A Model of Linking Behaviour

Daniel Yule

April 6, 2014
Contents

Abstract ................................................................. iv
List of symbols ......................................................... v

1. Introduction ......................................................... 1
   1.1. Structure of this document ................................. 3

2. A new Hypertext Model ........................................... 4
   2.1. Document Layer ............................................... 6
   2.2. Display Layer ................................................ 7
   2.3. Context Layer ................................................ 8
       2.3.1. Moving Backwards ..................................... 12
   2.4. Hypertext Layer ............................................ 13
   2.5. User Actions ............................................... 14

3. Algebraic Analysis ................................................. 15
   3.1. Link Semi-automata ......................................... 15
   3.2. Modelling Link Behaviour .................................. 16
       3.2.1. The algebra of the $\ell$-functions ..................... 18

4. Applying the Model ................................................ 20
   4.1. Comparison with other models ............................. 20
       4.1.1. Hypertext Abstract Machine .......................... 20
5. Conclusion and New Applications

A. Hypertext Definitions

References

Glossary
Abstract

This report proposes a model of hypertext focused on link behaviour. Unlike other hypertext models (HAM, Dexter, Amsterdam), this model allows for alternate actions to take place in response to selecting a link. The model consists of four layers: document (the organization of information), display (the physical representation of information), context (the user’s environment) and hypertext (changes to the user’s context in response to their actions). The model is compared to existing models, and is applied to existing hypertexts and hypertext systems.
List of symbols used in model

**Standard Symbols**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}(A)$</td>
<td>The powerset of set $A$</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>The real numbers</td>
</tr>
<tr>
<td>$\mathbb{N}$</td>
<td>The natural numbers</td>
</tr>
<tr>
<td>$\mathbf{M}_{n,n}$</td>
<td>An $n \times n$ matrix of real numbers</td>
</tr>
</tbody>
</table>

**$\ell$-functions** see §3.2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_{\text{create}}$</td>
<td>: create a new window</td>
</tr>
<tr>
<td>$\ell_{\text{remove}}$</td>
<td>: remove content from a window</td>
</tr>
<tr>
<td>$\ell_{\text{destroy}}$</td>
<td>: destroy a window</td>
</tr>
<tr>
<td>$\ell_{\text{insert}}$</td>
<td>: insert a node into a window</td>
</tr>
<tr>
<td>$\ell_{\text{transform}}$</td>
<td>: change the transformation function of a window</td>
</tr>
</tbody>
</table>
## Defined Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning (with link to definition in document)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{M}$</td>
<td>The universe of media ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>The universe of all documents ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>The universe of all presentations of documents ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{W}$</td>
<td>The universe of all windows ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{V}$</td>
<td>The universe of all views ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>The universe of all environmental information ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{C}$</td>
<td>The universe of user contexts ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{A}$</td>
<td>The universe of user actions ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>The universe of $\ell$-functions ($\text{def}^n$)</td>
</tr>
<tr>
<td>$m_{\text{data}}$</td>
<td>The data contained in a chunk of media $m$ ($\text{def}^n$)</td>
</tr>
<tr>
<td>$m_{\text{type}}$</td>
<td>The type of the media chunk $m$ ($\text{def}^n$)</td>
</tr>
<tr>
<td>$s$</td>
<td>A selector function ($s : M \to M$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$n$</td>
<td>A node function ($N : C \to M$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$w$</td>
<td>A window $(\text{region, location, projection, } p)$ ($\text{def}^n$)</td>
</tr>
<tr>
<td>$c$</td>
<td>The user’s context ($c : T \to \mathcal{E} \times \mathcal{V}$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$T$</td>
<td>A set to represent time ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{render}$</td>
<td>A rendering function ($\text{render} : \mathcal{D} \to \mathcal{P}$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{proj}$</td>
<td>A linear function that transforms the presentation within a window ($\text{proj} : \mathcal{P} \to \mathcal{P}$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{proj}_{\text{trans}}(\Delta x, \Delta y)$</td>
<td>A projection function that transposes by $(\Delta x, \Delta y)$</td>
</tr>
<tr>
<td>$\text{history}$</td>
<td>The history function ($\text{history} : T \to \mathcal{V}$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{sensor}$</td>
<td>The sensory function ($\text{sensor} : T \to \mathcal{E}$) ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{update}$</td>
<td>A function $\text{update} : \mathcal{C} \times \mathcal{L} \to \mathcal{C}$ which changes the view. ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{now}$</td>
<td>A function $\text{now} : \mathcal{C} \to T$ which gives the current time in the system ($\text{def}^n$)</td>
</tr>
<tr>
<td>$\text{undo}$</td>
<td>The inverse of the $\text{update}$ function ($\text{def}^n$)</td>
</tr>
</tbody>
</table>
1. Introduction

The central concept of most modern hypertext, that which differentiates it from other forms of media, is the link. Different systems treat the link differently, but it lies beneath every extant hypertext system. This report concerns itself with links which change the user’s context in response to a user action, although other types of linking are possible, such as spatial hypertext. These other concepts are outside the scope of this paper, although I will briefly discuss spatial hypertext in Section 4.2.2.

Researchers have studied the structure of links (e.g. [26], [38] or [31]), the topology imposed by links (e.g. [3] or [43]), different types of links (e.g. [24], [1] or [42]) as well as different ways of linking (see [7] for an early overview, or [27] for a somewhat more recent one), but there has been little study of what happens when a link is activated. This report proposes a mathematical model for link behaviour, and investigates its applications.

Before I can discuss the concept of linking in more detail, I must first define what hypertext is. Hypertext is commonly understood to mean “text with links in it,” but there is considerably more subtlety to the idea than that. Computerized hypertext was coined and described in the early 1960s by Theodore H. Nelson and implemented independently by Douglas Englebart around the same time. Nelson defined it simply “to mean a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper.” [33]. Englebart framed it in more of a sense of what it enables, defining hyper “as a prefix meaning to enhance access, manoeuvrability, study-ability and (re)utilization.” [10].
There have been many attempts to find an accepted definition for hypertext (see Appendix A or the discussion on grandTEXTauto in 2004 [48]), but so far there has been no agreement. For the purposes of this document, I will follow Conklin’s lead from his seminal overview of the field [7] and concern ourselves particularly with hypertexts involving links. I therefore define the following terms (for more elaboration, please see the glossary):

- **node** A chunk of content (or lexia [2])
- **link** An action that changes the structure of nodes visible to the user
- **hypertext system** A collection of nodes and links

This definition uses *text* in its looser sense, meaning simply a body of work, and so includes all types of media.

A **node** is a chunk of content, but this definition does not establish where this content might have come from. Classical hypertext systems (Intermedia [49], HAM [5], gIBIS [8], etc.) as well as the standard Dexter Model[§4.1.3], assume that chunks are static, i.e. stored in a database somewhere and then retrieved when needed. They could be edited, of course, but if no one was editing the system, every user would see the same content in the same node.

