The NXT Object Model

Stefan Evert,
Jean Carletta, Timothy J. O'Donnell, Jonathan Kilgour,
Andreas Vögele, and Holger Voormann

Version 2.1
24 March 2003

Contents

1 Introduction

This document describes the **NXT object model**, a mathematical formalisation of the data structures needed for the representation of richly annotated multi-modal language data. Its technical perspective focuses on the objects and their relations, as well as constraints on the complexity of structures that help to keep the data manageable and searchable. This underlying representation may be quite different from the (interpretation of the) information that is represented in it, which is the subject of the **NXT data model** (partly specified in [1]).[1] See [2] for an overview and explanation of the relation between object model and data model.

Multi-modal data is usually and obviously described by reference to a common **timeline**. Although sufficient for the immediate annotation of events and their temporal relations, this simple approach is not sufficient for the full complexity of linguistic annotations. The major challenge that such annotations present is that they combine **hierarchical** and **sequential structure**, in various ways that may seem contradictory at times. The standard representation of such data takes the form of an ordered tree, as in the XML object model. It is thus hardly surprising that XML has become a popular format and *de facto* standard in the domain of natural-language processing (NLP). However, complex annotations that combine different linguistic levels and different modalities will rarely fit into this limited framework.

At an abstract level, an XML document is simply an **ordered tree** of element nodes, to which attribute nodes are attached. While element nodes define the *structure* of an XML document, their attributes store the actual *data* in the form of character strings.[2] This abstract view is known as the XML document object model, or DOM. The files with lots of angle brackets that most people associate with the term XML are merely a **serialisation** format for data storage and exchange.

Experience in the MATE project [3] and other research has shown that linguistic data, including the complex interactions that are observed in the domain of multi-modal communication, can often be represented as a collection of intersecting hierarchies, which draw on the same basic annotations (often referred to as transcription layers) in different ways. Some of these hierarchies may be structurally independent, but are synchronised through a common timeline. The NXT object model is an extension of the formalism of ordered trees to such multiple intersecting hierarchies, maintaining the integration of hierarchical and sequential structure as far as possible.[3]

While an implementation of the NXT object model does not necessarily depend on XML technology, it is natural to use XML as a storage and exchange format. Therefore, Section 6 defines an XML extension syntax for the serialisation of a NOM corpus into a collection of XML files. Note that this specification does not make provisions for the storage of metadata (referring both to the "implicit" metadata summarised in Section 4 and to additional information about the corpus), which is described in [1].

[1] As an example, formally identical tree structures can represent a hierarchy of annotation codes or the syntactic analysis of a sentence. Although these examples are highly dissimilar from the perspective of a user, there is no need for a distinction in the object model.

[2] This short account glosses over other node types such as comments, text nodes, and processing instructions. These nodes are particular to the use of XML as a human-readable document format and have no impact on the information stored in an XML document (for instance, text nodes can equivalently be represented as elements with a single attibute holding the content of the text node).

[3] This is in marked contrast to e.g. directed acyclic graphs (DAGs), which can be thought of as generalised hierarchies and abandon sequential ordering altogether.

One advantage of our approach is that existing XML resources can easily be integrated into a NOM corpus, requiring only minor changes to the original files. In order to maximise compatibility, we have to take established XML practice into account, notably the tendency to store data in text nodes rather than attributes.[4] It should be clear from the definition of the NXT object model in Section 2 that the use of text nodes is discouraged as being inconsistent with the underlying model. However, special provision for textual content is made in form of the TEXT() operator (see Section 2.6 for details).

1.1 Some background in graph theory

The structural properties of an object model are often discussed and defined in terms of graph theory. Therefore, we will briefly review the basic concepts and terminology of the mathematical theory of graphs in this section. Particular emphasis is given to ordered trees, which are an essential ingredient of the NXT object model.

A **directed graph** is a set of **nodes** (also called **vertices**) connected by **edges** (which are often pictured as little arrows between the nodes). In the NXT object model, nodes are referred to as elements (following XML terminology) and we write E for the set of all elements. The edges are represented by a binary relation R on E, which is a subset of the Cartesian product $E \times E$. Each element $(x, y) \in R \subseteq E \times E$ stands for an edge from node x to node y. For an **undirected graph**, the edge relation must be symmetric, i.e. $(x, y) \in R \iff (y, x) \in R$. A **path** from x to y is a sequence of consecutive edges $(x, z_1), (z_1, z_2), \ldots, (z_n, y)$. A **loop** in x is a path from a node x to itself. A graph is **acylcic** iff it does not contain any loops.

Certain acylcic graph structures can be pictured in a two-dimensional plane with all edges pointing in the same direction, usually downwards. For an edge (x, y), x is then referred to as the **parent** of y and y is a **child** of x (nodes that are children of the same parent are also called **siblings**). Using this terminology, a **tree** is an acyclic directed graph where every node has at most one parent (i.e. there is at most one edge (x, y) for every element $y \in E$), and there is exactly one node without parent, called the **root** node. Without the latter condition, the graph can be partitioned into a collection of unconnected trees (one for each root node), and is sometimes called a **grove**. A node without children is called a **leaf** node, and we refer to the set of all leaves in a tree or grove as its **boundary**.

The children, grandchildren, etc. of a node x in a tree or grove are collectively referred to as the **descendants** of x. Likewise, the node's **ancestors** are its parents, grandparents, etc.

Ordered trees (also called **plane trees**) extend the purely hierarchical structure of trees with a sequential ordering (and thus go beyond the general framework of directed graphs). This combination of hierarchical and sequential structure has earned them a central place in many applications (including linguistics and NLP) and at the heart of the XML format. As with trees, the hierarchical dimension is usually pictured on the vertical axis (top to bottom) and sometimes described as **dominance**; the complementary sequential dimension is pictured on the horizontal axis (left to right) and referred to as **precedence**.

As noted above, ordered trees are beyond the scope of a simple graph formalism. Several different frameworks for their representation are described in the literature, but there does not seem to be an agreement on a standard representation. Typically, an ordered tree is

[4] There is much uncertainty in the XML community about when it is appropriate to store information in attributes. Every XML textbook explains somewhere within the first twenty pages that `<w orth="dog" pos="NN"/>` and `<w> <orth>dog</orth> <pos>NN</pos> </w>` are (almost) equivalent representations of the same data. The accompanying examples often mix attributes and text nodes in a highly unsystematic way.

described by specifying an ordered list of children for each (parent) element. In addition to an implicit representation of the dominance graph, these ordered lists define a precedence relation among siblings, which is then extended to descendants: if x and y are siblings and x precedes y, then any descendant of x precedes any descendant of y. Any two nodes x, y of an ordered tree are *either* in a dominance relationship *or* in a precedence relationship, but not both.

