Chapter 4

HyperContext: A Framework for Adaptive and Adaptable Hypertext

4.1 Introduction

HyperContext is a new framework for adaptive and adaptable hypertext. HyperContext supports adaptive hypertext by providing users with individualised assistance during information seeking. It supports adaptable hypertext by allowing users to contribute towards the growth and organisation of information in the hyperspace. This chapter discusses the structure which supports adaptive and adaptable hypertext. Chapter 5 discusses the actual adaptive and adaptable features of HyperContext.

HyperContext describes an approach to achieve adaptivity in a heterogeneous domain-independent hypertextual information base. Its users will likewise be heterogeneous, and they are likely to have short-term as well as long-term interests (and will use the hypertext to find information related to both types of interest). They may also be considered as experts in some domains, and novice in others. Thus, there will be no significant difference between a typical HyperContext user and a typical WWW user.

An adaptive hypertext system supports individual users with different and changing requirements in their search for information. An adaptable hypertext system can allow users to contribute towards the structure of information and its organisation. To support adaptability the hypertext system must allow users to add information to the hypertext and to create associations, via hyperlinks, between any information in the hyperspace. Adaptivity is supported by understanding individual users' information requirements and presenting views of the hyperspace which best deliver to the user, or help the user find, the required information.

According to Brusilovsky [15] an adaptive hypertext system must (at least semi-automatically) adapt to the user. The two aspects of a hypertext which can adapt are nodes and links, the basic building blocks of a hypertext. Nodes can be adapted by modifying their content (*adaptive presentation*), whereas links can be adapted (*adaptive navigation*) by hiding them, annotating them, recommending them, or dynamically changing their destination (and thereby modifying the apparent structure of the hypertext).

In order to adapt to a user, something pertinent about that user must be known and represented. Depending on the domain of the adaptive hypertext system it may be sufficient to stereotype a user ([71], [72]), or it may be necessary to closely model a user's beliefs, goals, and plans ([55], [57]). In Information Retrieval (IR), the information typically known about the user is the query, or history of queries, submitted and, if the IR system supports relevance feedback, the relevance judgements the user has given to documents. In Chapters 3.3.1 and 3.4, we suggested that there may be a considerable difference between a user's short-term and long-term interests. In IR there is typically no distinction between the two. Generally, an IR system is not intended to have a long-term relationship with its users, and so all interactions with it are treated in the same way (however, adaptive IR systems, such as Adaptive HyperMan [70], do maintain a long-term relationship with the user). Although some non-adaptive hypertext systems (such as the WWW), and some adaptive hypertext systems (such as WebWatcher) have only a short-term relationship with the user, an adaptive hypertext system can model and update user interests over a number of separate encounters (user sessions). In hypertext, unlike IR, there is no requirement for the user to explicitly provide any information which may usefully represent an interest. The only information obtainable by observing user interactions are the nodes (pages or documents) which the user has accessed, and the links used to access those nodes. It is possible to observe, and perhaps draw conclusions from, user *behaviour* while browsing ([22], [75]). In any event, a user model which represents user interests and which differentiates between long- and short-term interests is necessary in order to adapt a general-purpose hypertext to the user.

The user model's representation is highly dependent on the representation of information in the hyperspace (information content and links). In IR, a user's query must be mapped to the document representations so that a matching algorithm can determine which documents are relevant. In Intelligent Tutoring Systems (ITSs), a user model is typically a stereotype or an overlay model. In the former case, documents may be defined in terms of attribute-value pairs corresponding to the stereotype. In the case of an overlay model, the user model is tightly bound to the definition and organisation of the underlying domain. In an adaptive general-purpose hypertext system, there should also be an appropriate mapping between the representation of documents and links and the user

model for the user model to influence the node representation and structure of the hypertext. A method of updating the user model is also necessary. A user model captures only a snap-shot of a user at a particular time. Just as the user's interests can change over time, so must the user model change accordingly. An ITS typically contains information which can be used to update the user model (so that if the user has successfully completed some task, the user model can be updated to reflect that the user is competent in the knowledge and application of concepts associated with the task). Through relevance feedback, an IR system can update the user query to modify under- or over-stated query terms originally used to describe a relevant document. An adaptive general-purpose hypertext system should define how and when to update the user model, to capture and reflect changes in users' long- and short-term interests.

In adaptive general-purpose hypertext systems, the user model is applied to the representation of documents and links to achieve adaptive presentation and adaptive navigation respectively. Adaptive presentation can be achieved by modifying the content of a document prior to its presentation to the user, or by dynamically creating the document on-the-fly. Adaptive navigation can be achieved by dynamically determining a link's destination; recommending links to additional documents not originally linked to by the document's author (by adding context-free links to "See also" references); adaptive ordering of author-provided context-free links; or by modifying a link's attributes (by hiding it; changing the anchor text's colour; or changing the annotated textual description of the document at the link destination). In order to dynamically determine a link's destination, to recommend links, or to adaptively order context-free links, it is necessary to compare the user model, which reflects a user's interests, with a representation of the documents in the hypertext.

4.2 HyperContext: A three-layer approach

A hypertext network is a collection of content-bearing nodes and links. For simplicity and clarity, the HyperContext framework is divided into three layers (figure 4.1).