*Adaptive hypertext* paved the way for a new type of node: a **dynamic node**. A dynamic node changes based on the user’s **context**, that is, it changes based on the user’s past navigation of the system, as well as other information gleaned from sensors or from a user’s profile.

Of course, if nodes can be dynamic, so too can links. In some systems e.g. Aquanet [29]) links are a type of node, so in this case, so there is no sensible distinction between them. If a link is a change in structure in response to a user’s action, then how the structure changes can be modified dynamically based on user **context**.
Most hypertext systems assume that activating a link happens a) by a user’s explicit selection and b) replaces the entire view with new content. This is the default paradigm on the most popular hypertext system, the World Wide Web. However, many other link behaviours are possible.

1.1. Structure of this document

In the following chapter, Chapter 2, I will define the model used for describing hypertext behaviour. This is a model of a complete hypertext system although the focus is on link behaviours. Chapter 3 will give a more detailed algebraic analysis of the proposed model. Chapter 4 will discuss application of the model to existing systems and models. Finally, Chapter 5 will summarize the findings so far and suggest some new applications of the model.
2. A new Hypertext Model

There are several existing models of hypertext systems (e.g. Trellis§4.1.2, the Dexter model§4.1.3 and the Amsterdam model§4.1.4) but none deal with link behaviour; indeed all three implicitly assume the default behaviour discussed above.

The following model is inspired by the three models mentioned above, but does not directly use any of their ideas. For a comparison of this model with existing models, see section 4.1. Instead, this model is most directly influenced by Rosenberg’s discussion of hypertext activity [38]. This model is implicitly concerned with consumption of hypertext, as opposed to authoring, but providing a model for reading is an essential step in understanding how to write.

Figure 2.1 gives an overview of the system. The model is comprised of objects, functions, function objects and relationships. **Objects** are elements of the system which contain data or other objects. **Functions** transform one object into another object. **Function objects** act like objects, but are defined as a function. A computer program file is an example of a function object. It is made up of data, but that data can be interpreted as instructions, which can then transform other data (or itself). Finally, **relationships** are between two objects. There are two types of relationships used in this model: *has-a* and *uses*¹.

The model defines four layers, each of which builds on the previous ones. The four layers are: the **document layer**, the **display layer**, the **context layer** and the **hypertext**

---

¹These relationships are borrowed from Software Engineering. See any Software Engineering text (for example Sommerville [39]) for an overview of the relationships
Figure 2.1.: The proposed hypertext model
layer. The **document layer** defines the media chunks used by the system, and the documents made out of them. The **display layer** defines the presentation of a document, the windows to show presentations and the view to collect the windows. The **context layer** defines the history of views and the sensory data from the environment. Finally, the **hypertext layer** defines nodes and links. I will now detail each layer in turn.

### 2.1. Document Layer

The document layer is concerned with the fundamental building blocks of hypertext: media chunks and the documents made out of them. A **hypertext** can contain any type of media. I define a **media chunk** \( m \) as a double \((m_{type}, m_{data})\) where \( m_{type} \) consists of type information and \( m_{data} \) is a linear sequence of data. Note that some types of media have a hierarchical or other non-linear structure, (such as an XML document or a video game) but they must be represented linearly. This restriction is not onerous as any computer files are necessarily stored linearly as well.

The model does not dictate how the type information is provided, simply that it must be. Example type systems include MIME [11] and file extensions. I then define \( \mathcal{M} \) as the set of all media content the system can present.

A **document** is an ordered tree of media. The value of each node in a tree can be empty or a chunk of media. The media itself can be hierarchical (e.g. an SVG image) or even a hypertext itself, but that does not affect the document tree. Although the tree separates the various media it contains, it allows for mixed, non-linear layout of the media, through the **render** function defined in Section 2.2.

I define \( \mathcal{D} \) as the universe of all documents possible in a system.

The document layer also defines the concept of a **selector**. A selector filters the document, leaving only a portion. This portion can include sub-documents, media chunks, parts of media chunks, or all three.

A selector is a function \( s: \mathcal{D} \rightarrow \mathcal{D} \), with the constraint that \( s(D) \) is isomorphic to a
subtree of $D$. Further, each media chunk in $s(D)$ must be a subset of its corresponding media chunk in $D$.  

The universe of all selectors is denoted $S$. 

2.2. Display Layer

A hypertext is bound to the technology used to present it. For example, Walker’s analysis\cite{46} of \textit{afternoon a story}\cite{21} is a hypertext. It is available both as a standard paper (physically printed) and as an HTML document\cite{3}. The document’s structure may be identical whether it is printed or in HTML, but the manner in which that structure is presented is different. On the web, links to other works are followable hyperlinks, and the headings are coloured. In printed form, the document is presented in the standard two column format, without coloured headers or clickable links.

Although the two documents have identical content, they have a different \textit{presentation}. I define a \textit{presentation} to be the physical manifestation of a \textit{document}. Some examples of presentations include a raster image (for a computer screen) or a printed page (for a book).

Because of the wide variety of technologies available, and the impossibility of predicting future ones, the model takes as primitive $\mathcal{P}$, the universe of presentations of content. A presentation $p$ is an $n$-dimensional array. The most common type of presentation is a two dimensional image, but it can also be sound, haptic feedback or any other form of media output.

Since a \textit{document} is only a structural representation of media, before it can be consumed by a person, it must be \textit{rendered} into a \textit{presentation} by a function $render$ defined as $render : D \rightarrow \mathcal{P}$.

\textit{Presentations} are stored within \textit{windows}. A window is an $n$-dimensional viewport

\footnote{I define the subset of a vector as is commonly done in programming languages (R, MATHLAB, Python) as containing some or all of the same elements as the original vector, with their order preserved.}

\footnote{see \url{http://jilltxt.net/txt/afternoon.html}}
showing either all of or a portion of a document’s presentation. A window may also contain other windows, which will occlude the presentation of the containing window.

A window \( w \) is defined as a quadruple

\[
w = (p, region, location, container, projection)
\]

(2.1)

where \( p \) is a document’s presentation, \( region \) is an \( n \)-dimensional surface (usually a rectangle with the top left at the origin) that dictates the viewable part of the window, \( location \) is a \( (n,\text{-tuple}) \) that specifies the offset of the window’s origin from its containing \( view \), \( container \) is either a window or \( view \) (defined shortly) that this window is contained within and \( proj \) is a projection function that represents a transformation on a presentation \( p \). That is, \( proj \) is a linear endomorphism of the form \( proj : P \rightarrow P \).

The process of displaying a window is shown graphically in Figure 2.2.

In the most common cases, \( proj \) will simply be a function which scrolls the presentation some amount. I will denote a function which scrolls the presentation \( \Delta X \) horizontally and \( \Delta Y \) vertically as \( proj_{\text{trans}}(\Delta x, \Delta y) \). For a more precise definition of how \( proj \) works, see Figure 2.3.

I denote the universe of all windows as \( \mathcal{W} \).

