Thywin – Design Document

Geschreven door: Bobby Bouwmann, Thomas Gerritsen, Thomas Kooi, Erwin Janssen, Imre Woudstra

Datum: 14-5-2014

Project begeleider: Alice de Groot

Opdrachtgever: Joost Kraaijeveld

Table of contents

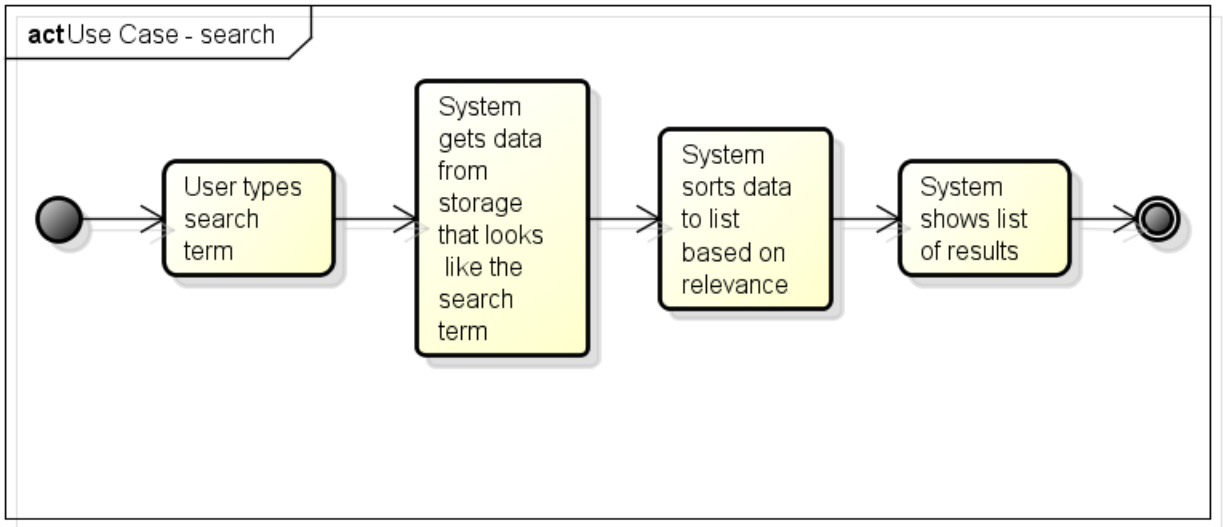
Use Cases

Use Case 1: Search

The user goes to http://thywin.com. The browser then shows a search page. The user types a search term in the search field. The server then shows all the results based on the search term.

Primary Actor: User	
Stakeholders: Site owners	
Preconditions: The site is online and the database is available.	
Post conditions: The user gets results based on their search term.	
Main success scenario:	
1. User goes to **http://thywin.com**.	2. Webserver sends page back.
3. User types search term in the search field.	4. Webserver shows results based on search term.
Extensions: (or Alternative flow)	
	[If no results] 4a. Webserver shows message “0 results”.

