The Carnforth Model of Accessible Adaptive Hypermedia

Robert Dodd
University of Teesside
School of Computing
Tees Valley TS1 3BA, UK
+44 (0)1642 342656
r.dodd@tees.ac.uk

Dr Steve Green
University of Teesside
School of Computing
Tees Valley TS1 3BA, UK
+44 (0)1642 342656
s.j.green@tees.ac.uk

Dr. Elaine Pearson
University of Teesside
School of Computing
Tees Valley TS1 3BA, UK
+44 (0)1642 342656
e.pearson@tees.ac.uk

ABSTRACT

With the growth of script-intensive web pages, particularly those using AJAX technology, the adaptation of Web content to match the needs and capabilities of individual users has become increasingly problematic. New versions of well-known websites, including for example Google Select, which is an AJAX driven variant of their standard search page, are now largely opaque to screen reading technology such as Jaws. Taken together with the trend to surf the Web on small hand-held devices, which causes its own accessibility problems, a new approach to expressing heavily scripted content is needed. This research returns to first principals, and considers the underlying Dexter Model of Hypertext, and how that may be placed within a broader model of document content that is amenable to adaptation of content to user needs either through configuration, or through dynamic self-adaptation. The model proposed consider a document in terms of five individual abstractions: content, inventory, semantics, navigation, and adaptation. A simple (fully working) example, taken from a small fragment of Google Maps, is presented to demonstrate how such a model may operate in practice, adapting between two different user profiles on demand.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: user interfaces.
K.4.2 [**Social Issues**]: Assistive Technologies for persons with disabilities.

General Terms

Measurement, Design, Human Factors.

Keywords

Abstract User Interface, Hypertext, Hypermedia, Dexter Model.

1. INTRODUCTION

The World Wide Web as it exists today is a collection of electronic documents that reference each other using a particular model of navigation that is a subset of the Dexter Model of Hypertext [1]. The layout of content within a document, and the content itself, may change, but those changes do not affect the links between the documents. Documents in this sense are Web pages, not websites, and this is where the problems for assistive technology begin; Web pages are pre-defined “chunks” of the whole website, with the chunks selected for some unexplained (at least to the web browser) logical grouping, screen size, or user cognitive load. If the content needs to adapt to a user’s physical or cognitive capabilities, and perhaps migrate from the visual to the sonic design space, it is very difficult to ”|de-chunk” back to the original semantic model of content, in order to re-group and re-organize the content; screen reading applications such as Jaws [2] do not even attempt to do so.

Some semantic information within a Web page is available to assistive technology, and with the latest HTML 5 draft [3], the amount of explicit semantic content is growing. Semantic information is available through a very limited and stylized set of document tags that describe a document in terms of heading, paragraphs, images, and tabular content. A very limited number of interaction modalities are also supported, from which semantic inferences may be drawn. In practice however, is semantic tags are misused to describe document layout, and scripting, particularly AJAX [4] scripting, is used to create new interaction modalities that are largely opaque to assistive technology. Keyword completion in Google Select [5], where a drop-down window appears and populates with possible completions for the search string being entered, is an example of these new modalities; this research has shown screen reader applications to become either mute to the offered options, or not notice subsequent updates to the presented options after their first display.

The first step taken in approaching these problems was to produce a general model of a document as a whole, for example a website, in terms of its content, semantics, and navigation, and then to consider what it means to adapt such content.

2. THE DEXTER MODEL

The Dexter Model of Hypertext [1] is a synthesis of the capabilities of existing hypertext models in 1988; no single hypertext system supported the entire feature set. The Dexter Model was conceived as a layered model of content navigation with a runtime layer that depended upon a storage layer, which delegated document structure largely to individual components in a within-component layer. As part of this research, a simplified, semantically based description of the model was created and is shown in Figure 1.

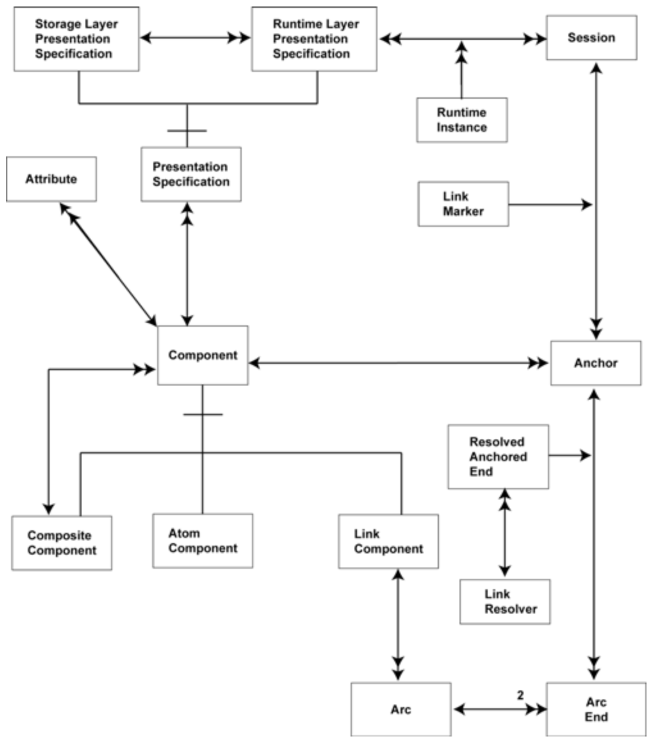


Figure 1 – Dexter Model

Figure 1 shows how Components are related to each other, with a Component being a document, or part of a document. A Component may decompose into many s amaller components, or it may be a Link Component between two or more Components (i.e. a hyperlink). For any one Component, there may be many Presentation Specifications, which provide hints to the Runtime Layer as to how to express the component. The Dexter Model does not describe the content or format of such hints, but rather identifies the capability. Components are further characterized by Attributes that describe semantics notions about the Component that are of use to the Runtime Layer. Again the format of the attributes, and potential values is left to specific implementations. Link Components within the Dexter Model are not required to be constant; they are required only to be known when navigated. This allows for adaptive environments where navigation is rule-based, for example where a component is a search in Google; that there may be search results is certain, but the actual value depends upon the search terms given. This feature is not directly supported in HTML except by use of web page scripting.

3. AMSTERDAM MODEL

The Amsterdam Model of Hypermedia [6] is an extension of the Dexter Model of Hypertext that brings the concept of continuous time to hypertext, and is shown in Figure 2. In Dexter, presentation of content is essentially static, with changes in presented Components occurring when a user navigates a link, resulting in an event-driven, discrete model of time. With the Amsterdam Model, video and audio streams are introduced as navigable, and synchronized content, so that for example, certain hyperlinks may be valid for particular scenes of a movie, or an audio track may begin playing after a specific time during a movie. The Amsterdam Model also attempts to handle competition for browser and/or scarce device resources, allowing synchronization and queuing for audio channels, video play-out, and potentially screen real estate. This model of time and synchronization underpins the SMIL language [7] for synchronized multimedia on the web. SMIL itself is a component of the Multimedia Messaging System format (MMS) [8] used to send multimedia messages between mobile phones. SMIL, and the Amsterdam Model are to mobile phones what HTML and the Dexter Model are to web browsing, so any accessibility issues that impact the use of the Dexter Model also impact upon web browsing and the Amsterdam Model, and any accessibility issues with the Amsterdam Model impact upon both multimedia web browsing and the Multimedia Messaging System used in mobile phones.

