PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ESCUELA DE POSGRADO

PROPUESTA DE PATRÓN DE DISEÑO DE SOFTWARE ORIENTADO A PREVENIR LA EXTRACCIÓN AUTOMATIZADA DE CONTENIDO WEB.

Tesis para optar el título de Magister en Informática con mención en Ingeniería de Software.

AUTOR
Edson Bryan Castañeda Rojas

ASESOR
Dr. Héctor Andrés Melgar Sasieta

JURADOS:
Mg. Claudia María del Pilar Zapata del Río.
Mg. Luis Alberto Flores García

LIMA - PERU
2016

DEDICATORIA

A Isaac Huashua

Tu coraje para hacerle frente a lo desconocido me inspira día a día, gracias por aquellas entrañables conversaciones sabatinas.

INDICE GENERAL

INDICE DE FIGURAS

INDICE DE TABLAS

Siglas y Acrónimos

API	Interfaz de Programación de Aplicaciones. *Application Program Interface*
SWF	*Small Web Format* (Formato de archivo de Adobe Flash)
W3C	*World Wide Web Consortium*
HTML	Lenguaje de Marcado de Hipertexto. *Hypertext Markup Language*
FTP	Protocolo de Transferencia de Archivos. *File Transfer Protocol*

RESUMEN

Web scraping o extracción de datos *Web* es el proceso de recolección de información de uno o más sitios *Web* de manera automatizada, emulando la interacción entre un usuario y un servidor, dicho proceso se basa en el análisis de estructuras HTML y no requiere la autorización de los propietarios.

El uso de estructuras repetitivas o plantillas, facilita el funcionamiento de un programa informático que extrae contenido *Web*, dicha intrusión genera un incremento considerable en el uso de recursos, considerando la permanente ejecución de instrucciones para obtener tanto contenido como sea posible.

Con la finalidad de reducir la vulnerabilidad de los sitios *Web* frente a procesos de extracción de contenido masivo, en el presente trabajo se planteó un patrón de diseño de software tomando como referencia el patrón *Template View* de Martin Fowler, al cual se agregó una capa de aleatorización que permita generar estructuras HTML no predecibles.

Mediante la aplicación de una herramienta de extracción de contenido a un sitio *Web* de prueba, cuya capa de presentación se desarrolló tomando en cuenta el patrón de diseño propuesto, se logró verificar una reducción considerable de la cantidad de datos extraídos.

CAPÍTULO

1

Generalidades

1.1. Problemática

En los últimos años, las herramientas de extracción de información *Web* vienen perfeccionando su proceso, al punto de poder obtener grandes cantidades de información con una inversión mínima de recursos [6].

Los administradores de sitios web intentan que la información sea consumida por personas, sin embargo, actualmente existe una completa industria de robots que obtienen información de forma masiva [19].

Muchas empresas invierten recursos en infraestructura, con la finalidad de agilizar sus procesos, pero desafortunadamente los programas que extraen contenido se favorecen igual o mejor que un usuario normal, se calcula que el 40% de las visitas que reciben las páginas web, analizadas en [6], son programas informáticos y no personas.

Algunos casos de acceso no ético a sitios web haciendo uso de programas informáticos han llegado a ser judicializados, los cuales involucran a empresas dedicadas al comercio electrónico, apuestas en línea e inclusive redes sociales [3, 5, 27, 29].

La facilidad con que un programa informático extrae contenido web está relacionada directamente con el diseño de la estructura interna de un sitio, lo que en desarrollo de software se conoce como capa de presentación. Normalmente los patrones de diseño de software proponen estructuras repetitivas o plantillas [35 p. 340] utilizadas para mostrar información, con lo cual, basta con una simple revisión de la estructura *html* de la plantilla para llevar a cabo el proceso de extracción.

El uso de estructuras repetitivas o plantillas, facilita el funcionamiento de un programa informático que extrae contenido web, dicha intrusión genera un incremento considerable en el uso de recursos [6], considerando que una persona tiene la limitante de realizar una navegación para encontrar el contenido deseado, a diferencia de un

programa que ejecuta instrucciones repetidamente para extraer tanta información como sea posible.

La información que es obtenida por el programa informático, es almacenada para su posterior utilización, usualmente se crean sitios *Web* "agregadores", los cuales muestran información extraída de diferentes páginas *Web* sin el permiso de los propietarios [36]. El uso de recursos se incrementa aún más, si los sitios *Web* "agregadores" incluyen enlaces a ficheros o contenidos multimedia alojados en los servidores de los sitios *Web* de donde se extrajo la información [6, 26].

1.2. Objetivo General

Describir un nuevo patrón de diseño de software orientado a la capa de presentación, basado en el patrón *Template View,* que reduzca la vulnerabilidad de los sitios *Web* frente a la extracción automatizada de contenido *Web*.

1.3. Objetivos Específicos

- OE.1 Adaptar el patrón de diseño de Software *Template View* añadiendo una capa de aleatoriedad que brinde la capacidad de frustrar intentos de extracción de información.
- OE.2 Implementar el algoritmo que permita generar etiquetas *html* aleatorias.
- OE.3 Implementar un sitio web que contenga un catálogo de productos cuya estructura interna considere los lineamientos del patrón de diseño propuesto.

1.4. Resultados Esperados

- RE.1 Descripción del patrón de diseño de software propuesto considerando los elementos recomendados por *Gang of Four*.
- RE.2 Diagrama de clases del patrón de diseño propuesto.
- RE.3 Diagrama de secuencia del patrón de diseño propuesto.
- RE.4 Framework generador de etiquetas *html* aleatorias orientado a capas de presentación.
- RE.5 Sitio web de prueba implementado con una capa de presentación de etiquetas *html* aleatorias.
- RE.6 Resultados del uso de herramientas software de extracción de contenido *Web* dirigidas a un sitio *Web* construido considerando el patrón de diseño de software propuesto.

1.5. Métodos y Procedimientos

La Tabla 1 agrupa los métodos y procedimientos a seguir para alcanzar los objetivos planteados.

Tabla 1: Métodos y Procedimientos

Resultado esperado	Objetivo	Método	Herramientas
RE.1	OE.1	Descripción del patrón de diseño propuesto considerando: • Nombre • Problema • Solución • Aplicabilidad • Consecuencias	Patrones de Diseño de Software *Gang of Four*
RE.2	OE.1	Generar el diagrama de clases del patrón de diseño propuesto.	UML, Patrones de Diseño de M. Fowler
RE.3	OE.1	Generar el diagrama de secuencia del patrón de diseño propuesto.	UML, Patrones de Diseño de M. Fowler
RE.4	OE.2	Implementar el algoritmo generador de etiqueta *html* aleatorias.	JavaScript
RE.5	OE.3	Implementar sitio web con una capa de presentación “aleatoria”.	HTML, CSS, JSON
RE.6	OE.3	Aplicación de herramienta software de extracción de datos (*Ver Tabla 2*) a un sitio *Web* construido con los lineamientos del patrón de diseño propuesto.	Web Scraper IO, Sitio *Web AntiScraping*

1.6. Justificación

Los patrones de diseño son planteados para solucionar problemas comunes en el desarrollo de software, proporcionan catálogos reusables para el diseño efectivo y estandarizado de las diferentes capas de un sistema informático [34, 35].

Contar con un patrón de diseño de software orientado a reducir la vulnerabilidad de aplicaciones *Web*, frente al crecimiento de procesos de extracción masiva de contenido [6], favorece a la protección de sitios *Web* en general, y mayoritariamente a aquellos cuya información es actualizada constantemente, como páginas de noticias, tiendas en línea, apuestas en línea, redes sociales, entre otras [3, 5, 27, 29].

Tener un catálogo reusable de lineamientos que reduzcan la extracción masiva de contenido, permite que administradores de páginas *Web* tengan la posibilidad de elegir la información que deseen proteger y la que sea disponible para la lectura por parte de programas externos [6].

El aporte general del presente trabajo es brindar una herramienta para el desarrollo de aplicaciones orientadas a internet, considerando la vulnerabilidad actual de las aplicaciones *Web* frente al acceso sin autorización de programas informáticos.

1.7. Delimitación

La extracción de contenido *Web* se extiende a diferentes orígenes de datos como librerías API, servicios *Web*, servidores FTP, archivos de texto, entre otros [7, 8, 15], el presente trabajo sólo considera la protección del contenido de sitios *Web* frente a intrusiones de extracción de contenido no autorizadas.

Las pruebas realizadas están dirigidas a sitios *Web* de comercio electrónico, específicamente centradas en el catálogo de productos que ofrece una tienda en línea.

Durante las pruebas de validación se identificó la posibilidad de considerar otros sitios *Web* cuya actualización de contenido sea permanente (noticias, apuestas en línea, redes sociales, entre otros), pero finalmente no fueron incluidos en el alcance.

CAPITULO

2

Marco Conceptual

En este capítulo se abordan los principales conceptos relacionados a la extracción de contenido *Web*, sus aplicaciones, las herramientas que utiliza, los desafíos que afronta y los métodos que pueden contrarrestar su aplicación, todos ellos planteados en investigaciones previas.

Los conceptos descritos en el presente capítulo permitirán contextualizar el problema, además de facilitar la lectura de las palabras clave utilizadas a lo largo del presente documento.

2.1. Extracción de datos Web.

Es el conjunto de metodologías que permiten obtener información de diferentes orígenes de datos en línea, los cuales pueden ser: sitios *Web*, servicios *Web*, librerías API, servidores FTP, archivos PDF, archivos XML, entre otros [8, 9, 15, 18].

Es el proceso automático de recolección de información de múltiples páginas web de manera sistemática, también conocido como *Web Data Scraping* es una de las técnicas de extracción de contenido *Web* más antiguas [13].

También conocida como *Web scraping* es una técnica usada para obtener información de páginas *Web* usualmente de manera automática y sin el permiso del propietario [17, 36].

La extracción de datos *Web*, se lleva a cabo habitualmente en documentos estructurados los cuales comúnmente usan lenguajes de marcado (HTML, XML, etc.) para mostrar su estructura interna [37].

El proceso de extracción de datos orientado a una página *Web* puede ser dividido en 5 pasos [31]:

- Paso 1: Interacción con el sitio *Web*.
  El programa extractor interactúa con el sitio *Web* mediante una dirección URL.
- Paso 2: Soporte para la generación y ejecución del contenedor.

El programa extractor necesita la estructura de la sección *html* como parámetro de entrada, dicha sección es conocida como contenedor o *wrapper* e incluye detalladamente la estructura de etiquetas que contiene la información a ser extraída.

La Figura 1 muestra un ejemplo de cómo puede estar estructurado un contenedor en una tienda virtual, en ese caso se cuenta con una etiqueta <*html*> la cual hace el papel de contenedor principal, seguida de una etiqueta <*div*> que contiene los datos de un producto, dentro de ella se aprecian 3 etiquetas <*img*>, <*p*> *y* <*label*> las que contienen la imagen del producto, la descripción y el precio respectivamente.

- Paso 3: Planificación
  Se refiere al proceso repetitivo de la extracción. Un programa informático puede ejecutarse permanentemente sobre un sitio *Web*, intentando extraer más información o esperando alguna actualización de contenido.
- Paso 4: Transformación
  Una vez obtenida la información, se procede al filtrado o refinamiento de los datos extraídos.
- Paso 5: Provisión
  Finalmente la información extraída es enviada a aplicaciones externas.

Figura 1: Representación de un *wrapper* o contenedor.
Fuente: Elaboración propia.

La información extraída podría ser estructurada y almacenada para su posterior uso, teniendo como principal beneficio el hecho de recolectar eficientemente datos de diferentes orígenes con un mínimo esfuerzo humano [20].

2.1.1. Desafíos de la extracción de datos web.

Los desafíos de la extracción de datos web pueden resumirse en 5 puntos [20]:

- Usualmente requiere de la ayuda de expertos.
- Debería ser capaz de procesar grandes volúmenes de información en un tiempo relativamente corto.
- Aplicaciones que manejan información personal deben proveer sólidas garantías de seguridad.
- Mejoras relacionadas al aprendizaje automático usualmente requieren de un significativo entrenamiento sobre etiquetado (indexación) manual de páginas *Web* lo cual evidentemente se encuentra propenso a errores.
- Los orígenes de datos pueden pasar continuamente por procesos de cambio estructural, lo cual es impredecible.

Otro de los desafíos de la extracción de datos *Web* actual es automatizar la creación de contenedores aplicando aprendizaje automático, considerando como principal estrategia la comparación de diferentes fuentes de información, proyectos como *SG-WRAM* y *Lixto* [20] son algunos ejemplos.

La mayor preocupación relacionada a la extracción de datos desde plataformas *Web* sociales es la privacidad. Muchos estudios, en efecto, muestran la posibilidad de revelar información privada de usuarios aprovechando que la información se encuentra públicamente disponible. Inclusive redes sociales como *Goggle+* o *Twitter* promueven el manejo masivo de información social pública facilitando el acceso a dicha información mediante librerías API [15, 20, 22, 30].

2.2. Sistema de extracción de datos *Web.*

Un sistema de extracción de datos *Web* es una plataforma que implementa una secuencia de procedimientos que extraen información de orígenes *Web*, asimismo proponen la división de dichos sistemas en 2 niveles, nivel empresarial y nivel social [20].

Las aplicaciones del nivel empresarial son:

- Publicidad de Contenido Consciente
- Atención al Cliente
- Construcción de Bases de Datos
- Ingeniería de Software
- Inteligencia de Negocios e Inteligencia Competitiva
- Proceso de Integración *Web* y Administración de Canales
- Pruebas Funcionales de Aplicaciones *Web*
- Comercio Comparativo

- Escenarios de Aplicaciones *Web* Híbridas
- Minería de Opinión y Análisis de Sensibilidad
- Bases de Datos de Citas Bibliográficas
- Accesibilidad *Web*
- Extracción de Contenido Principal
- Archivado *Web*

En la Figura 2 se observa la arquitectura típica de un sistema de extracción de información *Web* planteada en [31]. El proceso inicia cuando el usuario genera los contenedores *html,* para una lista de sitios previamente identificados, de forma automática o mediante la interfaz visual, dependiendo del conocimiento previo de cada sitio *Web*. Para la generación automática es necesario contar con datos en el repositorio de contenedores. A partir del conocimiento del contenedor *html*, se ejecuta el programa, obteniendo los datos en texto plano que luego serán filtrados y depurados para su posterior envío a las aplicaciones de destino.

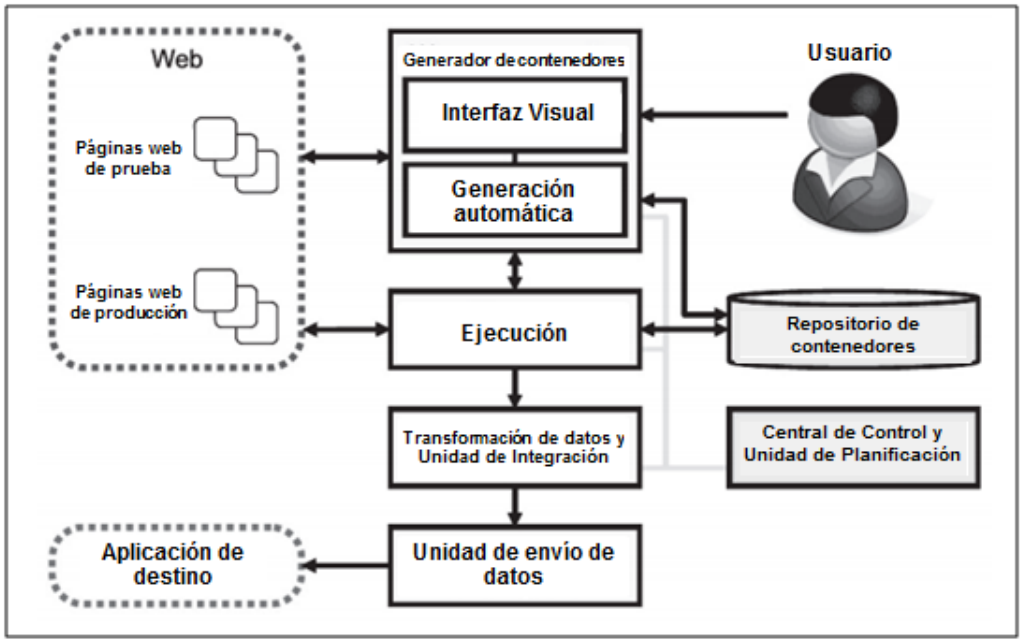

Figura 2: Arquitectura típica de un sistema de extracción de información *Web* [31].

2.2.1. Web Scraper

También conocido como *Web crawler,* es un programa que atraviesa la estructura de hipertexto de la *Web*, tomando como base una lista de documentos y recuperando recursivamente información accesible desde esa lista. Los principales motores de búsqueda emplean potentes *Web crawlers* que atraviesan internet continuamente, intentando descubrir y recuperar tantas páginas *Web* como sea posible [24].

Un *Web scraper* emula la interacción *Web* entre un usuario y un servidor *Web* pero a través de una serie de procesos automáticos [13].

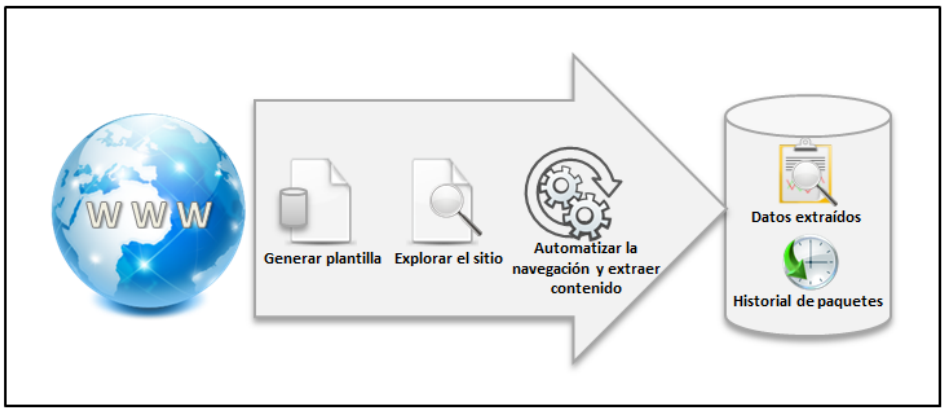

Figura 3: Funcionamiento de un *Web scraper* [39].

Tabla 2: Sistemas de extracción de información web disponibles comercialmente [15].

#	Nombre	Prod / Serv.	Descarga Inmediata	Sitio Web	Precio
1	Automation Anywhere	Producto	Si	www.automationanywhere.com	Desde $955
2	Connotate	Servicio	No	www.connotate.com	No especificado
3	Denodo	Servicio	No	www.denodo.com	No especificado
4	Djuggler	Producto	Si	www.djuggler.com	Desde $249
5	iOpus iMacros	Producto	Si	www.iopus.com	Desde $495
6	Kapow	Servicio	Si	www.kapowsoftware.com	No especificado
7	Lixio	Servicio	No	www.lixto.com	No especificado
8	Mozenda	Servicio	Si	www.mozenda.com	Desde $299/mes
9	Visual Web Riper	Producto	Si	www.visualwebripper.com	Desde $299
10	WebSundew	Producto	Si	www.websundew.com	Desde $69

2.3. HTML

Lenguaje compuesto de una serie de etiquetas o marcas que permiten definir el contenido y la apariencia de las páginas *Web*. Existen cientos de etiquetas con diferentes atributos. W3C es la organización encargada de estandarizar la mayoría de tecnologías ligadas a la *Web* [32].

2.4. Comercio Electrónico

Es el uso de internet, de la *World Wide Web* y de las aplicaciones de software móviles para hacer negocio. El término suele usarse cuando se hace referencia a las aplicaciones móviles, aunque también se usa para referirse ocasionalmente a las aplicaciones para computadoras de escritorio [33].

2.5. Interfaz de Programación de Aplicaciones

También conocido como API por sus siglas en inglés, definen un conjunto de reglas y especificaciones para la interacción de aplicaciones software, mejoran la reutilización del código, reducen el costo del desarrollo y promueven la productividad de los programadores [30].

Una gran parte de los proyectos de software actuales dependen fuertemente del uso de librerías API [38].

2.6. Métodos para prevenir la extracción de información Web.

Métodos que permiten mantener protegido un sitio *Web* de posibles ataques sistemáticos de extracción de información.

Basado en la revisión de la literatura [8, 9, 15, 18], patentes [1, 2] y foros de programación es posible organizar los métodos de prevención de extracción de información conocidos en 3 grupos: métodos de conversión de formato, métodos de manejo de etiquetas y métodos que requieren la intervención de usuarios, los cuales se detallan a continuación.

2.6.1. Métodos de conversión de formato

Dentro de este grupo se consideran aquellas técnicas que implican la conversión de cadenas de texto a un formato diferente, para lo cual consideramos formatos de imagen, SFW (flash) y texto hexadecimal.

2.6.1.1. Conversión de cadenas de texto a formato hexadecimal

La información seleccionada para ser protegida es convertida a cadenas de texto hexadecimal, de tal forma que un navegador *Web* pueda interpretarla y mostrarla al usuario de manera habitual, pero que imposibilite la comprensión de un *Web scraper*. La principal ventaja de este método es que no implica ninguna modificación en el diseño del sitio *Web* ni tampoco genera incremento considerable en el tiempo de carga, pero el principal problema es la facilidad de retornar el formato hexadecimal a texto plano.

2.6.1.2. Conversión de cadenas de texto a imagen

La información seleccionada para ser protegida, es convertida a imagen aplicando herramientas de transformación. La disponibilidad de *frameworks* de conversión en la mayoría de lenguajes de programación facilita la aplicación de éste método, asimismo la sencillez de su implementación y la dificultad que tienen los programas informáticos para vulnerar dicha seguridad lo convierten en una de las opciones más seguras, pero implica al mismo tiempo mayor rigidez en el diseño lo cual es su principal desventaja, además, el tiempo de carga de la página podría incrementarse de manera considerable.

2.6.1.3. Conversión de cadenas de texto a formato SWF

La información seleccionada para ser protegida es convertida a formato SWF usando una plantilla construida previamente. Su principal ventaja es la gran dificultad que propone a los *Web scrapers* para obtener la información que contiene pero a cambio se corre el riesgo de que no funcione correctamente en todos los navegadores *Web*, el tiempo de carga de la página podría incrementarse y la implementación implica una complejidad media.

2.6.2. Métodos de manejo de etiquetas

Dentro de este grupo de técnicas se encuentran aquellas que están relacionadas al manejo de etiquetas HTML.

2.6.2.1. Etiquetas aleatorias

Este método implica mostrar la información mediante etiquetas HTML que cambian tras cada carga de página de tal forma que el programa extractor se encuentre con una estructura diferente en cada intento de obtener información. La principal ventaja de este método es que el tiempo de carga del sitio *Web* se mantiene o tiene un incremento mínimo. La implementación puede ser considerada como desventaja ya que tiene un nivel de complejidad mediano a alto.

2.6.2.2. Celdas aleatorias

Método que se orienta a mostrar información distribuida en filas y columnas, replicando la información almacenada en una base de datos, pero considerando un orden aleatorio para cada celda. Luego de la carga inicial cada celda es ubicada según un código de ordenamiento teniendo como resultado final una tabla con información coherente para el usuario pero

desordenada para un programa extractor de contenido *Web*. Al igual que el anterior método la principal ventaja es que el tiempo de carga de la página no se altera o tiene un incremento mínimo y se considera como desventaja el hecho de tener una complejidad de implementación de nivel mediano a alto además de que sólo está orientada a esquemas de filas y columnas.

2.6.3. Métodos que requieren la intervención del usuario

2.6.3.1. Mostrar información a petición del usuario

Este método implica mantener oculta la información a filtrar hasta que el usuario la desbloquee mediante una acción en el navegador. Sus principales ventajas son: la imposibilidad de que un programa informático desbloquee la información filtrada y el bajo nivel de complejidad en su implementación, no obstante, podría generar cambios en el diseño del sitio *Web*.

En la Tabla 3. Se muestra la comparativa de métodos para prevenir la extracción de contenido *Web* considerando tres aspectos: tiempo de carga, compatibilidad con navegadores y nivel de protección.

Tabla 3: Comparación de métodos para prevenir la extracción de información *Web*.

Método / Aspectos a evaluar	Tiempo de carga del sitio web	Compatibilidad con navegadores	Nivel de protección
Conversión a formato hexadecimal	Incremento mínimo	Alta	Medio – Alto
Conversión a imagen	Incremento considerable	Alta	Muy Alto
Conversión a formato SWF	Incremento considerable	Mediana – Baja	Muy Alto
Etiquetas aleatorias	Incremento mínimo	Alta	Alto
Celdas aleatorias	Incremento mínimo	Alta	Alto
Mostrar información a petición del usuario	Incremento mínimo	Alta	Alto

2.7. Patrón de Diseño de Software.

Un patrón de diseño de software describe un problema común en la industria del desarrollo de software, detalla también la solución a ese problema la cual debe ser reutilizable. Los patrones de diseño son descripciones de comunicación entre objetos y clases que pueden adaptarse para resolver un problema de diseño general en un contexto particular [34, 35].

2.7.1. Elementos de un patrón de diseño.

Los elementos de un patrón de diseño son: el nombre, el problema, la solución y las consecuencias [34].

- Nombre. Identificador que se utiliza para describir un problema de diseño, sus soluciones y consecuencias en una o dos palabras.

- Problema. Describe cuando aplicar el patrón, explica el problema y su contexto.

- Solución. La solución describe los elementos que conforman el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño concreto o implementación particular porque un patrón es como una plantilla que puede ser aplicada en diferentes situaciones. El patrón provee una descripción abstracta de un problema de diseño y cómo una lista de elementos (clases y objetos) lo solucionan.

- Consecuencias. Son los resultados de la aplicación del patrón, incluyen el impacto en términos de flexibilidad, extensibilidad o portabilidad.

2.7.2. Clasificación de los Patrones de Diseño

Clasificación de los patrones de diseño según *Gang of Four* [34]:

- Patrones Creacionales
  Patrones de diseño orientados a solucionar problemas de creación de instancias. Ayudan a generar un sistema independiente de cómo sus objetos son creados, compuestos y representados.

- Patrones Estructurales
  Patrones que muestran interés por cómo son compuestos los objetos y las clases para formar estructuras más grandes. Los patrones estructurales usan herencia para componer interfaces o implementaciones con la finalidad de solucionar problemas de composición (agregación) de clases y objetos.

- Patrones de Comportamiento
  Los patrones de comportamiento son aquellos interesados en algoritmos y asignación de responsabilidades entre objetos. Describen patrones de objetos o clases además de los patrones de comunicación entre ellos.

La lista completa de patrones *Gang of Four* se muestra en la Tabla 4.

Tabla 4: Patrones de Diseño *Gang Of Four* [34].

Grupo	Patrones
Patrones Creacionales	- *Abstract Factory* - *Builder* - *Factory Method* - *Prototype* - *Singleton*
Patrones Estructurales	- *Adapter* - *Bridge* - *Composite* - *Decorator* - *Facade* - *Flyweight* - *Proxy*
Patrones de Comportamiento	- *Chain of Reposibility* - *Command* - *Interpreter* - *Iterator* - *Mediator* - *Memento* - *Observer* - *State* - *Strategy* - *Template Method* - *Visitor*

Por otro lado, Martin Fowler et. Al. proponen una lista de patrones de diseño considerando un agrupamiento diferente y tomando en cuenta la necesidad de contar con patrones orientados a las interfaces gráficas de usuario [35] *Ver Tabla 5*.

Tabla 5: Patrones de Diseño de M. Fowler [35]

Grupo	Patrones
Lógica de Dominio	- *Transaction Script* - *Domain Model* - *Table Module* - *Service Layer*
Arquitectura de Fuentes de Datos	- *Table Data Gateway* - *Row Data Gateway* - *Active Record* - *Data Mapper*
Objeto-Relacional de Comportamiento	- *Unit of Work* - *Identity Map* - *Lazy Load*
Objeto-Relacional de Estructura	- *Identity Field* - *Foreign Key Mapping* - *Association Table Mapping* - *Dependent Mapping* - *Embedded Value* - *Serialized LOB* - *Single Table Inheritance* - *Class Table Inheritance* - *Concrete Table Inheritance* - *Inheritance Mappers*
Objeto-Relacional de *Metadata*	- *Metadata Mapping* - *Query Object* - *Repository*
Presentación Web	- *Model View Controller* - *Page Controller* - *Front Controller* - *Template View* - *Transform View* - *Two Step View* - *Application Controller*

2.7.3. Patrón *Template View*

Es un patrón definido en [35] incluido en el grupo de Patrones de Presentación Web, cuya finalidad es agregar marcadores dentro de páginas HTML estáticas al momento de ser creadas. Cuando la página es invocada, los marcadores son reemplazados por el resultado de un cálculo (por ejemplo una consulta a base de datos).

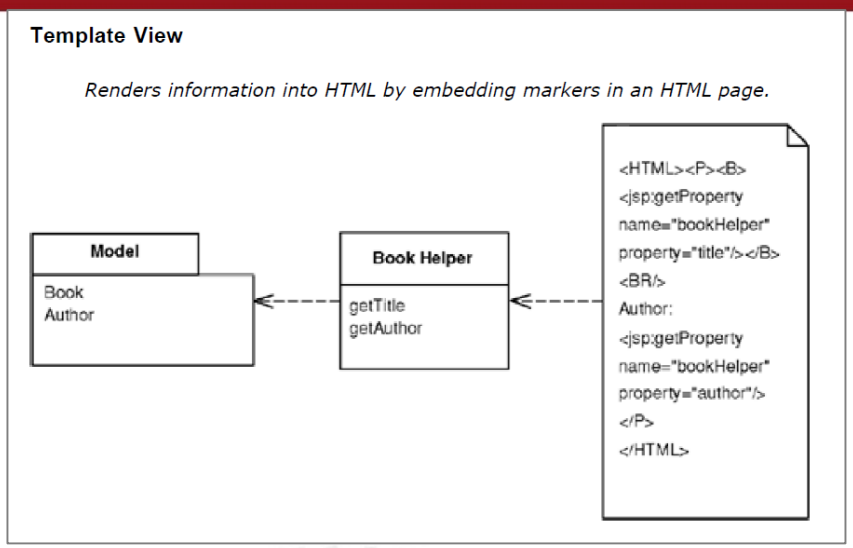

Figura 4: Diagrama del Patrón de Diseño *Template View* [35].

2.7.4. Patrón *Transform View*

Al igual que el patrón Template View, se encuentra en el grupo de Patrones de Presentación *Web* incluido en [35]

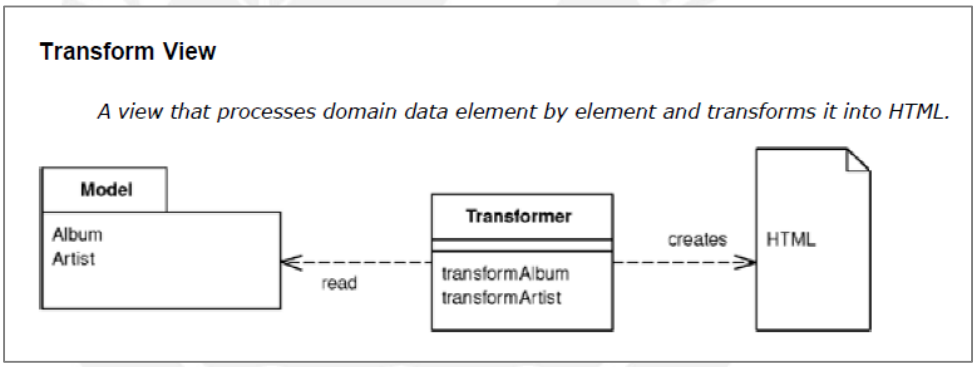

Figura 5: Diagrama del Patrón de Diseño *Transform View* [35].

El objetivo del Patrón *Transform View* es escribir un programa que examine los datos orientados al dominio y los convierta a HTML. La principal diferencia con el Patrón *Template View* es la forma en que la vista (*view*) está organizada, mientras que Template View está organizada alrededor del resultado (*output*), *Transform View* está organizada sobre transformaciones separadas para cada tipo de elemento de entrada (*input*).

3

Revisión de la Literatura

En el presente capítulo se describen los procesos de obtención de información sobre *Web scraping*, las iteraciones definidas, los resultados que se lograron a partir de la revisión de las investigaciones seleccionadas y finalmente las conclusiones que permiten visualizar un panorama actual sobre la extracción de datos *Web*.

La búsqueda de información fue separada en 2 partes, la primera comprende investigaciones en las que se promueve la utilización de técnicas de extracción de datos o se plantean mejoras a las que ya existen, y la segunda, que agrupa los trabajos que proponen técnicas de prevención.

Fueron considerados como fuentes de datos los indexadores *Scopus*, *Science Direct* y *ACM*.

3.1. Fases de la Revisión de la Literatura

Para la presente revisión se consideraron 3 fases o iteraciones. La primera fase permitió filtrar artículos referidos a la medicina siendo determinante la palabra “scraping” por generar una gran cantidad de resultados irrelevantes para la investigación. La segunda fase consistió en contrastar los resultados de las 3 fuentes de datos seleccionadas y obviar las investigaciones repetidas. Finalmente la tercera fase permitió excluir artículos sobre temas no considerados en el alcance del proyecto como extracción de contenido en dispositivos móviles o aplicaciones de *screen scraping*[1].

3.2. Aplicaciones de *Web scraping*.

Con la finalidad de encontrar las aplicaciones de *Web scraping*, se generaron los parámetros de búsqueda mencionados en la Tabla 6.

Dichos parámetros generaron los resultados mostrados en la Tabla 7, donde se describe la variación según cada fase definida.

[1] Se refiere a la práctica de recopilación de datos teniendo como origen una captura de pantalla.

Tabla 6: Parámetros de búsqueda – Objetivo 1.

Objetivo 1	Encontrar aplicaciones de *web scraping* (métodos, técnicas y herramientas)
Parámetros	(TITLE ("scraping") OR TITLE ("web scraping") OR TITLE ("web harvesting") OR TITLE ("web data extraction") OR TITLE ("data scraping"))

Tabla 7: Resultados de la búsqueda – Objetivo 1.

Objetivo 1	Encontrar aplicaciones de *Web scraping* (métodos, técnicas y herramientas)				
Resultados	Fuentes de datos \ Fases	0	1	2	3
	Scopus	737	94	82	81
	Science Direct	425	4	4	2
	ACM	61	54	32	30

Con la finalidad de conocer las principales tendencias en el uso y la mejora del *Web scraping* se definieron palabras clave para clasificar las 113 investigaciones previamente seleccionadas, siendo *wrapping* y *tool* los tópicos que se mencionan en un mayor número de casos. La Tabla 8 contiene la lista completa de palabras clave encontradas.

Tabla 8: Principales palabras clave relacionadas a las investigaciones sobre extracción de datos *Web*.

Palabras clave	Número de artículos	%
Wrapping	27	24%
Tool	25	22%
Applied to:	16	14%
Tree-based	16	14%
Deep web	11	10%
HTML to XML	10	9%

Browser-oriented	6	5%
Ontology	6	5%
Semantic	5	4%
Standard use	4	4%
Data Mining	3	3%
Regular Expressions	3	3%
Legal	1	1%

3.3. Métodos y técnicas para prevenir la extracción de datos web.

Con la finalidad de encontrar métodos y técnicas de prevención, se generaron los parámetros de búsqueda mencionados en la Tabla 9.

Tabla 9: Parámetros de búsqueda – Objetivo 2.

Objetivo 1	Encontrar métodos y técnicas para prevenir la extracción de datos web.
Parámetros	((TITLE ("scraping") AND TITLE ("anti")) OR (TITLE ("scraping") AND TITLE ("prevent")) OR (TITLE ("web scraping") AND TITLE ("anti")) OR (TITLE ("web scraping") AND TITLE ("prevent")) OR (TITLE ("web harvesting") AND TITLE ("anti")) OR (TITLE ("web harvesting") AND TITLE ("prevent")) OR (TITLE ("web data extraction") AND TITLE ("anti")) OR (TITLE ("web data extraction") AND TITLE ("prevent")))

La búsqueda anterior no permitió obtener resultados en primera instancia. Analizando algunas recomendaciones a enlaces externos se pudo ubicar un artículo en IEEE [36]. La falta de referencias en este aspecto hizo que se realice una búsqueda adicional en repositorios de patentes, lo cual permitió conocer 2 trabajos más [1, 2].

El filtrado por IP, análisis de tráfico, conversiones de formato y la aleatorización de etiquetas de marcado o *markup randomizer* (*Fig. 6 y 7*) son algunos de los métodos que describen en [36].

```
1  do until todos los id/class sean modificados
2      Obtener un id/class desde la base de datos
3      Crear un nombre aleatorio
4      Reemplazar en la base de datos y plantilla
5  end do
```

Figura 6: Seudocódigo del método *Markup Randomizer* [36].

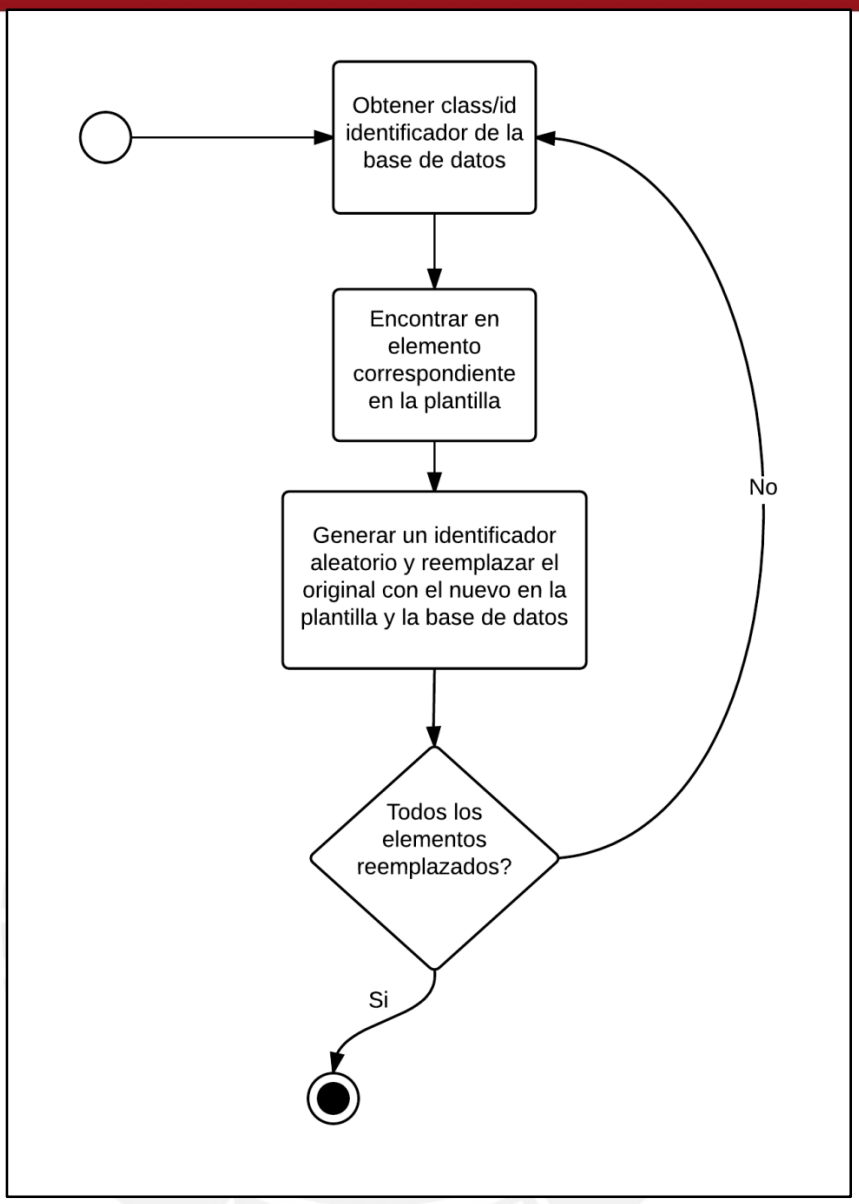

Figura 7: Diagrama de Flujo del método *Markup Randomizer* [36].

Es posible frustrar ataques de extracción de contenido *Web* mediante la implementación de un algoritmo que permita generar filas y columnas HTML en orden aleatorio, de tal forma que un programa externo tenga acceso a información ilegible mientras que la ordenación “post-carga” en el navegador permita la lectura fiable por parte de un usuario normal [1, 2]. El proceso completo puede ser visualizado en la Figura 8.

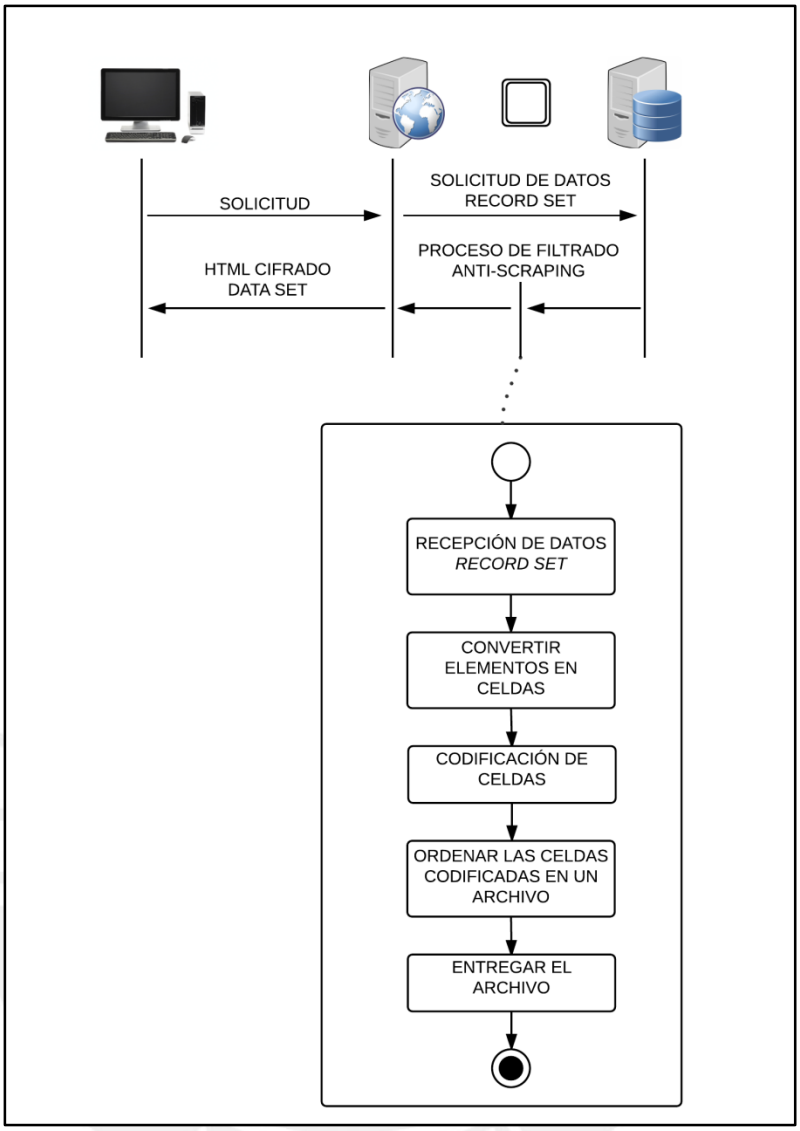

Figura 8: Filtrado *Anti-Scraping* por Wetterström & Andersson [1].

3.4. Conclusiones

En la actualidad la tendencia de las investigaciones en el ámbito de la extracción de datos *Web* demuestra un esfuerzo mayoritario por perfeccionar las técnicas existentes a diferencia de los trabajos relacionados a la prevención los cuales aún no tienen una relevancia considerable, dicho sea de paso, el perfeccionamiento de métodos de extracción de contenido *Web* está relacionado a estudios en los cuales se requiere analizar grandes cantidades de datos, y en su gran mayoría, con fines que generen un aporte a la sociedad, sin embargo ello no exime la utilización de dichas mejoras para otros fines.

El gran avance de las herramientas de extracción de datos *Web*, propone un desafío muy alto a las propuestas que intenten contrarrestarlas, cualquier solución planteada deberá ser actualizada y mejorada ya que en poco tiempo podría quedar obsoleta.

El proceso de *wrapping* o generación de contenedores es uno de los temas que viene tomando relevancia en investigaciones sobre extracción de contenido *Web*, los esfuerzos por lograr una generación eficiente y automática de contenedores han sido tema de discusión en la mayoría de artículos analizados. En la Figura 9 se muestra la distribución de los principales tópicos y la cantidad de investigaciones en las cuales se mencionan. Los artículos que proponen herramientas de extracción y las aplicaciones de *web scraping* en áreas específicas son otros de los tópicos más populares, siendo éste último detallado en la Tabla 10.

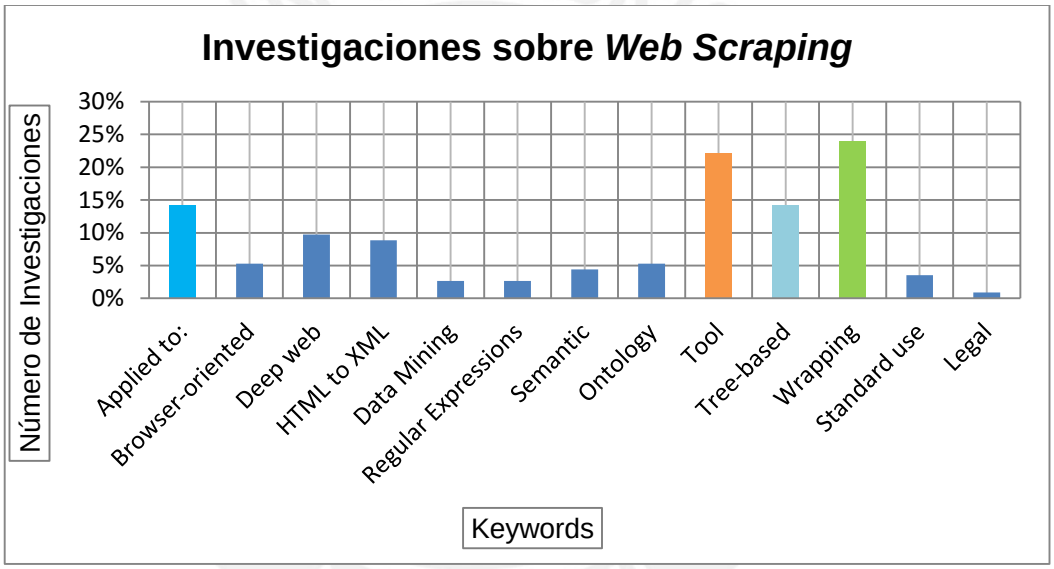

Figura 9: Principales tópicos relacionados a las investigaciones sobre extracción de datos web.

Tabla 10: Aplicación de *web scraping* en áreas específicas.

Áreas de Aplicación	Nº Artículos
Aprendizaje Online	1
Finanzas	1
Literatura Académica	2
Marketing Online	3
Compras Online	3
Turismo	2
Clima	1
Eventos Deportivos	1
Foros, Blogs y Noticias	1

De las investigaciones que proponen herramientas de prevención frente a ataques de extracción de contenido [1, 2, 36], se puede concluir que todas concuerdan en minimizar el impacto que la técnica de prevención pueda tener en términos de rendimiento, intentando que la experiencia del usuario sea la misma antes y después de su aplicación.

En [1, 2] se orientan específicamente a soluciones programáticas, mencionando algunas recomendaciones incluidas también en [36], como las soluciones de filtrado IP, análisis de tráfico, firewalls, entre otras. El concepto de aleatoriedad incluido en [2] reduce su aplicación a estructuras de filas y columnas, mientras que en [36], se mejora dicho concepto permitiendo que el campo de acción de la solución planteada sea mucho más amplio.

CAPÍTULO

4

Extracción de Datos Web

En esta sección se muestra en detalle el flujo de extracción de contenido *Web* mediante un ejemplo básico, además se describen las alternativas de prevención desde el punto de vista del programa que extrae el contenido.

4.1. Flujo de extracción de contenido web.

Los pasos descritos a continuación están basados en el flujo representado en la Figura 4.

4.1.1. Paso 1 – Analizar dirección URL.

El primer paso es acceder al sitio web mediante una dirección URL.

URL de prueba: http://soft.pe/antiscraping/index.php

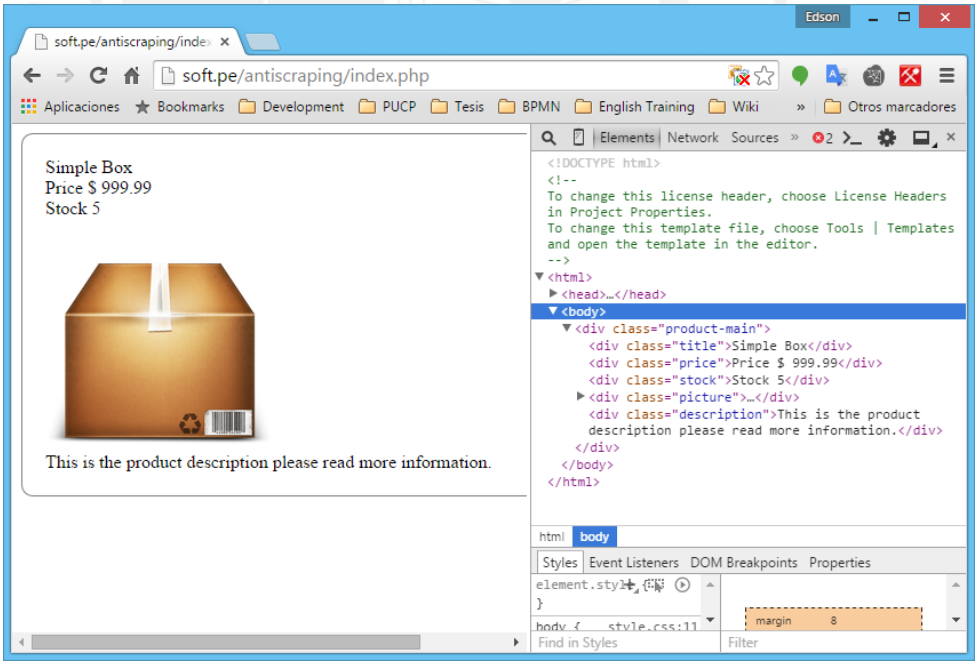

Figura 10: Ejemplo de contenido web a extraer.

4.1.2. Paso 2 – Obtener contenido HTML

El segundo paso consiste en obtener la estructura HTML correspondiente a la dirección URL especificada.

```
<html>
<head>
    <link rel="stylesheet" href="style.css"/>
</head>
<body>
<div class="product-main">
    <div class="title">Simple Box</div>
    <div class="price">Price $ 999.99</div>
    <div class="stock">Stock 5</div>
    <div class="picture">
        <img src="pictures/1401289161_box.png" />
    </div>
    <div class="description">
    This is the product description please read more information.
    </div>
</div>
</body>
</html>
```

Figura 11: Contenido web en formato HTML.

4.1.3. Paso 3 – Extraer datos.

El tercer paso es realizar la extracción de datos la cual puede llevarse a cabo mediante el acceso por etiquetas y/o por identificadores.

4.1.3.1. Acceso por etiquetas

- Establecer contexto de extracción (*product-main*)
- Obtener contenedor de datos.
- Obtener datos en un arreglo (*data[]*)
- Separar valores (*data[0], data[1], data[2],…*)

4.1.3.2. Acceso por identificadores

- Establecer contexto de extracción (*product-main*)
- Obtener datos directamente mediante el identificador (*title, price, stock,…*).

4.1.4. Paso 4 – Formatear datos

Como cuarto paso es posible considerar escenarios en los cuales sea necesario dar formato al texto obtenido.

4.1.5. Paso 5 – Almacenar datos

Finalmente los datos obtenidos son almacenados para su posterior uso.

El patrón de diseño propuesto en la presente investigación, se basa en las alternativas de prevención mencionadas a continuación.

4.2. Alternativas de prevención.

La presente investigación no comprende bloqueo de acceso al contenido *Web* por lo tanto no se discutirá el análisis de URL ni la obtención del contenido HTML (Pasos 1 y 2), la solución estará completamente dirigida al Paso 3 del flujo de extracción descrito.

El acceso por etiquetas supone el hecho de que el programa extractor de contenido puede acceder a la información sin considerar el identificador de cada etiqueta (`id` o `class`), simplemente obtiene las etiquetas que contienen los datos a buscar y almacena sus valores en un arreglo de cadenas de texto.

El ejemplo muestra el uso de etiquetas `<div>` para mostrar la información, lo cual no significa que una plantilla tenga necesariamente etiquetas repetidas, es posible combinarlas y obtener un resultado idéntico, de esta forma en lugar de usar un arreglo directamente se deberían obtener los datos en variables separadas teniendo como única llave de acceso las etiquetas (`<div>`, `<span>`, `<p>`, `<h1>`, `<h2>`, etc).

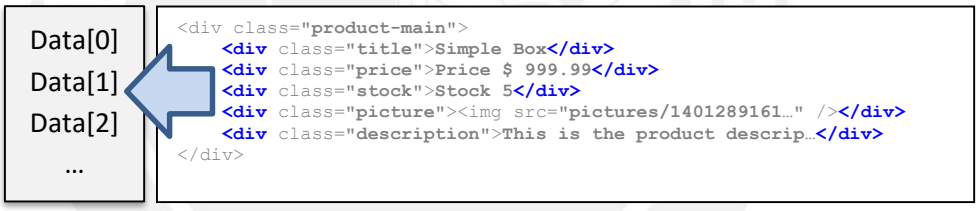

Figura 12: Uso de una etiqueta repetida.

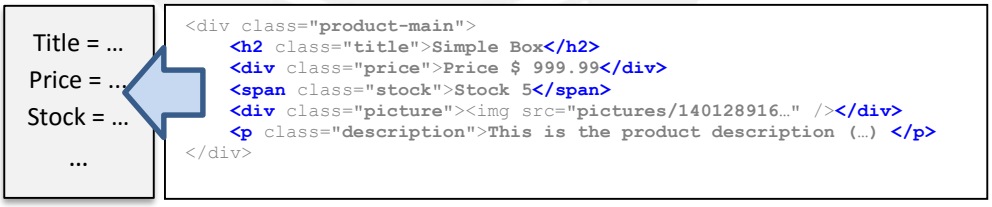

Figura 13: Uso de etiquetas diferentes.

Una solución coherente para este tipo de acceso es mostrar diferentes etiquetas HTML cada vez que cargue la página. A continuación se muestra un ejemplo de la solución. Asumiendo que en la primera carga el programa extractor sepa la estructura correcta para obtener la información:

- Carga 1

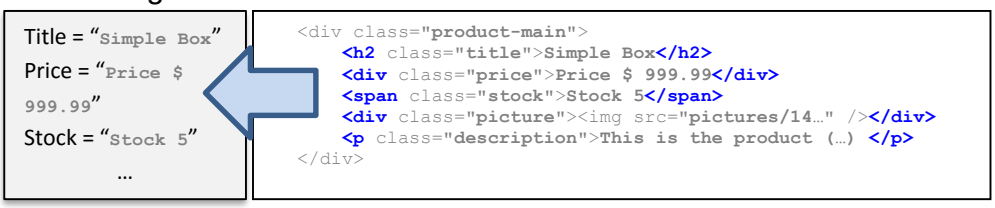

Figura 14: Acceso por etiquetas – Carga 1.

- Carga 2

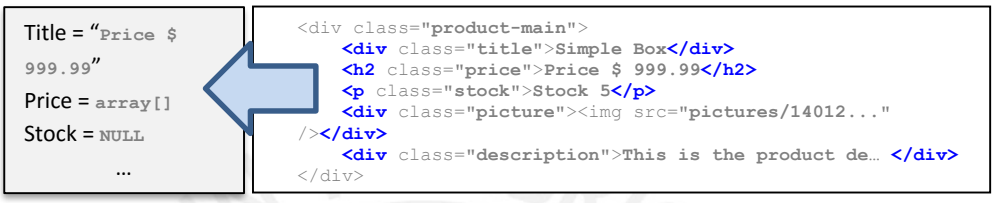

Figura 15: Acceso por etiquetas – Carga 2.

…

- Carga N

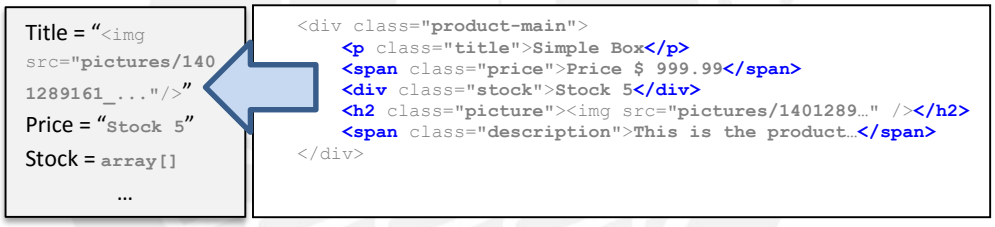

Figura 16: Acceso por etiquetas – Carga N.

La alternativa 2 implica la obtención de datos teniendo como llave de acceso los identificadores (`id`, `class`). Cabe resaltar que las etiquetas HTML para este caso se utilizan de forma referencial (Se puede acceder al valor usando el identificador y la etiqueta o sólo el identificador).

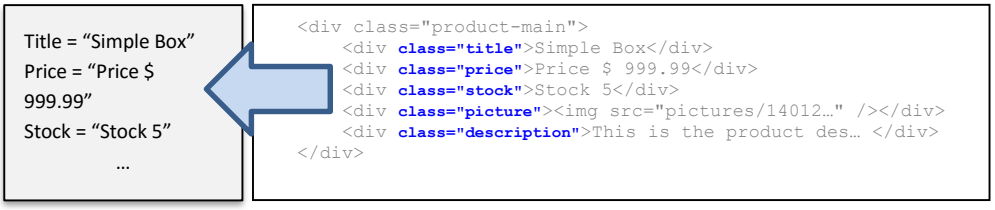

Figura 17: Acceso por identificadores.

Considerando que, el *scraper* busca un identificador específico para cada valor, la solución más coherente es modificar los identificadores en cada carga de página, como se muestra a continuación:

- Carga 1

Figura 18: Acceso por identificadores – Carga 1.

…

- Carga N

Figura 19: Acceso por identificadores – Carga N.

El patrón de diseño propuesto plantea una solución combinando las 2 alternativas mencionadas en este capítulo.

CAPITULO

5

Patrón de Diseño de Software *AntiScraping View*

En el presente capítulo se describe el patrón de diseño propuesto, el cual tiene la finalidad de mostrar lineamientos para la creación de aplicaciones *Web* las cuales puedan evitar ataques de extracción de contenido. La estructura utilizada para la descripción del patrón de diseño es la recomendada por Gang of Four [34], donde se incluyen el nombre, el problema, la solución, su aplicabilidad y las consecuencias de su aplicación.

5.1. Nombre

Antiscraping View

5.2. Problema

Las páginas *Web* dinámicas toman la información desde una base de datos mediante consultas e incrustan dicha información en la estructura HTML. Desde el punto de vista del usuario final la página *Web* puede parecer diferente tras cada petición pero desde la perspectiva de los desarrolladores es posible encontrar similitudes debido a que la gran mayoría aplica la técnica de *Template View* o diseño de vistas usando plantillas. La reutilización de la plantilla combinada con los registros obtenidos de la base de datos facilita el análisis de contenido *Web* por parte de programas externos.

5.3. Solución

Manejar la creación de la estructura HTML mediante un *Helper* que combine la generación de etiquetas HTML y sus respectivos atributos identificadores de forma aleatoria para cada registro obtenido de la base de datos, este proceso permitirá que la página muestre diferentes estructuras tras cada petición con la finalidad de que el análisis de contenido *Web* automatizado por parte de programas externos tenga una complejidad mucho más alta. Esta solución está basada en el Patrón denominado *Template View* descrito en [35].

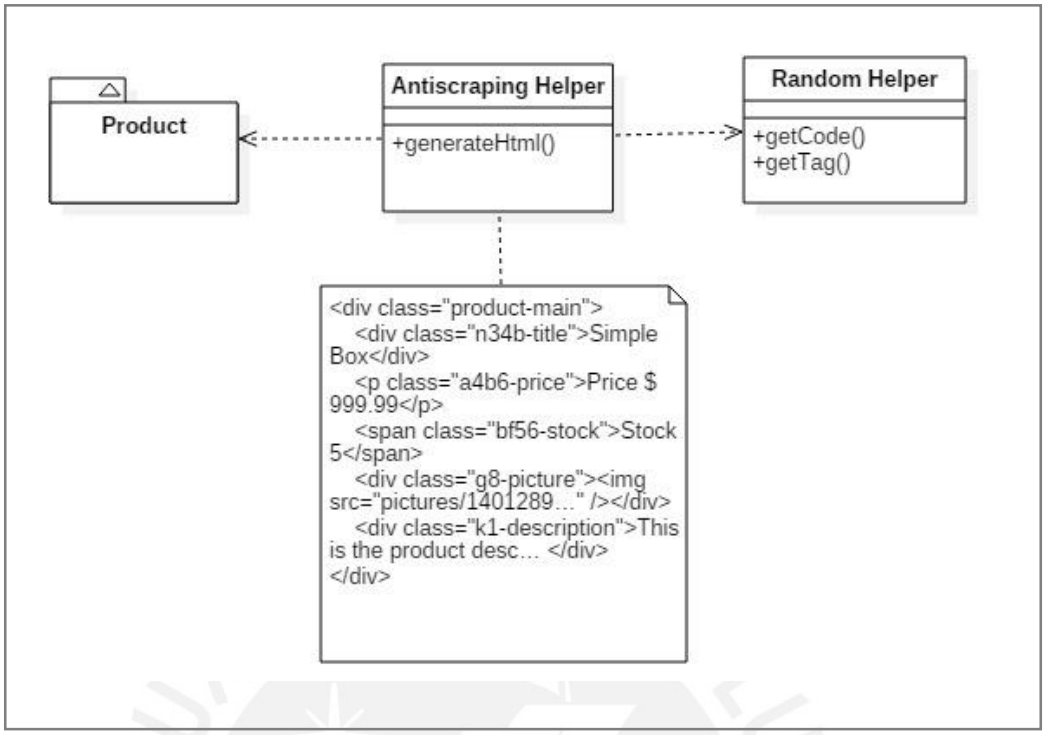

Figura 20: Diagrama del Patrón de Diseño Antiscraping View.

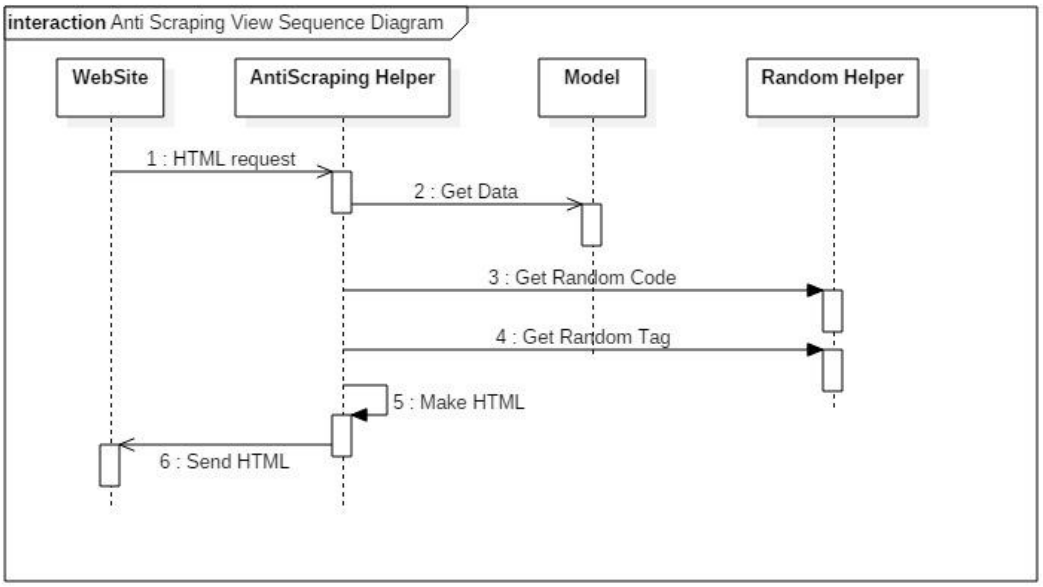

Figura 21: Diagrama de Secuencia del Patrón de Diseño Antiscraping View.

5.4. Aplicabilidad

Orientado a páginas *Web* dinámicas que manejen el diseño de vistas utilizando plantillas.

Los ejemplos de sitios *Web* que distribuyen su información mediante plantillas son diversos, entre ellos se encuentran catálogos de productos

de una tienda electrónica, lista de contactos de una libreta de direcciones, lista de resultados de una consulta en un buscador, etc, en conclusión cualquier funcionalidad que incluya listar registros luego de consultar una base de datos.

5.5. Consecuencias

- La utilización de identificadores únicos obliga a que las declaraciones de estilo se repitan por cada uno de los registros obtenidos y posteriormente convertidos a HTML.

- Imposibilita la reutilización de hojas de estilo CSS en un gran porcentaje del sitio *Web*.

- Puede complicar el mantenimiento de la aplicación para personas con el perfil de no-programadores.

- No se cuenta con una visión completa de la estructura del sitio *Web* antes de que se realice una petición en el navegador, ya que gran parte de la estructura será creada luego de dicha petición.

6

Resultados

En el presente capítulo se muestran los resultados obtenidos a partir de la ejecución de herramientas de extracción de contenido a un sitio *Web* antes y después de la implementación del patrón de diseño propuesto, realizando una comparativa que permita demostrar la eficacia del patrón frente a ataques de extracción de contenido.

6.1. Caso de Prueba

Con la finalidad de verificar la eficacia del patrón de diseño *AntiScraping View*, se implementó una aplicación *Web* siguiendo los lineamientos de dicho patrón. En esta sección se describirá el caso de prueba utilizando la herramienta *Web Scraper IO*[2]. Para efectos comparativos, se realizaron pruebas contrastando los resultados de la aplicación de la herramienta de extracción dirigida a una página *Web* tradicional y a una página web "*anti-scraping"*.

6.1.1. Web Tradicional

La aplicación de la herramienta de extracción de información fue dirigida a una página web tradicional, es decir utilizando plantillas para mostrar registros de una base de datos. La Figura 22 muestra un ejemplo básico de una plantilla HTML vulnerable.

```
<div class="product-main">
    <div class="title">{Item Title}</div>
    <div class="price">{Item Price}</div>
    <div class="stock">{Item Stock}</div>
    <div class="picture">
        {Item Picture}
    </div>
    <div class="description">
       {Item Description}
    </div>
</div>
```

Figura 22: Estructura HTML vulnerable.

[2] Web Scraper. A Browser-Oriented Tool for Web Scraping. http://webscraper.io/

La Figura 23 muestra los resultados del proceso de extracción de datos dirigida a una reconocida tienda virtual. El *Anexo 1*, muestra más pruebas de extracción de información sobre reconocidas tiendas virtuales.

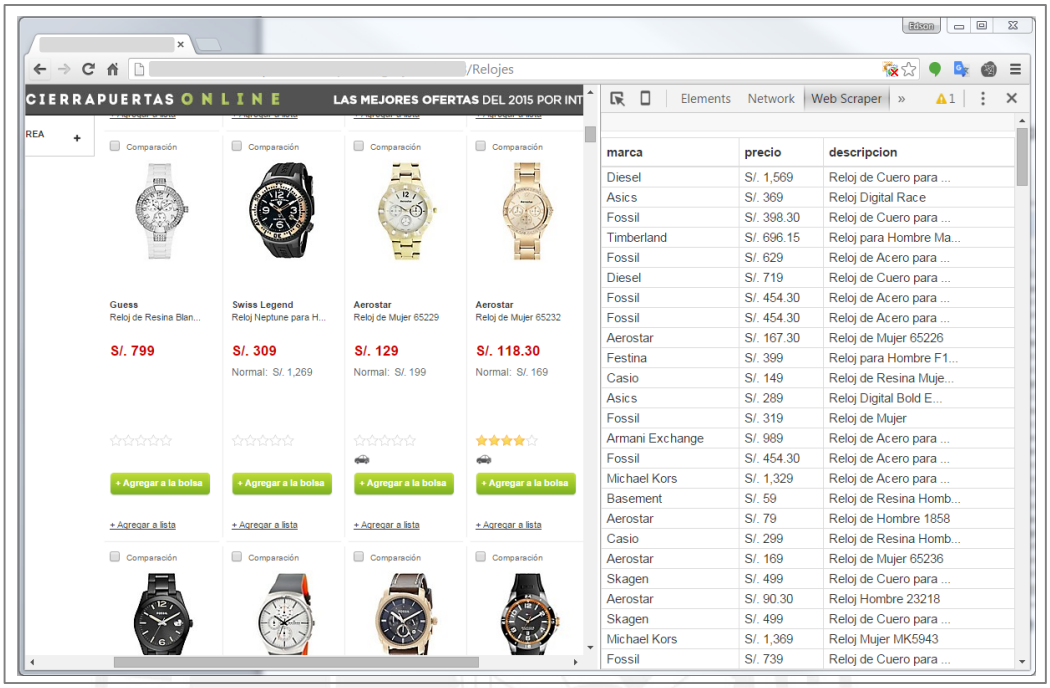

Figura 23: Aplicación de una herramienta de extracción de datos web dirigida a una tienda virtual.

6.1.2. Web Anti-Scraping

La herramienta de extracción fue aplicada a un sitio *Web* similar al mostrado en la Figura 23, pero creado siguiendo los lineamientos del patrón de diseño *AntiScraping View*. La estructura de ejemplo que corresponde al contenedor de un producto se puede apreciar en la Figura 24, donde `{random-tag[i]}` son las etiquetas HTML y `{random-id[i]}` los identificadores relacionados a las etiquetas, ambos generados aleatoriamente.

```
<{random-tag[0]} class="product-main-{random-id[0]}">
    <{random-tag[1]} class="title-{random-id[1]}">
        Simple Box
    </{random-tag[1]}>

    <{random-tag[2]} class="price-{random-id[2]}">
        Price $ 999.99
    </{random-tag[2]}>

    <{random-tag[3]} class="stock-{random-id[3]}">
        Stock 5
```

```
    </{random-tag[3]}>

    <{random-tag[4]} class="picture-{random-id[4]}">
       <img src="pictures/1401289161…" />
    </{random-tag[4]}>

    <{random-tag[5]} class="description-{random-id[5]}">
         This is the product description
    </{random-tag[5]}>

</{random-tag[0]}>
```

Figura 24: Estructura HTML anti-scraping.

La Figura 25 muestra los resultados del intento de extracción de información dirigida a una página *Web* construida siguiendo los lineamientos del patrón de diseño *AntiScraping View*, a diferencia del anterior intento, mostrado en la Figura 23, en este caso la herramienta no logra extraer los datos de la página *Web*.

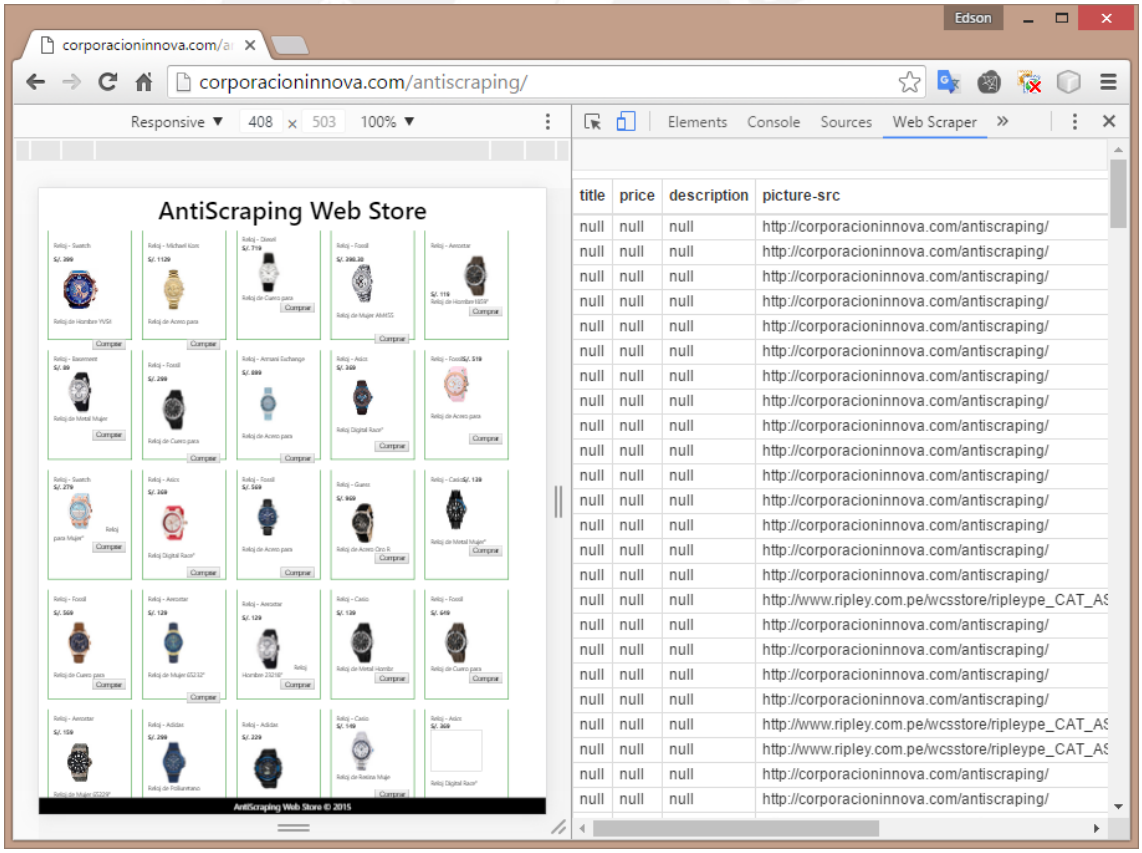

Figura 25: Aplicación de una herramienta de extracción de datos web dirigida a una tienda virtual *Anti-Scraping*.

CAPÍTULO

7

Conclusiones

En la presente tesis se realizó una revisión bibliográfica sobre la protección frente a intentos de extracción de contenido *Web*, los enfoques actuales de la utilización de herramientas extractivas y patrones de diseño de software orientados a estandarizar la construcción de aplicaciones *Web*, todo ello con la finalidad de describir un nuevo patrón de diseño de software con lineamientos que prevengan la extracción de contenido *Web*.

La revisión de los artículos de investigación permitió conocer las distintas formas con las que se nombra a la extracción de datos *Web*, siendo *Web scraping*, *Web harvesting* y *Web data extraction* las principales.

La revisión de la literatura permitió tener una visión general sobre extracción de datos *Web*, la utilización de recursos ajenos mediante los “agregadores *Web”*, el alto porcentaje de consumo de información por parte de programas informáticos en lugar de personas y la utilización de datos sin el consentimiento de los propietarios sustentan la problemática existente.

La tendencia de las investigaciones que fomentan la utilización de técnicas de extracción de datos *Web* es perfeccionar la generación de contenedores el cual es el punto inicial del proceso de extracción.

El campo de aplicación de las técnicas de extracción de contenido es amplio, considerando el aprendizaje en línea, finanzas, literatura académica, marketing en línea, comercio electrónico, turismo, clima, eventos deportivos, foros, blogs y noticias, obtenidos mediante la revisión de la literatura.

Los trabajos revisados, sobre técnicas de prevención, coinciden en añadir una capa de aleatoriedad a la tradicional manera de mostrar los registros de una base de datos.

La revisión de la literatura permitió conocer técnicas de prevención alternativas como el filtrado IP, creación de *firewalls*, análisis del tráfico, entre otras que pueden ser aplicadas en escenarios específicos por ser consideradas "bloqueantes" y particularmente para sitios *Web* de comercio electrónico

podrían resultar contraproducentes por la necesidad de que el contenido sea indexado por buscadores como *google*, *bing* o *yahoo*.

La revisión de los patrones de diseño *Gang of Four* y Martin Fowler, específicamente los orientados a capas de presentación *Web*, permitió describir de forma precisa el patrón *AntiScraping View* considerando la estructura recomendada por los autores mencionados.

Se pudo comprobar que las herramientas de extracción orientadas a navegadores son muy fáciles de utilizar, son capaces de extraer grandes cantidades de información y exportarlas a diferentes formatos. La herramienta utilizada en los casos de prueba no tuvo problemas para extraer datos de reconocidas tiendas virtuales peruanas.

Se logró comprobar la eficacia del patrón de diseño propuesto poniendo a prueba un sitio *Web* construido bajo los lineamientos del patrón de diseño *Anti-Scraping View*.

Los avances en legislación aún no permiten contar con una normativa clara con respecto a la extracción de grandes cantidades de datos usando herramientas software.

El planteamiento de técnicas preventivas debe enfocarse en la generación de contenedores por ser éste el principal punto al cual apuntan las mejoras en la extracción de contenido *Web*.

Se debe considerar que todo método de prevención implica un incremento en los requerimientos, las aplicaciones *Web* de gran alcance deben considerar un análisis de infraestructura con la finalidad de no afectar la experiencia del usuario.

La aplicación de un método de prevención implica tener en cuenta la interacción del usuario con la aplicación *Web*, de tal forma que la experiencia no difiera antes y después de la implementación del método elegido.

BIBLIOGRAFIA

[1] R. Wetterström and S. Andersson, “Web information scraping protection,” U.S. Patent 20110185434A1, July 28, 2011.

[2] R. Kraft, J. Myllymaki, and J. Ruvolo, “System and method for preventing automated crawler access to web-based data sources using a dynamic data transcoding scheme,” U.S. Patent 6938170B1, Aug. 30, 2005.

[3] U.S. Court for The Northern District of California, “eBay Inc. v. Bidder's Edge Inc.,” [Online] Available: http://pub.bna.com/lw/21200.htm, Accessed on: May 6, 2015.

[4] Google Support, “Contenido duplicado,” [Online] Available: https://support.google.com/webmasters/answer/66359?hl=es, Accessed on: April 20, 2015.

[5] D. Gross, “Feds accuse Jerk.com of scraping Facebook, cheating users - CNN.com,” [Online] Available: http://edition.cnn.com/2014/04/07/tech/web/jerk-ftc-charges/, Accessed on: April 20, 2015.

[6] M. Ward, “Conozca a los *scrapers*, los ladrones de páginas de internet,” [Online] Available: http://www.bbc.co.uk/mundo/noticias/2013/10/131002_tecnologia_screen_scrapers, Accessed on: April 19, 2015.

[7] Y. Yang, L. Wilson, J. Wang, “Development of an automated climatic data scraping, filtering and display system,” *Computers and Electronics in Agriculture*, vol. 71, pp. 77-87, 2009.

[8] S. Robinson, T. Robinson, and R. Burson, “Trained predictive services to interdict undesired website accesses,” U.S. Patent 20110131652A1, Jun. 2, 2011.

[9] A. Razzaq, Z. Anwar, H. Ahmad, K. Latif, and F. Munir, “Ontology for attack detection: An intelligent approach to web application security,” *Computers & Security*, vol. 45, pp. 124-146, 2014.

[10] D. Huynh, S. Mazzocchi and D. R. Karger, “Piggy Bank: Experience the Semantic Web inside your web browser”, *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, pp. 16-27, 2007.

[11] A. Ankolekar, M. Krozsch, T. Tran, and D. Vrandecic, “The Two Cultures: Mashing up Web 2.0 and the Semantic Web,” *Web Semantics: Science, Services And Agents on the World Wide Web*, vol. 6, pp. 70-75, 2008.

[12] D. Karger, and D. Quan, “What Would It Mean to Blog on the Semantic Web?,” *Web Semantics: Science, Services And Agents On The World Wide Web*, vol. 3, pp. 2-3, 2005.

[13] D. Glez-Peña, A. Lourenco, H. López, M. Reboiro, and F. Fdez, “Web scraping technologies in an API World,” *Brief in Bioinformatics,* vol. 15, p.788, 2014.

[14] C. Bonifacio, T. E. Barchyn, C. H. Hugenholtz, and S. W. Kienzle, “A free Canadian climate data scraping tool,” *Computers & Geosciences*, vol. 75, pp. 13-16, 2015.

[15] T. Grigalis, and A. Cenys, “Stateof-the-art web data extraction systems for online business intelligence”. 2013.

[16] C. Hernandez, “Aplicación de Técnicas de Web Scraping al Boletín Oficial de Castilla y León,” *M.S. thesis, E.U. Informática, Univ. Valladolid, España*, 2014.

[17] A. Silva, V. Gracia, and A. Galán, “Metodología de web scraping para Android y ejemplificación mediante la aplicación UPMDroid,” *Thesis, E.T.S.I. y Sist. de Telecom., Univ. Politec. Madrid, España*, 2014.

[18] S. M. Kerner, “ScrapeDefender Protects Data by Thwarting Web-Scraping Attempts,” [Online] Available: http://www.eweek.com/security/scrapedefender-protects-data-by-thwarting-web-scraping-attempts.html, Accessed on: April 23, 2015.

[19] P. Grlica, “Web scraping techniques,” *Ph.D. thesis, Fac. of Comp. and Inf. Science, Univ. of Ljubljana, Slovenia*, 2013.

[20] E. Ferrara, P. D. Meo, G. Fiumara, and R. Baumgartner, “Web data extraction, applications and techniques: A survey,” *Knowledge-Based Systems,* vol. 70, pp. 301 – 323, 2014.

[21] R. M. Dewan, M. L. Freimer, and Y. Jian, “A Temporary Monopolist: Taking Advantage of Information Transparency on the Web,” *Management Information Systems*, vol. 24 pp. 167-194, 2007.

[22] S. O'Reilly, “Nominative Fair Use and Internet Aggregators: Copyright and Trademark Challenges Posed By Bots, Web Crawlers and Screen-Scraping Technologies,” *Loyola Consumer Law Review,* vol. 19, pp. 273-288, 2007.

[23] J. I. Fernández-Villamor, J. Blasco-García, C. A. Iglesias, and M. Garijo, “A Semantic Scraping Model for Web Resources - Applying Linked Data to Web Page Screen Scraping,” *Filipe & A. L. N. Fred (eds.),* vol. 2, pp. 451-456, 2011.

[24] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, “An investigation of web crawler behavior: characterization and metrics,” *Computer Communications*, vol. 28, pp. 880–897, 2005.

[25] *Adobe Flash Player 10.1 Administration Guide,* Adobe Systems Incorporated, 2010.

[26] M. Mohirta, A. S. Cernian, D. Carstoiu, A. M. Vladu, A. Olteanu and V. Sgarciu, “A semantic Web based scientific news aggregator,” *Applied Computational Intelligence and Informatics,* pp. 285-289, 2011.

[27] U.S. Court for The Central District of California, “Ticketmaster Corp. v. Tickets.com, Inc.,” [Online] Available: http://www.internetlibrary.com/cases/lib_case25.cfm, Accessed on: May 5, 2015.

[28] U.S. Court for The Southern District of New York, “Register.com, Inc. v. Verio, Inc.,” [Online] Available: http://www.internetlibrary.com/cases/lib_case23.cfm, Accessed on: May 5, 2015.

[29] N. Sleeth, “Century 21 v. Zoocasa: Contract and Copyright in the Electronic World,” [Online] Available: http://www.iposgoode.ca/2011/09/century-21-v-zoocasa-contract-and-copyright-in-the-electronic-world, Accessed on: May 7, 2015.

[30] D. Qiu, B. Li and H. Leung, “Understanding the API usage in Java,” *Information and Software Technology,* vol. 73, pp. 81-100, 2016.

[31] R. Baumgartner, W. Gatterbauer, G. Gottlob, “Web data extraction system,” in *Encyclopedia of Database Systems*, USA: Springer US, 2009, pp. 3465-3471.

[32] S. Luján, “HTML,” in *Programación de aplicaciones web: historia, principios básicos y clientes web*, España: Edit. Club Universitario, 2002, ch. 6, pp. 91-104.

[33] K. Laudon, C. Traver, “La revolución acaba de empezar,” in *E-commerce 2013 Negocios, tecnología y sociedad*, México: Pearson Educ., 2014, 9th ed, ch. 1, p. 12.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Introduction to Design Patterns” in *Design patterns: Elements of reusable object-oriented software*, USA: Addison-Wesley, 1995, ch. 1, pp. 11-44.

[35] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, “Web Presentation,” in *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003, ch. 4, pp. 70-77.

[36] A. Haque, S. Singh, “Anti-scraping application development,” *Advances in Computing, Communications and Informatics, 2015 International Conference*, pp. 869-874, 2015.

[37] R. A. Gultom, R. F. Sari, and B. Budiardjo, “Implementing web data extraction and making Mashup with Xtractorz,” *IEEE 2nd International Advance Computing Conference, pp.385-393,* 2010.

[38] Y. M. Mileva, V. Dallmeier, M. Burger and A. Zeller, “Mining trends of library usage,” *Principles of Software Evolution and Software Evolution Workshops,* pp. 57-72, 2009.

[39] The Computer Advisor, “Web site scraper – The most effective tool for web data extraction,” [Online] Available: http://www.thecomputeradvisor.net/web-site-scraper-the-most-effective-tool-for-web-data-extraction, Accessed on: April 20, 2015.