Activity diagram

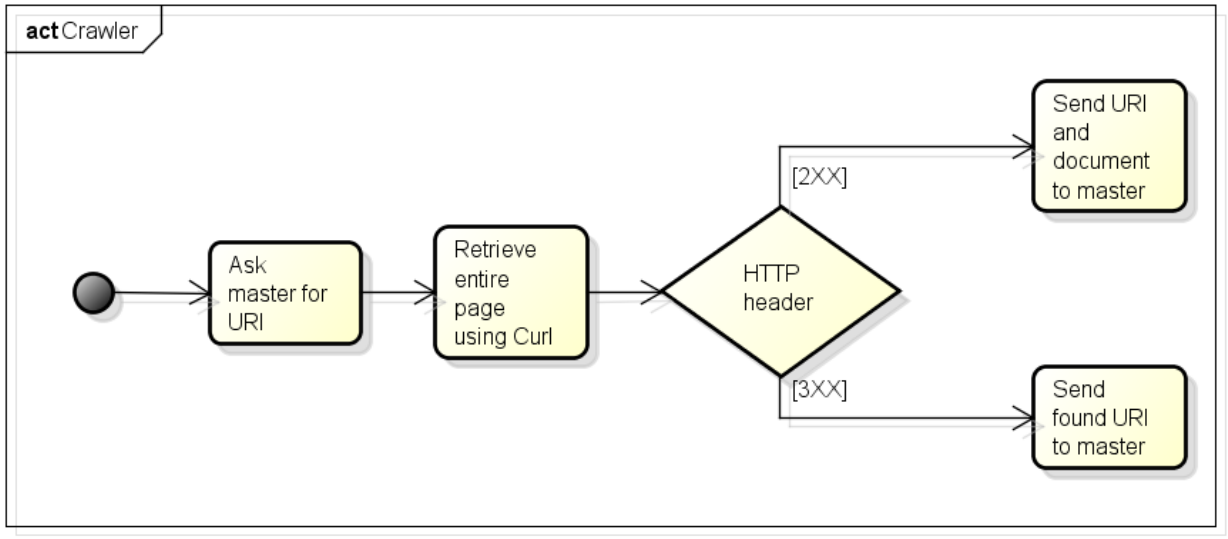


Use Case 2: Crawler

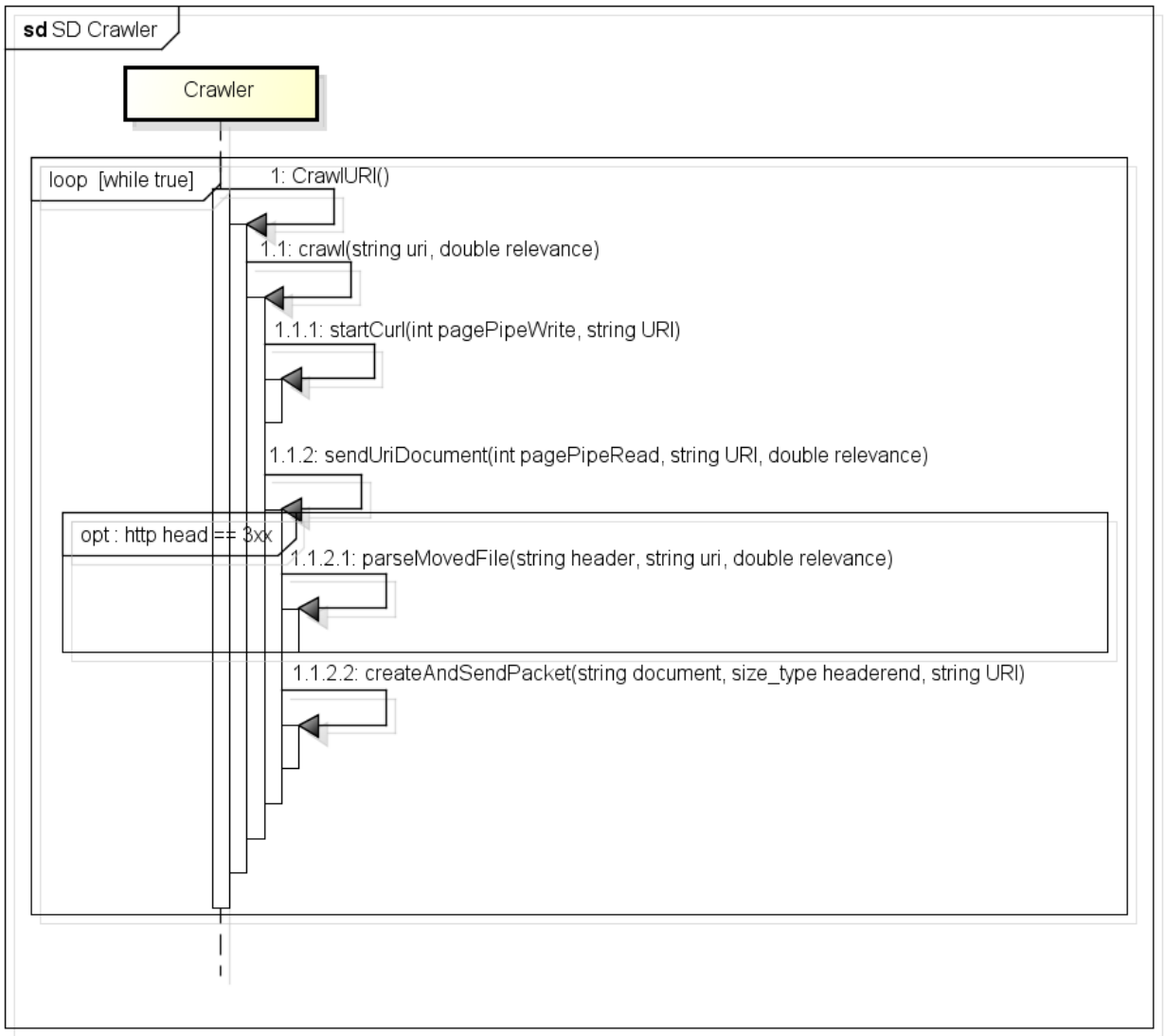
The crawler asks the master for an URI. The master then sends a URI to the crawler. The crawler then uses Curl to get the data from the URI. This crawled data and the URI will be send back to the master.

Primary actor: Crawler	
Stakeholders & interest: Scheduler, Parser	
Preconditions: Connection to the master, connection to the internet	
Main success Scenario	
1. Crawler asks master for a URI to crawl.	2. The master sends an URI to the crawler.
3. Crawler gets the data from the URI using Curl.	
4. Crawler sends the URI and the crawled data to the master.	
Extensions: (or Alternative flow)	
3a. Curl shows a HTTP 300 message and gets a new URI.	
4a. The crawlers sends the new found URI to the master.	