A \( view \) \( v \) is defined as a set of windows. The universe of views \( \mathcal{V} \) is given by \( \mathcal{V} = \mathcal{P}(\mathcal{W}) \).

### 2.3. Context Layer

An important component of hypertext is that it can change based on the user’s past behaviour. As such, I define the \( history \) as the \( view \) at any particular time. That is, I define an ordered infinite set \( T \) to represent time. \( T \) may be either continuous or discrete. A computerized hypertext will typically use the Natural numbers \( \mathbb{N} = \{0, 1, 2, \ldots\} \), with each number corresponding to milliseconds (or ticks) passed. History is defined by the function \( history : T \rightarrow \mathcal{V} \).
(a) The window’s position is shown relative to its container, as well as its visible region, designated by diagonal shading.

(b) The presentation associated with a window uses the coordinate system of the window, with the origin at \((x, y)\).

(c) The presentation is transformed by a projection function. In this case, it is translated (or scrolled) by \(\Delta x\) and \(\Delta y\) using the function \(projtrans(\Delta x, \Delta y)\) (see Figure 2.3).

(d) Finally, the presentation is clipped such that only the area within the visible region is shown.

Figure 2.2.: The process of displaying a window
For an overview of the linear algebra in this section, see for example [35], or any text on computer graphics.

Since $\mathcal{P}$ is an $n$-dimensional space, we usually represent points in that space with an $n$-tuple $(\dim_1, \dim_2, \ldots, \dim_n)$. However, in order to conveniently represent common projection functions, I instead use homogeneous coordinates of the form $(\dim_1, \dim_2, \ldots, \dim_n, 1)$.

Then, as $\text{proj}$ is a linear mapping from $\mathcal{P}$ to $\mathcal{P}$, it can be represented as a $n+1 \times n+1$ matrix.

In the most common two dimension case, a point $(x, y)$ can be represented as $(x, y, 1)$ and the projection function as a $3 \times 3$ matrix like so:

$$
\text{projection} = \begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
$$

where $a, b, c, d, e, f, g, h$ and $i$ are real numbers.

If $p = (\dim_1, \dim_2, \ldots, \dim_n, 1)$ is a point in a document’s presentation, then its position in the window will be given by $\text{proj} \cdot p$. To take the two dimensional example above, the point $p$’s position in the window $p_w$ will be given by

$$
p_w = \text{projection} \cdot p = \begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
$$

The most common case for $\text{proj}$ will be two dimensional translation (i.e. scrolling), resulting in a matrix of the form

$$
\text{projection}_{\text{trans}(\Delta x, \Delta y)} = \begin{bmatrix}
1 & 0 & \Delta x \\
0 & 1 & \Delta y \\
0 & 0 & 1
\end{bmatrix}
$$

where $\Delta x$ and $\Delta y$ are the amounts to scroll the presentation horizontally or vertically respectively. However, other transformations (rotation, scaling, skew) are possible as well, both separately and together.

Figure 2.3.: A note on the linear algebra of the projection function
Modern computers, especially mobile ones, have a wide array of sensory information about their environment (e.g. GPS, accelerometer, camera). Adaptive hypertext takes advantage of this information, as well as any other information that can be inferred about the user to create content and structure specific to them. Defining exactly the kinds of information collected is outside the scope of this model, so I define the \textit{environment} as a collection of measurements about the environment. So, I define $\mathcal{E}$ as the universe of all possible sensor readings, and a function for finding what the sensors were at a particular time as $\text{sensor} : T \to \mathcal{E}$.

Finally, the user’s context involves not only their browsing history (as represented by $\text{history}$), but their environment (as represented by $\text{sensor}$). So, I define the user’s context $c$ as a function $C : T \to \mathcal{V} \times \mathcal{E}$ where

$$C(t) = (\text{history}(t), \text{sensor}(t))$$  \hspace{1cm} (2.4)

That is, the user’s context is a function from time to the user’s view and sensor data at that time. I define the universe of all contexts as $\mathcal{C}$.

Note that although $c$ is defined as a function, it is more appropriate to think of it as a sequence of doubles indexed by $T$. That is, $C$ can be defined as

$$c_t = (\text{history}(t), \text{sensor}(t))$$  \hspace{1cm} (2.5)

and this is the notation that will be used from here on.

When the view changes, it changes based on the context. Functions which change the view are called $\ell$-\textit{functions}, and are defined as follows

$$\ell : \mathcal{C} \to \mathcal{V}$$  \hspace{1cm} (2.6)

I call the universe of all $\ell$-functions $\mathcal{L}$. $\ell$-functions are described in more detail in...
Section 3.2.

I also define a function $update$ used for changing the context. The $update$ function is defined by $update : \mathcal{C} \times \mathcal{L} \rightarrow \mathcal{C}$.

That is, the $update$ function uses an $\ell$-function to change the context. It does so by updating the $history$ such that the history at the given time is the output of the $\ell$-function. More precisely,

$$update(c, \ell)_t = \begin{cases} c & \text{if } t < now(c) \\ (\ell(c), sensor(t)) & \text{otherwise} \end{cases}$$  \hspace{1cm} (2.7)

The $now$ function referenced above in (2.7) is defined as

$$now : \mathcal{C} \rightarrow T$$  \hspace{1cm} (2.8)

It simply gives the time $t$ that corresponds to the current state of the system. $t$ could be the number of milliseconds elapsed since the user began their browsing session, or it could be the number of user actions taken. The only requirement on $now$ is that it must increase (not necessarily monotonically) with the passage of time.

2.3.1. Moving Backwards

Most hypertext systems have a mechanism for going back to a previous state. However, as Rosenberg [38] points out, there are two different semantic meanings to this action. The first is that the reader has finished with what they are reading, and wish to go back to their previous view to pursue a different path. The second is that the reader followed a link by mistake and does not wish to view this node.

In the first case, the context should be updated to remember that the user has seen this view. As we shall see in Section 2.4, this is really just a type of $link$, and can be treated as such.
The second case is more of an undo. This action is built into the system, through the function \( \text{undo} : \mathcal{C} \rightarrow \mathcal{C} \). The undo function is defined as the inverse of the \( \text{update} \) function.

Note that I do not address the common “forward” button found in many web browsers, as this concept views history as a stack, rather than temporal as the proposed model does. See [20] for more.

### 2.4. Hypertext Layer

A **node** is a function \( n : \mathcal{C} \rightarrow \mathcal{M} \). That is, a node is a function from the user’s context to some media content. For the case where \( n(c) \) is constant for all \( c \in \mathcal{C} \), then \( N \) is a **static node**. Otherwise, it is a **dynamic node**.

Before I define a link generally, I must briefly describe its behaviour. A link’s behaviour is a series of functions that update the view. More precisely, a link behaviour \( B \) is a function \( B : \mathcal{C} \rightarrow \mathcal{C} \) defined in terms of a series of \( \ell \)-functions \( \ell_1, \ell_2, \ldots, \ell_n \) of the form \( \ell_i : \mathcal{C} \rightarrow \mathcal{V} \) such that

\[
B(C) = \text{update}(\text{update}(\ldots\text{update}(C, \ell_n), \ldots), \ell_2), \ell_1) \tag{2.9}
\]

That is, a behaviour changes the current view using an \( \ell \)-function and the \( \text{update} \) function defined in Section 2.3. The \( \text{history} \) function is updated to reflect the change in the view. For more information on \( \ell \)-functions, see Section 3.2.

