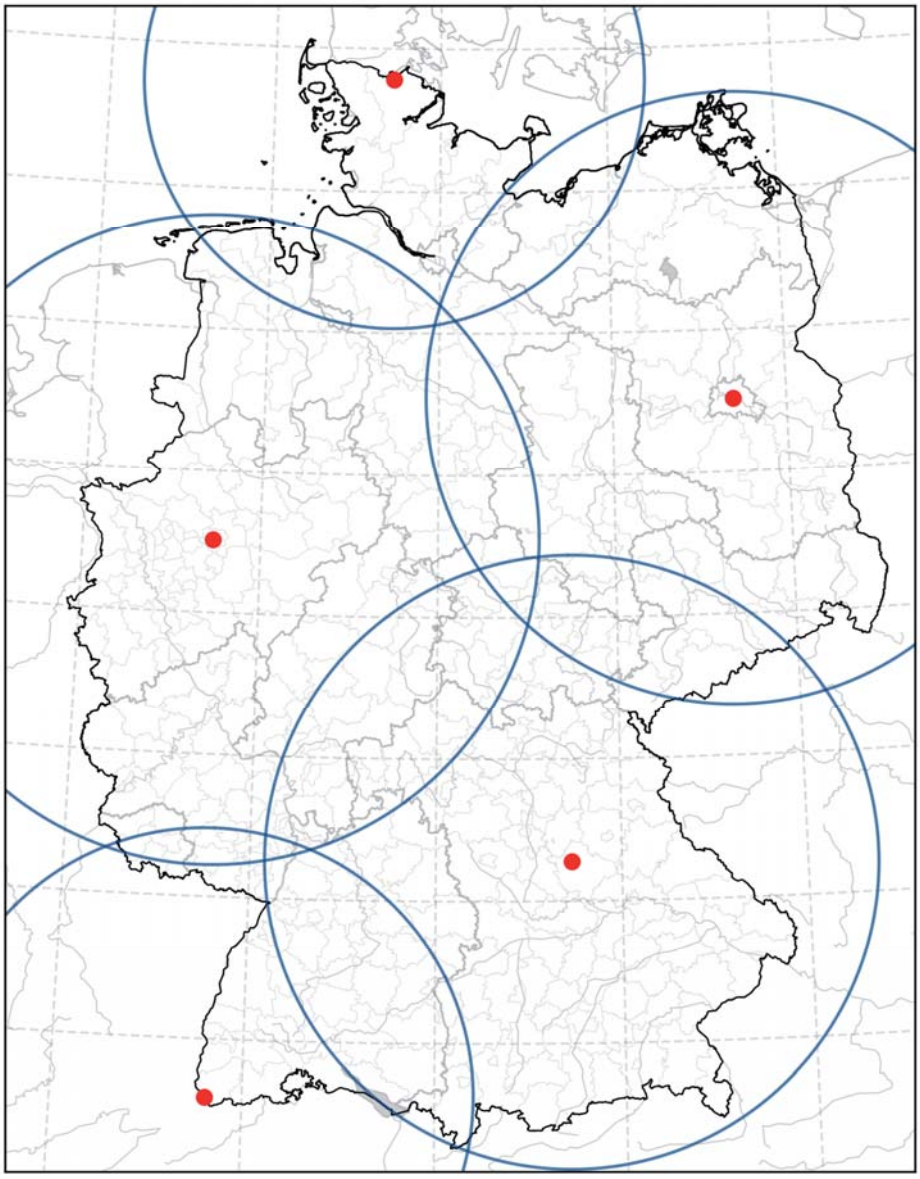
Nun kann der Suchraum mit Hilfe der Suchkriterien partitioniert und für jede Partition eine eigene Abfrage gestartet werden. In diesem Fall bietet sich das Datum als erste Stufe für eine Partitionierung an. Anstatt eine Suchanfrage für den gesamten Zeitraum zu verwenden, kann für jeden Tag eine eigene Suchanfrage gestellt werden. Bei der Partitionierung mittels des Datums sind die Partitionen disjunkt solange sich die als Kriterium angegebenen Zeiträume nicht überlappen. Zunächst scheint das Problem der Abfragebegrenzung gelöst. Um alle Veranstaltungen im Juli zu erfassen, wurden 31 Abfragen durchgeführt. Doch eine nähere Prüfung zeigt, dass es drei Tage gibt, an denen die Suche wieder genau 200 Ergebnisse liefert. Folglich reicht die Partitionierung mittels des Datums noch nicht aus.

Als zweite Stufe für eine Partitionierung bieten sich die Felder *PLZ* und *Umkreis* an. Da mit einer Postleitzahl als Zentrum und einem Radius für die Ausdehnung nur eine kreisförmige Region beschrieben werden kann, muss eine Menge von Kreisen gefunden werden, die Deutschland vollständig abdeckt – in der sich die Kreise jedoch möglichst wenig überlappen (vergl. Abbildung 1). Zusätzlich muss das jeweilige Zentrum der Kreise an einem geografischen Ort liegen, der mit einer Postleitzahl beschrieben werden kann. Die angestrebte Größe der Kreise ergibt sich aus der Anzahl von möglichen Suchergebnissen in einer Region. Als Ausgangspunkt könnte Deutschland durch fünf Kreise abgedeckt werden, die jeweils durch eine Kombination von *PLZ* und *Umkreis* beschrieben sind.

| PLZ | Ort | Radius |
|---|---|---|
| 10115 | Berlin | 240 |
| 24939 | Flensburg | 195 |
| 44147 | Dortmund | 255 |
| 79576 | Weil am Rhein | 210 |
| 92318 | Neumarkt in der Oberpfalz | 240 |

Sollte der Fall eintreten, dass auch mit dieser Partitionierung die Abfragegrenze erreicht wird, muss eine Aufteilung in kleinere Regionen gefunden werden. Da durch

die Natur der kreisförmigen Partitionen Überlappungen notwendig sind, müssen die Suchergebnisse, nach dem Erfassen aller Partitionen, von Duplikaten befreit werden.



**Fünf kreisförmige Regionen, die Deutschland vollständig abdecken**

## Passwortgeschützte Bereiche

Wenn Inhalte von Seiten erfasst werden sollen, welche durch eine Zugangskontrolle geschützt sind, sind die folgenden Aspekte zu berücksichtigen:

Zunächst ist die Legalität des Erfassens besonders zu hinterfragen. Der Schutz der Inhalte durch einen Authentifizierungsschritt kommuniziert die Intention, dass die Inhalte nur einem begrenzten Personenkreis zugänglich gemacht werden sollen. Daher ist zu prüfen, ob das Erfassen der Daten oder die beabsichtigte Weiterverwendung der Daten den berechtigten Interessen desjenigen zuwiderläuft, der die Daten über die Website bereitstellt.

Des Weiteren ist die technische Umsetzung der Zugangskontrolle im Einzelfall zu berücksichtigen. Es existiert ein breites Spektrum an technischen Lösungen für die Zugangskontrolle in Websites. Grundsätzlich kann man zustandsbehaftete und zustandslose Lösungen unterscheiden.

Zustandsbehaftete Lösungen erzeugen bei erfolgreichem Login einen Datensatz auf dem Server, welcher dokumentiert, dass dem Benutzer Zugang gewährt wurde. Dieser Datensatz ist verknüpft mit der aktuellen Sitzung (*Session*) des Benutzers. Die Sitzung wird dem Benutzer durch eine temporäre ID zugeordnet, welche bei jeder Anfrage mitgeschickt werden muss.