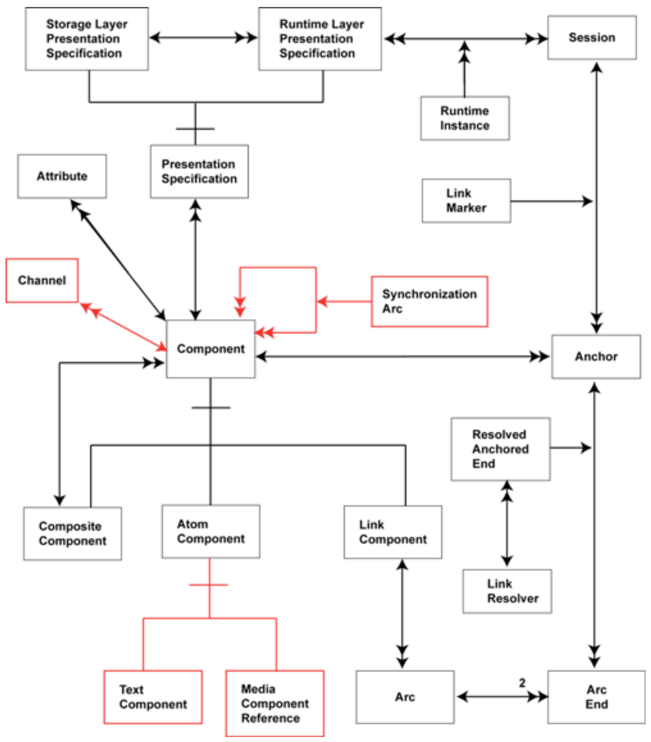


Figure 2 – Amsterdam Model

Figure 2 shows how media synchronization is added to the Dexter Model to create the Amsterdam Model; additions to the Dexter Model are highlighted in red. The Atom Component, plain text content in Dexter, is further classified into Text Component and Media Component Reference in order to allow the model to refer to images, video, and audio held elsewhere (the Dexter Model is assumed to contain all the content that it presents). Synchronization between components is achieved through the use of Synchronization Arcs that describe timing relationships between components. Components are explicitly grouped into Channels for presentation. (Note that Figure 2 is a somewhat simplified, and a more comprehensive description is given in [6]).

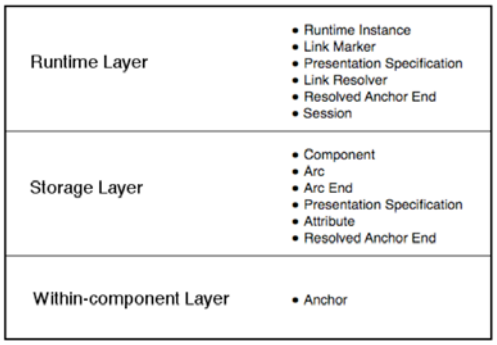
Accessibility issues exist around the semantic meaning of synchronized content in the Amsterdam Model; if a web browser does not know whether an audio stream is background music or spoken text, it becomes difficult to adapt multimedia content between design spaces, for example when adding additional text-to-speech. It is also important to understand why content is being synchronized. Taking the simple example of alert notification on a mobile phone, it is important to know that there is an incoming call immediately, but notification of an incoming text message can be briefly delayed to allow, say, play-out of a text-to-speech message to complete; This requires an understanding of event priorities, requiring in turn semantic knowledge of the content.

The CISNA Document Model, created as part of this research, breaks the semantic view of the Amsterdam Model into five separate concerns: (i) raw content, (ii) core navigation between components, (iii) resolution of links between components, (iv) presentation of components, and (v) the semantic meaning of the content referred to as "within-component".

4. FIVE LAYER DOCUMENT MODEL

The original Dexter three-layer model, and the new CISNA (Content Inventory Semantics Adaptation) five-layer document model are shown for comparison in Figure 3 and Figure 4.

At the centre of the Dexter Model is the Storage Layer, which describes navigation between documents and between Components within a document. Description of semantic meaning is delegated to the Within-component Layer, which exposes Anchors (potential sources and destinations for navigating the documents) to the Storage Layer. The Storage Layer presents Components and potential navigation paths to the Runtime Layer, which is responsible for interacting with the user, and for resolving any "dangling links", such as, say, Google's search results which are not known until a search has been executed.



Runtime Layer	• Runtime Instance • Link Marker • Presentation Specification • Link Resolver • Resolved Anchor End • Session
Storage Layer	• Component • Arc • Arc End • Presentation Specification • Attribute • Resolved Anchor End
Within-component Layer	• Anchor

Figure 3 – Three-Layer Dexter Model

The five-layer CISNA model peels most of the Dexter Runtime Layer away to form separate Development and Runtime Systems that have visibility of the full five layers (Development/Runtime is distinction that exists within the detail of the Dexter Model itself). What remains of the Runtime Layer is link resolution, which forms part of the Adaptation Layer. Responsibility for rendering content to the user is explicitly moved out of the layers into the Runtime System, leaving an Abstract User Interface

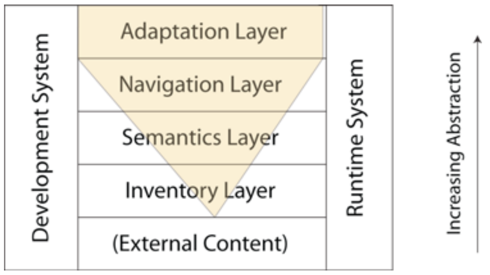


Figure 4 –Five-layer CISNA Model

describing content and interaction. In doing so, it follows the trend within web-content standards towards full separation of content from rendering, for example XForms [9]. What remains of the Dexter (and Amsterdam) Model is then considered within five independent abstractions, each relating to the immediately adjacent layers in the diagram. Note that the Adaptation Layer is adjacent to all other layers. In terms of the level of abstraction from the physical content of text and multimedia, abstraction builds from bottom-to-top within the diagram.

The lower four layers of the CISNA model are described below, but discussion of the Adaptation Layer is postponed until later.

4.1 Navigation Layer

The Navigation Layer of CISNA and the Storage Layer of the Dexter/Amsterdam models are closely related.

The Navigation Layer, shown in Figure 5, is primarily a collection of Nodes. A Node is something a user can navigate to or from, a *Component* in Dexter-speak. Nodes may also be collected form sub-groups of a Node (but not including itself), and this corresponds to the *Composite Component* of Dexter. Sub-groups exist primarily to support grouping of video, audio and captioning. It is not intended to describe "pages" (of documents or of websites), which are considered rather to be Views of content.

A View is a collection of navigable Nodes that are presented contemporaneously to the user. Presented content may not be visible/navigable to the user at all times, for example when content is presented visually in a scrollpane. Views are how the CISNA model supports the Dexter 1:M *Link Components* (represented by multiple *Arcs* for each *Link Component*). A View may also represent something else: a "chunk" of document, a web page, a chapter of a book, a volume of an encyclopedia; in fact any level of document decomposition relevant to presentation of content to the user. Views are not inherently hierarchical, although they can be, and the same Nodes may appear multiple times in different Views. The Dexter Model has just one View at this level of abstraction: the underlying content.

