A Knowledge Extraction Process Specification for Today's non-semantic Web[*]

J.L. Arjona, R.Corchuelo and M. Toro
Departamento de Lenguajes y Sistemas Informáticos
Avda. de la Reina Mercedes s/n. Sevilla, E-41012 (Spain)
{arjona, corchuelo, mtoro}@lsi.us.es

Abstract

The semantic web shall enable web agents an efficient, precise, and comprehensive extraction of knowledge. Nevertheless, this new web is not likely to be adopted in the immediate future.

In this article, we present a specification of a new framework in order to extract knowledge from today's dynamics non-semantic web. Our proposal is novel in that it associates semantics with the information extracted, which improves agent interoperability; it can also deal with changes to the structure of a web page, which improves adaptability; furthermore, it achieves to delegate the knowledge extraction procedure to specialist agents, easing software development and promoting software reuse and maintainability.

1. Introduction

In recent years, the web has consolidated as one of the most important knowledge repositories. A major challenge for web agents has become sifting through an unwieldy amount of data to extract meaningful information. This process is difficult because of the following reasons: first, the information on the web is mostly available in human-readable forms that lack formalised semantics that would help agents use it [3]; second, the information sources are likely to change their structure, which usually has an impact on their presentation but not on their semantics [4, 15].

The Semantic Web 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 [10], SHOE [16] or RDF-Schema [5]. The Semantic Web shall simplify and improve 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 [8].

In this article, we present a specification of a new solution in order to extract semantically-meaningful information from today's non-semantic web. We address this issue by developing *knowledge channels*, or KCs for short. They are agents [20] 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.

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 initials 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 [6, 11, 14, 18] are one of the the most popular mechanisms for extracting information from the web. They are algorithms that use a number of extraction rules generated by means of automated learning techniques such as inductive logic programming, statistical methods, and inductive grammars to extract information from similar pages automatically.

Although induction wrappers are suited to extract information from the web, they do not associate semantics

[*]The work reported in this article was supported by the Spanish Interministerial Commission on Science and Technology under grants PCB-02-001 and TIC2000-1106-C02-01.

with the data extracted, this being their major drawback [1]. Thus, we call current inductive wrappers syntactic because they extract syntactic information devoid of semantic formalisation that expresses its meaning.

Another problem appears when we work with wrappers, the problem is due to the dynamism embodied in current web sites. Wrappers are based on some properties of a web page or in characteristics of the information to extract. Changes in a web page can invalidate a wrapper. Therefore, it is necessary to have mechanisms (verification algorithms) that detect if the information extracted is valid, making it possible that the extraction rules could be regenerated if there are changes in the layout of a web page [12, 18].

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.

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.

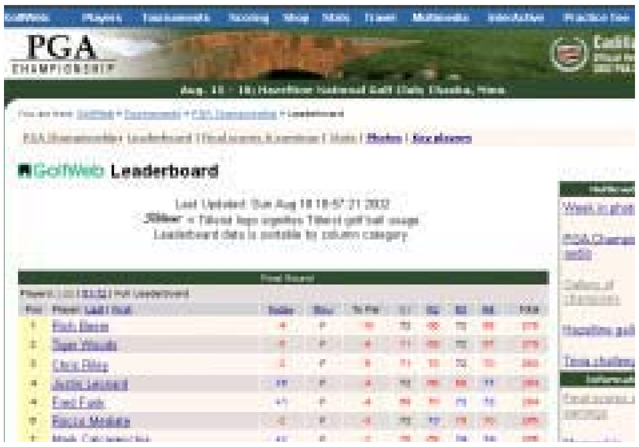

Figure 1. A web page with information about scores in a golf championship.

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 [2, 3]. Other issue is that the layout and the aspect of a web page may change unexpectedly, which may invalidate unexpectedly the automatic extraction methods used so far [4, 15].

3.2. Dealing with knowledge

There are many formalisms to dealt with knowledge, namely: semantic networks [19], frames systems [17], 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 have specified a formal model for ontologies; our formalisation builds on the work by Heflin in his PhD dissertation [9].

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 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.

