Semantic Annotations for Digital Video

Alexander Nakhimovsky[1], Chris Hellmuth[1], Tom Myers[2]

[1] Colgate University, Computer Science Dpt, 13 Oak Drive
Hamilton NY 13346 USA
Alexander Nakhimovsky, adnakhimovsky@colgate.edu
[2] N-Topus.com, 56 Payne Street
Hamilton NY 13346 USA
tommyers@dreamscape.com

Abstract. This paper describes a functioning system that associates semantic annotations (in the form of a table of triples of identifiers) with digital video. The first part describes a subsystem (in existence since 2002, desktop version 1995) of adding annotations to digital video. The second part builds on that subsystem, creating additional semantic annotations that enable semantic-web retrieval. The key component of the second subsystem is a glossary for the text being indexed, created by the user in interaction with Princeton's WordNet. The resulting glossary is a microformat within an XHTML document, which we validate using our microformat validator. Additional techniques for harvesting metadata from the domain expert are described.

1 Multimedia Annotator - XML (MannX)

MannX consists of two programs, MannX-author and MannX-player. MannX-author creates MannX applications that are accessible via MannX-player. The main functionality of MannX-author is to establish a segment-by-segment correspondence between time-segments of video and space-segments of associated text (which may or may not be the transcript of the video). This is done in the point-and-click fashion; an hour of video can be synchronized with its text in about 90 minutes.

MannX-player provides the following functionaliity:

Navigation by text and by video: the ability to find a video segment corresponding to a selected stretch of text and vice versa. We have, in effect, random access to time-based video based on its content. In other words, text segments serve as indices of an associative array whose values are video segments. The converse is also true: given a segment of video (showing, e.g., a particularly tricky step in the bypass operation) the user can click to see the corresponding segment of the expanatory text.

Segment replay: the ability to replay a segment of video while viewing the corresponding text and commentary.

Hypermedia annotation of text and video: the ability to attach arbitrary annotations to a segment of text (and therefore to the corresponding video segment), attach more than one set of annotations to the text, and easily switch between them. (For

instance, in language-learning contexts one may want to have grammatical, lexical, stylistic and cultural commentaries.) The reason we can have multiple annotations is that we do NOT attach them directly to the video file, as in MPEG 7, but via a text file with matching segments.

Dictionary lookup: typically, a MannX application has a glossary (bilingual or monolingual) of associated terms. The glossary can be associated with a single application or a group of applications that together form a course or a manual. This functionality is relatively straightfoward, but the glossary is an important component of the system of semantic annotations described in this paper.

The screenshot below shows a typical MannX screen with four frames: a video recording of a lecture; its transcript; an English-Pashto glossary (Pashto is one of two official languages of Afghanistan); and a commentary that in this case consists of translations of transcript segments into Pashto and slides accompanying the lecture.