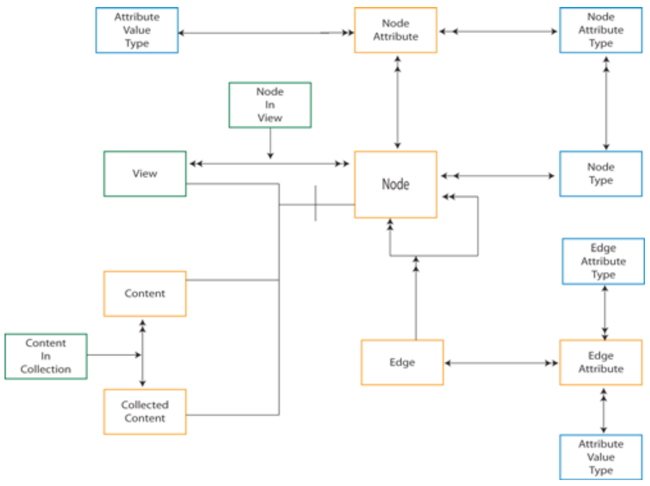


Figure 5 - Navigation Layer

Navigation between Nodes is described by Edges, where an Edge is a navigable path between two Nodes. Edges correspond to individual *Link Components* in Dexter. There may be multiple edges between the same two Nodes as we may navigate between them for different reasons: the "back" button and HTML fragment, "<a src=http://…"> can lead to the same Web page.

Both Nodes and Edges may have related Node or Edge Attributes. These attributes have a number of possible uses, one of which is timing synchronization. In a multimedia document for example, some edges are only available for navigation at particular times, and some nodes within a view may also be presented for a limited time. It is also possible to envision more complex synchronization issues; for example the visibility of certain Nodes in a View may be controlled by the Edge by which the user arrives at the View. This corresponds to use of *Synchronization Arcs* in the Amsterdam Model. Unlike the Amsterdam Model, detailed timing synchronization of multimedia elements, for example timing of captions on video, is assumed to take place in the lower layers of CISNA (unless the user can independently navigate to/from individual video/audio/caption elements.

4.2 Semantics Layer

The Semantics Layer, shown in Figure 6, is a rule-based meta-model, similar in approach to Prolog [10]; Prolog has 'facts' and 'rules' whilst the Semantics Layer has 'rules' and 'statements'. The Semantics Layer also has nouns, verbs, ontologies, and notions.

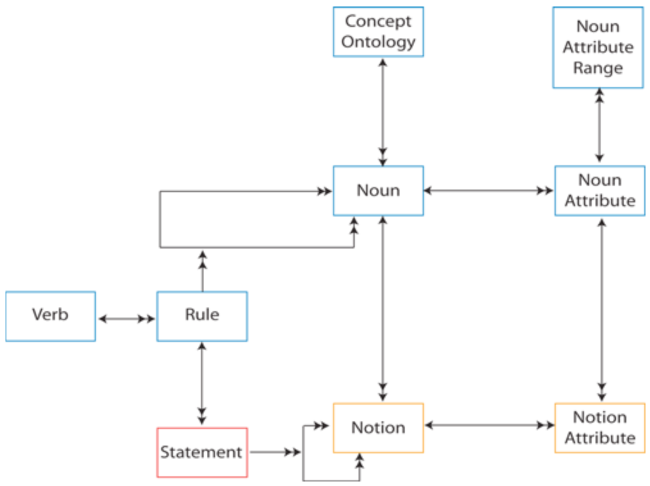


Figure 6 - Semantics Layer

The idea behind this approach is to create a model of semantics that can express anything from a word processing document, to a website, to an interactive game, to an augmented reality system. Each of these "document types" has differing semantic rules on how information is organized, and yet all have general abstract rules and concrete statements that can be expressed by the nine-element model of Figure 6.

Those nine elements group information according to Content Ontologies. Example ontologies include Container, Coordinate System, Media and Menu. For each ontology there exists a set of nouns that describe relevant entities within that ontology, for example Menu and Menu Item are both 'nouns' that describe entities within a Menu ontology. Similarly, Title, Protagonist, and Director may be 'nouns' in a Media ontology. Whilst verbs can also be considered part of an ontology, they are deliberately left outside of the definition so that Verbs have the same conceptual meaning for all ontologies in order to simplify the model. Examples of Verbs in this context include: 'is a', 'contains', 'scopes', 'expands upon', and 'follows'.

Nouns and verbs are used to make simple noun-verb-noun rules that describe underlying semantics. Simple example are; "Menu contains Menu Item", "Menu Item follows Menu Item", "Heading scopes Section". These rules are the analogue of "facts" in Prolog.

One verb stands out from the rest: "is a". A Menu is (sometimes) a List, a List Item is (sometimes) a List, and a Menu Item is (sometimes) a List Item. Logically therefore, if a List contains List Items, then a Menu contains Menu Items. Consequently, the Semantics Layer is constructed around the concept of *conditional multiple inheritance.* Conditional multiple inheritance is chosen to allow easy expression of concepts such as a Menu is sometimes is list of items, and sometimes a grid of items, for example the top-level menu on the Apple iPhone.

To support the Semantics Layer, a simple graphical notation was developed to express inheritance relationships and rules. An example of its use is shown in Figure 7.

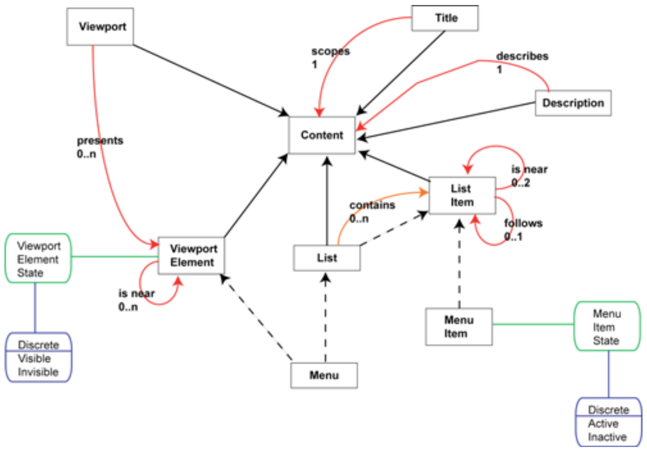


Figure 7 - Semantic Rules

In Figure 7:

- Rectangular boxes represent nouns.
- Solid, unlabeled arrowed lines represent "A is a B".
- Dashed, arrowed lines represent "A is sometimes a B".
- Labeled arrowed lines represent all other non-verb-noun associations; the text label represents the verb and its multiplicity.
- Rounded boxes represent attributes of the related noun.
- Rounded boxes with value lists represent the range of values for the related attribute.

Whilst the ontologies play no active part in the notation above, they form an important role in the overall CISNA Model. The Runtime System must understand the nouns contained within the ontologies in use, in order to be able to render the content. So, in the same way XML [11] uses schemas and DTDs, CISNA use ontologies to identify the kind of semantic structures in use within the Semantics Layer. So, the first task of the Runtime system is to validate that only supported ontologies are used by the document.

The notation itself is designed to be relatively simple to draw by hand, and easy to implement as a tool within the Development System. The small number of symbols and connectors make it a simple matter to create a validating editor application.

