Genre and Multimodality (GeM):
a computer model of genre and
document layout

GeM Annotation Manual

Version 2.0

GeM project report 2003/04

authors: Renate Henschel

email: rhenschel@uni-bremen.de

website: www.purl.org/net/gem

Funded by
ESRC

University
of Bremen

University
of Stirling

1 Introduction

The GeM project assumes that language, layout, image and typography are all purposive forms of communication, and aims to analyze all these elements within one common framework. The focus of the project is to extricate how the use of the two modes – the visual and the verbal – varies across different genres (cf. Delin, Bateman & Allen 2002). The GeM annotation therefore considers these two communication modes as the main perspectives that must be captured in its annotation scheme. It identifies textual elements (verbal mode) and layout elements (visual mode) in a multi-layered annotation, and specifies how these elements are grouped into a hierarchical structure (the rhetorical structure for textual elements, the layout structure formed by the layout elements). The alignment between these two intersecting hierarchies is achieved by specification of the 'GeM base' – a list of the basic units out of which the document is constructed. In accordance with the goal of the project, the granularity of the linguistic basic units employed in the annotation is approximately the sentence level. Each layer is represented formally as a structured XML (Extensible Markup Language) specification (Consortium 2000), whose precise informational content and form is in turn defined by an appropriate Document Type Description (DTD).

The markup for one document consists generally of the following four layers:

Name	**content**	**dtd**
GeM base	base units	gem-base.dtd
RST base	rhetorical structure	gem-rst.dtd
Layout base	layout properties and structure	gem-layout.dtd
Navigation base	navigation elements	gem-nav.dtd

They will be described in detail below. The formal document type definitions for each of the four data files are given at `www.purl.org/gem`.

2 Base level annotation

2.1 Basic constituents

The purpose of the base level annotation is to identify the minimal elements which can serve as the common denominator for textual elements as well as for layout elements. Where speech-oriented corpora use the time line as basic reference method, and syntactically oriented corpora use the sequence of characters or words, the GeM annotation operates at a less delicate level and uses bigger chunks (mostly sentences and graphical page elements) as the base of the markup. Everything which can be seen on each page of the document has to be included. How the material on each page is broken up into basic units is given by the following list. We mark as base unit:

- orthographic sentences
- sentence fragments initiating a list
- headings, titles, headlines
- photos, drawings, diagrams, figures (without caption)
- captions of photos, drawings, diagrams, tables

- icons
- tables cells
- list headers
- list items
- list labels (itemizers)
- menu items in an interactive pop up menu
- page numbers
- footnotes (without footnote label)
- footnote labels
- running heads
- horizontal or vertical lines which function as delimiters between columns or rows
- lines, arrows, polylines which connect other base units

Sentences divided by a page or column break should be marked as two base units. The caption of figures, tables etc. is always marked as an extra base unit. Horizontal or vertical lines are marked as separate base units if they cannot be viewed as part of another layout element. Examples for lines to be marked are vertical lines which serve to separate columns in newspapers, horizontal lines which serve to separate paragraphs, whereas lines appearing at the top and the bottom of a figure are seen as the figure's border and are not marked as base units.

The base annotation has a flat structure, i.e. it consists of a list of base units. This list is comprehensive, i.e. it comprises everything which can be seen on the page/pages of the document.

The tag used to mark base units is the <**unit**>. Each base unit has the attribute **id**, which carries an identifying symbol. If the base unit consists of text, the start and end of this text is marked by the <**unit**> tag. Illustrations, however, are not copied into the GeM base. Thus, base units which represent an illustration or another graphical page element are empty XML-elements. They can optionally be equipped with an **scr** and/or an **alt** attribute however. The value of src is the source location of the illustration, if there is one available. The value of alt gives a name to the graphical element which reminds the user of its content, or – for HTML documents – the alt value may be taken directly from the source file.

The following examples illustrate how the annotation looks for different elements of a document.

Sequence of sentences in a text:

```
<unit id="u-21.07">Huge (90cm) unmistakable seabird.</unit>
<unit id="u-21.08">Watch for white, cigar-shaped body and long
                   straight, slender, black-tipped wings.</unit>
<unit id="u-21.09">In summer, yellow head of adult inconspicuous.</unit>
<unit id="u-21.10">Plunges spectacularly for fish.</unit>
<unit id="u-21.11">Sexes similar.</unit>
```

Illustration:

```
<unit id="u-21.06" alt="gannet-photo"/>
```

Horizontal line:

```
<unit id="u-21.05">----------------------------------------</unit>
```

2.2 Embedded base units

In certain cases, we diverge from the flat structure of the base file, and allow nested markup, i.e., base units inside base units. This is envisaged for the following situations:

- emphasized text portions in a sentence/heading
- icons or similar pictorial signs in a sentence
- text pieces in a diagram or picture
- arrows and other graphical signs in a diagram or picture
- document deictic expressions occurring in a sentence

Generally any text portion which is differentiated from its environment by its layout (e.g. typographically, background, border) should be marked as a base unit. Whether it constitutes a separate unit in the main level of the base unit list, or an embedded unit inside another base unit depends at the extent to which it can be moved around the page without disrupting the content.

In headings, the chapter/section numbering part should be marked as an embedded base unit. Below we give some examples:

- Emphasized text:

  Adult has *white plumage* with, in breeding season, *faint yellow-pink tinge*; usually looks pure white at distance.

  ```
  <unit id="u-21.06" Adult has <unit id="u-21.06.1">white plumage</unit>
     with, in breeding season, <unit id="u-21.06.2">faint yellow-pink
     tinge;</unit> usually looks pure white at distance.</unit>
  ```

- Headings:

  4.3.2 Modal Adjuncts

  ```
  <unit id="u-32.1">
     <unit id="u-32.1.1">4.3.2</unit>
     Modal Adjuncts
  </unit>
  ```

- Document deictic expressions (see below and: Paraboni & van Deemter 2002):

  If you still don't hear Dial tone, try the registration procedure on page 12.

  ```
  <unit id="u-7.18">If you still don't hear Dial tone, try
        <unit id="u-7.18.1">the registration procedure</unit> on
        <unit id="u-7.18.2">page 12</unit>.
  </unit>
  ```

Before you can use handsets, you will need to fit battery packs and fully charge the batteries, as described on page 5 and 6.

```
<unit id="u-2.17">Before you can use handsets, you will need to fit
      battery packs and fully charge the batteries, as described on
      <unit id="u-2.17.1">page 5 and 6.</unit>
```

As shown in the examples, there are instances of document deictic expressions where two embedded base units have to be marked – one for the content (e.g. *the registration procedure*) and one for the address (e.g. *page 12*), and cases with only one embedded base unit, which represents either the content or the address (e.g. *page 5 and 6*).

The general principle for base unit annotation is that everything on the page is marked. However, in documents where identical text pieces or pictures are used repeatedly on different pages it is possible (but not necessary) to represent these only once in the GeM base. This is, for instance, often the case with running heads in paper books, or with navigation menus on web pages. But each actual existing instance of such a shared base unit requires its own representation as layout unit in the layout base.