The Structure Layer and Object Layer together represent the (multimedia) documents referred to from within the hypertext, and their relationships. The Presentation Layer contains the user model, prepares documents for presentation to the user, and provides an environment within which the user can interact with HyperContext.

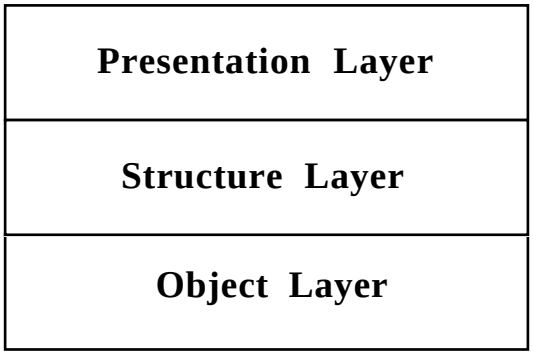

Figure 4.1: The three layers of the HyperContext framework

The HyperContext framework's layers are similar to the Dexter Hypertext Reference Model's layers [45]. HyperContext's Object Layer roughly corresponds to Dexter's Within-Component Layer, except that the Object Layer does not contain the multimedia documents themselves, but rather contains representations of them. The definition of the multimedia documents themselves is beyond the scope of HyperContext. The Object Layer representation of a multimedia document will, amongst other things, specify how to manipulate a document of a given type (such as an HTML document stored on an HTTP server) within HyperContext. The Structure Layer corresponds to Dexter's Storage Layer, which specifies a network of nodes and links. Finally, Dexter's Run-time Layer corresponds to HyperContext's Presentation Layer.

Although the model for the WWW does not directly correspond to either the Dexter Model or the HyperContext framework, it is useful to present an example of HyperContext, in terms of the WWW, briefly illustrating how and when interactions between the three HyperContext layers would occur. One of the biggest discrepancies between the Web on the one hand, and the Dexter Model and HyperContext framework on the other, is that in the Web link destinations are embedded within the source documents. In Dexter and HyperContext, link references are separated from the documents. In effect, the WWW has two layers - the Object Layer is defunct, as the Structure Layer would also contain the documents themselves. Nevertheless, with some imagination and at a risk of being pedantic, we will proceed with our example.

If the HyperContext framework is used to describe the WWW, the Object Layer contains *descriptions* of Web documents (rather than the documents themselves), details of the location of the documents, and specifications of the documents' type (such as HTML and GIF) and methods of retrieval (usually HTTP); the Structure Layer contains information about how the documents are linked to form a hypertext network; and the Presentation Layer processes the documents to resolve presentation requirements, such as how links appear to the user, prior to the presentation of the document. The user interacts with the

hypertext through this layer. Clicking on the visual reference for a link in a displayed document (to follow the link to another document) results in a request being passed to the Structure Layer to resolve the link destination. Once the link destination has been determined, the Structure Layer instructs the Object Layer to determine the location of the Web document identified as the link destination. The Object Layer retrieves the document according to the specified method in the document's description. The Object Layer then returns the retrieved Web document to the Structure Layer. The Structure Layer passes the Web document, along with any relevant structural information, back up to the Presentation Layer. Prior to displaying the document to the user, the Presentation Layer processes it according to the relevant structural, and user, information. This can include requesting the retrieval of other Web documents which should be contained within the document (in-line graphics, for example), processing HTML tags, and ensuring that the document is displayed in accordance with user preferences.

HyperContext also describes how new documents can be added to the hypertext, and how users can create links between arbitrary documents, even if the link author is not the same as the document author. A document can be added to the hypertext by registering its description and location with the Object Layer. Links are authored through the Structure Layer. As the document itself exists outside of the HyperContext environment, it can be modified only by the users who have access to it. However, any HyperContext user can change a document's description in the Object Layer, and create or modify structural information for it in the Structure Layer.

HyperContext is not a mathematical model for adaptive hypertext. It is a description, rather than a prescription, of a layered approach to adaptive general-purpose hypertext, but is not intended to be a formalisation against which it is possible to compare other models or implementations of adaptive hypertext. A prototype based on HyperContext has been implemented, and is described in Chapter 7. We have experimented with a number of elements of the framework and prototype. The experiments, and their results, are described in Chapter 9.

4.3 Interpreting information in context: the Structure and Object Layers

In a most flexible adaptive hypertext system, knowledge of a user is applied to knowledge of the information contained in an information-base to generate a relevant information-containing node. Rather than being limited to specific anchors in the node which are linked, a user is free to communicate any requirement to which the system will

dynamically attend. Indeed, the system may even anticipate user needs and will satisfy them in advance.

In the adaptive hypertext literature, the process of selectively presenting information to users is variously referred to as *personalising* [60], *adapting* [14], or *individualising* [31], the information. In HyperContext, we call this process *interpreting* the information. Information is not interpreted arbitrarily - something causes the interpretation to happen. We call the cause of the interpretation of information *context*. Context and its rôle in HyperContext are discussed in greater detail in Chapter 6. For the time being, we describe context as the influencing factor which causes information to be interpreted in one way rather than another.

