Operational Concept Document
For the
Semantic Web Crawler
20 March 2012
Prepared by:
Michael Bastos
Davendra Patel
Manuel Covarrubias Jr

Approvals

Title	Printed Name	Signature	Date
Client Representative	Dr. Appel		
Project Manger	Manual Covarrubias Jr.		
Project Leader	Davendra Patel		
Project Leader	Michael Bastos		

Revision Changes

The revision change record will contain the revision number, date of revision, engineering change order (ECO) number, description of what was modified, added or deleted, and the individual's name responsible for the change.

Revision Change Record

Revision Number	Revision Date	ECO Number	Description of Change	Author
0	13-Mar-2012		Original	Manager/Leaders
1	20-Mar-2012		Revision	Manager/Leaders

Table of Contents

1.0 Introduction

This document comprises the Operational Concept Document (OCD) for development of the Semantic Web Crawler (SWC) Product.

1.1 Purpose

The purpose of this OCD is to explain the SWC both internally and externally, give proper use cases as well as determine the direction of the project for the client. The SWC Product scans or 'crawls' through Internet pages to create an index of the data that it is looking for. The SWC will be programmed for long term usage so that the core can require very little maintenance down the road and all the real changes should be made to the User Interface (UI) which is built separately. It will provide the client relevant data from websites the client predefines. The primary purpose of the web crawler is to collect data so that when the client points it at a website, it can provide the client relevant and immediate information in either Excel or Comma Separated Value (CSV) formats pulled directly from that site.

The intended audience for this OCD is for the client: **Professor Dr. Jeffery Appel.**

But other audience may also include:

- SWC Development Team
- Senior Management
- Investors

This document's objective is to provide a comprehensive description of the software objectives, unfamiliar terms will be defined, and make references to any additional documents to provide further background. The content of this document will be readable and relevant to the audience no matter what their background or role is. The level of technical jargon is kept at a minimum for the majority of the document, especially at the beginning when the product's scope and vision are discussed. However, the level of technical expertise necessary to fully understand the document increases when the document switches to specific functions. This document talks about software, so naturally there will be some level of technical information included.

1.2 Glossary

This subsection will provide the definitions of all items, acronyms, and abbreviations required to properly interpret the OCD.

1.2.1 Definitions

Amazon EC2	A Web-based service that allows subscribers to run application programs in the Amazon.com computing environment
Amazon S3	A service provided by Amazon.com that allows web designers to store huge amounts of data online

Bandwidth	Refers to the amount of data that can be sent through a network or modem connection
C/C++	A general-purpose programming language that is statically typed, free-form, multi-paradigm, and compiled
Cache	A location used to store data in order to have faster subsequent retrievals
Cloud Based Environment	Refers to applications and services offered over the Internet. Data centers that offer services are referred to as the 'cloud'
Command Line Interface	A means to interact with software by inputting commands to perform tasks
Compile	Source code that is translated into machine code or object code
Core	Memory that consists of tiny doughnut shaped masses of magnetic material
Database	Collection of data/information that is organized so that a computer program can quickly retrieved the desired data/information
Dynamic	An action taking place at the moment rather than in advance. Opposite of static
Eucalyptus	An open-source software infrastructure for implementing cloud computing on clusters
Git	A distributed version control system that is free and open source. Designed to handle any sized projects with efficiency and speed
Hash	A function that maps large sets of data into smaller sets of data
Java	An object-oriented programming language that is structured around classes instead of functions
Multi Threading	A technique in which a single set of code can be used by several processors at different stages of execution
MySQL	A RDBMS that runs as a server to provide multi-client access to a number of databases
Object Oriented	Type of programming that creates re-usable objects by combining data structures and functions

Open Source	Computer software that is available in source code form to permit users to study, change, improve, and distribute the software
PHP	Stands for PHP Hypertext Preprocessor. A general-purpose scripting language that is particularly suited for Web development and can be embedded into HTML
Parse	To analyze or separate components so they can be more easily processed
Perl	A programming language designed to process text. Short for Practical Extraction and Report Language
Process	A section of a running software program or operation that performs a single task
Python	A portable and interpreted object-oriented programming language
Ruby	A object-oriented programming language
Scalability	A computer application or product that has the ability to maintain functionality when its size changes to meet the client need
Script	A list of commands that are executed by a program or scripting engine
Scripting Language	A simple programming language used to write scripts
Semantic	The meaning of a string in a programming language
Server	A computer used to manage data storage or as a network communications resource. A server provides and organizes access to the resources for the computers that are linked to it
Static	Elements that are fixed and cannot be changed or capable of action. Opposite of dynamic.