The purpose of the GeM base is only to **identify** the base units . Every unit can be viewed as a layout object on the one hand or as a sign carrying semantic meaning on the other. We strictly separate these two perspectives in the annotation. The **layout base** specifies layout units (sets of base units) and assigns layout properties to them. From the semantic perspective, we have (i) base units which contribute directly to the content of the document, these are the RST segments, (ii) base units which only serve to help the reader navigate through the document, the navigation elements, and (iii) units which are both. The **rst base** determines which base units (or groups of base units) serve as segments for a rhetorical structure analysis of the document and represents such an analysis. The **navigation base** lists the navigation elements and their function. The distribution of the different kinds of elements is shown in Figure 1.

3 Layout base

The layout base consists of three main parts: (a) layout segmentation – identification of the minimal layout units, (b) realization information – typographical and other layout properties of the basic layout units, and (c) the layout structure information – the grouping of the layout units into more complex layout entities. We explain these three components in detail below.

3.1 Layout segmentation – Identification of the layout units

In typography, the minimal layout element (in text) is the glyph. In GeM, however, we are primarily concerned with typographical and formatting effects at a more global level for a page; therefore we do not go into such detail, and instead consider the paragraph as minimal layout element. This means, any sequence of sentences with the same typographical characteristica which makes up one paragraph is marked as one layout unit. In addition to this we mark all graphically realized elements from the GeM base as layout units. Also

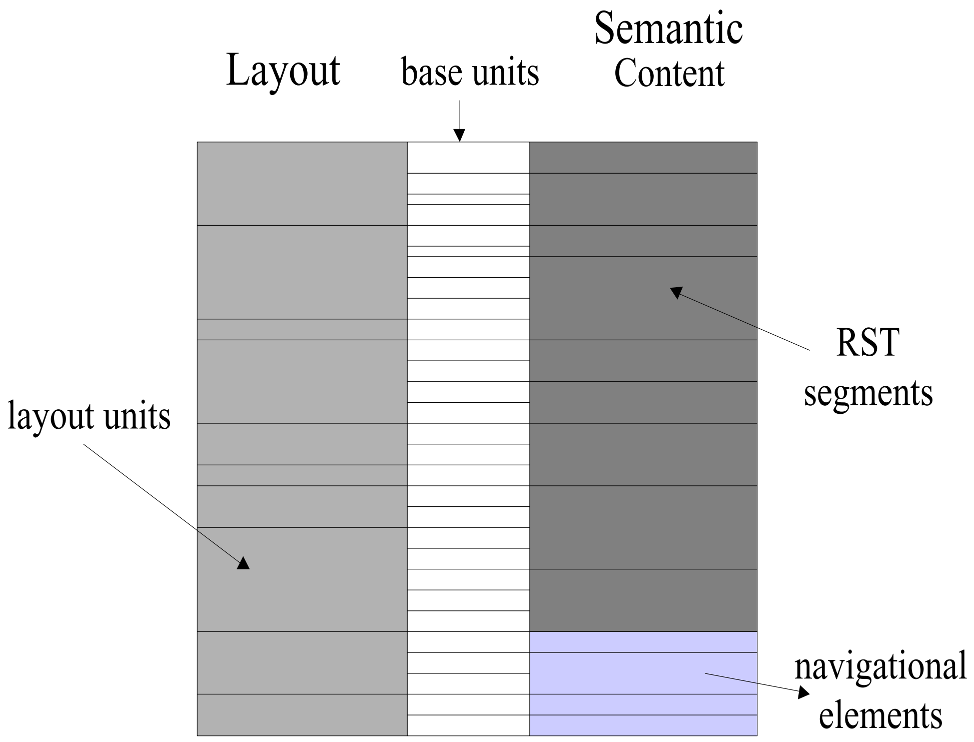

Figure 1: The distribution of base elements to layout, rhetorical and navigational elements

highlighted text pieces in sentences, or text pieces within illustrations, are marked as layout units. Is is both possible – to represent embedded base units in the layout segmentation as embedded layout units as well or to move them to the main level of layout unit segmentation. Hence the same list which has been given for the markup of the base units applies here, but with paragraphs instead of orthographic sentences.

- paragraphs
- headings, titles, headlines
- photos, drawings, diagrams, figures (without caption)
- captions of photos, drawings, diagrams, tables
- text in photos, drawings, diagrams
- icons
- tables cells
- sentence fragments initiating a list
- list items
- list labels
- items in a menu
- page numbers
- footnotes (without footnote label)
- footnote label
- running heads
- emphasized text

- horizontal or vertical lines which function as delimiter between columns or rows
- lines, arrows, polylines which connect other base units

The tag for a layout unit is <**layout-unit**>. Each layout-unit has the attribute **id**, which carries an identifying symbol, and the attribute **xref** which points to the base units which belong to this layout unit. It is possible, but not necessary, to store the corresponding text portions of the original text file between the start and end tag of a layout-unit:

```
<layout-unit id="flegg-text" xref="u-21.07 u-21.08 u-21.09 u-21.10 u-21.11">
   Huge (90cm) unmistakable seabird. Watch for white,
   cigar-shaped body and long straight, slender, black-tipped
   wings. In summer, yellow head of adult inconspicuous.
   Plunges spectacularly for fish. Sexes similar.
</layout-unit>
```

3.2 Realization information

The second part of the layout base is the realization. Each layout unit specified in the layout segmentation has a visual realization. The most apparent difference is which mode has been used – the verbal or the graphical mode. Following this distinction, the layout base differentiates between two kinds of elements: textual elements and graphical elements marked with the tags <**text**> and <**graphics**> respectively. These two elements have different sets of attributes describing their layout properties. For textual elements, the following typographical attributes are annotated:[1]

xref	*ids of the layout units which are realized with the following values*
font-family	*family-name* | inherit
font-size	*length* | inherit
font-style	normal | italic | oblique | backslant | inherit
font-weight	normal | bold | extra-bold | light | inherit
case	caps | mixed | smalls | smallcaps | inherit
color	*color* | inherit
justification*	left | right | justified
border*	none | rectangular | circular | lined | underlined | overlined | vertical-lined | speech-bubble)
background-color*	*color* | inherit

We usually mark the font size in points; so font-size="8" would mean 8 point. For web pages, however, we use the sizes 1 to 6, which are normally used in HTML documents.

Justification has to be marked only on textual elements which form more than one line. justification="left" is used for right-ragged text, justification="right" for left-ragged text, justification="justified" for text which is justified at both margins.

[1]Attributes with a star are optional.

Embedded textual base units which are typographically highlighted against their environment are marked with the tag <**hi-text**> and are annotated with exactly the same attributes as ordinary <text> elements, but have an extra **context** attribute, which refers to the id of the embedding <layout-unit>. Those properties of the <hi-text> element which make it stand out against its context are annotated with values, those properties which are shared with the context receive the value "inherit".