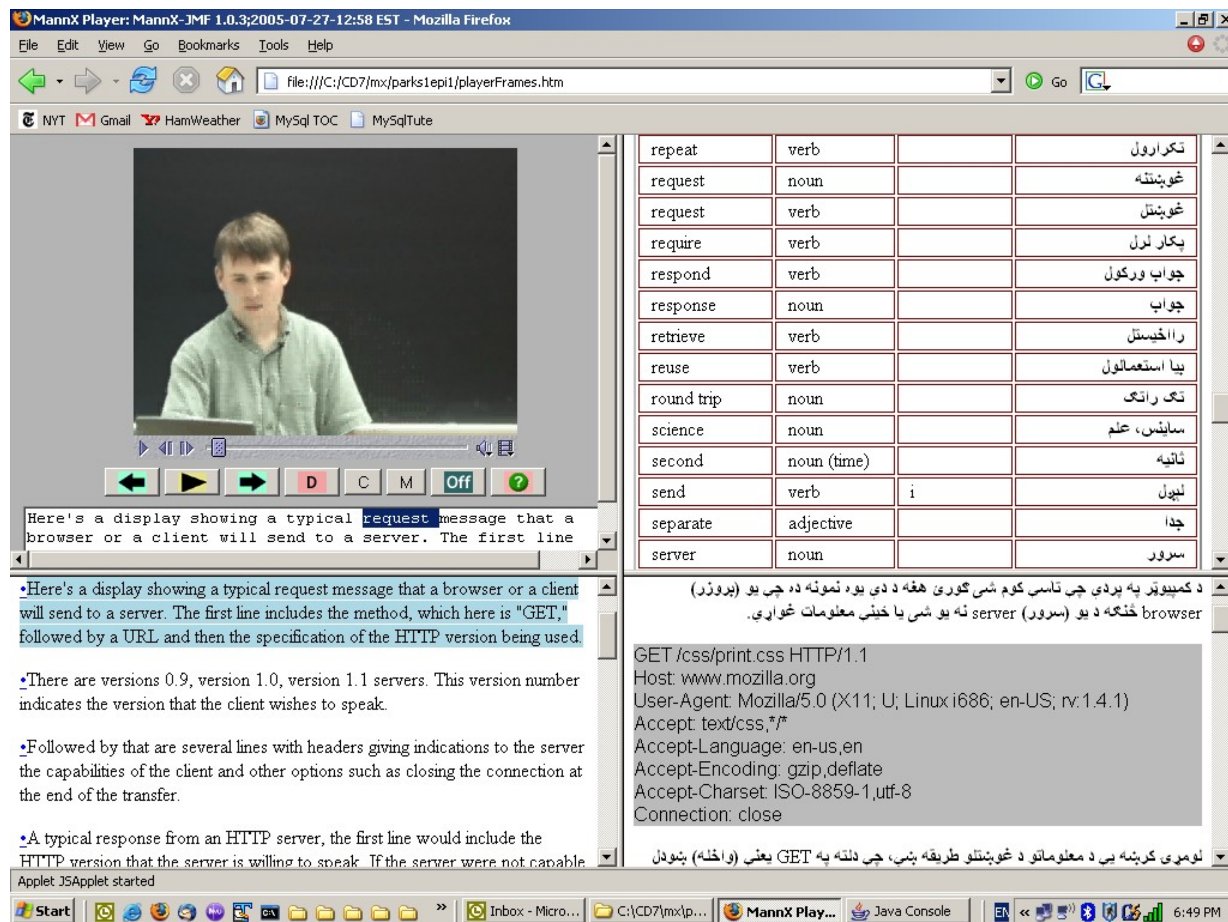


Fig. 1. The MannX Screen

The first version of MannX (Nakhimovsky, 1997) was a desktop application for Windows and Mac OS, written in C++, in which text segments were indicated by byte offsets, and adding functionality was difficult. The current version (Nakhimovsky and Myers, 2003) is written in Java, JavaScript and XSLT, so text segments and glossary entries are indicated by XHTML "microformats" (see, e.g., Dubinko 2005), and the system is open to the entire array of XML and Semantic Web technologies, including those that move effortlessly between sets of triples and RDF, RDFS and OWL. We test on Red Hat Linux, Windows XP and OSX. In this paper we describe an extension of the MannX system that adds knowledge-based indexing and retrieval of digital video and associated text, using an enriched glossary and a database of triples that represent semantic relations between items in the glossary and the text.

2 Microformats and Validation

In this section we present and illustrate with examples the new or revised components of the MannX system. They include:

Glossary, revised as an XHTML microformat ntGloss;

Triples table, also a microformat ntTriples. We use the prefix nt, as in n-topus.com, as a lightweight substitute for a namespace.

XhtmlDecoder, a general JavaScript class for building and validating microformat objects. Both ntGloss and ntTriples are subclasses of XhtmlDecoder. (See Nakhimovsky and Myers 1998 for OOP in JavaScript.)

The alpha.html utility for building glossaries from text. It has been greatly revised from the earlier version to include semantic information retrieved from WordNet (http://wordnet.princeton.edu/perl/webwn).

