Slug: A Semantic Web Crawler

Leigh Dodds
Engineering Manager, Ingenta

Jena User Conference
May 2006

Overview

- Do we need Semantic Web Crawlers?
- Current Features

- Crawler Architecture
- Crawler Configuration
- Applications and Future Extensions

Do We Need Semantic Web Crawlers?

- Increasing availability of distributed data
  - Mirroring often only option for large sources
- Varying application needs
  - Real-time retrieval not always necessary/desirable
- Personal metadata increasingly distributed
  - Need a means to collate data
- Compiling large, varied datasets for research
  - Triple store and query engine load testing

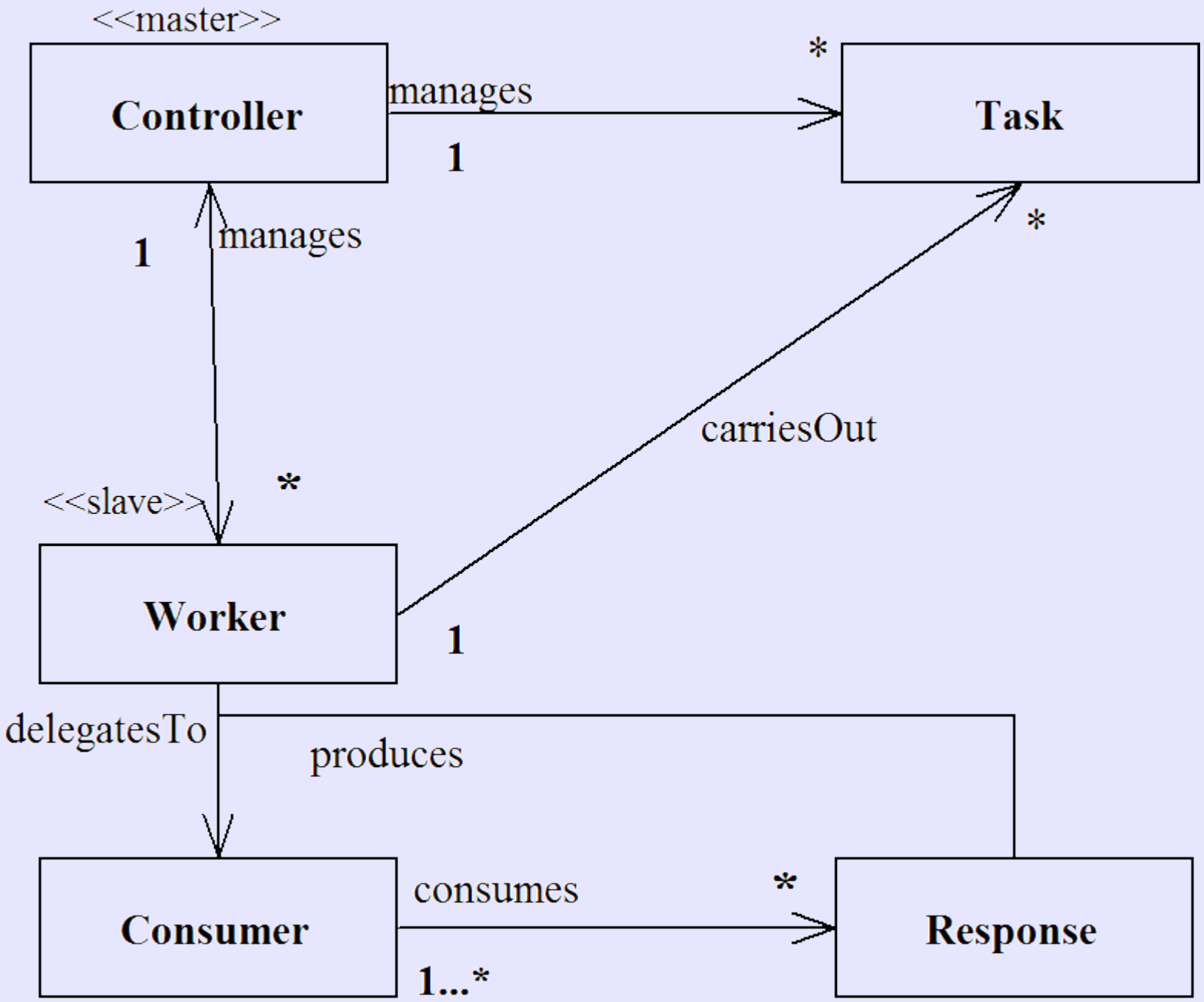
Introducing Slug

- Open Source multi-threaded web crawler
- Supports creation of crawler “profiles”
- Highly extensible
- Cache content in file system or database
- Crawl new content, or “freshen” existing data
- Generates RDF metadata for crawling activity
- Hopefully(!) easy to use, and well documented