1.2.2 Acronyms and Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
COTS	Commercial Off-The-Shelf
CSV	Comma Separated Values
ECO	Engineering Change Order
GB	Gigabyte
GUI	Graphical User Interface

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol with Secure Sockets Layer
IDE	Integrated Drive Electronics
I/O	Input/Output
IP	Internet Protocol
MB	Megabytes
Mbps	Megabits per second
MHz	Megahertz
OCD	Operational Concept Document
OS	Operating System
PHP	PHP Hypertext Preprocessor
RAM	Random Access Memory
regex	regular expression
RDBMS	Relational Database Management System
RDS	Relational Database Service
SVGA	Super Visual Graphics Array
SVN	Subversion
SWC	Semantic Web Crawler
UI	User Interface
VGA	Visual Graphics Array

1.3 References

Performance Comparison - C++ / Java / Python / Ruby/ Jython / JRuby / Groovy by Dhananjay Nene (2008). http://blog.dhananjaynene.com/2008/07/performance-comparison-c-java-python-ruby-jython-jruby-groovy/

The Debian Foundation - The Computer Language Benchmarks Game - January 2012 http://shootout.alioth.debian.org/

Pressman, Roger S., Software Engineering: A Practitioner's Approach, Seventh Edition, McGraw Hill, 2010.
Used for definitions. www.webopedia.com accessed 15 March 2012.

What is Application Software? www.techopedia.com accessed 8 March 2012.

1.4 Overview

The OCD will include the abstract definition of the SWC Product and involves the client of the product. The crawler will be a web based and server backed application that will utilize client specifications and requirements that will allow the client to specifically pinpoint the actual search parameters that are useful to the client or the client's client. Additionally, the OCD will describe the potential solutions and the specific strategies that are implemented in the overall product. Processes that are involved in the product include entering a website to extract specific data and save it into a database for future retrieval on a cached basis that will be updated per the

client's requirements via the UI. The OCD will engage the hardware, software, and the eventual client requirements. The remainder of the OCD will be further defined in the following sections:

- Section 2.1 provides a definitive statement of the problem to be solved by this product.
- Section 2.2 identifies the general characteristics of the eventual user.
- Section 2.3 identifies the solution strategies for the system.
- Section 2.4 details the external processes provided by the product.
- Section 2.5 identifies hardware, software, and workplace requirements and constraints.
- Section 2.6 details the product features that will be available in Build 1 (Prototype), Build 2 (Modest) and Build 3 (Enhanced).
- Section 2.7 establishes high-level criteria to pass the test phase and delivery requirements.

2.0 Product Definition

The Semantic Web Crawler allows the client to give the application a web address and some specific variables to quickly and easily crawl hundreds of pages of a website in a matter of minutes parsing and saving only what data is valuable to the client saving it in either a database or CSV and Excel spreadsheet format. With proper caching this process can be repeated hundreds of thousands of times with hundreds of websites in various ways allowing the client to better narrow down the type of data they wish to pull from the website without necessarily impacting the crawled site's overall performance or bandwidth.

A software solution is required to support

- High scalability.
- High performance.
- Support for all conventional aspects of crawling automation.
- Maintain relevant crawled data statically.
- Maintain required semantic crawled information in databases.
- Readily accept new parsing commands through the Application Programming Interface (API).
- Use standard Crawling and data collection protocols.
- Use web based or client side Graphical User Interface (GUI) for ease of client deployment.

2.1 Problem Statement

2.1.1 Background

There is always readily available data online that can be useful if it can be crawled and parsed in an accurate and timely fashion. Whether the client is an e-commerce company trying to see how much their competitors are charging for a large set of products or a real estate agent trying to collate foreclosure property data and want to immediately crawl other sites like zillow.com and crime statistic information for clients, having a crawler that is both versatile and fast is a necessary but absent solution in most cases.

