XML based information systems and formal semantics of programming languages

Thierry Despeyroux

Abstract Web sites and XML based information systems must be up to date and coherent. This last quality is difficult to insure because sites can be updated very frequently, may have many authors or be partially generated, and in this context, proof-reading is a real challenge. In this chapter, we make a parallel between programs and Web sites or information systems. Semantic constraints that one would like to specify (constraints between the meaning of categories and sub-categories in a thematic directory, consistency between the organization chart and the rest of the site in an academic site...) are similar to semantic constraints in programs (for example coherence between the use of objects and their declared types). Human knowledge is often represented using ontologies. Ontologies can be seen as type systems. Semantic constraints and types have been heavily studied in the context of the semantics of programming languages. We explore how techniques used in this context can be used to enforce the quality of information systems.

1 Introduction

Starting as an amazing way of exchanging information, the Web is now a federative agent for a large number of information systems to which it provides a uniform man-machine interface. In fact, with the semantic Web, it becomes also a machine-machine interface.

Web sites are now ordinary products, and consumers are aware of the notion of quality of a Web site. For consumers, the quality of a site is linked to its graphic appearance, to the easiness of accessibility and also to other criteria such as the fact that the site is up to date and coherent. This last quality is difficult to insure because sites can be updated very frequently, may have many authors or be partially generated, and in this context, proof-reading is very difficult. The same piece of

Thierry Despeyroux
Inria - Paris-Rocquencourt, France, e-mail: thierry.despeyroux@inria.fr

information may be found at different occurrences in a document, but also in meta-data, sometime by use of ontologies, leading to the need for consistency checking.

The coherence problem becomes even more accurate in the context of the semantic web as an error may be rapidly propagated to other information systems. An ad-hoc way of managing information systems is no more possible. Time is now to Web and information system engineering, using formal techniques as ontologies and formal semantics.

The life cycle of a Web site can be complex: many authors, frequent updates or redesigns. Very often they are heterogeneous objects: some parts are static, other parts may be generated, use databases etc. As for other "industrial" products, one can think of the notion of quality of a Web site, and then look for methods to achieve this quality.

The main effort developed in the domain of the Web is the Semantic Web [3, 5]. Its goal is to ease computer based data mining, formalizing data which is most of the time textual. This leads to two main directions:

- Giving a syntactical structure to documents. This is achieved with XML, DTDs, XML-schema, style sheets and XSLT [40]. The goal is to separate the content of Web pages from their visual appearance, defining an abstract syntax to constrain the information structure. In this area, there is, of course, the work done by the W3C, but we can also mention tools that manipulate XML documents, taking into account the conformance to DTD: Xduce [23] and XM-λ [32]. This means that by using these languages we know that the documents which are produced will conform to the specific DTDs that they use, which is not the case when one uses XSLT.
- Annotating documents to help computerized Web mining. One will use ontologies with the help of RDF [40, 4], RDF-Schema or DAML+OIL [41]. The goal is to get a computer dedicated presentation of knowledge [19, 13, 12] to check the content of a page with the use of ontologies [20] or to improve information retrieval.

This is not sufficient. Web sites, as many other types of information systems, contain naturally redundant information. This is in part due to accessibility reasons. Web pages can be annotated and the same piece of information can exist in many different forms, as data or meta-data. We have to make sure that these different representations of the same piece of knowledge are consistent. Some parts of a Web site can also be generated, from one or more databases, and again, one have to make sure of the consistency between these databases and the rest of the site. With the development of Web based systems one now speaks of Web engineering [6, 37, 33, 21].

Obviously, traditional proof-reading (as it may be done for a book) is not possible for Web sites. A book may contain a structure (chapters, sections, etc.) and references across the text, but its reading can be essentially linear. An information system such as a Web site looks more like a net.

These difficulties lead to the need for a better understanding of the formal properties of Web sites. Many approaches can be used that call for different areas of computer sciences: model checking and linear temporal logic [15], term rewriting techniques [27], rules based approach [1, 2].

We propose to apply techniques from software engineering, in particular from semantics of programming languages and type theory to increase the quality level of Web sites. In the context of Web sites, it is not possible to make proofs as it is the case for some sorts of programs or algorithms, but we will see that some techniques used in the area of formal semantics of programming languages [18] can be successfully used in this context.

With this approach, we are concerned in the way Web sites are constructed, taking into account their development and their semantics. In this respect we are closer to what is called content management.

The work presented here is limited to static Web sites and documents. It can be easily extended to more general information systems as many of them provide a Web interface or at least can generate XML documents, answering the two questions that we try to solve:

- How can we define verification tools for Web sites an more generally information systems?
- How can we mechanize the use of these tools?

In the same way of thinking, we make a parallel between ontologies and types, expecting again to reuse some notions and formal techniques.

The following section presents in an informal manner some basic notions about syntax and semantics of programs. Section 3 makes a parallel between programs and Web sites. Sections 4 presents the genesis of a specification formalism to specify semantic constraints in Web sites. Section 5 presents ontologies as a type systems. Finally section 6 gives some examples of applications and some implementation notes.

2 Syntax and semantics of programming languages

Drawing a line between syntax and semantics is not a trivial task. When giving names to objects in a program or to tags in an XML file, we try to use the syntax to reflect an intentional semantics. We will define syntax and semantics in term of local and global constraints.