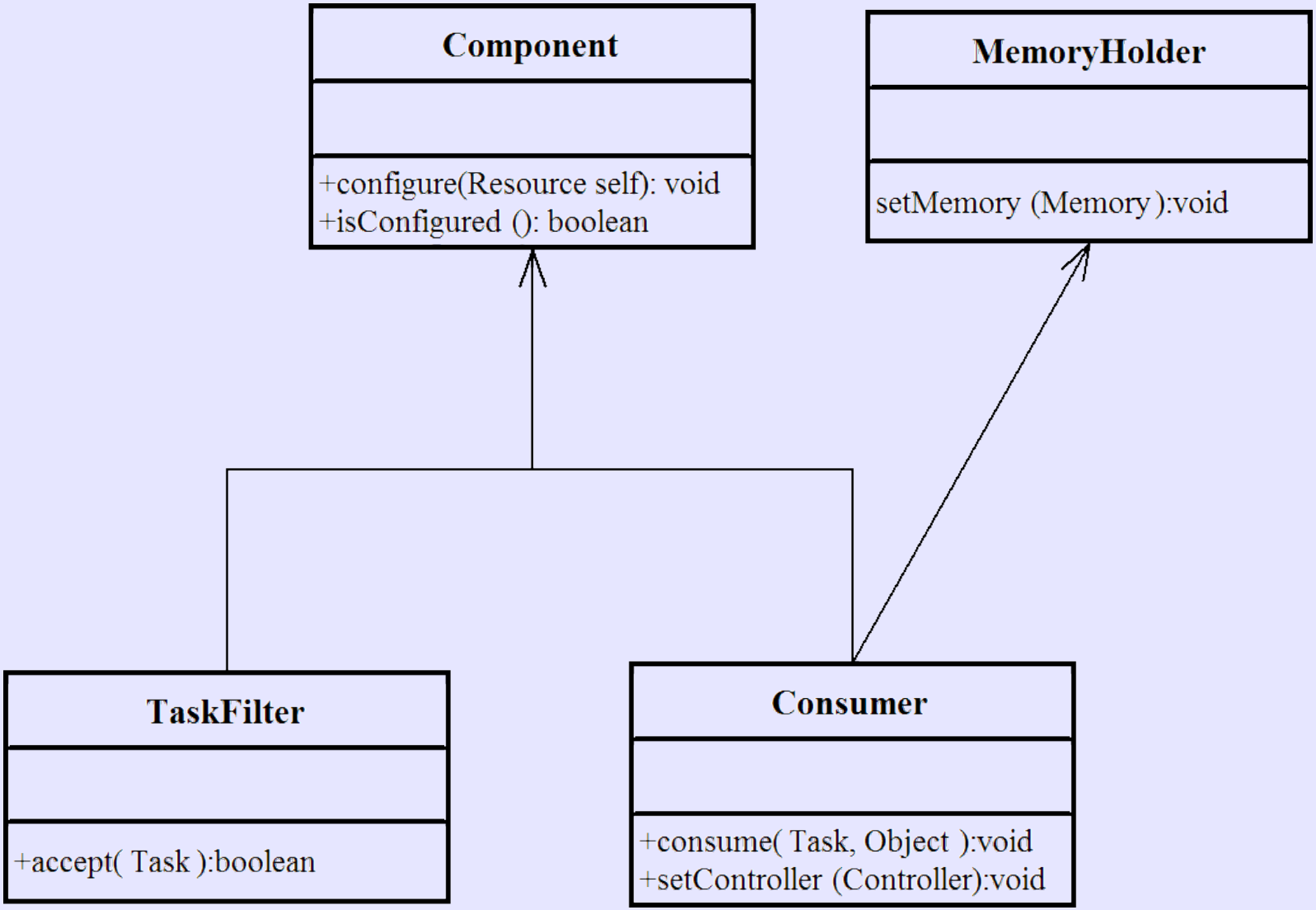
CRAWLER ARCHITECTURE

Crawler Architecture

- Basic Java Framework
  - Multi-threaded retrieval of resources via HTTP
  - Could be used to support other protocols
  - Extensible via RDF configuration file
- Simple Component Model
  - Content processing and task filtering components
  - Implement custom components for new behaviours
- Number of built-in behaviours
  - e.g. Crawl depth limiting; URL blacklisting, etc


<<master>>
Controller
manages
1
*
Task
1
manages
*
<<slave>>
*
carriesOut
Worker
1
delegatesTo
produces
Consumer
consumes
1...*
*
Response

Component Model



Consumers

- Responsible for processing results of tasks
  - Support for multiple consumers per profile
- `RDFConsumer`
  - Parses content; Updates memory with triple count
  - Discovers `rdfs:seeAlso` links; Submits new tasks
- `ResponseStorer`
  - Store retrieved content in file system
- `PersistentResponseStorer`
  - Store retrieved content in Jena persistent model

Task Filters

- Filters are applied before new Tasks accepted
  - Support for multiple filters per profile
  - Task must pass all filters to be accepted
- `DepthFilter`
  - Rejects tasks that are beyond a certain “depth”
- `RegexFilter`
  - Reject URLs that match a regular expression
- `SingleFetchFilter`
  - Loop avoidance; remove previously encountered URLs