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 formula (Wff), which is a subset of the formula 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 formula 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, thus f belongs to the set of all Well–formed formula that we can obtain from F (theory of F).

Ontology

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

An **instance** of a concept, specified in an ontology, is an interpretation of this concept over some domain. In informa-

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

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

tion 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*:

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

A **Knowledge Base** (KB) is a tuple (O, K), where O is an ontology and K a set of instances of concepts specified in $\mathcal{O}$. A KB is specified as follows:

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

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

Example 1 *The following object defines a KB in our study case (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):*

$$
\begin{array}{l}
KB_0 = \langle\!| \; O \rightsquigarrow \; \langle\!| \; P \rightsquigarrow \{\mathit{Person}, \mathit{Golfer}, \mathit{Score}, \mathit{Position}\}, \\
\qquad\qquad\qquad A \rightsquigarrow \{\forall\, x \bullet \mathit{Golfer}(x) \Rightarrow \mathit{Person}(x), \\
\qquad\qquad\qquad\quad \forall\, x \bullet \exists\, y \bullet \mathit{Golfer}(x) \Rightarrow \mathit{Score}(x, y), \\
\qquad\qquad\qquad\quad \forall\, x \bullet \exists\, y \bullet \mathit{Golfer}(x) \Rightarrow \mathit{Position}(x, y)\} \; |\!\rangle, \\
\qquad\quad K \rightsquigarrow \{\mathit{Golfer}(\mathit{Rich\_Beem}), \mathit{Score}(\mathit{Rich\_Beem}, 278), \\
\qquad\qquad\quad \mathit{Position}(\mathit{Rich\_Beem}, 1)\} \; |\!\rangle
\end{array}
$$

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, changing 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.

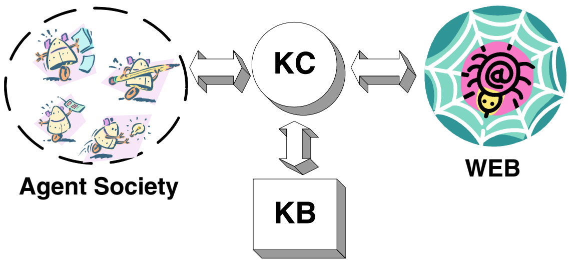

Figure 2. Knowledge Channels.

A KC is responsible for managing a local knowledge base (KB). This knowledge is extracted from a web site using semantic wrappers. Before storing the knowledge in the KB, it is verified using semantic verification to check the existing relations amongst the different concepts that give semantics to the information extracted. Thus, a KC can answer inquiries from other agents that need some knowledge to accomplish its goals.

4.1. Knowledge extraction

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

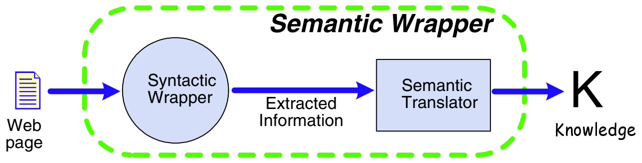

Figure 3. A semantic wrapper.

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

Next schema specifies a syntactic wrapper:

$$
\begin{array}{l}
[\mathit{String}, \mathit{WebPage}] \\
\mathit{Datum} == \mathbb{P}\,\mathit{String} \\
\mathit{Data} == \mathrm{seq}\,\mathit{Datum} \\
\mathit{Information} == \mathbb{P}\,\mathit{Data}
\end{array}
\qquad
\begin{array}{|l}
\mathit{Wrapper} : \\
\quad \mathit{WebPage} \nrightarrow \mathit{Information} \\
\hline
\mathrm{dom}\,\mathit{Wrapper} \neq \varnothing
\end{array}
$$

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 take 0 or more values).