When a node is interpreted the intention is to draw (from anywhere in the hyperspace) information relevant to the user closer to the node being interpreted. Part of the process of interpreting a node involves describing the node in a way that is relevant to the user. Another part of the process is changing the apparent location of the node in hyperspace so that it is closer to information that the user will find relevant, and further away from information the user would find irrelevant. This is similar to relevance feedback techniques in Information Retrieval systems which represent information using a vector space model [63]. In vector-based IR systems, documents in the information base are conceptually represented in *n*-dimensional space. A query is plotted into this *n*-dimensional space and the query's nearest neighbours are those documents which are most similar to the query. (A similarity measure, such as the cosine similarity measure [78], is used to compute this distance). Relevance feedback is a process whereby either the user's query is made more similar to the descriptions of documents in the vector space (conceptually moving the query closer to some documents and further away from others) ([74], [49], [78]), or else it can be used to change the descriptions of documents in the vector space to make them more, or less, similar to the query (conceptually moving documents towards, or away from, the query in the vector space) [10]. In hypertext systems, another document can be brought closer to a node by linking to it, and an already linked to document can be moved further away from a node by removing the link. This approach is, obviously, not as fine-grained a solution as that used in vector-based IR models.

If we take an extreme view, we can say that a node does not have a location in HyperContext hyperspace until it is interpreted. A less extreme view might be that a node occupies a potentially large but countable number of locations in the same hyperspace, but the node must be manifested in only one location each time it is accessed, which location is determined by the context in which it is interpreted. This is a small departure from the

vector-based model of IR, because in that model a document certainly cannot be imagined in more than one location and generally remains fixed in that location until the vector space is re-computed. In HyperContext, a node's location in hyperspace is described by the path taken to access the node, and the location of the node's children. A node is interpreted in HyperContext's Presentation Layer (Section 4.5), but a node's possible interpretations exist in the Structure Layer.

HyperContext is a descriptive *framework* for adaptive and adaptable hypertext. As a framework, it should not constrain specific implementations of HyperContext to use one particular design approach over another. For example, in a specific implementation document interpretations need to be represented using a data structure. HyperContext describes how document representations are used to achieve adaptability and adaptivity in hypertext, but it should distance itself from how the documents should be represented. On the other hand, different implementations of HyperContext should be inter-operable, so some minimum standard needs to be imposed. To lay the way for inter-operability without imposing design restrictions, the HyperContext framework distinguishes between internal and external services and structures. HyperContext imposes a standard for internal services and structures, and requires that properly defined interfaces are used to interact with, and convert data between, arbitrary external services and structures. Internal and external services and structures will be identified when appropriate, otherwise all references to services and structures are assumed to be internal. Implementors can choose arbitrary external representations, so long as they provide translators to transform the representations from the external formats to HyperContext's internal formats.

Internally, an interpretation is composed of a description, or representation, of the node in context, and a set of out-links. An out-link has an anchor in the source node and a destination. An interpretation is composed of a vector of weighted terms which describe the node in this context. The term weight indicates the term's relative importance to the interpretation. The same term in different interpretations can have different weights, reflecting greater or lesser importance of the term in each interpretation. Indeed, terms may be zero-weighted in some interpretations, while not in others. The non-zero weighted terms which describe an interpreted node are called *labels*. If the interpreted node has out-links then the anchor for the source of each link will be one of the interpretation's labels. A label with the same name cannot occur more than once, and cannot be the source of more than one link in a given interpretation. As terms are labels only if they are non-zero weighted, zero-weighted terms cannot be used as the source of a link. Labels and links are stored separately from the node's actual content. They can be

accessed, modified, searched and otherwise processed without the need to retrieve the multimedia document which the interpretation represents.

Two functions are available to retrieve interpretations and links for a node from the Structure Layer. These are getInterpretation(N, C), and getLinks(N, C). The function getInterpretation(N, C) retrieves the label weights of node N in the context C, and so describes node N in the context C. Similarly, getLinks(N, C) retrieves the links of node N in the context C. A link is a $node_1$-label-$node_2$ triple, which represents the label in node $node_1$ acting as the link source, and the node $node_2$ which is the destination of the link.

A context provides an environment within which a node can be interpreted. The context must have some kind of discriminatory power which will allow the node to be interpreted correctly. At its most expressive, a context could be a precise representation of a user's state, essentially enabling a machine to interpret information just as that particular user would. However, it is not yet likely that a user's state can be accurately modelled to this desirable degree of precision in heterogeneous hypertexts. Nonetheless, the user is an important aspect of an adaptive system and must, sooner or later, be taken into account. In the Structure Layer, however, the user is not represented - at least, not directly. We require something more easily computable and consistent to provide context. Following a link in HyperContext causes the node at the link's destination to be interpreted. It seems sensible, then, that *context* is closely associated with the process of *information access*. The most immediate information available is the identity of source node from which access is made, and the label in the source node which acts as the link anchor in the source node.

