Text Polishing: Surface-Oriented Smoothing of Generated Text via Markup-Based Revision Rules

Michael White
CoGenTex, Inc.
The Village Green
840 Hanshaw Road, Ithaca, NY 14850, USA
mike@cogentex.com

Abstract

In this paper, we introduce *text polishing,* a novel, surface-oriented approach to using revisions in natural language generation in order to improve the fluency of generated texts. In this approach, markup-based revision rules are used to implement sentence planning operations such as aggregation, referring expression generation and discourse marker insertion. By working close to the surface, we suggest that it becomes easier to partially automate the acquisition of revision rules, and that it becomes possible to improve the scalability of template-based generation systems. We describe our current implementation, including an initial evaluation of a tool for bootstrapping revision rules from examples, and conclude with a discussion of related work.

1 Introduction

In this paper, we introduce *text polishing,* a novel, surface-oriented approach to using revisions in natural language generation in order to improve the fluency of generated texts. In this approach, markup-based revision rules are used to implement sentence planning operations such as aggregation, referring expression generation and discourse marker insertion.[1] In contrast to previous approaches to using revisions in sentence planning (Rambow and Korelsky, 1992; Robin, 1994; Wanner and Hovy, 1996; Callaway and Lester, 1997), where abstract text structures are transformed prior to being sent to a surface realizer, our surface-oriented approach assumes essentially concrete texts[2] as input, which however include enough XML markup — encoding (partially) rhetorical, referential, semantic and morpho-syntactic structure — to enable the revision rules to operate.

We see two principal advantages to our surface-oriented approach. First, working close to the surface makes it easier to partially automate the acquisition of revision rules. To that end, we have developed a tool for bootstrapping revision rules from examples. The bootstrapping tool generates revision rules that are guaranteed to transform each source XML tree into its target XML tree counterpart. The rule designer can then focus on the more interesting task of generalizing such rules to cover the desired range of cases. In an initial evaluation, we found that 67 percent of the desired rule actions could be obtained from the bootstrapped versions. Since manually writing an initial version of a revision rule for a source and target pair is a tedious and error-prone process, we consider this figure to be a lower bound on the labor savings.

The second principal advantage of our approach is that it provides a way to improve the scalability of NLG systems that make use of generalized templates[3] (Busemann, 1996; Geldof and de Velde, 1997; Busemann and Horacek, 1998; White and Caldwell, 1998; Pianta and Tovena, 1999; Cawsey, 2000). With all template-based approaches, as the amount of desired variation increases, it can become difficult to deal with all possible ways in which fragments can be juxtaposed. Adding a text polishing component can simplify the task of specifying the templates, which need only deal with canonical contexts as long as the text polisher can be relied upon to smooth away any problems with stilted or awkward juxtapositions.

The rest of the paper is organized as follows. In Section 2, we describe how the approach can improve the scalability of template-based systems and provide a pair of illustrative revisions. In Section 3, we describe the tool for bootstrapping revision rules from examples, and provide an initial evaluation of

[1]Since sentence planning (or microplanning) is normally conceived of as bridging the gap between text planning and surface realization (Reiter and Dale, 2000), and since our markup-based revision rules are not strictly limited to implementing sentence planning operations, we decided to introduce the new term *text polishing.*

[2]By *essentially concrete texts*, we mean texts that at most need only some morphological synthesis and orthographic processing (including sentence-initial capitalization, sentence-final periods, and insertion of white space sensitive to punctuation) for completion, unlike the sentence specifications that form the inputs to surface realizers.

[3]By *generalized templates* we mean techniques that make use of canned text but go beyond simple fill-in-the-blank processing; as an example, cf. the recursive template-filling capability of XSLT (W3C, 1999b).

this tool. In Section 4, we conclude with a discussion related work.