**Links** are a critical component of a **hypertext**. In defining them, I follow Jim Rosenberg’s [38] idea of a link as a change in document structure in response to a user action. That is, a link is defined

\[
L : \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{C} \tag{2.10}
\]

where \( \mathcal{A} \) is the universe of all possible user actions on the system.

So a link is a function that modifies the view in some way based on a user’s action.
Some examples of user actions are clicking an anchor, scrolling a window or twisting a “more detail” knob [34].

2.5. User Actions

Traditional hypertext systems usually consider only one type of user action, that is the clicking of a mouse. Different systems had different ways of responding to mouse clicks, but a link was typically selected as a result of a mouse click. Certainly this is a critical part of the proposed system. A mouse click on a window is one of the user actions in $\mathcal{A}$.

However, many other types of user actions are possible. As early as 1970, Nelson was proposing the use of a throttle to control the amount of information present in the text [34]. Pull back the throttle, and only a summary of the ideas would be present. Push it forward, and detailed explanations and arguments would insert themselves into the text. Such a situation can be handled gracefully by the proposed model. In this case, the user action being taken is moving the knob, and the $\ell$-functions would consist of $\ell_{\text{insert}}$ when the throttle is pushed and $\ell_{\text{remove}}$ when the throttle is pulled.

Other user actions might be scrolling the window, or otherwise changing the presentation’s projection function. It might include a mouse hover, or in a system with eye tracking, it might respond to eye movement. The system might respond to more complicated gestures, such as the drawing of a circle on a touchpad. Or, in a tangible interface (such as a book), it might involve the user physically moving something in their environment (such as turning a page). This model does not attempt to proscribe what user actions a system should respond to, just that it must respond to some.
3. Algebraic Analysis

3.1. Link Semiautomata

A semiautomaton is a triple \((Q, \Sigma, \delta)\) where \(Q\) is a non-empty set of states, \(\Sigma\) is an input alphabet and \(\delta\) is a transition function that dictates the change in state in response to input. When \(Q\) is finite, a semiautomaton is analogous to a finite state machine with no starting state or output. [18]

If we let the set of states \(Q\) be equal to the universe of contexts \(C\), and the input alphabet to be \(A\), the universe of all user actions, then our transition function \(\delta\) is simply \(L\), the link function.

Hypertext has been analysed in the context of state machines previously. Park [37], following Rosenberg [38], provided a framework for modeling hypertext by drawing on formal language theory. Park’s model was based on a finite state automata analysis of the Dexter model.

Park’s analysis is very powerful, as there has been much work on finite state automata, which can be immediately applied to hypertext analysis. However, the Dexter model (see §4.1.3) that forms the basis for this analysis is limited in that it assumes a finite number of states, and therefore a finite number of nodes. Although this works very well for many self-contained hypertexts, a global hypertext such as the World Wide Web will have an effectively infinite number of states\(^1\).

\[^1\] Although the number of hosts and nodes on those hosts at any moment in time is finite, the number is growing so quickly that modelling the number nodes as infinite is simpler than assuming it is finite.
My proposed model gives up much of the theoretical advantage of Park’s FSA model, but provides a framework for discussion and analysis of all link based hypertext.

3.2. Modelling Link Behaviour

A link is some change to the view of the user, but this does not tell us very much about what kinds of changes might be made. In order to do so, I break down link behaviour into \( \ell \)-functions: \( \ell_{create} \), \( \ell_{remove} \), \( \ell_{destroy} \), \( \ell_{insert} \) and \( \ell_{transform} \), defined as follows.

\[
\begin{align*}
\ell_{create} : & \mathcal{C} \to \mathcal{V} \quad (3.1) \\
\ell_{remove} : & \mathcal{C} \to \mathcal{V} \quad (3.2) \\
\ell_{destroy} : & \mathcal{C} \to \mathcal{V} \quad (3.3) \\
\ell_{insert} : & \mathcal{C} \to \mathcal{V} \quad (3.4) \\
\ell_{transform} : & \mathcal{C} \to \mathcal{V} \quad (3.5)
\end{align*}
\]

An \( \ell_{create} \) function generates a new window with an empty document\(^2\). \( \ell_{remove} \) is a function that removes a part of a document. \( \ell_{destroy} \) closes a window. \( \ell_{insert} \) puts the content of a node into a document. \( \ell_{transform} \) changes the transformation function of a window (typically causing it to scroll to a new location).

The functions themselves do not describe what precisely they do (which node is being inserted by an \( \ell_{insert} \) function, for example), but their generator functions do. A generator function is a partial function which takes arguments specifying the action of the \( \ell \)-function it generates, and whose output is an \( \ell \)-function.

For example, the generator function for \( \ell_{insert} \) has the form

\[
insert : \mathcal{W} \times \mathcal{S} \times \mathcal{N} \to \mathcal{L} \quad (3.6)
\]

\(^2\)An empty document is a tree with one valueless node.
That is, given a window \( w \), a selector \( s \) and a node \( n \), the insert generator will create an \( \ell \)-function that inserts \( n \) at \( s \) in window \( w \). The other generator functions work similarly:

\[
create : \mathbb{R}^n \times \mathcal{P}(\mathbb{R}^n) \times \mathcal{W} \cup \mathcal{V} \rightarrow \mathcal{L} \tag{3.7}
\]

\[
remove : \mathcal{W} \times \mathcal{S} \rightarrow \mathcal{L} \tag{3.8}
\]

\[
destroy : \mathcal{W} \rightarrow \mathcal{L} \tag{3.9}
\]

\[
transform : \mathcal{W} \times \mathbb{M}_{n+1,n+1} \rightarrow \mathcal{L} \tag{3.10}
\]

The create function generates an \( \ell \)-function that adds a window to the current view. The first argument is a point in \( n \)-dimensional space indicating the location of the window. The second argument, is a region that defines the clipping surface for the window. This will usually be a rectangle. The last argument is the window or view that contains this window.

The remove function generates an \( \ell \)-function which removes the media designated by a selector from a window. The first argument is the window to remove the media from, and the second is the selector specifying which media to remove.

The destroy function generates an \( \ell \)-function that removes a window from the current view. It is the opposite of the create function. Its only argument signifies which window to destroy.

The transform function generates an \( \ell \)-function that changes the transformation function of a window. The first argument is the window to change the transformation on and the second is the new transformation function, represented as a \( (n + 1) \times (n + 1) \) matrix (see Figure 2.3).

