RDF Tutorial

Pierre-Antoine Champin

February 23, 2000

Introduction

This document is a presentation of the *Resource Description Framework* (RDF) recommanded by the *World Wide Web Consortium* (W3C), to model metadata about the resources of the web. It is described in both documents [1] and [2]; the former focuses on syntactical aspects while the later addresses the definition of vocabularies (often named *schemas*). Though this is a pragmatic approach, I will use a slightly different plan, more suited to my own goals (using RDF in knowledge representation systems). The first section will describe the RDF model, which is its fundamental syntax. The second section will present the semantic aspects of RDF, the concepts and the correpsonding vocabulary. The last section will describe the XML syntax proposed in [1].

1 Model

1.1 URIs

RDF identifies resources with *Uniform Resource Identifiers* (URI) as described in [3], but with a slight difference: RDF uses what we will call *qualified* URIs, that is, URIs with an optional fragment identifier (a text added to the URI with a "#" between them). For [3], the fragment identifier returns “a property of the data resulting from a retrieval action”; however, RDF considers every qualified URI (with or without fragment identifier) as a full resource by itself[1]. In the rest of this tutorial, we will therefore use the word “resource” in the RDF point of view, unless specified otherwise.

[1] It could then lead to more than one fragment identifier in a URI... This would not be incompatible with [3] but I've never seen such URI used.

1.2 Triples and graph

The base element of the RDF model is the triple : a resource (the *subject*) is linked to another resource (the *object*) through an arc labeled with a third resource (the *predicate*). We will say that <subject> has a *property* <predicate> *valued* by <object>. For example, the triple in figure 1 could be read as "Champin is the creator of index.html".

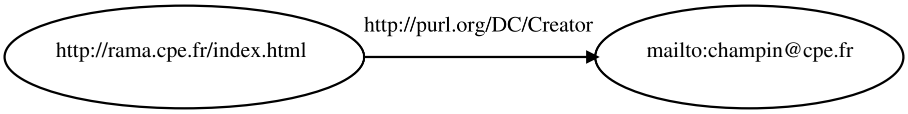

Figure 1: A triple

All the triples result in a direct graph, whose nodes and arcs are all labeled with qualified URIs. Note in figure 2 that a resource may have more than one value for a given property.

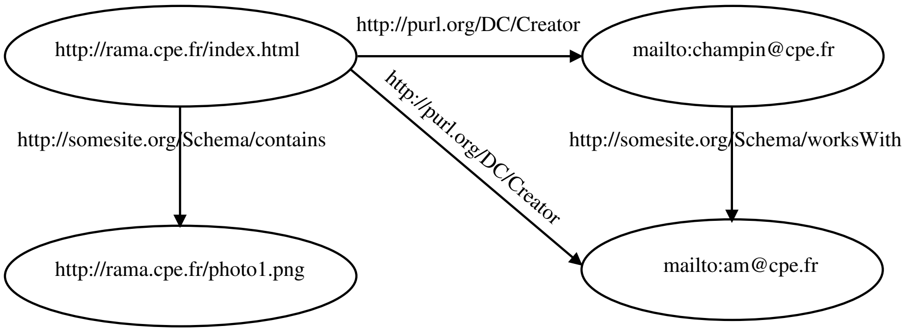

Figure 2: An RDF graph

1.3 Literals

In the RDF recommandation, pits of the graph can be pieces of text instead of resources; those pieces of text are called literals. Though this is syntactically useful in RDF documents (being able to write directly some text rather than storing it in another resource), I think handling it as an exception to the model is not appropriate, and that there is a cleaner way of doing it: the piece of text probably has a fragment identifier[2] and so the literal node could

[2]in an XML syntax, *XPointer* ([4]) could be used

be replaced by a standard URI node in the RDF graph. Note that this alternative is compatible with the recommandation since it uses a subset of the model.

1.4 Uniformity

The RDF model is very simple and, above all, uniform. Mostly, the fact that the only vocabulary is URIs allows the use of the same URI as a node *and* as an arc label. This makes things like self-reference and reification possible, just like in natural languages. This is appreciable in a user-oriented context (like the web), but may become difficult to cope with in knowledge-based systems and inference engines.