In the same way the Semantics Layer uses nouns, verbs, and rules to describe underlying semantic relationships, it also uses notions and statements to describe the concrete semantics of a particular document. A notion expresses some renderable concept, such as a specific menu, menu item, viewport, indicator (e.g. a message to a user); as such, notions are a specialization of nouns. In this way a Search Menu is a specialization of Menu, and Site Title is a specialization of Title. A statement links two notions together with a rule describing that relationship. The rule must be valid between both noun specializations in the statement. Statements are analogous to rules in Prolog.

The notation used to express rules was extended to support traceability from notion to noun and statement to rule. An example of its use is given in Figure 8.

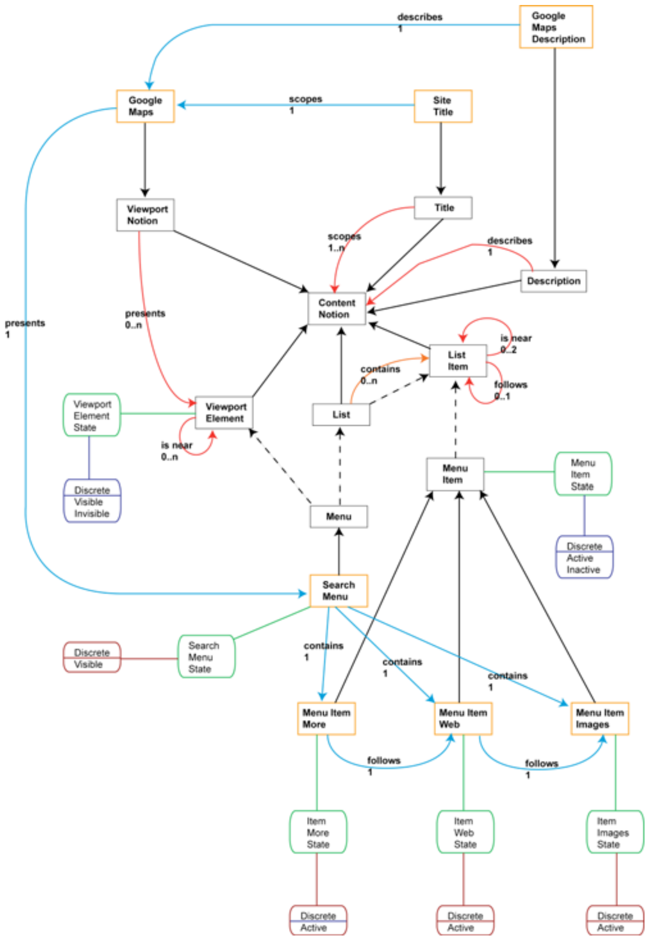


Figure 8 - Semantics Statements

In Figure 8:

- Rectangular boxes may also represent notions.
- Labeled arrowed lines may also represent statements.
- Rounded boxes may also represent notion attributes.
- Rounded boxes with a value represent notion attribute values.

Missing from Figure 8 are the relationships between notion attributes and the noun attributes that they represent, these have been left off for clarity.

One of the ideas behind the use of rules and statements in the Semantics Layer is the idea of having validatable models. Rules are common across many documents, statements describe specific content in a document; validating rules is analogous to validating an XML document using DTDs and Schemas.

For a statement to be true, the notions on each side of the rule to be applied must inherit from Nouns for which the rule is true.

For an inheritance relationship to be true, a notion must contain a notion attribute for each noun attribute in the inheritance tree, and for no others. Further, it must contain exactly one attribute, however many times the noun appears in the inheritance tree. This is needed to make the multiple inheritance work; we would not want two sets of attributes for Content (if it had any).

The chosen rules to validate non-inheritance statement 'A rule B' are:

1. 'A' and 'B' must be notions.
2. 'A' must inherit a noun on the left-hand side of the 'rule'.
3. 'B' must inherit a noun on the right-hand side of the 'rule'.
4. The implied inheritance trees for 'A' and B' must be valid.
5. The multiplicity of the statement must be equal to, or stricter than the multiplicity of the referred-to 'rule'.

The chosen rules to validate inheritance statement 'A is a B' are:

1. For each attribute noun between 'A' and 'B' there must exist a counterpart notion attribute in 'A'.
2. Each counterpart notion attribute must be in the same attribute range as the noun attribute and must hold a valid value for that range.
3. There must be no duplicate notion attributes in 'A'.
4. Any other 'orphan' notion attributes in 'A' must be counterpart attributes in another valid inheritance statement that involves a non-inheritance statement either from, or to, 'A'.

4.3 Inventory Layer

The Inventory Layer, shown in Figure 9, expresses all of the content available for interaction with a user. In terms of the Dexter/Amsterdam models, the Inventory Layer expresses the knowledge of content contained within a Component, and associates Anchors with individual inventory elements rather than with Components.

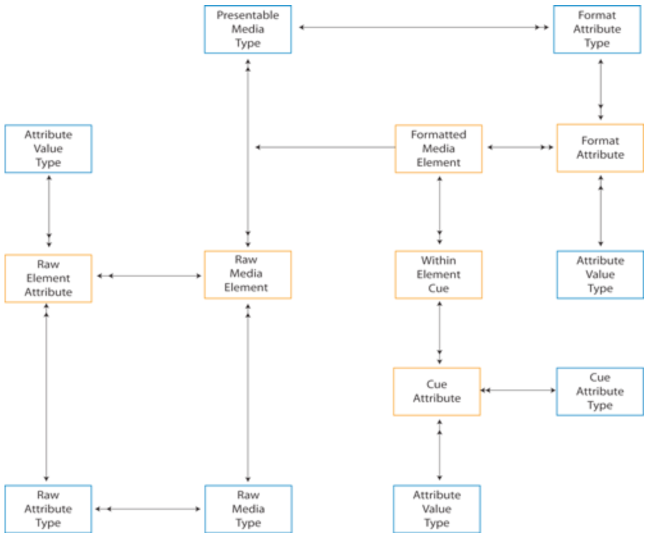


Figure 9 - Inventory Layer

The primary goal of the Inventory Layer is to provide a buffer between content and semantic meaning by providing an indexing service. In doing so, it steps away from the Dexter Model by making all content storage external to the model This is a logical extension of the Amsterdam Model's method of content referencing for multimedia.

An example of usage is that of adding bullet points to a document. An image used for a bullet point may be used multiple times in the same document, and may appear additionally on a tool-bar of bullet styles for the user to choose from. This gives the same image two semantic meanings (a bullet point, and the style of a bullet point), and multiple instances of its use. The Inventory Layer records only the properties of the image such as size, resolution, and encoding format, plus a reference to the location of the content (including mirror locations if required). Semantic meaning is left to the Semantics Layer, and grouping of bullet points for presentation is left to the Navigation Layer.

A second example of usage is buffering between content and semantics is the interaction with tangible items. This occurs with augmented reality systems, and with tangible user interfaces such as the marble answer-phone [12]. In both cases there exists within the 'document' physical items whose position and properties have semantic meaning within the document, and user manipulation of them causes logical navigation between groups of content. With the marble answer-phone, each message recorded delivers one marble into a bowl; feeding a marble back into the machine causes that particular message to play. An augmented reality example would be a navigation aid for users with low vision, where visual object recognition is used to identify known items in the real world and report their status and location to the user using ear-cons. Both the marbles, and the real-world items are referenced as *Media Elements*.