Activity diagram



Sequence Diagram

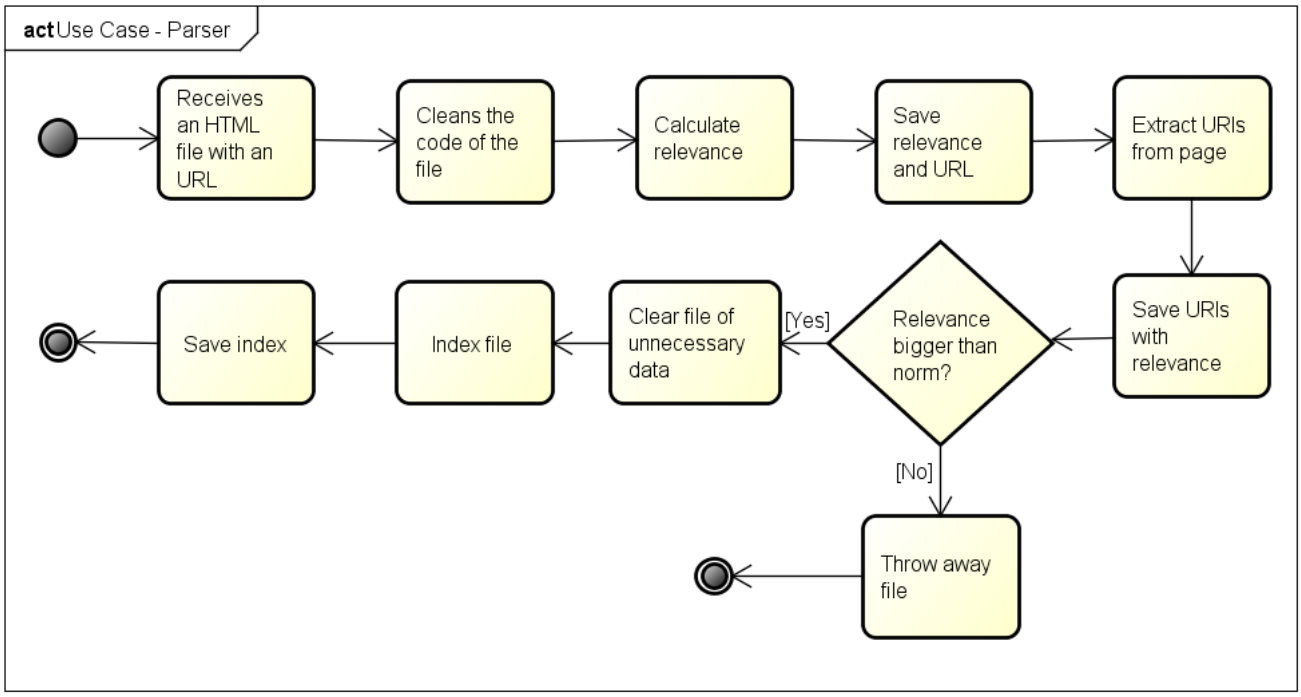


Use Case 3: Parsen

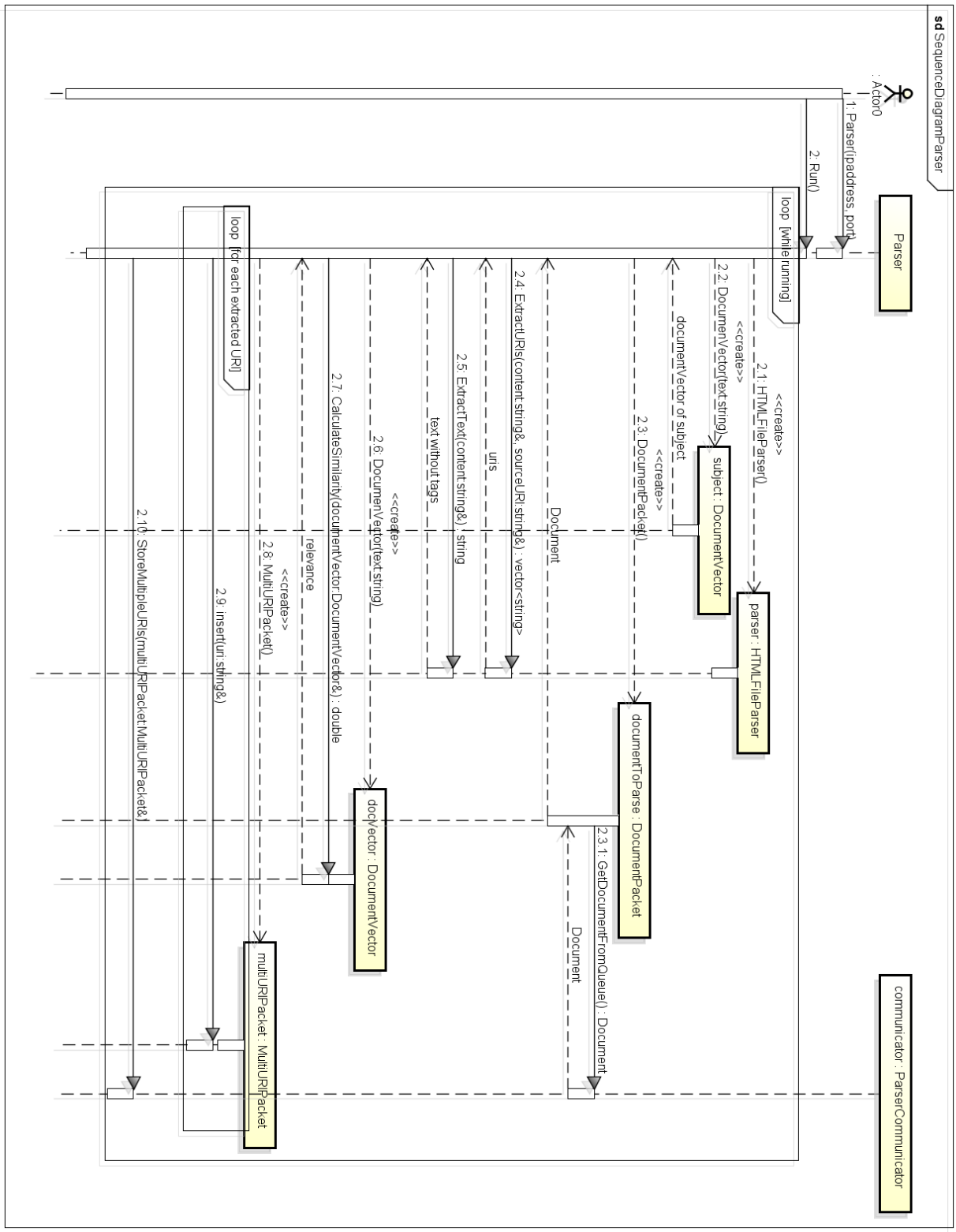
The parser requests an document and the related URI from the master. The parser then extracts all the URI's from the document. The parser then removes the html tags. The parser is now ready to determine the relevance of the document. A document vector is generated with all the words and with their word occurrence, also called index. After this is done the parser sends the found URI's and the index to the master.

Primary Actor: Parser	
Stakeholders: Master, Crawler, Database	
Preconditions: Connection with the master, file is HTML format	
Post conditions: File is indexed and the index, URIs, index and relevance is send to the master	
Main success scenario:	
1. The parser requests a document from the master.	2. The master send the URI and document to the parser.
3. The parser extracts the URI's from the document.	
4. The parser removes the html tags.	
5. The parser determines the relevance of the document and stores all the words in an vector with their occurrence(index).	
6. The parser sends the URI's with the relevance and the index to the master.	
Extensions: (or Alternative flow)	

Activity diagram



Sequence diagrams



Use Case 4: Master

Get URI

Primary Actor: Client	
Stakeholders: client	
Preconditions: System is in waiting state.	
Post conditions: De client has an URI. The URI queue has an element less.	
Main success scenario:	
1. Client asks for an URI.	2. System gets the first element of the URI queue.
	3. System sends the URI to the crawler.
	4. System removes the URI from the URI queue.
Extensions (Alternative flow)	
	[No element in queue] 2. System gets an URI of the default Queue (start point).

Put URI

Primary Actor: Client	
Stakeholders: Client	
Preconditions: System is in waiting state.	
Post conditions: The URI queue has one more element.	
Main success scenario:	
1. Client sends an URI to the system.	2. System checks if the URI exists in the queue.
	3. [URI is Unique] System put URI in the queue.
Extensions (Alternative flow)	
	[URI already exists in URI queue] 3a. System does nothing with the URI.

Get Document

Primary Actor: Client	
Stakeholders: Client	
Preconditions: System is in waiting state.	
Post conditions: The client has a document from the document queue. The document queue has one element less.	
Main success scenario:	
1. Client asks for a document	2. System gets the first document from the document queue.
	3. System sends the document to the parser.
	4. System removes the document from the document queue.
Extensions (Alternative flow)	
	[No document in document queue] 2a. System waits until the document queue has a new document.

Put Document

Primary Actor: Client	
Stakeholders: Client	
Preconditions: System is in waiting state.	
Post conditions: The document queue has one more element.	
Main success scenario:	
1. Client send a document to the system.	2. System checks if the document exists in the document queue.
	3. [document is unique] Systeem plaatst Document in de queue.
Extensions: (or Alternative flow)	
	[document already exists in document queue] 3a. System does nothing with the document.