The code sample below shows the ntGloss and ntTriples microformats:

```
<table id="glossTable" class="ntGloss">
<tr id="sound_n1" class="ntDef">
  <td class="ntTerm">sound</td><!-- the lemma of the entry-->
  <td class="ntPos">noun</td>  <!-- Part of Speech -->
  <td class="ntDesc">          <!-- description of meaning-->
    the particular auditory effect produced by a given cause
  </td>
  <td class="ntExList"><!-- List of examples -->
      <span class="ntEx">the sound of rain on the roof</span>
      <span class="ntEx">the beautiful sound of music
      </span></td>
  <td class="ntPatList">
    <span class="ntPat">sound</span>
  </td>
</tr></table>

<table id="triplesTable" class="ntTriples">
 <tr class="ntRel">
     <td class="ntRelX">sound_n1</td>
     <td class="ntRelR">subClass</td>
     <td class="ntRelY">perception_n3</td>
  </tr></table>
```

Our microformat implementation depends on using CSS class attributes which are also the names of JavaScript classes. A microformat structure is defined by a JavaScript object (associative array), which serves the function of a DTD or a grammar. Consider the definitions below:

```
var ntGlossParts = {ntGloss:{ntDef:"*"},

ntDef:{ntTerm:1,ntPos:1,ntDesc:1,ntExList:"?",ntPatList:"?"},
                ntTerm:{},
                ntPos:{},
                ntDesc:{},
                ntExList:{ntEx:"*"},
                ntPatList:{ntPat:"*"},
                ntEx:{},
                ntPat:{}
}
var ntTriplesParts = {ntTriples:{ntRel:"*"},
                ntRel:{ntRelX:1,ntRelR:1,ntRelY:1},
                ntRelX:{},
                ntRelR:{},
                ntRelY:{}
};
BuildDecoderClassesFor(ntGlossParts);
BuildDecoderClassesFor(ntTriplesParts);
```

These are reasonably readable: for instance, ntGloss consists of 0 or more ntDef's, where an ntDef consists of a term, Part of Speech, description, an optional list of examples, and an optional list of Regular Expression patterns to match when looking for instances of this term in the text. (In the example, the list of patterns consists of just the literal identical to the term.) The function BuildDecoderClassesFor() uses the class definitions to build subclasses of XhtmlDecoder that conform to those definitions. Being a subclass of XhtmlDecoder is good because you can use the builder() method of that class to create an object of the specific subclass from XHTML data, and to validate it in the process. If the data does not conform to the microformat, an exception will be thrown:

```
function doIt(x){
  try{
    builder(top.document.getElementById(x),ntTriplesParts);
  }catch(ex){alert("ERROR in building "+x+"; "+ex);
}
```

The builder() method recursively traverses the DOM subtree specified by the first argument, and checks all its elements against the class definition provided by the second element. The code of XhtmlDecoder (close to 200 lines without comments) forms a self-standing module that can be used in many microformat applications, providing the benefit of validation, providing the benefits of validation, standardized data access and some automatic processing. For example, the first ntDef definition within an ntGloss object can be accessed within an ntGloss method as

```
this.xdParts.ntDef[0]
```

This is because each non-leaf class is constructed with an xdParts object which contains its part-objects, as well as a reference to the XHTML DOM element from which each object came; in this case

```
this.xdParts.ntDef[0].xdDomElement
```

If we provide an "onclick" method for the ntDef class in Javascript, then the glossary initialization will automatically pass that onclick to each XHTML element which is recognized and processed as an ntDef.

3 The Glossary and the Triples

In this section we show simple examples of the Glossary and the Triples structures, and step through code that shows them in operation. The examples are coming from a simple XHTML page that contains a glossary table, a triples table, and the buttons to invoke the builder() method on them (see Figure 2).

First consider an example of Glossary, with definitions coming from WordNet:

```
<table border="1" id="glossTable" class="ntGloss">
<tr id="sound_n01" class="ntDef">
  <td class="ntTerm">sound</td>
  <td class="ntPos">n</td>
  <td class="ntDesc">
    the particular auditory effect produced by a given cause
  </td>
  <td class="ntExList">
    <span class="ntEx">the sound of rain on the roof</span>
    <span class="ntEx">the beautiful sound of music</span>
  </td>
  <td class="ntPatList"><span class="ntPat">sound</span></td>
</tr></table>
```

A click on the buildGloss button invokes the doIt() function of the preceding section, with “glossTable” as its argument. Similarly, a click on the buildTriples button invokes the doIt() function on the Triples table, whose first line goes like this:

```
<table border="1" id="triplesTable" class="ntTriples">
```