A third example of usage is concerned with text. The Inventory Layer provides a buffer between content and semantic meaning, but the nature of text is that, at the most basic level, it is a stream of letters and punctuation bound together by semantics. In terms of the Inventory, it is necessary to consider how much text is referenced as a single *Media Element*. The working definition chosen for this Model is:

A text Media Element comprises the largest block of text content possible that contains within it no semantic detail of interest to the Semantics Layer of the parent document.

Each Media Element is characterized by a Raw Media Type, with the range of types specific to the mark-up notation supporting the Model. Raw Media Types, particularly in adaptive systems, may not necessarily represent the format presented to a user. A simple example is text, where text may present visually using fonts, or as computer-generated speech, or as Braille; in terms of defining the construction of a user interface, it is the properties of the presented content that is of interest, not the original raw form. Tabular and graphical data may similarly have multiple forms of expression. For this reason, the CISNA Model expands on the Amsterdam Model's concept of media type, defining the Inventory Layer as containing 'formatted' elements in addition to raw content. A Formatted Media Element references the raw content, but it is the Presented Media Type that declares the final presented form, and the Format Attributes that define how this is to be achieved (e.g. text-to-speech using a specific).

One of the less considered areas of hypermedia is transient content. Whilst the Dexter Model does provide a view of live content modification, the underlying assumption is that content is static unless users explicitly request modification. Consideration of transient content is of particular importance to any adaptive system or assistive technology that relies on transcoding content between design spaces, for example from visual to sonic. It is necessary to know what content is transient, when it is valid for presentation, and when its content changes, in order to ensure that appropriate synchronization is provided where there is competition for resources as with audio channels. It is also necessary to deal with the case where ear-cons must be added to notify changes in, say, textual content when content plays out in text-to-speech. Within CISNA, meaning and validity are modeled within the Semantics Layer, but the placeholders for such content also exist as Media Elements in the Inventory Layer.

4.4 (External) Content Layer

Within CISNA, all content is considered external to any descriptions of its use and meaning. This is an extension of the Amsterdam Model's concept of a Media Component Reference where all multimedia content is considered to be external to the model. CISNA's separation of semantic structure from storage allows for re-use of text elements within a document, and for text element to be referred to in, say, external databases.

The CISNA model places no restrictions on what may be considered content, so long as provision exists to adequately reference that content from the Inventory Layer. It may, for example, be images on the Web, text in an encyclopedia database, or physical items in an augmented reality environment.

5. BRIDGES AND COUNTERPARTS

The five-layer CISNA document model introduced in Section 4, expresses the content and structure of a document as five independently modeled abstractions. The abstractions are considered as layers within a stack of increasing abstract concepts.

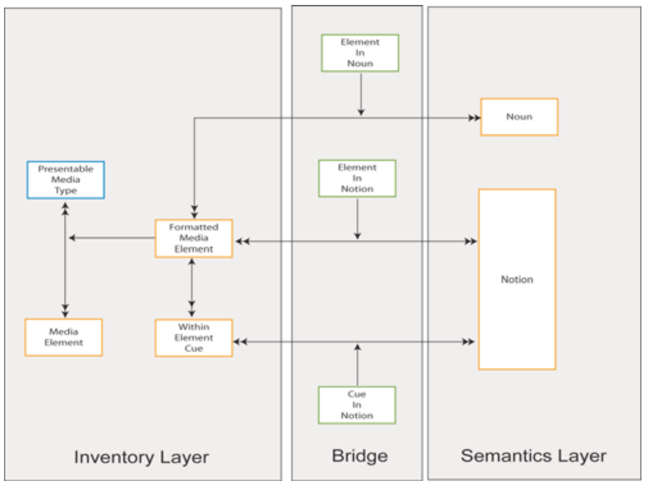


Figure 10 - Inventory & Semantics Bridge

Those layers that are adjacent to each other express semantically related concepts, and that collection of concepts is defined to be a *bridge* between the two related layers. This approach is a variation on the Shlaer-Mellor [13] object oriented analysis method that describes an application as a hierarchy of dependent, but independently modeled problem domains, with the dependencies described by a bridge. In CISNA, there are no dependencies between the models/layers, but simply relationships that are expressed as models.

The Inventory & Semantics Bridge, shown in Figure 10, expresses the three relationships that link the Inventory and Semantics layers: Element in Noun, Element in Notion, and Cue in Notion. The three relationships map media elements to the notions and nouns of the semantic layer. For example the text "Google Maps" may relate to notion "Site Title". Many media elements may map to the same notion (for example to support multiple languages) and may be of different raw media types (for example strings of Rebus Symbols [14] to support users with specific learning difficulties). Where there are multiple alternative mappings, a reason code is provided for each mapping in the bridge.

The Semantics & Navigation Bridge, shown in Figure 11, maps between notions in the Semantics Layer and nodes in the Navigation Layer. For example the notion, "Google Maps" may map to the View node, "Google Maps Application", as may the notion, "Google Maps Description". This occurs because a bridge describes counterparts, i.e. "Notion A exists when we have Node B", and **not** "Notion A is Node B".

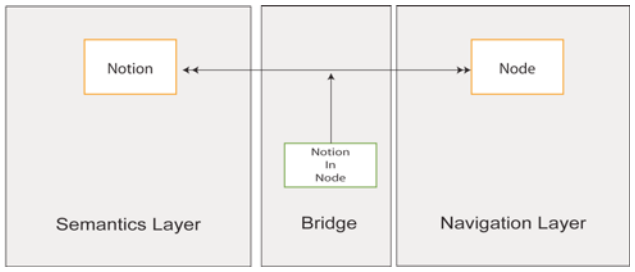


Figure 11 - Semantics & Navigation Bridge

6. Adaptation Layer

The Content, Semantics, Inventory and Navigation Layers describe possible organizations of content, semantics and navigation within a document. What makes the document concrete is the specific selection of instances of each layer, where an instance is a transaction that expresses change to the layers in terms of add/modify/delete. Because an instance is a kind of transaction, the order of application of instances is important, hence the relationship between Instance and InstanceApplication, shown in Figure 12. Example instances may be "Default User", "Default to Low vision user", "Default User to Blind user", "Blind User to Blind User with restricted mobility".

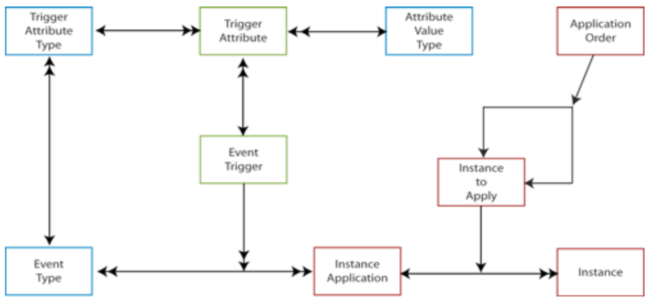


Figure 12 - Adaptation Layer

There are two types of document to consider during adaptation of content: passive and dynamic. A document is passive if no new instances are realized within the document during a single session with a user. A newspaper is passive because once printed, no further modification is made. It may be pulled apart and shared between users, but that is still within the navigation and semantic models of "NEWSPAPER". Similarly, a no-script HTML web page is passive, because once rendered by the browser (possibly with reference to a style sheet) no further change is made beyond possibly scrolling the content. Note that the ability for a user to change a style sheet in the browser does not make a document dynamic as in practical terms browsers will (or at least should) re-render the complete page, restarting the session with the user.