For graphical elements, the following attributes are marked:

xref	*ids of the layout units which are realized with the following values*
type	illustration | photo | diagram | two-d-element
color-no*	*number of used colors*
colors	*list of used colors* | color | inherit
border*	none | rectangular | circular | lined | underlined | overlined | vertical-lined | speech-bubble

The value "two-d-element" (two-dimensional graphical element) is used for lines, arrows, icons, etc. If the value 'two-d-element' has been chosen as the type of some graphical element, then three additional attributes have to be annotated on that element:

two-d-element-type	line | polyline | rect-polyline | arrow | bi-arrow | arc | square-bracket | brace | square | triangle | diamond | icon
element-style	inherit | solid | dashed | dotted | double
element-weight	normal | bold | extra-bold | light | *width* | inherit

Two-d graphical elements of type "arrow" are marked with the extra attribute **direction**:

direction	up | left | right | down | vertical | horizontal | diverse

Lines can optionally be marked with:

length	*length*

Length values without measure units are interpreted relatively as a percentage of the dimensions of the surrounding area. Absolute values have to be given with the measuring unit.

The **border** attribute is optional in textual and graphical elements. If the actual element has in fact a border, then we also give the following additional information about the layout properties of that border:

border-color	*color* | inherit
border-style	hidden | dotted | dashed | solid | double | shadow
border-weight	normal | light | bold | extra-bold | *width* | inherit

Different layout units with an identical typographical realization can be represented with only one <text> or one <graphics> element. Every text and graphics element has an **xref** attribute, under which the ids of the layout units which share the typographical properties given in this text/graphics element are stored. The `id` of each layout unit of the segmentation part of the layout base has to occur exactly once under `xref` in the realization part: either in a <text> or a <hi-text> or a <graphics> element. In the following coding example, we have five layout units which share their typographical characteristica. These correspond to the five table cells in the first column of a table.

```
 <text xref="lay-21.12 lay-21.14 lay-21.16 lay-21.18 lay-21.20"
       font-family="sans-serif"
       font-size="10" font-style="normal" font-weight="bold"
       case="mixed" justification="right" color="black"/>
```

In general, the layout annotation does not attempt to give always the precise numbers or names as their attribute values, particularly for the typographical attributes font-family and font-size. All attributes which allow arbitrary alphanumeric values (CDATA in the document type definition) can be annotated with coarse values or names which are used document-internally in a consistent manner, i.e. if the typographical layout is identical between different layout elements, the annotation should reflect this by choosing identical attribute values.

The following example shows the annotation of a highlighted text piece:

The IN USE/CHARGE light comes on when the handset is correctly positioned in the charging cradle.

```
<hi-text id="lay-6.09.1" xref="u-6.09.1" context="lay-6.09"
         font-family="inherit" font-size="inherit" font-style="inherit"
         font-weight="inherit" case="caps" color="black">
   IN USE/CHARGE
</hi-text>
```

And the last examples are realization XML elements for two graphical layout units, a diagram and a vertical line:

```
<graphics xref="lay-2.03a" type="diagram" color-no="2"
          colors="black white" border="none"/>

<graphics xref="lay-line-2.1" type="two-d-element"
          two-d-element-type="line" colors="black"
          border="none" element-style="solid"
          element-weight="extra-bold"/>
```

3.3 Layout structure

Similar to the RST structure which groups sentential text segments into larger text spans, some of the layout units identified in the segmentation part of the layout base can be grouped into larger layout chunks. For instance, the heading and its belonging text form together a larger layout unit, or the cells of a table form the larger layout unit “table”. The criterion for grouping layout elements into chunks is that the chunk should consist of elements of the same visual realization (font-family, font-size, ...), or the chunk is differentiated as a whole from its environment *visually* (e.g. by background colour or a surrounding box). In Reichenberger, Rondhuis, Kleinz & Bateman (1995), the authors propose identifying layout chunks by applying a decreasing display resolution to the document. The grouping into chunks usually can be applied in several steps, thus forming larger and larger layout chunks out of

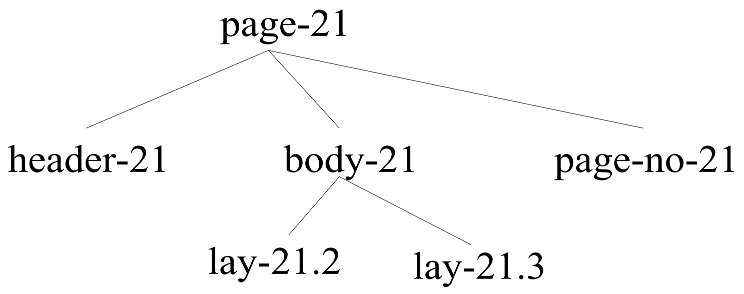

Figure 2: Example page layout

the basic layout units up to the entire document. Note that one chunk can consist of layout elements of different realizations (text and graphics).

The third part of the layout base then serves to represent this hierarchical layout structure. Generally we assume that the layout structure of a document is tree-like with the entire document being the root. Each layout chunk is a node in the tree, and the basic layout units, which have been identified in the segmentation part of the layout base, are the terminal nodes of that tree. In our annotation, we use five different tags for the nodes in the layout tree. <**layout-root**> is the element describing the entire document, <**layout-chunk**> are all non-terminal nodes in the layout tree except for the root, and <**layout-leaf**> represents the terminal nodes. Non-terminal nodes in the layout structure tree which form an ordered or unordered list are marked with the special tag <**list**> instead of <layout-chunk>. The itemizers used in a list are terminal nodes and are marked with the tag <**itemizer**> instead of <layout-leaf>.

The elements <layout-chunk> and <list> have the following attributes:

id	
location	cell-11 | cell-12 | ... row-1 | row-2 |... col-1 | col-2... | multi | delimiter
area-ref	*id of an area defined in the area model*
halign*	top | center | bottom | indent
valign*	left | center | right | indent

The elements <layout-leaf> and <itemizer> have the following attributes:

xref	*id of a layout-unit*
location	cell-11 | cell-12 | ... row-1 | row-2 |... col-1 | col-2... | multi | delimiter
area-ref	*id of an area defined in the area model (see below)*
halign*	top | center | bottom | top-indent | bottom-indent
valign*	left | center | right | indent | right-indent

Note that for each list, its itemizers are represented only once as child in the layout structure, although it appears several times in the actual document and in the layout segmentation. The itemizer's **xref** attribute must then contain a set of ids. The <layout-root> has an **id** attribute only.

The layout structure is represented by hierarchical specification of the children chunks/leafs for each layout chunk. A page in a two-column scientific journal, for instance, could consist of three main layout elements – a header, a body and a page number. The body itself is formed out of two layout chunks, the text in the first column (layout-unit lay-21.2), and the text in the second column (layout-unit lay-21.3). This layout structure is shown visually in Figure 2; it is described by the XML annotation below:

```
<layout-root id="page-21">
      <layout-leaf xref="header-21"/>
      <layout-chunk id="body-21">
         <layout-leaf xref="lay-21.2"/>
         <layout-leaf xref="lay-21.3"/>
      </layout-chunk>
      <layout-leaf xref="page-no-21"/>
</layout-root>
```

However, the page or page segment layout is not fully determined by grouping layout units into a tree structure; further information is required about the actual position of each unit in the document (on or within its page). For this, we introduce the area model, which serves to determine the position of each layout-chunk/layout-leaf in an abstract way.