2 Concepts and vocabulary

We can distinguish three kinds of concepts in RDF: fundamental concepts, schema-definition concepts (useful for defining new vocabularies) and utility concepts (concepts which are not absolutely necessary, but likely to be useful in any application domain).

All these concepts have been given a URI. These URIs are defined as fragment identifiers of the URIs of the W3C documents defining RDF. For the sake of clarity, we will rather use the XML non-expanded notation ; that is, prefixes `rdf:` and `rdfs:` will be used instead of `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222#` and `http://www.w3.org/TR/1999/PR-rdf-schema-19990303#` respectively. The membership of one or the other namespace may not always seem logical, and must have historical reasons mostly.

2.1 Fundamental concepts

2.1.1 rdfs:Resource

RDF is about describing resources ; according to [1], "resources are always named by URIs" and "anything can have a URI". So RDF can theoretically be used to describe anything. Yet it was mainly designed to handle "network retrievable" resources.

[3] underlines that "the resource is the conceptual mapping to an entity (...), not necessarily the entity which corresponds to that mapping *at any particular instance in time*". However most of the time we are interested in entities themselves. It is therefore important to note that the metadata we

express about resources may require different levels of interpretation, which may be valid over various periods of time.

For example, the page `http://rama.cpe.fr/index.html` is a web document, this is inherent to that kind of URI. Its size in kylobytes, on the other hand, is a property of the file retrieved by the HTTP server. Its author depends on the content of that file. Levels of interpretation are even harder to cope with when the resource is not "retrievable": in figure 1, an e-mail URI is used to identifty a person, but in another context, it may not be the case (especially for an application dealing with the different e-mail addresses of that person).

Interpretation of RDF triples is therefore highly contextual, since URI don't have a unique interpretation. That should be kept in mind by RDF users and implementors.

2.1.2 rdf:Property

The properties are resources used as predicate of triples; the semantic of a triple clearly depends on the property used as predicate. Two things are very important with the concept of property:

First, RDF considers properties as first class object, unlike most object modelling languages, where properties are attributes of a class. Even though the concept of class exists in RDF (see subsection 2.2), properties can be defined and used independently of classes.

Secondly, as it has been mentioned in subsection 1.4, the fact that properties are resources allows to describe them with RDF itself. This will be widely used by the following concepts.

2.1.3 rdf:Statement

A statement is a resource reifying a triple. Such a resource must have at least 3 properties[3]: rdf:subject, rdf:object and rdf:predicate, valued by the corresponding resources.

The reification of triples may seem a utility concept rather than a fundamental concept. Nevertheless it is defined as a part of the model in the W3C recommandation. This supports the will to use RDF as its own meta-system, to make every element of RDF describable in RDF itself.

[3]Actually, [1] defines a fourth property for statements, rdf:type, which must be valued by rdf:Statement. Since this is generalized by the concept of Class in [2], we don't mention it here.

2.2 Schema definition concepts

All those concepts are defined in [2], the second document of the W3C, to allow the definition of schemas, that is, vocabularies of resources to use with RDF. Some specialized agents, limited to a pre-defined vocabulary, may be unaware of these concepts.

In schemas, new resources can be defined as *specialization* of old ones, thus allowing to infer implicit triples. Schemas also constrain the context in which defined resources may be used, inducing the notion of *schema validity*. We will see that those two notions can be seen as one, in a point of view based on predicate[4] logic. They all can be expressed as rules allowing to infer new facts (basically, new triples or *negations* of triples). In these rules, the 3-ary logical predicate $\mathcal{T}(subject,\ predicate,\ object)$ will be used to represent a believed triple.

2.2.1 rdfs:subPropertyOf

Any property denotes a relation between resources (the set of resource couples linked by an arc labeled with the property). rdfs:subPropertyOf applies to properties and must be interpreted as the subset relation between the relations they denote. Thus the following rule stands:

$$\forall s, p_1, o, p_2\ \ \mathcal{T}(s,\ p_1,\ o) \wedge \mathcal{T}(p_1,\ \texttt{rdfs:subPropertyOf},\ p_2) \Rightarrow \mathcal{T}(s,\ p_2,\ o)$$

For example, if “mother” is a subproperty of “parent”, any triple having “mother” as predicate must also be considered as having “parent” as predicate. This property is very important in schema definitions for interoperability between RDF agents. In the example above, an agent not knowing the semantic of “mother” could at least treat it as “parent” (assuming it knows the semantic of “parent”).