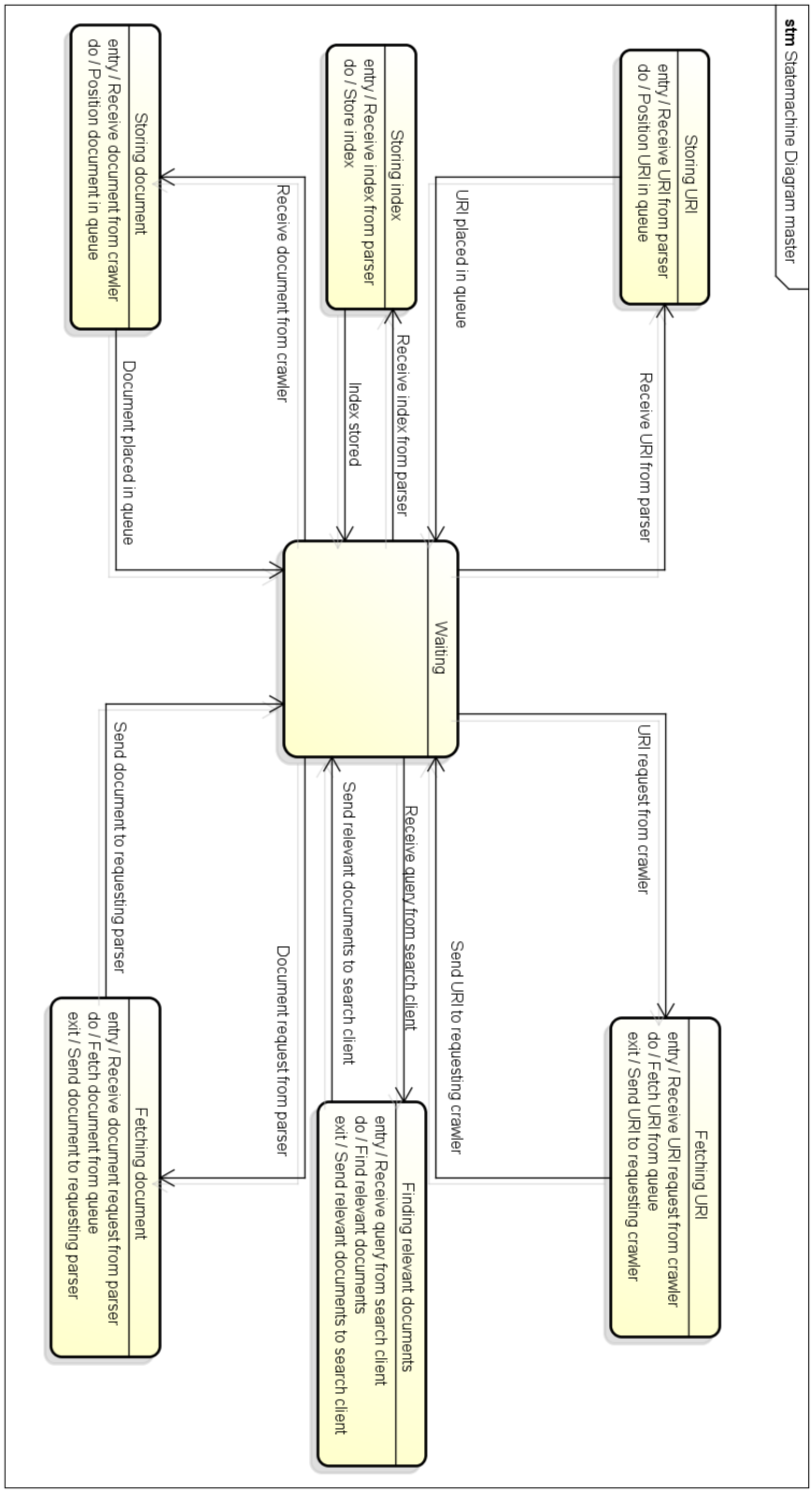
Put MultiURI's

Primary Actor: Client	
Stakeholders: Client	
Preconditions: System is in waiting state.	
Post conditions: The URI queue has one or more element added.	
Main success scenario:	
1. Client sends a list of URI's to the system.	2. Continue in use case Put URI
Extensions (Alternative flow)	

Put DocumentVector

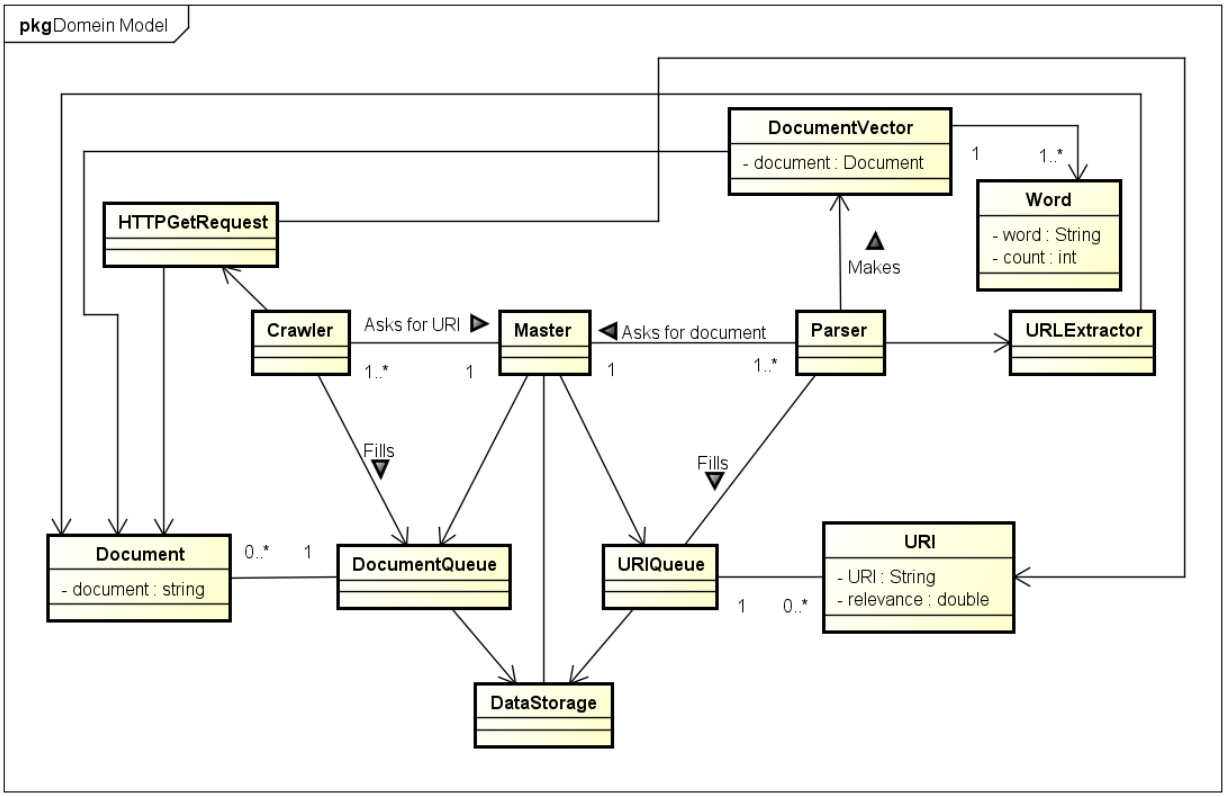
Primary Actor: Client	
Stakeholders: Client	
Preconditions: System is in waiting state.	
Post conditions: The indices table is updated	
Main success scenario:	
1. Client sends a list of words with their occurrences and the URI of that page to the master	2. The master stores the words with their occurrences and URI id in the database.
Extensions (Alternative flow)	