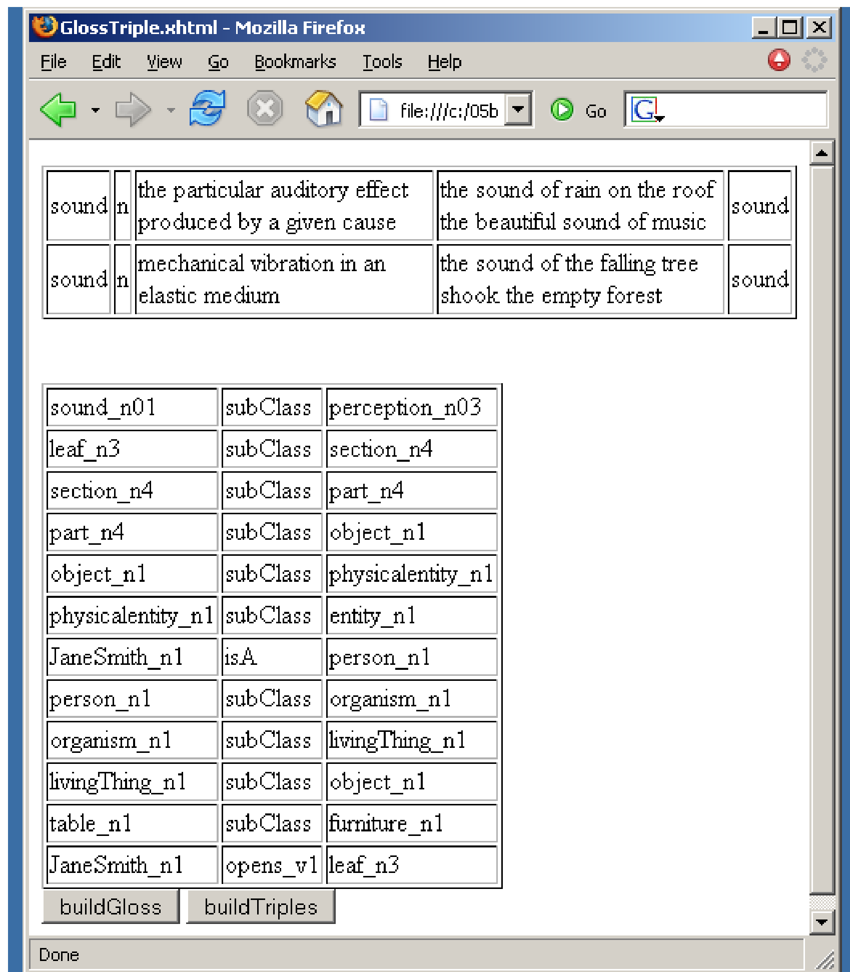
The result of these invocations of doIt() is that two JavaScript objects are created, structured as described in Section 2. At this point, one detail of the structure of the Glossary becomes important. As you recall, ntGloss consists of 0 or more ntDef’s, where the ntDef structure is defined as follows:

```
ntDef:{ntTerm:1,ntPos:1,ntDesc:1,ntExList:"?",ntPatList:"?"}
```

In addition to static fields, an ntDef object also has the onclick() method defined in the code sample below. Note that while onclick() is defined as an ntDef method, it is actually executed through a click on the associated DOM element, and so the “this” keyword in the second line of the code refers to the DOM element, not to the ntDef object:

```
function ntDef_onclick(evt){
```

```
    var ntDefId=this.getAttribute("id");
    var theGlossTable=top.builder.obs['glossTable'];
    theGlossTable.selectedDef=ntDefId;
}
ntDef.prototype.onclick=ntDef_onclick;
```



sound	n	the particular auditory effect produced by a given cause	the sound of rain on the roof the beautiful sound of music	sound
sound	n	mechanical vibration in an elastic medium	the sound of the falling tree shook the empty forest	sound

sound_n01	subClass	perception_n03
leaf_n3	subClass	section_n4
section_n4	subClass	part_n4
part_n4	subClass	object_n1
object_n1	subClass	physicalentity_n1
physicalentity_n1	subClass	entity_n1
JaneSmith_n1	isA	person_n1
person_n1	subClass	organism_n1
organism_n1	subClass	livingThing_n1
livingThing_n1	subClass	object_n1
table_n1	subClass	furniture_n1
JaneSmith_n1	opens_v1	leaf_n3

Fig. 2. Glossary and Triples pages

Since every line in the XHTML glossary table has the class attribute ntDef, they all acquire an onclick event handler that stores the clicked-upon definition in a JavaScript variable. This makes it available for further manipulation, as described in the next section.