2.1.1.1 Problem

The problem with most modern web crawlers is that they are built upon a scripting language of some form, whether it's python, perl, PHP Hypertext Preprocessor (PHP), or ruby, most scripting languages have advantages and disadvantages when designing a web crawler. Few crawlers are ever built in lower level languages such as C or C++ because it is believed that having the freedom to modify the crawler outweighs any performance increase the system may gain from having it already compiled in a production setting. Java is one of those few exceptions in that the Java run time engine makes up for any lag created with being able to modify the code and check for changes directly. This way the interpreting is only done once in the application's life cycle unless it changes. The advantages to using a scripting language are that the crawler's parsing can be directly modified to be semantic and to only pull the requested data from the site, the crawler can also be easily fixed and modified based on the needs of the client without the need to compile it and can be changed on an as need basis. Most scripting languages are easy to learn and thus the cost of labor and maintenance is significantly reduced. Many of these advantages are also disadvantages for a scripted crawler. Scripts tend to run slower because they have to be fed through an interpreter and this can be a time hit when deciding how quickly the customer wants the crawler to perform its actions. In most cases a crawler's resources are not necessarily as important as the system may want to mimic a client visiting a website and thus don't want the crawler to run through its steps too quickly. Though when resources are measured in usage increments or seconds such as in a cloud based environment, how fast or efficiently a crawler operates may impact a real physical cost in the time used to run the servers.

2.1.1.2 Solution

The SWC will create a compiled crawler in C++ with a Java User Interface and will be versatile enough to compete with anything built in a scripting language, but as fast as some of the most powerful crawlers. Since the team already knows many of the things that a crawler will want to do, it can be built directly into the program's object oriented core. Using a command line interface or API to communicate with the core, the crawler will have everything it needs remotely to operate at highly complex speeds allowing it to be both dynamic in its structure and rigid in its form and function.

2.2 User Characteristics

The end user characteristics are somewhat diverse. The pool of clients will include personal requirements for entertainment, research, and any number of utilities the client can think of in the area of searching for specific information. From a professional perspective a business client may require data extraction that will expand the business model of the client's company as well as reduce labor costs for information gathering of the website content. In the event that the client is seeking data for entertainment purposes, this client may have a relatively low experience with computer technology thus the use of a relatively easy to understand GUI is required. On the other hand, a business power client may have extensive knowledge of the computer environment and may want to quickly modify the crawl commands directly to the core or create templates for repetitive crawls. It is because of these different use cases that the Development Team will need to be able to deliver a sound requirement package to the client. Conclusively, the client of the

product will possess the need for data that is relevant to their specific personal or business need and the final product will be able to tailor itself to the client's preferences.

2.3 Solution Strategy

This section deals with the types of strategies that the Development Team can use for the SWC Product. This section will explain the three types of solutions the Development Team looked into. The three types of solution strategies are: code development, Commercial Off-The-Shelf (COTS), and hybrid.

2.3.1 Code Development

2.3.1.1 Procedure

Application software is a program or group of programs designed for the clients. These programs are divided into two classes: system software and application software. While system software consists of low-level programs that interact with computers at a basic level, application software resides above system software and includes database programs, word processors, spreadsheets, etc. Application software may be grouped along with system software or published alone. Software can be developed for a variety of purposes. The three most common purposes are:

- Meet the specific needs of a specific client/business (custom software).
- Meet a perceived need of some set of potential clients (commercial and open source software).
- For personal use.

Custom software development is aimed at clients who are not satisfied with market-available software that might not cover all of their needs or offer unnecessary functionality. The aim of fully developed software is to create software that would cover all of the client's requirements.

2.3.1.1.1 Advantages

- The software exactly covers all of the client's specific needs. The software adapts to the specific processes and the development of the UI and system administration is in accordance to future client's needs.
- System Scalability. The further expansion of the software is possible at any time. Since the software was custom-built, there is no waiting time for a new version.
- Software Architecture. The software is designed to effectively utilize the system infrastructure and to ensure fast access to all information that is needed.
- Methodology and technologies used when developing software.
- Maximum flexibility in design and implementation.

2.3.1.1.2 Disadvantages

- High cost to develop.
- Development and deployment schedule may be extremely lengthy.

- To optimize operations, coding expertise is required in several programming languages.
- Intensity of work.
- The high risks involved with original software development.
- The total commitment of the management in quality systems. Solutions that are not implemented can be frustrating.