A document is dynamic if new instances are realized within the document during a single session with a user. An HTML Web page becomes dynamic when it includes scripting. Using *onMouseOver="…"* to implement a button rollover for example, makes the page dynamic since what is rendered is modified during presentation. Similarly, some Microsoft Word documents become dynamic if they include interactive macros, or embed interactive content within them. Even such simple actions as rollovers are important to consider, as there may be equivalent behavior to consider when content is adapted to other design spaces, for example button rollovers may become changes in pitch of the avatar's voice as it reads the "alt text" related to the button.

The version of the Adaptation Layer presented in this Section, addresses only passive documents. The model of adaptation presented, describes how instances are selected and applied to a document as a result of defined events. Note that these are pre-defined instances that can, for example, add/change a language, or map content between design spaces, for example switching from the 'Default User' to a 'Low Vision User'.

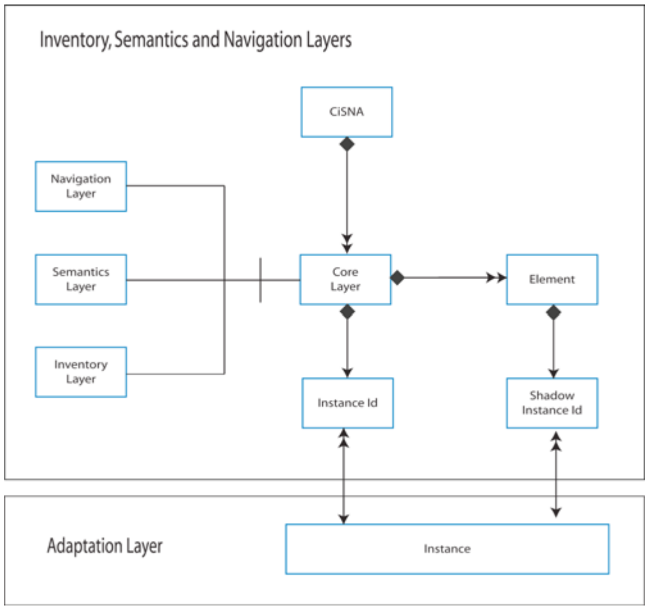


Figure 13 - Adaptation Bridges

The Adaptation Layer has bridges to all other layers, as shown in Figure 13. Instances in the adaptation layer map to instances of individual layers i.e. transactions of add/modify/delete applied to a named "instance". Each "core" layer is composed of multiple instances of elements (the objects described in rectangular boxes in each layer model), and each instance of each element is related by the bridge mapping to a specific instance in the Adaptation Layer. Put simply, all information regarding the default user is mapped to a "Default" instance, all information regarding modification of the "Default" instance to support a "Low vision" user is mapped to the "Default to Low Vision" instance. When an element in one instance refers to an element in another instance, the referred to element exists as a "shadow" of that instance. This is analogous to the "extern" statement in programming languages such as C and C++, and to importing package content in Java.

7. PROTOTYPE

A simple, fully functioning prototype, based on a small fragment of Google Maps [15], is presented to illustrate the practical use of the CISNA Model. A screenshot of the relevant part of Google Maps is shown in Figure 14.

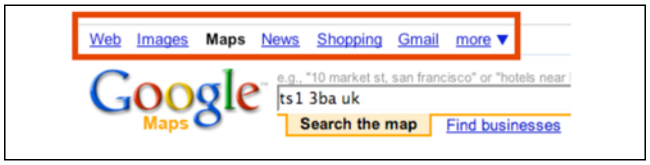

Web Images Maps News Shopping Gmail more ▼
Google Maps
e.g., "10 market st, san francisco" or "hotels near
ts1 3ba uk
Search the map
Find businesses

Figure 14 - Search Menu

The red highlighted area of Figure 14 may be interpreted in many ways. For this example, it is assumed to represent a top-level navigation menu for Google's applications, with the "Maps" text representing the page title; even though the menu/text is a sequence of text hyperlinks, the downward facing triangle at the end of the list suggests that it is a menu, and that menu item "more" has a sub-menu. To further shorten the example, only the first three links plus "more" and the triangle are considered. From this foreshortened menu, two users are considered: a default user, and a low vision user who requires text-to-speech in addition to the default visual representation.

Five text elements exist in the Inventory Layer as Raw Media Elements: "Web", "Images", "Maps", "More" and the triangle character. For the default user, these require formatting, i.e. expressed as FormattedMediaElements for use with fonts. For the low vision user, they require formatting additionally for text-to-speech.

Both font and text-to-speech representations, map through the Inventory & Semantics Bridge to notions representing the site, menus, and menu items. The resulting semantic model is shown in Figure 8.

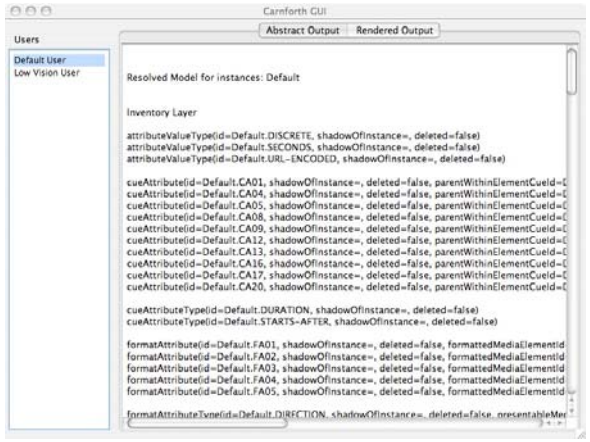
For the prototype, an XML-based notation was developed to express each layer and bridge. An example fragment of that notation for the Semantics Layer is shown in Figure 15. The XML for the Adaptation Layer is shown in full in Figure 18Figure 18.

The prototype application was constructed in Java, using the FreeTTS [16] implementation of the Java Speech API [17]. The application provides a menu to select a user profile, and two output windows, one for the rendered "abstract model", and for the rendered "concrete" model. The abstract output, shown in Figure 16, is in the form of a relational database table dump, with the tables populated with the resolved, adapted content.

