# The ContentMine scraping stack: literature-scale content mining with community maintained collections of declarative scrapers

Richard Smith-Unna
University of Cambridge

Peter Murray-Rust
University of Cambridge

## 1 Introduction

The need to mine content from the scholarly literature at scale is inhibited by technical and political barriers. Publishers have partially addressed this by providing application programming interfaces (APIs). However, we observe that many publisher APIs come with restrictions that inhibit data mining at scale, and that while only a few publishers provide APIs, almost all publishers make their content available on the web. With current web technologies it should be possible to harvest and mine the entire scholarly literature as it is published, regardless of the source of publication, and without using specialised programmatic interfaces controlled by each publisher. To address this challenge as part of the ContentMine project (http://contentmine.org) we developed the following tools:

- **scraperJSON**, a standard for defining reusable web scrapers as JSON objects
- **thresher**, a web scraping library that uses scraperJSON scrapers and headless browsing to handle many idiosyncrasies of the modern web
- **quickscrape**, a command-line web scraping suite that uses thresher
- a library of **scraperJSON scrapers** to extract data and metadata from many academic publishers

We will demonstrate the use of this stack for scraping the scholarly literature.

## 2 Software description

### 2.1 scraperJSON

We defined a new JSON schema, scraperJSON, to enable declarative scraping of structured data from many different sites without duplicating code. A scraperJSON scraper minimally defines the url(s) to which it applies and the set of elements that it can extract. Elements can be extracted by CSS, XPath, or using the Open Annotation standard, and can be post-processed using regular expression capture groups.

#### *2.1.1 Implementation*

We chose to use JSON as the data format because it is widely supported and is the natural data format of JavaScript, in which our web apps and scraping software are written (via Node.js).
The declarative approach enables building tools to enable non-programmers to create and maintain scrapers. This is prohibitively difficult with existing scraper collections, such as that used by the open source reference-manager Zotero.

#### *2.1.2 Related work*

To our knowledge, only one open-source declarative scraping system, libKrake (http://github.com/KrakeIO/libkrake), exists. However, this software seems to no longer be maintained and did not support many of the features we require, such as simulated interactions and file downloads.

#### *2.1.3 Simulating user interaction*

Many modern websites do not specify their content in the raw HTML served when a URL is visited, but lazy-load the content, for example by making AJAX calls. These loading events are triggered by user interactions such as scrolling and clicking. To enable to scraping of full content, we therefore added to scraperJSON the ability to specify a list of predefined user interactions.

#### *2.1.4 Structuring results*

The structure of the scraperJSON *elements* object defines the structure of the results. Thus, each key in the *elements* of the scraper will be reflected as a key in the *elements* of the results. Elements can contain other elements, so that the results of the child elements are grouped into objects reflecting the structure of the parent element. This allows powerful capture of structured data representing entities and their properties, such as the name, affiliation and email of an author.

#### *2.1.5 Downloads*

When scraping the scholarly literature, it is usually of interest to download files such as fulltext PDFs and HTML, XML and RDF documents where available, and to capture full-sized figure images and supplementary materials. This feature is supported in scraperJSON: elements specifying URLs can have their target(s) downloaded and optionally renamed to a specified format.

#### *2.1.6 Nested scrapers*

Content associated with a page is often available in a more extensive form in a linked resource. This is observed with full figures on the websites of many acadmic publishers, where a thumbnail is displayed in the article and a link must be followed to expose the full image. This situation can be handled in scraperJSON: if an element targets a URL, the URL can be followed and one or more child elements extracted.

### 2.2 Scraping with *thresher* & *quickscrape*

#### *2.2.1 thresher*

**Implementation**
***thresher*** is a web scraping library that uses scraperJSON scrapers. It is written in Node.js, is cross-platform, is fully

covered by tests, and is released under the MIT license.
JavaScript was selected as the development langauge because it is the de-facto language of the web. Node.js in particular allows seamless creation of APIs, command-line tools and GUI tools or webapps using the same codebase, and because the ecosystem of packages simplifies rapid cross-platform development.
Pages are rendered either using the Node.js package **jsdom**, or in a headless WebKit browser via PhantomJS. PhantomJS was selected over alternatives, such as Selenium WebDriver, due to the requirement that our scraping stack can run on servers without any X windowing system.
The output of each scraping operation is a JSON document that mirrors the structure of the scraper's scraperJSON definition. The format is BibJSON-compatible, allowing conversion to other scholarly metadata standards.