A context, then, is a node-label pair. Interestingly, a context only has an effect when a link is followed. We know, from the earlier discussion of interpretations, that labels are dependent on interpretations. In one interpretation of a node, a label may have a non-zero weight, whereas in another interpretation of the same node, the same label may have a zero-valued weight. We also know that labels become the source of links only once a node has been interpreted, so that in one interpretation a label may be the source of a link which leads to child N; in another interpretation the same label may be the source of a link which leads to child N+2; in other interpretations it may be free; and in yet others the label may have a zero-valued weight, and so cannot be used as a link source. Therefore, a context can have an effect only when it is actually applied by following a link from a node on a label to a child. The context determines how the child will be interpreted.

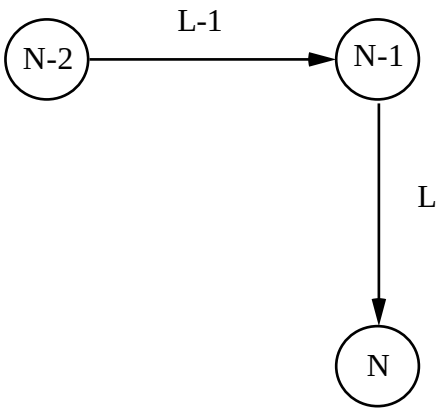

Figure 4.2: The interpretation of node N-1

In figure 4.2, three interpretations (N-2, N-1, and N) are shown. Some context (which is unspecified in the diagram) has given rise to the interpretation N-2. In that interpretation label L-1 is linked to N-1. In the context of (N-2, L-1), interpretation N-1 is linked to N on label L. In their respective interpretations, labels L and L-1 must have non-zero weights (because they are used as link sources). getInterpretation(N-1, (N-2, L-1)) would retrieve label L as its result, and getLinks(N-1, (N-2, L-1)) would retrieve (N-1, L, N).

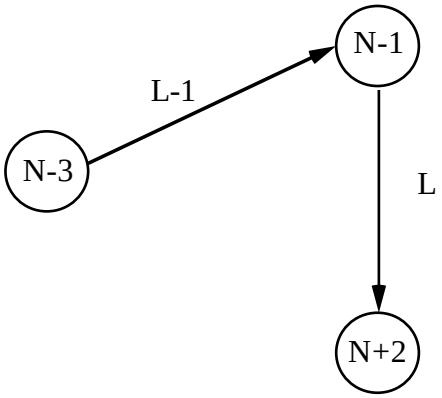

Figure 4.3: A different interpretation of node N-1

Consider the scenario when node N-1 also has an interpretation in the context (N-3, L-1), as depicted in figure 4.3. Label L is once again non-zero weighted in getInterpretation(N-1, (N-3, L-1)), but this time getLinks(N-1, (N-3, L-1)) retrieves (N-1, L, N+2) as its result. Conceptually, the label L in all interpretations of N-1 behaves as a multi-headed link, where the context of N-1 determines the actual destination of the link.

In theory, an arbitrary context can be applied to an arbitrary node to obtain an interpretation and a set of links for that node. In practice, as is described in Chapter 5.2, a user assists in the creation of an interpretation, the selection of labels in the interpretation

to act as link sources, and the selection of link destinations[1]. Interpretations are stored in the Structure Layer, and are available to future HyperContext users. If an interpretation for some context applied to some node already exists in the Structure Layer, then it is retrieved, otherwise one of two things can happen: either the user can create an interpretation for the node in that context, or else the function can retrieve a context-free *universal interpretation* for the node.

The getInterpretation and getLinks functions retrieve a node's context-free universal interpretation whenever the node has not previously been specifically interpreted in a given context or when the node's context cannot be determined. A node's context cannot be determined if a user accesses a node directly, without specifying a context in which to interpret the node. This is similar to accessing a Web document through a Web browser by specifying the document's URL, rather than by following a link to the document. In HyperContext, the process of accessing a node requires it to be interpreted, and as interpretations are applied in context, non-specification of a context implies that it will be accessed in the universal context, called bottom. HyperContext is a closed world in which the context bottom always applies unless it is overridden by a specific context. In figures 4.2 and 4.3 above, nodes N-2 and N-3 respectively were interpreted in an unspecified context. We now know that the nodes would have been interpreted in the context bottom (figure 4.4).

The Structure Layer is composed of the context bottom, an arbitrary number of interpretations of nodes, and an arbitrary number of links. Links always connect a parent interpretation to a child interpretation (both of which may be different interpretations of the same node) on a specific label. The contexts of a node may be retrieved through the contexts(N) function, as a list of node-label pairs and bottom. If a context C is not in contexts(N), and the Structure Layer receives a request to access node N in the context C, then the node N will be interpreted in the context bottom, unless the user creates a new interpretation for N in the context C (Chapter 5.2).

It is assumed that all the information required to interpret a node is available in the context immediately containing that node. This means that in the structure depicted in figure 4.4, node N is interpreted in the context of (N-1, L) only, and not in the context of (N-1, L), where N-1 is interpreted in the context (N-2, L-1), where N-2 is interpreted in the context bottom.

---

[1] We discuss the issue of automatically interpreting documents in arbitrary contexts versus user-created interpretations in Chapter 6.3.