2 The Approach

To date, generalized template–based generation systems have included relatively little in the way of linguistic annotation in their output, and thus linguistic post-processing has tended to be limited as well. Consequently, tasks such as sentence aggregation, referring expression generation, and insertion of discourse markers are often folded into the initial text structuring component, rather than taking place in a separate sentence planning (or microplanning) component, as is usually the case in systems employing more complex machinery (Reiter and Dale, 2000). We will show that by including more linguistic annotation, it becomes possible to perform such operations in a separate component, while still working with essentially concrete texts in the template fragments.

To illustrate, let us consider an example from a natural language interface to a personal information manager, which we have partially implemented to test out our approach. Suppose that the system is asked to summarize the appointments and tasks for a certain period. In this context, consider the two possible outputs below:

(1) There are three appointments scheduled for this period. There is one task scheduled for this period. All of the appointments are Normal priority. The task is High priority. ...

(2) There are two appointments scheduled for this period. Both of the appointments are Normal priority. ...

While the outputs shown in (1) and (2) can be generated relatively straightforwardly by a generalized template–based system, the results are somewhat stilted, and certainly less fluent than those in (3) and (4):

(3) There are three appointments and one task scheduled for this period. All of the appointments are Normal priority, whereas the task is High priority. ...

(4) There are two appointments scheduled for this period, both of which are Normal priority. ...

One way to achieve the more fluent results would be to add new templates, for example including a separate template for introducing both appointments and tasks in a single sentence, as in (3), vs. just a single template for introducing either appointments or tasks in separate sentences, as in (1) and (2). Of course, a drawback of this method is that the number of templates necessary to achieve the desired results can become unmanageable as the level of variation increases.

```
<seg role="totals-by-type">
  <sent rhet="list">
    <clause>
      <expl>there</expl>
      <aux lex="be" num="3"/>
      <np def="-" num="3" rel="sbj"
          ref="the-appt(s)">
        <card num="3"/>
        <n lex="appointment" num="3"/>
      </np>
      <v>scheduled</v>
      <pp>for this period</pp>
    </clause>
  </sent>
  <sent rhet="list">
    <clause>
      <expl>there</expl>
      <aux lex="be" num="1"/>
      <np def="-" num="1" rel="sbj"
          ref="the-task(s)">
        <card num="1"/>
        <n lex="task" num="1"/>
      </np>
      <v>scheduled</v>
      <pp>for this period</pp>
    </clause>
  </sent>
</seg>
```

Figure 1: Text with markup for (1)

2.1 Adding XML Markup

Rather than trying to anticipate all possible juxtapositions in advance, text polishing can be used to smooth texts such as (1) and (2) into the more fluent versions (3) and (4), as long as the template-based part of the system includes XML markup sufficient to enable the text polisher's revision rules. This should simplify the overall task of specifying the templates, since including markup in the templates should be easier than handling a proliferation of cases.

In order to employ the text polishing approach, one must determine how much and what kinds of markup to include to enable the revision rules. Here there is a tradeoff to be considered between using a minimal amount of markup, to reduce effort, and using a consistent markup scheme, to enable more general revision rules. In our experience so far, we have found it useful to include markup for parts-of-speech and major syntactic constituents, referential identity and rhetorical structure. We suspect it will additionally prove useful to include more semantic markup in certain cases (perhaps via cross-reference to a semantic structure). On the other hand, it is also possible to make do with less markup

```
<sent role="totals-by-type">
  <clause>
    <expl>there</expl>
    <aux lex="be" num="2"/>
    <np def="-" num="2"
        ref="the-appt(s)" rel="sbj">
      <card num="2"/>
      <n lex="appointment" num="2"/>
    </np>
    <v>scheduled</v>
    <pp>for this period</pp>
  </clause>
</sent>
<sent rhet="elab-obj-attr" role="priorities">
  <clause>
    <np def="+" num="2" rel="sbj">
      <det>both</det>
      <prep>of</prep>
      <np def="+" num="pl"
          ref="the-appt(s)" rel="obj">
        <det>the</det>
        <n lex="appointment" num="pl"/>
      </np>
    </np>
    <v lex="be" num="2"/>
    <np num="sg" rel="obj">
      <adj>Normal</adj>
      <n>priority</n>
    </np>
  </clause>
</sent>
```