Zustandslose Lösungen stellen dem Benutzer bei erfolgreichem Login ein Token aus, welches Informationen beinhaltet, die ausreichen, damit der Server ein valides Login wiedererkennen kann. Tokens arbeiten i. d. R. mit einer digitalen Unterschrift und einem Verfallsdatum. Auch Tokens müssen nach dem Login bei jeder nachfolgenden Anfrage mitgeschickt werden.

Zustandslose Lösungen haben eine Reihe von Vorteilen für die Server-Architektur, machen aber im Kontext des Screen Scrapings keinen wesentlichen Unterschied aus. In beiden Fällen muss dem Erfassen von Informationen eine Login-Anfrage vorangestellt werden. Im Falle eines erfolgreichen Logins muss ein Teil der Antwort aufbewahrt und bei jeder nachfolgenden Anfrage mitgeschickt werden. Der einzige Unterschied besteht darin, dass bei zustandsbehafteten Lösungen nach dem Erfassen der Daten eine Logout-Anfrage geschickt werden sollte, damit der Server die Möglichkeit bekommt die Sitzung zu zerstören. Dies ist bei zustandslosen Lösungen nicht notwendig, da das Verwerfen des Tokens einem Logout gleichkommt.

Es gibt eine Vielzahl von Varianten bei der Übermittlung der Sitzungs-ID bzw. dem Token bei Anfragen. U. a. sind die folgenden Wege üblich:

- Übergabe als Abfrageparameter in der URL
- Übergabe in einem Cookie
- Übergabe als standardisiertes oder proprietäres HTTP-Header-Feld (z. B. `Authorization`)
- Übergabe als Formular-kodiertes Feld in POST-Anfragen
- Übergabe als Teil einer kodierten Datenstruktur im Körper der Anfrage (z. B. XML oder JSON)

## Fehlende Benachrichtigung über Datenänderungen

Soll ein Datenbestand für eine Anzahl von Entitäten für die Weiterverarbeitung vorgehalten werden, können die Informationen der Entitäten mittels Screen Scraping aus einer Website extrahiert werden. Werden die Informationen der Entitäten auf der Website jedoch aktualisiert, veraltet der durch Screen Scraping abgerufene Datenbestand.

Ein einfacher, wenn auch nicht immer praktikabler Weg ist es, das Screen Scraping immer nur dann auszuführen, wenn die Daten in aktueller Form benötigt werden. Das wird als *synchrones Screen Scraping* bezeichnet. Ist der Datenbestand entsprechend groß, das Screen Scraping zeitaufwändig und werden die Daten häufig weiterverwendet, muss das Screen Scraping von der Weiterverarbeitung zeitlich entkoppelt werden. Dann sind regelmäßige Abrufe der Website erforderlich, um die Daten aktuell zu halten. Das wird als *asynchrones Screen Scraping* bezeichnet. Welches Intervall angebracht ist und welche Webseiten abgerufen werden müssen, richtet sich dabei nach der Natur der Entitäten.

### Beispiel

Eine Website stellt eine Liste mit Veranstaltungen zur Verfügung. Sie präsentiert dabei bereits bekannte Veranstaltungen in der Zukunft, und die Veranstaltungen aus der Vergangenheit, die nicht länger als 12 Monate zurückliegen. Die Veranstaltungen sind auf mehrere Webseiten verteilt, wobei jede Webseite die Veranstaltungen für jeweils einen Monat enthält. Es kommt regelmäßig vor, dass Veranstaltungen abgesagt werden und dann aus der Liste verschwinden.

Soll diese Liste durch asynchrones Screen Scraping für die Weiterverarbeitung erfasst werden, können die Webseiten für die verschiedenen Monate mit unterschiedlichen Intervallen abgefragt werden. Monate die in der Vergangenheit liegen, brauchen nur einmal abgefragt werden. Denn es ist nicht zu erwarten, dass sich die Informationen zu Veranstaltungen in der Vergangenheit ändern. Anders sieht es mit dem aktuellen Monat und den Monaten in der Zukunft aus. Monate die weit in der Zukunft liegen, müssen nur relativ selten aktualisiert werden. Der aktuelle Monat und die in naher Zukunft liegenden Monate hingegen sollten jeden Tag aktualisiert werden.

## Aktive Inhalte

Viele moderne Websites verwenden aktive Inhalte, wie z. B. JavaScript, um die Interaktivität und den Komfort für menschliche Besucher zu steigern. Dabei kann die Aufgabe der aktiven Anteile der Website von einfachen Animationen bis hin zu wesentlichen Funktionalitäten der Website reichen. Eine Extremform sind die sogenannten Single-Page-Websites (vergl. [11]).

Immer dann, wenn gesuchte Informationen nur dann angezeigt werden, wenn aktive Inhalte ausgeführt werden, sind sie für das Screen Scraping relevant. Sobald eine Website von JavaScript Gebrauch macht, ist es deshalb i. d. R. notwendig die Rolle des JavaScript-Codes in der Website zu untersuchen. Ein einfacher Test ist, JavaScript im Browser zu deaktivieren und anschließend zu überprüfen, ob die Website noch die gesuchten Informationen darstellt (vergl. [12], [13]).

Sollte JavaScript-Code verantwortlich dafür sein, dass die gesuchten Informationen angezeigt werden, kann das unterschiedliche Konsequenzen für das Screen Scraping haben. Im Folgenden sollen einige mögliche Fälle beschrieben werden.

1. Sichtbarkeitsänderungen
2. AJAX mit HTML-Inhalten
3. AJAX mit strukturierten Inhalten (z. B. JSON oder XML)
4. Berechnung von Inhalten im Browser

In einigen Fällen enthält der HTML-Code bereits die gesuchten Informationen, er ist jedoch mittels CSS ausgeblendet und wird erst nach dem Betätigen einer Schaltfläche, die JavaScript-Code ausführt, sichtbar. In diesen Fällen kann der JavaScript-

Code ignoriert werden, da die Informationen beim Screen Scraping, unabhängig vom referenzierten CSS, direkt aus dem HTML extrahiert werden können.

Wird das JavaScript dazu eingesetzt, dynamisch HTML-Code-Blöcke nachzuladen (AJAX), welche die gesuchten Informationen enthalten, können die URLs untersucht und nachgebildet werden, mit denen die HTML-Schnipsel nachgeladen werden. Anschließend werden die HTML-Schnipsel zum Ziel für das Screen Scraping.

Wird das JavaScript dazu eingesetzt, dynamisch strukturierte Daten nachzuladen (AJAX), welche die gesuchten Informationen enthalten, verlässt die Lösung beinahe den Bereich des Screen Scraping. Denn offensichtlich stellt der Web-Server die gesuchten Informationen über entsprechende URLs in strukturierter Form, z. B. JSON oder XML, zur Verfügung und das Analysieren des HTML-Codes entfällt.

Eine Sonderrolle nimmt der Fall ein, dass die gesuchten Informationen gar nicht vom Web-Server geladen werden, sondern durch ein JavaScript-Programm im Browser berechnet werden. In einem solchen Fall könnte es möglich sein, den JavaScript-Code zu extrahieren und offline, vollständig ohne den Kontakt zum Web-Server, auszuführen. Das Format der Datenausgabe kann in diesem Fall durch Anpassung des JavaScript-Codes selbst bestimmt werden. Wird der Quellcode des JavaScript-Programms in einem solchen Szenario eingesetzt und evtl. angepasst, müssen die Lizenzbestimmungen beachtet werden, unter denen das JavaScript-Programm veröffentlicht wird.

Ist der Aufbau der Website komplex und wird JavaScript an vielen Stellen eingesetzt, so dass ein Reverse Engineering der Abläufe bis hin zum Ursprung der gesuchten Informationen zu aufwändig erscheint, kann sich der Einsatz von *PhantomJS* lohnen (vergl. [14]). PhantomJS ist ein Browser der keine grafische Ausgabe besitzt, jedoch durch eigenen JavaScript-Code gesteuert werden kann. Mit PhantomJS ist es möglich, Webseiten aufzurufen und wie in einem gewöhnlichen Browser mit dem Nachladen von Ressourcen und dem Ausführen von JavaScript-Code aufzubauen. Anschließend kann eigener JavaScript-Code mit der Website interagieren, den dynamisch entstehenden DOM analysieren und die gesuchten Informationen extrahieren.

## Identifikatoren

Die Entitäten die beim Screen Scraping erfasst werden, müssen eindeutig identifizierbar sein. Das ist vor allem wichtig, wenn das Screen Scraping wiederholt durchgeführt und dabei ein bereits existierender Datenbestand aktualisiert wird. Dabei ist entscheidend, in welchem Kontext die verwendeten Identifikatoren eindeutig sind.

Werden auf einer Website Detailseiten für die Entitäten präsentiert, existiert mit hoher Wahrscheinlichkeit bereits ein Identifikator – z. B. eine Zahl oder eine alphanumerische Zeichenkette – der in der URL für die Detailseite enthalten ist. Es ist jedoch zu beachten, ob der Identifikator nur in einer Klasse von Entitäten oder für alle Entitäten eindeutig ist.

Weiterhin ist wichtig zu erkennen, ob ein Identifikator *natürlich* oder *künstlich* ist. Natürliche Identifikatoren leiten sich direkt aus den wesentlichen Eigenschaften einer Entität ab. Besitzen die gesuchten Entitäten z. B. eine Eigenschaft, in der sie sich eindeutig unterscheiden und die naturgemäß unveränderlich ist, kann diese Eigenschaft als *natürlicher Identifikator* genutzt werden. *Künstliche Identifikatoren* sind semantisch unabhängige Daten, deren wichtigste Eigenschaft die Eindeutigkeit in einem definierten Kontext ist. Das können inkrementierte Zahlen sein, zufällige Zahlen deren Eindeutigkeit bei der Vergabe gesichert wird, zufällige alphanumerische Zeichenfolgen oder UUIDs bzw. GUIDs (vergl. [15]). Ist es schwierig für die Entitäten einen Identifikator aus den Eigenschaften zu bilden, werden i. d. R. künstliche Identifikatoren eingesetzt.

Natürliche Identifikatoren besitzen den Reiz, dass sie direkt aus einer Entität abgeleitet werden können, egal woher deren Informationen stammen. Sie bergen jedoch auch das Risiko, dass sie aus einer Eigenschaft abgeleitet werden, von der angenommen wurde, dass sie sich nicht ändern kann, sich in der Zukunft aber doch ändert.

Künstliche Identifikatoren besitzen den Reiz, dass sie Entitäten stabil identifizieren, auch wenn sich deren Eigenschaften wesentlich verändern, ziehen jedoch zusätzlichen Aufwand nach sich, wenn Dubletten erkannt und evtl. zusammengeführt werden müssen. Des Weiteren ist bei künstlichen Identifikatoren unbedingt der Kontext zu beachten in dem sie eindeutig sind. In diesem Sinne gibt es schwache Identifika-

toren, die nur in einem genau definierten Kontext eindeutig sind – z. B. inkrementierte Ganzzahlen – und starke Identifikatoren wie UUIDs, die nahezu uneingeschränkt eindeutig sind.

Die im Folgenden beschriebene Strategie bietet eine hohe Sicherheit bei verhältnismäßig geringem Aufwand:

Beim Erfassen der Rohdaten werden, soweit möglich, Identifikatoren verwendet, die auch in der Website und deren URLs verwendet werden. Beim Zusammenführen der Teilinformationen zu vollständigen Entitäten werden neue künstliche Identifikatoren vergeben und in den Rohdaten nachträglich ergänzt. Dadurch ist ein Abgleich der Rohdaten mit neu erfassten Daten von der Website leicht möglich, da die Identifikatoren der Website verwendet werden können. Und auch der Abgleich der Rohdaten mit den Entitäten ist leicht möglich. Der neue künstliche Identifikator hat zusätzlich den Vorteil, dass Entitäten aus verschiedenen Quellen – z. B. mehreren Websites mit ähnlichen Inhalten – leichter zusammengeführt werden können, da keine Kollision von Identifikatoren aus unterschiedlichen Quellen auftreten können.

# Fehlerquellen

Es gibt eine ganze Reihe von Fehlerquellen, die sich bereits gut während des Entwurfs oder der Implementierung eines Screen-Scraping-Programms berücksichtigen lassen. Im Folgenden sollen die wesentlichen Fehlerquellen genannt und Möglichkeiten zur Begegnung beschrieben werden.

- Netzausfälle und Server-Fehler
- Änderungen an der Struktur der Website

## Netzausfälle und Server-Fehler

Bei länger laufenden Screen-Scraping-Abfragen kann es immer wieder vorkommen, dass auf der Netzwerkebene Unterbrechungen auftreten oder der Web-Server einen internen Fehler meldet. Deshalb ist es wichtig, ein Screen-Scraping-Programm so zu konzipieren, dass es mit Störungen bei HTTP-Anfragen umgehen kann. Zunächst muss sichergestellt werden, dass Fehler aus der Netzwerkebene korrekt abgefangen werden. Dazu gehören z. B. Namensauflösungsfehler, Zeitüberschreitungen

beim Verbindungsaufbau, unerwartete Verbindungsabbrüche, unvollständige Antworten, leere Antworten und HTTP-Statuscodes im Bereich 5xx. Dann muss eine Strategie für den Umgang mit dem Fehler implementiert werden. Zunächst sollte der Fehler in irgendeiner Form protokolliert werden. Dies ist für eine spätere Fehlersuche unerlässlich.

Anschließend kommen verschiedene mögliche Reaktionen auf eine fehlgeschlagene HTTP-Abfrage in Frage, z. B.:

1. Programm sofort beenden
2. Fehler ignorieren und zur Abfrage der nächsten Seite übergeben, falls eine weitere abzurufende URL bekannt ist
3. Sofort erneut versuchen
4. Einige Zeit warten und anschließend erneut versuchen

Wenn ein erneuter Versuch unternommen wird, kann ein Zähler inkrementiert, und der Vorgang bei einer maximalen Anzahl an Versuchen abgebrochen werden.

Wichtig ist, dass ein Fehler auf der Netzwerkebene oder ein interner Server-Fehler nicht derart interpretiert wird, dass eine abzurufende Entität nicht mehr existiert. Vielmehr ist ihr Zustand solange unbekannt, bis eine erfolgreiche Abfrage durchgeführt wurde. Eine Antwort mit einem HTTP-Statuscode 404 kann hingegen oft so interpretiert werden, dass die Entität nicht mehr existiert. Als Folge kann sie im lokalen Datenbestand gelöscht oder als gelöscht markiert werden.

## Änderungen an der Struktur der Website

Wird die Struktur oder das Design einer Website durch den Website-Betreiber geändert, ist es sehr wahrscheinlich, dass das Screen Scraping die gesuchten Informationen nicht mehr extrahieren kann. Grundsätzlich lässt sich sagen, dass Änderungen an der Struktur einer Website unvorhersehbar umfangreiche Anpassungen am Screen-Scraping-Programm erforderlich machen.

Es wäre möglich, dass die Navigationsstruktur der Website geändert wird und Unterseiten über einen anderen Pfad erreichbar werden. Dabei ändert sich nicht zwangsläufig aber möglicherweise auch die URL einer Web-Seite mit gesuchten Informationen. Als Konsequenz müssen zukünftig lediglich die URLs für das Screen Scraping anders gebildet werden.

Es wäre aber auch möglich, dass der HTML-Code neu strukturiert wird, ohne dass sich das sichtbare Design der Website wesentlich ändert – z. B. in einem sog. Refactoring. In diesem Fall kann es notwendig sein, die ganze Analysestrategie neu zu konzipieren und evtl. wesentliche Teile des Screen-Scraping-Programms neu zu schreiben.

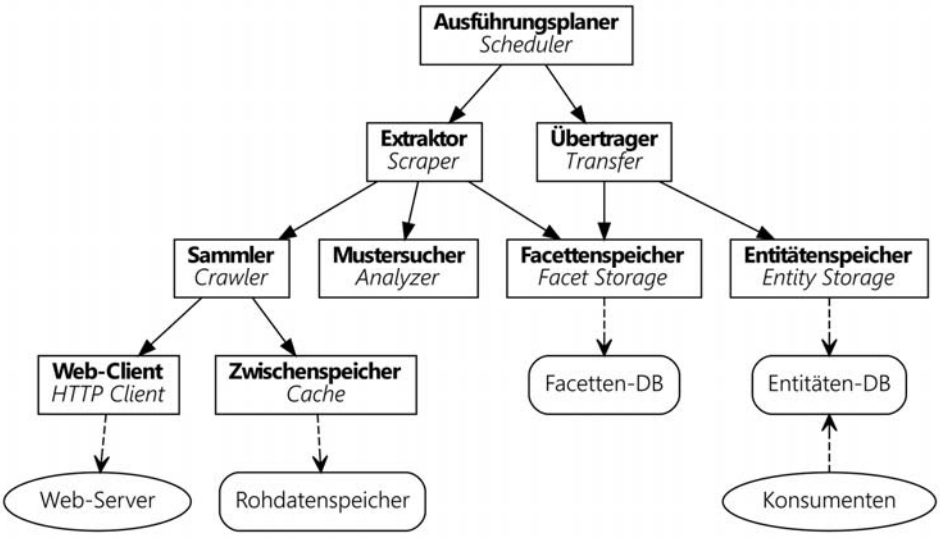
Die wichtigste Erkenntnis in diesem Kontext ist, dass Screen Scraping als regelmäßige Datenquelle nur zuverlässig funktioniert, wenn permanent ein Softwareentwickler mit den erforderlichen Kenntnissen zur Verfügung steht, um das Screen-Scraping-Programm bei Bedarf an Änderungen der Website anzupassen.

# Architektur

In diesem Abschnitt wird eine beispielhafte Architektur vorgestellt, welche die Erkenntnisse aus den vorangegangenen Abschnitten berücksichtigt. Sie kann als Vorlage verwendet werden, unabhängig von der gewählten Navigations-, Abfrage- und Analysestrategie.

Die Architektur gliedert sich in *Module* und *Fremdsysteme*. Einige Module sind generischer Natur und können in mehreren Screen-Scraping-Systemen wiederverwendet werden. Andere Module müssen für jedes Screen-Scraping-System individuell entworfen und implementiert werden, da ihre Logik stark von der Struktur der Website und den gesuchten Informationen abhängt.

- Module
- Fremdsysteme
- Allgemeine Aspekte




**Architekturübersicht**

## Module

Im Folgenden werden die einzelnen Module der Architektur beschrieben. Ziel ist es, dass jedes Modul im Wesentlichen nur eine Aufgabe wahrnimmt.

| Name | Englisch | Komponente | Typ | Strategie |
|---|---|---|---|---|
| Ausführungsplaner | Scheduler | Scheduler | generisch | Abfrage |
| Extraktor | Scraper | Scraper | individuell | Navigation |
| Sammler | Crawler | Scraper | generisch | Abfrage |
| Web-Client | HTTP Client | Scraper | generisch | Abfrage |
| Zwischenspeicher | Cache | Scraper | generisch | |
| Mustersucher | Analyzer | Scraper | individuell | Analyse |
| Facettenspeicher | Raw Storage | Scraper, Transfer | individuelles Schema | |
| Übertrager | Transfer | Transfer | individuell | |
| Entitätenspeicher | Entity Storage | Transfer | individuelles Schema | |

## Ausführungsplaner

Der *Ausführungsplaner* startet die Extraktion der Informationen aus den Web-Seiten und die Zusammenführung von Entitäten in den Entitätenspeicher. Je nach Anwendungsfall kann dies zu regelmäßigen oder zufälligen Zeitpunkten erfolgen. Seine Aufgabe beschränkt sich auf das Anstoßen des Screen Scrapings. Er implementiert den Aspekt des Abfragezeitpunkts aus der Abfragestrategie und begegnet damit der Herausforderung von fehlenden Benachrichtigungen bei Datenänderungen.

Der Ausführungsplaner ist unabhängig von den Spezifika einer Website und daher ein *generisches* bzw. wiederverwendbares Modul.

### Merkmale

- Modul: Scheduler
- Systemkomponente: Scheduler
- Typ: generisch
- Aufgabe: Zeitsteuerung für den Extraktor und den Übertrager
- Strategie: Abfragestrategie

## Extraktor

Der *Extraktor* bildet aus der Konfiguration und dem Zustand des Facettenspeichers eine Menge von Ziel-URLs. Diese Ziel-URLs übergibt er dem Sammler, um die HTML-Quellen abzufragen. Das Ergebnis dieser Abfragen wird dem Mustersucher übergeben, der entweder Facetten von Entitäten findet oder, der Navigationsstrategie der explorativen Traversierung folgend, weitere URLs. Gefundene Informationen werden im Facettenspeicher abgelegt. Nicht mehr existierende Informationen, die während einer früheren Abfrage extrahiert und im Facettenspeicher abgelegt wurden, werden aus dem Facettenspeicher entfernt oder als gelöscht markiert. Er implementiert die Navigationsstrategie und muss den meisten Herausforderungen begegnen.

Der Extraktor ist stark von der Navigationsstrategie und dem Aufbau der Website abhängig, daher muss er für jedes Screen-Scraping-System *individuell* entworfen und implementiert werden.

### Merkmale

- Modul: Scraper
- Systemkomponente: Scraper
- Typ: individuell
- Aufgabe: Abarbeitung der Ziel-URLs unter Nutzung der Schnittstellen von Sammler und Mustersucher
- Strategie: Navigationsstrategie

## Sammler

Der *Sammler* nimmt URLs entgegen und liefert die HTML-Quelle oder eine Fehlermeldung zurück. Er greift zunächst auf den Zwischenspeicher zu, um die HTML-Quelle für eine URL abzufragen. Erst wenn dieser keinen Treffer liefert, übergibt er die URL dem Web-Client. Erhält er vom Web-Client die HTML-Quelle, speichert er sie für spätere Anfragen im Zwischenspeicher. Er kann auch das Abfragen von referenzierten Ressourcen übernehmen, um Browser-typische Abfragemuster nachzuahmen. Er implementiert auch die Aspekte Verzögerung von HTTP-Anfragen und Parallelisierung der Abfragestrategie.

Der Sammler wird lediglich durch die Abfragestrategie beeinflusst und kann als *generisches* Modul implementiert werden. Dadurch wird der Sammler für mehrere Screen-Scraping-Systemen wiederverwendbar. Die notwendigen Freiheitsgrade zur Anpassung an die jeweilige Abfragestrategie lassen sich leicht durch entsprechend flexible Parametrisierung unterstützen.

### Merkmale

- Modul: Crawler
- Systemkomponente: Scraper
- Typ: generisch
- Aufgabe: Organisation von HTTP-Anfragen
- Strategie: Abfragestrategie

## Web-Client

Der *Web-Client* nimmt URLs entgegen, führt die HTTP-Kommunikation mit dem Web-Server durch, und gibt die HTML-Quelle zurück. Wenn es zu Störungen beim

Abruf der Web-Seite kommt, gibt er stellvertretend eine Fehlermeldung zurück. Er implementiert den Aspekt HTTP-Header der Abfragestrategie.

Der Web-Client ist unabhängig von den Spezifika der Website und kann als *generisches* Modul implementiert werden. Die zu sendenden HTTP-Header zur Anpassung an die jeweilige Abfragestrategie lassen sich leicht durch entsprechende Parametrisierung unterstützen.

### Merkmale

- Modul: HTTP Client
- Systemkomponente: Scraper
- Typ: generisch
- Aufgabe: HTTP-Kommunikation mit dem Web-Server
- Strategie: Abfragestrategie

## Zwischenspeicher

Der *Zwischenspeicher* nimmt eine URL entgegen und liefert die dazugehörige HTML-Quelle oder einen Fehlschlag zurück. Er kann die Effizienz und die Abfragegeschwindigkeit steigern, wenn URLs mehrfach abgefragt werden. Wichtiger ist seine Aufgabe während der Entwicklungszeit und wenn Anpassungen am Mustersucher vorgenommen werden müssen.

Mit einem Zwischenspeicher ist es prinzipiell möglich alle erforderlichen URLs nur einmal abzufragen und anschließend verschiedenen Varianten bzw. Versionen von Extraktoren und Mustersuchern durchlaufen zu lassen ohne erneut Abfragen an den Web-Server schicken zu müssen. Das erhöht die Entwicklungsgeschwindigkeit maßgeblich und schont die Kapazitäten des Web-Server. Wird im produktiven Betrieb ein Fehler bei der Mustersuche entdeckt, kann dieser behoben, die Extraktion erneut durchgeführt und damit der Rohdatenbestand bereinigt werden.

Der Zwischenspeicher sollte neben der URL und der HTML-Quelle auch einen Zeitstempel speichern, der angibt zu welchem Zeitpunkt die Web-Seite abgerufen wurde. Es sollten auch HTTP-Status-Code und Zugriff- bzw. Übertragungszeit gespeichert werden. Damit enthält der Zwischenspeicher ein vollständiges Protokoll der HTTP-Kommunikation mit dem Web-Server, was im Fehlerfall die Analyse und Fehlerbehebung erleichtert. Zusätzlich sind auch aussagekräftige Statistiken bei einer Überwachung des Screen Scraping möglich.

Der Zwischenspeicher ist von den Spezifika der Website und den gesuchten Informationen unabhängig und kann als *generisches* Modul implementiert werden. Für die eigentliche Speicherung (*Rohdatenspeicher*) kann entweder direkt das Dateisystem oder ein dafür geeignetes Datenbanksystem genutzt werden.

### Schema

- URL
- Zeitstempel
- HTTP-Status-Code
- HTML-Quelle

### Merkmale

- Modul: Cache
- Systemkomponente: Scraper
- Typ: generisch
- Aufgabe: HTML-Quellen für mehrfache Abfragen zwischenspeichern

## Mustersucher

Der *Mustersucher* analysiert den DOM einer HTML-Quelle mit dem Ziel die gesuchten Informationen aus einer Web-Seite zu extrahieren. Je nach Navigationsstrategie und aktuell zu analysierender Web-Seite sucht er nach Facetten von Entitäten, oder nach Navigationsschaltflächen, um weitere URLs für die Abfrage zu ermitteln. Er implementiert damit die Analysestrategie.

Der Mustersucher ist stark vom Aufbau der Web-Seiten und den gesuchten Informationen abhängig und muss daher *individuell* für jedes Screen-Scraping-System implementiert werden. Teilaspekte der Mustersuche können jedoch gut in wiederverwendbare Programmierbibliotheken ausgelagert werden.

### Merkmale

- Modul: Analyzer
- Systemkomponente: Scraper
- Typ: individuell
- Aufgabe: Facetten von Entitäten und URLs in HTML-Quellen finden
- Strategie: Analysestrategie

## Facettenspeicher

Der *Facettenspeicher* nimmt vom Extraktor die extrahierten Informationen entgegen und speichert sie, damit sie vom Übertrager später abgerufen und zusammengeführt werden können. Der Facettenspeicher ermöglicht es, das Extrahieren der Informationen aus den Web-Seiten vom Bilden konsistenter Entitäten zu entkoppeln.

Facetten werden durch Identifikatoren identifiziert, die aus dem Kontext der Website stammen. Sobald aus den Facetten konsistente Entitäten gebildet und im Entitätenspeicher abgelegt wurden, wird zu jeder Facette der Identifikator der abgeleiteten Entität gespeichert.

Es ist ratsam, zu jeder Facette die URL jener Web-Seite zu speichern, aus der ihre Informationen extrahiert wurden. Verschwinden die Informationen einer Entität von der Website, müssen die bereits gespeicherten Facetten der Entität als gelöscht markiert werden. Dazu kann ein Indikator-Feld an den Facetten dienen. Des Weiteren sind zwei Zeitstempel je Facette hilfreich. Einer der den Zeitpunkt der ersten Abfrage festhält, bei der die Informationen der Facette extrahiert wurden. Und ein weiterer für den Zeitpunkt der letzten Abfrage, aufgrund welcher die Facette zuletzt aktualisiert oder bestätigt wurde.

Der Facettenspeicher ist zwar von den Spezifika der Website unabhängig, muss jedoch ein individuelles Schema für die Facetten-Typen unterstützen. Solange das Schema der Facetten flexibel konfiguriert werden kann, ist der Facettenspeicher in mehreren Screen-Scraping-System wiederverwendbar. Für die eigentliche Speicherung ist es sinnvoll, ein Datenbanksystem zu verwenden (*Facetten-DB*).

### Schema

- Facetten-Typ
- Entitäten-Typ
- Identifikator (Website)
- Identifikator (Entitätenspeicher)
- URL
- erster Abfragezeitpunkt
- letzter Abfragezeitpunkt
- Indikator für Löschung

- *Facettendaten*

### Merkmale

- Modul: Raw Storage
- Systemkomponente: Scraper, Transfer
- Typ: individuelles Schema
- Aufgabe: Facetten von Entitäten vom Extraktor entgegennehmen, speichern und wieder an den Übertrager ausliefern

## Übertrager

Der *Übertrager* durchsucht den Facettenspeicher nach Facetten von Entitäten und überträgt vollständige und konsistente Entitäten in den Entitätenspeicher. Findet er Entitäten im Entitätenspeicher, deren Facetten im Facettenspeicher fehlen oder als gelöscht markiert sind, entfernt er diese auch aus dem Entitätenspeicher. Als Ergebnis enthält der Entitätenspeicher ein konsistentes Abbild der aus der Website extrahierten Entitäten.

Beim Zusammenführen der Facetten zu vollständigen Entitäten vergibt er bei Bedarf neue Identifikatoren. Diese neuen Identifikatoren vermerkt der Übertrager auch an den Facetten im Facettenspeicher um sich den erneuten Abgleich zwischen den Facetten und den Entitäten zu erleichtern.