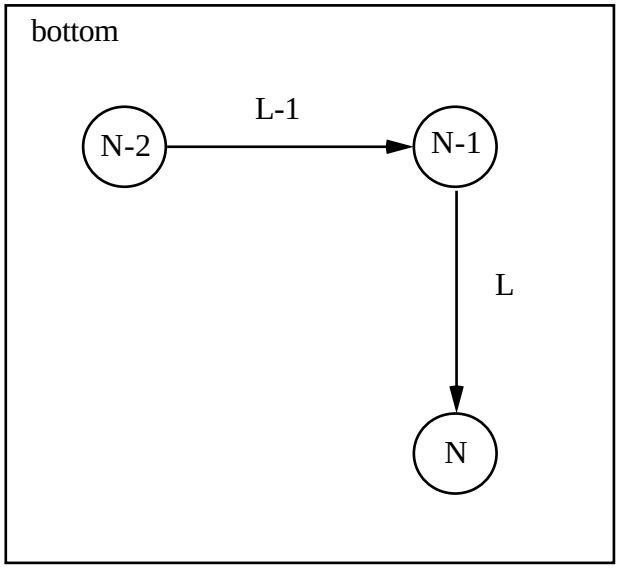

Figure 4.4: Node N-2 interpreted in the context bottom

The reasons for this assumption are entirely pragmatic. If we consider, for a moment, that the possible interpretations of a node are dependent on the regressive interpretations of its ancestors (on a specific path of traversal) then in order for a node to be successfully interpreted we must have the influencing factors of its ancestors explicitly available at the moment of interpretation. This would add a significant overhead to the process of interpretation. More importantly, given that HyperContext users assist in the creation of interpretations, the creation of an interpretation would require the user to be explicitly aware of the interpretations of the node's ancestors on a particular path. This would add a considerable cognitive overhead. We reduce the complexity of the problem by making the assumption that all the information required to interpret a node is available in that node's immediate context, and that a user can create an interpretation of a node irrespective of the interpretations of that node's ancestors. In Chapter 5.8 we begin a discussion about the process of determining a user's interests in order to automatically identify and locate relevant information. We will show how interpretations of nodes on an access path can help identify terms that describe a user's interest, and, significantly, that these terms should be used to identify relevant information which the user has not already visited on the current path of traversal.

A side-effect of the assumption that all the information needed to interpret a node is available in the context immediately containing that node is that it is possible for many interpretations of the same parent to give rise to the same context within which to interpret the same destination document. For example, consider that for all contexts C_i to C_n in contexts(N) the intersection of getLinks(N, C_j) and getLinks(N, C_k) (where $i \leq j$, $k \leq n$ and $j \neq k$) is the link (N, L, N+5). This signifies that in both interpretations of N in the contexts C_j and C_k, label L is linked to node N+5. If label L is traversed from either

context in order to access node N+5 using getInterpretation(N+5, (N, L)), then the same interpretation of N+5 will be retrieved. The is true regardless of in how many different interpretations of N label L is linked to node N+5. This discussion is continued in Chapter 6.4.

The Object Layer is the interface between the Structure Layer and multimedia documents residing outside HyperContext. Each object in the Object Layer represents a single multimedia document and contains details of the document's location, its type, and which protocol to use to access the document. In the Structure Layer, the getLinks function retrieves $node_1$-label-$node_2$ triples, denoting the label in the interpreted $node_1$ which is the source of a link to the destination $node_2$. The Object Layer is responsible for binding the label to the actual region in the multimedia document which will be displayed as the link anchor. An object in the Object Layer is called a *profile*, and is retrieved through the getProfile(N) function.