CRAWLER CONFIGURATION

Scutter Profile

- A combination of configuration options
- Uses custom RDFS Vocabulary
- Current options:
  - Number of threads
  - Memory location
  - Memory type (persistent, file system)
  - Specific collection of Consumers and Filters
- Custom components may have own configuration

Example Profile

```
<slug:Scutter rdf:about="default">
  <slug:hasMemory rdf:resource="memory"/>
  <!-- consumers for incoming data -->
  <slug:consumers>
    <rdf:Seq>
      <rdf:li rdf:resource="storer"/>
      <rdf:li rdf:resource="rdf-consumer"/>
    </rdf:Seq>
  </slug:consumers>
</slug:Scutter>
```

Example Consumer

```
<slug:Consumer rdf:about="rdf-consumer">

 <dc:title>RDFConsumer</dc:title>

 <dc:description>Discovers seeAlso links in
  RDF models and adds them to task
  list</dc:description>


  <slug:impl>com.ldodds.slug.http.RDFConsumer
  </slug:impl>

</slug:Consumer>
```

Sample Filter

```
<slug:Filter rdf:about="depth-filter">
 <dc:title>Limit Depth of Crawling</dc:title>

  <slug:impl>com.ldodds.slug.http.DepthFilter
  </slug:impl>

  <!-- if depth >= this then url not
  included. Initial depth is 0 -->
  <slug:depth>3</slug:depth>
</slug:Filter>
```

Sample Memory Configuration

```
<slug:Memory rdf:about="db-memory">
 <slug:modelURI
   rdf:resource="http://www.example.com/test-model"/>

 <slug:dbURL>jdbc:mysql://localhost/DB</slug:dbURL>

 <slug:user>USER</slug:user>
 <slug:pass>PASSWORD</slug:pass>
 <slug:dbName>MySQL</slug:dbName>
 <slug:driver>com.mysql.jdbc.Driver</slug:driver>
</slug:Memory>
```


CRAWLER
MEMORY

Scutter Vocabulary

- Vocabulary for crawl related metadata
  - Where have I been?
  - What responses did I get?
  - Where did I find a reference to this document?
- Draft Specification by Morten Frederiksen
- Crawler automatically generates history
- Components can store additional metadata

Scutter Vocab Overview

- Representation
  - “shadow resource” of a source document
  - `scutter:source` = URI of source document
  - `scutter:origin` = URIs which reference source
  - Related to zero or more Fetches (`scutter:fetch`)
  - `scutter:latestFetch` = Most recent Fetch
  - May be skipped because of previous error (`scutter:skip`)

Scutter Vocab Overview

- Fetch
  - Describes a `GET` of a source document
  - HTTP Headers and Status
  - `dc:date`
  - `scutter:rawTripleCount`, included if parsed
  - May have caused a `scutter:error` and a `Reason`
- Reason
  - Why was there an error?
  - Why is a `Representation` being skipped?

List Crawl History for Specific Representation

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX scutter: <http://purl.org/net/scutter/>
SELECT ?date ?status ?contentType ?rawTripleCount
WHERE
{
  ?representation scutter:fetch ?fetch;
          scutter:source <http://www.ldodds.com/ldodds.rdf>.
  ?fetch dc:date ?date.
  OPTIONAL { ?fetch  scutter:status ?status. }
  OPTIONAL { ?fetch scutter:contentType ?contentType. }
  OPTIONAL { ?fetch scutter:rawTripleCount ?rawTripleCount. }
}
ORDER BY DESC(?date)
```


WORKING WITH
SLUG

Working with Slug

- Traditional Crawling Activities
  - E.g. Adding data to a local database
- Maintaining a local cache of useful data
  - E.g. Crawl data using file system cache
  - ...and maintain with "`-freshen`"
  - Code for generating LocationMapper configuration
- Mapping the Semantic Web?
  - Crawl history contains document relationships
  - No need to keep content, just crawl...

Future Enhancements

- Support the Robot Exclusion Protocol
- Allow configuration of the User-Agent header
- Implement throttling on a global and per-domain basis
- Check additional HTTP status codes to "skip" more errors
- Support white-listing of URLs
- Expose and capture more statistics while in-progress

Future Enhancements

- Support Content Negotiation to negotiate data
- Allow pre-processing of data (GRDDL)
- Follow more than just `rdfs:seeAlso` links
  - allow configurable link discovery
- Integrate a “smushing” utility
  - Better manage persistent data
- Anything else?!


Questions?
http://www.ldodds.com/projects/slug
leigh@ldodds.com

Attribution and Licence

The following images were used in these slides

http://flickr.com/photos/enygmatic/39266262/

http://www.flickr.com/photos/jinglejammer/601987

http://www.flickr.com/photos/sandyplotnikoff/105067900

Thanks to the authors!