2.1 Syntax versus semantics

To execute a program you have, most of the time, to use a compiler which translates the source code to executable code, unless you are the end-user and someone else did this for you.

The goal of a compiler is not only to generate object code but also to verify that the program is legal, i.e., that it follows the *static semantics* of the programming language. For example, in many programming languages one can find some declarative parts in which objects, types and procedures are defined before being used in some other places in statements or expressions. One will have to check that the use of these objects is compatible with the declarations.

The static semantics is defined by opposition to the *dynamic semantics* which describe the way a program is executed. The static semantics give some constraints that must be verified before a program is executed or compiled.

A particularity of such constraints is that they are not local but *global*: they may bind distant occurrences of an object in a unique file or in many different files. A second particularity is that these constraints are *not context-independent*: an “environment” that allows us to know what are the visible objects at a particular point in the program is necessary to perform the verifications.

Global constraints are defined by opposition to local constraints. As a program may be represented by a labeled tree, a local constraint is a relation between a node in this tree and its sons. For example, if we represent an assignment, the fact that the right hand part of an assignment must be an expression is a local constraint. On the other hand, the fact that the two sides of an assignment must have compatible types is a global constraint, as we need to compute the types of both sides using an environment.

Local constraints express what is called the *(abstract) syntax* of the programming language and global constraints express what is called its *static semantics*. The abstract syntax refers to the tree representation of a program, its textual form is called its *concrete syntax*.

Representing programs by means of a tree is quite natural. Using a B.N.F. to describe the syntax of a programming language gives already a parse tree. Most of the time this tree can be simplified to remove meaningless reduction level due to precedence of operators. The grammar rules express local type constraints on the parse tree and explain what is a syntactically correct program.

Some programming languages use trees as based objects, for example Lisp and Prolog. A straight way of representing a program in a language like Prolog is to use (completely instantiated, i.e., with no logical variables) terms, even if Prolog terms are not typed.

The following example shows how a statement can be represented as a Prolog term.

```
A := B + (3.5 * C);

assign(var('A'),
       plus(var('B'),
            mult(num(3.5),var('C'))))
```

A Prolog term can itself be represented into XML as shown in the following example:

```
<assign>
   <var name="A"/>
   <plus>
      <var name="B"/>
      <mult>
         <num value="3.5"/>
         <var name="C"/>
      </mult>
   </plus>
</assign>
```

By defining a DTD for our language, we can constrain XML to allow only programs that respect the syntax of the programming language.

```
<!ELEMENT assign (var, (plus|mult|num...))>
```

This is equivalent to giving a signature to the Prolog terms

```
assign : Var * Exp -> Statement
```

where `Var` and `Exp` are types containing the appropriate elements.

However, using types or DTDs it is not possible to express semantic constraints, as for example “The types of the left and right hand sides must be compatible”. If the variable `A` has been declared as an integer, the statement given as example is not legal.

2.2 Formal semantics

Three main methods are used to formalize the semantics of programming languages : denotational semantics, operational semantics and axiomatic semantics.

We will use here the *Natural Semantics* [7, 24] which is an operational semantics derived from the structural operational semantics of Plotkin [36] and inspired by the sequent calculus of Gentzen [39]. One of the advantages of this semantics is that it is an executable semantics, which means that semantic definitions can be compiled (into Prolog) to generate type-checkers or compilers. Another advantage is that it allows the construction of proofs.

In Natural Semantics, the semantics of programming languages are defined using inference rules and axioms. These rules explain how to demonstrate some properties of “the current point” in the program, named *subject*, using its subcomponents (i.e., subtrees in the tree representation) and an environment (a set of hypothesis).

To illustrate this, the following rule explains how the statements part of a program depends on the declarative one:

$$\frac{\emptyset \vdash Decls \rightarrow \rho \qquad \rho \vdash Stmts}{\vdash \textbf{declare}\ Decls\ \textbf{in}\ Stmts}$$

The declarations are analyzed in an empty environment $\emptyset$, constructing ρ which is a mapping from variable names to declared types. This environment is then used to check the statements part, and we can see that in the selected example the statements part does not alter this environment.

These inference rules can be read in two different ways: if the upper part of the rule has been proved, then we can deduce that the lower part holds; but also in a more operational mode, if we want to prove the lower part we have to prove that the upper part holds.

The following rule is an axiom. It explains the fact that in order to attribute a type to a variable, one has to access the environment.

$$\rho \vdash \mathbf{var}\, X : T \qquad \{X : T\} \in \rho$$

"Executing" a semantic definition means that we want to build a proof tree, piling up semantic rules which have been instantiated with the initial data.

As we have seen, there are two important points here: *declarations* and *environments*. Depending of the programming language, the type system can be more or less strong, allowing multiple declarations (that must be in general compatible) or not. The environment is the link between declarations and locations where objects are used. Managing this environment is of course more easy if locations in which declarations can appear is well restricted or even unique.

3 Xml files viewed as programs

XML files, in opposition to traditional text files, are structured documents as programs are. However they mix structures, natural languages, and references to other objects as images.

3.1 Similarities and differences

Web sites (we define a Web site simply as a set of XML pages) are very similar to programs. In particular, they can be represented as trees, and they may have local constraints expressed by means of DTDs or XML-schemas. As HTML pages can be translated to XHTML, which is XML with a particular DTD, it is not a restriction to focus only to XML web sites.