Since rdfs:subPropertyOf denotes a subset relation, the transitivity rule also stands:

$$\begin{gathered}\forall p_1, p_2, p_3 \\ \mathcal{T}(p_1,\ \texttt{rdfs:subPropertyOf},\ p_2) \wedge \mathcal{T}(p_2,\ \texttt{rdfs:subPropertyOf},\ p_3) \\ \Rightarrow \mathcal{T}(p_1,\ \texttt{rdf:subPropertyOf},\ p_3)\end{gathered}$$

Note that it is considered invalid by [2] to have cycles in rdfs:subPropertyOf, though it doesn't define a way to express this constraint in RDF[5]. Anyway,

[4] I will keep usign the word *predicate* in the RDF meaning in the following, unless specified otherwise.

[5] Furthermore, this constraint may seem too strong: cycles could be used to express equivalence between properties.

the corresponding logical rule is the following (since any cycle would result, with transitivity, in a property being its own subproperty):

$$\forall p \;\; \neg\mathcal{T}(p,\, \texttt{rdfs:subPropertyOf},\, p)$$

Note also that there is no standard URI for the universal property (superproperty of any property).

2.2.2 rdfs:Class, rdf:type and rdfs:subClassOf

Classes are resources denoting a set of resources, by the mean of the property rdf:type (instances have property rdf:type valued by the class). Since all sets of resources presented in this section are resources (they have a URI), they have by definition the property rdf:type valued by rdfs:Class. On the other hand, all properties (defined in W3C recommandation or in any schema) have rdf:type valued by rdf:Property.

Classes are structured the same way as properties, in a subset hierarchy denoted by the property rdfs:subClassOf. As for rdfs:subPropertyOf, cycles must not exist though it could be used to express equivalence, but contrary to the property hierarchy, the class hierarchy has a maximum element: it is of course rdf:Resource (so any class implicitly has rdfs:subClassOf valued by rdf:Resource). The following rules, similar to the rules related to rdfs:subPropertyOf, stand:

$$\forall i, c_1, c_2 \;\; \mathcal{T}(i,\, \texttt{rdf:type},\, c_1) \wedge \mathcal{T}(c_1,\, \texttt{rdfs:subClassOf},\, c_2) \\ \Rightarrow \mathcal{T}(i,\, \texttt{rdf:type},\, c_2)$$

$$\forall c_1, c_2, c_3 \;\; \mathcal{T}(c_1,\, \texttt{rdfs:subClassOf},\, c_2) \wedge \mathcal{T}(c_2,\, \texttt{rdfs:subClassOf},\, c_3) \\ \Rightarrow \mathcal{T}(c_1,\, \texttt{rdf:subClassOf},\, c_3)$$

$$\forall c \;\; \neg\mathcal{T}(c,\, \texttt{rdfs:subClassOf},\, c)$$

2.2.3 rdfs:domain and rdfs:range

These properties apply to properties and must be valued by classes. They are used to restrict the set of resources that may have a given property (the property's *domain*) and the set of valid values for a property (its *range*). A property may have as many values for rdfs:domain as needed, but no more than one value for rdfs:range:

$$\forall p, r_1, r_2 \;\; \mathcal{T}(p,\, \texttt{rdfs:range},\, r_1) \wedge r_1 \neq r_2 \Rightarrow \neg\mathcal{T}(p,\, \texttt{rdfs:range},\, r_2)$$