Prinzipiell sollte es jederzeit möglich sein, den Inhalt des Entitätenspeichers zu verwerfen und diesen mittels des Übertragers aus dem Facettenspeicher wiederherzustellen. Dies kann sehr hilfreich sein, wenn bei der Zusammenführung von Entitäten Fehler auftreten. Dann kann der Fehler im Übertrager behoben und der Entitätenspeicher neu aufgebaut werden, ohne dass Web-Seiten vom Web-Server erneut abgerufen werden müssen.

Der Übertrager ist abhängig von der Aufteilung der Entitäten in einzelne Facetten. Auch die Art der Identifikatoren von Facetten und Entitäten und die Konsistenzeigenschaften der Entitäten beeinflussen die Logik des Übertragers. Daher muss der Übertrager für jedes Screen-Scraping-System *individuell* implementiert werden.

### Merkmale

- Modul: Transfer
- Systemkomponente: Transfer

- Typ: individuell
- Aufgabe: Facetten von Entitäten aus dem Facettenspeicher zusammenführen und konsistente Daten im Entitätenspeicher ablegen

## Entitätenspeicher

Der *Entitätenspeicher* nimmt vom Übertrager konsistente Entitäten entgegen, speichert sie und stellt sie Konsumenten für weitere Verarbeitung zur Verfügung. Bei der Ablage im Entitätenspeicher erhalten neue Entitäten einen künstlichen Identifikator. Werden Facetten einer Entität als gelöscht markiert, wird die Entität aus dem Entitätenspeicher entfernt.

Wird eine Entität aktualisiert, weil sich deren zugrundeliegende Facetten geändert haben, sollte der Zeitpunkt der Aktualisierung festgehalten werden. Dies gilt auch für die erste Zusammenführung einer Entität. Zu jeder Entität sollte ihr Ursprung gespeichert werden. Das ist mindestens dann sinnvoll, wenn Entitäten gleichen Typs aus verschiedenen Websites extrahiert werden.

Der Entitätenspeicher muss ein individuelles Schema für die Entitäten-Typen unterstützen. Solange das Schema der Entitäten flexibel konfiguriert werden kann, ist der Entitätenspeicher in mehreren Screen-Scraping-Systemen wiederverwendbar. Werden mehrere Screen-Scraping-Systeme eingesetzt, um Entitäten aus mehreren Websites zu sammeln, kann ein einzelner Entitätenspeicher als gemeinsame Schnittstelle zwischen den Screen-Scraping-Systemen und den Konsumenten verwendet werden. Für die eigentliche Speicherung ist es sinnvoll, ein Datenbanksystem zu verwenden (*Entitäten-DB*).

Sind die Konsumenten während des Entwurfs eines Screen-Scraping-Systems bekannt, kann bei der Gestaltung der öffentlichen Schnittstelle des Entitätenspeichers auf deren technische Bedürfnisse Rücksicht genommen werden. Andernfalls ist eine standardisierte Schnittstelle wie SQL oder HTTP-REST eine gute Option.

### Schema

- Entitäten-Typ
- Identifikator
- Ursprung
- Zeitpunkt der letzten Aktualisierung
- *Entitätendaten*

### Merkmale

- Modul: Entity Storage
- Systemkomponente: Transfer
- Typ: individuelles Schema
- Aufgabe: Entitäten vom Übertrager entgegennehmen, speichern und Konsumenten zur Verfügung stellen

## Fremdsysteme

Ein Screen-Scraping-System kommuniziert mit mindestens zwei Fremdsystemen – einem Web-Server und einem Konsumenten.

### Web-Server

Der *Web-Server* ist der Ursprung der extrahierten Informationen. Er liefert die Website über das HTTP-Protokoll aus, in deren Web-Seiten die gesuchten Informationen enthalten sind. Der Web-Server besitzt eine bestimmte Kapazität – Rechenkapazität, Übertragungsrate, Zugriffszeiten, maximale Anzahl paralleler Anfragen – und steht unter der Verantwortung eines Betreibers.

Bei dem Web-Server muss es sich nicht zwingend um einen Hardware-Server oder eine Programminstanz handeln, vielmehr ist er der logische Endpunkt für eine Domain oder einen Dienst. Der Betreiber eines Web-Servers hat viele Möglichkeiten die Kapazitäten eines Web-Servers mit Hilfe von Load-Balancern weit über die Kapazitäten eines einzelnen Hardware-Servers hinaus zu steigern. Umgekehrt kann ein Hardware-Server aber auch eine Vielzahl von Web-Servern ausführen, wobei sich alle Web-Server die Kapazitäten eines Hardware-Servers teilen müssen.

Bei der Entwicklung eines Screen-Scraping-Systems besteht i. d. R. nicht die Möglichkeit auf technische Eigenschaften des Web-Servers Einfluss zu nehmen. Daher muss das Screen-Scraping-System sich in seiner Implementierung und Konfiguration immer am Web-Server und den von ihm ausgelieferten Web-Seiten ausrichten.

### Konsumenten

Die *Konsumenten* sind Systeme, welche über den Entitätenspeicher an das Screen-Scraping-System angebunden sind. Sie sind an konsistenten Entitäten in einem

möglichst aktuellen Zustand interessiert und können diese direkt aus dem Entitätenspeicher abfragen. Konsumenten können die Entitäten beliebig weiterverarbeiten. Sie können entweder im Pull-Prinzip, bei Bedarf oder regelmäßig, auf den Entitätenspeicher zugreifen. Oder sie lassen sich über eine zusätzliche Schnittstelle des Entitätenspeichers über neue Entitäten, Aktualisierungen und Löschungen im Push-Prinzip informieren.

## Allgemeine Aspekte

Bei der Implementierung der Module sollte auf einige allgemeine Aspekte Rücksicht genommen werden.

### Protokollierung

Generell sollte ein Screen-Scraping-System eine ausführliche Protokollierung ermöglichen. Besonders wichtig ist dies für Systeme, die unter Umständen lange benötigen, um alle gesuchten Informationen aus der Website zu extrahieren.

Die Protokollierung dient primär zwei Zielen:

1. Die Ursache für ein Fehlverhalten des Systems ermitteln
2. Die Aktivität des Systems überwachen

Es ist sinnvoll, alle Teilschritte in unterschiedlicher Granularität zu protokollieren. Üblich sind Detailgrade wie z. B. `Verbose`, `Info`, `Warning` und `Error`. Alle Code-Abschnitte die mit externen Systemen kommunizieren oder unmittelbar mit den abgefragten HTML-Quellen zu tun haben, müssen eine sorgfältige Fehlerbehandlung enthalten. Auftretende Fehler sollten zunächst sofort protokolliert werden (`Warning` oder `Error`), bevor entsprechend der Konfiguration und der logischen Möglichkeiten fortgefahren wird.

Jeder Protokolleintrag sollte mit einem präzisen Zeitstempel, dem Detailgrad bzw. Typ der Meldung und dem Namen des Ursprungsmodul versehen sein. Dadurch lässt sich ein Protokoll im Nachhinein leichter filtern und die Vorgänge, die zu einem Fehler geführt haben, sauber rekonstruieren.

## Konfiguration

Es ist sinnvoll, alle Stellgrößen die für die Algorithmen in den Modulen benötigt werden, als Konfigurationsparameter zu implementieren. So dass sie in einer ausgelagerten Konfigurationsdatei leicht verändert werden können und für deren Anpassung keine Änderungen im Quelltext des Screen-Scraping-Systems notwendig sind. Besonders komfortabel ist eine Konfiguration, wenn sie aus mehreren priorisierten Quellen zusammengesetzt wird – z. B. Konfigurationsdatei, Umgebungsvariablen und Befehlszeilen-Parameter.