There are also differences between Web sites and programs:

- Web sites can be spread along a great number of files. This is the case also for programs, but in this case, these files are all located on the same file system. With Web sites we will have to take into account that we may need to access different servers.

- The information is scattered, with a very frequent use of forward references. A forward reference is the fact that an object (or a piece of information) is used before it as been defined or declared. In programs, forward references exist but are most of the time limited to single files so the compiler can compile one file at a time. This is not the case for Web sites and as it is not possible to load a complete site at the same time, we have to use other techniques.
- The syntax can be poorly, not completely or not at all formalized, with some parts using natural languages.
- There is the possibility to use “multimedia”. In the word of programs, there is only one type of information to manipulate: text (with structure), so let’s say terms. If we want to handle a complete site or document we may want to manipulate images for example to compare the content of an image with its caption.
- The formal semantics is not imposed by a particular programming language but must be defined by the author or shared between authors as it is already the case for DTDs. This means that to allow the verification of a Web site along its life one will have to define what should be checked.
- We may need to use external resources to define the static semantics (for example one may need to use a thesaurus, ontologies or an image analysis program). In one of our example bellow, we call the `wget` program to check the validity of URLs in an activity report.

Meta-data are now most of the time expressed using an XML syntax, even if some other concrete syntax can be used (for example the N3 notation [4] can be used as an alternative to the XML syntax of RDF[26]). So, from a syntactic point of view, there is no difference between data and meta-data. Databases also use XML, as a standard interface both for queries and results.

As we can see, the emergence of XML gives us a uniform framework for managing heterogeneous data. Using methods from software engineering will allow a web-master or an author to check the integrity of its information system and to produce error messages or warnings when necessary, provided that they have made the effort to formalize the semantics rules.

Every programmer can relate some anecdote in which somebody forgets to perform an action as recompiling a part of a program, leading to an incoherent system. Formalizing and mechanizing the management of any number of files is the only way to avoid such misadventure.

Again, at least one solution already exists. It is a utility program named “make” [38], well known to at least Unix programmers. This program can manage the dependencies between files and libraries, and it can minimize the actions (such as the calls to the compilers) which must be done when some files have been modified. This means that each time a set of modifications is applied, or at least each time a set of files is installed on a server, the “make” program must be used to check the integrity of the whole system. However, this program can only manage files located on the same file system and must be adapted to handle URLs when a Web site is scattered over multiple physical locations.

3.2 Semantic constraints in XML documents

For a better understanding of the notion of semantic constraints we will now provide two examples.

- A thematic directory is a site in which documents or external sites are classified. An editorial team is in charge of defining the classification. This classification shows as a tree of topics and subtopics. The intentional semantics of this classification is that a subtopic "makes sense" in the context of the upper topic, and this semantics must be maintained when the site is modified.
  To illustrate this, here is an example:

  Category: Recreation
  Sub-Category: Sports, Travel, Games, Surgery, Music, Cinema

  The formal semantics of the thematic directory use the semantics of words in a natural language. To verify the formal semantics, we need to have access to external resources, maybe a thesaurus, a pre-existing ontology or an ontology which is progressively constructed at the same time as the directory; how to access to this external resource is not of real importance, the important point is that it can be mechanized.
  In the former example, the sub-category "Surgery" is obviously a mistake.

- An academic site presents an organization: its structure, its management, its organization chart, etc. Most of the time this information is redundant, maybe even inconsistent. As in a program, one have to identify which part of the site must be trusted: the organization chart or a diary (which is supposed to be up to date) and can be used to verify the information located in other places, in particular when there are modifications in the organization. The "part that can be trusted" can be a formal document such as an ontology, a database or a plain XML document.

The issue of consistency between data and meta-data, or between many redundant data appears in many places, as in the following examples.

- Checking the consistency between an image and its caption; in this case we may want to use linguistic tools to compare the caption with annotations on the image, or to use image recognition tools.
- Comparing different versions of the same site that may exist for accessibility reasons.
- Verifying the consistency between a request to a database, with the result of the request to detect problems in the query and verify the plausibility of an answer (even when a page is generated from a database, it can be useful to perform static verifications, unless the raw data and the generation process can be proved, which is most of the time not the case).
- Verifying that an annotation (maybe in RDF) is still valid when an annotated page is modified.

One characteristic of Web or XML documents is that they contain pieces of natural languages, that is of course much more difficult to manipulate that programming languages. Concepts can be formalized using ontologies, and meta-data (also called semantic annotations) can easily refer to ontologies. However, this is not the case for plain text and to check some consistency between text and annotations some special techniques such as entity extraction coming from the information retrieval and extraction community are useful.

4 A specification language to define the semantics of XML documents and Web sites

In this section we first try to give a list of requirements to justify the design of a specific language dedicated to the definition of semantic constraints in XML documents. This language is also presented and will be used in some examples given in section 6.