For a triple to be valid, the subject must match the range (if any) of the predicate (that is, it must have rdf:type valued by the corresponding class or one of its subclasses), and the object must match at least one of the domains (if any) of the predicate (Note that if the predicate has superproperties, this must also be checked recursively for all of them). This can be logically expressed by:

$$\forall s, p, o \;\; \mathcal{T}(s,\ p,\ o) \wedge \exists d \; \mathcal{T}(p,\ \texttt{rdfs:domain},\ d)$$
$$\Rightarrow \exists d' \; (\mathcal{T}(p,\ \texttt{rdfs:domain},\ d') \wedge \mathcal{T}(s,\ \texttt{rdf:type},\ d'))$$

$$\forall s, p, o, r \;\; \mathcal{T}(s,\ p,\ o) \wedge \mathcal{T}(p,\ \texttt{rdfs:range},\ r) \Rightarrow \mathcal{T}(o,\ \texttt{rdf:type},\ r)$$

It's worth noting than although this two rules are intended to be used for validity checking only, and although the first one (rdfs:domain rule) can actually only be used this way (it can't be used to performed inference since its consequence is existentially qualified), the second one (rdfs:range rule) has different interpretations depending on our hypothesizing a closed or open world. In the closed world hypothesis, any missing triple is considered negated, so the rdfs:range rule has only to be verified. But in an open world hypothesis, missing triples are not necessarily false, so the rule could be used to perform inference instead. Since the "natural" field of RDF is the web, where information is by essence distributed and incomplete, the open world hypothesis seems much more reasonnable.

2.2.4 rdfs:Literal

[2] defines a resource rdfs:Literal, denoting the set of literals, declared as a class (though literals are not resources, according to the recommendation!) so as to be a valid value for rdfs:range, which is its sole use.

2.3 Utility concepts

Those concepts may have been defined in external schemas, but since they are of very common use, they have been defined once for all in the core schema.

2.3.1 rdfs:Container

Containers are collections of resources. They are modelled by an instance of one of the three subclasses of rdfs:Container: rdf:Bag (an unordered collection), rdf:Seq (an ordered collection) or rdf:Alt (an alternative). Membership is modelled by automatically generated properties rdf:_1, rdf:_2, *etc.* These properties are all instances of rdfs:ContainerMembershipProperty, a subclass of rdf:Property[6].

This mechanism for modelling collections of resources is much controversed among RDF implementors: the numbering of membership properties is contentious especially when neither rdf:Bag nor rdf:Alt need an order relation between their members. Furthermore, these properties can't be expressed in any schema since there is an unbounded number of them. This can even cause infinite loops in agents using the XML syntax (see annex A).

2.3.2 rdfs:ConstraintResource and rdfs:ConstraintProperty

It can be interesting for an RDF agent to be informed that an unknown resource (or more specifically a property) is defining a validity constraint. The set of such resources is rdfs:ConstraintResource. Its subclass rdfs:ConstraintProperty is of course a subclass of rdf:Property too. Properties rdfs:domain and rfds:range defined above are instances of rdfs:ConstraintProperty.

2.3.3 rdfs:seeAlso and rdfs:isDefinedBy

A given resource may be described in more than one place over the internet. The rdfs:seeAlso property can be used to point to alternative descriptions of the subject resource. Its subproperty rdfs:isDefinedBy more specifically points to an original or authoritative description.

2.3.4 rdfs:label and rdfs:comment

It can be useful to describe a resource with human readable text in addition to "pure" RDF properties ; this is the role of rdfs:label and rdfs:comment. The former is used to give a human-readable name of a resource, the latter, to give a longer description. Note that they may have multiple values for internationalization needs.

3 XML syntax

This section describes the XML syntax recommanded by [1] (we assume that the reader is already familiar with XML). It uses XML namespace notations

[6] a common superproperty, in my opinion, would have been more useful than a class, to federate those membership properties.

([5]), but expanded names are obtained simply by concatenating the namespace to the element name. Hence we will use the same convention as in the previous section for prefixes `rdf:` and `rdfs:`.

3.1 Descriptions

An RDF document is a list of descriptions. Each description applies usually to one resource, and contains a list of properties. Property values are either URIs, literals or others Descriptions.

In XML, RDF metadata are embeded in an element named rdf:RDF. This element contains a sequence of elements named rdf:Description. Those elements can have one of the two attributes rdf:about or rdf:ID (but not both).