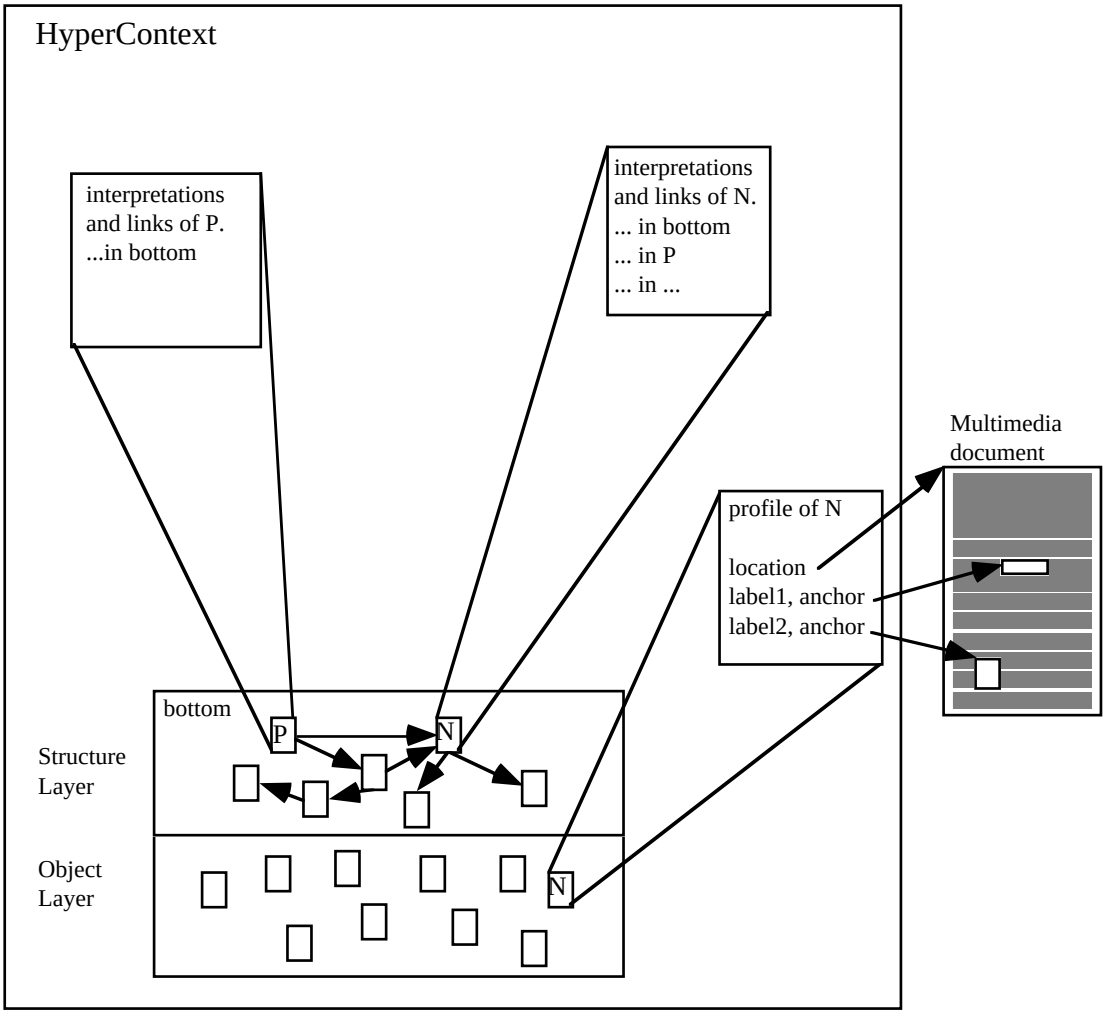

Figure 4.5: The Structure and Object Layers and their interaction with the outside world

The relationships between the Structure Layer and Object Layer, and the Object Layer's interaction with multimedia documents in the universe outside HyperContext is shown in figure 4.5.

HyperContext is explicitly distributed. The Object and Structure Layers can be distributed across many HyperContext servers. Normally, however, a multimedia document and its profile and interpretations will reside on the same server, although hosting each on different servers is also supported.

4.4 A description of HyperContext in terms of the city metaphor

HyperContext's Structure and Object Layers are the building blocks of an adaptive hypertext system. In previous chapters, we used the metaphor of a city's streets and corners to describe links and nodes in a normal (non-adaptive) hypertext. Although this metaphor is suitable for describing (relatively) static hypertexts, and the interactions of town-dwellers and users of static hypertexts within their respective environments, HyperContext requires a more fluid analogy before we can return to this metaphor.

Imagine that we are immersed in a virtual building. As we interact with the building, we have the impression that it is solid and pre-built. In reality, however, the environment is created for our individual pleasure as we interact with it. It is similar to modern computer-based adventure games: the map shows us only where we have been - the rest remains to be discovered, and as such, it could take any form. However, whereas in an adventure game the unknown areas are there to test our skills to the limits, in our example the building is subservient and always tries to aid our progress rather than impede it.

Imagine that within a virtual room (corresponding to an interpretation in HyperContext), virtual doors (links) indicate exit points from the room, and walking through a door will place us in another room. Imagine that the doors do not have fixed locations in space, but they are located wherever seems best for us. We know that if a door exists, it will lead us to another room that we want to visit. In the underlying model of the virtual building, rooms are described without doors, and consequently, without any detail about the connections between the rooms. Rooms are doorless until we enter them, and when we enter a room, doors simply appear in the right locations, leading to other rooms. The presence of doors, and the rooms they lead to, are conditioned by our requirements, so it is not even necessarily the case that the door through which we enter a room will be there once we are in the room. Two different people entering the same room may see doors in different locations, and even if a door appears to be in the same location, it may lead each

person to a different room. Doors allow people to move from one room to another, and all rooms are potentially accessible from any other room. Doors are like worm-holes in space - they may appear anywhere, and may lead anywhere. Unlike worm-holes, however, a door will always lead to somewhere the traveller wants to go.

Is there a need for a room to have more than one door or exit point? If we can ensure that all doors from a room would lead to other suitable rooms for the traveller, then perhaps all we would ever need to provide is one door which leads to the room which the traveller would have selected anyway. After all, a traveller cannot simultaneously exit a room through two or more different doors. As long as the traveller has the choice to not walk through a door, even if it is the only one available, then we cannot guarantee that the traveller will walk through it. She may decide to go back to a previously visited room, leave the building through the fire escape, or even close her eyes and conjure herself up in another room. Do we even need doors to move from room to room? Why can the traveller not move freely through walls, floors, and ceilings, always ending up in the best room to be in? Let's assume, for the time being, that a user will press the index finger of her left hand against the part of the floor, ceiling, or wall through which she wishes to pass, and she will be transported to, and placed in the centre of, the most appropriate room.