Note that a typical link’s behaviour is made up of several \( \ell \)-functions executed in sequence (see §2.4). For example, the default behaviour of a web browser opening a link
is an $\ell_{\text{remove}}$ function (to remove the page currently in the window) followed by an $\ell_{\text{insert}}$ function that inserts the media generated by the requested page. See Section 4.2.3 for a detailed application of this model to the World Wide Web.

Note that it is entirely possible to invoke, for example, an $\ell_{\text{destroy}}$ function on a context which does not have the window referenced by the $\text{destroy}$ generator function. In this case, $\ell_{\text{destroy}}$ simply maps the context to its most recent view.

The $\ell$-functions form an algebra of link behaviour. By combining the five $\ell$-functions defined here in different ways, many different kinds of link behaviours can be achieved. The definition of these functions is the major contribution of this work.

3.2.1. The algebra of the $\ell$-functions

This section investigates the $\ell$-functions from the perspective of abstract algebra.

Although I define the algebra in terms of the $\ell$-functions, to be completely correct, I mean an $\ell$-function composed with the $\text{update}$ function, as discussed in Section 2.4.

I have said that the $\ell$-functions form an algebra; however, I have not said what kind of algebra they form. At first they might seem to be an excellent candidate for a sort of group [18], there is a problem. A group must satisfy these conditions:

1. Have a single operation (in this case, function composition)
2. Be closed under that operation
3. Be associative
4. Have an identity element
5. Every element of the group must have an inverse.

Any kind of function composition satisfies the first four conditions, but only bijections are invertible, and these functions are not bijective. In particular, they are not one-to-one. For example, for two $\ell_{\text{destroy}}$-functions $\ell_1$ and $\ell_2$, where $\ell_1$ destroys window $w_1$ and
\( \ell_2 \) destroys window \( w_2 \), for a context whose current view contains neither window, they will both map to the same value.

As such, the \( \ell \)-functions form a monoid, which is a group without inverses. In fact, the \( \ell \)-functions form a left monoid action together with \( \mathcal{C} \). A monoid action is a binary operation \( \cdot \) of the form

\[
\cdot: M \times S \to S
\]

where \( M \) is a monoid with the operation \( \ast \) and \( S \) is any set, together with the rule that for all \( m, n \in M \) and \( s \in S \),

\[
m \cdot (n \cdot s) = (m \ast n) \cdot s \tag{3.12}
\]

In this case, the monoid is \( \mathcal{L} \) and the set is \( \mathcal{C} \). The operation that brings them together is application. Using the above notation, the operation \( \cdot \) is defined as

\[
\ell \cdot c = \ell(c) \tag{3.13}
\]

Therefore, we can represent link behaviour as a string of symbols representing \( \ell \)-functions like so \( \ell_1 \ell_2 \ldots \ell_3 \). The associated behaviour will be the sequential application of these functions.
4. Applying the Model

4.1. Comparison with other models

There are many existing linked hypertext models [14, 41, 51]. However, with the exception of the Trellis model (§4.1.3), they assume static links, and none of them can handle modelling link behaviour.

There is also the related idea of a computational model of user navigation, such as SNIF-ACT [12], Bloodhound [6] or CoLiDeS [23]. However, these theories are for modelling the user’s thought process, not the system itself, which the proposed system is designed for.

Here, I compare the proposed model with three of the most popular existing hypertext models and show that each can be mapped into it.

4.1.1. Hypertext Abstract Machine

The Hypertext Abstract Machine (HAM) [5] was one of the first attempts at a hypertext model. It was designed to be used as the back end of a system, i.e. it did not include a user interface. Other programs would work as the front end, displaying the content stored in the HAM. HAM was designed around both reading and authoring, so there are some features which do not easily translate into the proposed system.

The HAM has five components: graphs, contexts, nodes, links and attributes. The graph is a hypertext document, roughly (and confusingly) corresponding to the context of the proposed system. The context of a HAM system is a sophisticated concept related
to authoring, which has no direct analogue in the proposed system, or indeed any subsequent hypertext model. Contexts in a HAM system are a kind of version controlled workspace. Each one contains a version of each node and link within it, with connections to previous versions of nodes.

Nodes in HAM are very similar to nodes in the proposed system, except that HAM nodes are editable and are not dynamically generated. Links are very different, as they function like conventional hypertext links, connecting two nodes in either or both directions. Attributes are simply key value pairs representing information about the system.

Although HAM has some interesting features, such as built in version control and bi-directional linking, it makes too many assumptions about the underlying system to be truly abstract. Furthermore, it lacks any specification of link behaviour, as it is only concerned with the storage of hypertext, not its navigation.

4.1.2. Trellis

Trellis [13, 40] can be considered an extension of the graph-based models used in most other hypertext models (e.g. Dexter §4.1.3). Trellis uses Petri net formalism to analyze hypertext structures. Petri nets have an advantage over the directed graph formalism in that they specify sophisticated execution semantics.

In Trellis, nodes are represented as Petri net places, and links as Petri net transitions. A place can have a token in it, in which case the place is called “marked.” Nodes which are marked are the nodes which are displayed to the user. Transitions are multi-links, with an arbitrary number of input places and output places. When all input places for a transition are marked, then the transition is enabled, and may fire. When it fires, it removes the tokens from its input places and adds them to its output places. Users choose which transition to fire by selecting a link from a menu.

Using this model, Trellis is able to specify a very complex relationship between nodes.
Instead of a link being between two nodes, a link is between two collections of nodes. With the addition of coloured markers for multiple simultaneous browsers and timings on transitions, the Trellis model is able to describe sophisticated hyperprograms.

Trellis is closer to the proposed model than any other model discussed here, in that it treats a hypertext as a kind of automata to be executed, but it still too inflexible and does not go far enough in discussing the changes to the structure that might take place at a link. The Trellis model can be emulated using the proposed model, by creating links that are only active when certain nodes are displayed.

Trellis, and its successor caT [32] still assume that the entirety of the hypertext is finite and that selecting a link will completely replace the visible content. They do not adequately model link behaviours such as inserting content within an already displayed node.

4.1.3. Dexter Model

The Dexter Model [16] was created between 1989 and 1990. It was intended to be a model that could be applied to any sort of hypertext system, and is still mostly used as the reference model today.

The Dexter model is based on three layers: the storage layer, the within-components layer and the runtime layer. The storage layer is made up of components (nodes). These components have data in them (text, images, sound) as well as meta-data (title, author, anchors). Links are a particular type of component, the content of which specifies two or more anchors.

An anchor can be specified through three layers of indirection. First, the specifier function indicates which UIDs are involved. Second, the accessor function maps the UIDs to the components they represent. Thirdly, the anchor information within the component indicates what region of the component is the target or source of the link.

The within-component layer is responsible for managing the hierarchy of components.
A composite component can be made out of other components. The only requirement is that a component cannot directly or transitively include itself.

The runtime layer is concerned with presenting the components and links to the user. It does not specify how they should be presented, simply that they are. It allows for dynamic computation of the presentation of links and components based on runtime information.