2.3.1.2 Feasibility

This solution is not feasible for implementation because of the high risk of exceeding scheduling, budget constraints, and target goals. Even though it is advantageous to meet and fulfill the needs of the client by coding the SWC Project, the risk of a late delivery and cost overruns far exceed its benefits.

2.3.2 Commercial Off-The-Shelf (COTS)

2.3.2.1 Procedure

COTS-Based Software Development has emerged as an approach aiming to improve a number of drawbacks found in the software development industry. The main idea is the reuse of well-tested software products, known as COTS components, that will be assembled together in order to develop a better software product. One of the most critical activities in COTS-based development is the identification of the COTS candidates to be integrated into the software under development. Nowadays, the Web is the most used means to find COTS candidates. Thus, the use of search engines turns out to be crucial.

2.3.2.1.1 Advantages

- Reduced costs for application development and implementation.
- Shorter development time while ensuring quality.
- Low risk of application functional failure.
- Quicker time to market.
- Better reliability.
- More end user functionality when compared to custom-developed components.
- Support for components across different hardware and environments.
- Stricter requirements because of its release for general use.

2.3.2.1.2 Disadvantages

- May only meet client's core requirements.
- Use involves learning curve and need for integration and further customization.
- May not meet all the client requirements because it is intended for general use.
- Can be difficult to support because source code may not be provided and propriety constraints.
- Vendors may discontinue support or cease business.
- Inconsistent GUIs between COTS products.
- Lack of object-oriented approach to development thus limiting future use and reuse.

- Lack of on-line help to integrate tools for a single purpose.

2.3.2.2 Feasibility

This solution is not feasible for implementation because of the limited capability of tailoring the product to meet and fulfill the client's needs. Even though this solution is quick and easy to implement, there is a high level of risk due to the integration of dissimilar applications and technology. Additional risks and unforeseen development scopes can occur through the requirements of GUI formats. This solution does not support an object-oriented approach to development so reusability and modularity are not supported.

2.3.3 Hybrid Software

2.3.3.1 Procedure

Software that is built with the integration of software components that are of different nature and origins is referred to as Hybrid Software. The components of Hybrid Software can include software developed by third parties, known as COTS, web services, and the development of software. The software is built considering its capacity for being integrated into a single software, to inter-operate in a transparent manner, and the resources that are required for the software's adoption and integration. Using the hybrid combination of COTS products and coding will meet the requirements of the SWC Product in the most efficient and cost effective manner. The tools of the COTS products that meet the requirements of the SWC will be integrated with coding that will be developed specifically for this product. The COTS products implemented into the SWC Product will satisfy the core processes required for the application.

2.3.3.1.1 Advantages

- Highly flexible.
- Adopts best of breed. The best product of its type.
- Meets all of the requirements of the SWC Product.
- Cost constraints of the project are met.
- Development time line and schedule are met.

2.3.3.1.2 Disadvantages

- Doesn't conform to typical standards.
- Every Hybrid Model is different.
- Proprietary code access to tools for integration may be difficult and/or expensive.

2.3.3.2 Feasibility

The use of the vast variety of professional tools and applications available through COTS will be suitable for the majority of the requirements for the SWC Product. By integrating the latest programming available and the development tools on the commercial market will make easy

implementation and support any future development. The hybridization of COTS products and customized coding will reduce the risks of cost overruns and schedule failures in the implementation and development of the Semantic Web Crawler Product.

2.3.4 Solution Decisions

2.3.4.1 Hypertext Transfer Protocol (HTTP)/Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) Crawler: Local Server vs Amazon EC2 Virtual Cloud Servers

Solution: Amazon EC2

Reason: Crawlers are synonymous with blocked Internet Protocol (IP) addresses and overwhelmed servers, by using Amazon's EC2 Cloud Service we will auto generate server instances with their own unique IP address from nothing to handle the massive amounts of data that a crawler could pull, analyze, and then turn off as the website crawls finish. The client will have the option of switching to their own personal cloud server infrastructure using the Open Source Eucalyptus platform on any server if they decide to leave Amazon down the road.