State diagram

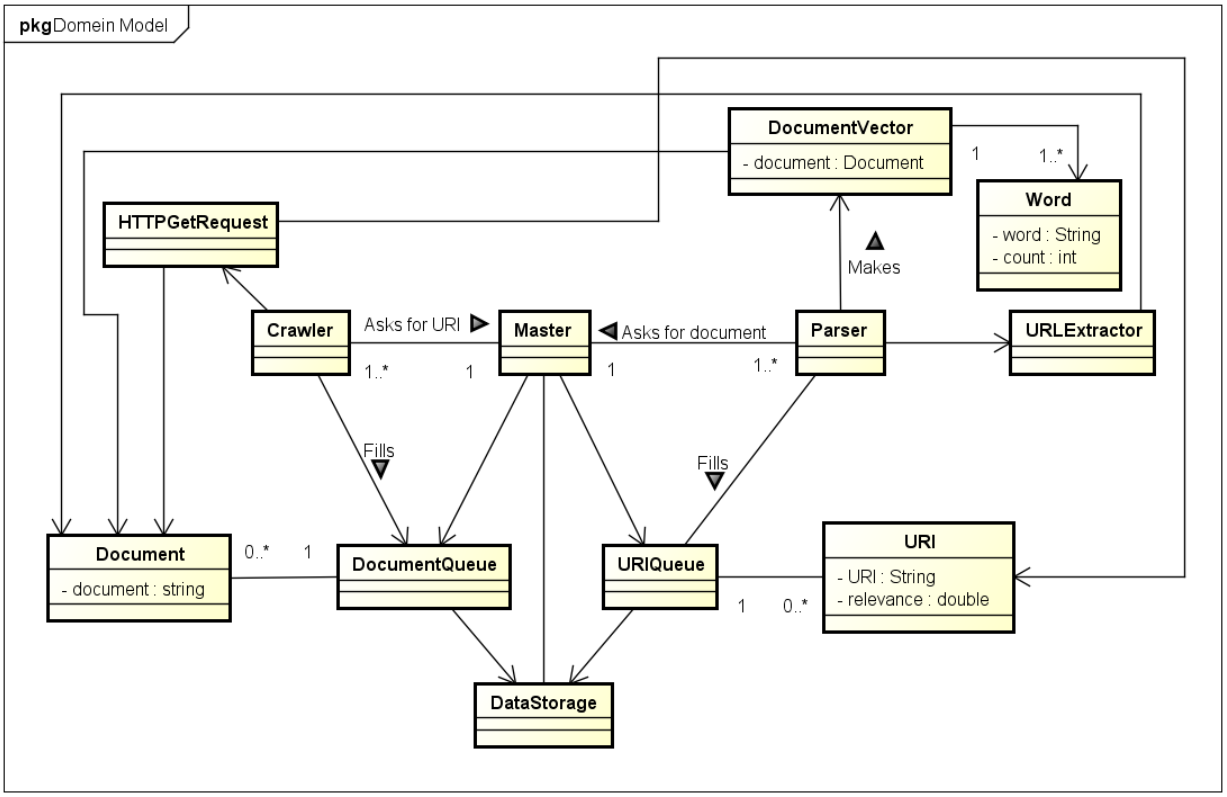


Domain Model

Versie 1.1:



Versie 1.0:


pkgDomein Model
DocumentVector
- document : Document
1
1..*
Word
- word : String
- count : int
HTTPGetRequest
Makes
Crawler
Asks for URI
Master
Asks for document
Parser
URLExtractor
1..*
1
1
1..*
Fills
Fills
Document
- document : string
0..*
1
DocumentQueue
URIQueue
URI
- URI : String
- relevance : double
1
0..*
DataStorage

Class Diagrams

Dit hoofdstuk beschrijft de class diagram ontwerpen voor de verschillende onderdelen van het Thywin Project. Wij hebben er voor gekozen om de verschillende taken binnen het algehele proces op te splitsen en te verdelen over andere processen die schaalbaar zijn.