Many of the parts of the Dexter model have direct analogues in the proposed model. The components of the Dexter model correspond to nodes in the proposed model, although nodes are dynamically generated based on the context, whereas components are static.

The Within-Component layer of the Dexter model is roughly similar to the Document Layer (§2.1) of the proposed model. Both specify a hierarchical collection of component’s content. The proposed system says that atomic components (called media) are different from composite components (called documents), whereas Dexter lumps them in as one, but the idea is the same.

The presentation specification is where the dynamic nature of components appears in the Dexter Model. This corresponds to the render function of the proposed model, which can also vary depending on the context. As such, presentations can be changed depending on how the document is accessed.

There are other small differences between the systems: the proposed system has no concept of UIDs (it simply assumes the system can specify a node, but does not say how), and does not enforce the rule that all links must point to an existing node.

A major divergence between the Dexter model and the proposed one is the treatment of links. The Dexter model essentially treats a hypertext as a graph, where components are vertices and links are edges. The proposed system also uses a graph, but vertices are states, and the edges are transitions.

Since links are treated so differently, the concept of anchors from the Dexter model
also are not present in the proposed model, rather they are replaced by selectors. A link in the Dexter model can be seen as a pair of selectors, which generate a link function in the proposed model. The link is a click action in a region, specified by the first selector mapped to an $\ell_{\text{remove}}\ell_{\text{insert}}$ action, where the remove is on the current window, and the insert is specified by the second selector.

The Dexter Model does not specify any user interactions, whereas the proposed model takes them as primitive, and uses them as the basis of linking. In abandoning the graph model, some model checking ability is lost, but much more expressive power is gained. The Dexter model has no facility for viewing more than one component at a time, or for changing links based on what is currently visible. It also has no facility for modelling link behaviour, which is the primary purpose of the proposed system.

4.1.4. Amsterdam Model

The Amsterdam Model[17] is specifically designed to describe time-based multimedia. It extends the Dexter model to handle time and also to discuss presentation at a more sophisticated level. As such, the Amsterdam model defines various presentation attributes that define how a composite node is presented. A great deal of thought is put into how to synchronize media that is being viewed, when multiple media types are displayed concurrently. Much of the model’s extension is based around this idea.

The proposed model does not explicitly handle synchronization of multimedia, although it can be handled through defining links whose behaviour starts only after a certain amount of time.

Although with some clever use of automatic links, the proposed model could be used to emulate the Amsterdam model, the Amsterdam model’s handling of multimedia and presentation is easier and more direct. For modelling multimedia presentations, Amsterdam is an appropriate choice. However the proposed model is more suitable for modelling link behaviour.
The proposed model explicitly deals with different types of media, and through the concept of *documents* can represent more than one media type on a page. However, there is no concept of synchronization, or any finesse in terms of how time-based media relate to each other. Adopting the temporal logic and channels of the Amsterdam model is a direction for future research.

4.2. Hypertext Systems

4.2.1. Books

Although hypertext is often defined in terms of what it offers above and beyond standard text (see Appendix A, for example), a traditionally bound book (i.e. a codex) is a hypertext, just perhaps not as sophisticated as a computerized one.

In applying the model to a book, there are several options. We can consider each page a *node*, and the currently visible page as a *window*. The act of turning the page is following a link, whose ℓ-functions are a remove (to remove the text of the visible page) and an insert (to insert the text of the next page). Alternately, we could think of the entire book as a node, and each page has a different projection function. Or perhaps each lexia[2] is its own node, with each page a document made out of several lexia, and scrolled so that its starting point corresponds with the ending point of the previous page.

4.2.2. Spatial Hypertext

Spatial hypertext [30] is a radically different way of thinking about hypertext, which the proposed model does not handle well at all. In a spatial hypertext, nodes are related together by their physical location rather than through a link. Discussing link behaviour does not make any sense in this context. Furthermore, spatial hypertexts are explicitly designed to operate in environments where the readers and writers are the same people, whereas the proposed model is designed only for reading.
Although the proposed model can be used to display a spatial hypertext, through assigning one node per window, and using windows as the spatial nodes, without editing and the ability to associate nodes visually, the proposed model is not very useful for such circumstances. Describing spatial hypertext is outside the scope of this model.

4.2.3. The World Wide Web

The World Wide Web is the most popular hypertext system today. Although it seems to lack many of the features of other hypertext systems, the modern web is at least as powerful as any other hypertext system [50], in the sense that any other hypertext system can be created using web technologies.

This model handles many web behaviours, much more than the default of “click to follow a link, which replaces the page.” On the web, pages directly correspond to nodes, and HTML documents correspond to this model’s documents. Windows are browser windows, or tabs (which for the purposes of the model are a special kind of window).

The only component of this model without a direct analogue on the web is the undo function. The browser’s back button is simply a go-back link (as discussed in Section 2.3).

The anchor element, represented by the a tag affords opportunities for many types of linking, aside from the basic replacement. The use of the fragment identifier in URIs allows for linking to specific parts of a document, or to a different part of the visible document. That is, for some links the only action is an \( \ell_{\text{transform}} \), or includes an \( \ell_{\text{transform}} \) as the last step. For a discussion of within document vs between document linking, see Gunder’s Aspects of Linkology [15].

Also, through the use of the target attribute of the a tag in HTML, clicking a link on a page can cause an already loaded window to change its content, or for a new window to be created, meaning that the web can allow, in a basic way, for actions on one window to affect those in another. This feature is much more prominent in systems like Trellis [13]
(and indeed, the authors of Trellis tried to bring this concept to the web [25] although it did not catch on), but it can still be modelled, as the proposed system does not limit behaviours to only affecting windows where the user action took place. Indeed, it is possible that a user action could take place not on any window (such as turning a knob in their environment), and still the contents of windows could be affected.

As described above, the default link behaviour is to completely replace the contents of a window, but through JavaScript and certain HTML elements, much more can be accomplished.

Firstly, through the usage of inline frames, entire pages can be embedded within other pages. These inline frames can be inserted as the page loads, meaning that the link resulting in the page load has an extra $\ell_{\text{insert}}$ function associated with it, or they can be inserted by JavaScript, in which case the action that set off the JavaScript is a link action (for example, pressing a button, or filling out a form).

Even without inline frames, new content can be added to websites in response to user actions, through the use of JavaScript and the Model View Controller paradigm. A new node is requested from the server, then processed by the JavaScript before being inserted into the page, possibly replacing some of the content. An example is Google’s GMail webmail service. The page initially shows a listing of the email that the user has received. When a user clicks on a message from the list, the list is removed (via an $\ell_{\text{remove}}$) and the new message is retrieved from the server, then inserted in the space where the list was removed from. The surrounding header and sidebar do not change. In this example, the nodes are mail messages.

### 4.3. Hypertext Analysis

In addition to allowing for analysis of hypertext systems, as above, the model provides avenues for analysis of individual hypertexts as well. I will examine two. First the Facebook newsfeed, using the model to generally discuss its function and second afternoon:
applying the model mathematically.