One widely used formalism is the so-called **bracketing notation**, which represents an ordered tree as a sequence of opening and closing brackets and is isomorphic to the XML file format (with start tags corresponding to opening brackets and end tags corresponding to closing brackets). In this representation, a node x (which is called and element in XML terminology) precedes another node y iff the closing bracket of x comes before the opening bracket of y in the bracket sequence (i.e. the end tag of x precedes the start tag of y in the XML serialisation). Likewise, x dominates y iff the opening bracket of x comes before that of y and the closing bracket of x comes after that of y (and similarly for the start and end tags in the XML serialisation).

In addition to the precedence ordering (which is a strict partial ordering on the set of nodes), the nodes of an ordered tree can be **enumerated** by traversing the tree in a systematic fashion. The conventional enumeration is based on a top-down, left-to-right, depth-first traversal. In contrast to the partial ordering of precedence, enumeration of the nodes induces a linear ordering of all nodes in the tree. Despite its theoretical interest, this enumeration ordering is of little practical import. A more useful linear ordering is obtained by restricting the precedence relation to a suitable subset of the tree. This subset must satisfy the condition that there are no dominance relations between its nodes (i.e. no node x in the subset may dominate any other node y in the subset) and is referred to as a **(horizontal) axis**. The **axial ordering** induced on a horizontal axis is linear: since there are no dominance relations within the axis, any two nodes x, y must be in a precedence relationship. Examples of horizontal axes are (i) the set of all nodes at a certain "vertical distance" from the root node (the set of its children, the set of its grandchildren, etc.) and (ii) the boundary of a tree.

Notes on the mathematical terminology for ordered sets: A **partial ordering** of a set E is a binary relation $\prec$ on E, which must be transitive ($x \prec y \wedge y \prec z \implies x \prec z$), antisymmetric ($x \prec y \wedge y \prec x \implies x = y$), and reflexive ($x \prec x \ \forall x \in E$). If $\prec$ is irreflexive ($x \not\prec x \ \forall x \in E$, and hence $x \prec y \implies y \not\prec x$) instead, it is called a **strict** ordering. Two elements $x, y \in E$ are **comparable** iff $x \prec y$ or $y \prec x$. An ordering is **linear** iff any two different elements $x, y \in E$ are comparable (because such an ordering arranges the elements of E in a linear sequence).

2 The NXT object model

The components of a **NOM corpus** are (i) atomic units of information stored as string values (**attributes**), which are grouped into **elements** (Section 2.1); special attributes linking (some) elements to a common **timeline**; (ii) **structural relations** between the elements, namely hierarchical **dominance** (Section 2.2) and sequential **precedence** (Section 2.3); (iii) unconstrained **pointers** between elements (Section 2.4); (iv) a decomposition of the corpus into **layers**, which define **named hierarchies** (Section 3); and (v) **metadata** (Section 4), providing information about element types, the domains of attributes, pointer roles, layers and named hierarchies, as well as further optional information not defined in this document.[5]

[5]This additional information may include restrictions on the structural relations and pointers allowed between certain element types and/or layers, valid element types for each layer, coding schemes, a list of filenames for serialisation, external resources (such as video or audio recordings), etc. Note that the term *metadata* is

Formally, a NOM corpus is a 10-tuple $(E, A, \Uparrow, \prec, \rightarrow, T, \tau, R, N, \lambda)$. A detailed discussion of the components of this tuple is given in the following sections and in Section 3.

2.1 Elements and attributes

A NOM corpus is based on a set E of **elements**. A string-valued partial function $f : E \rightarrow \Sigma^*$ is called an **attribute**, and A is the set of all attributes. (The free monoid Σ^* is the set of all strings over the alphabet Σ. In an implementation of the NXT object model, Σ will usually correspond to the set of Unicode characters,[6] but special 8-bit encodings may be used as well.) In the following, small letters $x, y, z, \ldots \in E$ and $f, g, h, \ldots \in A$ stand for elements and attributes, respectively.

The basic metadata includes an inventory T of **element types**[7] and a function $\tau : E \rightarrow T$ (called a **type mapping**) that assigns each element to its type (see also Section 4). The domains of attributes may be restricted to certain element types, but such restrictions are not formalised here (the domain $\mathrm{dom}(f)$ of an attribute f is the set of all elements $x \in E$ for which $f(x)$ is defined).

There are two special real-valued **time attributes**, which are usually valid for all element types.[8] The `start` and `end` attributes $f_{\texttt{start}} : E \rightarrow \mathbb{R}$ and $f_{\texttt{end}} : E \rightarrow \mathbb{R}$ link elements to a common **timeline**, and their domains must coincide (i.e. $\mathrm{dom}(f_{\texttt{start}}) = \mathrm{dom}(f_{\texttt{end}})$). $f_{\texttt{start}}(x)$ and $f_{\texttt{end}}(x)$ represent the start and end points of a time interval associated with the element x, and must satisfy $f_{\texttt{start}}(x) \leq f_{\texttt{end}}(x)$. $f_{\texttt{start}}(x) = f_{\texttt{end}}(x)$ is explicitly allowed and identifies a point on the timeline. (See Section 2.5 for a further discussion of the time attributes).

2.2 Dominance and hierarchies

The **dominance relation** $\Uparrow \subseteq E \times E$ is a **transitive** ($x \Uparrow y \wedge y \Uparrow z \implies y \Uparrow z$) and **irreflexive** (there is no $x \in E$ with $x \Uparrow x$) binary relation on E,[9] which must satisfy the **unique path criterion** (**UP**, see below). x is said to **dominate** y iff $x \Uparrow y$ holds; following the usual terminology for hierarchical data structures, x is also called an **ancestor** of y, and y a **descendant** of x. Given two elements $a, b \in E$ with $a \Uparrow b$, the **path** $P(a, b)$ from a to b is the set

$$P(a, b) := \{x \in E \mid a \Uparrow x \wedge x \Uparrow b\} \cup \{a, b\}$$

often used for additional information attached to an entire corpus or individual observations in the corpus (e.g. information about speakers/agents, date of recording, annotators and the annotation process). From the technical perspective of this document, such information is ordinary *data* rather than true metadata. [1] defines a standard set of metadata used in the NXT toolkit (and allows for some user-defined metadata in the non-technical sense, stored per corpus or per observation), including a standardised XML encoding. This approach also encompasses an orthogonal decomposition of layers along the dimensions of agents, transcriptions, codings, and observations (cf. Section 3).