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

$$\{\langle\{\text{Rich\_Beem}\},\{278\},\{1\}\rangle,\langle\{\text{Tiger\_Woods}\},\{279\},\{2\}\rangle,$$
$$\langle\{\text{Chris\_Riley}\},\{283\},\{3\}\rangle,\langle\{\text{Justin\_Leonard}\},\{284\},\{4\}\rangle,$$
$$\ldots\}$$

Using syntactic wrappers allows us to apply syntactic verifiers to the information extracted[3]. They are algorithms that aims at decide if the wrapper works correctly, or on the contrary, is it invalid because of changes in the web page structure. For instance, the algorithm `Rapture` [13] defined by Kushmerick uses statistical features, such as length, number of words, number of special characters etc. to characterize the extracted data. It learns the parameters of normal distributions describing the feature distributions of the extracted data. This information helps to decide if the wrapper is valid by means of analysing the statistical values of the information extracted.

The syntactic wrapper specification can be extended to deal with changes in web. The idea is to use a syntactic verifier predicate on the information extracted, that assure that syntactic wrapper works correctly. This objective is achieved by the wrapper maintenance task, that consist on reconstructing the wrapper if the syntactic verifier predicate does not satisfy.

$$\begin{array}{|l}
\mathit{Wrapper} : \mathit{WebPage} \pfun \mathit{Information} \\
\lambda : \Lambda \\
\hline
\operatorname{dom} \mathit{Wrapper} \neq \varnothing \\
\forall w : \mathit{WebPage} \mid w \in \operatorname{dom} \mathit{Wrapper} \bullet \\
\quad \mathit{SyntacticVerification}(\lambda, \mathit{Wrapper}(w))
\end{array}$$

The Λ type references to some parameters and if we give a concrete definition for Λ, we would have a syntactic checker of wrappers. In our experiments we use the `Rapture` syntactic verifier, and Λ is defined as a set of real number that stores normal distributions of some measurements (html tag density, that is number of $\langle$ and $\rangle$ symbols; also the length and number of words are measured) of the data extracted. If the wrapper does not extract the information correctly we need to train the wrapper again with new sample data.

[3]We call them syntactic verifiers because the decision about the wrapper validity is only based on syntactic properties of the extracted information.

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. It 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.

$$\begin{array}{|l}
\mathit{SemanticWrapper} : \mathit{WebPage} \pfun \mathbb{P}\, \mathit{Wff} \\
\hline
\forall p : \mathit{WebPage} \mid p \in \operatorname{dom} \mathit{Wrapper} \bullet \\
\mathit{SemanticWrapper}(p) = \mathit{SemanticTranslator}(\mathit{Wrapper}(p))
\end{array}$$

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.

A **semantic description** (SD) is a representation of the relationships amongst the symbols of predicates from an ontology and the positions that their arguments occupy in the *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[4]) 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.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{SemanticDescription}} \\
\hline
O : \mathit{Ontology} \\
S_p : \mathbb{P}\, \mathit{Ident}_p \\
\mathit{Pos} : \mathit{Ident}_p \pfun \operatorname{seq}_1 \mathbb{N} \\
\hline
S_p \subseteq O.P \\
\operatorname{dom} \mathit{Pos} = S_p \\
\hline
\end{array}$$

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

$$\langle\!| \, O \leadsto o_0, S_p \leadsto \{\mathit{Golfer}, \mathit{Score}, \mathit{Position}\},$$
$$\mathit{Pos} \leadsto \{\mathit{Golfer} \mapsto \langle 1 \rangle, \mathit{Score} \mapsto \langle 1, 2 \rangle, \mathit{Position} \mapsto \langle 1, 3 \rangle\} \, |\!\rangle$$

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 $Pos(Score) = \langle 1, 2 \rangle$[5],

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

[5]The arguments in a predicate follows a strictly 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*.

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.

