

# AN OVERVIEW OF SHALLOW XML-BASED NATURAL LANGUAGE GENERATION

Graham Wilcock  
University of Helsinki

## Abstract

The paper gives an overview of shallow XML-based natural language generation, including XML pipeline architectures, text planning with XSLT templates, and transformations from text plan trees to text specification trees. The work is based on practical experience in a spoken dialogue system, and examples from this system are presented.

## 1. Introduction

The paper gives an overview of shallow XML-based natural language generation (NLG) including XML pipeline architectures, text planning with XSLT templates, and transformations from text plan trees to text specification trees, using open-source software. The ideas are based on practical experience in developing an XML-based generation component for a spoken dialogue system (Jokinen et al. 2002). Some examples from this system are given in Section 2, and the basic methods are described further in Section 3, followed by more general discussion in Section 4.

### 1.1. Pipelines

A pipeline is the most widely-used NLG architecture (Reiter and Dale 2000). In XML-based generation it is easy to use a pipeline as powerful methods for organizing XML pipelines are available. For example, Apache Cocoon (Apache Cocoon Project) is an industrial-strength, scalable XML pipeline processor. The pipeline architecture adopted here follows the textbook by Reiter and Dale (2000) as shown in Figure 1.

The interface between text planning and microplanning is a text plan, a tree whose leaves are domain-specific concept messages. The interface between microplanning and realization is a text specification, another tree whose leaves are linguistic phrase specifications. Both the text plan trees and the text specification trees can be naturally represented in XML, as illustrated in Section 2.

### 1.2. Templates

The status of template-based generation has been debated by NLG researchers (Becker and Busemann 1999). Generally, templates are considered suitable only for shallow forms of generation, in which the templates contain predefined surface strings. However, if template-based means “making extensive use of a mapping between semantic

- Document Planning (or Text Planning)
  - content determination
  - document structuring
- Microplanning
  - lexicalization
  - referring expression generation
  - aggregation
- Realization
  - linguistic realization
  - structure realization (e.g. HTML)

Figure 1: NLG Pipeline

structures and representations of linguistic surface structures that contain gaps” (van Deemter et al. 1999), then templates can also have a role in deeper forms of generation. In either case, NLG templates can be naturally implemented in XML by means of XSLT templates (Wilcock 2001: 2002).

The approach adopted here is to use templates as a good way to create initial text plan trees that contain gaps to be filled later when the concept messages are turned into phrase specifications in the text specification tree. This is not template-based generation, it is template-based text planning. The text plan is then passed through the various stages of the generation pipeline for further processing. The implementation of this form of template-based text planning using XSLT templates is described in Section 3.1.

### 1.3. Transformations

The text plan tree is transformed into a text specification tree by the microplanning stages, and the text specification tree is transformed into the required output by the realization stages. This requires tree-to-tree transformation processing. Section 3.2 describes an approach in which the required transformations are performed by a sequence of XSLT stylesheets.

When both the text plan tree and the text specification tree are represented in XML, transformation from one to the other is XML-to-XML transformation. When the required output is XHTML for web pages or Java Speech Markup Language (Sun Microsystems 1999) for speech output, the realization stage also performs XML-to-XML transformation.

## 2. Examples: Spoken Dialogue Responses

Examples of XML-based response generation within a spoken dialogue system are discussed by Jokinen and Wilcock (2003). Some of the examples are repeated here. The generation component, which performs bilingual generation of responses in Finnish and English for a Helsinki bus timetable enquiry system, has been demonstrated by Wilcock (2003). The responses depend on the dialogue context and can vary from full sentences to short elliptical phrases.

For spoken dialogue response generation, content determination is done by the dialogue manager, and document structuring is greatly simplified because the generated response is typically very short. The starting point for the generation component in a spoken dialogue system is therefore a specification of the utterance content which is determined by the dialogue manager, as described in Section 2.2.

### 2.1. Generation from NewInfo

In the approach taken here, dialogue response planning starts from the new information focus, known as *NewInfo*. This approach to generation from *NewInfo* was developed by Jokinen (Jokinen et al. 1998; Jokinen and Wilcock 2003). One of the tasks of the generator is to decide how to present the *NewInfo* to the user: whether it should be presented by itself or whether it should be *wrapped* in a link to the Topic.

- (1) User: *Which bus goes to Malmi?*  
System: *Number 74.*
- (2) User: *How do I get to Malmi?*  
System: *By bus - number 74 goes there.*

In Example 1 *NewInfo* is the information about the bus number, while in Example 2 *NewInfo* concerns the means of transportation. In both cases, *NewInfo* is presented to the user by itself, without linking to the Topic.

- (3) *When will the next bus leave for Malmi?*
  - (a) *2.20pm*
  - (b) *It will leave at 2.20pm*
  - (c) *The next bus to Malmi leaves at 2.20pm*

Whether *NewInfo* should be wrapped or not depends on the changing dialogue context. When the context permits a fluent exchange of contributions, wrapping is avoided and the response is based on *NewInfo* only, as in Example 3a. When the context requires more clarity and explicitness, *NewInfo* is wrapped by Topic information as in Example 3b in order to avoid misunderstanding. When the communication channel is working well, wrapping can be reduced, but when there are uncertainties about what was actually said, wrapping must be increased as in Example 3c to provide implicit confirmation. These examples are discussed in more detail by Jokinen and Wilcock (2003).