The analysis of a hypertext using this model involves two steps: the first step involves identifying what in the system corresponds to nodes, documents, windows and the other objects. The actual analysis is usually carried out on the link function.

4.3.1. Facebook Newsfeed

Although I am specifically analyzing the Facebook newsfeed, many other social media sites have a very similar concept, which could be analyzed in a similar fashion.

The Facebook newsfeed consists of a vertical series of “stories” from the user’s friends as well as from advertisers. The stories might be a status update, photos, or a notification about event attendance. Each news story is a node. These nodes are hypertext documents themselves. Each node has an image of the friend’s avatar, along with their name, and the content of the story (some mixture of media, text and click-based links).

The link action we are concerned with is scrolling to the bottom of the newsfeed. Let $W$ be the window containing the user’s Facebook page, $w, h$ be the width and height respectively of the window’s region and $proj_{\Delta x, \Delta y}$ be the window’s projection function. Let $U$ be the user action triggered when $h - \Delta y < \epsilon$ for some value of $\epsilon$ chosen by Facebook. Then $L(U, C)$ is simply a series of $\ell_{insert}$ functions, one for each new node added.

However, there is much more consideration than that. Facebook does not add every one of your friends possibly stories. The link function decides, based on factors which are not publically known, which stories are most important, and displays those. Also, the stories the link function will insert are usually older than the stories which are already present. As such, the link function is not constant for a given view, but changes dynamically based on the context.
4.3.2. afternoon: a story

afternoon: a story by Michael Joyce [21] is a story within a story, ostensibly about a man who is worried he saw his ex-wife and son in a car accident on his way to work. The outer story consists of a discussion with a character representing the author about the inner story.

afternoon was written in the Storyspace hypertext system, and so must be analyzed within that context. Storyspace allows the user to create many nodes, and link between them arbitrarily. In reading afternoon, the user encounters the same node many times, or in some cases, what seems to be the same node many times, creating what Bernstein, Joyce and Levine call contours [4].

In afternoon, there is only one window, which the user reads the story in. The window has several buttons, each of which activates a link action. The link behaviour is the same for each button, a complete removal of the node in the window, followed by the insertion of a new node. What is interesting is what node is inserted in response to various user clicks.

There is a back button, which changes the view to whatever it was previously. There is also a forward button. Each node has a node which follows it in the linear sequence of the story, and clicking the next button (or almost anywhere in the window) will insert the next node in sequence. The default sequence yields a mostly linear story from only one character’s point of view. Reading the story in this way almost exactly mimics a traditional book, and serves as a gentle introduction for a new reader [46].

However, there are other ways of moving through the story. A user can click on buttons labelled ‘y’ and ‘n’. Most of the time, these buttons simply advance the story along the default path, but sometimes they answer a question asked by the text. There is also a menu option which allows the user to select which node they want to replace the current node with.

Most prominently though, are the clickable words within the story themselves. These
“words which yield” [21], as they are labelled by the text, replace the current node with
a node that is not along the default path but is connected in some way to the clickable
word. There is also a menu option for seeing all such words and their link source.

Let words \( : \mathcal{N} \times \mathcal{C} \rightarrow \mathcal{A} \) be a function which gives all clickable words for a particular node
as user actions and \( \text{links} : \mathcal{A} \rightarrow \mathcal{N} \) be a function which associates a node with a clickable
word. I also define \( \text{nodeaction} : \mathcal{N} \rightarrow \mathcal{A} \), which creates a user action corresponding to
selecting a particular node from the list. This generates the set \( \text{nodes} = \text{nodeaction}(\mathcal{N}) \).

If I define \( \text{current} \) as the currently visible node, then the set of user actions on node \( \mathcal{N} \)
is \( \mathcal{A}_{\text{current}} = \text{words}(\text{current}, \mathcal{C}) \cup \text{nodes} \cup \{ \text{select}_y, \text{select}_n, \text{select}_d \} \).

The link function then can be defined as follows (ignoring the \( \text{update} \) functions):

\[
L(C, A) = \ell_{\text{remove}}(\text{current}) \ell_{\text{insert}}(\text{nodefor}(C,A))
\]

That is, the function removes the current node from the window, and inserts the
appropriate node based on the user action. The function \( \text{nodefor} \) is defined piecewise as

\[
\text{nodefor}(C, A) = \begin{cases} 
N & \text{if } A = \text{nodeaction}(\mathcal{N}) \\
\text{links}(A) & \text{if } A = \text{words}(\text{current}, \mathcal{C}) \\
\text{next}(\text{current}) & \text{if } A = \text{select}_d \\
\text{yes}(\text{current}) & \text{if } A = \text{select}_y \\
\text{no}(\text{current}) & \text{if } A = \text{select}_n 
\end{cases}
\]

where \( \text{next} : \mathcal{N} \rightarrow \mathcal{N}, \text{yes} : \mathcal{N} \rightarrow \mathcal{N} \) and \( \text{no} : \mathcal{N} \rightarrow \mathcal{N} \) are author defined functions dictating
which node should be inserted on clicking the default, yes and no buttons respectively.

That is, the next node will be the node selected if the action is to select the node from
the list, the node associated with a word if that word is clicked, the default next node
if a default action is taken, the yes node if the yes button is pressed, and the no node if
the no action is pressed.

Note that because *words* is defined in terms of the context $C$, some links can be followed only if the reader has visited certain other nodes, and others are not available if a reader has visited certain other nodes.
5. Conclusion and New Applications

Throughout this report, we have seen many styles of link behaviours in different hypertext systems. However, with a model for behaviour, I can suggest some new ways of linking. For example, imagine a hypertext designed for teaching mathematics. As the student reads through a node on number theory, there is a side window with questions in it. Scrolling the content window is a link, which uses the context to insert questions on the node current visible into the question window. If the student had previously studied calculus, then questions which use integrals to solve problems could be included. If, on the other hand the student had previously studied abstract algebra, then questions that relate group theory and number theory could appear. This is an application of adaptive hypertext to a linking model.

Other possibilities include systems which will insert only parts of a node (using a selector) into the current document (similar to Augment[9]), or links which insert and remove content across several windows (in an extension of the ideas in Trellis).

Data linking, in particular the semantic web, is a very popular topic. However, semantic linking systems up to now consist only of showing how data are linked. My thesis will investigate applying the proposed model of link behaviour to linked data.

Other future extensions to the model include an expanded approach to multimedia, including timing and synchronization of media content, as well as a framework for discussing spatial hypertext. A more detailed algebraic analysis of the $\ell$-function analysis may also yield some interesting results.
This report has outlined a new model for analyzing link behaviour in a hypertext. This model distinguishes itself from other models in that it focusses on links as a response to user actions, rather than on modeling the structure of the hypertext system, although it has facilities for that as well. This model allows for a mathematical analysis of hypertext systems, and a precise description of the user’s experience when reading them. As an example, both the Facebook newsfeed and *afternoon: a story* were analyzed using the model.
A. Hypertext Definitions

I have collected here various hypertext scholar’s attempts at definitions of hypertext and related concepts, in no particular order, except that Nelson’s original definition comes first.