4 Search through the Text

In this section, we connect the Glossary and the Triples into a system that enables extended text search, augmented by triples and regular expressions. The text that is searched is a MannX script text divided into segments corresponding to video segments, so we are, in effect, searching through the video as well. (Every "hit" can be immediately replayed.) The current functionality is as follows: given a glossary entry (either selected directly by a click or resulting from a lookup), we can search the text for all MannX segments that contain the same term or any of its ancestors in the WordNet hierarchy of concepts. In other words, given a term T and a current segment S, the program will find the next segment containing T' such that T stands in the "itemIn" relationship to T', where itemIn is the reflexive transitive closure of isA and subclass:

```
x itemIn x.
x itemIn z :- (x isA y | x subClass y) & y itemIn z.
```

For instance, if the term is "protocol" and the lexical item is "protocol_n1," then the search will find all the instances of "protocol," "protocols" and "protocol stack" (if that phrase has an entry in the glossary, but it will also find all the instances of "HTTP" because the pattern "HTTP" is associated with the lexical item HTTP_n1, and the triples table has a triple HTTP_n1 isA protocol_n1.

In order to achieve this functionality, we go through several processing steps, starting with the XHTML "script" of the video. (The word "script" is in quotes because it can be any text that has been broken into segments synchronized with segments of the video.) The script contains synchronization markup, such as 869, where 869 is the number of milliseconds from the beginning of the video, but this markup is irrelevant for our processing steps. The steps are as follows:

Create a glossary structured as the ntGloss microformat. This is done interactively, utilizing WordNet.
Create an (additional) triples table structured as the ntTriples microformat.
Add markup to the script linking text words to glossary items.

These steps are all performed by code in alpha.js, invoked from alpha.xhtml. They are further detailed and illustrated with examples below.

4.1 Creating a Glossary

This is in itself a multi-step process, proceeding as follows:

Step 1: Convert text to an alphabetical list of words and phrases. This removes punctuation and numerals; identifies phrases using the patterns of capitalization: "Computer Science" or "Professor Tom Parks" become entries; breaks the text into

words and phrases, and alphabetizes them. This code is run by clicking the wordsToList button.

Step 2: Convert the list of words and phrases into an HTML table structured as the ntGloss microformat. Also at this stage, N most common words are removed from the list, where N<=500. This code is run by clicking the listToTable button. The output of this step is placed into a text area for post-editing: additional items can be removed, items like 802.11 can be restored.

Step 3: Build the glossary object. In this step we fill in the cells of the table created in the preceding step with dictionary data. The code of this step is run by clicking the buildGloss button. This places the XHTML code of an ntGloss table into a scroll <div> element (with yellow background in the Figure below).