[6] In this simple formalism, sequences of combining characters – together with the base character they apply to – have to be treated as a single "extended" character (see [6] for details and terminology).

[7] Element types correspond to the concept of **element names** in XML. It seems more appropriate to speak of *types* – from a fixed inventory – rather than *names* in our setting, but keep in mind that element types will be expressed as element names in an XML-based implementation. [2] speaks of the *simple type* of an element.

[8] Although it might make sense not to allow `start` and `end` values for elements representing "timeless" information.

[9] These two conditions make $\Uparrow$ a strict partial ordering on E. Antisymmetry (which takes the form $x \Uparrow y \implies y \not\Uparrow x$ for a strict ordering) follows automatically: if $x \Uparrow y$ and $y \Uparrow x$, then by transitivity also $x \Uparrow x$, in contradiction to the irreflexivity of $\Uparrow$.

The unique path criterion requires that

$$\text{(UP)} \qquad \forall x, y \in P(a,b): \quad x \Uparrow y \ \vee \ y \Uparrow x \ \vee \ x = y$$

for every path $P(a,b)$ (i.e. $\forall a, b \in E$ with $a \Uparrow b$). In other words, the restriction of the dominance relation to any path $P(a,b)$ must be a (strict) **linear ordering**, since $x \neq y$ are always comparable.

x **immediately dominates** y ($x \uparrow y$) iff $x \Uparrow y$ and $P(x,y) = \{x, y\}$ (i.e. there is no intervening element "between" x and y). In this case, x is also called a **parent** of y, and y a **child** of x. For the set of **parents** of X and the set of its **children**, we use the notation $\mathit{Parents}(x) := \{y \in E \mid y \uparrow x\}$ and $\mathit{Children}(x) := \{y \in E \mid x \uparrow y\}$. $\uparrow \subseteq E \times E$ defines a graph structure on the set of elements, which we refer to as the **parent-child** graph. Its transitive closure $\uparrow^*$ recovers the dominance relation:

$$x \uparrow^* y \iff x \Uparrow y \ \vee \ x = y$$

Proof: Since $x \uparrow y$ implies $x \Uparrow y$ and $\Uparrow$ is transitive, we have $x \uparrow^* y \implies x \Uparrow y \vee x = y$. Conversely, for $x \Uparrow y$, consider the path $P(x,y) \supseteq \{x, y\}$. Arranging the elements of $P(x,y)$ according to the linear ordering induced by the dominance relation (because of the UP), we have $P(x,y) = \{x = w_0, w_1, \ldots, w_n = y\}$ with $w_i \uparrow w_{i+1}$ (because any element $z \in E$ with $w_i \Uparrow z \Uparrow w_{i+1}$ would also belong to $P(x,y)$ and would have to be listed *between* w_i and w_{i+1} in this enumeration). Thus, $x = w_0 \uparrow w_1 \uparrow \ldots \uparrow w_n = y$ and hence $x \uparrow^* y$.

The parent-child graph structure and the equivalence above explain the graph-theoretical terminology used, the definition of $P(a,b)$ as the path between a and b, and the unique path criterion. The terms parent, child, ancestor, descendant, and dominance correspond to the respective graph-theoretical notions (as introduced in Section 1.1) on the parent-child graph. The elements of $P(a,b)$, linearly ordered by the dominance relation, are the nodes of a path from a to b, and the UP ensures that this path is unique. When further terms from graph theory are introduced below, their definitions are (nearly) equivalent to the usual graph-theoretical meaning applied to the parent-child graph.

Note that the definition of $\Uparrow$ as a transitive and irreflexive relation results in a **directed acyclic graph** (DAG), as there can be no loops in the parent-child graph. Hence the fundamental difference between dominance in the NXT object model and DAGs lies in the unique path criterion. DAGs allow multiple paths between nodes, whereas the UP guarantees that $P(a,b)$ (if it exists) is the unique path between a and b.

A **subcorpus** is a subset $C \subseteq E$ together with all attributes and relations restricted to C, i.e. the 10-tuple $(C, \{f|_C \mid f \in A\}, \Uparrow|_C, \prec|_C, \rightarrow|_C, T, \tau|_C, R, N, \lambda|_C)$.[10] A **root element** in the subcorpus C is an element $x \in C$ which has no parents in C, i.e. where $\mathit{Parents}(x) \cap C = \emptyset$; likewise a **leaf element** y has no children in C, i.e. $\mathit{Children}(y) \cap C = \emptyset$ (these definitions extend to the case where $C = E$). For any $x \in E$, the subcorpus $O(x)$ of all descendants of x, including x itself, is called the **offspring** of x. Similarly, the subcorpus $A(x)$ of all its ancestors, including x itself, is called its **ancestry**.

$$O(x) := \{y \in E \mid x \Uparrow y\} \cup \{x\} = \{y \in E \mid x \uparrow^* y\}$$
$$A(x) := \{y \in E \mid y \Uparrow x\} \cup \{x\} = \{y \in E \mid y \uparrow^* x\}$$

A **hierarchy** $H \subseteq E$ is a subcorpus that satisfies the following two conditions:

[10] $f|_C$ marks the restriction of a function f on E to the subset $C \subseteq E$. Likewise, $\Uparrow|_C$ is the restriction of a relation on E to C: $\Uparrow|_C = \Uparrow \cap (C \times C)$.

1. every element $x \in H$ has at most one parent in H, i.e. there is at most one $y \in H$ with $y \uparrow x$ (or, equivalently, $|Parents(x) \cap H| \leq 1$);

2. for every $x \in H$, the hierarchy also contains the entire offspring of x, i.e. $O(x) \subseteq H$.

It is shown below that, because of property 1, every hierarchy H is a collection of tree structures rooted in the root elements of H, i.e. a grove. (This may be somewhat different from the conventional notion of a hierarchy as a single tree.) Property 2 ensures that hierarchies are closed wrt. their descendants, extending the graph-theoretical concept of a grove. This property will turn out to be important when intersections of hierarchies are considered and it helps to ensure the consistency of the different precedence orderings on the hierarchies. (Interestingly, property 2 can be rephrased as the requirement that every leaf in H is also a leaf in E.) Property 2 also implies that a root element in H, i.e. an element without parent in H, cannot have any other ancestors (grandparents etc.) in H either.