2.3.4.2 Input/Output (I/O) Operations: Command Line vs API

Solution: Command Line

Reasons: The Development Team decided to go with Command Line Input in order to better improve the crawler's overall ability to adapt to newer and interesting technologies. Though a direct API may be something to consider down the road, for speed and adaptability with other systems the Command Line still seems to be our best option.

2.3.4.3 Content Storage: Local Drive Storage vs Amazon S3 Virtual Cloud Storage

Solution: Amazon S3

Reason: In order for this system to use as little bandwidth as possible, it must be capable of storing data at different timely intervals, the default will be set to weekly but the API will allow for changes to that so that the core knows when it needs to send a copy of a web page to the parser or if it needs to send the entire page from scratch. The client will have the option of switching to their own personal cloud storage infrastructure on any server if they decide to leave Amazon down the road.

2.3.4.4 Hypertext Markup Language (HTML) Parser: Java vs. C++

Solution: C++

Reason: The Java run time engine can provide the same level of speed in some cases that C++ provides while allowing the use to be cross platform and more versatile in the language implementation, as long as certain things are implemented such as multi threading it can come

very close to the performance speeds of C++. We could even compile Java into Bytecode but have decided to use C++ simply due to the simplicity of it's implementation in the environment we will be building the crawler for, in this case it's an Ubuntu Server infrastructure. Parsing through the data quickly is a critical component and must be as fast as possible to save time which in this case is equal to money.

2.3.4.5 Database Storage: Local MySQL vs. Amazon Relational Database Service (RDS) (MySQL) Virtual Cloud RDS

Solution: Amazon RDS

Reason: Amazon RDS (MySQL) is not free, but it's a scalable and robust Relational Database Management System (RDBMS) and easy-to-use interfaces are provided for many different languages. The Development Team has experience with MySQL but chose to go with RDS because it is server independent. The current version of MySQL has overcome many of the limitations of the past and provides the required functionality. The client will have the option of switching to their own personal cloud database infrastructure using the Open Source MySQL platform on any server if they decide to leave Amazon down the road.

2.3.4.6 User Interface: Client-Server vs. Web-based

Solution: Both.

Reasons: By creating the Core to be headless, the actual architecture won't matter. The Development Team will develop a Web-based GUI but could also implement an installed client for future iterations to speed up the client's ability to optimize performance and interoperability.

2.4 Processes to be Provided

The local and remote capabilities to be provided by the SWC Product include:

- HTTP/HTTPS crawler.
- I/O Operations.
- Content Storage.
- HTML & Data Parser.
- Database Management.
- User Interface.

2.4.1 HTTP/HTTPS Crawler

A semantic web crawler functions much like a simple web browser, it will be able to download individual web pages and accept session ID's as well as other necessary minor requirements in order to properly view the page's contents.

2.4.2 I/O Operations

The client will send a command to crawl a website in which case a cloud instance will begin and start the crawl using the core software. Commands will be sent through to the parser on which lines of a downloaded page will be read and properly analysed as well as told what type of data to pull out. Once the crawl has ended and the data has been queued into the RDS database system, then the instance will be immediately turned off allowing for proper resource allocation and to save the client money on hosting and or server costs.

2.4.3 Content Storage

The first time the program downloads a website, before it enters the parser, it will be uploaded to a static storage medium with a website address, time, and date stamp. This prevents sites from getting crawled repeated times because something went wrong during the parsing process. It's essentially like copying a website locally to view it at a later time.

2.4.4 HTML Parser

The secondary part of the Core software will be the Parser itself, in this case there will be multiple parser internal pieces that will function with different forms of filtering through the data itself. All functions within the Parser will be templated and recursive so that it may be used again and again throughout the parsing process. All regular expression (regex) and or parsing comments to include pagination will be passed through as well allowing the Core software to be as nimble as necessary in order for it to perform it's function as efficiently as possible.

2.4.5 Database Management

All possible forms of collected data will be placed into a hash and sent to the third part of the Core for database integration and or implementation. A Cloud managed MySQL database through the RDS Amazon services will be utilized to design and store the database, due to the modularity of the system, this is a benefit to the client as the client may choose to port the entire system to a local machine if they no longer wish to utilize Amazon Web Service (AWS) or if the service gets discontinued at some point.

