The Extensible XML Information Set

Erik Wilde
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology, Zürich

TIK Report 160 (February 2003)

Abstract

XML and its data model, the XML Information Set, are used for a large number of applications. These applications have widely varying data models, ranging from very simple regular trees to irregularly structured graphs using many different types of nodes and vertices. While some applications are sufficiently supported by the data model provided by the XML Infoset itself, others could benefit from extensions of the data model and assistance for these extensions in supporting XML technologies (such as the DOM API or the XSLT programming language). In this paper, we describe the *Extensible XML Information Set (EXIS)*, which is a reformulation of the XML Infoset targeted at making the Infoset easier to extend and to make these extensions usable in higher-level XML technologies. EXIS provides a framework for defining extensions to the core XML Infoset, and for identifying these extensions (using namespace names). Higher-level XML technologies (such as DOM or XPath) can then support EXIS extensions through additional interfaces, such as a dedicated DOM module, or XPath extension mechanisms (extension axes and/or functions). In order to make EXIS work, additional efforts are required in these areas of higher-level XML technologies, but EXIS itself could be used rather quickly to provide a foundation for well-defined Infoset extensions, such as XML Schema's PSVI contributions, or the reformulation of XLink as being based on a data model (rather than a syntax).

Contents

1 Introduction

The *Extensible Markup Language (XML)* [5] and its data model, the *XML Information Set (XML Infoset)* [7], today serve as a foundation for many data formats and applications. While it is possible to map a large variety of data models onto the tree structure provided by XML, in many cases it would be useful to extend the data model of XML, and to use this extended data model together with the growing set of XML technologies (such as DOM, SAX, XSLT, and XQuery) as well as in applications.

Coming from a background of XML and hypermedia, and in particular the *XML Linking Language (XLink)* [8], we were drawn to the conclusion that it can have substantial advantages to specify a data model before a syntax. As an example, the lengthy debate about whether XHTML 2.0 should use XLink or use its own linking model, was largely clouded by discussions about the XLink *syntax*, rather than the *link model* defined by XLink. What this paper is about is a reformulation of the XML Infoset (which does not change anything about what is in the core Infoset and what is not), which makes it easier and more well-defined to extend, and thus enables layered XML applications augmenting the data model of XML with new items and/or properties. We call this reformulation the *Extensible XML Information Set (EXIS).*

The proposal of such an extensible XML data model is controversial. While it would certainly add to the complexity of XML-related specifications, it would be transparent to users not interested in it and would only have to be used by specification authors or users who are interested in making their particular piece of XML technology extensible and accessible through standard XML mechanisms, such as XPath or DOM. We see the divide between XML users as following:

- *Users satisfied with the XML data model*

  For this class of users, who are content with the data model that XML and the Infoset provide, EXIS probably is of little interest. There is one exception, which is the support of non-standard XML encodings. Some people may be happy with XML's data model, but may have problems with its verbose syntax. For these users, a framework enabling the use of multiple syntaxes may be interesting (in Section 5.2, this particular point will be discussed in more detail).

- *Users requiring a more sophisticated data model*

  Some users may want to see particular facets of their data model reflected in the data model, rather than having to map them onto the core XML data model. In this case, a way to extend the data model and then manipulate data with this extended model through standard XML technologies is certainly useful. Users finding themselves in this situation probably will like the approach of EXIS with its more flexible and powerful way of using XML-based data.

We start this paper with Section 2 about the original motivation behind EXIS, and why we think that EXIS would be a good way to make XML-based technologies and applications more flexible. We then continue by shortly describing the current XML Infoset specification in Section 3, in particular its shortcomings (when it comes to well-defined extensibility). Moving on, we go into detail about EXIS giving a list of requirements (Section 4), describing the EXIS model itself (Section 5), and then give some examples of application areas where EXIS will provide a benefit to specification and application authors (Section 6). After that, we finish the paper by listing open issues (Section 7) and giving some concluding remarks (Section 8).

2 Motivation

XML's data model, the XML Infoset, defines a number of *Information Items* and *Properties*, which represent the information contained in an XML document. The Infoset ignores some information, such as whitespace in tags, or the order of attributes in a tag[1]. The Infoset can be regarded as a somewhat arbitrary, but widely accepted decision of what is relevant in an XML document and what is not. Because users may disagree, the Infoset explicitly allows extensions or restrictions of the Infoset. However, the Infoset is rather vague about the way how to extend or restrict it, and thus there is no established way to do it, and then to use these "derived infosets".

We therefore suggest to leave the Infoset's data model as it is, but to make more explicit the ways in which the Infoset may be used for deriving other infosets, in particular extensions to it. The following areas need clarification:

- *How are Infoset extensions identified?*
- *In which way(s) may the Infoset be extended?*
- *How are Infoset extension instances represented?*

To regular XML users, who are glad to have XML as a globally accepted and supported syntax to exchange structured data, the idea of extending the Infoset may seem excessive in terms of complexity. However, looking from a perspective of specification designers as well as authors of applications with complex data models, it makes a lot of sense to extend XML's core data model with additional information. Two examples of XML technologies which could benefit from an extended XML data model are CSS (which basically is a method to assign formatting semantics to elements), and XLink (which adds a link structure to an XML document). In order to illustrate the motivation behind the idea of extending the Infoset, we describe the XLink scenario in more detail.

A first proposal of XLink as a data model (rather than a syntax) has been published by Walsh [17], but this model was not complete and geared towards the presentation of XLinks. In a generalization [19], the idea of an XLink data model was elaborated further, and seeing that this reformulation of XLink as a data model required a solid foundation on which it could be built, we have made the point that the Infoset should be extensible in a general way [18].

While it is certainly possible to manipulate documents containing XLinks through XPath or DOM, this is rather error-prone and cumbersome: It is necessary to identify the XLink attributes by their namespace, and the fact that some element is participating in a link is not reflected in the element itself, but through a hierarchical relationship in the document structure. Furthermore, third-party links (links coming from another source than from the document they are linking) are not reflected in the core XML data model at all, but it would certainly be helpful to have them in the data model, too (if the application is enabled to retrieve third-party links).

Consequently, while it is theoretically possible to deal with documents containing XLink through standard XPath or DOM, this is not a very good way to go. As a result, one would either create some sort of proprietary XLink handling for manipulating these documents (in effect creating a DIY "DOM XLink module"), or one could argue that this is only one example

[1]See Appendix D of the XML Infoset Specification [7].

for other scenarios also introducing this kind of problem when using XML. We decided that it would be better to look at the root of the problem, and the result of this work is the idea to make XML's data model extensible in an open way.

Now one could argue that the Infoset is already open to extensions (as well as restrictions), and this is certainly true. However, this openness is only vaguely defined and not supported in any of the upper layers of the XML technologies. The most prominent example for this is *XML Schema* [16], which introduces a number of additional Infoset properties, the *Post Schema Validation Infoset (PSVI)* contributions. From a modelling perspective, this was the way to go. However, because there is no clearly defined interface for this extended Infoset, it is impossible to access these PSVI contributions from XPath or DOM in a standardized way.

Finally, it is interesting to look at the similarities between XLink and XML Schema Infoset augmentation, because it shows a general tendency of why it may be reasonable to use an extended data model: An application may get an XML document, and then retrieve XLinks which augment the document's original data. In much the same way, XML Schema defines validation as taking an XML document (a regular Infoset), and then validating it by augmenting the document's data with the results of the validation (the PSVI contributions). In both cases, the data model extensions are not inherently part of the data, but are created through an additional processing step (link retrieval or validation).

3 The XML Information Set

Historically, the *XML Information Set (XML Infoset)* [7] was created when specification writers recognized that rather than being based on a syntax, it would make much more sense for many XML specifications to be built on top of a data model, i.e., an abstraction of the XML syntax. The result of these efforts to define an XML data model, the XML Infoset, defines 11 types of *Information Items*, each having a number of *Properties*. While the Infoset explicitly allows to create sub- or supersets, it does not say how to do so. For example, it is not clear which datatypes may be used for properties, and what the possible relationships are between different items (the core Infoset uses sets and lists). Additionally, the Infoset uses some "special values" for properties (in particular, *no value* and *unknown*), and there is no definition of how to exactly use these values, and whether there may be other special values.

Furthermore, it is completely unclear how to specify derived Infosets (the Infoset specification uses a non-formal, text-based way to define the items and properties), and how these derived Infosets may be identified and used by other specifications or applications. Thus, only little use of the Infoset's flexibility has been seen so far. The XML Schema PSVI example mentioned in the previous section demonstrates this by using a non-formal way of specifying the PSVI contributions[2]. The downside of this laudable effort to keep things in line with existing specifications is that implementors of XML Schema processors have no standardized way of exposing PSVI contributions to applications, and consequently users of XML Schema processors have no standardized access to PSVI information.

Consequently, it can be seen that even though the XML Infoset was immensely important for moving XML from a syntax-only tree representations to a data model that can be used for complex higher-level technologies, such as APIs (such as DOM and SAX) and a way of addressing into XML documents (XPath and layers on top of it, such as XSLT and XQuery), the Infoset's design has also made it hard to move one step further and make XML's data

[2]See Section C.2 of the XML Schema Specification [16].

model easily extensible. It is important to note that our approach to changing the Infoset does not change any of the items or properties of the core Infoset, it simply puts them into a context where they can be reused more easily.

4 Requirements

Based on the possible applications for an extensible Infoset described in Section 2, the requirements for an improved version of the XML Infoset can be summarized as follows:

- *Provide a formal notation for Infoset extensions*

  Since Infoset extensions refer to other Infoset modules (the modules they are extending), it must be possible to formally (i.e., in a machine-readable way) identify the items of interest of an Infoset extension (the items of interest are the module itself, and the properties and the items it defines).

- *Provide a way to identify Infoset extensions*

  Infoset extensions always refer to other Infosets (either to the core Infoset, or to another Infoset extension), and thus it must be possible to identify Infoset extensions in a globally unique way.

- *Hierarchies of extensions must be possible*

  Infoset extensions form an acyclic graph, with the core Infoset as the root node, and all other Infoset extensions referring directly or indirectly to it. Thus, extensions can be extended, if necessary.

- *Identify the possible structural elements*

  The current Infoset specification only uses items and properties. It is necessary to decide whether these are the only structural elements allowed in Infosets, or whether other structural elements are possible.

- *Support datatypes for properties*

  On request, Cowan, one of the authors of the Infoset specification, stated that “every property has a type given, either ‘set of items’, ‘list of items’, or a simple type; and ‘no value’ is a special value, and ‘unknown’ is used when we are dealing with partial infosets.” However, this characterization of properties’ types is rather vague, and has to be clarified by introducing a proper type system for properties.

- *Provide a way to specify syntaxes for extensions*

  While an Infoset extension is sufficient to define the data model of a given extension of XML’s core data model, it is not clear how to represent data (i.e., instances of this data model). In order to be able to refer to representations of Infoset extensions, it is necessary that syntaxes can be identified somehow, so that software can make statements such as “this parser accepts syntaxes ‘a’ and ‘b’ for the Infoset extension ‘c’.”

Based on these requirements, we designed the *Extensible XML Information Set (EXIS)* described in the following section. It should be kept in mind, though, that at the time of writing the current design has still some open issues (listed in Section 7) and thus may undergo minor revisions before becoming stable.

5 The Extensible XML Information Set

In this section, we describe the *Extensible XML Information Set (EXIS)*, which is a method for formally describing extensions of Infosets. Revisiting the questions from Section 2, EXIS provides the following features:

- *How are Infoset extensions identified?*

  Infoset extensions are identified by namespace names (see Section 5.1.1 for details).

- *In which way(s) may the Infoset be extended?*

  Infoset extensions may add items and/or properties. Properties may be added to existing or newly defined items. Properties may be sets or sequences of references to items, or may use any XML Schema simple type (see Section 5.1.2 for details).

- *How are Infoset extension instances represented?*

  Infoset extensions may either be represented in a (extremely verbose) standard XML syntax, or in any other syntax for which a mapping from the Infoset's data model to syntactic constructs is defined. Syntaxes are defined by namespace names (see Section 5.2 for details).

In the following sections, we describe in detail how EXIS is defined. EXIS itself is defined as an XML Schema (EXIS modules are XML documents), which is augmented by some Schematron code for specifying additional constraints. The XML documents shown in the following sections have been shortened for clarity and to illustrate the features being discussed in the respective sections.

5.1 Data Model Extensions

An *EXIS module*, the definition of an Infoset extension, is identified by a name and defines a number of items and/or properties. In the following sections, these issues are described in detail.

5.1.1 Extension Names

An EXIS module must have a name. The name mainly serves two purposes, the first one is to make the EXIS module suitable for further extensions, and the second one is to enable implementations (for example, DOM or XSLT software) to communicate about the EXIS modules it is supporting. Since there is a well-known and widely used naming mechanism in the realm of XML technologies, the *XML Namespaces* [4] recommendation, EXIS uses namespace names as its way to uniquely name a certain module.

While namespace names are mostly used to identify sets of names for XML documents, this is not the only possible way to use namespaces: XML Schema uses namespaces to identify type names, introducing the concept of a *symbol space*[3], which is not very different from the *namespace partition* concept introduced by the XML Namespaces specification. In a similar fashion, we use a namespace name to identify the names of an EXIS module (we distinguish between item and property names).

[3]See Section 2.5 of the XML Schema Specification [16].

```
<infoset xmlns="...">
  <head>
    <name ns="..."/>
    <extends id="xlink" ns="..."/>
  </head>
  ...
</infoset>
```

Figure 1: Identification of Infoset extensions

In Figure 1 it is shown how the namespace name of an EXIS module is declared using the `name` element in the `infoset`'s `head` section. The `extends` element, on the other hand, references an EXIS module that is being extended, in this case it gets assigned the `id` `xlink` (for local references shown in the following section), and it is referenced through its namespace name specified using the `ns` attribute.

5.1.2 Items and Properties

An EXIS module defines items and/or properties extending the EXIS modules it is referencing[4]. Items can be easily defined to consist of properties, and properties are defined to either use *XML Schema Datatypes* [2], or some of the few special datatypes defined by EXIS itself.

For an illustration of an EXIS module, we show some excerpts from the EXIS module for the XLink data model [19]. The complete XLink EXIS module defines four information items and a total of 32 properties.

In Figure 2 it is shown how an item is defined. Each `item` element has a `name` through which it may be referenced (for example, by other EXIS modules), and a field for describing its purpose (non-formally, using plain text). The item is then defined by listing its properties (in this example, we show only one property). Each `property` element has a `name` through which it may be referenced, and a type, which is defined by the `property` element's content.

In this example, the property's type is an enumeration of values, defined by an XML Schema simple type[5]. Because the property may also have the value 'unknown', this case is also allowed by using EXIS's `special` element, which references the special values that properties may have.

EXIS also allows to define properties for items of modules which are being extended. Figure 3 shows an example of the syntax for this case. The `properties` element under the `body` level[6] contains all properties which are defined for items outside of the EXIS module (i.e., in modules which are referenced directly or indirectly by `extends` clauses).

Properties in this second case are defined exactly in the same way as they are defined directly inside of items, with one exception: Since there is no context to determine which item this property belongs to, this must be made explicit by one or more `item` elements which refer to the respective item(s) by using the `id` of the EXIS module they are defined in

[4]Choosing the terms carefully, it would be more appropriate to speak of item and property *types*, since items and properties appear in instances rather than the modules. However, so far we have not introduced the "type" terminology.

[5]We currently only allow anonymous types, so that the type must be defined within the `property` element and may not be referenced by name.

[6]Note that this is the same element as the `properties` elements contained in the `item` element shown in Figure 2. The parent of the `properties` element determines its use.

```
<infoset xmlns="..." xmlns:xs="...">
 <head>
  <name ns="..."/>
  <extends id="core" ns="..."/>
 </head>
 <body>
  <items>
   <item name="link">
    <desc>...</desc>
    <properties>
     <property name="type">
      <desc>...</desc>
      <xs:simpleType>
       <xs:restriction base="xs:token">
        <xs:enumeration value="simple"/>
        <xs:enumeration value="extended"/>
       </xs:restriction>
      </xs:simpleType>
      <special type="unknown"/>
     </property>
     ...
    </item>
    ...
   </properties>
  </items>
  <properties>...</properties>
 </body>
</infoset>
```

Figure 2: Definition of Items and Properties

(as declared in the `extends` clause) and the items' name(s).

The last interesting observation in this example is the type of the property. Since this property contains a reference to a `single` item, it uses a special `reference` element to declare this. Other allowed values of the `reference` element's `type` attribute are `sequence` and `set`, thus enabling properties to reference items in different ways. Because references often are restricted to certain kinds of items, all allowed items are listed by using `item` elements inside the `reference` element (if the `reference` element is empty, there is no such restriction of allowed items). Since the allowed item in our example is a local item (i.e., declared in the XLink EXIS module itself), it doesn't need to be referenced by using an `id`.

In summary, the syntax for EXIS modules is a simple schema language for defining items and properties, and includes a type system that mainly references XML Schema's datatypes, but also provides some types of its own. The actual syntax of EXIS modules is a little bit more complicated than shown in these examples, but the difference is negligible and is mainly caused by documentation features.

5.2 Data Model Syntaxes

A data model can be very useful for handling data (accessing it through an API such as DOM or manipulating using mechanisms such as XSLT or XQuery), but the main reason for XML's

```
<infoset xmlns="..." xmlns:xs="...">
 <head>
  <name ns="..."/>
  <extends id="core" ns="..."/>
 </head>
 <body>
  <items>...</items>
  <properties>
   <property name="xlink resource">
    <desc>...</desc>
    <item id="core" name="element"/>
    <reference type="single">
     <item name="resource"/>
    </reference>
    <special type="novalue"/>
   </property>
   ...
  </properties>
 </body>
</infoset>
```

Figure 3: Definition of Properties of external Items

success is its creation of a universally accepted interchange format. Thus, when dealing with Infosets and Infoset extensions, it is also important to think about representations of instances of these data models. In a very simple approach, it is possible to define a mapping from an Infoset to a syntax for it, and TOBIN and THOMPSON have done this already, defining an XML Schema for the core XML Infoset[7]. We have done a very similar thing, but extended it so that it handles EXIS modules. Thus, by using this mapping, there is an XML representation for every EXIS module. However, this approach leads to very large and hardly readable documents. Consequently, in many cases better syntaxes will have to be defined.

Going back to the example of XLink, it can be seen that its data model, as implicitly defined by the XLink specification (and made explicit in our EXIS module shown in the previous section), is represented by an XML syntax using attributes only. This syntax could (and, in a cleanly separated world of specifications, should) be defined separately from the data model. In this case, it would be rather easy to define parts of the XHTML syntax as simply another syntax for the XLink data model. Or, if XHTML required some extensions of the XLink data model (as seems to be indicated by the *HLink* [14] draft published by the XHTML working group), then the XLink EXIS module could have been extended, adding the required items and/or properties, thus rendering XHTML linking a proper superset of XLink, while still maintaining the shared foundation of the XLink data model.

In order to make this possible, it is necessary to make syntaxes identifiable. By doing this, a syntax specification could claim to be some representation of a data model (by referencing the namespace name of its EXIS module), and make itself globally recognizable through a namespace name. Applications could then not only list the data models they support, but also the syntaxes the recognize. A properly designed DOM EXIS module (see Section 6.1 for details) would allow users to query a DOM implementation for the supported data models and

[7]The schema is available at http://www.w3.org/2001/05/serialized-infoset-schema.

syntaxes. For example, while one implementation would support the XLink data model but only recognize XHTML syntax, another implementation would also recognize XLink syntax, thus enabling users to process XHTML as well as XLink documents and use a common link data model for both of them.

Going one step further, it would also be possible to allow alternative syntaxes for the core XML Infoset. While XML 1.0 syntax would certainly be required by every implementation, it would be possible to cleanly identify alternative representations of XML, such as the *Millau* XML encoding by Girardot and Sundaresan [10, 15]. While the majority of XML users is satisfied with XML 1.0 syntax, some users in high-volume and/or high-speed application areas have serious problems dealing with the verbose XML 1.0 syntax, and would be happy to have a portable way of integrating the handling of other syntaxes into application.

Currently, EXIS does not place any restrictions on a data model syntax, and it would be very hard to do so, seeing that it doesn't even have to be based on XML 1.0. However, EXIS requires that syntaxes representing EXIS modules properly reference the module by its namespace name, and declare a namespace name that can be used to identify the syntax.

6 Applications

The EXIS module syntax defined in the previous section is a good tool for cleanly modelling data models on top of XML's core data model, and thus may serve as a tool for specification writers. In fact, the Infoset often is quoted as being for "specification writers only". However, we think that the Infoset as it is today as well as the EXIS approach are important to XML users also, and thus XML users should be educated about this foundation of XML technologies. Apart from the knowledge of XML's data model, what benefit is there for users if EXIS gets adopted?

As a simple example, we can look at *Cascading Style Sheets (CSS)* [13]. Sadly, CSS is one of the few W3C specifications to completely ignore the Infoset. While this is understandable from the point of view that it has been developed for HTML rather than XML, other technologies have made the transition, in particular DOM3 which finally integrates the Infoset. If CSS made the transition to a proper support of the Infoset, then it would be possible to define an additional CSS module for EXIS, defining new selectors for selecting EXIS-defined items or properties, rather than only XML elements or attributes. However, since EXIS is more targeted at applications processing data than simply presenting it, we think that the most interesting application of EXIS are in this area, which is discussed in the following sections.

6.1 Document Object Model

The *Document Object Model (DOM)* [12] in its latest version, DOM3, is still under development. DOM3 introduces a number of new modules when comparing it to DOM2, and also for the first time provides an Infoset-based view of a document[8]. Following this modular structure, it would be possible to define a DOM EXIS module. This module would enable users to uses extended data models through a standard DOM interface, thus eliminating the need for implementing proprietary solutions for dealing with extended data models. Two of the most important interfaces of this module would enable users to query a DOM implementation for

[8]Previous DOM versions used a data model that had a number of minor deviations from the Infoset model.

the supported data models and syntaxes. Users could then use other interfaces to access and manipulate extended data structures through this module.

Reiterating the XLink/XHTML example from Section 5.2, an application could use an DOM EXIS module supporting the XLink data model as well as the XLink and XHTML syntaxes to uniformly handle linking information, being independent from the fact whether the linking information had been represented in XLink or XHTML.

Using another example, we could think of PSVI being rewritten as an EXIS module. The most obvious advantage of this approach would be that PSVI contributions would be accessible through the DOM EXIS module, while currently accessing PSVI contributions is impossible or entirely proprietary[9].

Defining a generic DOM EXIS module leaves open the possibility to define dedicated modules for popular data models, but it also makes it easier for application authors to access EXIS data without having to wait for dedicated DOM modules. Of course, if any data model would gain so much popularity that users would like to have a more specialized, more specifically designed interface, it would always be possible to define a new DOM module for it (such as DOM XLink or PSVI modules). However, software not supporting this module could still provide the information through the generic interface, while users of the dedicated DOM module would enjoy a better interface design.

6.2 XML Path Language

In the same sense as the XML Infoset serves as a foundation for a number of XML technologies, the *XML Path Language (XPath)* [6] does the same. XPath is based on the XML Infoset and serves as a foundation for XSLT, XML Schema, and XPointer. In its next version, *XPath 2.0* [1] and its accompanying data model [9], will also serve as the foundation for the upcoming *XML Query Language (XQuery)* [3] as well as for *XSLT 2.0* [11]. XPath defines a mapping of the Infoset's model of items and properties to a model of *nodes*, basically providing a slight simplification of the Infoset model[10]. XPath then defines a language for selecting parts of the node tree.

Currently, the node tree is hard-coded into the XPath data model, in XPath 1.0 it is based on the Infoset only, in XPath 2.0 on the Infoset and PSVI contributions. We suggest to extend this data model to include EXIS items and properties, reflecting the data model extensions described by EXIS modules in the XPath node tree. XPath already has some extension mechanisms, currently only for referencing extension functions which may be used in XPath expressions.

However, it is easy to think of ways to integrate EXIS into the XPath model without introducing backwards compatibility problems, because XPath still has some "open spots" in its syntax. The most conservative way would be to access EXIS information through built-in or even extension functions, but this would make the authoring and reading of XPaths using EXIS unnecessarily hard. Several other alternatives are possible (assuming that the data model of XPath 2.0 is extended to map EXIS items and properties to the XPath node tree):

- *Extension axes*

[9]For example, Apache's Xerces parser provides an interface to access PSVI via DOM, but using this proprietary interface makes it hard to port an application to another parser.

[10]XPath 2.0 also adds information from PSVI contributions, thus making it clear that non-core information may also be part of the XPath data model.

Some items in XPath are addresses through axes, in particular by using the `attribute` and the `namespace` axis. These axes navigate from the context node to the attribute or namespace node(s) associated with it. In a similar way, extension axes could navigate from nodes to associated EXIS nodes. However, since successful navigation not only requires the target type of the node to be known, but also the property through which to navigate, this scenario may lead to problems when looking at the most general cases.

- *Extension node tests*

  As the second component of a location step, a node test is specified. This node test may be either a *name test* or a *kind test*. A name test tests named nodes (for example, a comment node does not have a name), while a kind test tests for a specific node kind, in XPath 2.0 the defined kinds are `processing-instruction`, `comment`, `text`, and `node` (testing for any kind). The kind test syntax (identified by a trailing '`()`') could be extended to include namespace-prefixed names, identifying EXIS nodes.

In both cases, the resulting syntax would be rather concise, making it possible to extend the compact XPath syntax to a more flexible and extensible data model. However, it should be noted that some aspects of the Infoset (such as parent/child relationships and certain other kinds of relations established by properties referring to items) are buried deep in the model and the syntax of XPath, and it would certainly be a non-trivial task to extend XPath to fully support EXIS.

However, we think it could be worth the effort, since this would enable users to define their own data models using EXIS, and then use a very high-level language to address into this data. For example, using an EXIS-enabled XQuery, it would be trivial to write XQueries that query into XLink-authored data and return it as XHTML or vice versa. No programming at all would be necessary.

7 Open Issues

In the following list of open issues, we describe some of the questions in the design of EXIS and some of the questions of how to integrate it with other technologies, which are still unresolved.

- *Naming of "imported" items and properties*

  In every Infoset (with the single exception of the core Infoset), the items and properties being defined of this Infoset are the union of the Infoset itself and all the Infosets it is referring to directly or indirectly. Now the question arises how to name the imported items and properties. They could either be referred to by their original Infoset namespace name, or by the namespace name of the Infoset which imported the other Infoset(s). Both approaches ("importing" the names or using the original name) have advantages and drawbacks, and so far we haven't decided which way to go.

- *Reserved properties*

  Even though Infoset extensions are free to name the items and/or properties they are defining, it might make sense to reserve some properties for special purposes. For example, the `children` and `parent` property names could be reserved to always denote parent/child relations[11], and the `name` property name could be reserved to always denote an item's name.

These reserved properties could then be used to make possible certain easier ways to use an Infoset extension, for example the parent/child relationship making possible the hierarchical navigation in a tree (such as a DOM tree or an XPath node tree), or an item's name making it possible to use standard XPath name tests (a special kind of a node test) for selecting the item.

- *Extensibility of properties*

  In our current version, Infoset extensions can extend Infosets by adding items and/or properties. It would also be possible to think of an extension which only extends a property's datatype, for example allowing new values in a set of predefined values for a property. Unfortunately, XML Schema does not allow to extend simple types, so if we would like to keep XML Schema as the foundation for our datatypes, this type of extensibility would have to be defined as follows: "A property extended in an Infoset must be extended in a way so that the simple type of the original property is a legal restriction of the extended property." We are not sure whether this definition would be a good way to go.

We are sure that this list of open issues is not complete. In particular, when investigating in more detail the three possible application areas described in Section 6, CSS, DOM, and XPath, it is almost certain that new questions will arise.

Another issue is the question of politics. Creating and successfully pushing a new XML technology involves a lot of politics, and since EXIS is placed in the very heart of XML, it will certainly be hard to convince people to accept it. Irrespective of its technical qualities, many people will not like the idea of new DOM modules and a revised XPath 2.0. XPath 2.0 and XQuery 1.0 are currently under construction and will remain so for some time to come, even without major redesigns. So the question of how enthusiastic people feel about something like EXIS still remains to be seen, and will certainly be important for the future of our work.

8 Conclusions

In this paper, we have described a new way of looking at XML's data model. The *Extensible XML Information Set (EXIS)* is an extensible way of using the XML Infoset. While EXIS does not change anything about the items and properties of the core Infoset, it creates a framework for defining Infoset extensions, packaged into EXIS modules. The goal of this formalization of the Infoset is to make Infoset extensions better supported in higher-level XML technologies such as DOM and XPath. EXIS has been inspired by the efforts to cleanly design and handle a linking model for XML, and is currently still under constructing. In particular, the possibilities for its support in higher-level XML technologies still have to be investigated.

[11]Interestingly, this relation isn't necessarily symmetric. The core Infoset defines elements to be the parents of attributes, while the attributes are not children of the elements.

References

[1] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0. World Wide Web Consortium, Working Draft WD-xpath20-20021115, November 2002.

[2] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502, May 2001.

[3] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Working Draft WD-xquery-20021115, November 2002.

[4] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. World Wide Web Consortium, Recommendation REC-xml-names-19990114, January 1999.

[5] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-20001006, October 2000.

[6] James Clark and Steven J. DeRose. XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[7] John Cowan and Richard Tobin. XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[8] Steven J. DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[9] Mary F. Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. XQuery 1.0 and XPath 2.0 Data Model. World Wide Web Consortium, Working Draft WD-query-datamodel-20021115, November 2002.

[10] Marc Girardot and Neel Sundaresan. Millau: An Encoding Format for Efficient Representation and Exchange of XML over the Web. In *Proceedings of the Nineth International World Wide Web Conference*, pages 747–765, Amsterdam, Netherlands, May 2000. Elsevier.

[11] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20021115, November 2002.

[12] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Thomas Nicol, Jonathan Robie, Mike Champion, and Steven Byrne. Document Object Model (DOM) Level 3 Core Specification. World Wide Web Consortium, Working Draft WD-DOM-Level-3-Core-20021022, October 2002.

[13] Eric A. Meyer and Bert Bos. CSS3 Introduction. World Wide Web Consortium, Working Draft WD-css3-roadmap-20010523, May 2001.

[14] Steven Pemberton and Masayasu Ishikawa. HLink: Link recognition for the XHTML Family. World Wide Web Consortium, Working Draft WD-hlink-20020913, September 2002.

[15] Neel Sundaresan and Reshad Moussa. Algorithms and programming models for efficient representation of XML for Internet applications. *Computer Networks*, 39(5):681–697, August 2002.

[16] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. World Wide Web Consortium, Recommendation REC-xmlschema-1-20010502, May 2001.

[17] Norman Walsh. XML Linking and Style. World Wide Web Consortium, Note NOTE-xml-link-style-20010605, June 2001.

[18] Erik Wilde. Making the Infoset Extensible. In *Proceedings of XML 2002*, Baltimore, Maryland, December 2002.

[19] Erik Wilde. A Proposal for XLink Infoset Contributions. Technical Report TIK-Report No. 148, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, August 2002.