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.

$$\begin{array}{|l}
sd : SemanticDescription \\
SemanticTranslator : Information \rightarrow\!\!\!\!\!\!+\; \mathbb{P}\, Wff \\
\hline
\forall\, i : Information \mid i \in \operatorname{ran} Wrapper \bullet \\
\quad SemanticTranslator(i) = \\
\qquad \bigcup\{d : Data \mid d \in i \bullet buildWffs(d)\}
\end{array}$$

Function *buildWffs* returns the set of well formed formula for each *data* in an *Information* structure. It is defined as follows[6]:

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

The function *BuildPredicates* is specified as follows:

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

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

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

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

[6]The filtering operator ($\upharpoonright$) takes from a sequence the elements in a set. For instance: $winter = \{sep, oct, nov, dec, jan, feb, mar, apr\}$, then $\langle jun, nov, feb, jul\rangle \upharpoonright winter = \langle nov, feb\rangle$

4.2. Semantic Verification

The semantic verification allows us to check the existing relations amongst the different concepts that endow the information extracted with semantics. Thus, we can be aware of non-syntactic errors of the information provided by a web site. Semantic verification detects inconsistences in the information source used to feed a web site, whereas syntactic verification detects changes in the layout of a web site that invalidates the extraction process. Then, the solution to semantic errors is not to rebuild the syntactic wrapper since it works well, but to wait for the information to be corrected or to look for another site that offers the same information.

The semantic verification function is specified as follows:

$$\begin{array}{|l}
\forall\, k : KB \bullet SemanticVerification(k) \Leftrightarrow \\
\quad (\neg \exists f : Wff \mid f \in k.K \bullet \{k.O.A, k.K\} \vdash f \land \\
\quad \{k.O.A, k.K\} \vdash \neg f)
\end{array}$$

Semantic verification is defined as a predicate. It is satisfied if the knowledge extracted is consistent with the axioms in the ontology under consideration. That is, it does not exist a formula f in the extracted knowledge, such that f and $\neg f$ belong to the theory obtained from the axioms in ontology and the formula that represent the instances.

4.3. A model for KCs

The schema bellow formalises an 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.

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

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.

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

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 the agent 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.

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

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 or an empty set of formula if the KB is erroneous.

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

If we launch the query $Q = \{\textit{Golfer}\}$ the KC would reply with $R = \{\textit{Golfer}(\textit{Rich\_Beem})\ , \textit{Golfer}(\textit{Tiger\_Woods}), \textit{Golfer}(\textit{Chris\_Riley}), \ldots\}$. This knowledge can be used 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 specialised knowledge channels agents. It improves on other proposals in that it associates semantics with the extracted information, and can also deal with changes because the information is extracted by means of current wrappers. Furthermore, our proposal achieves a separation of the knowledge extraction procedure from the base logic that web agents encapsulate, thus easing the design and implementation of clean, reusable, understandable agents.

References

[1] 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.

[2] T. Berners-Lee, R. Cailliau, and J.-F. Groff. The World-Wide Web. *Computer Networks and ISDN Systems*, 25(4–5):454–459, Nov. 1992.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[4] B. Brewington and G. Cybenko. Keeping up with the changing web. *Computer*, 33(5):52–58, May 2000.

[5] D. Brickley and R. Guha. Resource description framework schema specification 1.0. Technical Report http://www.w3.org/TR/2000/CR-rdf-schema-20000327, W3C Consortium, Mar. 2000.

[6] W. Cohen and L. 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-2001)*, 2001.

[7] O. Corcho and A. Gómez-Pérez. A road map on ontology specification languages. In *Workshop on Applications of Ontologies and Problem solving methods. 14th European Conference on Artificial Intelligence (ECAI'00)*, 2000.

[8] D. Fensel, editor. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press, 2002.

[9] J. Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. PhD thesis, University of Maryland, College Park, 2001.

[10] I. Horrocks, P. Patel-Schneider, and F. 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.

[11] C. 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.

[12] N. Kushmerick. Regression testing for wrapper maintenance. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99); Proceedings of the 11th*

Conference on Innovative Applications of Artificial Intelligence, pages 74–79, Menlo Park, Cal., July 18–22 1999. AAAI/MIT Press.

[13] N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[14] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[15] L. Lim, M. Wang, S. Padmanabhan, J. Vitter, and R. Agarwal. Characterizing web document change. *Lecture Notes in Computer Science*, 2118:133–144, 2001.

[16] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In W. 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.

[17] M. Minsky. *A framework for representing knowledge*. McGraw-Hill, New York, 1975.

[18] 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.

[19] M. R. Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.

[20] M. Wooldridge and M. 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:

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

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.

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

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

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:

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

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