A hierarchy with a single root element is called a **tree**. It is an unordered tree in the graph-theoretical sense (with respect to the parent-child graph). For every $x \in E$, the offspring $O(x)$ is a tree rooted in x, and the ancestry $A(x)$ is an "inverted" tree (where *parent* and *offspring* are replaced by *child* and *ancestry* in the definition above). Every hierarchy H is a union of disjoint trees (one for each root element in H), called the **components** of H:

$$H = \overset{\circ}{\bigcup} \{O(x) \mid x \text{ is a root element in } H\}\,.$$

In particular, every tree T can be written as the offspring of its root element x: $T = O(x)$. An important property of the set of all hierarchies in a corpus is that it is closed under intersection: for any two hierarchies $H_1, H_2 \subseteq E$, their intersection $H_1 \cap H_2$ is also a hierarchy. Note that the intersection of two trees *need not* be a tree itself (but is always a hierarchy).

Proofs: Property 2 stipulates that for any $x \in H$, including any root element x, $O(x) \subseteq H$. Furthermore, if x is not a root element, then it must belong to the offspring tree of some root element $y \in H$: $x \in O(y)$ (which can be found by moving up along the acyclic parent-child graph until an element without parents in H is reached). The offspring trees of two different root elements in a hierarchy H cannot intersect because of property 1. Otherwise, for any $z \in O(x) \cap O(y)$ we consider the set $P := P(x, z) \cup P(y, z) \subseteq H$. Since no $w \in P$ can have more than one parent in $P \subseteq H$, the set P is linearly ordered by the dominance relation. (This is shown easily by induction starting from z, making use of the fact that all elements $w \in P$ are connected to z in the parent-child graph.) Since $x, y \in P$ and $x \neq y$, we have either $x \in O(y) \subseteq H$ or $y \in O(x) \subseteq H$, so that either x or y must have ancestors in H and cannot be a root element. It is obvious that properties 1 and 2 also hold for the intersection of two hierarchies, which is thus itself a hierarchy. However, the intersection of two trees can have more than one root element (in which case it is a hierarchy, but not a tree).

We can interpret a NOM corpus as a collection of intersecting hierarchies. More specifically, the E is the union of the (usually not disjoint) trees $O(x)$ for all root elements x in E:

$$E = \bigcup \{O(x) \mid x \text{ is a root element in } E\}\,.$$

2.3 Precedence: multiple sequential orderings

The **precedence** relation defines a strict partial ordering on each hierarchy in the corpus, which is restricted to the components of H (so that elements from different components of H are never comparable in H). The precedence ordering on H extends every component T to an ordered tree in the graph-theoretical sense (cf. the properties listed in Section 1.1).

As a further requirement, the precedence orderings of different hierarchies H_1 and H_2 must be consistent, i.e. they must be identical on every component of the intersection $H_1 \cap H_2$. Alternatively, precedence can be defined as a linear ordering of each set of siblings[11] in the corpus, and then extended to precedence orderings on the hierarchies (so that the consistency requirement is automatically satisfied).

The **precedence relation** $\prec \subseteq E \times E \times E$ is a ternary relation on E that induces a family of strict linear orderings $\{\prec_z\}_{z \in E}$, where $\prec_z$ is a linear ordering on $\mathit{Children}(z)$. Using the notation

$$x \prec_z y \iff \prec (z; x, y),$$

we have the following formal requirements on $\prec$:

1. $x \prec_z y \implies x, y \in \mathit{Children}(z)$, i.e. $\prec_z$ defines a binary relation on $\mathit{Children}(z)$; (equivalently, $\prec \subseteq \big\{(z; x, y) \in E \times E \times E \;\big|\; z \uparrow x \wedge z \uparrow y\big\}$)
2. $\prec_z$ is a strict linear ordering on $\mathit{Children}(z)$, i.e. for all $x, y \in \mathit{Children}(z)$ exactly one of three mutually exclusive conditions holds: $x = y$ or $x \prec_z y$ or $y \prec_z x$.

A tree $O(x)$ together with the local orderings $\{\prec_z\}_{z \in O(x)}$ for its element nodes is an **ordered tree** in the graph-theoretical sense. This holds in particular for the components of a hierarchy H. Therefore, the sequential orderings (**precedence**, **enumeration**, and **axial ordering**) introduced in Section 1.1 for ordered trees can be applied to the components of hierarchies, and thus to the hierarchies themselves. Note that in the extension to hierarchies, elements from different components are never comparable (with respect to either one of the orderings).

In the following, we give an explicit definition of the precedence ordering on hierarchies, which is most important for the data model. (The axial ordering is obtained by restriction of the precedence relation to a horizontal axis, and will be further explained in Section 5.2.) For each hierarchy H, the extension of $\prec$ to precedence orderings on the components of H is written $\prec_H \subseteq H \times H$, and is called the H**-precedence** relation. Formally, $\prec_H$ is defined by

$$x \prec_H y \iff \exists z, x', y' \in H : (x' \uparrow^* x) \wedge (y' \uparrow^* y) \wedge (x', y' \in \mathit{Children}(z)) \wedge (x' \prec_z y').$$

The condition $x', y' \in \mathit{Children}(z)$ is redundant since it is implied by the last term. It is intended as a reminder that z is a common ancestor of x and y, which ensures that elements from different components are never comparable.

The extensions of $\prec$ to different hierarchies are **consistent**. For any two hierarchies H_1 and H_2, the H_1-precedence and H_2-precedence relations are identical on every component T of the intersection $H_1 \cap H_2$:

$$x \prec_{H_1} y \iff x \prec_{H_2} y \iff x \prec_{(H_1 \cap H_2)} y \quad \forall x, y \in T$$

or, more concisely,

$$\prec_{H_1}|_T = \prec_{H_2}|_T = \prec_{(H_1 \cap H_2)}|_T.$$

Note that the orderings need not be consistent on the entire hierarchy $H_1 \cap H_2$: elements x and y from different components of $H_1 \cap H_2$ may belong to a single component of H_1 (or H_2, or both), so that they are comparable in $\prec_{H_1}|_{H_1 \cap H_2}$ but cannot be comparable in $\prec_{(H_1 \cap H_2)}$.

[11] As in graph theory, a set of siblings is defined as the set of children $\mathit{Children}(x)$ of an element x. Note that here, unlike in the case of trees in Section 1.1, an element y can belong to more than one set of siblings (when it has multiple parents).