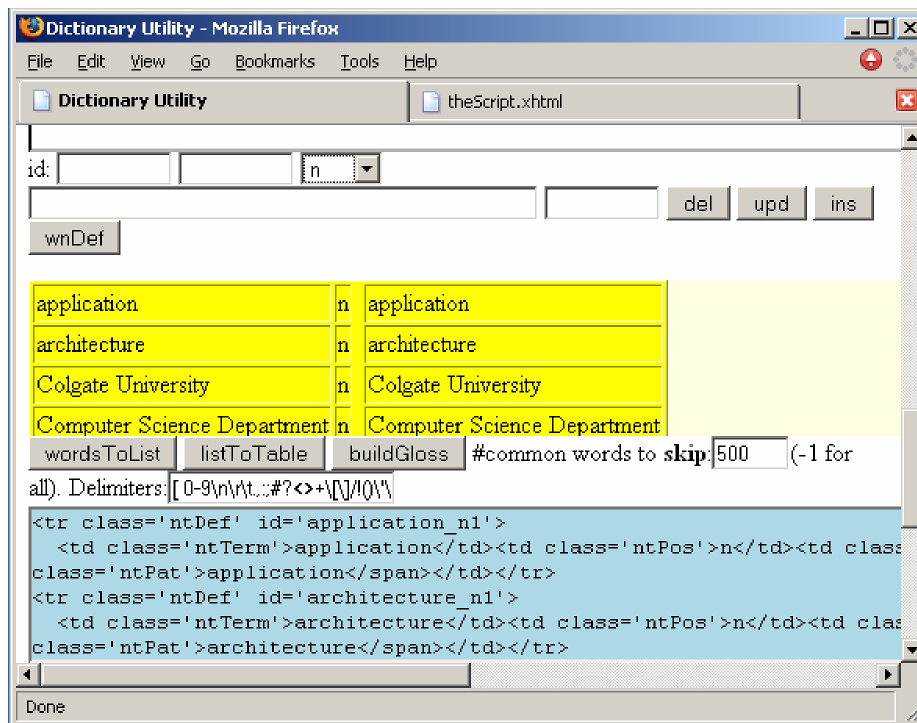


Fig. 3. The alpha utility: creating the glossary

Recall that each row in the table has the "ntDef" class attribute and an XHTML id attribute yet to be filled. The cells of the row contain the required subparts of the ntDef class: an ntTerm which is the term itself, an ntPos which is the part of speech (defaults to noun), and an ntDesc which is the definition. The last cell is an ntPatList; it will contain the patterns to be used in identifying occurrences of this term in the script. In addition, each row has an onclick method, inherited from the XhtmlDecoder class. Clicking on a row results in sets of actions (see Figure 4). First, the one-row form for entry components is filled in with data: the Part of Speech is n (the default,

accurate in this case, but changeable from a drop-down list); the id gets content_n1; the term is content; and the pattern is the same as the term. Second, the i-frame above opens the WordNet website with the definition of “content” in it. At this point the user can delete the entry, or change the data as needed, and copy and paste the definition from WordNet. When finished with the entry, click the upd(ate) button and proceed to the next entry. At this point, the textarea containing the XHTML source of the emerging glossary is also updated and can be used to save our work for later continuation.

Dictionary Utility | theScript.xml

WordNet Search - 2.1

Return to WordNet Home

Glossary - Help

SEARCH DISPLAY OPTIONS: (Select option to change) Change

Enter a word to search for: content Search WordNet

KEY: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Noun

- S: (n) content#1 (everything that is included in a collection) "he emptied the conten were similar in content"
- S: (n) message#2, content#2, subject matter#1, substance#6 (what a communicatior
- S: (n) content#3 (the proportion of a substance that is contained in a mixture or alloy
- S: (n) capacity#3, content#4 (the amount that can be contained) "the gas tank has

id: content_n1 content n

upd ins wnDef

architecture	n	architecture
Colgate University	n	Colgate University
Computer Science Department	n	Computer Science Department
content	n	content

wordsToList listToTable buildGloss #common words to **skip**: 500 (-1 for all).

Fig. 4. The alpha utility: using WordNet

When the glossary is thus prepared, we proceed to the next stage of processing, which is creating the triples table. This can be semi-automated in the same point-and-click way: select a relationship from a drop-down list, then click on two id attributes

in the glossary to select the first and third elements of the triple. In addition to a drop-down list of pre-selected relationships, they can also come from the glossary itself: the verb open_v1 can be selected as relationship between Agent and Object. For the searches described in this paper, we only need the isA and subClass relationships, whcih we retrieve from the WordNet hierarchy.

4.1 Marking up the Script

The next stage is to add markup to the script linking text words to glossary items. To be fully automated, this task would require a complete syntactic and morphological analysis of the text. We do a semi-automated version based on regular expressions, with post-editing. Specifically, given a pattern string and a lexical item (both retrieved from the glossary), we find all segments that contain a word that matches the pattern, and we insert a no-text invisible <a> link into those segments. The matching is case-insensitive prefix matching, which works pretty well for small English vocabularies.