Area model. Each page usually partitions its space into sub-areas. For instance, a page is often designed in three rows – the area for the running head (row-1), the area for the page body (row-2), and the area for the page number (row-3) – which are arranged vertically. The page body space can itself consist of two columns arranged horizontally. These rows/columns need not be of equal size. For the present, we restrict ourselves to rectangular areas and sub-areas, and allow recursive area subdivision. The partitioning of the space of the entire document is defined in the **area-root**, which structures the document (page) into rectangular sub-areas in a table-like fashion.[2]

The tag to represent the area root is <**area-root**>, which has the following obligatory attributes:

id	
cols	*number of columns*
rows	*number of rows*
hspacing	*list of percentages* | 100 | flexible | equal
vspacing	*list of percentages* | 100 | flexible | equal

The tag to represent the division of a sub-area into smaller rectangles is <**sub-area**>, and this has the same attributes plus a **location** attribute:

id	
location	row-1 | row-2 |... col-1 | col-2 |... cell-11 | cell-12 ...
cols	*number of columns*
rows	*number of rows*
hspacing	*list of percentages* | 100 | flexible | equal
vspacing	*list of percentages* | 100 | flexible | equal

The number of columns and rows specified in an area element define a grid on the available space (the page).[3] We stipulate that the sub-areas thus defined (the rectangles in this grid)

[2]Note that the area-root need not be a page; it is the entire book or brochure, if the document to be annotated is a book or brochure.

[3]In case of a book, the cols-value is the number of double pages.

have generic names: **cell-11** for the area of the first cell in the first row, **cell-12** for area of the second cell in the first row and so on. These names then serve as values for the **location** attribute in the definition of sub-areas. If the space is divided into columns only (no rows), we mark `rows=‘‘1’’`. For rows only, we use `cols=‘‘1’’` analogously. We use the names **row-1, row-2**, ... and **col-1, col-2**, ... as location values for entire rows or columns in documents with `cols=‘‘1’’` or `rows=‘‘1’’`. Note that in the area model, the location value of a <sub-area> always has to be understood with respect to its parent's grid structure given with the cols and rows numbers of this parent.

The parent area occupies a certain space, and the number of columns and rows specifies how this space is topologically partitioned into sub-areas. The two spacing attributes specify the size of each sub-area as a percentage of the whole area. **vspacing** gives the partition of the height of the parent area into the heights of its constituting rows; **hspacing** gives the partition of the width of the parent area into the widths of its constituting columns. If the columns number is "1" (the area is divided into rows only), then `hspacing` has to be specified as "100"; if the rows number equals "1", then `vspacing` has to be specified as "100".

In the above page example, the distribution of the page height to its rows – the running head, the page body, and the page number – would be something like `vspacing=‘‘10 85 5’’`; this means that the running head takes 10% of the entire page height, the page body 85% and the page number 5%. The page body consisting of two columns would have a hspacing of `hspacing=‘‘50 50’’` with the meaning that both columns are equal in width and take half of the page's width.[4] Our example's area model would then consist of a specification of the area-root (called "page-frame"), and the specification of one particular sub-area located in row-2 (called "body-frame"):

```
<area-root id="page-frame" cols="1" rows="3" hspacing="100"
           vspacing="10 85 5" height="16cm" width="14cm">
   <sub-area id="body-frame" location="row-2" cols="2"
             rows="1" hspacing="50 50"  vspacing="100"/>
</area-root>
```

This area model is visualized in Figure 3.

In many documents spacing is not predetermined, but comes out as a result of the sequential arrangement of the layout children. Each child gets exactly as much space as it needs, and after that the next child is printed. A typical example for this is the arrangement of paragraphs in a text. Paragraphs form rows in their parent area. The vertical space (the height) for each paragraph is not a feature designed by the page designer, but depends mainly on the length of each paragraph, the type-size and the width of the parent chunk. We mark this kind of spacing with the value **flexible**.[5] Another non-numerical value for the spacing attributes is **equal**. In this kind of partitioning, the space is equally split up between the sub-areas. Equal spacing is often used in tables or lists. Also our above two-columns structure could have a hspacing="equal" markup instead of `hspacing=‘‘50 50’’`.

[4]For the time being, we ignore space for margins, at least as long as they do not contain footnotes or other text.

[5]Note, however, that it is often difficult to differentiate between a fixed spacing and a flexible spacing in ready-made documents. If you are not sure, always prefer the percentage annotation!

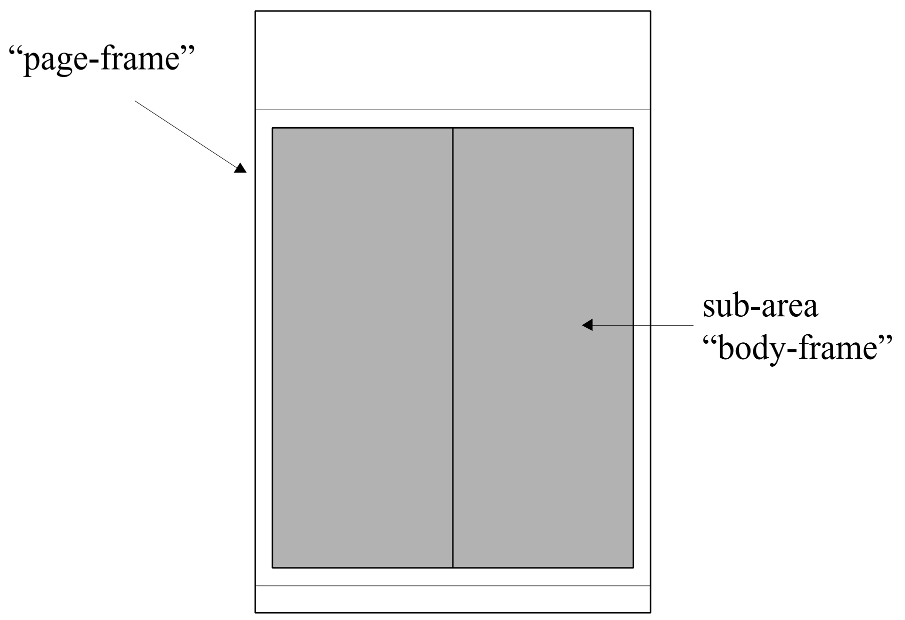

Figure 3: Visualized area model

Location. The area model above defined specifies sub-areas in a document, which we will use to allocate location values to the layout-chunks and leafs of the hierarchical layout structure. A location of a layout-chunk/leaf is sufficiently determined by saying in which cell of which area/sub-area it is located. This is realized with the attributes **location** and **area-ref**. The location value is one of the generic location values explained above, and **area-ref** refers to the id of a particular area of the area model with respect to which the location value has been chosen. We also allow the col- and the row-values in a table structure for cases where one layout-chunk/leaf occupies more than one cell of a table.

