such items called a list does not exist. However, some more advanced systems do recognize a very small amount of hierarchy (perhaps 'sequentiality' would be a better word): Microsoft *Word* has named styles for 'continuation items'.

It is on hooks like this that *DynaTag* can hang its markup. You map input styles to output elements, either named or based on combinations of styles represented in the conversion. Style information can be applied to the derived markup, which can then be saved as a separate stylesheet (see Figure 104).

The use of an intermediate format to enable accurate mapping makes this one of the most powerful tagging applications available. The graphical interface only runs under MS-Windows, but the mappings can be saved and used in the Unix batch product, which will perform command-line driven conversions for the same document type pairs (input of a word processor file in that format and output according to the specified DTD).

5.3.3. SGML Author for Word (Microsoft)

Version 1.1 for MS-Windows
`http://www.microsoft.com/`

Despite its name, this is a conversion tool, not an editor or authoring program. It's a plug-in for Microsoft *Word* (6 or 7) which enables *Word* documents to be saved as SGML files or loaded from SGML files, using a mapping between DTD and stylesheet.

The objective is to allow *Word* users to continue using *Word* without knowing anything about SGML (possibly not even that it exists), but enable their files to be used in SGML systems; or to take SGML files and produce *Word* versions which can be re-imported into SGML afterwards.

This will obviously only work if the users can be trained to stick rigidly to a predefined set of styles and never depart from them. Provided this is done, however, it is possible to use the context of the styles to map to a DTD and *vice versa*, and thus to provide translation in both directions.

Once this is established, it would be possible to maintain your corporate or institutional information base in SGML, but have authors create and update the texts using *Word*.

5.3.3.1. Setting up

Setting it up may cause some problems: support is difficult to get, even in the USA, as Microsoft's own HelpDesk staff are largely unaware of the program's existence. It was apparently not intended to be used (or even available) outside the USA, although a number of non-US institutions are now reported to be using it. If you have access to Microsoft's *Select* installation CD-ROMs, especially outside the USA, install it from them in preference to the retail package.

If you have to install from the retail package, make sure that once you have *Word* 7 installed, go to the **Tools**|***Options*** menu item, click on the **File Locations** tab and check that both |***User Templates*** and |***Workgroup Templates*** have file locations associated with them (I am indebted to Brian Widman for this tip). Then start installing *SGML Author for Word*. If it claims you don't have *Word* installed, you may need to manually edit your Registry to provide the 'right' entries: unfortunately it provides no clues as to what it needs, so you may want to call their HelpDesk at this stage.

Once it's installed, it adds |***Save as SGML*** to the **File** menu and creates some new directories in your *Word* installation folder. There is no |***Open as SGML*** because all files opened are native *Word* files. If you have installed (or are going to install) Microstar's *Near&Far Author for Word*, which *is* an editor, you should familiarize yourself with the distinction between the menu items which *SGML Author for Word* adds to the **Save** dialog and those which Microstar's software adds (the 'Save As...' file type |***SGML - Near&Far Author***).

The manual is reasonably clear, and makes a good effort at explaining what you need to know at each stage, but it aimed at the administrator, not the end user, so it does require a significant level of understanding of text and text-handling, well above the level needed for general office word processing. This is probably acceptable, given that the apparent intention is to shield the end user from the SGML bits. One thing it does make very clear is why this is just a conversion engine: why write a new editor when you have a working one already?

The mechanics of setting up a conversion are fairly intricately explained, although if you already know SGML, and have worked with stylesheets before, a lot of it begins to look familiar after a while:

- You need to have a DTD, of course, and the SGML Declaration if needed;
- There's a catalog file provided, `entm.cat`, in standard format but with the `PATH` keyword on the first line giving the directory paths to search for relative file (System identifier) references. Your DTD file(s) need to be entered in this catalog;
- You create a 'declaration file' with the file type `.dcl` for each DTD you use, in which goes a copy of the SGML Declaration followed by the DocType Declaration — but not the DTD itself.

- You bind the element names from the DTD to stylesheet styles in a `.map` file, which you create using menus in *SGML Author for Word*.

The `.dcl` file type is an unfortunate choice, as it is already used by many DTDs for their own SGML Declaration, which will probably *not* contain a DocType Declaration at the end, so some renaming or careful directory selection is needed. If you are using a technical DTD for math, and you want conversion to or from the *Word* equation editor, you need to modify the DTD to include the equation fragment provided (based on the ISO TR 9573 fragment: see section 2.3.8.1.1). You get a warning if this is not done: there's a sample `skeleton.dtd` supplied which serves as an example of how to use it.

5.3.3.2. Operation

Converting from SGML to *Word* or *vice versa* means assigning a template file to your DTD, which is done from the **File**|***New*** menu the first time round by picking a template and a `.dcl` file. If you don't have any templates of your own (this would apply if you don't use *Word* and are perhaps preparing conversions for people who do), you can pick one of the standard *Word* ones in `\MSOffice\Templates\*`, which is the proper place (unfortunately it's not where the **File**|***New*** menu looks for them). Clicking on the `.dcl` file which you created for your DTD starts the parsing of the DTD and reads the template file. Any parser errors at this stage pop up in an edit window, and have to be corrected before you can proceed.

The catalog file provided unfortunately fails to include the ISO Box and Line Drawing character entity file, the Russian Cyrillic, Non-Russian Cyrillic, and Alternative Greek Symbols ones: you may not use them but some DTDs reference them and won't validate without them unless you disable the references or add the files to your file system and include the relevant lines in the catalog.

