A Reference Architecture to Devise Web Information Extractors[*]

Hassan A. Sleiman and Rafael Corchuelo

Universidad de Sevilla, ETSI Informática,
Avda. Reina Mercedes, s/n, Sevilla E-41012
{hassansleiman,corchu}@us.es
http://tdg-seville.info/

Abstract. The Web is the largest repository of human-friendly information. Unfortunately, web information is embedded in formatting tags and is surrounded by irrelevant information. Researchers are working on information extractors that allow transforming this information into structured data for its later integration into automated processes. Devising a new information extraction technique requires an array of tasks that are specific to this technique and many tasks that are actually common between all techniques. The lack of a reference architectural proposal in the literature to guide software engineers in the design and implementation of information extractors, amounts to little reuse and the focus is usually blurred because of irrelevant details. In this paper, we present a reference architecture to design and implement rule learners for information extractors. We have implemented a software framework to support our architecture, and we have validated it by means of four case studies and a number of experiments that prove that our proposal helps reduce development costs significantly.

Keywords: Information Extraction, Rule Learning Reference Architecture.

1 Introduction

The Web contains a huge amount of information and it is a still growing data container. This unlimited repository aroused enterprises' interests in exploiting this information from the Web by devising new applications that consume and analyse this information. Unfortunately, integrating this information into business processes is a costly task since web information is usually embedded in HTML tags and buried in other superfluous contents. This has motivated many authors to work on web information extractors that allow extracting relevant information from web pages and structuring it in relational tables, which can be easily

[*] This work was supported by the European Commission (FEDER funds), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

consumed by business processes. Information extractors allow extracting information from free-text web pages such as news and blogs, or from semi-structured web pages such as search results and web pages with detailed information about some items. We focus on information extractors from semi-structured web pages.

An information extractor is a general algorithm that can be configured by means of rules to extract the information of interest from a web page and to return it according to a structured model. These rules can be handcrafted [9, 4], predefined as heuristics [3, 5], learnt using supervised techniques that require the user to annotate information of interest in a set of training pages [14, 19], or learnt using non-supervised techniques which learn rules to extract the information the technique considers as data of interest [10, 15]. Common information extraction rules range from regular expressions to context free grammars, first-order rules, XPath templates, and transducers. Our work focuses on techniques that learn transducers to extract information for semi-structured web pages.

The literature reports on few proposals to help researchers and software engineers build their rule learners [2, 8, 13, 20, 17]. These proposals claim to offer an environment in which rule learning techniques can be developed and tested. However, these proposals were presented as tools and the description of their architecture is not clear. Furthermore, they are neither available nor maintained any more. UIMA [12] and Gate [11] are frameworks that can be used to manage large volumes of information, but they both focus on free text pages.

Several proposals referenced the lack of a reference architecture to help software engineers develop extraction rule learners from semi-structured web pages [7, 21, 22]. This is problematic insofar researchers need to implement their proposals from scratch in order to validate them, i.e., they need to pay attention to a variety of details that are ancillary and common to many other proposals, but do not constitute the core of their research [1]. The lack of a reference architecture has also led to a variety of terminologies, which makes communication amongst software engineers difficult, and to experimental results that are not comparable empirically due to differences in the designs and the implementations.

In this paper, we present a reference architecture to devise learners of information extraction rules. To support our reference architecture, we have implemented a software framework and we have validated it by means of four case studies and a series of experiments that prove that it helps reduce development costs significantly and compare the techniques we have implemented empirically. We use the 4 + 1 architectural view model proposed by Kruchten [18] to describe our reference architecture. In Sections 2–4, we describe the logical, the development, and the scenarios views, respectively. Since our proposal is not intended to be a functioning system, the process and the physical views, which focus on non-functional requirements like concurrency, distribution, topology or communication, do not actually make sense in this case. Section 5 reports on the accompanying framework and on the implementation of several rule learning techniques from the literature, which are compared empirically. Finally, Section 6 concludes our work.

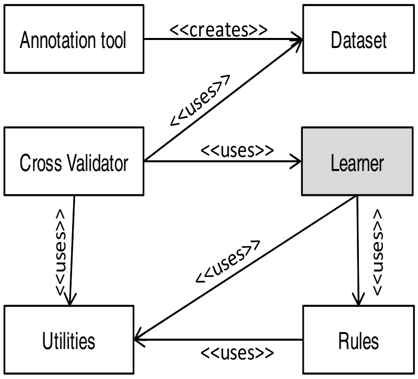


Fig. 1. Relationships amongst subsystems

2 Logical View

The logical view of an architecture represents the functional requirements the system should provide to its end user. In our case, the end user is a software engineer who aims at devising a new extraction rules learning proposal, so in this view we will describe the subsystems, the services they provide, and the interactions amongst them.

The architecture is divided into the following subsystems, whose relationships are shown in Figure 1:

Annotation tool: The reference architecture relies on an annotation tool with which users can download and annotate web pages according to an OWL ontology in which he or she describes classes, properties and their relationships. Ontology classes are used to represent records of information, object properties represent nested records, and data properties represent attributes.

Dataset: This subsystem provides services that allow end users to work with annotations and persist them. During the annotation process, this subsystem allows users to instantiate ontology classes and properties in addition to their position in the corresponding web page. During the learning process, end users can use a dataset to work with a text view or a tree view of the pages they have annotated, get the annotations sorted according to their position or to their type, obtain separating texts between annotations or work with DOM trees and annotation nodes. During the extraction process, this subsystem allows end users to persist the information that is extracted to OWL files.

Learner: This subsystem provides end users with services to develop rule learners. For example, there is a service to create the skeleton of a transducer for a given dataset, i.e., its states and transitions, but not the transition conditions. It saves end users from the burden of inferring the structure of a transducer from

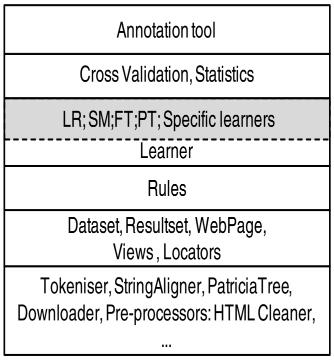

Annotation tool
Cross Validation, Statistics
LR; SM;FT;PT; Specific learners
Learner
Rules
Dataset, Resultset, WebPage, Views , Locators
Tokeniser, StringAligner, PatriciaTree, Downloader, Pre-processors: HTML Cleaner, ...

Fig. 2. Reference Architecture layers

the annotations in a dataset, since this is common to every learning algorithm. Note that this subsystem is a point of variability where software engineers only have to focus on devising their own learning algorithms to learn extraction rules.

Rules: This subsystem provides a service to construct extraction rules step by step and to execute them on web pages in order to extract information.

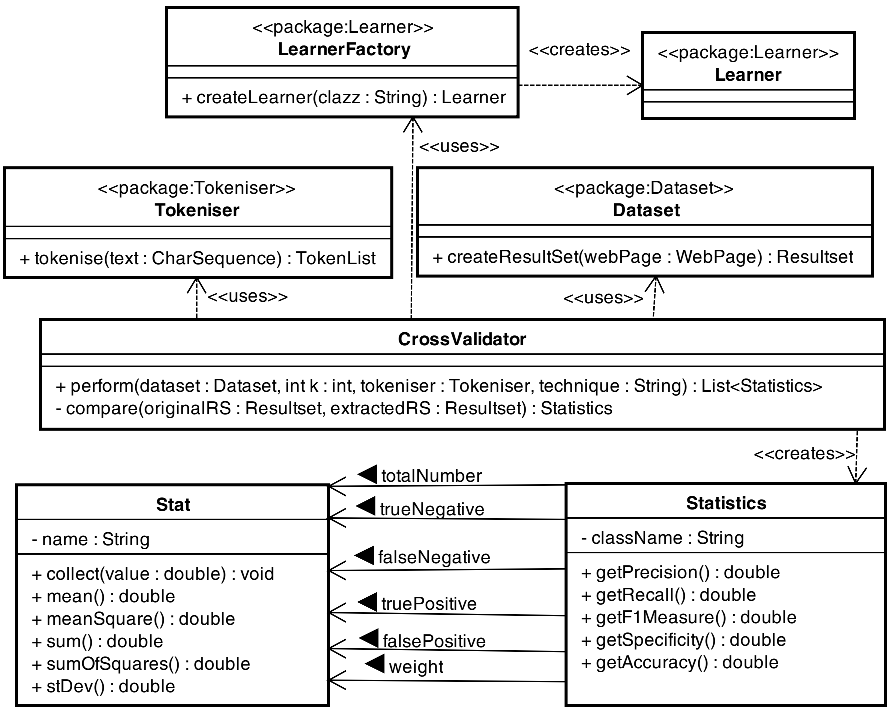
Cross Validator: This subsystem provides a tool with which end users can k-cross validate their rule learners. It helps collect precision, recall, specificity, accuracy, and the $F1$-measure. Thanks to this tool, the results about a given proposal are empirically comparable to other proposals.

Utilities: This subsystem offers some utilities to the rest of subsystems, namely: a configurable tokeniser, a web page downloader, preprocessors such as an HTML cleaner or data region identifier, and a few string and tree alignment algorithms.

3 Development View

This view shows the system from a developer's perspective by illustrating the module organisation of the system and the class diagrams of each module which are the basis for assigning work packages to the members of a development team. Our reference architecture is composed of layers each of which has a well defined responsibility and provides services to the layers above it. These layers are illustrated in Figure 2 and are studied below.

The annotation tool layer: This is the upper layer in our reference architecture. It contains a tool for users to create Datasets. It uses the lower layers to download, clean HTML, add annotations, and to save Datasets. Ontologies are used to create the annotations of each web page.



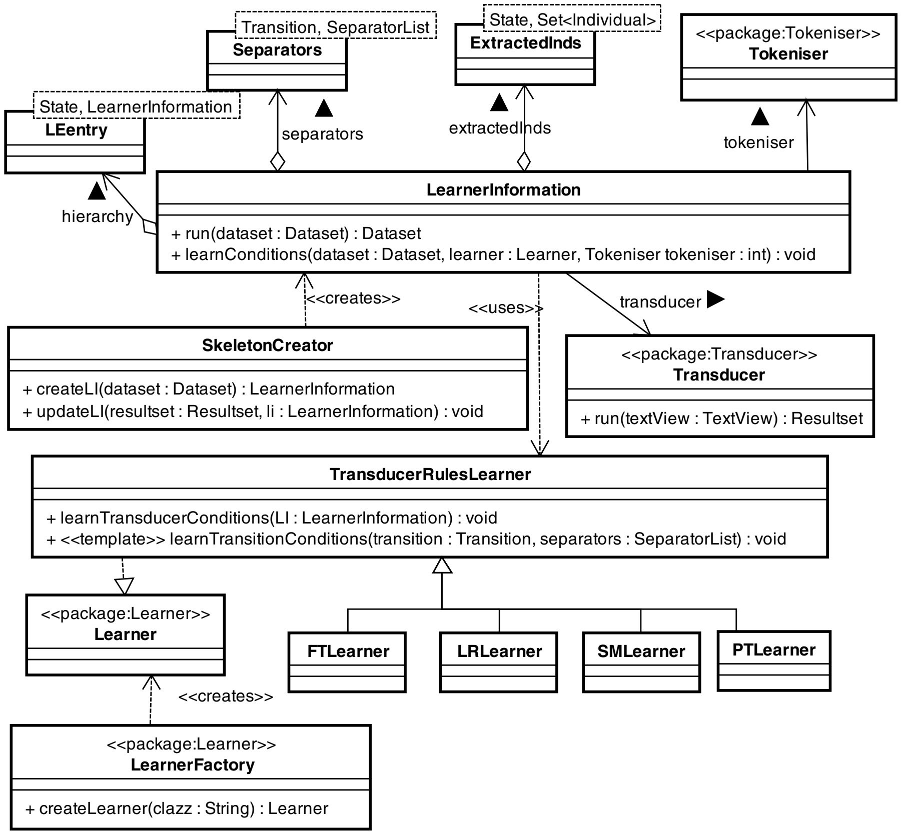
<<package:Learner>>
LearnerFactory
+ createLearner(clazz : String) : Learner
<<creates>>
<<package:Learner>>
Learner
<<uses>>
<<package:Tokeniser>>
Tokeniser
+ tokenise(text : CharSequence) : TokenList
<<package:Dataset>>
Dataset
+ createResultSet(webPage : WebPage) : Resultset
<<uses>>
<<uses>>
CrossValidator
+ perform(dataset : Dataset, int k : int, tokeniser : Tokeniser, technique : String) : List<Statistics>
- compare(originalRS : Resultset, extractedRS : Resultset) : Statistics
<<creates>>
Stat
- name : String
+ collect(value : double) : void
+ mean() : double
+ meanSquare() : double
+ sum() : double
+ sumOfSquares() : double
+ stDev() : double
totalNumber
trueNegative
falseNegative
truePositive
falsePositive
weight
Statistics
- className : String
+ getPrecision() : double
+ getRecall() : double
+ getF1Measure() : double
+ getSpecificity() : double
+ getAccuracy() : double

Fig. 3. UML Diagram for the Cross Validation Layer

The cross validator layer: Cross validation is used to estimate the system performance in the practice for a given web site and to obtain comparable results on a Dataset. Given a Dataset obtained from a web site, the k-fold cross validation technique partitions the Dataset into k subsets of web pages with their corresponding Resultsets and then starts iterating over them. At each iteration, it considers one of these subsets for testing, whereas the remaining subsets are considered as a unique set which is used to learn rules. The rules learnt at each iteration are tested on the selected subset, and some statistics are collected. At the end, the weighted arithmetic mean of each calculated value is returned.

The fifth layer contains the the classes of the CrossValidator and the classes to collect results during a k-cross validation Figure 3. These classes are:

– CrossValidator: This class provides methods to perform a k-cross validation on a given a Dataset for a specific Learner. It uses a LearnerFactory to create the Learner to be tested, and during the cross validation process, it compares the Resultsets obtained with the annotated ones to collect the previous Statistics.
– Statistics: This class provides methods to collect the following statistics: true positives, false positives, true negatives, false negatives, the total number of annotations at each iteration and the weight of each class and property in



Transition, SeparatorList
Separators
State, Set<Individual>
ExtractedInds
<<package:Tokeniser>>
Tokeniser
State, LearnerInformation
LEentry
separators
extractedInds
tokeniser
hierarchy
LearnerInformation
+ run(dataset : Dataset) : Dataset
+ learnConditions(dataset : Dataset, learner : Learner, Tokeniser tokeniser : int) : void
<<creates>>
<<uses>>
transducer
SkeletonCreator
+ createLI(dataset : Dataset) : LearnerInformation
+ updateLI(resultset : Resultset, li : LearnerInformation) : void
<<package:Transducer>>
Transducer
+ run(textView : TextView) : Resultset
TransducerRulesLearner
+ learnTransducerConditions(LI : LearnerInformation) : void
+ <<template>> learnTransitionConditions(transition : Transition, separators : SeparatorList) : void
<<package:Learner>>
Learner
FTLearner
LRLearner
SMLearner
PTLearner
<<creates>>
<<package:Learner>>
LearnerFactory
+ createLearner(clazz : String) : Learner

Fig. 4. UML Diagram for the Learners Layer

the Dataset. The provided methods in this class to calculate effectiveness and efficiency values use these attributes.

– Stat: A statistics class in which values can be collected. It provides methods to calculate statistical measures such as arithmetic mean and standard deviation.

The learners layer: The key of this layer is that it is open, i.e., is intended to provide an extension point software engineers should use to create their own rule learners, cf. the gray band in Figure 2. In our accompanying framework we have implemented several extensions to implement learners that learn transducers and that are based on SoftMealy [14], LR [19], FivaTech [15], and IEPAD [6]. The classes in this layer are the following:

– LearnerFactory: This class is intended to provide a method to create a Learner of a given type.

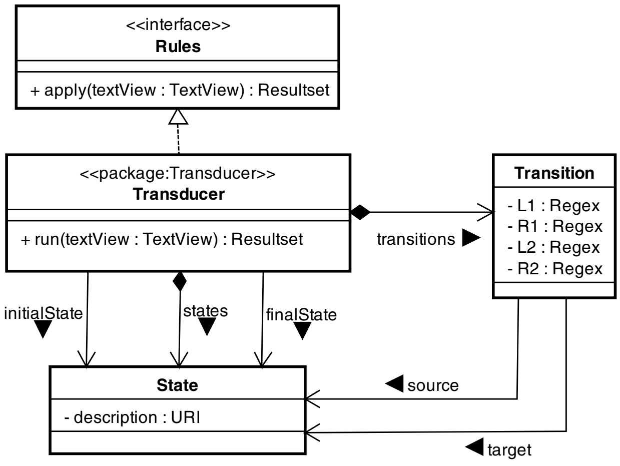


Fig. 5. The UML diagram for the Rules Layer

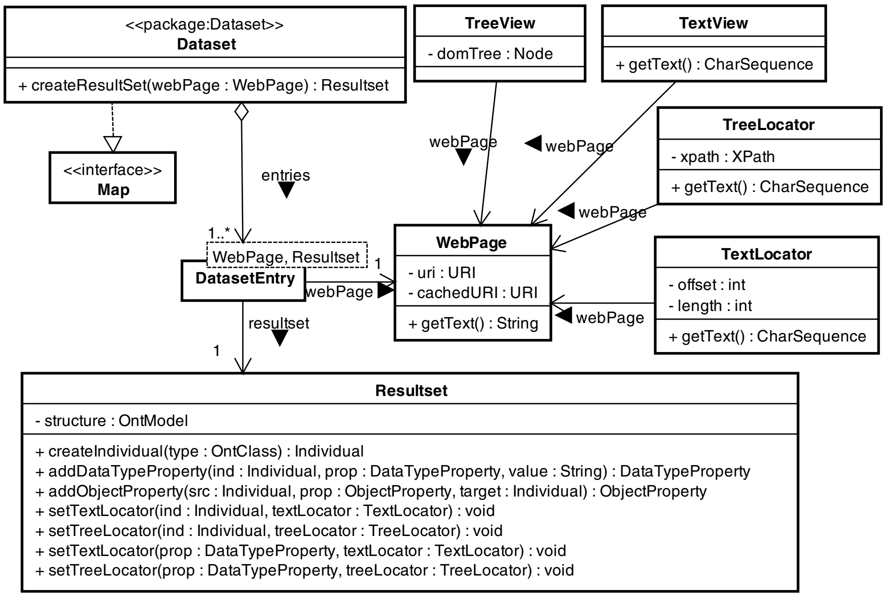
- Learner: An interface that should be implemented by the rule learners devised by users.
- SkeletonCreator, LearnerInformation, and TransducerRulesLearner : These classes are created to model the rule learners that learn transducers. The SkeletonCreator allows creating the skeleton of a transducer, LearnerInformation models information related to each state and to each transition in the transducer, and the TransducerRulesLearner is intended to learn the transition condition using a specific learning technique.
- FTLearner, LRLearner, PTLearner, and SMLearner: These classes model specific learners that learn transition conditions using techniques inspired in FivaTech [15], LR [19], IEPAD [6], and SoftMealy [14] respectively.

The rules layer: The third layer contains the rules created by learners and used for information extraction. In our accompanying framework we have implemented the rules of type transducers, c.f. 5. The classes in this layer are the following:

- Rules: This is an interface that should be implemented by the different types of extraction rules learnt by the Learners in the upper layer.
- Transducer: This class models a type of rules some information extraction techniques learn. It contains a collection of States and Transitions. Inside each Transition reside its conditions which are regular expressions.

The dataset layer: The second layer contains the dataset classes that help maintain the user annotations and the extracted data. The classes contained in this layer are shown in Figure 6, namely:

- Dataset: It models a collection of Resultsets and web pages. Each contains a map from a set of web pages onto their corresponding Resultsets.



<<package:Dataset>>
Dataset
+ createResultSet(webPage : WebPage) : Resultset
<<interface>>
Map
TreeView
- domTree : Node
TextView
+ getText() : CharSequence
TreeLocator
- xpath : XPath
+ getText() : CharSequence
TextLocator
- offset : int
- length : int
+ getText() : CharSequence
WebPage
- uri : URI
- cachedURI : URI
+ getText() : String
DatasetEntry
WebPage, Resultset
entries
webPage
resultset
1..*
1
Resultset
- structure : OntModel
+ createIndividual(type : OntClass) : Individual
+ addDataTypeProperty(ind : Individual, prop : DataTypeProperty, value : String) : DataTypeProperty
+ addObjectProperty(src : Individual, prop : ObjectProperty, target : Individual) : ObjectProperty
+ setTextLocator(ind : Individual, textLocator : TextLocator) : void
+ setTreeLocator(ind : Individual, treeLocator : TreeLocator) : void
+ setTextLocator(prop : DataTypeProperty, textLocator : TextLocator) : void
+ setTreeLocator(prop : DataTypeProperty, treeLocator : TreeLocator) : void

Fig. 6. The Dataset layer

– Resultset: This class models the annotations on a web page and allows to save them as an instance of an ontology.
– WebPage: This class is used to represent a web page. It keeps a reference to where it was downloaded to local cache (cachedURI) and its original location (uri).
– TextLocator and TreeLocator: These classes are used to provide locations to the annotations. Each instance of a class and property in the Resultset has both locators. The TextLocator saves information about the offset and length of an annotation whereas the TreeLocator contains the XPath of the annotation in the DOM Tree. Both locators allow obtaining the text contained in each annotation.
– TextView and TreeView: There classes model views used in the learning process. The TextView provides a view over the text content of a WebPage. The TreeView provides a view over the DOM tree.
– DatasetPersistence: This is a class used to save and load Datasets.

The utility layer: At the bottom of the layers of our architecture resides the utility classes that are used by the upper layers. These utilities include the following classes:

– Tokeniser: A class to implement a configurable tokeniser, cf. Figure 7. The tokeniser is configured by means of an XML file in which hierarchy between different token classes are defined besides the regular expression that defines

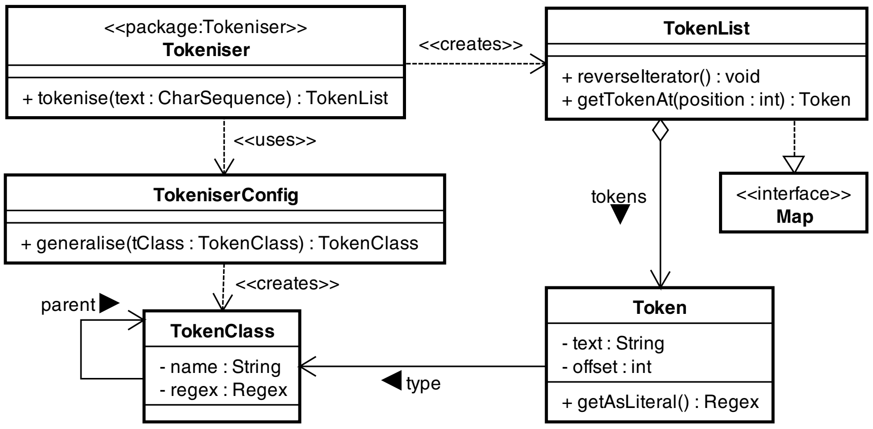


Fig. 7. UML Diagram for the Tokeniser

each type of TokenClass. When a text is tokenised, the Tokeniser returns a TokenList which is a Map of Tokens sorted by their offset in the tokenised text. The methods generalise in TokeniserConfig and getAsLiteral in class Token allow generalisation and specialisation of tokens, respectively.

– StringAligner: This class provides an implementation of a multiple string alignment algorithm that is similar to the one presented in [15].
– PatriciaTree: This class constructs a PatriciaTree starting from a set of token sequences. It provides methods to be updated by adding a new token sequences and to build the regular expression that corresponds to this tree.
– Downloader: This class downloads web pages locally. To ensure the reproducibility of tests, annotated web pages are downloaded and saved locally to avoid that changes in web sites affect user annotations and tests.
– Pre-processors: They are usually used before rule learning or information extraction. HTMLCleaner can be used to fix the HTML code of downloaded web pages while region extractors can be used in some learning techniques to identify the region in which the data of interest resides.

4 Scenarios View

According to Kruchten [18], this view shows how an architecture is instantiated in typical use cases. It serves two purposes: as an illustration of how the architecture can be used and as a validation of the reference architecture since the described scenarios are supposed to be an abstraction of the most important requirements.

SCEN1: Developing a new learning technique: This scenario intends to show how a software engineer can devise his own technique to learn extraction rules of type transducers and integrate it into our reference architecture. The steps he or she should perform are as follows:

- User imports the rule learning framework.
- The user creates a class that extends class TransducerRuleLearner which implements interface Learner.
- A Tokeniser should be created by defining an XML file with the tokenisation hierarchy that will be used by the new technique.
- The user defines the template method learnTransitionConditions in the new class created. This method includes the code necessary to learn the transition conditions for the learnt transducer.

SCEN2: Testing a new learning technique: This scenario intends to show how a software engineer can test his own technique and obtain comparable results by using a 10-folds cross validation, for example. The steps to validate a learning technique is defined below:

- The user downloads our testing Datasets.
- The method perform from the class CrossValidator is called. This method receives $k = 10$, the Tokeniser and the name of the learning technique that will be tested.
- The CrossValidator performs a 10 fold cross validation and save results in Statistics. These Statistics are returned by the CrossValidator and allow to obtain measurements such as Precision and Recall for each class and property in the used ontology.

SCEN3: Learning extraction rules: This scenario describes how user can learn extraction rules for a given web site and save them for future use. The steps to learn these rules, represented as transducers in this case, are the following:

- The user annotates a Dataset using web pages from the web site for which he or she wishes to learn extraction rules. Annotations can be performed using an annotation tool provided with our framework.
- The user should create a SkeletonLearner and call the method create with the Dataset as a parameter. It returns a LearnerInformation object.
- The user creates an object of the new learning technique.
- The user now can learn the transition conditions by calling the method LearnConditions with the Dataset as input.

SCEN4: Applying rules on an input web page: This scenario describes how users can apply extraction rules on web pages to extract information. User should perform the following steps:

- The user loads a Transducer using the TransducerPersistence class.
- The user should create a Resultset for the web page of interest. The extracted information will be saved there.
- The user creates a TextView over the input web page and calls method apply of the transducer with the TextView, a zero to indicate the starting offset, and the Resultset as parameters.

Table 1. Comparing implementation times for NLR, SM, FT and PT

Technique	Using Java 1.6 only	Using our framework	Reduction percentage
NLR	145hrs	32hrs	77.94%
SM	123hrs	87hrs	29.27%
FT	176hrs	61hrs	65.34%
PT	110hrs	30hrs	72.72%

- The transducer runs on this TextView and saves the information extracted into the Resultset. A Dataset with the input WebPage and the final Resultset is created.
- DatasetPersistence should be used now to save the resulting Dataset.

5 Experimental Results

The aim of our reference architecture is to reduce the costs of devising rule learners and to allow comparing learners with each other. To validate it, we have conducted two experiments following the guidelines reported in [16].

First, we have developed a framework to check the viability of our reference architecture. The framework is available for the research community at `http://www.tdg-seville.info/Hassan`. This framework was used to develop a number of the most cited proposals in the literature that are based on transducers or that can be adapted to be used with transducers. We have implemented NLR [19], SM [14], FT [15] and PT [6].

In [16], the authors presented a detailed proposal to guide experimental evaluations. They identified a number of common threats in which we have not incurred:

- Using students in experimental validations: We implemented our techniques with the help of Master Degree students, all of which were junior software engineers who had been working in the industry for a year at least.
- The techniques with which the comparison was performed are not representative: We selected four of the most cited techniques in the literature.
- The experiments are not repeatable: Our proposal and our experiments are available to the research community; anyone can download it, go through the implementation and the documentation and repeat the experiments as long as he or she has a Java 1.6 virtual machine available.
- The results are not comparable: We used 10-fold cross validation, which is the de facto standard to compare experimental results in the field of machine learning.
- Timings are not accurate: We have used the java.lang.management package to measure timings, and every experiment was repeated several times to make sure that the results were accurate. Note that this package allows to measure the time consumed by a single thread, without interferences from other threads or processes.

Table 2. Comparing precision and recall of NLR, SM, FT and PT techniques

Dataset	NLR		SM		FT		PT	
	P	R	P	R	P	R	P	R
soulfilms.com	0.912	0.765	0.993	0.800	1.000	0.000	1.000	0.000
albanianfilmdatabase.com	0.874	0.245	0.874	0.304	0.962	0.371	1.000	0.283
disneymovieslist.com	0.731	0.731	0.989	0.460	1.000	0.000	1.000	0.000
imdb.org	0.753	0.855	0.985	0.845	1.000	0.124	1.000	0.124
citwf.com	0.915	0.915	0.981	0.878	0.992	0.892	0.991	0.343
awesomebooks.com	1.000	0.946	0.830	0.315	1.000	0.676	1.000	0.485
betterworldbooks.com	0.993	0.915	0.877	0.844	0.920	0.514	0.979	0.985
manybooks.net	0.974	0.824	0.746	0.067	0.770	0.536	0.992	0.214

Our first experiment was conducted to check if relying on our reference architecture and an accompanying framework, development costs were reduced remarkably. For this purpose, we requested four postgraduate students with a degree in Software Engineering, to study the previous proposals. Then, they were requested to implement them using Java 1.6. The time to study the requirements, design, develop and test these proposals was measured in hours. Since their development processes were totally independent, each of the participants had to create their datasets for testing.

New postgraduate students were requested to develop the same techniques, but this time using our framework. First, they went through to a training period that was added to the total time that was necessary to study the requirements, design, develop and test the developed techniques. In this case, datasets were reused between the different participants to compare the techniques side by side too. The results of our first experiment are reported in Table 1.

Table 1 shows the time in hours that was necessary to develop and test the techniques on which we report in the first column. The second and the third columns show the time that was necessary to develop these techniques using only Java libraries and using our framework, respectively. The fourth column shows the time reduction for each technique. The costs reduction is clear since the framework allowed reusing components during the development phase and reusing datasets in the testing phase. The last column shows the reduced time percentage; the arithmetic mean of the reduction percentage is 61.31%.

The second experiment was conducted to show the results of the different techniques developed using the framework can be compared now under homogeneous conditions. The implementations obtained in the previous steps were used to perform this empirical study. To the best of our knowledge, this is the first time an empirical comparison between different information extraction techniques, using the same technology and on the same datasets have been reported in literature. Only eight datasets, which were chosen arbitrary, were used in this experiment since our aim is to show how the results are comparable side by side.

Table 2 reports on the results of applying these techniques in practice on several datasets. The first column is the site from which each dataset was obtained. The other columns provide precision (P) and recall (R), which are de facto standard measurements in the information extraction domain. Precision is the percentage of correct information extracted by an information extractor, whereas recall is the percentage of correct information that is successfully extracted. When $P = 1.0$, this means that all the extracted information by the technique is correct, and when $R = 1.0$, this means that all the information that the technique should extract, was extracted correctly. When we apply FT technique over *soulfilms.com* and *disneymovieslist.com* datasets, the technique does not extract any information, which means that all the extracted information is correct and then $P = 1.0$, but none of the correct information is extracted and then $R = 0.0$.

Each dataset contains 30 web pages from the studied site, and the results regarding precision and recall were calculated using 10-fold cross validation. Note that the studied techniques may obtain better results regarding precision and recall by studying sites more thoroughly and adding to the dataset web pages that better represent the web site and that may obtain better extraction rules for this web site, but this is not the focus in this paper.

6 Conclusions

In this paper, we have presented a reference architecture to help software engineers devise new learning techniques in the domain of information extraction. This is the first reference architecture in the literature which provides an abstract, reusable, easy-to-maintain, and easy-to-adapt design that should allow software engineers and researchers face the development of a new rule learning technique without incurring the high costs of developing it from scratch. The reference architecture was validated by an accompanying framework, which was used to show the possibility of developing techniques from literature using our reference architecture, with an overall time reduction that goes beyond 60%.

References

1. Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Lyytinen, K., Loucopoulos, P., Robinson, B., Ralph, P., Wand, Y.: Design Requirements Engineering: A Ten-Year Perspective, vol. 14. Springer, Heidelberg (2009)
2. Adelberg, B.: NoDoSE - a tool for semi-automatically extracting semi-structured data from text documents. In: SIGMOD Conference, pp. 283–294 (1998)
3. Álvarez, M., Pan, A., Raposo, J., Bellas, F., Cacheda, F.: Extracting lists of data records from semi-structured web pages. Data Knowl. Eng. 64(2), 491–509 (2008)
4. Arocena, G.O., Mendelzon, A.O.: WebOQL: Restructuring documents, databases, and webs. TAPOS 5(3), 127–141 (1999)
5. Buttler, D., Liu, L., Pu, C.: A fully automated object extraction system for the world wide web. In: ICDCS, pp. 361–370 (2001)
6. Chang, C.-H., Lui, S.-C.: IEPAD: information extraction based on pattern discovery. In: WWW, pp. 681–688 (2001)

7. Chiticariu, L., Li, Y., Raghavan, S., Reiss, F.: Enterprise information extraction: recent developments and open challenges. In: SIGMOD Conference (2010)
8. Cohen, W.W., Hurst, M., Jensen, L.S.: A flexible learning system for wrapping tables and lists in HTML documents. In: WWW, pp. 232–241 (2002)
9. Crescenzi, V., Mecca, G.: Grammars have exceptions. Inf. Syst. 23(8), 539–565 (1998)
10. Crescenzi, V., Mecca, G., Merialdo, P.: RoadRunner: automatic data extraction from data-intensive web sites. In: SIGMOD Conference, p. 624 (2002)
11. Cunningham, H., Humphreys, K., Gaizauskas, R.J., Wilks, Y.: Software infrastructure for natural language processing. CoRR, cmp-lg/9702005 (1997)
12. Ferrucci, D., Lally, A.: Uima: an architectural approach to unstructured information processing in the corporate research environment. Nat. Lang. Eng. 10, 327–348 (2004)
13. Han, W., Buttler, D., Pu, C.: Wrapping web data into XML. SIGMOD Record 30(3), 33–38 (2001)
14. Hsu, C.-N., Dung, M.-T.: Generating finite-state transducers for semi-structured data extraction from the web. Inf. Syst. 23(8), 521–538 (1998)
15. Kayed, M., Chang, C.-H.: FiVaTech: Page-level web data extraction from template pages. IEEE Trans. Knowl. Data Eng. (2010)
16. Kitchenham, B., Pfleeger, S.L., Pickard, L., Jones, P., Hoaglin, D.C., Rosenberg, J., Emam, K.E.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Software Eng. 28(8), 721–734 (2002)
17. Kiyavitskaya, N., Zeni, N., Cordy, J.R., Mich, L., Mylopoulos, J.: Cerno: Lightweight tool support for semantic annotation of textual documents. Data Knowl. Eng. (2009)
18. Kruchten, P.: The 4+1 view model of architecture. IEEE Software 12(6), 42–50 (1995)
19. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. Artif. Intell. 118(1-2), 15–68 (2000)
20. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S.: DEByE - data extraction by example. Data Knowl. Eng. 40(2), 121–154 (2002)
21. Lavelli, A., Califf, M.E., Ciravegna, F., Freitag, D., Kushmerick, N., Giuliano, C., Romano, L., Ireson, N.: Evaluation of machine learning-based information extraction algorithms: criticisms and recommendations. Language Resources and Evaluation (2008)
22. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: Conference on Innovative Data Systems Research, pp. 342–350 (2007)