In our page example the header layout unit would be in “row-1” of the page-frame, the page body in “row-2” of the page-frame, and the page number in “row-3” of the page-frame; moreover, the text in the first column of the page-body would be in “col-1” of the body-frame, the text in the second column in “col-2” of the body-frame. The layout structure annotation is as follows:

```
<layout-root id="page-21">
   <layout-leaf xref="header-21" location="row-1" area-ref="page-frame"/>
   <layout-chunk id="body-21" location="row-2" area-ref="page-frame">
     <layout-leaf xref="lay-21.2" location="col-1" area-ref="body-frame"/>
     <layout-leaf xref="lay-21.3" location="col-2" area-ref="body-frame"/>
   <layout-leaf xref="page-no-21" location="row-3" area-ref="page-frame"/>
   </layout-chunk>
</layout-root>
```

In this example the partitioning of the page area into sub-areas is isomorphic with the hierarchical layout structure. Note that this is not always the case. A more complex relationship between area model and layout structure can be seen in the complete annotation example

flegg-page at `www.purl.org/net/gem` under `corpus`. In non-isomorphic cases it often appears that a layout chunk is not located in a single area defined by the area model, but is composed out of children layout chunks/leafs which have a precise area allocation. The location of the parent chunk is then the 'sum' of the areas of the children; and this is marked with `location=‘‘multi’’`.

It is assumed as default that the itemizers of a list are placed at the left edge of each row of the area in which the list is located. In this case we do not specify the itemizer's location attribute. In some cases, the itemizers are placed outside the list area in the adjacent area to its left. This area (usually a cell) has then to be specified under the location attribute of the itemizer element as well as the area-ref value for this cell, and `valign="right"`.

Alignment. Layout children (chunks and leafs) vertically arranged in the cells of one and the same column are assumed to be aligned with each other with respect to their left edge. Alignment at the top edge is assumed for children arranged horizontally in one and the same row. For layout-chunks/leafs which do not fit into this alignment assumption, the optional attributes **halign** and **valign** allow the specification of other possibilities.[6] If none of the abstract values (left/right/center) apply for valign, then we mark `valign=‘‘indent’’` or `valign=‘‘right-indent’’`, and add an extra markup for the identation, e.g. `hindent=‘‘5mm’’`. This applies in a similar way to `halign` and `vindent`. Note that indentation under `valign` (values "indent", "right-indent") is horizontal and appears with `hindent`; whereas indentation under `halign` (values "top-indent", "bottom-indent") is vertical and appears with `vindent`.

In the following we discuss cases of layout structure which do not fit into the annotation scheme presented so far. Two problematic cases will be discussed:

- **Insets:** Layout elements can displace another layout element or intrude into the space of another layout element.
- **Separators:** Certain graphical elements (lines, arrows) do not always fit into a grid structure, but serve to indicate column or row separation.

Insets. One layout element – the **inset** – displaces another layout element – the **flow object**. The content of the flow object is arranged around the inset, and both elements are children of the same parent layout chunk. The inset chunk does not respect any structure which the parent chunk has introduced. It is marked with `location=‘‘inset’’`. To determine the precise location however, the height and the width of the inset as well as its alignment inside the parent chunk's space have to be specified in addition. And the attribute **displace** specifies the id of the flow-object which is displaced by the inset. The flow-object, on the other hand, refers to the inset-id under the optional attribute **flow-around**.

A slightly different situation occurs, where the inset layout element intrudes into the location space of another layout element without displacing it. The latter does not need to be a flow-object. One example for this is when text intrudes into the empty space surrounding illustrations, and the background-colour of the illustration is the same as the background colour of the text. Here both layout elements have their own location on the page. These locations are adjacent. To annotate this "overlapping", we specify the location (cell, row or

[6]Note that halign and valign are here used with an opposite semantics as in HTML.

column) whose space is used under the attribute **overlap** and the id of the area **overlap-area** to which this location belongs. Both these attributes are assigned to the layout element which intrudes into the neighbour's space.

Separators. Graphical elements of type "line" are often used as delimiters between cells, columns or rows, rather than as forming cells etc. by themselves. Graphical elements of this kind should be marked with `location=''delimiter''`. Their actual location has then to be marked with the additional attributes:

delimiter-before	cell-*xy*, row-*x* or col-*y*
delimiter-after	cell-*xy*, row-*x* or col-*y*

It is possible that one of the two attributes is not specifiable because the line indicates the beginning (only delimiter-before) or the end of a structure (only delimiter-after).

4 RST base

4.1 RST for multimodal documents

The RST base specifies the rhetorical structure of the document. The rhetorical structure is annotated following the Rhetorical Structure Theory (RST) of Mann & Thompson (1988). RST investigates the relations which hold between subsequent clauses, or bigger adjacent fragments of a text – the so-called text **spans**. An RST relation typically combines two adjacent text spans of unequal importance, the text span which is central to the writer's communicative goal, the **nucleus**, and the text span which supports the message of the nucleus, the **satellite**. Multinuclear relations between two or more spans of equal importance (e.g. list, joint, sequence, ...) are also possible. Adjacent text spans which are related by an RST relation form then together a bigger text span, which itself is the nucleus or satellite of an RST relation. So the RST relations are recursively applied on ever longer text portions, resulting in a tree structure the root of which is the whole text, and the terminal nodes are the clauses of this text.

Some characteristics of RST vary between different research traditions, especially the granularity of the segmentation, the assumed set of rhetorical relations and the branching style of the rhetorical structure tree. Original RST allows multiple branching, whereas the Marcu annotation approach (e.g., Cristea, Ide, Marcu & Tablan 2000) works with binary branching only. We commit ourselves in GeM to the following assumptions:

- We build on a sentence based segmentation.
- We use the extended relation set (see `www.sil.org/~mannb/rst/toolnote.htm`).
- We allow multiple branching for multinuclear nodes.
- We allow more than one satellite per nucleus in the case that the relation holding between each satellite and the nucleus is one and the same.

RST has been developed for traditional linear text. If one wants to apply RST to modern, often multimodal, documents, new issues arise. In semiotics and design research, relations have been proposed which hold particularly between text and image (Schriver 1997, Barthes 1977). In contrast instead of adding new text-image relations, (André 1995) parameterizes the existing RST relation set by a mode parameter. We favour this second approach. The relations proposed by Schriver and by Barthes, in most cases, can be easily reduced to traditional RST relations with one constituting partner being in the graphical mode. However, there are other problems when generalizing RST to multimodal documents, which have not been addressed previously:

- The prominence of graphics in multimodal documents makes it often difficult to decide upon nuclearity in multimodal relations.
- The linear order of the constituents of the document is lost.
- The minimal unit for RST segmentation cannot be restricted to a clause or clause-like phrase.

Nuclearity in multimodal relations. Graphical illustrations are often used to *rephrase* a text passage; but it is difficult to decide which of the two segments – the illustration or the text passage – is in fact nuclear and which is the satellite. This seems to be a particular problem of graphics-text relations. To model this problem, we use the multinuclear **restatement** relation. A similar relation can also be found in Schriver under the name **supplementary**.