4.1 Formalizing Web sites

First experiments have been made using directly Natural Semantics [10, 11]. These experiments showed that this style of formal semantics perfectly fits our needs, but it is very heavy for end-users (authors or web-master) in the context of Web sites. Indeed, as we can see in the former rules, the recursion is explicit, so the specification needs at least one rule for each syntactical operator (i.e., for each XML tag). Furthermore, managing the environment can be tedious, and this is mainly due to the forward declarations, frequent in Web sites (Forward declarations means that objects can be used before they have been defined. This implies that the verifications must be delayed using an appropriate mechanism as coroutines, or using a two-passes process).

It is not reasonable to ask authors or web-masters to write Natural Semantics rules. Even if it seems appropriate for semantic checking, the rules may seem too obscure to most of them. Our strategy now is to specify a simple and specialized specification language that will be compiled in Natural Semantics rules, or more exactly to some Prolog code very close to what is compiled from Natural Semantics [8].

We can make a list of requirements for this specification language:

- No explicit recursion,
- Minimizing the specification to the points of interest only,
- Simple management of the environment,
- Allowing rules composition (points of view management),
- Automatic management of forward declarations.

In a second step we have written various prototypes directly in Prolog. The choice of Prolog comes from the fact that it is the language used for the implementation of Natural Semantics. But it is indeed very well adapted to our needs: terms are the basic objects, there is pattern matching and unification directly available.

After these experiments, we are able to design a specification language. Here are some of the main features of this language:

- Patterns describe occurrences in the XML tree. We have extended the language XML with logical variables. In the following examples variable names begin with the "`$`" sign.
- A local environment contains a mapping from names to values. The execution of some rules may depend on these values. A variable can be read ("=") or assigned ("`:=`").
- A global environment contains predicates which are assertions deduced when rules are executed. The syntax of these predicates is the Prolog syntax, but logical variables are marked with the "`$`" sign. The sign "=>" means that its right hand side must be added to the global environment.
- Tests enable us to generate warnings or error messages when some conditions do not hold. These tests are just now simple Prolog predicates. The expression "`? pred / error`" means that if the predicate `pred` is false, the error message must be issued. The expression "`? pred -> error`" means that if the predicate `pred` is true, the error message must be issued. Messages are general XML expressions. So in fact it is possible to use them to produce new pieces of information that are more results than error.

A semantic rule contains two parts: the first part explains when the rule can be applied, using patterns and tests on the local environment; the second part describes actions that must be executed when the rule applies: modifying the local environment (assignment), adding of a predicate to the global environment, generating a test.

Recursion is implicit. It is also the case for the propagation of the two environments and of error messages.

4.2 Definition of SeXML

In this section, we give a complete description of our specification language, called SeXML.

4.2.1 Patterns

As the domain of our specification language is XML documents, XML elements are the basic data of the language. To allow pattern-matching, logical variables have been added to the XML language (which is quite different from what is done in XSLT).

Variables have a name prefixed by the "`$`" sign, for example : `$X`. Their also exist anonymous variables which can be used when a variable appears only once in a rule : "`$_`". For syntactical reasons, if the variable appears in place of an element, it should be encapsulated between "<" and ">" : `<$X>`.

In a list, there is the traditional problem of determining if a variable matches an element or a sublist. If a list of variables matches a list of elements only the last variable matches a sublist. So in

```
<tag> <$A> <$B> </tag>
```

the variable `$A` matches the first element contained in the body of `<tag>` and `$B` matches the rest of the list (which may be an empty list).

When a pattern contains attributes, the order of the attributes is not significant. The pattern matches elements that contain at least the attributes present in the pattern.

For example, the pattern

```
<citation year=$Y> <$T> <$R> </citation>
```

matches the element

```
<citation type="thesis" year="2003">
   <title> ... </title>
   <author> ... </author>
   ...
   <year> ... <year>
</citation>
```

binding `$Y` with `"2003"` and `$T` with `<title> ... </title>`. The variable `$R` is bound to the rest of the list of elements contained in `<citation>`, i.e., the list of elements

```
   <author> ... </author>
   ...
   <year> ... <year>
```

When this structure is not sufficient to express some configuration (typically when the pattern is too big or the order of some elements is not fixed), one can use the following "contains" predicate:

```
<citation> <$A> </citation>
   &  $A contains  <title> <$T> </title>
```

In this case the element `<title>` is searched everywhere in the subtree `$A`.

4.2.2 Rules

A rule is a couple containing at least a pattern and an action. The execution of a rule may be constrained by some conditions. These conditions are tests on values which have been computed before and are inherited from the context. This is again different from XSLT in which it is possible to get access to values appearing between the top of the tree and the current point. Here these values must explicitly be stored and can be the result of a calculus.

In the following rule, the variable `$A` is bound to the year of publishing found in the `<citation>` element while `$T` is found in the context.

```
<citation year=$Y > <$A> </citation>
   & currentyear = $T
```

The effect of a rule can be to modify the context (the modification is local to the concerned subtree), to assert some predicates (this is global to all the site) and to emit error messages depending on tests.

The following example sets the value of the current year to what is found in the argument of the `<activityreport>` element.

```
<activityreport year=$Y >
   <$A>
</activityreport>
      => currentyear := $Y ;
```

The two following rules illustrate the checking of an untrusted part of a document against a trusted part.