Proof and remark: It follows immediately from the definition of $\prec_H$ that the precedence ordering $\prec_{H|O(x)}$ induced on the offspring tree of an element $x \in E$ is the same for every hierarchy H with $x \in H$. Taking x as a root element of the intersection $H_1 \cap H_2$ (so that $O(x)$ is one of its components), we obtain the consistency requirement between different hierarchies. Conversely, any collection $\{\prec_H\}$ of consistent precedence orderings on the hierarchies in E can be derived from a suitable precedence relation $\prec \subseteq E \times E \times E$. For any $x \in E$, the ordering $\prec_x$ of its children is obtained directly from $O(x)$, while consistency ensures that the extension of these local orderings to hierarchies recovers the original precedence orderings $\{\prec_H\}$. (The consistency requirement is used in the form $\prec_{H|O(x)} = \prec_{O(x)}$ for every $x \in H$.)

Many NLP applications, especially in the field of multi-modal communication, make heavy use of flat transcription and annotation layers, which consist of sequences of elements without (internal) hierarchical structure.[12] In the NXT object model, just as in XML, it is necessary to introduce a common parent element (or a tree of ancestors) in order to express the sequential ordering of the elements within each layer. Since this element (or tree) does not represent actual corpus data, a special `stream` element should be used, which applications such as display and query modules can easily hide from the user (see also Section 6). The formal data model does not distinguish between `stream` elements and "ordinary" elements, except that no attributes should be defined on the former (including $f_{\texttt{start}}$ and $f_{\texttt{end}}$).

2.4 Pointers

Dominance and precedence, with the limitations imposed by the intersecting hierarchies model and the unique path criterion, are not sufficient to capture all the complexity of linguistic data. This is especially true for cross-references (such as traces and anaphora) and for cross-modality annotation, both of which would often violate the UP were they encoded in the parent-child graph. Therefore, the NXT object model allows additional **pointers** to express arbitrary connections between elements. Each pointers is a directed link from a **source** element to a **target** element, and is labelled with a **role**. There may be multiple pointers with the same role originating from a given source element or pointers with different roles between the same elements, but no two pointers between the *same elements* with the *same role*.

The formal definition of pointers involves a set R of **roles** and an arbitrary relation $\rightarrow \subseteq R \times E \times E$, called the **pointer relation**. For each role $r \in R$, $\rightarrow$ induces a binary relation $\rightarrow_r \subseteq E \times E$ on E:

$$x \rightarrow_r y \iff \rightarrow (r; x, y)$$

Each $\rightarrow_r$ is interpreted as a directed graph on E, so that the pointer relation defines a set of overlayed graph structures. Note that there are no constraints on the graph structures $\rightarrow_r$, which may therefore contain both loops and multiple paths.

This great expressive power comes at a price: we cannot connect the pointer graphs with the hierarchical or sequential structure of the corpus (in the form of the dominance and precedence relations). Moreover, applications such as the query processor have no direct access to the closures of local structures (in contrast to $\Uparrow$ and $\prec_H$, which are transitive closures), and must process pointers "one step at a time". These limitations of pointers are reflected in the NQL query language [4], which provides complex operators for hierarchical and sequential structure, but only basic support for pointers.

[12] Mathematically, such a flat layer can simply be represented as an ordered list of elements.

2.5 Time attributes

We use the term **timed element** for an element with timestamps, i.e. an element $x \in \mathrm{dom}(f_{\texttt{start}}) = \mathrm{dom}(f_{\texttt{end}})$. When two timed elements are in a dominance relationship, their timings must be consistent with the hierarchical structure:

$$x \Uparrow y \implies (f_{\texttt{start}}(x) \leq f_{\texttt{start}}(y)) \wedge (f_{\texttt{end}}(x) \geq f_{\texttt{end}}(y)) \quad \forall x, y \in \mathrm{dom}(f_{\texttt{start}}) = \mathrm{dom}(f_{\texttt{end}})$$

Timestamps need not be consistent with the various H-precedence orderings or with any of the pointer graphs. An implementation of the NXT object model should provide facilities to infer an element's timestamps from its descendants, setting

$$f_{\texttt{start}}(x) = \min \{ f_{\texttt{start}}(y) \mid y \in O(x) \cap \mathrm{dom}(f_{\texttt{start}}) \} ;$$
$$f_{\texttt{end}}(x) = \min \{ f_{\texttt{end}}(y) \mid y \in O(x) \cap \mathrm{dom}(f_{\texttt{end}}) \} .$$

Thus, only leaf elements[13] need explicit timestamps, and the consistency requirement above is always fulfilled. Note that in certain situations it may still be necessary to have explicit timestamps at different levels.[14] The object model makes no difference between "explicit" and "inferred" time attributes.

2.6 Textual content

In XML, information can be stored either in the form of attributes or in separate text nodes. The **textual content** of an element is then interpreted as the linear concatenation of all text nodes contained in the element's body.[15] In the NXT data model, the equivalent of an XML element body is the element's offspring tree together with its precedence ordering. When existing XML resources are integrated into a NOM corpus, the textual content of an element can be retrieved with the special TEXT() operator.

An XML-based implementation with support for text nodes has to make sure that the textual content of each element is computed correctly. Optionally, functionality can be provided to insert new text nodes into a corpus.[16]

The recommended implementation strategy is to represent text nodes by special "invisible" `text` elements. These elements must be leaves, cannot be the source or target of pointers, and are ignored by the dominance relation, the precedence relation, and the axial orderings (for instance, two siblings separated only by text nodes are still considered adjacent). The textual content TEXT(x) of an element x is then determined in the following way: First, the "extended" offspring tree $O'(x)$ including the relevant text nodes is generated together with its "extended" precedence ordering. Since text nodes have no descendants, they cannot be in dominance relationships with each other. Hence, the text nodes form a horizontal axis in $O'(x)$, which is linearly ordered by the restriction of extended precedence to the axis. Now TEXT(x) is given by the concatenation of the text nodes in their linear order.

[13] More generally, elements at the lowest level linked to the timeline, which is usually a transcription layer.

[14] E.g. when when timings are first recorded at a higher level (such as words) and later refined to a smaller granularity (such as individual phones).

[15] This includes both text nodes that are direct children of the element and text nodes embedded within descendants of the element.

[16] Text nodes will often stem from external XML documents that are integrated into the corpus. In this common case, only read access is necessary, but care has to be taken to preserve existing text nodes when new elements are inserted.