The concrete output, shown in Figure 17, is the rendered fragment of the Google Maps page. In addition to the apparent visual content, rendering the content also produces the spoken text "Application Google Maps. Menu Start. Item Web. Item Images. Item More. End menu.".

```
<cisna>
    <semantics instance="Default" >

        <model>

            <conceptOntology id="Default.MENU" />
            . . .
            . . .
            <noun id="Default.MENU ITEM"   ontologyId="Default.MENU" >
                <nounAttribute id="Default.MENU ITEM STATE" >
                    <attributeRange id="Default.AR01" type="DISCRETE" >
                        <rangeValue id="Default.STATE-ACTIVE" value="ACTIVE" />
                        <rangeValue id="Default.STATE-INACTIVE" value="INACTIVE" />
                    </attributeRange>
                </nounAttribute>
            </noun>
                ...
                ...
            <verb id="Default.IS A" />
            <verb id="Default.CONTAINS" />
            <verb id="Default.FOLLOWS" />

            <rule id="Default.R01.Is A" nounId1="Default.LIST"
                              verbId="Default.IS A"
                      nounId2="Default.CONTENT NOTION"
                   multiplicity="1"  />
            ...
            ...
        </model>

        <content>

            <notion id="Default.Menu Item Web"  nounId="Default.MENU ITEM"  >
                <notionAttribute id="Default.itemWebState"
                   nounAttributeId="Default.MENU ITEM STATE"
                              value="ACTIVE" />
            </notion>
            ...
            ...
            <statement id="Default.S02" notionId1="Default.Search Menu"
                             ruleId="Default.R02.Contains"
                           notionId1="Default.Menu Item Web"   />
            ...
            ...
        </content>

    </semantics>
</cisna>
```

Figure 15 - Semantics Layer as XML



```
Resolved Model for instances: Default

Inventory Layer

attributeValueType(id=Default.DISCRETE, shadowOfInstance=, deleted=false)
attributeValueType(id=Default.SECONDS, shadowOfInstance=, deleted=false)
attributeValueType(id=Default.URL-ENCODED, shadowOfInstance=, deleted=false)

cueAttribute(id=Default.CA01, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA04, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA05, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA08, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA09, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA12, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA13, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA16, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA17, shadowOfInstance=, deleted=false, parentWithinElementCueId=
cueAttribute(id=Default.CA20, shadowOfInstance=, deleted=false, parentWithinElementCueId=

cueAttributeType(id=Default.DURATION, shadowOfInstance=, deleted=false)
cueAttributeType(id=Default.STARTS-AFTER, shadowOfInstance=, deleted=false)

formatAttribute(id=Default.FA01, shadowOfInstance=, deleted=false, formattedMediaElementId
formatAttribute(id=Default.FA02, shadowOfInstance=, deleted=false, formattedMediaElementId
formatAttribute(id=Default.FA03, shadowOfInstance=, deleted=false, formattedMediaElementId
formatAttribute(id=Default.FA04, shadowOfInstance=, deleted=false, formattedMediaElementId
formatAttribute(id=Default.FA05, shadowOfInstance=, deleted=false, formattedMediaElementId

formatAttributeType(id=Default.DIRECTION, shadowOfInstance=, deleted=false, presentableMe
```

Figure 16 - Abstract Model Output

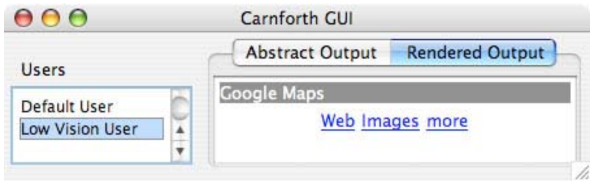
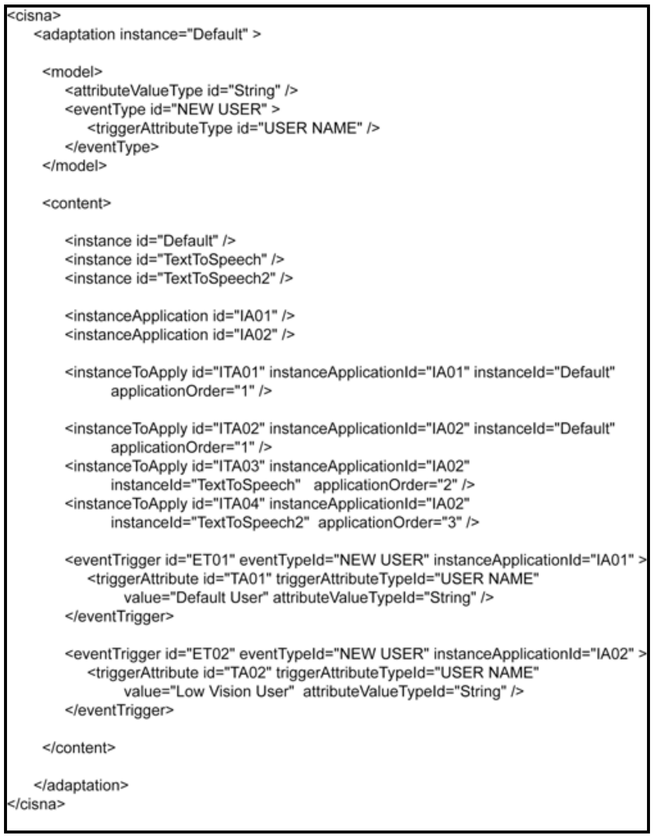


Figure 17 - Concrete Model Output



```
<cisna>
    <adaptation instance="Default" >

     <model>
         <attributeValueType id="String" />
         <eventType id="NEW USER" >
             <triggerAttributeType id="USER NAME" />
         </eventType>
     </model>

     <content>

         <instance id="Default" />
         <instance id="TextToSpeech" />
         <instance id="TextToSpeech2" />

         <instanceApplication id="IA01" />
         <instanceApplication id="IA02" />

         <instanceToApply id="ITA01" instanceApplicationId="IA01" instanceId="Default"
                  applicationOrder="1" />

         <instanceToApply id="ITA02" instanceApplicationId="IA02" instanceId="Default"
                  applicationOrder="1" />
         <instanceToApply id="ITA03" instanceApplicationId="IA02"
                  instanceId="TextToSpeech"   applicationOrder="2" />
         <instanceToApply id="ITA04" instanceApplicationId="IA02"
                  instanceId="TextToSpeech2"  applicationOrder="3" />

         <eventTrigger id="ET01" eventTypeId="NEW USER" instanceApplicationId="IA01" >
             <triggerAttribute id="TA01" triggerAttributeTypeId="USER NAME"
                   value="Default User" attributeValueTypeId="String" />
         </eventTrigger>

         <eventTrigger id="ET02" eventTypeId="NEW USER" instanceApplicationId="IA02" >
             <triggerAttribute id="TA02" triggerAttributeTypeId="USER NAME"
                   value="Low Vision User"  attributeValueTypeId="String" />
         </eventTrigger>

     </content>

    </adaptation>
</cisna>
```

Figure 18 – Adaptation Layer as XML

8. DISCUSSION

This paper has concentrated upon the abstractions within the five layers of the CISNA document model, with only a little discussion of the Development and Runtime Systems. To render this abstract user interface requires an understanding of user capability in order to identify the content of an "instance" of adaptation. It also requires that appropriate interaction modalities be selected when rendering to each design space (visual, sonic, or haptic). Both of these issues are addressed within a broader research project at the University of Teesside that extends the concepts within Nesbitt's Multi-sensory Design Space [18], with the CISNA document model forming the core.

By focusing upon the five-layer stack, the result is a model and discussion centered on abstract user interfaces, and within that context CISNA is but one of many approaches to the problem of separating content from presentation. XForms is perhaps the most notable of these in terms of practical implementation on the Web, but all W3C notations: HTML, XHTML, SVG, and SMIL, to a greater or lesser degree also attempt this separation of concerns through the use of a Document Object Model (DOM).

