A Logico-Categorical Semantics of XML/DOM

Carlos Henrique Cabral Duarte
carlos.duarte@computer.org

BNDES
Av. República do Chile 100, Centro
Rio de Janeiro, RJ, 20001-970, Brazil

Universidade Estácio de Sá
Rua do Bispo 83, Rio Comprido
Rio de Janeiro, RJ, 20261-902, Brazil

Abstract. The efforts of the World Wide Web Consortium in defining and recommending the adoption of an extensible markup language, XML, and a document object model, DOM, have been received with enthusiasm by the software development community. These recommendations have been continuously adopted as a practical way to define and realise two party interaction. Here we describe an attempt at providing a logico-categorical semantics for these language and model, which seems to be useful in the rigorous development of open distributed systems.

Keywords: Extensible Markup Language, Document Object Model, Formal Methods, Distributed Systems, Software Engineering.

1 Introduction

The efforts of the World Wide Web Consortium (W3C) in defining and recommending the adoption of an Extensible Markup Language (XML) [8] and a Document Object Model (DOM) [6] have been received with enthusiasm by the software development community. XML is a subset of the Standard Generalised Markup Language (SGML) [14] meant to be used on the World Wide Web (WWW) [4]. Like the Hypertext Markup Language (HTML) [3], XML was designed not only to ease the development of software tools but also to interoperate with other WWW standards. DOM, on the other hand, is a public application programming interface (API) for manipulating HTML and XML documents. These language and model are now becoming *de facto* standards in the design and implementation of open distributed systems and frameworks.

XML documents describe semi-structured data objects possibly associated to some processing instructions. As textual specifications written in a markup language, there is a standard way of writing and reading each document. That is, there is a standard grammar and interpretation for XML documents. In addition, each software client reading a XML document is expected to present the same data to any application, despite their final visual apparency. This means that there is no standard presentation style for XML documents.

DOM defines a public interface for programatically accessing and manipulating XML documents and their parts. The model permits document parts to be created, modified and erased, while allowing applications to navigate from one document element to another, following its current structure. DOM is language independent and implementation neutral, but programming language bindings have been defined so as to enable the use of this model in developing real systems.

Both XML and DOM have actually been defined in a rigorous but rather informal manner. The applications of this language in developing distributed systems based on point to point communication, wherein the understanding of messages and other exchanged structured documents must be precisely the same in both communication ends, allied to the necessity of defining language bindings to make effective use of this model, suggest that it would be interesting to count upon a formal semantics as

their alternative definition. The existence of a logical semantics, for instance, would permit the rigorous verification of DOM based distributed system properties, as well as it would facilitate the implementation of automated testing tools.

XML and DOM congregate a particular set of characteristics which naturally leads to the development of a logical semantics that is also categorical, in the sense of applying Category Theory in Computer Science as advocated by Joseph Goguen [13]. More specifically, the first of these characteristics is that many XML documents represent the same data object, but are in fact different by definition. Since what really matters in their manipulation, due to the independence of presentation, is the object of their representation, this indicates the existence of an equivalence notion. Secondly, documents have structure and are inherently related by containment or inclusion. Their relationships are clearly functional, compositional, admit identity and can be used a basis for defining equivalence up to abstracting document representation. That is, we have all the ingredients for defining a category of XML document representations. In fact, what one can read out of these observations is that the emphasis in a definition of XML can be placed in capturing relations between objects (documents and their structure) rather than in simply capturing the objects themselves. This is precisely what the application of Category Theory is about. *Mutatis mutandis*, the same rationale above is valid concerning DOM, programming interfaces and their respective implementations.

Contribution. In this paper, we propose a logico-categorical semantics addressing both XML and DOM. This semantics, which can be regarded as our original contribution here, consists in a particular application of a first-order many-sorted branching time logical system developed as part of our previous work [10]. This seems to be a relevant contribution since it allows us to specify and reason about the specific class of open distributed systems underlying the web, contributing to better understand their static and dynamic aspects. We are not aware of other formalisms treating both W3C recommendations simultaneously.

Related work. A substantial number of formal models have been developed to clarify the application of XML in particular contexts. These can be roughly classified in mathematical, logical and categorical models. The main purpose of the algebras developed by by Frasincar, Houben and Pau [12], among many others, is the mathematical study of XML query formulation and optimisation. A logical notion of satisfiability is developed by Arenas, Fan and Libkin in [2] to verify the consistency of XML documents with respect to type definitions with constraints. A type theoretic approach is proposed by Brown, Fuchs, Robie and Wadler [5] to check whether or not documents conform with type definitions endowed with keys. Alagic and Bernstein [1] develop a categorical method for schema integration that applies to XML document type definitions. The complexity of the aforementioned models seems to increase with the number of potential applications.

Organisation. Sections 2 and 3 provide brief descriptions of XML and DOM respectively; Section 4 outlines the proposed logico-categorical semantics; Section 5 presents some conclusions and prospects for future research.

2 Brief Description of XML

Physically, each XML *document* is a textual specification composed by syntactic unities called *entities*. Entities may contain unparsed and parsed data, the latter being formed by markups or other specific symbols. Define the logical structure of each document: *declarations*, *comments*, *elements*, *character references* to the ISO/IEC 10646 set and *processing instructions* (PIs) to potential applications. These logical structures are represented using markup tags that are delimited by < and > but which may also internally use

other punctuation marks. The *document entity* contains the whole textual specification, which necessarily includes a *root element*.

Elements define data objects, which are identified by *names* (tokens beginning with a letter or some punctuation marks). Names are used as part of tags to delimit the respective data object contents. Elements may have *attributes*, which can only hold plain values. Conversely, the organisation of element contents is forest like, meaning that they may also contain lists of disjoint nested elements. An example document appears in Fig. 1.

```
<?xml version="1.0"?>
<!-- card.xml -->
<CARD>
<HOLDER>CARLOS H C DUARTE</HOLDER>
<NUMBER>1234567890123456</NUMBER>
<BRAND>Supercard</BRAND>
<EXPIRYDATE>31/02/2003</EXPIRYDATE>
</CARD>
```

Fig. 1: A credit card XML document.

The CARD element in Fig. 1 is structured, having HOLDER, NUMBER, BRAND and EXPIRYDATE as members. The first two lines of that document contain a declaration and a comment, respectively to allow proper automated processing and to facilitate human comprehension. Not all XML documents have such a simple structure. Differently from CARD, the XML document in Fig. 2 is more complex, defining a BOOK with an empty EDITOR element, without specified value. In addition, the BOOK element is labelled by the value of an attribute, LABEL. In general, attributes are used to hold meta-data as is the case of LABEL.

The structure and content of classes of documents may be specified using type definitions. A *document type definition* (DTD) is named and may have a specification separated from documents of that type. Such definitions can be referenced in client documents using the DOCTYPE markup. A DTD consists in a list of declarations of notations, fixed entities, entity types or attribute lists. Element contents

```
<?xml version="1.0"?>
<!-- book.xml -->
<!DOCTYPE BOOK SYSTEM "book.dtd">
<BOOK LABEL="BROOKS1995">
<TITLE>The Mythical Man-Month:
       Essays in Software Engeneering</TITLE>
<AUTHOR>Frederick P. Brooks Jr.</AUTHOR>
<EDITION>2nd</EDITION>
<EDITOR/>
<PUBLISHER>Addison-Wesley Pub Co.</PUBLISHER>
<YEAR>1995</YEAR>
</BOOK>
```

Fig. 2: A book XML document.

and attribute values can be defined as blocks of characters or using a variant of regular expression syntax. Element members and their order can be determined. DTDs may be parameterised, with parameters marked with a preceding #, and have conditional sections, which are considered or not as part of the definition according to the fulfilment of defining conditions. An example DTD, satisfied by a class of book documents including that in Fig. 2, is presented in Fig. 3.

```
<?xml version="1.0"?>
<!-- book.dtd -->
<!ELEMENT BOOK
   (TITLE, AUTHOR, EDITION, PUBLISHER, YEAR) >
<!ELEMENT TITLE     (#PCDATA) >
<!ELEMENT AUTHOR     ANY      >
<!ELEMENT EDITION   (#PCDATA) >
<!ELEMENT EDITOR     ANY      >
<!ELEMENT PUBLISHER  ANY      >
<!ELEMENT YEAR      (#PCDATA) >
<!ATTLIST BOOK
          LABEL CDATA #REQUIRED >
```

Fig. 3: A DTD defining valid book documents.

Using the ENTITY markup, definitions of entities can be specified, whose name may be placed between the delimiters & and ; to imply in a content expansion when the document is processed. These may rely on external definitions, which are specified through the SYSTEM markup followed by a literal (quoted string)

pointing to the actual location of the defining document. An example of a composed document appears in Fig. 4.

```
<?xml version="1.0"?>
<!-- order.xml -->
<!ENTITY CHCDCARD SYSTEM "card.xml">
<!ENTITY MYTHBOOK SYSTEM "book.xml">
<ORDER>
&CHCDCARD; &MYTHBOOK;
<PRICE>39.00</PRICE>
<CURRENCY>USD</CURRENCY>
</ORDER>
```

Fig. 4: A book order XML document.

A XML document is said to be *well-formed* only if satisfying the following conditions:

1. it complies with the XML grammar specified in [8];
2. it satisfies some well-formedness constraints, such as to have matching start and end tags in the definition of each element;
3. all referenced entities are well-formed.

As a consequence of the third item above, documents cannot be directly or indirectly recursive. To any violation of well-formedness in reading an XML document corresponds a *fatal error*, which is not recoverable. The document in Fig. 1, for instance, is well-formed. A document is said to be *valid* if it is well-formed and complies with a given DTD. To each violation of validity in reading an XML document corresponds a recoverable *error*. For example, the document in Fig.2 is valid.

Although XML was designed to give rise to documents legible by humans, the language definition is based on client document processors and corresponding computer applications. The error handling treatment mentioned above is to be followed by any client and application. When these are implemented as part of a WWW browser, the W3C suggestions concerning the definition of presentation style separated from each document should be taken into account. The W3C has even proposed a stylesheet language (XSL) [9] and style transformations (XSLT) [7] to address this issue, but the study of the respective recommendations is out of the scope of our current work.

3 Brief Description of DOM

DOM defines a set of specifications for representing documents and their composition. Each document is represented as a tree of passive objects, not just as a data structure, so that it can present some observable behaviour. Documents and their components are addressed in an object-based way: they may have attributes representing state and methods that give rise to their behaviour. The respective hierarchy of class interfaces can be regarded as if organised by an inheritance relation, although this view is not mandatory. Note that, viewed in this way, DOM defines an abstract class structure that has to be mapped into a real programming language and refined with the implementation of each interface in order to support real applications. It is in this sense that DOM is considered just as an API.

The current DOM definition does not specify how entire documents are created [6]. Assuming that a certain document object exists, the creation of its components follows the so-called factory pattern, which is specified as a method in the scope of the document object specification. For instance, to create a `CARD` element within the scope of an `ORDER` document, it would be necessary to call the `createElement` method of `ORDER` providing `CARD` as an argument.

Each component of a document and documents themselves are regarded as complying with a primary specification called `Node`. It defines querying methods, such as `nodeType`, a type being that of documents, entities and other constructs; `nodeName`, which depends on the node type; and `nodeValue`. The specification also defines the methods `nodeParent`, `childrenNodes` and `ownerDocument`, all with intuitive mean-

ing; `firstChild` and `lastChild`, for recovering the first and the last elements in the list of nodes returned by `childrenNodes`; `previousSibling` and `nextSibling`, for navigating in the structure where the node may be connected to; `hasChildNodes` to verify if the node is structured, and `attributes`, returning any node attributes. A set of updating methods with intuitive behaviour is also defined: `insertBefore`, `replaceChild`, `removeChild`, `appendChild`, and `cloneNode`.

Apart from the features inherited from `Node`, the `Document` specification defines `doctype`, which points to a possibly existing DTD; `element`, pointing to the root element of the document; and `implementation`, which is explained below. In addition to those inherited from `Node`, the specification also defines factory methods for creating the various different types of constructs listed in the previous section, such as elements and declarations.

Although the main purpose of XML is the definition of entire documents, in their manipulation it is often found convenient to deal with document parts, which would not strictly satisfy the main production rule of the XML grammar. DOM specifies `DocumentFragment` as a subtype of `Node` endowed with copy/modify/paste methods, which allow their use in the role of clipboards. Other auxiliary object types with an intuitive semantics, such as `NodeList` and `NamedNodeMap`, are also defined by DOM.

The W3C recommendation concerning DOM is divided into three parts addressing the DOM core, XML and HTML. An interface for querying a specific implementation concerning the supported version of either of these languages is supplied as part of the model, `DOMImplementation` with method `hasFeature`. The distinguished support provided by each kind of implementation derives from the interpretation of tags, which is fixed in HTML and variable in XML. The model also specifies an interface for error handling, `DOMException` with a table of exception codes. These two implementation related issues are out of the scope of the present study.

Other standard document components are captured in the DOM core through the specifications `Element`, `Attr`, `Text` and `Comment`. The XML specifics are represented as part of the interfaces `Notation`, `CDATASection`, `DocumentType`, `Entity`, `EntityReference` and `ProcessingInstruction`. These directly inherit the features of `Node`, the exceptions being `Text`, `Comment` and `CDATASection`, for which this relation is indirect due to the existence of `CharacterData`, a virtual interface to support the HTML specifics as well. A diagrammatic representation of the abstract class structure implied by DOM appears in Fig. 5.

4 Semantics of XML/DOM

4.1 Core Semantics of XML

We develop our work based on the insight that documents which are not well-formed do not have a failure free semantics. To propose a failure semantics that could capture the subtleties of ill formed documents would be a very complex task, which is not our purpose to develop here. Therefore, we only deal with well-formed documents in this paper. We also take advantage of the fact that valid documents are well-formed and treat all of them without distinction. That a document is valid complying with a certain DTD is reflected here just in the additional obligation to ensure that all type definition restrictions are satisfied by the document interpretation.

A careful inspection of the XML notions shows that semantically not all of them have the same status. For instance, PIs do not affect how a document is understood, but just how it is to be dealt with. Their treatment clearly falls into the domain of pragmatics. The definition of entities, notations and comments serve just as syntactic sugar to ease human comprehension or automated processing. Consequently, these are ignored in the sequel.

The semantically rich part of XML remaining to be treated here is that of elements with their types and contents; attributes with their plain values; and entire documents. These def-

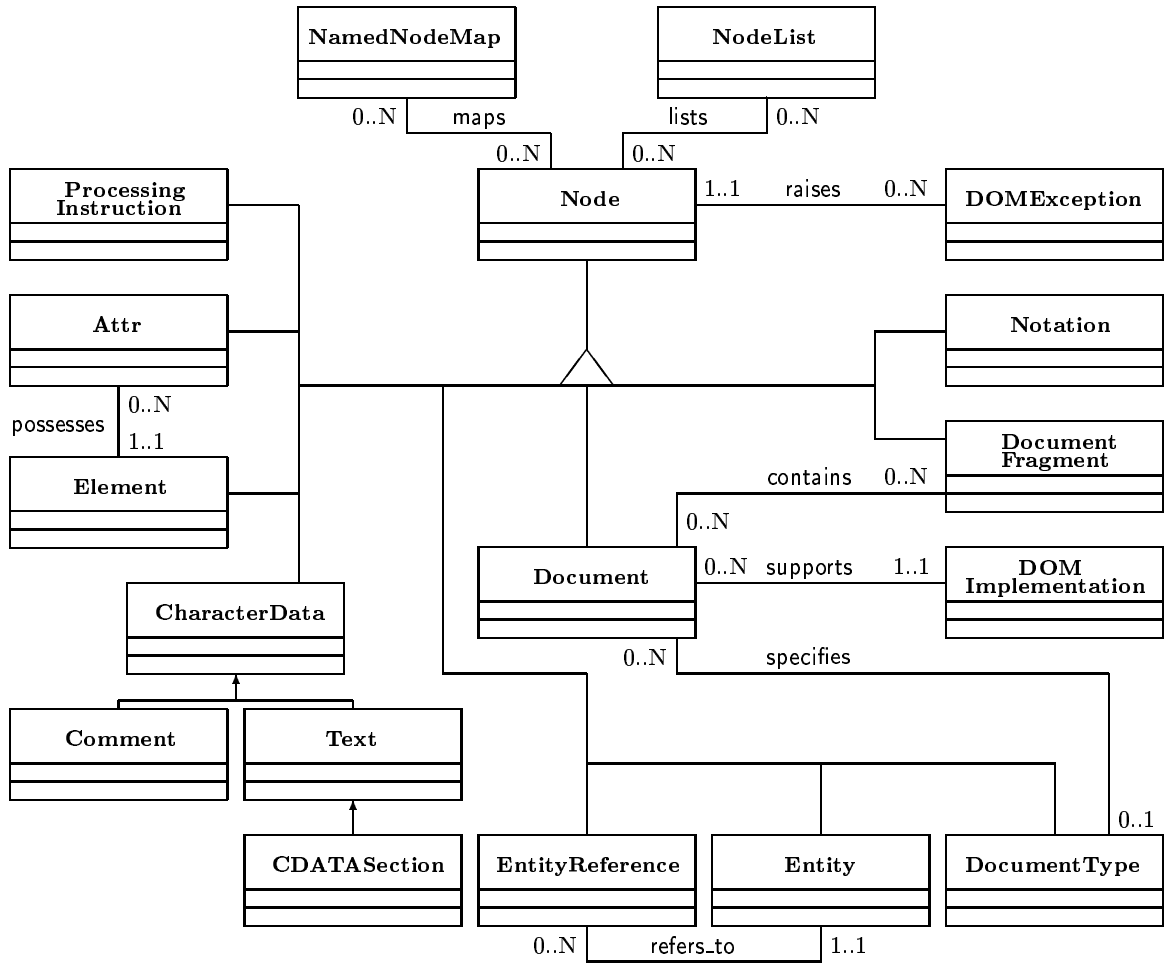

Fig. 5: Abstract class structure defined by DOM.

initions *a priori* do not have associated behaviour and can be represented through classical first-order theory presentations.

We provide in Fig. 6 two theory presentations describing elements and values. These rather simple presentations become important in the interpretation of other XML constructs. The presentation ELEMENT specifies a sort symbol to stand for the universe of elements, elem, with $\perp$ denoting the bottom element. To each element may be associated a type through a specific function, *type*. The constants of sort type serve as names for classes of elements and additional restrictions can be placed on these elements depending on whether or not there is a type definition constraining that particular class. VALUE specifies not only a sort symbol representing the plain values of attributes and some elements, val, but also a function *nil* denoting a distinguished bottom value.

It is in the interpretation of structured data objects that most of the complexity of XML lies. In our logical view of XML inspired by DOM, elements and values are generically called nodes. The defining presentations of these two notions are imported in the interpretation of nodes so that the respective symbols can be used in our axioms. Nodes have a forest like ordered structure. Due to this fact, we also import a theory presentation defining natural numbers (which is omitted here), nat, and specify two functions, *void* and *member*, denoting respectively a bottom node and an ordered membership function. For each node x and natural number n, $member(x, n)$ returns the n-th member of x or *void* if it does not ex-

ist. Each node may either hold an element and possibly define finitely many member nodes through *member* or hold a plain value, the reason for including in NODE the functions *node_elem* and *node_val*. The resulting presentation appears in Fig. 7.

Nodes may be atomic or have some structure. Axioms (3.1-3.3) define the relationships between bottom nodes, elements and values. Non-bottom nodes must be associated either to a value or to an element (3.5) and structured nodes necessarily stand for elements (3.6). In order to capture the structural properties of nodes, we introduce in NODE an auxiliary symbol *heir* and the corresponding defining axioms. For a pair of nodes x and y, $heir(x, y) = 1$ iff y is a descendant of x in its tree like structure. Axiom (3.10) says that the members of a node define its direct descendants. Axiom (3.11) specifies that the descendants of a node are also defined by the descendants of its members. These two axioms provide an inductive definition for *heir*. As constraints, we have (3.9) saying that nodes cannot be directly recursive; (3.12) stating the no node sharing property of trees; and (3.13) requiring that only descendants allowed by our inductive definition be considered as such.

Elements may be associated to attributes in a many to one relationship. The presentation ATTRIBUTE in Fig. 8 defines an *attr_elem* function capturing this relationship. The function *attr_val* returns a value for each attribute.

Concluding our definitions, we formalise documents in Fig. 8. We consider that they are endowed with a function to return each

Presentation ELEMENT (ELEM)
sorts elem, type
functions $\bot : \rightarrow$ elem
$type$: elem $\rightarrow$ type

Presentation VALUE (VAL)
sorts val
functions $nil : \rightarrow$ val

Fig. 6: Semantics of Elements and Values.

Presentation NODE
imports NAT, ELEM, VAL
sorts node
functions $void : \rightarrow$ node
$node\_val$: node $\rightarrow$ val
$node\_elem$: node $\rightarrow$ elem
$member$: node $\times$ nat $\rightarrow$ node
$heir$: node $\times$ node $\rightarrow$ nat
axioms x, y, z : node; e : elem; m, n : nat

$node\_val(void) = nil$ (3.1)
$node\_elem(void) = \bot$ (3.2)
$member(void, 0) = void$ (3.3)
$\exists n \cdot member(x, n) = void$ (3.4)
$x \neq void \rightarrow$ (3.5)
$(node\_elem(x) = \bot \leftrightarrow node\_val(x) \neq nil)$
$member(x, 1) \neq void \rightarrow node\_elem(x) \neq \bot$ (3.6)
$member(x, m) = void \rightarrow$ (3.7)
$(\forall n \cdot m < n \rightarrow member(x, n) = void)$
$heir(x, y) = 0 \vee heir(x, y) = 1$ (3.8)
$heir(x, x) = 0$ (3.9)
$(\exists n \cdot member(x, n) = y \wedge y \neq void) \rightarrow$ (3.10)
$heir(x, y) = 1$
$(\exists n \cdot member(x, n) = y \wedge y \neq void) \rightarrow$ (3.11)
$(\forall z \cdot heir(y, z) = 1 \rightarrow heir(x, z) = 1)$
$heir(x, y) = 1 \wedge heir(x, z) = 1 \rightarrow$ (3.12)
$y = z \vee heir(y, z) = 1 \vee heir(z, y) = 1$
$heir(x, y) = 1 \rightarrow (\exists n \cdot member(x, n) = y) \vee$ (3.13)
$(\exists z \cdot heir(x, z) = 1 \wedge heir(z, y) = 1)$

Fig. 7: Semantics of Nodes.

document root node. According to (5.1), documents must define a tree like structure, even if consisting only in the *void* node.

The semantics of XML can be formalised as an amalgamation of all the previous theory presentations. This construction can also be explained in categorical terms, considering the **imports** statement in each presentation as a definition of a family of identity morphisms including the named objects into their enclosing presentations. The resulting structure can be better visualised in a diagrammatic manner through the objects and arrows in Fig. 9.

4.2 Semantics of XML Documents

Our core XML semantics can be regarded as a meta-logical construction when it is used as a framework for capturing the semantics of par-

Presentation ATTRIBUTE (ATTR)
imports ELEM, VAL
sorts attr
functions $attr\_elem$: attr $\rightarrow$ elem
$attr\_val$: attr $\rightarrow$ val

Presentation DOCUMENT (DOC)
imports NODE, ATTR
sorts doc
functions doc_node : doc $\rightarrow$ node
axioms d : doc
$member(doc\_node(d), 1) = void$ (5.1)

Fig. 8: Semantics of Attributes/Documents.

ticular documents. In order to illustrate this application, we return below to the book order example proposed in Section 2.

We assign a distinct theory presentation to each document, which imports the whole XML semantics. Values mentioned in the document give rise to constants of sort val. Each type tag is represented as a constant of sort type. The elements, attributes, and their hierarchical organisation are captured setting the functions doc_node, $member$, $node\_elem$, $node\_val$, $attr\_elem$, $attr\_val$ and $type$ accordingly. The result is not a compact presentation, but it is an exact representation.

To illustrate this interpretation, we present in Fig. 10 the BOOK document of Section 2 annotated with labels identifying the kind of each construct therein. We assume the existence of constants x_i of sort node, $1 \leq i \leq 12$; v_j of sort val, $1 \leq j \leq 6$; t_k of sort type, $1 \leq k \leq 7$; and a_1 of sort attr. Values are represented underlined and nodes are surrounded by boxes.

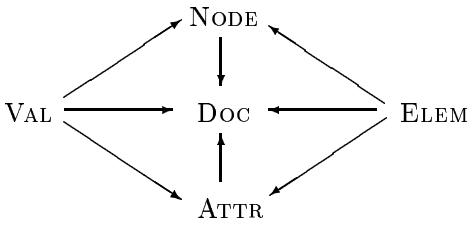

Fig. 9: Categorical semantics of XML.

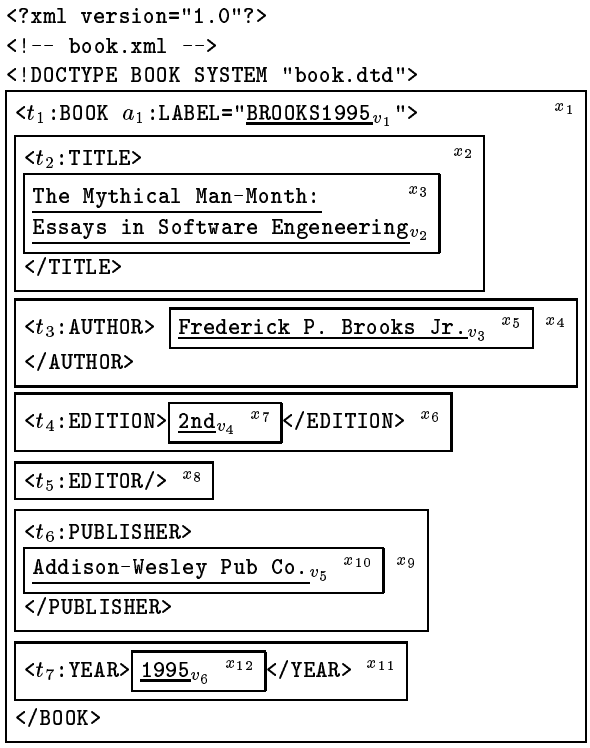

Fig. 10: Annotated BOOK document.

Figure 10 comprises a document which gives rise to a BOOK presentation containing an **import** DOC statement and axioms specifying as distinct and non-bottom the constants $\{d_1 : \mathsf{doc}; x_i : \mathsf{node}; v_j : \mathsf{val}; t_k : \mathsf{type}; a_1 : \mathsf{attr}\}$. The presentation specifies the root book node:

$$doc\_ent(d_1) = x_1 \quad (1)$$

It is important to mention that we avoid to over constrain doc_ent in order to allow the semantic composition of documents, as illustrated below. It would be troublesome, for instance, if above x_1 were required to be the only node in the image of this function.

The specified book element e_1 has just one attribute a_1 (LABEL) with value v_1 (BROOKS1995). This is mapped into the following axioms:

$$attr\_val(a_1) = v_1 \quad (2)$$
$$attr\_elem(a_1) = e_1 \quad (3)$$
$$\forall a_2 : \mathsf{attr} \cdot attr\_elem(a_2) = e_1 \rightarrow a_2 = a_1 \quad (4)$$

Now we have to deal with the complex structure of BOOK. Due to the limited graphical reso-

lution of this paper, it was impossible to make explicit in Fig. 10 the elements defined by the document, but we equally need to assume the existence of the respective constants e_l of type elem, $1 \leq l \leq 7$, in order to formalise these document components. The structure of BOOK is interpreted with the help of the previously introduced functions *member* and *node_elem*:

$$node\_elem(x_1) = e_1 \quad (5)$$
$$member(x_1, 0) = x_2 \quad (6)$$
$$\ldots$$
$$member(x_1, 5) = x_{11} \quad (7)$$
$$member(x_1, 6) = void \quad (8)$$

Note that (8) is required in order to restrict the interpretation of BOOK to the nodes in Fig. 10. Without this kind of equation in the interpretation of each specification, unbound (open) documents would be admissible. Although interesting, this kind of document is not not acceptable in XML. Our core semantics, however, regards them as acceptable because the axioms in Fig. 7 only require the existence of an unspecified natural number limiting the elements of each document (3.4).

The equations above capture only the two first hierarchical levels of the book document, containing the elements from TITLE to YEAR. To represent the whole document, we have to iterate this process until plain values are found. For example:

$$node\_elem(x_2) = e_2 \quad (9)$$
$$member(x_2, 0) = x_3 \quad (10)$$
$$member(x_2, 1) = void \quad (11)$$
$$node\_val(x_3) = v_2 \quad (12)$$
$$member(x_3, 0) = void \quad (13)$$

These equations say that the node x_2 (TITLE) defines an element e_2, which in turn contains a node x_3 holding value v_2, the book title.

Most of what remains to be interpreted of BOOK is uninteresting, a simple repetition of the cases above. The only exception is the interpretation of the empty element EDITOR, which is performed as follows:

$$node\_elem(x_8) = e_5 \quad (14)$$
$$member(x_8, 0) = void \quad (15)$$

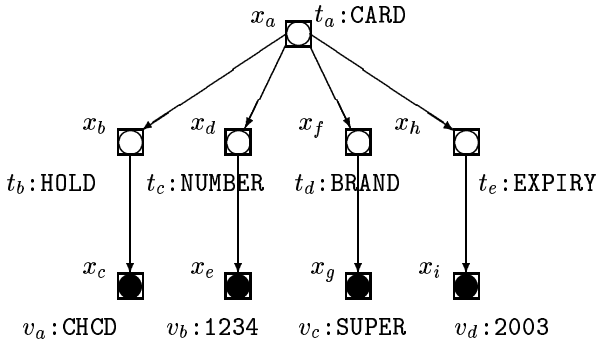

Fig. 11: Structure of the card document.

The process above may be used to produce an interpretation for any XML document. For instance, CARD can be interpreted as presented in Fig. 11 through a diagrammatic notation, where nodes are represented using squares, elements as empty circles and values as filled ones. Although we have omitted this detail from our examples, the semantics of DTDs can be produced in the same way. This process can be formalised through a functor mapping each XML syntactic construct into new components of a theory presentation, which is initially empty but is gradually augmented by the interpretation of each document component. We omit here the definition of this functor, whose signature is $[\!] : \texttt{XML} \times \mathbf{Pres} \to \mathbf{Pres}$.

4.3 Compositional Semantics

The semantics above determines a separate interpretation for each given XML document. One may wonder if some sort of compositionality is present in this kind of interpretation. The answer is affirmative.

To illustrate this, we take the presentations resulting from the interpretation of CARD and BOOK described in Section 4.2 and attempt to compose these objects using presentation morphisms. We recall from [11, 13] that the presentations and morphisms (functions) adopted here determine a category **Pres**. This allows us to make use of helpful categorical constructions. For instance, the commutative diagram in Fig. 12 describes the composition of CARD and BOOK resulting in $\text{CARD} \otimes_{\text{XML}} \text{BOOK}$.

The construction of CARD⊗BOOK can be explained by analogy with the process of defining BOOK from BOOK and DOC. We mentioned that DOC is imported into BOOK through an identity morphism (o_2) such that: (i) a unique new document with its root node are required to exist in the target presentation (creation); (ii) all the constants assumed to exist only in the target presentation are distinguished from the image of the values defined in the source (completion) and (iii) the root of the top level document becomes a member of the new document root (demotion). The definition of CARD is performed based on a morphism of the same family (o_1). Apart from obeying these rules with respect to DOC, CARD⊗BOOK must also be constructed in a minimalist way without equalising the image of the constants in CARD and BOOK. This is ensured by computing a *pushout*, a categorical construction defined here by the morphisms σ_1 and σ_2 of the same family. These are defined in such a way that the images of DOC imported through CARD and BOOK are collapsed in a single entity when they are put together. So we end up with just one *member* symbol in CARD⊗BOOK, which captures the complex node structure of both documents. Since the interpretation of each composite object is defined by the individual interpretations of its components when combined in the way suggested by our semantics, we have compositionality.

It is interesting to note that we can iterate this construction and obtain an interpretation for another of our example documents, ORDER, which contains elements defining the price and currency adopted in an electronic transaction. These elements are present only in ORDER, the interpretation of ORDER depicted in Fig. 12.

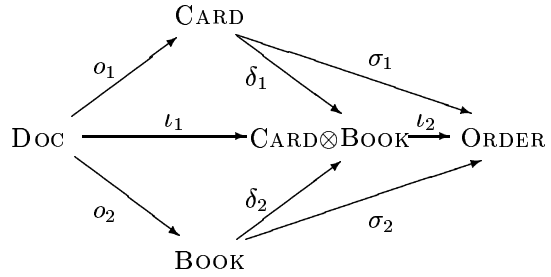

Fig. 12: Formal Semantics of our Example.

Presentation DOMATTR
imports BOOL, DOC
attributes $name, value$: val
actions $getName, isName$(val), $getVal, isVal$(val), $getSpec, isSpec$(bool), $setVal$(val)
axioms v : val

beg $\rightarrow value = nil$	(6.1)
$setVal(v) \vee (value = v \wedge \mathbf{X}(value = v))$	(6.2)
$setVal(v) \rightarrow \mathbf{X}(value = v)$	(6.3)
$getName \wedge name = v \rightarrow \mathbf{X}(isName(v))$	(6.4)
$getVal \wedge value = v \rightarrow \mathbf{X}(isVal(v))$	(6.5)
$getSpec \wedge value = v \rightarrow \mathbf{X}(isSpec(v \neq nil))$	(6.6)

Fig. 13: Semantics of DOM attributes.