Figure 2: Text with markup for (2)

```
<sent role="totals-by-type">
  <clause>
    <expl>there</expl>
    <aux lex="be" num="pl"/>
    <np def="-" num="pl" rel="sbj"
      ref="the-appt(s)-the-task(s)">
      <np def="-" num="3" rel="sbj"
        ref="the-appt(s)" rhet="list-e">
        <card num="3"/>
        <n lex="appointment" num="3"/>
      </np>
      <conj>and</conj>
      <np def="-" num="1" rel="sbj"
        ref="the-task(s)" rhet="list-e">
        <card num="1"/>
        <n lex="task" num="1"/>
      </np>
    </np>
    <v>scheduled</v>
    <pp>for this period</pp>
  </clause>
</sent>
```

Figure 3: Text with markup for (3)

```
<sent role="totals-by-type">
  <clause>
    <expl>there</expl>
    <aux lex="be" num="2"/>
    <np def="-" num="2" rel="sbj"
      ref="the-appt(s)">
      <card num="2"/>
      <n lex="appointment" num="2"/>
    </np>
    <v>scheduled</v>
    <pp>for this period</pp>
  </clause>
  <punct>,</punct>
  <clause rhet="elab-obj-attr-e"
    role="priorities">
    <np def="+" num="2" rel="sbj">
      <det>both</det>
      <prep>of</prep>
      <relpro def="+" num="pl" rel="obj"
        ref="the-appt(s)">which</relpro>
    </np>
    <v lex="be" num="2"/>
    <np num="sg" rel="obj">
      <adj>Normal</adj>
      <n>priority</n>
    </np>
  </clause>
</sent>
```

Figure 4: Text with markup for (4)

for some applications, especially if the revision rules are primarily for modifying subsequent references. Figures 1–4 show an indicative level of markup for the first two sentences of examples (1) and (2) and the corresponding first sentences of examples (3) and (4).

2.2 Markup-Based Revisions

In our current implementation, the text polishing component manages a prioritized list of revision rules, using a simple greedy algorithm where matching rules are executed in priority order until no further matches are found (or until a maximum number of iterations is reached, to prevent infinite loops if the rules erroneously lead to cycles). Each revision rule (or edit rule) consists of an XML structure matching condition paired with a list of actions (edits) to execute, plus an optional list of new elements to be inserted by some of the actions. Two sample rules are shown in Figures 5 and 6, and discussed at the end of this section.

The XML structure matching conditions may contain literal element, attribute and text matching conditions as well as more powerful matching conditions that make use of the XPath (W3C, 1999a) subset of XSLT (W3C, 1999b).[4] Matched elements

[4]Our current implementation was inspired by an

```
<!-- combine adjacent sentences in
     totals-by-type section -->
<edit-rule name="combineTotalsByType">
  <match>
    <seg erc:name="d0" role="totals-by-type">
      <sent erc:name="n1" rhet="list">
        <clause erc:name="n2">
          <expl erc:name="n3">there</expl>
          <aux erc:name="n4" lex="be"/>
          <np erc:name="n5" ref="the-appt(s)"/>
          <v erc:name="n8"/>
        </clause>
      </sent>
      <sent erc:name="d1" rhet="list">
        <np erc:path="$d1/clause/*" erc:name="n10"
          ref="the-task(s)"/>
      </sent>
    </seg>
  </match>
  <new-elements>
    <np def="-" erc:name="i0" num="pl"
      ref="the-appt(s)-the-task(s)" rel="sbj"/>
    <conj erc:name="i1">and</conj>
  </new-elements>
  <edits><![CDATA[
    XmlUtils.stripElement(d0);
    XmlUtils.removeNode(d1);
    XmlUtils.insertAfter(i0,n4);
    XmlUtils.insertAfter(i1,n5);
    XmlUtils.insertNodeAndFollowingSiblingsUnder(n5,i0);
    XmlUtils.insertNodeAndFollowingSiblingsAfter(n8,i0);
    XmlUtils.insertAfter(n10,i1);
    XmlUtils.copyAttributes(d0,n1);
    n1.removeAttribute("rhet");
    n4.setAttribute("num","pl");
    n5.setAttribute("rhet","list-e");
    n10.setAttribute("rhet","list-e");
  ]]></edits>
</edit-rule>
```