By far the most lengthy part is setting up the associations between *Word*'s stylesheet descriptors and the elements of your DTD. The display gives both side-by-side (see Figure 105) so the mechanics of it are fairly simple. However, the DTD elements are displayed in a form which makes available every possible variant of element combinations that is possible, and by default expects you to establish a style for all of them. You just ignore the ones you don't want, but it clutters the panel somewhat.

It is logically not possible to associate a style descriptor with more than one element or descendancy, because to do so would defeat the objective of providing a consistent mapping from stylesheet to DTD. This means that associating `<para>` with paragraph-style **Normal**, for

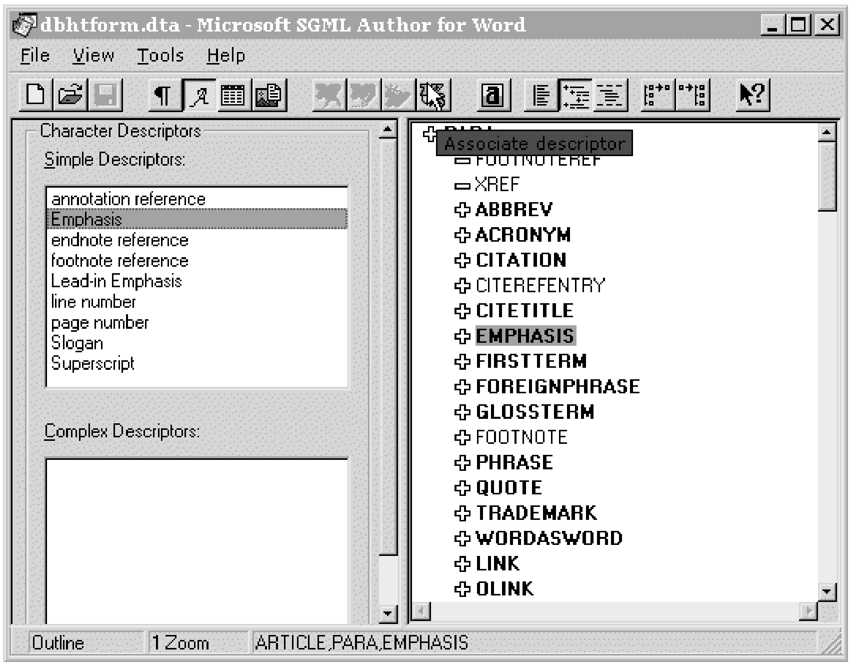

Figure 105. Associating an inline element with a character style descriptor in *SGML Author for Word*

example, makes it impossible to associate any other element with the Normal style, and if you try, it will keep trying to tell you that what you're setting up is a paragraph. You can, however, specify a default structural markup conversion (referred to as 'not in Mixed Content', which defaults to paragraph-style Normal) and a default inline markup conversion (referred to as 'in Mixed Content', which defaults to character-style annotation reference).

You can supply associations for inclusion elements, and for specific attribute values on elements. For *Word* conversion into SGML, you can establish default values for required attributes.

Because of the complexity of the setup, a detailed study of typical documents for the given DTD is essential before you start. It would also be useful to generate a list of the elements actually used, with their frequency of use. This will make it much easier to see where you need to concentrate your association efforts.

Once it's done, you save the association (`.dta`) file. This will take a few minutes, as the files take a frightening amount of disk space: in tests, an association file for the *DocBook* DTD took 8.5Mb even with version 1.1 (which the help documentation claims has been optimized for use with a 32–bit operating system: 'these performance improvements include an increase in speed, and a decrease in the size of the association file').

Finally, you can open a `.sgm` file in *Word*, pick the `.dta` file, and let it convert. It's fairly slow, and needs a lot of disk space: conversion of a 66Kb instance needed around 40Mb of temporary disk. Provided you have taken care in setting up the associations, the results are very good, and for bulk conversion the time and effort spent should be worth it.

The conversion process uses a considerable amount of smarts to achieve what the documentation describes as a 'least-cost' path to the conversion of unspecified or unspecifiable element combinations. This probably accounts for the slow speed by comparison with *Balise*, *Omnimark*, or *SGMLC*. The manual emphasizes that you test any conversion setup by using the standard 'round-trip' methodology. This means converting circularly back into the format the document was originally in, and then studying the results to see what got lost in the process.

5.3.4. **Roustabout** (Apropos)

Mac, PC/Win
`http://www.attd.com/`

Roustabout is a specialist program from Apropos Toy and Tool Development for converting files *QuarkXPress* export files into SGML.

QuarkXPress is one of the most popular and extensible page layout systems for Apple Macs and Windows 95 PCs. Its native file format is a binary 'end-system' for internal use, but the system provides a plaintext export format called Xpress Tags or 'Xtag' format, and it is this format that *Roustabout* translates. *Roustabout* is written in *Java* and can read Xtag files produced for both Mac and PC versions of *QuarkXPress*, with or without stylesheet formatting.

Roustabout uses a name mapping file to specify the translation between the *QuarkXPress* stylesheet names and the DTD element names (see Figure 106). Because *QuarkXPress* is restricted to the simple two-level model of paragraph styles containing character styles (see section 3.2), it is not possible to map structures any more deeply unless you use an SGML editor to post-process the output. The mapping specification files are themselves in SGML format, and the system can also produce XML with a suitable DTD.

The *Roustabout* program comes as a zipfile of some 250 classes: there is a demo version on the CD-ROM, limited to the first 5,000 characters of input.