3 Layers and serialisation

The NXT data model is formulated in terms of the disjoint layers introduced in this section. This layer structure also forms the basis for the transformation of a NOM corpus into a collection of XML files (or equivalent hierarchical data structures) for permanent storage, a process referred to as **serialisation**. It is also possible, of course, to store NOM corpora in an entirely different format, e.g. as a list of elements together with an explicit representation of all functions[17] and relations defined in the object model. However, such a graph-based format would not be able to take advantage of the structural properties of the intersecting hierarchies design (including the easy re-use of existing XML software and resources).

Every hierarchy $H \subseteq E$ can be serialised into a single XML file. If H has more than one component, a common root element must be inserted into the XML representation. Alternatively, each component tree may be written to a separate file. However, since the hierarchies intersect, this simple approach would duplicate many elements in the serialisation and might easily lead to inconsistencies (in addition to wasting disk space and processing time). It is therefore desirable to decompose a corpus into *disjoint* subsets, which we refer to as **layers**.

The layer structure can be expressed formally through a **layer mapping** $\lambda : E \to N$, where N is a set of **layer names**.[18] A **layer** $L \subseteq E$ is the subcorpus containing all elements that are mapped to the same layer name $n_l \in N$; in short, $L = \lambda^{-1}(n_l)$. An element $x \in L$ is a **leaf** in L iff it has no children in L, i.e. $\mathit{Children}(x) \cap L = \emptyset$.[19] It is an **interior element** of L iff all its children belong to L, i.e. $\mathit{Children}(x) \subseteq L$. Note that by these definitions, every leaf in the full corpus E is both a leaf *and* an interior element in its layer. A layer is called **linear** iff it contains only interior elements and leaves.[20] The **boundary** of a linear layer L is the set of all leaves in L.

Layers must be **vertically connected** (wrt. the parent-child graph), in the sense that for any $x, y \in L$ with $x \Uparrow y$, we have $P(x, y) \subseteq L$. When a layer is extended to include all descendants, the resulting subcorpus

$$H(L) := \bigcup \{O(x) \mid x \in L\}$$

must be a **hierarchy**. This requirement is equivalent to the condition that no two elements from the same layer may have a common descendant in the corpus (unless one dominates the other). Formally, for all $x, y \in L$ either $O(x) \subseteq O(y)$ or $O(y) \subseteq O(x)$ or $O(x) \cap O(y) = \emptyset$.[21]

Another formulation of the hierarchy requirement, which may be easiest to validate in an actual implementation, stipulates two conditions: (i) every element $x \in L$ has at most one parent in L; (ii) any two elements $x, y \in L$ that are not in a dominance relationship (i.e. neither $x \Uparrow y$ nor $y \Uparrow x$) must not have a common descendant in the full corpus E. The consistency checks can be further simplified for linear layers, where it suffices to validate (ii) for every leaf of L, scanning for common descendants *outside* L. It is still necessary to ensure the vertical connectedness of layers explicitly, though.

The conditions above can be summarised into the statement that layers are vertically connected subsets of hierarchies. An element $x \in L$ is a root element of the layer L iff it is a

[17] Recall that attributes are string-valued functions on the set of elements.

[18] Note that λ must not be a partial function, i.e. we require that $\mathrm{dom}(\lambda) = E$.

[19] Because of the vertical connectedness requirement introduced below, a leaf $x \in L$ can have no descendants in the same layer, i.e. $O(x) \cap L = \{x\}$.

[20] The term **linear layer** refers to the linear axial precedence ordering on the boundary of L. Section 5 introduces horizontal distance, which is computed by projection to the boundary of a layer. This interpretation of the boundary as a horizontal slice of a corpus is meaningful for linear layers only (although formally, the leaves of any layer form a horizontal axis).

[21] This condition implies that no element x may have more than one parent in its layer: for two different parents $z_1, z_2 \uparrow x$, we have $O(z_1) \cap O(z_2) \supseteq \{x\} \neq \emptyset$.

root element of the corresponding hierarchy $H(L)$. Each root element x together with its offspring within L (i.e. $O(x) \cap L$) is called a **component** of the layer. Intuitively speaking, a layer component is the connected "top part" of a component of the corresponding hierarchy. A layer L is the disjoint union of its components. As such it can be serialised into a single XML file (or a similar hierarchical data structure) by adding a common root element; or into several files, one for each component.

Among the many hierarchies that can be defined in a corpus, only some will be of interest to applications such as the query processor. Usually, these are the **named hierarchies** derived from layers by setting $H(n_l) := H(\lambda^{-1}(n_l))$ for every $n_l \in N$. The metadata may also define **collections of layers** and the corresponding **collections of named hierarchies**, especially with a factorial layer design[22] (Section 5 explains the usage of such collections).

4 Comments on metadata

The metadata associated with a NOM corpus can be split into two sets: **core metadata**, which is required by the object model and defined in this document; and **extended metadata**, which is described in [1] (and may be further extended by an implementation).

The **core metadata** consists of the element types T and the type mapping τ, pointer roles R, as well as layer names N and the layer mapping λ. This information is not encoded in the standard XML format defined in Section 6 and must be provided by separate metadata documents (however, some of it such as the type mapping τ and part of the layer mapping λ is implicit in the XML files).

The **extended metadata** should provide a list of attribute names (which are associated with the attributes of the formal data model), information about linear layers, and named collections of layers (and the corresponding hierarchies). For an XML serialisation of the corpus, a list of files and a mapping from filenames to layers are also necessary. In addition to this essential information, the extended metadata will include further information about the corpus such as links to video and audio recordings and their relation to the common timeline, date of each observation, lists of subjects and annotators, etc. (see [1] for details).

Note that the XML encoding assumes that element types, attribute names, and roles can be represented as Unicode strings.

5 Summary of structural information

This section gives a summary of the explicit and implicit structural information encoded in a NOM corpus, and specifies in what form it should be made available by an implementation of the NXT object model (especially to the query, transformation, and data extraction modules). The specification of the NXT query language [4] uses the terminology and definitions introduced here. There are six different kinds of structural information, which are detailed in the following sections.

1. **dominance** ("vertical") ordering, and **vertical distance**;

[22] A **factorial design** classifies layers according to several orthogonal **dimensions** such as *agent*, *coding*, and *annotator*. Collections of hierarchies arise naturally as cross-sections along these dimensions, e.g. taking all layers that belong to the same coding (but for different actors or made by different annotators).

2. **precedence** ("horizontal") ordering and **sequential distance** within each hierarchy;
3. **horizontal distance** obtained by projection to a horizontal axis;
4. multiple unrestricted **pointer graphs**; and
5. various **temporal orderings** that are derived from the timing information.