Another issue related to compositionality is the possibility of testing the structural equivalence of documents. Again it is feasible to rely on categorical techniques in this case. Given two documents, a positive answer for this kind of question is found if it is possible to determine two standard theory morphisms mapping their interpretations into each other in such a way that the resulting diagram commutes.

4.4 Core Semantics of DOM

When we come to capturing the semantics of DOM, the rationale concerning the use of presentations and morphisms is also applicable, but because DOM objects have observable behaviour, we are obliged to make use of the temporal aspects of the adopted logical system.

We define a theory presentation for each DOM object type. In each of these presentations, the XML semantics is imported; the attributes of the DOM interface are interpreted as attribute symbols and methods are captured using action symbols, with existing parameters represented as action arguments. Note that these steps are performed to provide a language rich enough for representing the dynamic properties of the respective DOM objects through some presentation axioms.

In Fig. 13 we present the interpretation of attributes in DOM (Attr). The presentation says that attribute values are initially undefined (6.1), only change if requested (6.2) and modify subsequently after a request (6.3); and that attribute names, values and status are returned immediately after any query (6.4-6.6).

5 Final Remarks

In this paper, we have outlined a logico-categorical semantics for both XML and DOM, which may be regarded as a formal alternative to their standard informal semantics definitions. Our semantics is structured in terms of theory presentations of a first-order many-sorted branching time logical system with equality, which were derived from the W3C recommendations and should be particularised in the interpretation of each document or application with their specific details. Using this work, it becomes possible to reason about the static and dynamic aspects of web-based open distributed systems and frameworks, manually or by using automated support. We believe that the most promising directions for applying this research will the study of integrating heterogeneous sources of information over the web and the design of web-based multimedia systems.

Many other formal models of XML have been proposed in the literature with specific purposes. Most of these are based on the use of mathematical or logical constructions to capture the studied tree like structure of XML documents (e.g. [2, 5, 9, 12]). These differ from our work due to the additional proof-theoretic treatment given to the XML notions here, which we consider better suited to support the rigorous development of software systems in general. The abstract categorical approach proposed in [1] keeps many similarities with our ideas, specifically related to the the use of morphisms to describe the structure and collective behaviour of XML based software systems. In particular, because our work is based on theory presentations and their morphisms, it is not difficult to see that the induced morphisms (functors) between the categories of models of these theories possess the reverse direction of the given morphisms, thus complying with the definitions in that work.

References

[1] S. Alagic and P. A. Bernstein. A model theory for generic schema management. In *Proc. 8th International Workshop on Database Programming Languages (DBPL'01)*, 2001.

[2] M. Arenas, W. Fan, and L. Libkin. On verifying consistency of XML specifications. In *Proc. 21th Symposium on Principles of Database Systems (PODS'02)*, 2002.

[3] T. Berners-Lee and D. Connolly. Hypertext Markup Language (HTML) – 2.0. RFC 1866, MIT/W3C, November 1995.

[4] T. Berners-Lee et al. The World-Wide Web (WWW). *Communications of the ACM*, 37(8):76–82, 1994.

[5] A. Brown, M. Fuchs, J. Robie, and P. Wadler. MSL – a model for W3C XML schema. In *Proc. 10th International World Wide Web Conference (WWW'01)*, pages 191–200. ACM Press, 2001.

[6] W. W. W. Consortium. Document object model (DOM) level 1 specification. W3C recommendation, `http://w3c.org`, October 1998.

[7] W. W. W. Consortium. XSL transformations. W3C recommendation, `http://w3c.org`, November 1999.

[8] W. W. W. Consortium. Extensible markup language (XML) 1.0 (second edition). W3C recommendation, `http://w3c.org`, October 2000.

[9] W. W. W. Consortium. Extensible stylesheet language (XSL) 1.0. W3C recommendation, `http://w3c.org`, November 2000.

[10] C. H. C. Duarte. *Proof-Theoretic Foundations for the Design of Extensible Software Systems.* PhD thesis, Department of Computing, Imperial College, London, UK, 1998.

[11] J. Fiadeiro and T. Maibaum. Temporal theories as modularisation units for concurrent systems specification. *Formal Aspects of Computing*, 4(3):239–272, 1992.

[12] F. Frasincar, G.-J. Houben, and C. Pau. Xal: An algebra for XML query optimization. In Z. Zhou, editor, *Proc. 13th Australasian Database Conference (ADC'2002)*, 2002.

[13] J. A. Goghen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.

[14] I. S. Organization. Information technology document description and processing languages. Technical Report ISO 8879:1986 TC2, 1998.