In the trusted part, the members of a team are listed (or declared). The context contains the name of the current team, and for each person we can assert that it is a member of the team.

```
<pers firstname=$F lastname=$L>
   <$_>
</pers>
   & teamname = $P
      => teammember($F,$L,$P) ;
```

In the untrusted part, we want to check that the authors of some documents are declared as members of the current team. If it is not the case, an error message is produced.

```
<author firstname=$F lastname=$L>
   <$_>
</author>
   & teamname = $P
      ? teammember($F,$L,$P)
         / <li> Warning: <$F> <$L> is not
           member of the team
```

```
            <i> <$P> </i>
            </li>;
```

4.3 Dynamic semantics

On each file, the XML tree is visited recursively. A local environment which is initially empty is constructed. On each node, a list of rules which can be applied (the pattern matches the current element and conditions are evaluated to true) is constructed, then applied. This means that all rules which can be applied are evaluated in the same environment. The order in which the rules are then applied is not defined.

During this process, two results are constructed. The fist one is a list of global assertions, the second is a list of tests and related actions. These are saved in files.

A global environment is constructed by collecting all assertions coming from each different environment files. Then using this global environment, tests are performed, producing error messages if there are some.

To produce complete error messages, two predefined variables exist: `$SourceFile` and `$SourceLine`. They contain respectively, the name of the current file which is analyzed and the line number corresponding to the current point.

5 Ontologies as types

An ontology is a way of representing human knowledge in a formal manner. One main goal of an ontology is to be shared between a group of people to fix a terminology and the relations between concepts. Ontologies are heavily used in semantic annotations to ease both human and machine interface and is one of the foundation of the semantic Web [5, 3].

An important word in this definition is ”shared”. It implies the fact that an ontology is a reference [17]. An ontology in the context of an information system can be compared to declarations or libraries in a programming context.

As for web pages, the evolution of ontologies and semantic annotations must be controlled. The process of evolution is studied for example in [30, 28, 31, 29].

The size of ontologies (in particular definitions), leads to the emergence of ontology engineering to allow the management of the creation, modification and development of huge ontologies. Some specialized software as editors are helpful (Protégé for example), and sometime database technology are required [42]. In many aspects, the development of formal systems as ontologies is closed to the development of programs as it is already the case for web-based information systems [37, 33, 8]. Traditional development of ontologies description formalisms refers to the world of logic [13, 22, 16, 34]. In this section we show that ontologies can be seen as type systems. Type systems have been heavily studied in the context of the semantics of programming languages [18]. Thus it should be possible to reuse many results coming from the types community to create more powerful and more secure definition formalisms to describe human knowledge, and this is of course not incompatible with the logical vision. Following P.-H. Luong [31, 28] we take a small ontology and use it by means of semantics annotations, then modifying this ontology we see the effects of this evolution of the ontology on the semantics annotations. In a second step we endorse the suit of a programmer and will implement the same ontology as a type system, applying then the same evolution.

```
Concepts: Person, Trainee, PhdStudent, Manager,
   Researcher, Director, Team, Project;

Researcher is-a Manager;
Director is-a Manager;
Manager is-a Person;
PhdStudent is-a Person:
Trainee is-a Person;

Properties: work, manage;

Person work Team;
Manager manage Project;
```

Let's take a list of semantic annotations, given as RDF triplets:

```
1. (r1 work v1)(r1 type Person)
2. (r2 work v2)(r2 type PhdStudent)
3. (r3 work v3)(r3 type Manager)
4. (r4 manage v4)(r4 type Manager)
5. (r5 work v5)(r5 type Researcher)
6. (r6 manage v6)(r6 type Researcher)
7. (r7 work v7)(r7 type Director)
8. (r8 manage v8)(r8 type Director)
```

In these annotations, `ri` and `vi` are instances of concepts that are inferred from the properties definitions and from type restrictions that are included in the annotations.

In [30], the effects of modifying the ontology on this set of annotations are studied. The original ontology is modified, deleting the concept `Director`, merging `PhdStudent` and `Trainee` into the concept `Student`. The new ontology that is obtained is defined below:

```
Concepts: Person, Student, Researcher, Director,
   Team, Project;

Researcher is-a Person;
```

```
Director is-a Person;
Student is-a Person;

Properties: work, manage;

Person work Team;
Director manage Project;
```

A consequence of the modifications of the ontology is that the annotations 2, 3, 4 and 6 become inconsistent.

We can make a parallel between ontologies and semantic annotations with programs that are also formal systems. Concepts can be viewed as types and subsomptions as type inclusions. In this context, properties get signatures that define their domains and co-domains.

The initial ontology can be described as follows:

```
Person, PhdStudent, Trainne, Manager, Researcher,
   Director, Team, Project : type;

PhdStudent <= Person;
Trainee <= Person;
Manager <= Person;
Researcher <= Manager;
Director <= Manager;

work : Person ->  Team;
manage : Manager -> Projet;
```

The sign <= denotes type inclusion.

The semantic annotations can be seen as expressions in a programming language. Instances are represented by objects (let's say constants). Types of object can be inferred or declared. We prefer to declare them as it is the case in languages with strong typing to optimize type verification and obtain as much error messages as possible.

```
r1 : Person;
r2 : PhdStudent;
r3, r4 : Manager;
r5, r6 : Researcher;
r7, r8 : Director;
v1, v2, v3, v5, v7  : Team;
v4, v6, v8 : Project;
```

If we apply the same modifications to the ontology as previously, the type system looks now as follow

```
Person, Student, Researcher, Director, Team,
   Project : type;

Student <= Person;
Researcher <= Person;
Director <= Person;

work : Person -> Team;
manage : Director -> Project;
```

Applying a traditional type checker to our set of semantic annotations, will produce error messages.