Eine detailierte Konfiguration ist sowohl während der Entwicklungszeit des Screen-Scraping-Systems hilfreich, als auch während des produktiven Einsatzes. Während der Entwicklungszeit kann das Verhalten des Systems so schnell für bestimmte Testsituationen oder zur Fehlersuche angepasst werden. Es können auch verschiedene Konfigurationsdateien für verschiedene Testszenarien verwendet werden. Und auch während des produktiven Einsatzes ist es immer wieder erforderlich, einzelne Stellgrößen anzupassen.

Im Folgenden werden einige Beispiele für mögliche Konfigurationsparameter genannt.

`Common.LogFile`
: *Dateiname für die Protokolldatei*

`Common.LogLevel`
: *Detailgrad von Protokolleinträgen*

`Scheduler.ScrapeInterval`
: *Intervall für das Starten des Screen Scraping*

`Scraper.CancelOnError`
: *Schalter für das Verhalten, wenn bei der Extraktion Fehler auftreten*

`Scraper.BaseUrl`
: *Domainname und Basis-Pfad für alle zu bildenden URLs*

`Scraper.<FacetType>.Url`
: *URL-Teil für das Erfassen eines bestimmten Facetten-Typs*

`Analyzer.<FacetType>.<Field>`
: *CSS-Selektor für das Auffinden eines Datums in der Web-Seite*

`Crawler.Offline`
: *Schalter für den Offline-Betrieb – Anfragen werden nur aus dem Zwischenspeicher beantwortet*

`HttpClient.UserAgentString`
: *Zeichenfolge für das HTTP-Header-Feld UserAgent*

`HttpClient.Throttle`
: *Schalter für das Aktivieren von Pausen zwischen Anfragen*

`HttpClient.MinThrottle`
: *Minimale Pausenzeit*

`HttpClient.MaxThrottle`
: *Maximale Pausenzeit*

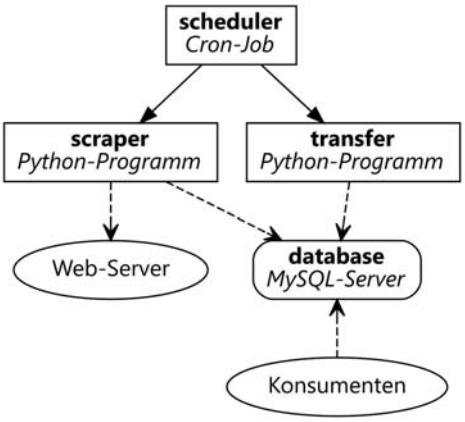
`HttpClient.Proxy`
: *Ein möglicherweise erforderlicher Proxy-Server*

`Cache.Directory`
: *Verzeichnis für das Speichern der Rohdaten*

`Cache.RetentionTime`
: *Zeitspanne nach der gespeicherte Web-Seiten verworfen werden*

# Implementierung

Im Folgenden wird kurz eine mögliche Implementierung eines Screen-Scraping-Systems beschrieben. Das System wird auf einem Linux-Betriebssystem ausgeführt und verwendet den Cron-Scheduler des Betriebssystems als *Scheduler*-Modul. Die übrigen Module werden in Python 3 implementiert und sind in zwei Prozesse aufgeteilt.




**Übersicht über eine mögliche Implementierung**

Der erste Prozess ist für das Erfassen und Extrahieren der Informationen zuständig und wird hier mit *scraper* bezeichnet. Er umfasst die Module *Scraper*, *Crawler*, *HTTP Client*, *Cache*, *Analyzer* und *Facet Storage*. Der zweite Prozess ist für das Zusammenführen der Facetten in konsistente Entitäten zuständig und wird hier mit *transfer* bezeichnet. Er umfasst die Module *Transfer*, *Facet Storage* und *Entity Storage*.

Da das Module *Facet Storage* in beiden Prozessen benötigt wird, wird es in einer Bibliothek implementiert, welche von beiden Prozessen verwendet werden kann. Allgemeine Aspekte wie Protokollierung und Konfiguration werden ebenfalls in einer Bibliothek implementiert und von beiden Prozessen referenziert.

Das Modul *HTTP Client* verwendet die Python-Bibliothek `requests` für die HTTP-Kommunikation mit dem Web-Server (vergl. [16]). Das Modul *Cache* speichert die Rohdaten als Dateien in einem Verzeichnis. Dabei wird der Dateiname für eine HTML-Quelle aus dem Zeitstempel des Abfragezeitpunktes, einem SHA-1-Hash der URL und dem HTTP-Status-Code gebildet (vergl. [17]). Das Modul *Analyzer* verwendet die Python-Bibliothek `bs4` (Beautiful Soup 4) für die DOM-Repräsentation und das Selektieren von HTML-Elementen mit der Bibliothek `lxml` für das Parsen von HTML (vergl. [18], [19]). Die Module *Facet Storage* und *Entity Storage* kommunizieren mit einem MySQL-Server und setzen dabei die Python-Bibliothek `sqlalchemy` für das OR-Mapping mit `pymysql` als SQL-Adapter ein (vergl. [20], [21]). Für die Facetten und die Entitäten werden zwei getrennte Datenbanken auf dem MySQL-Server verwendet. Konfigurationsdateien werden im INI-Format gespeichert und mit der Klasse `SafeConfigParser` aus der Python-Bibliothek `configparser` eingelesen

(vergl. [22]). Für das Verarbeiten von Befehlszeilen-Parametern wird die Python-Bibliothek `argparse` verwendet (vergl. [23]). Für die Protokollierung wird die Python-Bibliothek `logging` verwendet (vergl. [24]).

Konsumenten greifen direkt auf die Entitäten-Datenbank auf dem MySQL-Server zu.

# Empfehlungen

Eine gute Grundlage zur Implementierung eines Screen-Scraping-Programms ist das Python-Framework *Scrapy* (vergl. [25]). Scrapy ist ein Framework, welches weite Teile des Screen Scraping unterstützt. Es löst die Herausforderungen, die bei den HTTP-Anfragen entstehen, sehr gut und stellt eine solide Basis für die Analyse mittels Selektoren zur Verfügung. Darüber hinaus bietet es definierte Schnittstellen für Erweiterungen und Anpassungen.

Scrapy bietet u. a. die folgenden Funktionalitäten:

- Robuster und konfigurierbarer Crawler
- Parallele Ausführung mehrerer HTTP-Abfragen
- Statistiken
- Middleware für
  - Authentifizierung
  - Cookies
  - Cache
  - Abruf über Proxy
  - Verfolgen von Umleitungen
  - User-Agent überschreiben
  - Webseiten nach Fehler erneut Abfragen
  - URL-Filter
  - HTTP-Status-Code-Filter
- Analyse mittels
  - CSS-Selektoren
  - XPath-Selektoren
- Einstellungen über
  - Befehlszeilenargumente
  - Python-Code
- Ausgabe extrahierter Facetten

  - XML
  - CSV
  - Pickle (kompakte Python-spezifische Serialisierung)
  - pprint
  - JSON/JSON-Lines
- E-Mail-Versand für Benachrichtigungen
- Interaktive Python-Shell über eine Telnet-Konsole

Die folgenden Aspekte werden von Scrapy nicht direkt unterstützt:

- Speichern der extrahierten Facetten in einer relationalen Datenbank (nur über Erweiterung möglich)
- Zusammenführen von Facetten in Entitäten (nachgelagerter Transfer-Schritt notwendig)
- Verteilung der Ausführung auf mehrere Rechner (Ausführung von Crawlern auf einem Server mit *Scrapyd* möglich, vergl. [26])

Erfüllt Scrapy eine besondere Anforderung nicht, sollte zunächst geprüft werden ob diese mittels einer Erweiterung erfüllt werden kann, oder Scrapy als Teilkomponente einer größeren Architektur genutzt werden kann. Ist auch dies nicht der Fall, kann die Implementierung eines spezialisierten Programms in Erwägung gezogen werden. Der Aufwand dafür sollte jedoch nicht unterschätzt werden.