Figure 5: Revision rule for (1) to (3)

```
<!-- combine adjacent mono-clausal sentences
     in elab-obj-attr relation -->
<edit-rule name="relativizeElabObjAttr">
  <match>
    <sent erc:name="n1" erc:cond="count(clause)=1">
      <clause erc:name="n2">
        <np erc:path="$n2/*" erc:name="n5"
          ref="?ref" rel="sbj"/>
      </clause>
    </sent>
    <sent erc:name="d0" rhet="elab-obj-attr"
      erc:cond="count(clause)=1">
      <clause erc:name="n10">
        <np erc:path="$n10//*" erc:name="n14"
          def="+" ref="?ref"/>
      </clause>
    </sent>
  </match>
  <new-elements>
    <punct erc:name="i0">,</punct>
  </new-elements>
  <edits><![CDATA[
    XmlUtils.stripElement(d0);
    XmlUtils.insertAfter(i0,n2);
    XmlUtils.insertAfter(n10,i0);
    n14 = XmlUtils.changeTagName(n14,"relpro");
    XmlUtils.setSingleTextChildData(n14,"which");
    XmlUtils.copyAttributes(d0,n10);
    n10.setAttribute("rhet","elab-obj-attr-e");
  ]]></edits>
</edit-rule>
```

Figure 6: Revision rule for (2) to (4)

are given names via the `erc:name` attribute (the namespace `erc`, for edit rules compiler, serves to distinguish special matching attributes from literal ones). The XPath-based matching conditions are as follows:

flexible path conditions By default, in order for an element in an XML tree to match an element in the matching condition, it must appear in the same relative location to the preceding element — that is, as either the next sibling or first child of the preceding sibling or parent, respectively. This constraint may be relaxed by providing an explicit path condition via the `erc:path` attribute. For example, the path `$d1/clause/*` given for the `np` element named `n10` in Figure 5 — i.e., the NP which is extracted from the second sentence and coordinated with the subject of the first sentence — allows this element to appear as a child of a (the) `clause` child of element `d1`, rather than requiring it to be `d1`'s first child.

boolean XPath conditions Further conditions on an element may be specified via the `erc:cond` attribute, whose value may be any boolean XPath expression. For example, in Figure 6, the `sent` element named `n1` is required to contain exactly one `clause` child.

variable attribute and text matching Two or more attributes or text nodes may be required to have matching values, rather than the same literal values. For example, in Figure 6, the NP elements `n5` and `n14` are required to have matching `ref` attributes.

The actions to execute to perform an edit operation are written in Java and make use of references to the named elements. The available actions are those in the XML Document Object Model (DOM) specification (W3C, 2000) plus a set of convenience routines that we have written using these APIs in the class `XmlUtils`.

Since our text polishing approach does not require the XML markup to fully encode all potentially useful structure, the revision rules tend to be mostly application-specific. The `combineTotalsByType` rule shown in Figure 5 is an example of an application-specific rule. This rule is restricted to adjacent sentences in a segment whose role in the text has the application-specific value `totals-by-type`. Given the limited range of cases that this rule is intended to match, it may safely make use of various simplifying assumptions in the matching condition and list of actions. For instance,

earlier implementation that made use of XML-QL (http://www.w3.org/TR/NOTE-xml-ql), a query language for XML.

there is no need to check whether coordinating the subject NPs could lead to a collective/distributive ambiguity, or whether deleting all but the subject NP in the second sentence could lose important information.