What makes CISNA special, compared to the existing DOM approach, is the focus on adaptation of content, and a separation of concerns not only between abstract and concrete representation, but between the different subject areas of concern to adaptation, and by implication, to assistive technology and accessibility. The CISNA Model allows discussion and individual adaptation strategies to be applied when amending content, document semantics, document navigation, and ultimately to dynamically adaptive behavior, discussed later in this Section. Further, those adaptations are described in terms of simple instances of add/modify/delete for all areas of concern.

This concept of an instance is one of the big steps forward over the traditional DOM model, where changes to the DOM are expressed through scripting. Each of the CISNA layers, and each bridge are expressed as a Shlaer/Mellor Information Model [13], which is a close relative of the relational database, and this allows for the document as a whole to be viewed as a single, relatively simple, relational database. Once in database form, all of the standard, established, computer science database theory and methods become available to the rendering engine of the user interface. One of those approaches is the transaction, which allows for lists of changes based on add/modify/delete requests to be processed. It is this underlying transactioning approach that becomes the "Instance" object in the Adaptation Layer.

Instances are also important in terms of accessibility. One of the perennial problems in discussing accessibility is finding ways to efficiently explain the difference between, say, a default user interface, and one adapted to a specific user need. If a user interface is described using the CISNA Model, then the required changes can be expressed precisely in abstract terms, focusing on content, semantics, navigation, adaptivity as required. This provides one small step on the road to creating a formal language of accessibility. Further, being able to describe the differences to a user interface, gives rise to the possibility of creating a standard measure of accessibility: how much change is required to adapt a user interface for a given set of representative user profiles? If we consider the default and modified models as expressed in XML, as in the prototype example, we have two sequences of XML that we can compare. Comparing such sequences occurs frequently in DNA research, and one of the measures is the *Levenshtein-Distance;* this approach has also been recently proposed for analyzing web server log files to identify patterns and differences in navigation through hypertext. Measuring accessibility in this was is currently outside of the scope of the CISNA Model research but is clearly a route forward for future work.

The CISNA model as described in this paper, stops at the point of dynamically adaptive systems. The full model splits the Adaptation Layer into Configuration and Adaptivity subsystems. The adaptivity subsystem is shown in Figure 19 below.

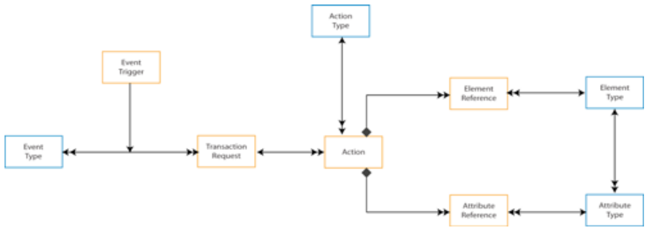


Figure 19 - Adaptivity Subsystem

The Adaptivity Subsystem still describes behavior in terms of an Information Model, with the key element, the Action object. An Action expresses procedure (e.g. scripting in a web page) and relates that procedure to the elements and attributes of the individual layers affected by the Action. This gives enough basic information for the designer of assistive technology to understand what content with the layers is transient when considering adapting content, or moving it between design spaces. The question then becomes: can an Action itself be adapted, and can that adaptation be expressed within the CISNA notation? The answer to that appears to be "yes", and part of the research surrounding the CISNA model treats Actions as Directed Graphs, and since Shlaer/Mellor Information Models express Directed Graphs, Actions may be expressed in the same notation. This is ongoing work.

Once noticeable effect of the CISNA Model on the XML descriptions in the prototype example is the sheer quantity of XML necessary to express even a trivial application, certainly in comparison to the short fragment of HTML that would be needed to express it purely visually. Part of this volume of XML is in fact stable content; the underlying semantic model of the ontologies, verbs, and rules of a document are constant and arguably should not be expressed in each and every document that use them. With this in mind, the XML separates each layer into <model> and <content> fragments, with the expectation that the model would be referred to in a manner analogous to a DTD referenced at the beginning of XML fragments. This reduces XML volume on the trivial prototype example by almost 50%. The volume of XML in the <content> fragments to some extent reflects the amount of additional information provided in comparison with the equivalent HTML. In the Inventory Layer, it is possible to express different renderings of content; if the CSS styles necessary to describe text-to-speech encoding are added to the HTML, the differences in XML-HTML volume disappear. Similar observations are possible for the Semantics and Navigation Layers.

9. CONCLUSION

Experience with the five-layer CISNA Document Model has demonstrated that it is capable of effectively expressing non-adaptive user interfaces, and ongoing work indicates that adaptive environments are also supportable. The most significant step forward in terms of accessibility is the ability to clearly and simply express what changes are required to adapt a document/user interface to a specific user's needs, and that list of changes can be supplied as a "patch" to the document. This is analogous to a style sheet in HTML, but rather than simply changing the decoration of the document, it can make substantive changes, adapting/augmenting the content, semantics, and navigation. In this respect, it is a significant step forward for accessible design of user interfaces and electronic documents.

10. REFERENCES

[1] Halasz, F. and Schwaertz M., The Dexter hypertext reference model, In *Communications of the ACM vol 37, issue 2, pp 30-39*, ACM, 1994.

[2] Freedom Scientific Inc product catalogue, 2008, http://www.freedomscientific.com

[3] W3C, HTML 5 Working Draft 22 January 2008, http://www.w3.org/TR/html5/

[4] W3C, The XMLHttpRequest Object Working Draft 26 October 2007 http://www.w3.org/TR/XMLHttpRequest/

[5] Google Inc, Google Select Website http://www.google.com/webhp?complete=1&hl=en Viewed 18 January 2008

[6] Hardman L. et al, The Amsterdam hypermedia model: adding time and context to the Dexter model, In *Communications of the ACM vol 37, issue 2, pp 50-62,* ACM, 1994

[7] W3C, Synchronized Multimedia, http://www.w3.org/AudioVideo/ Viewed 18 January 2008

[8] W3C, Synchronized Multimedia, Appendix B http://www.w3.org/AudioVideo/ Viewed 18 January 2008

[9] W3C, XForms specification version 1.0, 2006. http://www.w3.org/TR/2006/REC-xforms-20060314/

[10] ISO, ISO Prolog Standard ISO/IEC 13211-1:1995

[11] W3C, Extensible Markup Language (XML), Fourth Edition http://www.w3.org/TR/REC-xml/

[12] Svanaes D. and Verplank W., In search of metaphors for tangible user interfaces, In *Proceedings of DARE 2000 on Designing augmented reality environments*, ACM 2000.

[13] Mellor S.J. and Balcer M.J., *Executable UML: a foundation for model-driven architecture*, Addison-Wesley, Reading, MA, 2002.

[14] Widgit Software, The Widgit Symbols Development Project, http://www.widgit.com/widgitrebus/ Viewed 4 February 2008

[15] Google Inc, Google Maps website, http://maps.google.com/

[16] Sun Microsystems Inc, FreeTTS http://freetts.sourceforge.net/docs/index.php Viewed 4 February 2008

[17] Sun Microsystems Inc, Java Speech API http://java.sun.com/products/java-media/speech/ Viewed 4 February 2008

[18] Nesbitt K.V., Modeling the Multi-Sensory Design Space, In *Australian symposium on Information visualization, - Volume* 9 (CRPITS'01), Australian Computer Society, 2001