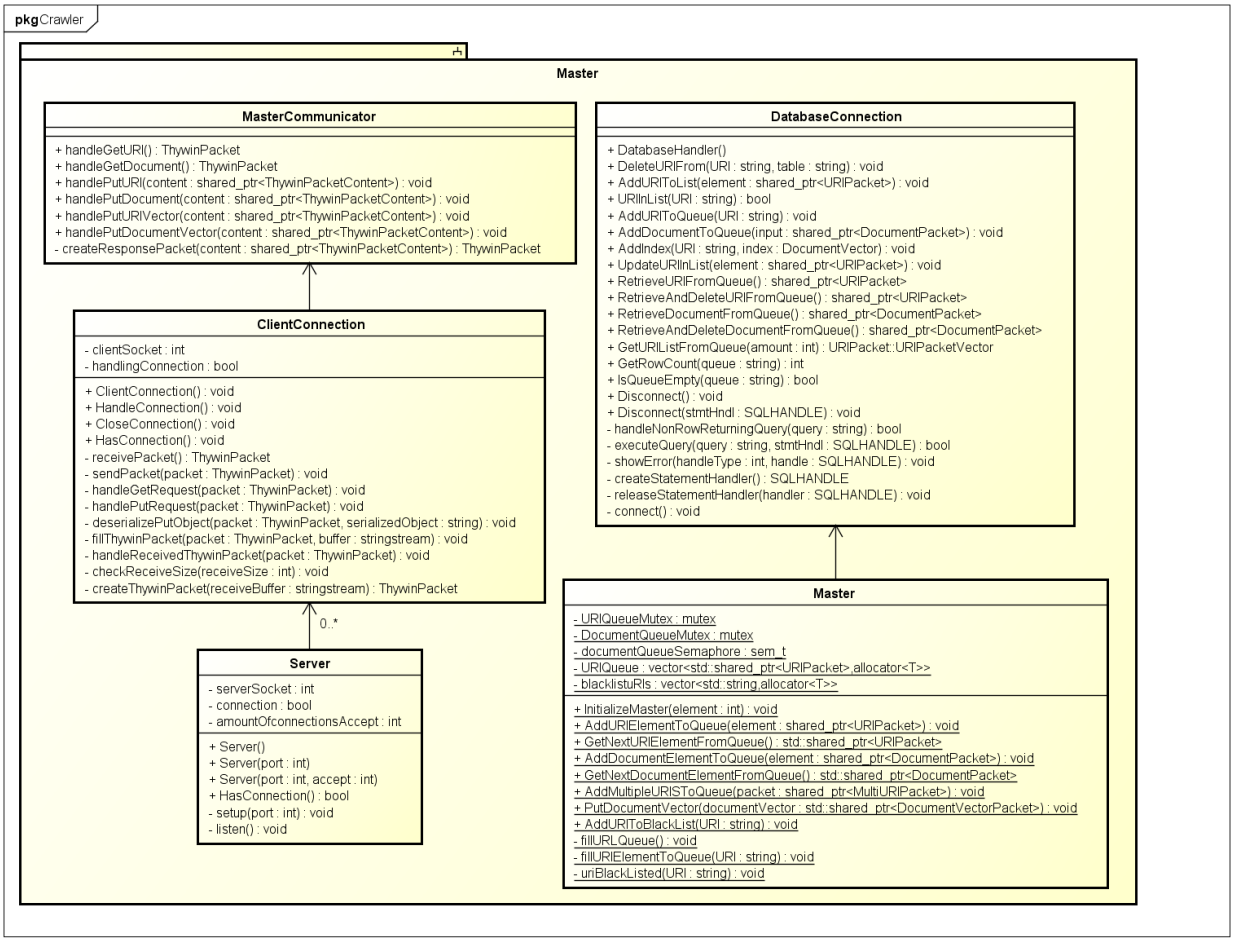
Deze processen zijn:

- Master Proces
- Parser Proces
- Crawler Proces
- Search Engine Proces

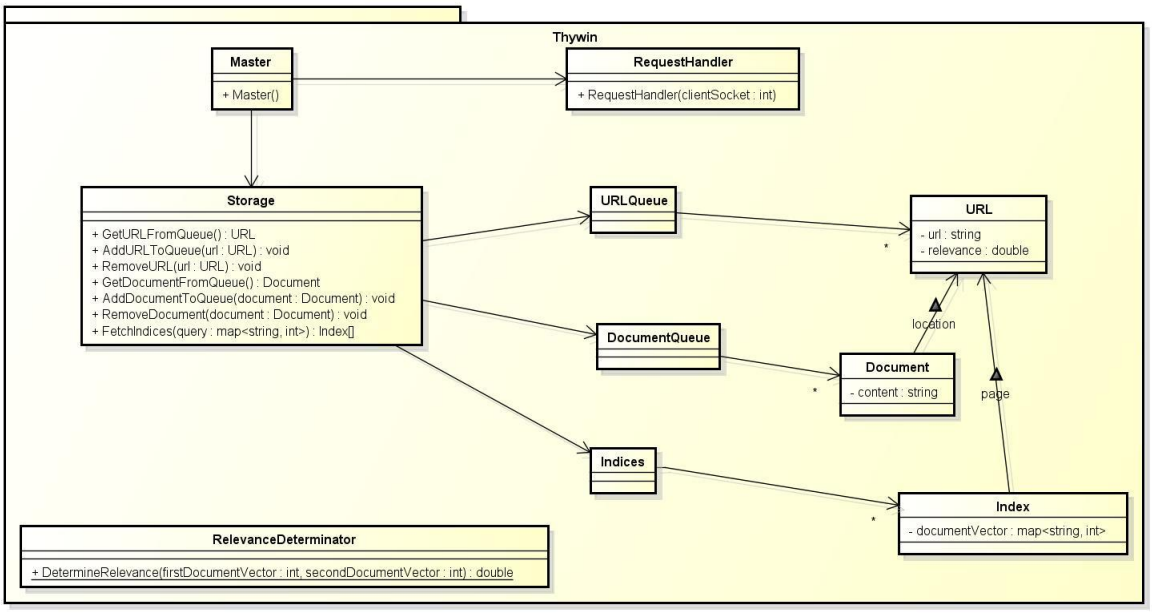
Master

Het Master process heft als taak het bij houden van de queues en opslaan van informatie. Het zorgt er voor dat de juiste URIs worden door gegeven en dat er indexen worden opgeslagen in de database.

Versie 1.1 (huidig):



Versie 1.0:

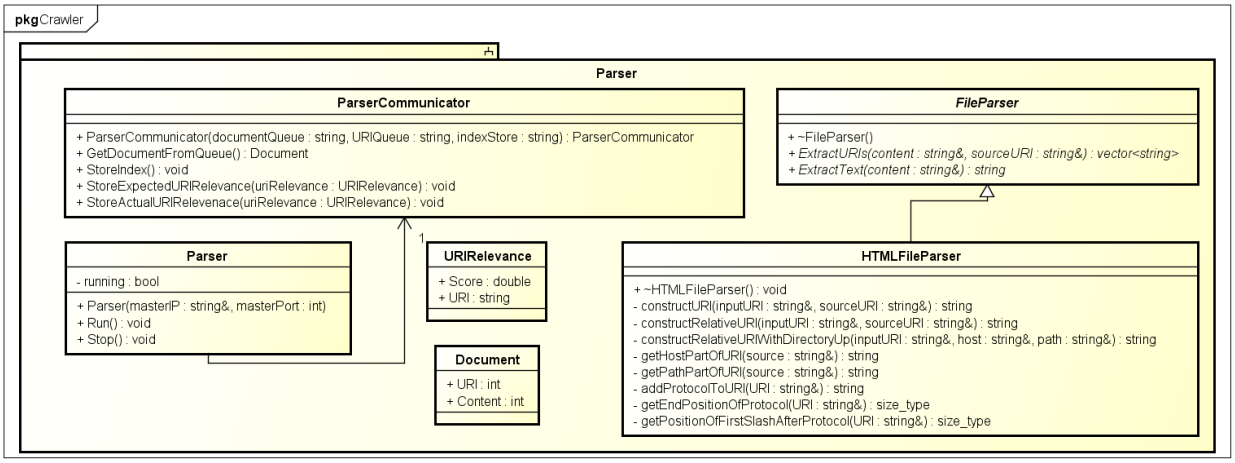

Thywin
Master
+ Master()
RequestHandler
+ RequestHandler(clientSocket : int)
Storage
+ GetURLFromQueue() : URL
+ AddURLToQueue(url : URL) : void
+ RemoveURL(url : URL) : void
+ GetDocumentFromQueue() : Document
+ AddDocumentToQueue(document : Document) : void
+ RemoveDocument(document : Document) : void
+ FetchIndices(query : map<string, int>) : Index[]
URLQueue
URL
- url : string
- relevance : double
DocumentQueue
Document
- content : string
location
page
Indices
Index
- documentVector : map<string, int>
RelevanceDeterminator
+ DetermineRelevance(firstDocumentVector : int, secondDocumentVector : int) : double

Parser

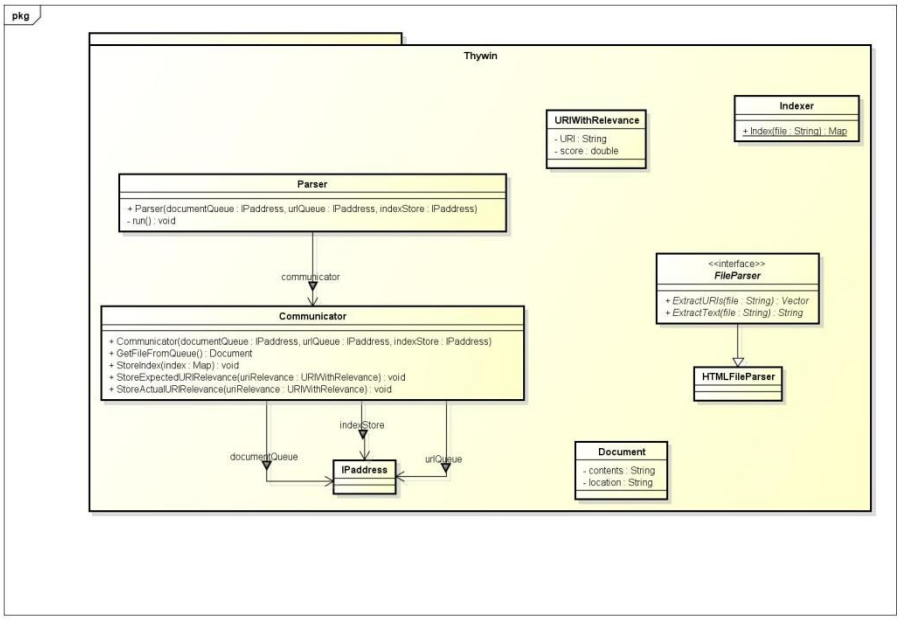
De Parser heeft als taak het verwerken van een opgehaald document. Dit betekend: Het tellen van unieke worden en het verzamelen van URIs van een document en deze door geven aan het Master proces voor opslag.

Versie 1.2 (huidig):

Versie 1.1:



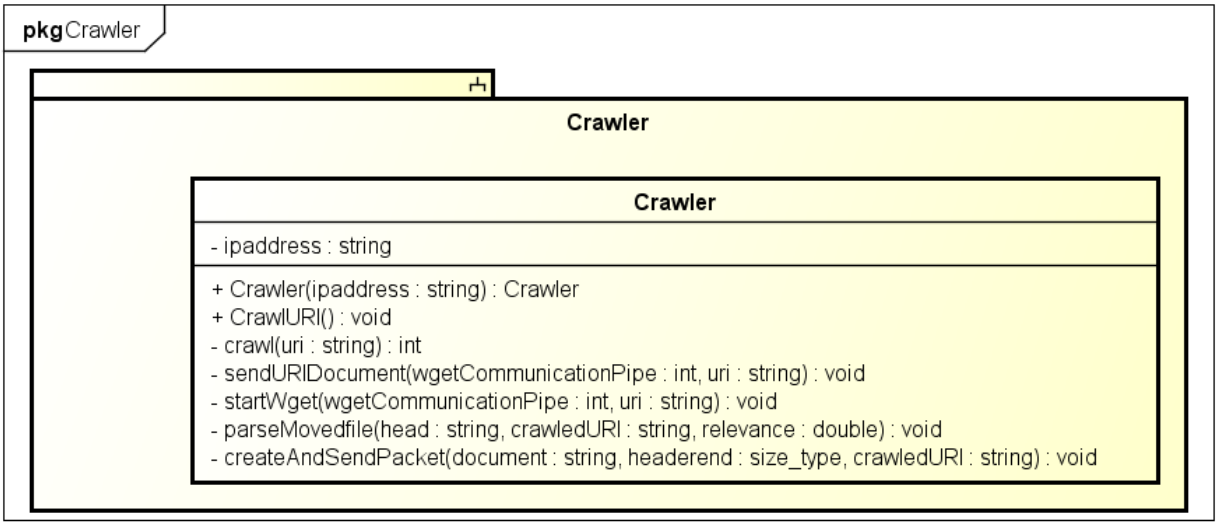
Versie 1.0:



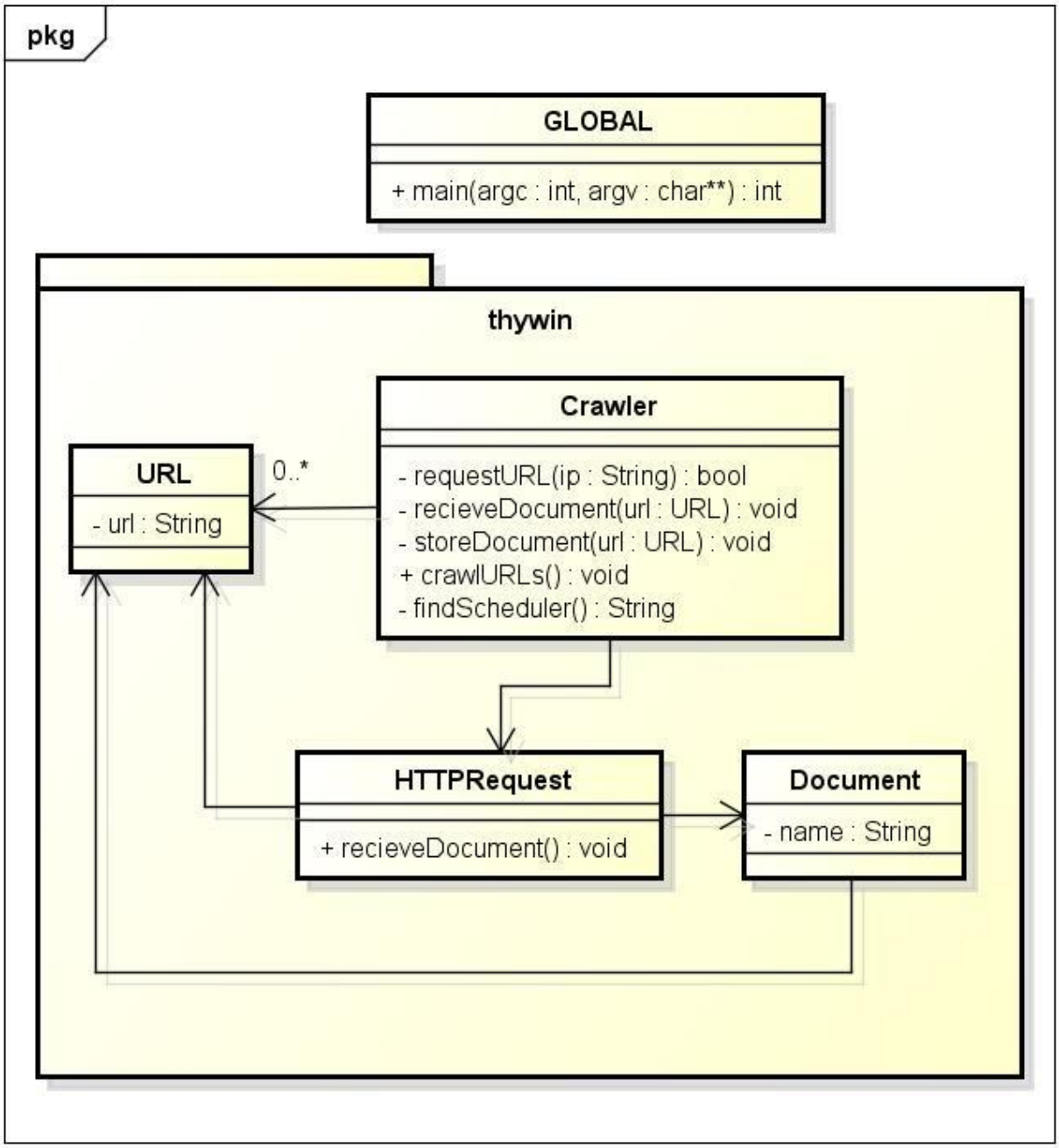
Crawler

De Crawler zorgt er voor dat het document bijbehorend bij een URI opgehaald wordt van het web.

Versie 1.1 (huidig):

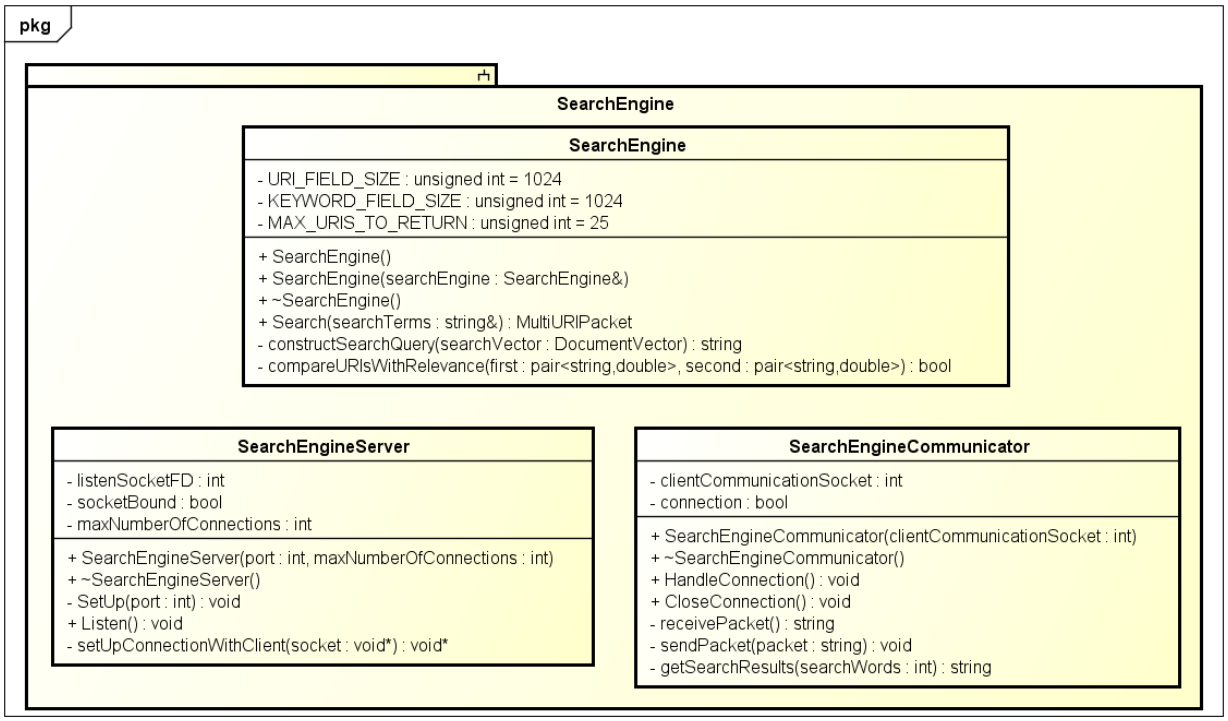


Versie 1.0:


pkg
GLOBAL
+ main(argc : int, argv : char**) : int
thywin
Crawler
- requestURL(ip : String) : bool
- recieveDocument(url : URL) : void
- storeDocument(url : URL) : void
+ crawlURLs() : void
- findScheduler() : String
URL
- url : String
0..*
HTTPRequest
+ recieveDocument() : void
Document
- name : String

Search Engine

Versie 1.0:



Thywinlib

Versie 1.0: