Coping with Web Knowledge

J.L. Arjona, R. Corchuelo, J. Peña, and D. Ruiz

The Distributed Group
Avda. de la Reina Mercedes, s/n, Sevilla (Spain)
{arjona,corchuelo,joaquinp, druiz}@lsi.us.es

Abstract. The web seems to be the biggest existing information repository. The extraction of information from this repository has attracted the interest of many researchers, who have developed intelligent algorithms (wrappers) able to extract structured syntactic information automatically.
In this article, we formalise a new solution in order to extract knowledge from today's non-semantic web. It is novel in that it associates semantics with the information extracted, which improves agent interoperability; furthermore, it achieves to delegate the knowledge extraction procedure to specialist agents, easing software development and promoting software reuse and maintainability.

Keywords: knowledge extraction, wrappers, web agents and ontologies

1 Introduction

In recent years, the web has consolidated as one of the most important knowledge repositories. Furthermore, the technology has evolved to a point in which sophisticated new generation web agents proliferate. A major challenge for them has become sifting through an unwieldy amount of data to extract meaningful information. This process is difficult because the information on the web is mostly available in human-readable forms that lack formalised semantics that would help agents use it.

The Semantic Web is "*an extension to the current web in which information is given well–defined meaning, better enabling computers and people to work in cooperation*" [3], which implies a transition from today's web to a web in which machine reasoning will be ubiquitous and devastatingly powerful. This transition is achieved by annotating web pages with meta–data that describe the concepts that define the semantics associated with the information in which we are interested. Ontologies play an important role in this task, and there are many ontological languages that aim at solving this problem, e.g., DAML+OIL [13], SHOE [17] or RDF-Schema [5]. The Semantic Web shall simplify and improve

[⋆] The work reported in this article was supported by the Spanish Interministerial Commission on Science and Technology under grants TIC2000-1106-C02-01 and FIT-150100-2001-78.

the accuracy of current information extraction techniques tremendously. Nevertheless, this extension requires a great deal of effort to annotate current web pages with semantics, which suggests that it is not likely to be adopted in the immediate future [9].

Several authors have worked on techniques for extracting information from today's non-semantic web, and inductive wrappers are amongst the most popular ones [6,14,15,16,19]. They are components that use automated learning techniques to extract information from similar pages automatically. Although induction wrappers are suited to extract information from the web, they do not associate semantics with the data extracted, this being their major drawback. Furthermore, adding these algorithms to logic that a web agent encapsulates, can produce tangled code and does not achieve a clear separation of concerns.

In this article, we present a new solution in order to extract semantically-meaningful information from today's non-semantic web. It is novel in that it associates semantics with the information extracted, which improves agent interoperability, and it delegates the knowledge extraction procedure to specialist agents, easing software development and promoting software reuse and maintainability.

We address these issues by developing *knowledge channels*, or KCs for short. They are agents [21] that allow to separate the extraction of knowledge from the logic of an agent, and they are able to react to knowledge inquiries (reactivity) from other agents (social ability), and act in the background (autonomy) to maintain a local knowledge base (KB) with knowledge extracted from a web site (proactivity). In order to allow for semantic interoperability, the knowledge they manage references a number of concepts in a given application domain that are described by means of ontologies. KCs extract knowledge from the web using *semantic wrappers*, which are a natural extension to current inductive wrappers to deal with knowledge on the web. Thus, we take advantage of the work made by researchers in the syntactic wrappers arena.

The rest of the paper is organised as follows: Next section glances at other proposals and motivates the need for solutions to solve the problems behind knowledge extraction; Section 3 presents the case study used to illustrate our proposal and some initial concepts related to knowledge representation; Section 4 gives the reader an insight into our proposal; finally, Section 5 summarises our main conclusions.

2 Related Work

Wrappers [8] are one of the the most popular mechanisms for extracting information from the web. Generally, a wrapper is an algorithm that translates the information represented in model M_1 to model M_2. In information extraction, they are able to translate the information in a web page to a data structure that can be used by software applications.

In the beginning, these algorithms were codified manually, using some properties of a web page, normally looking for strings that delimit the data that we need to extract. But hand-coded wrappers are error–prone, tedious, costly and time–consuming to build and maintain. An important contribution to this field was provided by Kushmerick [15]. He introduced induction techniques to define a new class of wrappers called inductive wrappers. These inductive algorithms are components that use a number of extraction rules generated by means of automated learning techniques such as inductive logic programming, statistical methods, and inductive grammars. These rules set up a generic algorithm to extract information from similar pages automatically. Boosted techniques [10] are proposed to improve the performance of the machine learning algorithm by repeatedly applying it to a training set with different example weightings.

Although induction wrappers are suited to extract information from the web, they do not associate semantics with the data extracted, this being their major drawback [2]. Thus, we call current inductive wrappers syntactic because they extract syntactic information devoid of semantic formalisation that expresses its meaning.

Our solution builds on the best of current inductive wrappers, and extends them with techniques that allow us to deal with web knowledge. Using inductive wrappers allows us take advantage of all the work developed in this arena, as boosted techniques or verification algorithms [15,19] that detect if there are changes in the layout of a web page that invalidate the wrapper.

3 Preliminaries

3.1 A Case Study

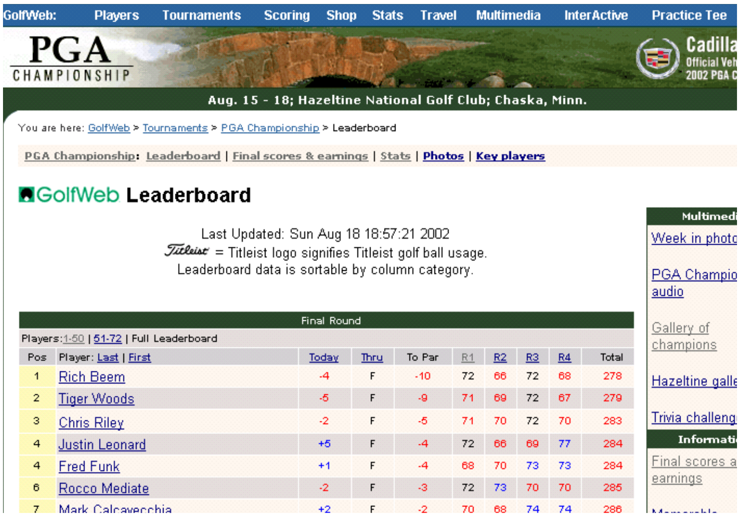
We illustrate the problem to solve by means of a simple, real example in which we are interested in extracting information about the score of golfers in a PGA Championship. This information was given at `http://www.golfweb.com`. Figure 1 shows a web page from this site.

Note that the implied meaning of the terms that appear in this page can be easily interpreted by humans, but there is not a reference to the concepts that describe them precisely, which complicates communication and interoperability amongst software applications [3,4].

3.2 Dealing with Knowledge

There are many formalisms to dealt with knowledge, namely: semantic networks [20], frames systems [18], logic, decision trees, and so on. Their aim is to represent ontologies, which are specifications of concepts and relationships amongst them in a concrete domain. Ontologies [7] allows us to specify the meaning of the concepts about which we are extracting information. Some authors [11,12] have specified a formal model for ontologies; our formalisation builds on the work by Heflin in his PhD dissertation [12].

Definition 1. *Let $\mathcal{L}$ be a logical language; an* ***ontology*** *is a tuple (P, A), where P is a subset of the vocabulary of predicate symbols of $\mathcal{L}$ and A is a subset of*



GolfWeb: Players Tournaments Scoring Shop Stats Travel Multimedia InterActive Practice Tee

PGA CHAMPIONSHIP

Aug. 15 - 18; Hazeltine National Golf Club; Chaska, Minn.

You are here: GolfWeb > Tournaments > PGA Championship > Leaderboard

PGA Championship: Leaderboard | Final scores & earnings | Stats | Photos | Key players

GolfWeb Leaderboard

Last Updated: Sun Aug 18 18:57:21 2002
Titleist = Titleist logo signifies Titleist golf ball usage.
Leaderboard data is sortable by column category.

Final Round

Players:1-50 | 51-72 | Full Leaderboard

Pos	Player: Last | First	Today	Thru	To Par	R1	R2	R3	R4	Total
1	Rich Beem	-4	F	-10	72	66	72	68	278
2	Tiger Woods	-5	F	-9	71	69	72	67	279
3	Chris Riley	-2	F	-5	71	70	72	70	283
4	Justin Leonard	+5	F	-4	72	66	69	77	284
4	Fred Funk	+1	F	-4	68	70	73	73	284
6	Rocco Mediate	-2	F	-3	72	73	70	70	285
7	Mark Calcavecchia	+2	F	-2	70	68	74	74	286

Multimedi
Week in photo
PGA Champio audio
Gallery of champions
Hazeltine galle
Trivia challeng
Informati
Final scores a earnings

Fig. 1. A web page with information about scores in a golf championship.

well–formed formula in $\mathcal{L}$ (axioms). Thus, an ontology is a subset of $\mathcal{L}$ in which concepts are specified by predicates and relationships amongst then are specified as a set of axioms.

First–order languages (FOL) offer us the power and flexibility needed to describe knowledge. Many knowledge representation languages and structures can be formulated in first–order logic [12]. Then, we are able to use a wide range of knowledge representation formalisms; we only need to define a mechanism to translate from some formalism to FOL and vice versa.

In Appendix A, we specify[1] some concepts related to logical languages that establish the basis of our model. In our proposal, a logical language ($\mathcal{L}$) is characterized by a vocabulary of constant identifiers ($Ident_c$), a vocabulary of variable identifiers ($Ident_v$), a vocabulary of function identifiers ($Ident_f$), a vocabulary of predicate identifiers ($Ident_p$) and a (in)finite set of well–formed formulae (Wff), which is a subset of the formulae derived from $\mathcal{L}$. For the sake of simplicity, we assume that $Ident_f = \varnothing$.

Next schema specifies an ontology. Three constrains are imposed: the former states that P and A are non–empty subsets of the set of predicate symbols and well–formed formulae of $\mathcal{L}$, respectively; the second, asserts that axioms are defined using the predicate symbols in P[2]; the latter asserts that the set of axioms is consistent. Predicate $\vdash$ references a theorem prover; let be $F : \mathbb{P}\, Wff$, and $f : Wff$ then $F \vdash f$ is satisfied if f is formally provable or derivable from F,

[1] In this paper we use notation Z as a formal specification language because it is an ISO standard [1], and an extremely expressive language.

[2] Function *PredSyms* is specified in Appendix A. It returns the set of predicate symbols in a formula.

thus f belongs to the set of all well–formed formulae that we can obtain from F (theory of F).

$$\begin{array}{|l}
\textit{Ontology} \\
\hline
P : \mathbb{P}\, \textit{Ident}_p \\
A : \mathbb{P}\, \textit{Wff} \\
\hline
P \neq \varnothing \land A \neq \varnothing \\
\forall f : A \bullet \textit{PredSyms}(f) \subseteq P \\
\neg \exists g : \textit{Wff} \bullet A \vdash g \land A \vdash \neg g \\
\hline
\end{array}$$

Definition 2. *An **instance** of a concept, specified in an ontology, is an interpretation of this concept over some domain. In information extraction, this domain is established by the information to be extracted.*

We model instances as ground predicate atoms. Thus, they are well–formed formula. We specify the set of all instances that we can derive from an ontology by the function *GroundPredicateAtoms*:

$$\begin{array}{|l}
\textit{GroundPredicateAtoms} : \textit{Ontology} \rightarrow \mathbb{P}\, \textit{Wff} \\
\hline
\forall o : \textit{Ontology} \bullet \textit{GroundPredicateAtoms}(o) = \\
\quad \{ f : \textit{Wff};\ ip : \textit{Ident}_p;\ sc : \mathrm{seq}_1\, \textit{Term};\ ic : \textit{Ident}_c \mid \\
\qquad (f = \textit{atom}(\textit{pred}(ip, sc)) \land \\
\qquad \forall c : \textit{Term} \mid c \in sc \bullet c = \textit{const}(ic) \land \\
\qquad \textit{PredSyms}(f) \subseteq o.P) \bullet f \}
\end{array}$$

Definition 3. *A **Knowledge Base** (KB) is a tuple (O, K), where O is an ontology and K a set of instances of concepts specified in O.*

A KB is specified as follows:

$$\begin{array}{|l}
\textit{KB} \\
\hline
O : \textit{Ontology} \\
K : \mathbb{P}\, \textit{Wff} \\
\hline
\forall f : K \bullet \textit{PredSyms}(f) \subseteq O.P \\
K \in \textit{GroundPredicateAtoms}(O) \\
\hline
\end{array}$$

The constrains imposed assert that the instances are formed with predicates defined in the ontology and they are ground predicate atoms.

Example 1. The following object defines a KB in the domain of a golf championship in which we were interested in modelling knowledge about the position and score of golfers in a PGA championship (for the sake of readability, we do not use the abstract syntax in Appendix A. The mapping between this syntax and the usual logic symbols is straightforward):

$$\begin{aligned} KB_0 = \langle\!|\, O \leadsto \langle\!|\; & P \leadsto \{Person, Golfer, Score, Position\}, \\ & A \leadsto \{\forall\, x \bullet Golfer(x) \Rightarrow Person(x), \\ & \qquad \forall\, x \bullet \exists\, y \bullet Golfer(x) \Rightarrow Score(x, y), \\ & \qquad \forall\, x \bullet \exists\, y \bullet Golfer(x) \Rightarrow Position(x, y)\} \,|\!\rangle, \\ K \leadsto \{ & Golfer(\text{Rich\_Beem}), Score(\text{Rich\_Beem}, 278), \\ & Position(\text{Rich\_Beem}, 1)\} \,|\!\rangle \end{aligned}$$

The ontology has four predicate symbols called *Person*, *Golfer*, *Score* and *Position*; the first axiom asserts that every *Golfer* is a *Person*; the second one states that every *Golfer* has a *Score*, where y represents the total number of points obtained; the last one asserts that every *Golfer* has a *Position* y in the championship. The instances in KB_0 can be interpreted using the ontology, and they asserts that *Rich Beem* is a golfer, and he is the first in the ranking with 278 points.

4 Our Proposal

Our proposal is a framework agent developers can use to extract information with semantics from non–annotated web pages, so that this procedure can be clearly separated from the rest in an attempt to reduce development costs and improve maintainability. This frameworks gives the mechanisms to develop core web agents called knowledge channels. Figure 2 illustrate this idea.

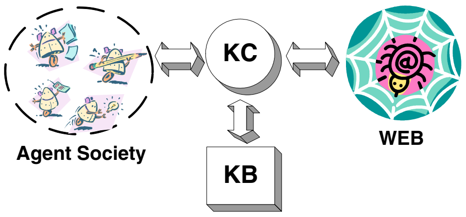


Fig. 2. Knowledge Channels.

A KC is responsible for managing a local knowledge base (KB). This knowledge is extracted from a web site using *semantic wrappers*. KCs answer also inquiries from other agents that need some knowledge to accomplish their goals.

4.1 Knowledge Extraction

A semantic wrapper is an extension to current syntactic wrappers, as shown in Figure 3. Thus, we first need to define such wrappers formally.

Definition 4. *A* ***syntactic wrapper*** *is a function that takes a web page as input, and returns structured information.*

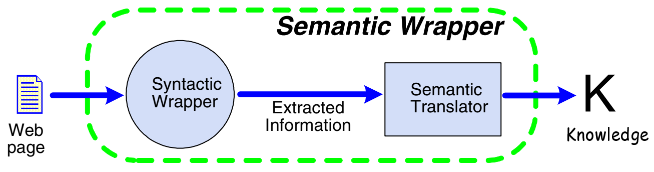


Fig. 3. A semantic wrapper.

Next schema specifies a syntactic wrapper:

$$[\mathit{String}, \mathit{WebPage}]$$
$$\mathit{Datum} == \mathbb{P}\,\mathit{String}$$
$$\mathit{Data} == \mathrm{seq}\,\mathit{Datum}$$
$$\mathit{Information} == \mathbb{P}\,\mathit{Data}$$

$$\mathit{Wrapper} : \mathit{WebPage} \nrightarrow \mathit{Information}$$
$$\mathrm{dom}\,\mathit{Wrapper} \neq \varnothing$$

A syntactic wrapper is modelled as a partial function because its domain is a subset of web pages. This subset defines the scope of the wrapper, and it references the web pages in which the wrapper can be used. The output is modelled as data type *Information*, which is a set of data type *Data*. *Data* is sequence of *Datum*, it allows us to have a structured vision of the data to be extracted and to set a location for each datum. Data type *Datum* represents facts, and it is specified as a set of strings; this allows us to deal with multi–valuated attributes (attributes that can have 0 or more values).

Example 2. If we were interested in extracting information about the position and score of golfers in a PGA championship, a syntactic wrapper would output the following *Information* from the web page in Figure 1:

$$\{\langle\{\text{Rich\_Beem}\}, \{278\}, \{1\}\rangle, \langle\{\text{Tiger\_Woods}\}, \{279\}, \{2\}\rangle,$$
$$\langle\{\text{Chris\_Riley}\}, \{283\}, \{3\}\rangle, \ldots\}$$

Definition 5. *A* ***semantic wrapper*** *is a function that takes a web page as input, and returns a set of instances of concepts defined in an ontology that represents the information of interest.*

A semantic wrapper is composed of a syntactic wrapper and a semantic translator. In order to extract knowledge from the web, it is necessary to feed the semantic wrapper with the web page that contains the information. The syntactic wrapper extracts the structured information from that web page, and the semantic translator assigns then meaning to it by means of an ontology.

$$\mathit{SemanticWrapper} : \mathit{WebPage} \nrightarrow \mathbb{P}\,\mathit{Wff}$$
$$\forall\, p : \mathit{WebPage} \mid p \in \mathrm{dom}\,\mathit{Wrapper} \bullet \mathit{SemanticWrapper}(p) = \mathit{SemanticTranslator}(\mathit{Wrapper}(p))$$

The semantic translator needs the user to specify a *semantic description* that relates the information to be extracted with the predicates defined in the ontology to perform this task.

Definition 6. *A **semantic description** (SD) is a representation of the relationships that hold amongst the symbols of predicates from an ontology and the positions that their arguments occupy in an Information structure. Thus, each predicate P is associated with n natural numbers, where n is the arity of P.*

An SD is modelled using the following schema, which is composed of three elements: an ontology (O), a set of predicate symbols (${S_p}^3$) and a function (*Pos*) that maps predicate symbols onto the location of *Datum* in *Data* belonging to the *Information* structure. This scheme also asserts that S_p is a subset of the set of predicates symbols in O, and the domain of *Pos* is a subset of the symbols in S_p.

$$\begin{array}{|l}
\hline \textit{SemanticDescription} \\
O : \textit{Ontology} \\
S_p : \mathbb{P}\, \textit{Ident}_p \\
\textit{Pos} : \textit{Ident}_p \nrightarrow \mathrm{seq}_1\, \mathbb{N} \\
\hline
S_p \subseteq O.P \wedge \mathrm{dom}\, \textit{Pos} = S_p \\
\hline
\end{array}$$

Example 3. In our study case, we can define the following semantic description:

$$\begin{array}{l}
\langle\!| \; O \leadsto o_0, S_p \leadsto \{\textit{Golfer}, \textit{Score}, \textit{Position}\}, \\
\quad \textit{Pos} \leadsto \{\textit{Golfer} \mapsto \langle 1 \rangle, \textit{Score} \mapsto \langle 1, 2 \rangle, \textit{Position} \mapsto \langle 1, 3 \rangle\} \; |\!\rangle
\end{array}$$

In this SD, predicate *Golfer* takes constant values from location *Pos*(*Golfer*) of each *Data* (sequence) in an *Information* structure, In this case, the first position of the sequence. Predicate *Score* takes its values from $\textit{Pos}(\textit{Score}) = \langle 1, 2 \rangle$[4], and so on. Thus, it is posible to generate automatically well-formed formula that express the meaning of the information for all the *Data* elements in an *Information* structure extracted.

Definition 7. *A **semantic translator** is a function that receives the Information structure obtained using a syntactic wrapper as input and uses a semantic description specified by the user, and outputs a set of instances.*

$$\begin{array}{|l}
\textit{SemanticTranslator} : \textit{Information} \nrightarrow \mathbb{P}\, \textit{Wff} \\
\textit{sd} : \textit{SemanticDescription} \\
\hline
\forall\, i : \textit{Information} \mid i \in \mathrm{ran}\, \textit{Wrapper} \bullet \\
\qquad \textit{SemanticTranslator}(i) = \bigcup\{d : \textit{Data} \mid d \in i \bullet \textit{buildWffs}(d)\}
\end{array}$$

[3] We might not need to use all the predicate defined in the ontology to give meaning to the information extracted.

[4] The arguments in a predicate follows a strict order. Using a sequence allows us to get arguments orderly. For instance, If *Pos*(*Score*) were $\langle 2, 1 \rangle$, the result would be erroneous: *Score*(278, Tiger␣Woods) states that the score of 278 is Tiger␣Woods.

Function *buildWffs* returns the set of well formed formula for each *data* in an *Information* structure. It is defined as follows[5]:

$$
\begin{array}{|l}
buildWffs : Data \nrightarrow \mathbb{P}\ Wff \\
\hline
\forall e : Data;\ t : \mathbb{P}\ (Ident_p \times Data) \mid e \in \bigcup ran Wrapper \land \\
\qquad\qquad t = \{x : sd.S_p \bullet (x, e \upharpoonright \{n : \operatorname{ran} Pos(x) \bullet e(n)\})\} \bullet \\
\qquad buildWffs(e) = \bigcup\{k : t \bullet BuildPredicates(k)\}
\end{array}
$$

The function *BuildPredicates* is specified as follows:

$$
\begin{array}{|l}
BuildPredicates : Ident_p \times \operatorname{seq} \mathbb{P}\ Ident_c \rightarrow \mathbb{P}\ Wff \\
\hline
\forall ip : Ident_p;\ ssc : \operatorname{seq} \mathbb{P}\ Ident_c \bullet \\
\qquad BuildPredicates(ip, ssc) = \{si : \operatorname{seq} Ident_c;\ n : \mathbb{N} \mid \\
\qquad\qquad n \in 1..\#ssc \land si(n) \in ssc(n) \bullet atom(pred(ip, si))\}
\end{array}
$$

It takes a pair composed of an identifier of predicate and a sequence of strings sets from an *Information* structure, and returns a set of predicates. The predicates are composed using the identifier of predicate and each element of the sequence.

Example 4. The following instances represent the knowledge extracted by a semantic wrapper from the web page in Figure 1:

$$
\begin{array}{l}
\{atom(pred(Golfer, \langle const(\text{Rich\_Beem})\rangle)), \\
\ atom(pred(Score, \langle const(\text{Rich\_Beem}), const(278)\rangle)), \\
\ atom(pred(Position, \langle const(\text{Rich\_Beem}), const(1)\rangle)), \\
\ atom(pred(Golfer, \langle const(\text{Tiger\_Woods})\rangle)), \\
\ atom(pred(Score, \langle const(\text{Tiger\_Woods}), const(279)\rangle)), \\
\ atom(pred(Position, \langle const(\text{Tiger\_Woods}), const(2)\rangle)), \\
\ atom(pred(Golfer, \langle const(\text{Chris\_Riley})\rangle)), \\
\ atom(pred(Score, \langle const(\text{Chris\_Riley}), const(283)\rangle)), \\
\ atom(pred(Position, \langle const(\text{Chris\_Riley}), const(3)\rangle)), \ldots\}
\end{array}
$$

4.2 A Model for KCs

The schema bellow formalises a KC. It has a declarative part containing two variables; the former (SW) references the semantic wrapper to be used, and the latter (SV) the semantic verifier.

$$
\begin{array}{|l}
\hline
\textit{KnowledgeChannel} \\
SW : SemanticWrapper \\
SV : SemanticVerificator \\
\hline
\end{array}
$$

[5] The filtering operator ($\upharpoonright$) takes from a sequence the elements in a set. For instance:

$$\langle jun, nov, feb, jul\rangle \upharpoonright \{sep, oct, nov, dec, jan, feb, mar, apr\} = \langle nov, feb\rangle$$

Next schema defines the overall state of our system. It is composed of a knowledge channel, a local KB and its environment. The environment (the perceivable features of the KC agent) is specified as set of web pages, and we constraint that the semantic wrapper must be defined for these web pages.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{KnowledgeChannelState}} \\
\hline
\textit{KC} : \textit{KnowledgeChannel} \\
\textit{KB} : \textit{KnowledgeBase} \\
\textit{Environment} : \mathbb{P}\ \textit{WebPage} \\
\hline
\textit{Environment} \subseteq \operatorname{dom} \textit{KC}.\textit{SW} \\
\hline
\end{array}$$

The motivation of the knowledge channel is to synchronize the knowledge that resides in web documents with the one in the knowledge base (KB). This motivation allows us define a KC as an autonomous piece of software. Next schema specifies an agent's motivation. Δ means that the system's state can change and the imposed constrains indicates that all knowledge on environment must be on local KB, and vice versa.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{KnowledgeChannelMotivation}} \\
\hline
\Delta \textit{KnowledgeChannelState} \\
\hline
\forall\, p : \textit{Environment} \bullet \textit{KB}.K \subseteq \textit{KC}.\textit{SW}(p) \\
\forall\, f : \textit{Wff} \mid f \in \textit{KB}.K \bullet \exists\, p : \textit{Environment} \bullet f \in \textit{SW}(p) \\
\hline
\end{array}$$

The KC server requests from agents, thus they show social ability. Delegating the task of knowledge extraction to KCs allows agent developers to achieve a complete separation between the knowledge extraction procedure and the logic or base functionality an agent encapsulates. Any agent can send messages to a KC in order to extract knowledge. Knowledge requests are expressed as predicate symbols. The reply from the KC are ground predicate atoms that satisfies the predicates in query.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{KnowledgeChannelQuery}} \\
\hline
\Xi \textit{KnowledgeChannelState} \\
Q? : \mathbb{P}\ \textit{Ident}_p \\
R! : \mathbb{P}\ \textit{Wff} \\
\hline
Q \neq \varnothing \\
R = \{f : \textit{KB}.K;\ \textit{ip} : \textit{Ident}_p;\ \textit{sc} : \operatorname{seq}_1 \textit{Term};\ \mid f = \textit{atom}(\textit{pred}(\textit{ip}, \textit{sc}) \bullet f\} \\
\hline
\end{array}$$

Example 5. If we launch the query $Q = \{\textit{Golfer}\}$ the KC would reply with $R = \{\textit{Golfer}(\text{Rich\_Beem}), \textit{Golfer}(\text{Tiger \_Woods}), \textit{Golfer}(\text{Chris\_Riley}), \ldots\}$. This knowledge can be used to infer new knowledge. It also makes it possible to reuse knowledge. For instance, an agent using the golfer ontology can infer that the golfers *Rich Beem* and *Tiger Woods* are people, according to the axiom $\forall\, x \bullet \textit{Golfer}(x) \Rightarrow \textit{Person}(x)$, these knowledge can be shared by applications in order to collaborate to accomplish a task.

5 Conclusions

The current web is mostly user–oriented. The semantic web shall help extract information with well–defined semantics, regardless of the way it is rendered, but it does not seem it is going to be adopted in the immediate future, which argues for another solution to the problem in the meanwhile.

In this article, we have presented a new approach to knowledge extraction from web sites based on semantic wrappers. In this article, we have presented a framework that is based on specialised knowledge channels agents that extract information from the web. It improves on other proposals in that it associates semantics with the extracted information. Furthermore, our proposal achieves a separation of the knowledge extraction procedure from the base logic that web agents encapsulate, thus easing both development and maintenance.

References

1. ISO/IEC 13568:2002. Information technology—Z formal specification notation—syntax, type system and semantics. International Standard.
2. J. L. Arjona, R. Corchuelo, A. Ruiz, and M. Toro. A practical agent-based method to extract semantic information from the web. In *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002*, volume 2348 of *Lecture Notes in Computer Science*, pages 697–700. Springer, 2002.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, May 2001.
4. T.J. Berners-Lee, R. Cailliau, and J.-F. Groff. The World-Wide Web. *Computer Networks and ISDN Systems*, 25(4–5):454–459, November 1992.
5. D. Brickley and R.V. Guha. Resource description framework schema specification 1.0. Technical Report http://www.w3.org/TR/2000/CR-rdf-schema-20000327, W3C Consortium, March 2000.
6. W.W. Cohen and L.S. Jensen. A structured wrapper induction system for extracting information from semi-structured documents. In *Proceedings of the Workshop on Adaptive Text Extraction and Mining (IJCAI'01)*, 2001.
7. O. Corcho and A. Gómez-Pérez. A road map on ontology specification languages. In *Proceedings of the Workshop on Applications of Ontologies and Problem Solving Methods. 14th European Conference on Artificial Intelligence (ECAI'00)*, pages 80–96, 2000.
8. L. Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.
9. D. Fensel, editor. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* The MIT Press, 2002.
10. Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.
11. N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy*, pages 3–15. IOS Press, June 1998.
12. J. Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment.* PhD thesis, University of Maryland, College Park, 2001.

13. I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. Technical Report http://www.daml.org, Defense Advanced Research Projects Agency, 2002.
14. C.A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000.
15. N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.
16. Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.
17. S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In W.L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 59–68, Marina del Rey, CA, USA, 1997. ACM Press.
18. M. Minsky. *A framework for representing knowledge.* McGraw-Hill, New York, 1975.
19. I. Muslea, S. Minton, and C. Knoblock. STALKER: Learning extraction rules for semistructured, web–based information sources. In *Proceedings of the AAAI-98 Workshop on AI and Information Integration*, 1998.
20. M. R. Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.
21. M.J. Wooldridge and M.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

A Well-Formed Formula

Let $Ident_c$, $Ident_v$, $Ident_f$, $Ident_p$ be the sets of identifiers of constants, variables, functions and predicates, respectively, in a first–order logical language. Then the complete language can be specified as the $\mathcal{Z}$ free type *Formula* in the following way:

$$
\begin{array}{l}
[Ident_c, Ident_v, Ident_f, Ident_p] \\
Term ::= const\langle\langle Ident_c\rangle\rangle \\
\qquad\qquad \mid var\langle\langle Ident_v\rangle\rangle \\
\qquad\qquad \mid func\langle\langle Ident_f \times \mathrm{seq}_1\ Term\rangle\rangle \\
Atom ::= pred\langle\langle Ident_p \times \mathrm{seq}_1\ Term\rangle\rangle \\
\qquad\qquad \mid not\langle\langle Atom\rangle\rangle \\
\qquad\qquad \mid and\langle\langle Atom \times Atom\rangle\rangle \\
\qquad\qquad \mid or\langle\langle Atom \times Atom\rangle\rangle \\
\qquad\qquad \mid implies\langle\langle Atom \times Atom\rangle\rangle \\
\qquad\qquad \mid iff\langle\langle Atom \times Atom\rangle\rangle \\
Formula ::= atom\langle\langle Atom\rangle\rangle \\
\qquad\qquad\quad \mid forall\langle\langle Ident_v \times Formula\rangle\rangle \\
\qquad\qquad\quad \mid exists\langle\langle Ident_v \times Formula\rangle\rangle
\end{array}
$$

This states that a *Formula* is either an *atom* or an universal quantifier over a formula or an existencial quantifier over a formula. An *atom* is either an n-ary predicate or the negation of an atom or the conjunction of two atoms or the disjunction of two atoms or the implication formed from two atoms or the bi–implication formed from two atoms. A term is an identifier of constant or an identifier of variable or a n-ary function. To illustrate the use of this free type, formula $\forall\, P(x) \Rightarrow Q(x)$ is represented by the following term:

$$forall(x, atom(implies(pred(P, \langle var(x)\rangle), pred(Q, \langle var(x)\rangle))))$$

A well–formed formula (Wff) is a formula that does not contain any *free* variables, that is, its variables are *bounded* by universal or existencial quantifiers. In order to define the set of the well–formed formula in a logical language, we need to specify axiomatically a recursive function called *FreeVars*. It obtains the free variables in a formula or atom or term.

$$FormulaAtomTerm ::= Formula \mid Atom \mid Term$$

$$
\begin{array}{|l}
FreeVars : FormulaAtomTerm \to \mathbb{P}\, Ident_v \\
\hline
\forall f : Formula;\ a, a1, a2 : Atom;\ iv : Ident_v;\ ip : Ident_p;\ if : Ident_f \bullet \\
\quad FreeVars(atom(a)) = FreeVars(a) \land \\
\quad FreeVars(forall(iv, f)) = FreeVars(f) \setminus \{iv\} \land \\
\quad FreeVars(exists(iv, f)) = FreeVars(f) \setminus \{iv\} \land \\
\quad FreeVars(not(a)) = FreeVars(a) \land \\
\quad FreeVars(and(a1, a2)) = FreeVars(a1) \cup FreeVars(a2) \land \\
\quad FreeVars(or(a1, a2)) = FreeVars(a1) \cup FreeVars(a2) \land \\
\quad FreeVars(implies(a1, a2)) = FreeVars(a1) \cup FreeVars(a2) \land \\
\quad FreeVars(iff(a1, a2)) = FreeVars(a1) \cup FreeVars(a2) \land \\
\quad FreeVars(pred(ip, st)) = \bigcup\{t : Term \mid t \in st \bullet FreeVars(t)\} \land \\
\quad FreeVars(var(iv)) = \{iv\} \land \\
\quad FreeVars(const(c)) = \varnothing \land \\
\quad FreeVars(func(if, st)) = \bigcup\{t : Term \mid t \in st \bullet FreeVars(t)\}
\end{array}
$$

Set *Wff* is specified as the set of logical formula that does not have any free variables.

$$
\begin{array}{|l}
Wff : \mathbb{P}\, Formula \\
\hline
\forall f : Formula \bullet f \in Wff \Leftrightarrow Freevar(f) = \varnothing
\end{array}
$$

We can also obtain the set of predicate symbols in a formula:

$$FormulaAtom ::= Formula \mid Atom$$

$$
\begin{array}{|l}
PredSyms : FormulaAtom \to \mathbb{P}\, Ident_p \\
\hline
\forall f : Formula;\ a, a1, a2 : Atom;\ iv : Ident_v;\ ip : Ident_p \bullet \\
\quad PredSyms(formula(a)) = PredSyms(a) \land \\
\quad PredSyms(forall(iv, f)) = PredSyms(f) \land \\
\quad PredSyms(exists(iv, f)) = PredSyms(f) \land \\
\quad PredSyms(not(a)) = PredSyms(a) \land \\
\quad PredSyms(and(a1, a2)) = PredSyms(a1) \cup PredSyms(a2) \land \\
\quad PredSyms(or(a1, a2)) = PredSyms(a1) \cup PredSyms(a2) \land \\
\quad PredSyms(implies(a1, a2)) = PredSyms(a1) \cup PredSyms(a2) \land \\
\quad PredSyms(iff(a1, a2)) = PredSyms(a1) \cup PredSyms(a2) \land \\
\quad PredSyms(pred(ip, st)) = \{ip\}
\end{array}
$$