Wird die Implementierung eines spezialisierten Programms in Betracht gezogen, oder wird Scrapy als Teilkomponente in eine größere Architektur eingebettet, könnte zur Orchestrierung der einzelnen Teilaufgaben (Webseitenabruf, Datenextraktion, Bereinigung) das Python-Framework *Luigi* genutzt werden (vergl. [27]). Das Framework unterstützt eine klare Gliederung der Teilaufgaben mit ihren Abhängigkeiten und ermöglicht die Koordination von verteilt ausgeführten Aufgaben. Dafür bietet es einen zentralen Dienst, welcher die Ausführung der Teilaufgaben überwacht und in einer Web-Oberfläche visualisiert.

Bei großen Datenmengen eignet sich der *PostgreSQL*-Server u.U. besser für die Speicherung von Facetten und Entitäten als der *MySQL*-Server bzw. dessen freier Fork *MariaDB*. Der *PostgreSQL*-Server bietet mächtigere Datentypen und seine SQL-Syntax ist mit der von *Amazon Redshift* kompatibel und eröffnet damit, falls nötig, eine leichtere Migration in die Cloud (vergl. [28], [29]). Ausschlaggebend für die Wahl des Datenbanksystems ist natürlich auch die Kompatibilität mit den Konsumenten der Entitäten.

# Quellen

[1] Hemmerich, Wanja A. : *Normalverteilung*. 2017. — URL https://mathe-guru.com/stochastik/normalverteilung.html

[2] Apsel, Matthias ; Schwarwies, Matthias : *selfhtml – HTML / Universalattribute / id*. 2017. — URL https://wiki.selfhtml.org/wiki/HTML/ … Universalattribute#id

[3] Apsel, Matthias : *selfhtml – CSS / Selektoren / Klassenselektor*. 2017. — URL https://wiki.selfhtml.org/wiki/CSS/Selektoren/Klassenselektor

[4] *XPath*. Wikipedia, 2017. — URL https://de.wikipedia.org/wiki/XPath

[5] Apsel, Matthias ; Schwarwies, Matthias : *selfhtml – CSS / Selektoren*. 2017. — URL https://wiki.selfhtml.org/wiki/CSS/Selektoren

[6] Diggs, Joanmarie ; McCarron, Shane ; Cooper, Michael ; Schwerdtfeger, Richard ; Craig, James : *Accessible Rich Internet Applications (WAI-ARIA) 1.1: W3C Recommendation 14 December 2017*. W3C, 2017. — URL https://www.w3 … .org/TR/wai-aria-1.1/

[7] Chang, Chia-Hui ; Lui, Shao-Chen : IEPAD: Information Extraction Based on Pattern Discovery. In: *Proceedings of the 10th International Conference on World Wide Web, WWW '01*. New York, NY, USA : ACM, 2001 — ISBN 1-58113-348-0, S. 681–688

[8] Chang, Chia-Hui ; Hsu, Chun-Nan ; Lui, Shao-Cheng : Automatic information extraction from semi-structured Web pages by pattern discovery. In: *Decision Support Systems* Bd. 35 (2003), Nr. 1, S. 129–147. — Web Retrieval and Mining

[9] Amitay, Einat ; Carmel, David ; Darlow, Adam ; Lempel, Ronny ; Soffer, Aya : The Connectivity Sonar: Detecting Site Functionality by Structural Patterns. In: *Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia, HYPERTEXT '03*. New York, NY, USA : ACM, 2003 — ISBN 1-58113-704-4, S. 38–47

[10] Chakrabarti, Deepayan ; Mehta, Rupesh : The Paths More Taken: Matching DOM Trees to Search Logs for Accurate Webpage Clustering. In: *Proceedings*

*of the 19th International Conference on World Wide Web, WWW '10*. New York, NY, USA : ACM, 2010 — ISBN 978-1-60558-799-8, S. 211–220

[11] *Single-Page-Webanwendung*. Wikipedia, 2018. — URL https://de.wiki … pedia.org/wiki/Single-Page-Webanwendung

[12] Podmanicki, Toni : *Wie Sie JavaScript in Ihrem Browser einschalten und wofür es gebraucht wird*. 2018. — URL https://www.enable-javascript.com/de/

[13] Maone, Giorgio : *NoScript – JavaScript/Java/Flash blocker for a safer Firefox experience*. 2018. — URL https://noscript.net

[14] Hidayat, Ariya : *PhantomJS – Full web stack, No browser required*. 2018. — URL http://phantomjs.org

[15] *Universally Unique Identifier*. Wikipedia, 2018. — URL https://de.wiki … pedia.org/wiki/Universally_Unique_Identifier

[16] Reitz, Kenneth : *Requests: HTTP for Humans*. 2017. — URL http://python- … requests.org

[17] *Secure Hash Algorithm*. Wikipedia, 2017. — URL https://de.wikipedia … .org/wiki/Secure_Hash_Algorithm

[18] Richardson, Leonard : *Beautiful Soup*. 2017. — URL https://www.crummy … .com/software/BeautifulSoup/

[19] Behnel, Stefan ; u.a. : *lxml – XML and HTML with Python*. 2017. — URL http://lxml.de/

[20] Bayer, Michael ; Kirtland, Jason ; de, Gaetan ; Clarke, Diana ; Trier, Michael ; Jenvey, Philip ; Aasma, Ants ; Johnston, Paul ; u. a. : *SQL Alchemy*. 2018. — URL http://www.sqlalchemy.org

[21] Naoki, Inada ; Black, Daniel ; Rodrigues, Marcel ; Hunt, Pete ; u.a. : *PyMySQL – Pure Python MySQL Client*. 2017. — URL https://github.com/PyMySQL … /PyMySQL

[22] Foundation, Python Software : *configparser – Configuration file parser*. 2017. — URL https://docs.python.org/3.5/library/configparser.html

[23] Foundation, Python Software : *argparse – Parser for command-line options, arguments and sub-commands*. 2017. — URL https://docs.python.org/ … 3.6/library/argparse.html

[24] Foundation, Python Software : *logging – Logging facility for Python*. 2018. — URL https://docs.python.org/3/library/logging.html

[25] Evans, Shane ; u.a. : *Scrapy – Extracting the data you need from websites*. 2018. — URL https://scrapy.org/

[26] Hoffman, Pablo ; u.a. : *Scrapyd – A service daemon to run Scrapy spiders*. 2018. — URL https://scrapyd.readthedocs.io/en/stable/

[27] Bernhardsson, Erik ; Freider, Elias ; u.a. : *Luigi – Python package for building complex pipelines and batch jobs*. 2018. — URL https://luigi.readthedocs …
.io/en/stable/

[28] PostgreSQL Global Development Group : *PostgreSQL*. 2018. — URL https://www.postgresql.org/

[29] Amazon Web Services : *Redshift*. 2018. — URL https://aws.amazon.com …
/de/redshift/