In the following declarations

```
r2 : PhdStudent;
r3, r4 : Manager;
```

the types (concepts) `PhdStudent` and `Manager` are not declared and these declaration are not legal.

In the annotations 2, 3 and 4, `r2`, `r3` and `r4` are not of type `Person` as declared by the signatures of the properties `work` and `manage`. In the annotation 6, `r6` is not of type `Director`.

[25] shows that modifying a part of an ontology can imply some inconsistencies somewhere else in this ontology or when it is used. [28] proposed some rules to detect these inconsistencies. Viewing an ontology as a type system, we can say that checking the consistency of ontologies and annotations can be done by a traditional type-checking

6 Applications

When defining a programming language, one will want to give its complete semantics. This semantics is given to the programmer by means of manuals. This is not possible for a Web site, we will rather try to give only some semantic constraints that we want to check. The designer itself or the authors of the pages have to chose this set of rules, with the possibility to specify some gravity levels as errors and warnings.

Our technologies can also be used to extract and re-compose pieces of information that are spread out in one or more documents, taking benefit of the environment.

This section illustrates these two points. Extraction has been also used with the help of a tagger to extract both textual and structural information in documents to perform different feature selections and classify these documents into clusters [9].

6.1 *Verifying a Web site*

The following example illustrates our definition language. We want to maintain an academic site. The current page must be trusted and contains a presentation of the structure of an organization.

```
<department>
   <deptname><$X></deptname>
   <$_>
</department>
      =>   dept:=$X
```

The left hand side of the rule is a pattern. Each time the pattern is found, the local environment is modified: the value matched by the logical variable `$X` is assigned to the variable `dept`. This value can be retrieved in all the subtrees of the current tree, as in the following example. `$_` matches the rest of what is in the `department` tag.

Notice that, unlike what happens in XSLT in which it is possible to have direct access to data appearing between the root and the current point in a tree, we have to store explicitly useful values in the local environment. In return, one can store (and retrieve) values which are computed and are not part of the original data, and this is not possible with XSLT.

```
<head><$P></head>
   &  dept=$X
      =>   head($P,$X)
```

We are in the context of the department named `$X`, and `$P` is the head of this department. `head($P,$X)` is an assertion which is true for the whole site, and thus we can add this assertion to the global environment. This is quite equivalent to building a local ontology. If, in some context, an ontology already exists, we can think of using it as an external resource. Notice that a triplet in RDF can be viewed as a Prolog predicate [35].

```
<agent><$P></agent>
      ? appointment($P,$X)
         / <li> Warning: <$P> line <$SourceLine>
                does not appear in the current
                staff chart.
           </li>;
```

This rule illustrates the generation of error messages. In the XML text, the tag `agent` is used to indicate that we have to check that the designated person has got an appointment in a department. The treatment of errors hides some particular techniques as the generated code must be rich enough to enable to locate the error in the source code.

6.2 Verifying a document and inferring new data

As a real sized test application, we have used the scientific part of the activity reports published by Inria for the years 2001 and 2002 which can be found at the following URLs:
http://www.inria.fr/rapportsactivite/RA2001/index.html and
http://www.inria.fr/rapportsactivite/RA2002/index.html.

The sources of these activity reports are LaTex documents, and are automatically translated into XML to be published on the Web.

The XML versions of these documents contain respectively 108 files and 125 files, a total of 215 000 and 240 000 lines, more than 12.9 and 15.2 Mbytes of data. Each file is the reflect of the activity of a research group. Even if a large part of the document is written in French, the structure and some parts of the document are formalized. This includes parts speaking of the people and the bibliography.

Concerning the people, we can check that names which appears in the body of the document are ”declared” in the group members list at the beginning. If it is not the case, the following error message is produced:

```
Warning: X does not appear in the list of
 project’s members (line N)
```

Concerning the bibliography of the group, the part called “publications of the year” may produce error messages like the following one:

```
Warning: The citation line 2176 has not been
 published during this year (2000)
```

We have also use Wget to check the validity of URLs used as citation, producing the following error messages:

```
Testing of URL
 http://www.nada.kth.se/ruheconference/
 line 1812 in file "aladin.xml" replies:
 http://www.nada.kth.se/ruheconference/:
 14:50:33 ERREUR 404: Not Found.

Testing of URL
 http://citeseer.nj.nec.com/ning93novel.html
 line 1420 in file "a3.xml" replies:
 No answer or time out during wget,
 The server seems to be down or does not
 exist.
```

Beyond these verification steps, using a logic programming approach allows us to infer some important information. For example we found out that 40 publications out of a total of 2816 were co-written by two groups, giving an indicator on the way the 100 research groups cooperate.

```
The citation  "Three knowledge representation
 formalisms for content-based manipulation of
 documents" line 2724 in file "acacia.xml" has
 been published in cooperation with
 orpailleur.
```

Our system reported respectively 1372 and 1432 messages for the years 2001 and 2002. The reasons of this important number of errors are various. There is a lot of misspelling in family names (mainly due to missing accents or to differences between the name used in publications and the civil name for women). Some persons who participated to some parts of a project but have not been present during all the year can be missing in the list of the team's members. There is also a lot of mistakes in the list of publications: a paper written during the current year can be published the year after and should not appear as a publication of the year. There are also a lot of errors in URLs and this number of errors should increase as URLs are not permanent objects.