The `relativizeElabObjAttr` rule shown in Figure 6 is an example of a rule intended to have broader applicability. It applies to adjacent sentences in the `elab-obj-attr` relation, i.e. the *elaboration-object-attribute* subtype of the *elaboration* relation from Rhetorical Structure Theory (Mann and Thompson, 1988). Since this rule is restricted to mono-clausal sentences — in order to keep the resulting syntactic complexity under control — and since its actions are essentially limited to appending the second clause and introducing a relative prononoun, it embodies considerably fewer simplifying assumptions than the `combineTotalsByType` rule. Nevertheless, of course, for a specific system it may be necessary to further restrict this rule to avoid unwanted invocations.

3 Bootstrapping Revision Rules

In order to speed up the process of writing revision rules, we have developed a tool for bootstrapping them from examples. This tool takes as an input a source and a target XML file along with the name of the desired rule, and produces as output a revision rule that will transform the source into the target. The rule designer can then generalize the bootstrapped rule to cover the desired range of cases.

An example bootstrapped rule appears in Figure 7, which will transform the XML in Figure 1 into the XML in Figure 3. In the previous section, we saw a manually generalized version of this rule in Figure 5. In the generalized version, the matching condition has been simplified, and the deletion of individual elements from the second sentence has been generalized to deleting all of the second sentence except for the extracted NP.

3.1 Algorithm

The bootstrapping tool operates in two phases. In the first phase, the source and target XML trees are aligned. The alignment consists of a mapping between source and target elements, where unmapped elements are considered to be deleted from the source tree or inserted in the target tree. The alignment also indicates which elements have been changed — i.e., which elements have differing tag names, attributes or text contents — as well as which elements have been moved from their expected location, where the expected location is calculated relative to an element's preceding sibling or parent, if none. In the bootstrapping tool's second phase, the alignment is used to generate an edit rule. In the generated rule, the matching condition is simply a literal match on the source tree, the new elements are those considered inserted by the alignment, and the actions are derived from the alignment in the order deletes, inserts, moves, and changes.

```
<edit-rule name="combineTotalsByType">
  <match>
    <seg erc:name="d0" role="totals-by-type">
      <sent erc:name="n1" rhet="list">
        <clause erc:name="n2">
          <expl erc:name="n3">there</expl>
          <aux erc:name="n4" lex="be" num="3"/>
          <np def="-" erc:name="n5" num="3"
            ref="the-appt(s)" rel="sbj">
            <card erc:name="n6" num="3"/>
            <n erc:name="n7" lex="appointment" num="3"/>
          </np>
          <v erc:name="n8">scheduled</v>
          <pp erc:name="n9">for this period</pp>
        </clause>
      </sent>
      <sent erc:name="d1" rhet="list">
        <clause erc:name="d2">
          <expl erc:name="d3">there</expl>
          <aux erc:name="d4" lex="be" num="1"/>
          <np def="-" erc:name="n10" num="1"
            ref="the-task(s)" rel="sbj">
            <card erc:name="n11" num="1"/>
            <n erc:name="n12" lex="task" num="1"/>
          </np>
          <v erc:name="d5">scheduled</v>
          <pp erc:name="d6">for this period</pp>
        </clause>
      </sent>
    </seg>
  </match>
  <new-elements>
    <np def="-" erc:name="i0" num="pl"
      ref="the-appt(s)-the-task(s)" rel="sbj"/>
    <conj erc:name="i1">and</conj>
  </new-elements>
  <edits><![CDATA[
    XmlUtils.stripElement(d0);
    XmlUtils.stripElement(d1);
    XmlUtils.stripElement(d2);
    XmlUtils.removeNode(d3);
    XmlUtils.removeNode(d4);
    XmlUtils.removeNode(d5);
    XmlUtils.removeNode(d6);
    XmlUtils.insertAfter(i0,n4);
    XmlUtils.insertAfter(i1,n5);
    XmlUtils.insertNodeAndFollowingSiblingsUnder(n5,i0);
    XmlUtils.insertNodeAndFollowingSiblingsAfter(n8,i0);
    XmlUtils.insertNodeAndFollowingSiblingsAfter(n10,i1);
    n1.removeAttribute("rhet");
    n4.setAttribute("num","pl");
    n1.setAttribute("role","totals-by-type");
    n5.setAttribute("rhet","list-e");
    n10.setAttribute("rhet","list-e");
  ]]></edits>
</edit-rule>
```