- rdf:about is used to describe any resource ; its value is either an absolute or a relative URI.

  ```
  <rdf:Description about="http://rama.cpe.fr/index.html">
    ...
  </rdf:Description>
  ```

- rdf:ID is used to define a resource ; its value is a fragment identifier (without the "#" character) to be added to the XML document URI. A resource may not be defined more than once.

  ```
  <rdf:Description ID="foo">
    ...
  </rdf:Description>
  ```

- a description without rdf:about nor rdf:ID is said to describe an *anonymous* resource.

  ```
  <rdf:Description>
    ...
  </rdf:Description>
  ```

An element rdf:Description contains a sequence of XML elements. Those elements are interpreted as properties, whose predicate's URI is the expanded name of the element. If the element is empty, it must have an attribute rdf:resource whose value is the object's URI (see 1^{st} dc:Creator in fig.3). Else, it can contain plain text (then interpreted as a literal - see dc:Title

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/DC/"
         xmlns:os="http://somesite.org/Schema/">
  <rdf:Description about="http://rama.cpe.fr/index.html">
    <dc:Creator rdf:resource="mailto:am@cpe.fr"/>
    <dc:Title> Index of my web site </dc:Title>
    <dc:Creator>
      <rdf:Description about="mailto:champin@cpe.fr">
        <os:worksWith rdf:resource="mailto:am@cpe.fr"/>
      </rdf:Description>
    </dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

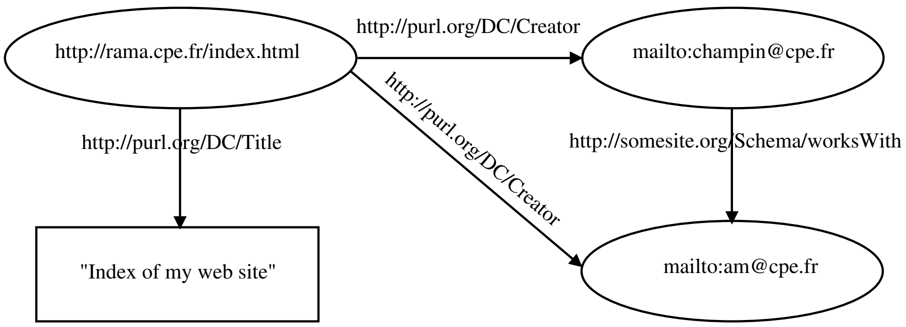

Figure 3: XML syntax simple example

in fig.3) or a single embeded rdf:Description element (see 2^{nd} dc:Creator in fig.3).

In case of ambiguity, the attribute rdf:parseType can be used in property elements with "Resource" or "Literal" value. The later can be used when a literal contains XML tags, to prevent their being parsed as a description.

The syntax may also be abbreviated in two ways:

- the description element name may be replaced by any URI. It is interpreted as an additionnal rdf:type property, valued by the named resource. Note that the context always allows to differ a typed description element from a property element.
- properties with literal values may be written as attributes of the description element rather than an embeded elements.

For example, the following description:

```
<rdf:Descritpion ID="fatherOf">
  <rdf:type rdf:resource=
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdf:label> father of </rdf:label>
  <rdfs:subPropertyOf rdf:resource="#parentOf">
</rdf:Descritpion>
```

may be abbreviated into

```
<rdf:Property ID="fatherOf" rdfs:label="father of">
  <rdfs:subPropertyOf rdf:resource="#parentOf">
</rdf:Descritpion>
```

3.2 Containers

As mentioned in the previous section, RDF containers are defined as a part of the XML syntax. A container node is described with a special element named rdf:Bag, rdf:Seq or rdf:Alt (those elements can be used instead of rdf:Description elements). Those special descriptions can only have rdf:ID attribute or be anonymous, they can't have rdf:about.