**Automatic scraper selection**
By harnessing the url specification in scraperJSON, thresher can process lists of URLs and automatically choose the best scraper for each from a provided collection of scrapers.

**Headless rendering**
Headless browsers are standalone web rendering engines with the GUI removed, commonly used for testing web-facing software. Thresher supports the optional user interaction feature in scraperJSON by rendering webpages and performing actions in PhantomJS if interactions are specified.

**Rate-limiting**
To encourage responsible scraping, and to avoid triggering denial of service by publishers, thresher implements a domain-based, customisable rate limit. A separate queue of URLs to be scraped is maintained for each domain, and the queues are processed in parallel.

**Authentication**
Because much of the scholarly literature is behind a paywall, thresher allows authentication by one of three commonly used methods:

1. using HTTP login/password authentication
2. using a proxy
3. by providing a set of session cookies

#### 2.2.2 *quickscrape*

***quickscrape*** is a cross-platform command-line suite that provides access to all the functionality of thresher. An overview of the interaction of the two systems is shown in figure 1.

### 2.3 Journal scrapers

We developed a collection of scraperJSON scrapers for major publishers. The scrapers collect article metadata, downloadable resources associated with the article, as well as citations and the full text of the article if available. Maintenance and expansion of the collection is done with the help of a community of volunteers. Each scraper is associated with a list of URLs for which the expected scraping results are known, and an automated testing system (using Github and Travis CI integration) checks that all the scrapers are functioning with perfect precision and recall (i.e. extracting the expected results) every day and after every change to a scraper. This allows a rapid community response to formatting changes by publishers. Accessory scripts are provided for automatically generating tests for scrapers, and for run-

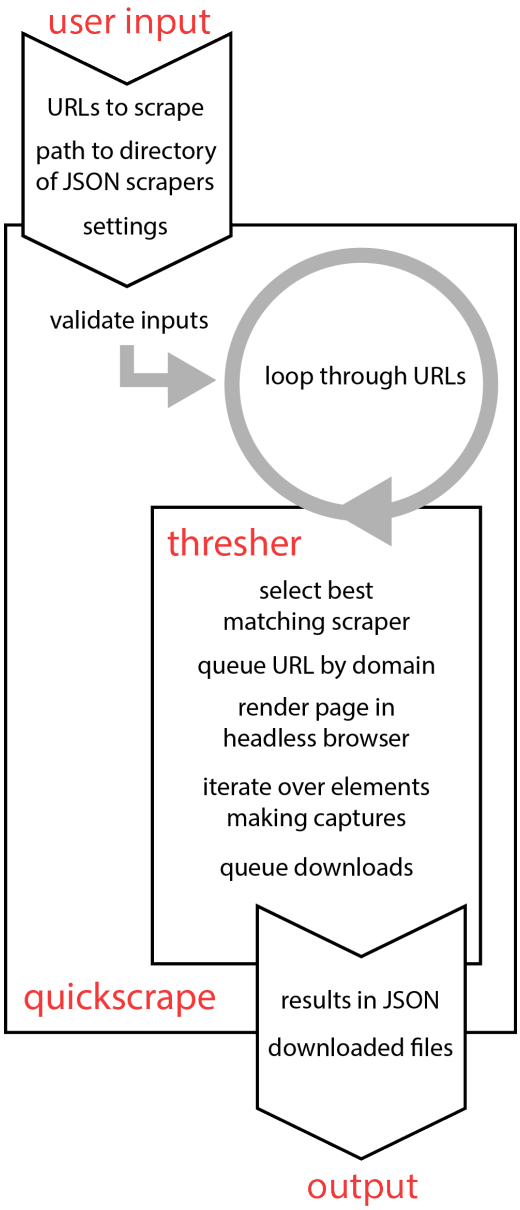


**Figure 1. Schematic overview of the scraping stack. quickscrape provides a command-line interface to the thresher library, allowing users to specify inputs and settings and retrieve results. thresher drives the scraping process by matching URLs to those in the collection of scraperJSON scrapers, rendering the target, and capturing the specified elements.**

ning the tests. The whole collection is released under the CC0 license.

## 2.4 Future work

In future work we will enhance the ContentMine scraping stack by creating a layer of web tools enabling non-programmers to develop and maintain scrapers, and to conduct text and data mining directly from the browser using our resources.

## 2.5 Availability

All software is open source and available from the ContentMine Github organisation: https://github.com/ContentMine.

# 3 Acknowledgements

PM-R thanks the Shuttleworth Foundation for a fellowship and grant.