5.1 Dominance and precedence

An implementation of the NXT object model should provide direct access to the **direct dominance** relation $\uparrow$ and its **closure** $\uparrow^*$ (or, equivalently, the **dominance** relation $\Uparrow$). This includes functions that return, for any element $x \in E$, the sets of children $\mathit{Children}(x)$, parents $\mathit{Parents}(x)$, descendants $O(x)$, and ancestors $A(x)$. The **vertical distance** between two elements $x \Uparrow y$ is the length of the path from x to y in the parent-child graph, i.e. $|P(x, y)| - 1$.

Precedence is defined either with respect to a named hierarchy ($\prec_H$) or with respect to a tree identified by its root element x ($\prec_{O(x)}$). The **sequential distance** between two elements $x \prec_H y$ in the hierarchy H is the number of *maximal* elements w between x and y, i.e.

$$\left|\left\{w \in H \;\middle|\; x \prec_H w \prec_H y \wedge (\nexists w' \in H : w' \Uparrow w \wedge x \prec_H w' \prec_H y)\right\}\right| + 1.$$

This definition of sequential distance is identical to the number of maximal regions between x and y in an XML serialisation of the hierarchy H (plus one).

The definition and computation of the precedence relation and sequential distance with respect to a **hierarchy collection** is a non-trivial task. The procedure outlined here gives a well-defined results that should be consistent with the users' intuitions when the hierarchy collections is defined in an appropriate manner. For a given collection $\{H_i\}_{i=1\ldots n}$ and two elements x and y, the intersection of all hierarchies in the collection to which both x and y belong also forms a hierarchy $H_{x;y}$:

$$H_{x;y} := \bigcap \{H_i \mid i = 1 \ldots n \wedge x \in H_i \wedge y \in H_i\}$$

We can now define that x precedes y wrt. to the hierarchy collection iff x precedes y in the hierarchy $H_{x;y}$, i.e. $x \prec_{\{H_i\}} y \iff x \prec_{H_{x;y}} y$. The sequential distance between x and y is also defined wrt. to the hierarchy $H_{x;y}$. Note that when the collection consists of a single hierarchy H_1, we have $H_{x;y} = H_1$ and hence $\prec_{\{H_1\}} = \prec_{H_1}$, so the definition above is indeed a generalisation of the "ordinary" precedence relation $\prec_H$.

5.2 Horizontal distance

The sequential distance between two elements x and y depends to a great extent on the underlying hierarchy H. Especially when x and y are positioned at different "levels" within a component tree, the resulting number may be counterintuitive. For instance, the distance between two noun phrases in a syntax tree would be given by the number of maximal elements in between (plus one), which can be a mixture of noun phrases, other maximal phrases, and even single words. It would certainly be more natural to measure the horizontal distance between noun phrases by the number of intervening words ("orthographic distance").

This kind of **horizontal distance** can be defined generally by projection to a **horizontal axis** $A \subseteq H$ in a hierarchy H. Recall from Section 1.1 that the elements of A must belong

to a single component of H and there must not be dominance relationships between them, so that $\prec_H$ induces a linear **axial ordering** on A. The **projection** of an element $x \in H$ (from the same component tree) onto A is $P_A(x) := P_{\uparrow A}(x) \cup P_{\downarrow A}(x)$, where $P_{\uparrow A}(x) := A(x) \cap A$ is the *upward projection* and $P_{\downarrow A}(x) := O(x) \cap A$ the *downward projection*. (Note that one of the two projections will be an empty set except for the case $P_{\uparrow A}(x) = P_{\downarrow A}(x) = P_A(x) = \{x\}$.)

Given two elements $x, y \in H$ with $x \prec_H y$ (so that both must belong to the same component of H), and given that $P_A(x) \neq \emptyset$ and $P_A(y) \neq \emptyset$, then $P_A(x)$ and $P_A(y)$ are disjoint subsets of A and the **horizontal distance** between x and y is the number of elements that fall between $P_A(x)$ and $P_A(y)$ on the axis A (with respect to the axial ordering on A).

Remarks: $P_A(x) \cap P_A(y) = \emptyset$ follows from the structural properties of the hierarchy H, especially the unique path criterion. It should be noted that the horizontal distance of "adjacent" elements is 0, whereas the corresponding sequential distance would be 1. Since the axis A is not required to be a "horizontally connected" subset of H, it is possible that $P_A(x) = \emptyset$ or $P_A(y) = \emptyset$ even if $x \prec_H y$. In this case, the horizontal distance between x and y is undefined.

The horizontal axis A in the definition above can be (i) the boundary of a tree or hierarchy;[23] (ii) the axis of all elements in a tree at a certain vertical distance d from its root element; or (iii) the boundary of a linear layer (for a flat transcription layer joined by `stream` elements, the boundary is identical to the layer itself).

It is desirable to generalise projections and the concept of horizontal distance to collections of linear layers (instead of a single horizontal axis A), but it is not clear yet how this can be achieved in a consistent and meaningful way.

5.3 Pointer graphs

For each role $r \in R$, the pointer relation defines an unrestricted directed **pointer graph** $\rightarrow_r \subseteq E \times E$ on E. The **generic pointer graph** $\rightarrow_*$ is the union of all these graphs, i.e.

$$\rightarrow_* = \bigcup_{r \in R} \rightarrow_r = \big\{(x, y) \in E \times E \;\big|\; \exists r \in R : x \rightarrow_r y\big\}$$

Note that a NOM implementation will usually not provide direct access to the closure of pointer graphs or other advanced graph-theoretical functionality.

5.4 Temporal orderings

Several different orderings on the set of timed elements can be derived from the timestamps associated with elements. All such orderings are represented by the symbol $\ll$ in the discussion below. For instance, strict temporal precedence (element x ends before y starts on the timeline) is defined by

$$x \ll y :\Longleftrightarrow\ f_{\texttt{end}}(x) < f_{\texttt{start}}(y).$$

While this definition gives a strict partial ordering in the mathematical sense, other definitions may produce non-transitive or non-antisymmetric relations (which can therefore not be understood as a mathematical ordering relation). One example is ordering by start time:

$$x \ll y :\Longleftrightarrow\ f_{\texttt{start}}(x) \leq f_{\texttt{start}}(y).$$

[23] In order to apply the formal definition of the horizontal distance, A is restricted to the single component to which elements x and y with $x \prec_H y$ must belong.