To process the text, we copy and paste its XHTML source into the next and last text area of the alpha.xhtml page (with green background). Clicking on the “split” button results in <a> elements insterted next to lexical items that match glossary patterns, as shown in Figure 5, where the word “Computer” is now followed by the this code:

```
<a target='theGlossary' href='theDict.html#computer_n1'></a>
```

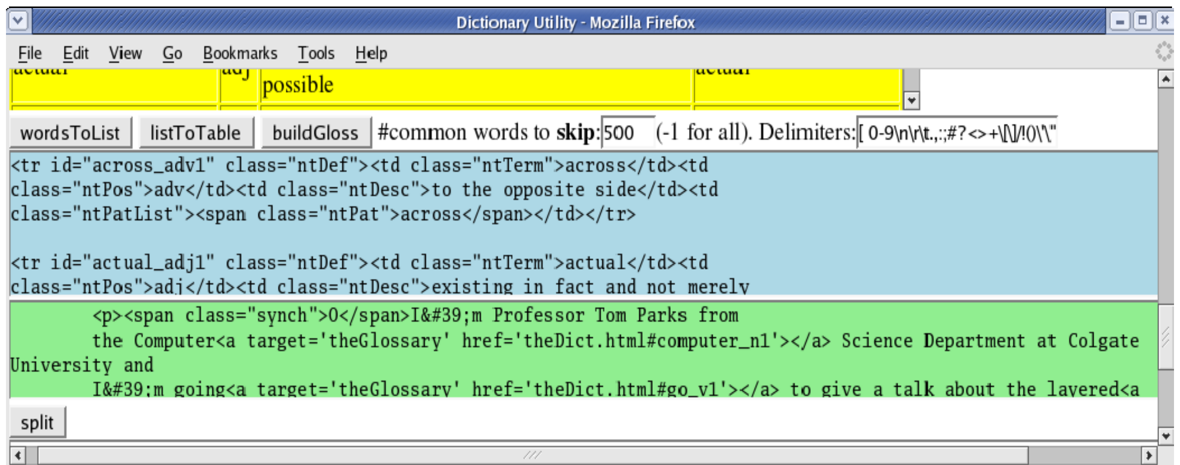


Fig. 5. Text-Glossary markup

This step also requires manual post-editing. Once it is complete, the search through the text that makes use of the relationships triples is enabled. The search utility will

cycle through the segments of the text looking for those that contain either the search term or a term that is an ancestor of the search terms in the WordNet hierarchy.

5. Conclusions and Future Plans

We have described the extension of the MannX video framework to include lightweight semantic inference. Our technique is based on XHTML markup described and validated within a pair of microformats, based on a microformat implementation that automatically associates a Javascript object with each implemented CSS class. One microformat is a simplification of WordNet's definition structure. The other is a version of textual triples representing assertions of the form R(x,y), based on the N3 format described in http://www.w3.org/2000/10/swap/Primer. Together, they form a simplified ontology. The resulting system is able to select and play video segments by using the properties of the entities referenced in the corresponding script segments.

In the immediate future, we expect to do some work on improving the user interface by which the annotator (usually a student who is not a programmer) creates the additional markup; this will include XSLT processing of WordNet's hierarchies. After that, we want to export our ontologies into the corresponding OWL ontologies. MannX videos are run by Java applets, and we can put Jena code into those applets to make larger ontologies and more sophisticated queries possible.

References

1. Dubinko, Micah, "What are microformats" XML-Deviant column. http://www.xml.com/pub/a/2005/03/23/deviant.html (March 2005)
2. Miller, George et al.."5 papers on WordNet," ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps" (1993)
3. Miller et al, "Nouns in WordNet: A lexical inheritance system." in Miller et al. (1993) 10-20.
4. Nakhimovsky, Alexander. "A Multimedia Authoring Tool for Language Instruction: Interactions of Pedagogy and Design". Journal of Educational Computing Research 17-3 (1997) 261-274
5. Nakhimovsky, Alexander, Tom Myers: JavaScript Objects.Wrox, Birmingham, UK (1998)
6. Nakhimovsky, Alexander, Tom Myers "Digital Video Annotations for Education." Proceddings of the Annual International Conference on Engineering Education. Valencia (2003)