How does the building know when and where to take the traveller? There needs to be some kind of expression from the traveller which the building can use to determine which room is the most appropriate to take her next and when she wants to go there. The traveller makes her wish to go to another room explicit by touching the wall with the index finger of her left hand. She needs to externalise the requirement, using some form of expression of intention which the building can recognise. Likewise, there needs to be some way of binding the expression of intention to visit another room with the room which is best to visit next. This expression of intention can take any form, as long as the traveller and the virtual building agree upon it. In hypertext systems, clicking on a link not only means that the user wishes to access another document, but it also means that the user wishes to be transported to whatever document the link leads to. Consequently, the building and the traveller might agree that she will collect objects from each room she visits and that when she touches the wall to visit another room, the building will select the room which is most relevant to the combination of objects she has in her possession.

The traveller carries a bag of objects and whenever she touches a wall the building will select a room based on the contents of the bag. How can the building, and the traveller, know that the room selected is, in fact, the most appropriate room available? If previous travellers can give feedback and somehow influence the choice made, then this may make the choice made more acceptable to the traveller. For example, based on previous

travellers' experiences, if the traveller's current bag contains certain objects then the traveller will be taken to a particular room ([50], [61], [2]). Otherwise, as is the case with HyperContext, previous travellers can create doors which lead to particular rooms, based not directly on the collection of objects in their bag, but on where the traveller was previously and which object in the current room the traveller reaches out for. It is important to note that in HyperContext we are not discarding the bag of objects that the traveller has collected. The contents of the bag will have an important rôle to play. However, unlike WebWatcher, Mathé and many others, the bag of objects is not the most important aspect of the adaptive hypertext system - the *structure*, or rather, the possible structures, of the hypertext are just as important. If previous travellers have influenced the structure of the building, then it makes sense to show travellers doors, rather than merely letting the traveller press her index finger against a wall, because the presence of a door will inform her that previous travellers, who entered the room in the same way that she just did, found the doors and their destinations useful.

If we return to the analogy of city streets and street corners, then although street corners (documents) exist in the underlying model of the city, visitors see and interact with virtual representations of them. Streets are created by previous visitors who see a need to connect corners. Rather than becoming a fixed part of the representation, a street and its destination are dependent on the visitor's location. Whenever a visitor approaches a street corner, the dynamic personalised city planner, using information from previous visitors and the knowledge of how the current visitor reached the corner, will determine which virtual streets are appropriate to build. Two visitors at the same street corner, will, if they arrived at the corner in different ways, potentially see different streets leading away from the corner, and even if they see the same street, following it may lead each visitor to a different corner.

The city's street corners are represented in HyperContext's Object Layer. The interpretations of street corners and the streets themselves exist in the Structure Layer. As a visitor walks through the city, the Presentation Layer interprets the corner that the visitor is at, shows the visitor the streets that can be followed from that corner, and determines the destination of each street.

4.5 Interacting with HyperContext: the Presentation Layer

The Presentation Layer acts as the interface between the user and HyperContext. The user makes requests for information, which the Presentation Layer passes to the Structure Layer, and prepares the information for presentation to the user. The Presentation Layer is responsible for collecting and manipulating information about the user. The short- and

long-term user models obtain their information from the interactions the user has with the Presentation Layer. The Structure Layer supports browsing between interpretations of information. The society of HyperContext users determines how information can be interpreted and organised, creating the contexts in which the information exists. Users browse through the hyperspace which has been constructed by previous HyperContext users.

As we will see in Chapter 5, the Presentation Layer takes advantage of the adaptable nature of the Structure Layer to provide users with adaptive navigation support. The Presentation Layer can use a user's long- and short-term interests to recommend links and paths to information in context; extrapolate a user's short-term interest based on paths of traversal; and support the user in the creation of new interpretations of information. In this section, we will discuss the properties of the Presentation Layer which enable this level and type of support.

The main data structure manipulated by the Presentation Layer is the *context session*. The context session records a path or chain of interpretations the user has visited. This information is used to update the short-term user model, which can provide feedback to the user's long-term user model. The short-term user model is used to establish the user's current interests, so that relevant nodes, paths, and links to relevant information can be suggested or recommended to the user.

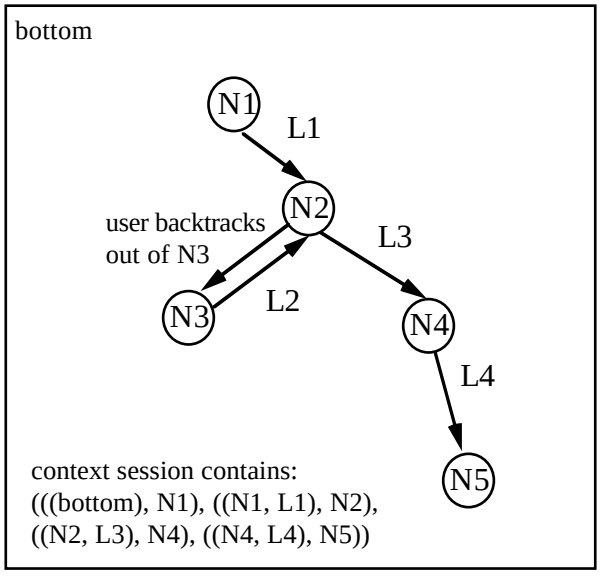

Figure 4.6: A context session without the user's visit to N3 in the context (N2, L2)

The context session is a list of context-node pairs, which records a chain of nodes the user has accessed, and in which context each node was accessed. The context session is not necessarily an exact record of a user's activity during the context session. If a user

backtracks out of an interpretation, the context-node details for that visit are deleted from the context session (figure 4.6).