Membership properties are not used as is, but instead the element rdf:li is used; the parser has to replace it by the appropriate numbered property. Figure 4 is an example of it.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Bag ID="mybag">
    <rdf:li resource="http://rama.cpe.fr/index.html"/>
    <rdf:li resource="mailto:champin@cpe.fr"/>
    <rdf:li> literal element </rdf:li>
  </rdf:Bag>
</rdf:RDF>
```

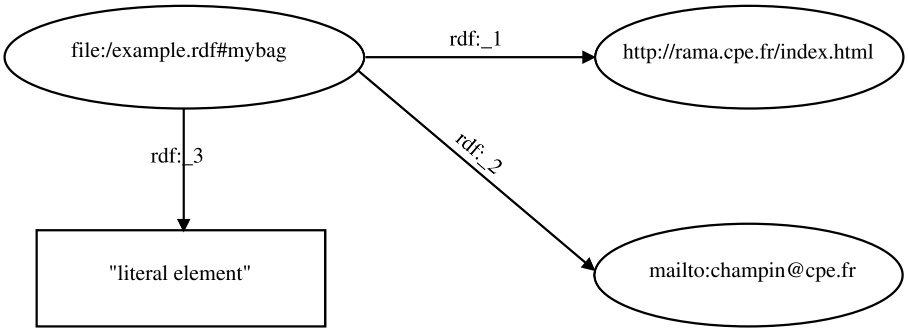

Figure 4: An RDF bag

3.3 Distributed descriptions

Instead of rdf:about, descriptions can also have attribute rdf:aboutEach or rdf:aboutEachPrefix. This allows to distribute a description over a set of resources.

- rdf:aboutEach's value must be a container's URI; the corresponding description applies to every member of the container.
- rdf:aboutEachPrefix's value is a string; the corresponding description applies to any resource whose URI starts with this string.

Implementations of RDF are not bound to keep information about those distributive descriptions - this is only a syntactic shortcut. It may result in unexpected behaviours of some agents, as soon as more than one RDF source is involved (resources defined in a source may match a distributed description in another one, but not be detected as such).

Moreover, it is not specified whether distributed description may be embeded as property values. Though this possibility is never mentionned, the formal syntax allows it. A natural interpretation would be that the property is valued by every resource matching the distributed description.

3.4 Reification

The XML syntax of RDF provides a way to reify asserted statements: a single arc can be reified by adding an rdf:ID attribute to the property element, which will define the URI of the reified statement. On the other hand, every arc generated by a description can be reified by adding an rdf:bagID attribute to the Description. This defines the URI of a new bag resource, whose members are the refied statements.

A A naughty example

This trivial example is likely to lead most RDF parsers into an infinite loop:

```
[1] <?xml version="1.0" encoding="UTF-8" ?>
[2] <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
[3]          xmlns:rdfs:"http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
[4]   <rdfs:Class rdf:ID="WebResource"/>
[5]   <rdf:Description aboutEachPrefix="http:" bagID="bag">
[6]     <rdf:type resource="#WebResource"/>
[7]   </rdf:Description>
[8] </rdf:RDF>
```

Line 4 generates a resource <this file>#WebResource, then generates an arc between it and rdfs:Class, labeled with rdf:type.

Then the description lines 5 to 7 is parsed; at this point, there is at least one resource starting with "http:" (rdfs:Class). So the distributed description will generate at least one arc, which will be reified and stored in the bag.

But the storing in the bag will generate a membership property rdf:_1, which starts with "http:". So the distributed description will generate another arc, another reification stored in the bag, and another membership property. That may repeat forever.

It is very important for RDF parsers to be able to avoid those infinte loops; a simple solution is to prevent any membership property from matching rdf:aboutEachPrefix descriptions. A cleaner solution would be to use another schema for containers...

References

[1] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification. W3C recommandation, 2 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[2] Dan Brickley and R.V. Guha. Resource description framework (rdf) schema specification. W3C proposed recommandation, 3 1999. http://www.w3.org/TR/1999/PR-rdf-schema-19990303.

[3] Tim Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform resource identifiers (uri): Generic syntax. RFC 2396, 8 1998. http://www.faqs.org/rfcs/rfc2396.html.

[4] Steve DeRose, Ron Daniel, and Eve Maler. Xml pointer language (xpointer). W3C working draft, 12 1999. http://www.w3.org/TR/1999/WD-xptr-19991206.

[5] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in xml. W3C recommandation, 1 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114.