Linear order. Conventional RST builds on the sequentiality of text segments. Relations are only possible (with some minor exceptions) between subsequent segments/spans (sequentiality assumption). With multimodal documents, the mutual spatial relations between the segments change (from relations in a string-like object to relations in a graph). Segments can have not only a left and a right, but also an upper and a lower neighbour segment. In general one can imagine neighbouring segments in any direction, not only the four which presuppose a rectangular-based page layout. In addition to this, there can be more than one neighbour in each direction. The simplest solution to apply RST (with its sequentiality assumption) to such a document would be to introduce a reading order on the segments of the document, which is then used as the sequence behind the RST structure. However, this can easily fail to reflect the actual reading behavior. A better, more straightforward generalization of the sequentiality assumption, which we will adopt here, is to restrict RST relations to pairs (sets) of document parts (segments/spans) which are adjacent in any direction. But again, in real documents, one can sometimes find a layout where the rhetorical structure obviously is in conflict with this adjacency condition. Our hypothesis here is that this is generally possible, but that in such a case an explicit navigational element is required so as to indicate the intimate relation between the two separated layout units.

Clause as segment. The clause usually serves as minimal unit in RST. There are also approaches, which allow prepositional phrases to be a segment on their own. This is straightforward because both approaches assume something which denotes an action, an event or a state – also called eventualities – as the basic unit. However, if we move to modern documents, particularly multimodal documents, it is questionable whether the clause/PP basis should be kept. Typical examples in multimodal documents are:

- a diagram picturing a certain object and a text label which identifies (puts a name to) this object
- a list with an initiating sentence fragment, as in:

In the box are:
◇ three cordless handsets
◇ the base unit
◇ a mains power lead with adapter
◇ a telephone line cable
◇ two charger pods

- an attribute-value table, as in:

Juvenile	Grey-brown, flecked becoming whiter, adult plumage after three years.
Nest	Mound of seaweed on bare rocky ledge.
Voice	Harsh honks and grating calls at colony.

The cited examples are all expressions of states, or of static relationships between two objects or between an object and a property such as: identification, location, possession, and predication relations. In a traditional linear text, such relations would have been expressed as *is-* and/or *has*-clauses. Each such clause would constitute **one** basic RST segment. In our examples above, however, the two constituents of such a static relation clause are broken out and printed as separate layout units—in the first example, they are even given in differing modes. It is their mutual arrangement on the page plus possible extra graphical devices that expresses the relation between them. This raises the question as to what counts as a minimal unit for an RST analysis in such documents. We have the following two possibilities:

1. We remain with the clause/PP assumption, and consider picture+label, sentence-fragment+list, attribute+value as **one** basic RST segment.
2. We allow text phrases and pictures which denote an object as minimal RST segments, and analyze the relations between them.

Possibility 1 has the disadvantage that we will not obtain deeper insights into the structure and design of multimodal documents if we consider the mentioned static relations without regard to their components.

Possibility 2 introduces an entirely new dimension to RST. It is clear that the static relations between objects and objects/properties differ qualitatively from the existing set of RST relations. But it is not only the fact that we need to introduce a few new relations (identification, predication, possession, location), we also have to face the fact that these relations do not exhibit a nuclear-satellite relationship. It is very questionable whether one should compromise the basic ideas behind RST to such an extent. Especially where the relations we are dealing with appear to correspond well to the process types already established in Systemic Functional Grammar (cf. Halliday 1985) for the analysis of clauses.

Because the GeM project is interested in the investigation of the relations between different layout elements in a document we propose a golden mean. We will ascribe an RST structure to clause-type elements only. But we will also analyse the object-object/property relations, if they are clearly separate layout units. We adopt the following five relations based on Halliday (1985), which we will collectively term 'intra-clausal relations':

Identification	identity assertion
Class-ascription	relation between an object and its superclass: isa(A,B), inst(A,B)
Property-ascription	relation between an object and its predicate: pred(A)
Possession	relation between possessor and possessed: has(A,B)
Location	relation between an object and its spatial or temporal location: loc(A,B)

In contrast to RST relations, intra-clausal relations do not hold between a nucleus and a satellite, but between two relation dependent semantic roles as shown in the following table:

Relation	**Roles**	
Identification	identified	identifier
Class-ascription	attribuend	attribute
Property-ascription	attribuend	attribute
Possession	possessor	possessed
Location	attribuend	location

4.2 RST annotation

The tag used to mark the basic RST units is <**segment**>. In order to find out which base units form segments, one has to filter out those base units which are in the document for navigational reasons only. These are, for example, page numbers, running heads, footnote labels, document deictic expressions. Headings, which we also mark as navigational elements, are nevertheless marked as segments as well. Hence the following base units are segments:

- orthographic sentences
- headings, titles, headlines
- photos, drawings, diagrams, figures (without caption), if they are not part of an identification relation
- captions of photos, drawings, diagrams, tables, if they are not part of an identification relation
- list items, if they are clauses
- footnote without footnote label

Sentences disrupted into two base units by page/column breaks will form only one segment in the RST base.

In addition to these proper RST segments, we mark as <**mini-segment**> all the base units between which an intra-clausal relation holds. Typical examples for intra-clausal relations are:

- diagram + label
- table cell$_{i,1}$ + table cell$_{i,2}$ in a two-column table
- list initiating sentence fragment + list items if the list items are NPs

Note that each partner of such an intra-clausal relation forms one mini-segment.

Base units which are **not** marked as segments are:

- embedded base units
- horizontal and vertical lines
- page numbers
- footnote labels
- document deictic expressions
- menu items in a web page navigation menu

Each segment and mini-segment has the attribute **id**, which carries an identifying symbol, and the attribute **xref**, which points to the corresponding base unit-id. The text inside the segment elements is optional and not necessary for the completeness of the annotation, although it may be useful for the annotator.

Sequence of sentences:

```
  <segment id="s-21.07" xref="u-21.07">Huge (90cm) unmistakable seabird.</segment>
  <segment id="s-21.08" xref="u-21.08">Watch for white, cigar-shaped body and long
straight, slender, black-tipped wings.</segment>
  <segment id="s-21.09" xref="u-21.09">In summer, yellow head of adult inconspicuous.
  </segment>
  <segment id="s-21.10" xref="u-21.10">Plunges spectacularly for fish.</segment>
  <segment id="s-21.11" xref="u-21.11">Sexes similar.</segment>
```

Itemized list

```
<mini-segment id="s-21.12" xref="u-21.12">Juvenile</mini-segment>
<mini-segment id="s-21.13" xref="u-21.13">Grey-brown, flecked becoming whiter,
           adult plumage after three years.</mini-segment>
<mini-segment id="s-21.16" xref="u-21.16">Nest</mini-segment>
<mini-segment id="s-21.17" xref="u-21.17">Mound of seaweed on bare rocky ledge.
</mini-segment>
```

Based on the defined segments, the RST structure is annotated as a flat list of spans. Reflecting the difference between multinuclear and mononuclear relations, we distinguish two (empty) markup elements to denote nonterminal RST spans:

The **span** with the attributes:

id	
nucleus	id *of the nucleus of this span*
satellites	*list of ids of the satellites to this nucleus*
relation	elaboration | circumstance | solutionhood | background | enablement | motivation | evidence | justify | volitional-cause | nonvolitional-cause | volitional-result | nonvolitional-result | purpose | antithesis | concession | condition | unless | otherwise | interpretation | evaluation | restatement | summary | preparation

and the **multi-span** with the attributes

id	
nuclei	*list of ids of the nuclei which form this span*
relation	joint | list | sequence | contrast | restatement

We do not make use of the common habit marking the relation between title and text body as an RST relation (preparation or summary). In order to incorporate title segments, a span or multi-span can optionally have an XML child <**title**> with the attribute **xref**. The xref value of the title element is the id of the segment which functions as title for this span. To form a valid RST base, every segment id has to appear exactly once as a nucleus or satellite or title xref value.