Figure 7: Bootstrapped rule for (1) to (3)

For the tree alignment algorithm, we considered using an optimal tree edit distance algorithm such as (Zhang and Shasha, 1989) or (Klein et al., 2000), but found them unsuitable because they require dominance relationships to be preserved in the source and target trees.[5] The reason this dominance requirement is problematic in the context of bootstrapping

[5]We also looked at the approach to aligning structured multilingual documents of (Ballim et al., 1998), but it did not appear suitable for the sort of fine-grained alignment needed here.

revision rules is that cases of moved constituents — such as the NP which is moved from the second sentence in Figure 1 into the coordinate subject of the first sentence in Figure 3 — will necessarily be treated as cases of deletion and insertion, which will not generalize properly.

Having not found a suitable existing algorithm,[6] we devised a new dynamic programming alignment algorithm that performs a top-down, bidirectional beam search for the least cost mapping between the source and target XML trees. The algorithm is parameterized by functions for computing the costs of deletes, inserts, moves and changes.

3.2 Initial Evaluation

To perform an initial evaluation of the bootstrapping tool, we set out to recreate the seven revision rules that were originally written by hand for the test application discussed in Section 2, from source and target examples such as those shown in Figures 1–4. After setting the cost functions for the tree alignment algorithm analytically, and with almost no tweaking, we were pleased to find that all the source and target trees were aligned correctly. We then ran the bootstrapping tool on the example pairs and generalized the resulting rules to recreate the functionality of the original rules. At this point we counted the number of actions in the generalized rules that came from the bootstrapped versions, and found that 29 out of 43, or 67 percent, of the desired actions were generated by the bootstrapping tool. As mentioned in the introduction, we consider this figure to be a lower bound on the labor savings, since manually writing a revision rule for a source and target pair is a tedious and error-prone process.

To date we have employed our text polishing approach in one commercial and one research prototype.[7] Since these prototypes are still in their early stages, we have yet to evaluate the success of the bootstrapping tool with these systems.

4 Discussion

We view our bootstrapping tool as a first step in the direction of inducing revision rules for handling sentence planning tasks such as sentence aggregation, referring expression generation, and insertion of discourse markers. Of these, we consider our approach to be particularly apt for sentence aggregation, since, as Reiter and Dale (2000) point out, the contexts in which various aggregation operations are appropriate are highly application-dependent; moreover, research on aggregation in NLG, such as (Shaw, 1998), is still in its early stages, having yet to have made extensive use of the considerable work in theoretical linguistics in this area.

In comparison to other revision-based approaches, we have argued in favor of sticking close to the surface. However, technically, nothing prevents our approach from being used in conjunction with a surface realizer, such as RealPro (Lavoie and Rambow, 1997) — our revision rules could just as easily be used to transform syntactic specifications as long as they were encoded in XML. Indeed, this could make certain revisions simpler or easier to generalize, as we have found to be the case with delaying some morphological synthesis and orthographic processing. One reason for preferring essentially concrete texts is that it reduces the need to reverse engineer abstract specifications for desired surface forms. Another reason is that in dealing with interacting linguistic phenomena, we prefer to work with concrete representations encoding information from various levels, rather than representations that abstract away from details such as word order. This use of information from various levels represents a similar departure from the traditional pipeline architecture as in (Mellish et al., 2000).

[6] We have not investigated adapting general purpose graph matching algorithms, such as the relaxation-based approach of (Gold and Rangarajan, 1996).