This definition has the advantage that any two timed elements are comparable, but it is obviously not antisymmetric (i.e. $x \ll y \wedge y \ll x$ does not imply $x = y$). It is interesting to note that a temporal ordering need not have a sequential nature as in the previous examples. This is shown by the "hierarchical" containment ordering

$$x \ll y :\Longleftrightarrow\ f_{\texttt{start}}(x) > f_{\texttt{start}}(y) \wedge f_{\texttt{end}}(x) < f_{\texttt{end}}(y).$$

Note that most orderings are phrased as a strict ordering because otherwise elements x and y with identical timestamps would always compare equal ($x \ll y$ and $y \ll x$), violating the antisymmetry requirement for a partial ordering in the mathematical sense.

6 XML encoding

In the standard XML encoding, a NOM corpus is represented as a collection of XML documents, which can either be stored in memory (e.g. using a DOM implementation) or serialised to XML files. Each XML file corresponds to a layer or layer component. The list of corpus files, the definition of layer names and their mapping to file names, and the definition of layer collections are part of the external metadata described in [1]. All attributes and elements with a special meaning are in the NITE namespace

`http://nite.sourceforge.net/`[24]

which is conventionally mapped to the prefix `nite:` .

The elements and attributes defined in the object model translate directly into XML elements and attributes. Element types are rendered as XML element names; likewise, attributes are identified by XML attribute names. Every element is assigned a unique ID value (which must at least be unique within the XML document containing it) stored in the `nite:id` attribute. The time attributes $f_{\texttt{start}}$ and $f_{\texttt{end}}$ are mapped to the XML attributes `nite:start` and `nite:end` with their real-number values converted to a suitable floating-point representation. The XML files may also contain text nodes accessible in the NOM by use of the TEXT() operator (cf. Section 2.6). If a corpus contains `stream` elements (cf. Section 2.3), they are represented by XML elements with the name `nite:stream`, which may have no attributes except for the required `nite:id` value.

Since XML documents are strictly hierarchical, the intersecting hierarchies of a NOM corpus must be decomposed into a collection of tree-like structures, connected by additional links in an XPointer-compatible stand-off annotation (the same stand-off annotation is used for pointers). The standard XML encoding is based on the decomposition into layers described in Section 3.

Every layer L is converted into a single XML document, inserting a root element named `nite:layer` that is *not* part of the corpus data. This root element is necessary when L has more than one component, but is also inserted for single components to make the XML

[24] This URL points to a project website for the NITE XML Toolkit (NXT) hosted by SourceForge. Although the XML standard does *not* formally require that the web address identifying a namespace actually exists, it is much better style (and helps to avoid name collisions) to use a URL that is under our control and to provide some information about the NXT object model and XML format on the associated web page. This website will also serve as a (more or less) permanent home for the NXT sourcecode after the end of the project.

An alternative solution would have been the use of a URN as namespace identifier. I believe that `urn:publicid:NITE:XML+1.0` would not have violated too many rules (cf. `http://www.ietf.org/rfc/rfc3151.txt`), although in general URNs should be registered officially.

document format more uniform.[25] Alternatively, each component of the layer can be stored in a separate document, again inserting a `nite:layer` element as root. Within each layer, dominance and precedence relationships are represented by the structure of the XML document. The children of an element x appear in their local precedence order $\prec_x$ in the body of the XML element representing x. Any children that belong to a different layer (i.e. those in the set $\mathit{Children}(x) \setminus L$) are replaced by special empty `nite:child` elements. Following the XLink standard, the target of the `nite:child` link is given by an `xlink:href` attribute, with a reference to an element ID in the local part.[26] For instance,

```
<nite:child xlink:href="phones.xml#p_120"/>.
```

Links to consecutive elements in a single XML document can be combined into a range syntax:

```
<nite:child xlink:href="phones.xml#p_120..p_124"/>.
```
[27]

The XML representation of an interior element contains no `nite:child` links, whereas the representation of a leaf element contains *only* `nite:child` links. Note that for elements which are neither interior elements nor leaves, the `nite:child` links must be inserted in their appropriate places in the list of children. Pointers (cf. Section 2.4) are represented by `nite:pointer` elements, which use the same `xlink:href` syntax as above and have an additional `role` attribute specifying the pointer's role. The `nite:pointer` elements may be freely intermixed with any other element content; they are ignored when determining the precedence ordering of the element's children.

For the efficient processing of large corpora, the corpus data must be split into moderately-sized files, which are then loaded only when necessary. With this load-on-demand strategy, links to all parent elements and "inverse" pointers must be explicitly represented in the XML format (otherwise, all files would have to be scanned for potential ancestors or pointer sources), using `nite:parent` and `nite:backpointer` elements. These elements share the XLink syntax of `nite:child` and `nite:pointer`. Like `nite:pointer` elements, they may appear anywhere in the XML element's body, and `nite:backpointer` elements must have a `role` attribute. A corpus containing `nite:parent` and `nite:backpointer` elements is considered *read-only*, and these elements must be stripped from the XML documents before any modifications are made. Note that long observations can often be split up into "time slices" stored in separate XML files, which are then recombined through a hierarchy of `nite:stream` elements in one or more additional files.

References

[1] Jean Carletta and Jonathan Kilgour. *The NITE Metadata Format.* `http://www.ltg.ed.ac.uk/NITE/metadata/meta.html`

[2] Jean Carletta, Jonathan Kilgour, Timothy J. O'Donnell, Stefan Evert, and Holger Voormann. The NITE object model library for handling structured linguistic annotation on multimodal data sets. In *Proceedings of the EACL Workshop on Language Technology and the Semantic Web (NLPXML 2003).* Budapest, Hungary, April 2003.

[25] In particular, the `nite:layer` elements simplify the automatic generation of partially specified DTDs.
[26] The prefix `xlink:` must be bound to the XLink namespace `http://www.w3.org/1999/xlink`.
[27] This range does *not necessarily* refer to the element IDs `p_120`, `p_121`, `p_122`, `p_123`, and `p_124`.

[3] D. McKelvie, A. Isard, A. Mengel, M. B. Møller, M. Grosse, and M. Klein. The MATE Workbench – an annotation tool for XML coded speech corpora. *Speech Communication* **33**(1-2), 2001, pages 97–112.
Available from `http://mate.nis.sdu.dk/`

[4] Stefan Evert and Holger Voormann. *The NITE Query Language.*
Available from `http://www.ltg.ed.ac.uk/NITE/documents.html`

[5] Stefan Evert. *NXT XML Format Example.* Available online.

[6] The Unicode Consortium. *The Unicode Standard, Version 3.0.* Addison-Wesley, 2000.
`http://www.unicode.org/`

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation, 6 Oct 2000.
`http://www.w3.org/TR/REC-xml`