It is clear that different writers have very different ideas about what hypertext is. Nelson’s original definition defines hypertext in terms of what it cannot do, although he later updated it (below) to encompass more types of media than text and to emphasize user interaction. Other definitions emphasize user choice and interaction (Wardrip-Fruin, Dexter Model), or structural properties (Rosenberg, Stotts & Furuta) or its non-linear nature (Paraunak, Nielsen)

What’s clear is that hypertext is many things to many people. However, despite the differences, a common theme emerges, that hypertext is a non-linear, user controlled medium that allows for the representation of complex information structures. In the above work, I restrict this to be a document which changes its context based on user actions.

Hypertext

Nelson “Let me introduce the word ‘hypertext’ to mean a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper.” [33]

Wardrip-Fruin “Hypertext is a term coined by Ted Nelson for forms of hypermedia
(human-authored media that branch or perform on request) that operate textually. Examples include the link-based discrete hypertext (of which the Web is one example) and the level-of-detail-based stretchtext.” [47]

**Marshall & Shipman** “Hypertext, in its most general sense, allows content to appear in different contexts.” [30]

**Dexter Model** “. . . hypertext systems [provide their] users with the ability to create, manipulate, and/or examine a network of information containing nodes.” [16]

**Rosenberg** “A hypertext is a document in which interactive structure operations are intermingled with the text.” [38]

**Stotts & Furuta** “Traditionally a hypertext is composed of information fragments (text, graphics, sound, video, etc.) and tangible relationships among these fragments.” [40]

**Paraunak** “A system of nodes of information through which people can move non-linearly.” [44]

**Nielsen** “Hypertext is non-sequential writing: a directed graph, where each node contains some amount of text or other information.” [36]

**Yankelovich** “A hypertext system allows authors or groups of authors to link information together, create paths through a body of related material, annotate existing texts, and create notes that direct readers to either bibliographic data or the body of the referenced text” [49]

**Landow** “Hypertext, . . . denotes text composed of blocks of text . . . and the electronic links that join them.” [27]

**Keep, McLaughlin and Parmar** “Hypertext is the presentation of information as a linked network of nodes which readers are free to navigate in a non-linear
fashion.” [22]

**Fiderio** “Hypertext, at its most basic level, is a DBMS that lets you connect screens of information using associative links.” [19]

**W3C** “Hypertext is text which is not constrained to be linear.” [45]

**Hyper-***

**Furuta & Stotts** “A hyperprogram associates user-manipulatable information (the hypertext) with user-directed execution behavior (the process). Consequently a hyperprogram can be said to integrate task with information.” [13]

**Nelson** “Hyper media are branching or performing presentations which respond to user actions, systems of prearranged words and pictures (for example) which may be explored freely or queried in stylized ways.” [34]

**Parunak** “Hyper media is a set of nodes of information (the hyperbase) and a mechanism for moving among them.” [43]

**W3C** Hyper media is “MultiMedia Hypertext. HyperMedia and HyperText tend to be used loosely in place of each other. Media other than text typically include graphics, sound, and video” [45]

**Links**

**Gunder** “A pointer leading the user from one textual spot to another.” [15]

**Fiderio** “Links are used to connect the nodes.” [19]

**Marshall, Halasz et al.** “Relational objects.” [29]
Dexter Model  “Links are entities that represent relations between other [nodes].”[16]

W3C  “A relationship between two anchors, stored in the same or different database.”[45]

Anchors

Larsen  “That portion of the link that the reader uses to activate the link.”[28]

Gunder  “The exact spot in a link source or a link destination to which links are attached.”[15]

Joyce  “Words that yield.”[21]

Dexter Model  Anchoring is “a mechanism for addressing (referring to) locations or items within the content of an individual [node].”[16]

W3C  “An area within a the content of a node which is the source or destination of a link. The anchor may be the whole of the node content. Typically, clicking a mouse on an anchor area causes the link to be followed, leaving the anchor at the opposite end of the link displayed. Anchors tend to be highlighted in a special way (always, or when the mouse is over them), or represented by a special symbol. An anchor may, and often does, correspond to the whole node. (also sometimes known as ‘span’, ‘region’, ‘button’, or ‘extent’)”[45]

Nodes

Fiderio  “[N]odes . . . consist of a single concept or idea. In theory, nodes are both semantically and syntactically discrete.”[19]
**Dexter Model** “contain the chunks of text, graphics, images, animations, for example, that form the basic content in the hypertext network.” ¹ [16]

**Marhsall, Halasz et al.** “Data containing objects” [29]

¹Although the Dexter model avoids the use of the word ‘node’ for political reasons, the word they do use, ‘component,’ is introduced as a neutral synonym.
Bibliography


[26] George P. Landow. Relationally encoded links and the rhetoric of hypertext. In *Pro-


[44] H. Van Dyke Parunak. Don’t link me in: set based hypermedia for taxonomic


Glossary

**adaptive hypertext**

Hypertext which changes its content or structure in response to the user’s *context*.

**anchor**

a specification of part or all of a node. Used as one end of a link.

**bijection**

A function which maps every element in its domain to a unique element in its range, and in which every element in its co-domain has a corresponding element in the domain.

**context**

Information about the user’s navigation of the system, along with information retrieved from sensors.

**document**

An ordered tree of *media*. The value of each node in a tree can be empty or a chunk of media.

**dynamic node**

A *node* that changes based on the user’s *context* (c.f. *static node*).
environment

Information about the physical device used for accessing the hypertext.

function

Transforms objects into other objects.

function object

Act like objects, but their data is a function.

generator function

A partial function which returns an \( \ell \)-function.

group

An algebraic structure with a single closed, associative operation with inverses.

history

The view displayed at all past points in time.

hyper media

Multimedia hypertext which responds to user actions.

hyperprogram

associates user-manipulatable information (the hypertext) with user-directed execution behavior (the process)[13].

hypertext

A system which responds to user’s actions by changing their context.

\( \ell \)-function

A link behaviour function. That is, a function which modifies the view in some way.
link

An action that changes the structure of nodes visible to the user.

media

A chunk of content consisting of linear data and its type. Defined as a double $(m_{type}, m_{data})$.

monoid action

A set $S$ and monoid $M$ with operation $\ast$ together with an operation $\cdot : M \times S \rightarrow S$ such that for all $m, n \in M$ and $s \in S$, $m \cdot (n \cdot s) = (a \ast b) \ast s$.

definition node

A chunk of content, optionally created dynamically.

definition object

Elements of a hypertext system which contain data or other objects.

definition presentation

The physical representation of a document.

definition relationship

An association between objects or function objects.

definition rendering

The process by which a document is transformed into a viewable presentation.

definition selector

A function that returns a subtree of a document.
static node

A node that is always the same, regardless of the user's context (c.f. dynamic node).

text

A body of work.

view

A collection of windows.

window

A viewport showing either all of or a portion of the physical presentation of a document.