Intra-clausal relations are marked as <**mini-span**>. In their structure, they resemble spans, but have have different attributes instead of nucleus and satellites:

id	
attribuend	id *of the base-unit which represents the identified, predicated, possessor or located*
attribute	id *of the base-unit which represents the identifier, predicate, possessed or location*
relation	identification | class-ascription | property-ascription | possesssion | location

Spans are specified as daughter nodes of the XML-element <**rst-structure**>; mini-spans as daughter nodes of the XML-element <**mini-structure**>. <Rst-structur> has the attribute **root**, where the id of the top span of the entire structure is specified. It is possible to have more than one rst structure for one document (e.g. in newspapers with semantically unrelated articles).

The following XML code is the annotation for a rhetorical structure fragment; the constituting segments are specified in the example code above.

```
<rst-structure root="flegg-gannet">
  <span id="flegg-gannet" nucleus="s-21.33" satellites="s-21.35"
        relation="elaboration"/>
  <multi-span id="s-21.33" nuclei="s-21.06 s-21.32" relation="joint"/>
  <multi-span id="s-21.35" nuclei="s-21.13 s-21.15 s-21.17 s-21.19 s-21.21"
        relation="joint"/>
  <span id="s-21.32" nucleus="s-21.31" satellites="s-21.11"
        relation="background"/>
  <span id="s-21.31" nucleus="s-21.7" satellites="s-21.8 s-21.9 s-21.10"
        relation="justify"/>
  ...
</rst-structure>
```

Example code for intra-clausal relations is shown below:

```
<mini-structure>
   <mini-span id="s-21.24" attribuend="s-21.12" attribute="s-21.13"
              relation="property-ascription"/>
   <mini-span id="s-21.25" attribute="s-21.16" attribuend="s-21.17"
              relation="class-ascription"/>
   ...
</mini-structure>
```

This RST annotation scheme, which has been adopted in GeM, is aimed to overcome some drawbacks found in existing RST annotation approaches. The two standards common in the RST community are Daniel Marcu's and Mick O'Donnell's annotation tools (`www.isi.edu/~marcu` and `www.sil.org/~mannb/rst/micktool.htm`). In both tools, the annotated output is primarily seen as the program-internal representation of RST structures to be visualized as graphical trees with the help of the tool, but not as output to be used for further XML processing. Marcu's tool produces a highly structured LISP expression and is restricted to binary branching; it also does not associate IDs to other spans but leaf nodes. O'Donnell's tool generates a flat XML output where the structure is encoded with the help of a parent attribute. He allows for different RST relations with one and the same nucleus and several 'non-classical' RST-like structures that violate the basic RST assumptions that we adopt above. In both approaches the relation is encoded as a property of the satellite.

The more recent RAGS standard for RST representation (RAGS Project 1999) has been developed with a different aim in mind. RAGS provides an interchangeable data structure for RST structures for use in the Natural Language Generation community. Such RST structures are not ordered and do not refer to a ready text. Therefore, the RAGS standard does not include a text segmentation.

For a distributable, interchangeable form of RST annotation of texts/documents, we view the following as desirable:

- flat characterization (+O'Donnell, -Marcu, -RAGS)
- IDs for larger spans (+O'Donnell, -Marcu, +RAGS)
- valid XML output accompanied with a DTD (+O'Donnell, -Marcu, +RAGS)
- separation between segmentation and rhetorical structure (+O'Donnell, -Marcu, -RAGS)
- the rhetorical relation is a property of the entire span, not of the satellite (-O'Donnell, -Marcu, +RAGS)

The GeM approach is designed to meet all these issues. It is similar to O'Donnell's approach, but limits the allowed structure to *one* relation per nucleus. It also uses a top-down encoding, which we consider far easier to read.

5 Navigation base

Navigation in a document is performed with the help of pointers, text pieces which tell the reader where the current text, or 'document thread', is continued or which point to an alternative continuation or continuations. The addresses used by such pointers are either names of RST spans or names of layout chunks. For long-distance navigation, typical nodes in the RST structure and in the layout structure have become established for use in pointers; in particular, chapter/section headings are names for RST spans and page numbers are names for page-sized layout-chunks, which tend to be used as navigation addresses. However, there can also be other name-carrying layout-chunks or RST spans such as, for example, figures, tables, enumerated formulas, and so on. The navigation base of a document lists all these "names", which have been defined in this document to be actually or potentially used as addresses in pointers. In paper documents, these names are generally placed immediately before or after the object to be indexed in the top-down or left-right reading order. It is typographically distinguished from the indexed object.

We call the name of a layout-chunk **index**. We call the name of an RST span **entry** because it is usually placed immediately before the text of this span. Whereas indices have no other function than being used as addresses in pointers, entries have besides their address function a semantic content, which tells the reader in short what to expect from the following text. Therefore, entries can be annotated in the navigation base as entry as well as in the RST base as segment. In some cases, also layout-chunks can have a title or header. Then we mark this title/header as entry rather than as index. The differentiating criteria between index and entry is that besides functioning as address an entry always has a semantic content relation with the following text.

We annotate an index with the XML element <**index**> and the following attributes:

id	
name	*name used as address*
layout-chunk	id *of the layout chunk which is named*
xref	id *of the base unit which represents this name*

The tag for an entry definition is <**entry**>. It has the following attributes.

id	
xref	id *of the base unit which is the heading*
rst-span*	*id of the RST span which the text in the base unit given under xref is heading*
layout-chunk*	*id of the layout chunk which the text in the base unit given under xref is heading*

From the entry attributes, one has to provide either an rst-span or a layout-chunk value.

The following are examples for an index and an entry markup. The text inside the tags is optional:

```
<index id="i-21" name="21" layout-chunk="flegg-page-21" xref="u-21.23">21</index>

<entry id="e-21.03" xref="u-21.03" rst-span="flegg-gannet">GANNET</entry>
```

Beside the list of entries and indices, which just defines addresses, the most important part of the navigation base is the set of all pointers occuring in the document. The surface realization of pointers are "document deictic expressions", a term coined by Paraboni & van Deemter (2002). Document deictic expressions occur either within sentences or as separate layout units. We have marked the first type as embedded base units and the second as main level base units in the GeM base.

In the navigation base, we specify the semantics of such a document deictic expression as **pointer**. A pointer at a crossroads in the countryside usually gives two pieces of information: first the name of a place or location, and second the direction where one can find this place. The third, implicit information is the actual location of the pointer – the place where it stands. We will analyse pointers in documents in a similar way. A pointer generally consists of three parts – its actual location represented by the **from** attribute, its goal represented by the **to** attribute, and the **address** which is used to label the goal. The address has to be either an entry or an index. The **from** and the **to** of a pointer can be expressed with respect to the layout structure or the RST structure of the document.