[7] The research prototype is part of the RIPTIDES system (http://www.cs.cornell.edu/Info/People/cardie/tides/).

Acknowledgements

The author gratefully acknowledges Ted Caldwell, Benoit Lavoie, Tanya Korelsky, Owen Rambow, Johanna Moore and Helmut Horacek for helpful discussion. This work was partially supported by DARPA TIDES contract N66001-00-C-8009.

References

A. Ballim, G. Coray, A. Linden, and C. Vanoirbeek. 1998. The Use of Automatic Alignment on Structured Multilingual Documents. In R. D. Hersch, J. André, and H. Brown, editors, *Electronic Publishing, Artistic Imaging, and Digital Typography*, Lecture Notes in Computer Science. Springer-Verlag.

S. Busemann and H. Horacek. 1998. A Flexible Shallow Approach to Text Generation. In *Proceedings of the 9th International Workshop on Natural Language Generation*, pages 238–247, Niagara-on-the-Lake, Ontario.

Stephan Busemann. 1996. Best-first surface realization. In *Proceedings of the 8th International Natural Language Generation Workshop (INLG-96)*, pages 101–110, Herstmonceux Castle, Sussex, UK.

Charles B. Callaway and James C. Lester. 1997. Dynamically Improving Explanations: A Revision-Based Approach to Explanation Generation. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan.

Alison Cawsey. 2000. Presenting tailored resource descriptions: will XSLT do the job? In *Proceed-*

ings of the 9th International Conference on the World Wide Web.

Sabine Geldof and W. Van de Velde. 1997. An architecture for template based hyper(text) generation. In *Proceedings of the 6th European Workshop on Natural Language Generation*, pages 28–37.

S. Gold and A. Rangarajan. 1996. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:377–388.

Philip Klein, Daniel Sharvit, and Ben Kimia. 2000. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms (SODA) 2000.*

Benoit Lavoie and Owen Rambow. 1997. RealPro – A Fast, Portable Sentence Realizer. In *Proceedings of the Conference on Applied Natural Language Processing (ANLP'97)*, Washington, DC.

William Mann and Sandra Thompson. 1988. Rhetorical structure theory: Towards a functional theory of text organisation. *Text*, 3:243–281.

C. Mellish, R. Evans, L. Cahill, C. Doran, D. Paiva, M. Reape, D. Scott, and N. Tipper. 2000. A Representation for Complex and Evolving Data Dependencies in Generation. In *Proceedings of the Conference on Applied Natural Language Processing (ANLP-2000)*, Seattle.

E. Pianta and L. M. Tovena. 1999. Mixing representation levels: The hybrid approach to automatic text generation. In *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*, pages 8–13, Edinburgh, Scotland.

Owen Rambow and Tanya Korelsky. 1992. Applied Text Generation. In *Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems.* Cambridge University Press.

Jacques Robin. 1994. *Revision-Based Generation of Natural Language Summaries Providing Historical Background.* Ph.D. thesis, Columbia University.

James Shaw. 1998. Segregatory coordination and ellipsis in text generation. In *Proceedings of COLING-ACL '98*, pages 1220–1226, Montreal, Canada.

W3C. 1999a. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath.

W3C. 1999b. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt.html.

W3C. 2000. Document Object Model (DOM) Level 2 Core Specification Version 1.0. http://www.w3.org/TR/DOM-Level-2-Core/.

Leo Wanner and Eduard Hovy. 1996. The HealthDoc Sentence Planner. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG-96)*, pages 1–10, Herstmonceux Castle, Sussex, UK.

Michael White and Ted Caldwell. 1998. EXEMPLARS: A Practical, Extensible Framework for Dynamic Text Generation. In *Proceedings of the 9th International Workshop on Natural Language Generation*, pages 266–275, Niagara-on-the-Lake, Ontario.

K. Zhang and D. Shasha. 1989. Simple Fast Algorithms For The Editing Distance between Trees and Related Problems. *SIAM J. COMPUT.*, 18(6):243–281.