The context session is not necessarily a precise account of a user's activities as it is possible for nodes the user has accessed to be ignored. A primary reason for this is that we assume that there is always a path from the node from which the user began searching to one which contains the information sought, so if a user accesses a node which is not on this path, then we will ignore it. Although this is obviously an enormous assumption, and one which is probably incorrect some of the time, we will stand by it. Only minor modifications to the framework are required to represent the context session as a graph, instead of a linear path, but it is probably harder to detect when the information seen in a backed out of node actually contributes information which is *necessary* to accurately determine the user's interests than to simply ignore it. The user is ultimately always in control of HyperContext, and is able to direct HyperContext to use or ignore an interpretation which would otherwise have been incorrectly discarded or included. With reference to the context session depicted in figure 4.6, the user could have instructed HyperContext that N3 should be included in the context session (even though it was backtracked from) and that N4 should be ignored (even though the user subsequently followed a link from it).

The main purpose of a context session is to capture a user's directed search for information. If the nature of a path is that as the path grows, children provide greater and greater detail about information contained in the parents, then a user's interests will become clearer and clearer to an observer, even if that interest is not communicated directly from the user to the observer. The nature of such a hierarchical organisation of information is, however, that as long as the user has a choice of paths from a given node, then there is an equal chance that the user will choose any of the nodes subsequently accessible from that node. In general hypertexts, there is no guarantee that children will provide more detail about their parents - indeed, a child could provide an overview of material discussed in detail in the parent, or even be a fairly arbitrary association, apparently taking the user further away from specific information. However, in HyperContext, users create interpretations by describing what information in the node is *relevant*. It is not necessarily the case that the description provided for a node in the context of one of its parents is directly relevant to one of the node's grandparents. However, by comparing interpretations of nodes that the user has accessed in the current context session, it may be possible to extrapolate general information about what the user is interested in, and it is this information which constitutes the user's short-term interests, along with any stated declarations the user may have made (Chapter 5.8).

We have previously observed that unless we can establish when a user's short-term interest has changed from one topic to another, especially if the change is gradual, there is a danger that automatic or automated attempts to capture the user's interests are likely to significantly misrepresent the user (Chapter 3.4). Consequently, a context session terminates either at the end of a user session, or else when a *context switch* is detected. A context switch can occur in any of three instances - the first instance is when a user accesses a node in the context bottom, the second is when the user, while visiting a node in a particular context, asks for that same node to be interpreted in a different context, and the final instance is when the user hyperleaps (jumps) to an arbitrary node in an arbitrary context instead of following a link to it. A single browsing session can contain several context sessions, each context switch representing the end of the user's short-term interest in the associated topic.

So long as a context switch has not occurred, each time the user accesses a node the context session and the user model reflecting the user's short-term interests are modified. The user can choose to influence HyperContext by explicitly requesting that specific nodes are to be used or ignored in the context session, and the user can also explicitly pre-declare an interest as well as directly modify the user model reflecting the interest.

The long-term user model can be used to instantiate a user's short-term user model (rather than have the short-term user model starting each context session with no information in it). The short-term user model can be used to update the user's long-term user model. Both of these interactions between the short- and long-term user models are outside the current scope of the HyperContext framework.

The interactions between the context session, short-term user model and the Structure Layer are described in more detail in Chapter 5.

4.6 Summary

We have described the HyperContext framework in terms of the structures required to support adaptive and adaptable hypertext using multiple interpretations of information. The framework is composed of three separate but interacting layers which bind HyperContext document representations to real documents stored outside HyperContext (the Object Layer); represent interpretations of documents and the links between them (the Structure Layer); and provide the interface between the user and HyperContext (the Presentation Layer).

The important constructs the framework provides are profile, interpretation, label, link, context, bottom, and the context session. A document's profile is its representation in the Object Layer. A document's interpretation is a description of the document in a context, and is represented in the Structure Layer. The description is composed of a vector of label weights, and links. A link is represented by the triple $node_1$-label-$node_2$, and a context is a node-label pair. A document can be accessed in the special context bottom to obtain a context-free universal interpretation of it. The context session is maintained by the Presentation Layer to record a user's directed path of traversal through hyperspace and is used to determine a user's short-term interests. A context session ends whenever a context switch occurs.

The layers communicate through function calls. The Presentation Layer requests a document interpretation through getInterpretation(node, context). The document's links associated with the interpretation are retrieved through getLinks(node, context). All of the contexts for which a document has an interpretation are obtained using the contexts(node) function. The Structure Layer obtains details of the document's location in the universe outside HyperContext, and link-anchor text binding information by submitting a getProfile(node) request to the Object Layer.