First, we consider pointers which are solely induced by layout breaks (page break, column break, etc.) – the so-called **layout-break pointers**. Layout-break pointers are to be found at places where a text is split into two parts which are not adjacent because the entire text does not fit into the available space. The division into the parts very often divides the text in the middle of a sentence, or paragraph. Layout-break pointers can point forwards or backwards. Typical examples are:

continued on page 177,
continued from page 101

The other and primary kind of pointer operates between semantically defined document parts. We presuppose a given RST tree and/or a document structure[7] for the document to be analyzed. Dependent on the location of the start and the goal node in such a tree structure, we can distinguish three types of pointers:

1. pointers from a nonterminal node down to a node in its subtree
2. pointers from a certain node up to an ancestor node
3. pointers between two nodes which do not stand in any ancestor-child relation

We will call these three pointer types

1. table of contents pointers (toc pointer)
2. top pointers
3. cross reference pointers

The most classical pointer type is the cross reference pointer. It appears in, at the front or the end of a text/rst span and points to another text/rst-span of the same document, or outside this document. Some examples are given below:

Before you can use handsets, you will need to fit battery packs and fully charge the batteries, as described on **page 5 and 6.**

If you still don't hear Dial tone, try **the registration procedure** *on* **page**. **12**

Call **our Service Department** *on* **01325 304473** *and ask for a quotation of the repair charge and details of where to send your Pegasys 8 Triple for repair.*

Typical examples for toc pointers can be found in table of contents, alphabetic indexes, navigation menus on web pages. They usually appear as a set of pointers, which we will mark with the XML element <**directory**>. Top pointers are typical for web sites where on each dependent web page a pointer to "home" can be found.

[7]Document structure is the chapter-section-paragraph hierarchy () of the document.

The XML representation of a pointer is given by the element **pointer** with the attributes:

id	
type	cross-reference | toc | top
from*	*id of the smallest RST span or document part in which the pointer appears*
to	*id of the smallest RST span or document part which the pointer points to*
range	in-view | scrollable | site-internal | document-internal | www | p-book | phone | corpus-external

Pointers which refer outside the current document are marked with respect to the goal medium: www, p-book or phone. The value corpus-external is used for pointers referring document-internally, but to a not analyzed part of the document.

Note that we do not restrict the `from` and the `to` values to be RST spans because it is not clear how close the RST structure represents the surface document. In the document generation process, it is possible that one RST span is split into two layout elements because it does not fit onto one page, or two spans out of a RST list are formed into one layout element because they together easily make up one page. In such cases, a layout chunk can appear as `from` or `to` value in the annotation.

Toc pointers do not have a **from** attribute. They appear as children of a <directory> element, which has the attribute **span**. The value of `span` supplies the `from` value for all its children pointers.

Besides these attributes, which specify the navigation structure or the link structure of the document, the XML pointer representation has two XML children, **address** and **content**, which inform about the surface realization of the pointer, the document deictic expression, which is used to establish this link. As one can see in some of the above examples, pointers are often formed out of two parts: the address part (e.g. page number, telephone number), and the content part, which is a short summary, often in a simple noun phrase, about what the reader can expect under this address (e.g. *registration procedure, Service Department*). This requires that both these parts are marked as separate embedded base units in the GeM base. In certain circumstances, one or the other of these two parts can be missing. The content is often not specified, if it can be easily inferred from the context. In web pages, the address part is usually not visible, and therefore not annotated. In toc pointers, both parts are essential.

The <address> element has the following attributes:

xref	*id of the base unit which denotes the document deictic expression*
address*	id *of an entry or an index*
address-type	absolute | relative | in-view
address-precision	start-only | in | start-end | unsure

The `address-type` attribute distinguishes between "absolute" (e.g. *page 5, chapter 12*) and "relative" (e.g. *the next page, the previous section*) pointers. We mark the address only for absolute pointers. It is either an entry (e.g. *section Installation*) or an index (e.g. *page 5*). `xref` refers to the base unit which realizes the pointer and has already been marked in the GeM base.

The <content> element has an **xref** attribute which also indicates the corresponding base unit. Optional **alt** and **src** attributes can be added, if the content is given graphically.

Layout-break pointers are marked with the XML element **layout-break-pointer**. They have the same attributes as ordinary pointers. The only difference is that the possible values for the attribute `type` are here `forward` and `back`. Furthermore, layout-break pointers have no content element.

See below for an example pointer markup:

```
<pointer id="p-7.1" type="cross-reference" from="s-7.18" to="span-12.3"
         range="document-internal">
   <content xref="u-7.18.1">the registration procedure</content>
   <address xref="u-7.18.2" address-precision="in" address="index-12"
            address-type="absolute">page 12</address>
</pointer>
```

References

André, E. (1995), *Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen*, Vol. 108, Infix, St. Augustin.

Barthes, R. (1977), *Image – Music – Text*, Hill and Wang, New York.

Consortium, W. W. W. (2000), 'Extensible markup language (XML) 1.0 (second edition) – W3C recommendation', Available at http://www.w3.org/TR/2000/WD-xml-2e-20000814.
URL: *http://www.w3.org/TR/2000/WD-xml-2e-20000814*

Cristea, D., Ide, N., Marcu, D. & Tablan, V. (2000), An empirical investigation of the relation between discourse structure and co-reference, *in* 'Proceedings of the International Conference on Computational Linguistics (COLING'2000)'.

Delin, J., Bateman, J. A. & Allen, P. (2002), 'A model of genre in document layout', *Information Design Journal* .

Halliday, M. A. K. (1985), *An Introduction to Functional Grammar*, Edward Arnold, London.

Mann, W. C. & Thompson, S. A. (1988), 'Rhetorical structure theory: Toward a functional theory of text organization', *Text* **8**(3), 243–281.

Paraboni, I. & van Deemter, K. (2002), Towards the generation of document-deictic references, *in* K. van Deemter & R. Kibble, eds, 'Information sharing: reference and presupposition in language generation and interpretation', CSLI Publications, pp. 333–358.

RAGS Project (1999), Towards a reference architecture for natural language generation systems, Technical Report ITRI-99-14 and HCRC/TR-102, Information Technology Research Institute (U. Brighton) and Division of Informatics/Human Communication Research Centre (U. Edinburgh), Brighton and Edinburgh. Contributors: Lynne Cahill, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, Donia Scott and Neil Tipper.

Reichenberger, K., Rondhuis, K., Kleinz, J. & Bateman, J. A. (1995), Effective presentation of information through page layout: a linguistically-based approach., *in* ‘Proceedings of ACM Workshop on Effective Abstractions in Multimedia, Layout and Interaction’, ACM, San Francisco, California.
URL: *http://www.cs.tufts.edu/ isabel/mmwsproc.html*

Schriver, K. A. (1997), *Dynamics in document design: creating texts for readers*, John Wiley and Sons, New York.