2.4.6 User Interface

The last but most important piece of the puzzle is the UI, it will really need to be able to allow the client to select certain elements of a specific page which will be crawled, the UI will then send an API command to the Core to do the actual crawling but the secret is how the API in the Core is formatted to accommodate the possible desires of the client. It'll also allow for future upgrades of the UI without the need to necessarily modify the core.

2.5 Processing Environment

Sections 2.5.1 through 2.5.7 detail the hardware and software requirements required to operate and develop the SWC Software System:

2.5.1 Hardware

2.5.1.1 Client

- Personal Computer with speed of 75 Megahertz (MHz) or higher (or equivalent).
- Internet connection 1Megabits per second (Mbps) or higher.
- Mouse or compatible pointing device and keyboard.
- Monitor - Visual Graphics Array (VGA) or higher resolution monitor (Super Visual Graphics Array (SVGA) recommended).
- 32 Megabytes (MB) of Random Access Memory (RAM) for operating system.

2.5.1.2 Server

- Linux Virtual Server Instances.
- 256 MB's of RAM or higher.
- No need for physical server hardware.

2.5.2 Software

2.5.2.1 Client

- An Operating System (OS) capable of supporting an Internet browser.
- Client side SWC application (Enhanced Version).
- Git or Subversion (SVN) Client Repository for shared development.

2.5.2.2 Server

- Ubuntu Server.
- SWC Core Software.
- Web-Based User Interface (Full Capability Version).
- Git or SVN Server Repository for shared development and historical data recovery.

2.5.2.3 Third Party Services

- EC2 - An Amazon scalable deployment of applications by providing a Web service through which a user can boot an Amazon Machine Image to create a virtual machine.
- S3 - Amazon Service that provides storage through web services interfaces.
- RDS - Amazon Service that provides users a relational database for use in their applications.

2.5.3 Work Place

The product development workplace must be compatible with the normal operating requirements of Personal Computers and provide a calm environment conducive to software development work.

2.5.3.1 Hardware

- Personal Computer with Internet connection 10Mbps or greater.
- 4 Gigabyte (GB) Ram.

2.5.3.2 Software

- An OS capable of generating an Internet connection.
- Internet browser.
- Eclipse Integrated Drive Electronics (IDE) for C++ and Java development.

2.5.3.3 Version Control

All version data will be recorded in a historical repository by means of GIT Repository. This repository utilizes the benefits of a cloud based storage system to capture all the specific versions of the project software.

2.6 Product Features

2.6.1 Prototype Version

The initial prototype will only contain the core program with the following feature abilities:

- The ability to download websites and cache them locally or to Amazon S3.
- The ability to parse the data and handle pagination dynamically through a command line set of instructions.
- The ability to save this data in a string format to a database for further processing by the end user in an RDS database.
- The ability to auto-generate instances for crawls by turning them off and on using only core giving the system a high level of scalability.

2.6.2 Full Capability Version

This subsection will determine what features provide all the desired functionality within the resource and time constraints that are determined by the client.

- Provided features will include product functionality that is directed from the client, to include example systems or sites the client may outline.
- The ability to provide data to the client at a cached level and will be updated as necessary per the client requirements.
- Web based interface for the client to interact with will allow the client to select what components of a web site they wish crawled and associate those components with the necessary commands for the Core crawler.

2.6.3 Enhanced Version

The Enhanced Version will be developed for other operating systems other than Windows. For example, the Linux, Mac, and Unix.

- It will offer support for utilization of Web services for commercial applications.
- It will be open source to give the client the opportunity to code the product to her specification.
- It will have the ability to create a client for the user to instantly generate and access instances for crawling purposes.

2.7 Acceptance Criteria

For purposes of having a finished and working product, the acceptable version of the software will do the following to complete the required mission.

- The client will be able to use a Web based service to map out the different components of a web page, the data types they want collected in an easy to understand UI.
- The client will be able to point the crawler at any number of websites which will immediately be crawled using Amazon on demand Instances using the components previously picked out using I/O Operations.
- The crawler will cache and store the sites crawled for specific lengths of time determined by the client using the UI and will first check for the cached version before going out and crawling a website using it's Content Storage.
- Data will be parsed and stored in a database to be collected by the client in any format they wish using it's HTML/Data Parser as well as it's Database Management system.
- The Core will be fast and nimble allowing it to collect websites and content quickly and in good order using it's HTTP/HTTPS Crawler.