It is to be noticed that we have worked on documents that have been generated, and that the original latex versions have been carefully reviewed by several persons. This means that proofreading is not feasible for large documents.

For future developments, other important indicators can also be inferred: how many PhD students are working on the different teams, how many PhD theses are published, etc. These indicators are already used by the management to evaluate the global performance of the institute but are compiled manually. An automatic process should raise the level of confidence in these indicators. They can also be compared mechanically with other sources of information as, for example, the data bases used by the staff office.

6.3 Implementation notes

All the implementation has been done in Prolog (more exactly Eclipse) except the XML scanner which has been constructed with flex.

An XML parser has been generated using an extension of the standard DCG. This parser has been extended to generate our specification language parser.

The rule compiler as been entirely written in Prolog.

Concerning the execution of the specification, the main difficulty comes from the fact that we have a global environment. The traditional solution in this case is to use coroutines or delays. As our input comes from many files, this solution was not reasonable, and we have chosen a two passes process. For each input file, during the first pass we use the local environment and construct the part of the global environment which is generated by the current file and a list of delayed conditions which will be solved during the second pass. During the second pass, all the individual parts of the global environment are merged and the result is used to perform the delayed verifications, producing errors messages when necessary.

A special mention should be made concerning pages that are dynamically generated on demand. These should not to be mixed up with pages that are statically generated by a compilation process. For pages generated on demand, one can perform some checks on this pages, but two points must be stressed : first, they must be checked in a fix global environment, and if for some reason the environment is modified, the modifications must not have an impact on the rest of the system; second, such a verification is a bit late, as it will be difficult to tell the end-user that a site produces an erroneous information. Other techniques to prove that the dynamic generation process is correct should be used.

Performances are of course an issue. Performances depend on two parameters : the size of the pages (viewed as Prolog terms), and the number of rules that must be performed. However the choice of Prolog here is a good point, as modern Prolog compilers make a smart use of hash-coding for clauses indexing.

7 Conclusion

Web sites and ontologies are formal objects and we have shown how techniques used to define the formal semantics of programming languages can be used in the context of the Web. This work can be viewed as a complement to other researches which may be very close: for example in [14], some logic programming methods are used to specify and check integrity constraints in the structure of Web sites (but not to its static semantics). Our work which is focused on the content of Web sites and on their formal semantics, remains original. It can be extended to the content management of more general information system.

As we have seen, the techniques already exist and are used in some other domains. The use of logic programming, and in particular of Prolog as an implementation language, is very well adapted to our goals. Our goal is to make these techniques accessible and easily usable in the context of Web sites with the help of a specific specification language. We are now convinced that the technology is adequate.

The gap between information systems and programs, ontologies and types is small. The world of software engineering and the type community have already investigate some important notions such as overloading, polymorphism, type parameters, and more generally modularity. A next step should be a fertilization step, importing these notions to construct more powerful languages and tools for the Web and other information systems.

References

1. Alpuente, M., Ballis, D., Falaschi, M.: Rule-based verification of web sites. Software Tools for Technology Transfer **8**(6), 565–585 (2006)
2. Alpuente, M., Ballis, D., Falaschi, M., Ojeda, P., Romero, D.: Fast algebraic verification service. In: Proceedings of the First International Conference on Web Reasoning and Rule Systems, LNCS. Springer-Verlag (2007)
3. Berners-Lee, T.: A Road Map to the Semantic Web (1998). W3C http://www.w3.org/DesignIssues/Semantic.html
4. Berners-Lee, T.: Ideas about web architecture - yet another notation: Notation 3 (2001). W3C http://www.w3.org/DesignIssues/Notation3.html
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. In: Scientific American (2001)
6. Deshpande, Y., Olsina, L., Murugesan, S.: Third ICSE workshop on Web engineering. In: Proceedings of ICSE'2002 (2002)
7. Despeyroux, T.: Executable Specification of Static Semantics. In: Semantics of Data Types, Lecture Notes in Computer Science,Vol. 173 (1987)
8. Despeyroux, T.: Practical semantic analysis of Web sites and documents. In: Proceedings of the 13th World Wide Web Conference (WWW2004). ACP Press (2004)
9. Despeyroux, T., Lechevallier, Y., Trousse, B., Vercoustre, A.M.: Experiments in clustering homogeneous xml documents to validate an existing typology. In: Proceedings of the 5th International Conference on Knowledge Management (I-Know). Vienne, Autriche (2005)
10. Despeyroux, T., Trousse, B.: Semantic verification of Web sites using Natural Semantics. In: RIAO 2000, 6th Conference on "Content-Based Multimedia Information Access", College de France, Paris, France (2000)
11. Despeyroux, T., Trousse, B.: Maintaining semantic constraints in web sites. In: AACE WebNet 2001 Conference, Orlando, Florida (2001)
12. Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H.P., Studer, R., Witt, A.: On2broker: Lessons Learned from Applying AI to the Web. Tech. rep., Institute AIFB (1998)
13. Fensel, D., Decker, R., Erdman, M., Studer, R.: Ontobroker : the Very High Idea. In: Proceedings of the 11th Internation FLAIRS Conference (FLAIRS-98) (1998)
14. Fernandez, M.F., Florescu, D., Levy, A.Y., Suciu, D.: Verifying integrity constraints on web sites. In: IJCAI, pp. 614–619 (1999). URL citeseer.nj.nec.com/fernandez99verifying.html
15. Flores, S., Lucas, S., Villanueva, A.: Formal verification of websites. In: Proceedings of the Third International Workshop on Automated Specification and Verification of Web Sites (WWV'07), Electronic Notes in Theoretical Computer Science. Elsevier Science (2008)
16. Grosof, B.N., Volz, R., Horrocks, I., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proceedings of the 12th World Wide Web Conference (WWW2003) (2003)
17. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2) (1993)
18. Gunter, C.A.: Semantics of Programming Languages. MIT Press (1992)
19. van Harmelen, F., Fensel, D.: Practical Knowledge Representation for the Web. In: D. Fensel (ed.) Proceedings of the IJCAI'99 Workshop on Intelligent Information Integration (1999)
20. van Harmelen, F., van der Meer, J.: Webmaster: Knowledge-based Verification of Web-pages. In: Twelfth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE'99 (1999)
21. Holck, J.: 4 perspectives on web information systems. In: Proceedings of the 36th Hawaii International Conference on System Sciences (2003)
22. Horrocks, I., Patel-schneider, P.F., Harmelen, F.V.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics **1**, 2003 (2003)
23. Hosoya, H., Pierce, B.: Xduce: A typed XML processing language. In: Proceedings of Third International Workshop on the Web and Databases (2000)
24. Kahn, G.: Natural Semantics. In: Proceedings of the Symp. on Theorical Aspects of Computer Science, TACS. LNCS 247, Springer-Verlag, Berlin, Passau, Germany (1987). Also Inria Research Report 601, February 1987

25. Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology versioning and change detection on the web. In: In 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02, pp. 197–212 (2002)
26. Klyne, G., Carroll, J.J.: Resource description framework (RDF): Concepts and abstract syntax (2004). W3C Recommendation http://www.w3.org/TR/rdf-concepts/
27. Lucas, S.: Rewriting-based navigation of web sites: Looking for models and logics. In: Proceedings of the First International Workshop on Automated Specification and Verification of Web Sites (WWV'05), vol. 157. Electronic Notes in Theoretical Computer Science, Elsevier Science (2005)
28. Luong, P.H.: Gestion de l'évolution d'un web sémantique. Ph.D. thesis, Ecole des Mines de Paris (2007)
29. Luong, P.H., Dieng-Kuntz, R.: A rule-based approach for semantic annotation evolution. The Computational Intelligence Journal **23**(3) (2007)
30. Luong, P.H., Dieng-Kuntz, R., Boucher, A.: Evolution de l'ontologie et gestion des annotations sémantiques inconsistantes. In: Proceedings of Extraction et gestion des connaissances (EGC'2007), Revue des Nouvelles Technologies de l'Information. Cpadus (2007)
31. Luong, P.H., Dieng-Kuntz, R., Boucher, R.: Managing semantic annotations evolution in the CoSWEN system. In: Proceedings of the Third National Symposium on Research, Development and Applicaion of Information and Communication Technology (ICT.rda'06) (2006)
32. Meijer, E., Shields, M.: XMλ: A functional programming language for constructing and manipulating xml document (1999). Draft, http://www.cse.ogi.edu/ mbs/pub/xmlambda/
33. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web engineering: A new discipline for development of web-based systems. In: Proceedings of the Web Engineering, Software Engineering and Web Application Development. Springer Verlag (2001)
34. Patel-Schneider, P.F., Horrocks, I.: A comparison of two modelling paradigms in the semantic web. In: Proc. of the Fifteenth International World Wide Web Conference (WWW 2006), pp. 3–12. ACM (2006). URL download/2006/PaHo06a.pdf
35. Peer, J.: A logic programming approach to RDF document and query transformation. In: Workshop on Knowledge Transformation for the Semantic Web at the 15th European Conference on Artificial Intelligence. Lyon, France (2002)
36. Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, Aarhus University (1981)
37. Pressman, R.S., Lewis, T., Adida, B., Ullman, E., DeMarco, T., Gilb, T., Gorda, B., Humphrey, W., Johnson, R.: Can internet-based application be engineered ? IEEE Software **15**(5) (1998)
38. Stallman, R.M., McGrath, R.: GNU Make: A Program for Directing Recompilation, for Version 3.79. Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA, Tel: (617) 876-3296, USA (2000). URL citeseer.nj.nec.com/stallman91gnu.html
39. Szabo, E.: The Collected Papers of Gerhard Gentzen. North-Holland, Amsterdam (1969)
40. W3C: Xml, xsl, xml schema and rdf recommendations or submissions. W3C http://www.w3.org/
41. W3C: Daml+oil (march 2001) reference description (2001). W3C http://www.w3.org/TR/daml+oil-reference
42. Weithöner, T., Liebig, T., Specht, G.: Efficient processing of huge ontologies in logic and relational databases. In: Proceedings of Ontologies, Databases, and Applications of Semantics Conference (ODBASE'2004) (2004)