## 2.2. Input: an Agenda in XML

The starting point for the generation pipeline is an *agenda*, a set of concepts determined by the dialogue manager. In Example 1 *Number 74*, the bus number information is supplied by the dialogue system's task manager, which consults the timetable database. The dialogue manager puts concepts into the XML agenda, as shown in Figure 2.

```
<agenda id="1">
  <concept info="Topic">
    <type>transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>Malmi</value>
  </concept>
  <concept info="Topic">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```

Figure 2: Agenda for Example 1

The root node of the XML tree is `<agenda>`, and its children are `<concept>` nodes, which here represent an unordered set. The dialogue manager labels each concept as `NewInfo` or `Topic`, using its knowledge of how the concepts relate to the current dialogue situation. These labels are represented in the XML agenda by attributes. Here, the concept 'busnumber' is labelled as `NewInfo`, and the other three concepts are labelled as `Topic`. The representation shown in Figure 2 is simplified for clarity.

## 2.3. A Text Plan in XML

```
<TextPlan id="1">
  <Message>
    <type>NumMsg</type>
    <concept info="NewInfo">
      <type>busnumber</type>
      <value>74</value>
    </concept>
  </Message>
</TextPlan>
```

Figure 3: Text Plan for Example 1

In text planning, the content determination stage simply extracts the concepts from the agenda. As the dialogue manager has already decided the relevant concepts and put them in the agenda, no other content determination is needed. However, the generator decides whether to generate only the NewInfo, or whether to include a Topic link in addition to NewInfo. In the case of Example 1, only NewInfo is generated.

The discourse structuring stage creates a text plan tree as shown in Figure 3 using a form of template-based text planning described in Section 3.1. In Example 1 there is only one message, which is typical in spoken dialogue responses. In multi-paragraph text generation there would be large numbers of messages. The text plans are XML tree structures containing variable slots, which will be filled in later by the microplanning stages. In the text planning stage, the concepts from the agenda are copied directly into the appropriate slots. In Example 1 there is only one NewInfo concept, so only this concept is copied from Figure 2 to Figure 3. This first example is therefore minimal, with one concept in one message.

#### 2.4. A Text Specification in XML

The processing during microplanning is done by a sequence of XSLT transformations, as described in Section 3.2. The text plan tree is transformed into a text specification tree, as shown in Figure 4.

```
<TextSpec id="1">
  <PhraseSpec>
    <subject cat="NP">
      <head>number</head>
      <attribute>74</attribute>
    </subject>
  </PhraseSpec>
</TextSpec>
```

Figure 4: Text Specification for Example 1

The messages in the text plan tree are replaced by phrase specifications in the text specification tree. In the referring expression stage of microplanning, domain concepts are replaced with linguistic referring expressions. In the text specification in Figure 4 the single concept of Figure 3 has been replaced by a linguistic specification.

#### 2.5. Output: Speech Markup in XML

The realization stage produces output marked up in Java Speech Markup Language (JSML) (Sun Microsystems 1999) as shown in Figure 5.

```
<jsgml lang="en">
  <div type="sent"> Number
    <sayas class="number">74</sayas> </div>
</jsgml>
```

Figure 5: Speech Markup for Example 1

The <head> words of the text specification (Figure 4) provide the main content of the output. In the speech markup, <div type="sent"> marks sentence boundaries and <sayas> tells the speech synthesizer how to say something - in Figure 5 it shows that 74 should be pronounced "seventy-four" not "seven four".

## 2.6. Another Example

In Example 1 *Which bus goes to Malmi?*, only the concept 'busnumber' is labelled as NewInfo, and the other three concepts are labelled as Topic. This leads to the minimal output *Number 74*. In the case of Example 2 *How do I get to Malmi?*, the dialogue manager specifies the means of transportation as NewInfo and the destination as Topic. This will be generated as *By bus*.

In addition, the dialogue manager provides further new information about the bus number, following a co-operative dialogue strategy. This will be generated as *Number 74 goes there*. In this case a more complex text plan is selected as shown in Figure 6. The text plan has two messages, and a prosody markup is inserted between the two messages. The text plan has four concepts. One concept is copied into the first message and three concepts are copied into the second message.

```
<TextPlan id="2">
  <Message>
    <type>TransportMsg</type>
    <concept info="NewInfo">
      <type>transportation</type>
      <value>bus</value>
    </concept>
  </Message>
  <prosody cat="pause" />
  <Message>
    <type>NumDestMsg</type>
    <concept info="NewInfo">
      <type>busnumber</type>
      <value>74</value>
    </concept>
    <concept info="NewInfo">
      <type>bus</type>
      <value>exists</value>
    </concept>
    <concept info="Topic">
      <type>destination</type>
      <value>Malmi</value>
    </concept>
  </Message>
</TextPlan>
```

Figure 6: Text Plan for Example 2

The text plan in Figure 6 is transformed by the microplanning stages into the text specification shown in Figure 7. The two messages are made into two phrase spec-

ifications, with the prosody element between them. Because the *destination* concept in Figure 6 is marked as Topic, it is pronominalized as *there* by the referring expressions stage. If the same destination concept were marked as NewInfo, it would be realized by the actual text value of the destination placename, in this case *to Malmi*.

```
<TextSpec id="2">
  <PhraseSpec>
    <adverbial cat="PP">
      <head>by</head>
      <object cat="NP">
        <head>bus</head>
      </object>
    </adverbial>
  </PhraseSpec>
  <prosody cat="pause"/>
  <PhraseSpec>
    <head>go</head>
    <features>3sg</features>
    <subject cat="NP">
      <head>number</head>
      <attribute>74</attribute>
    </subject>
    <adverbial cat="PP">
      <head>there</head>
    </adverbial>
  </PhraseSpec>
</TextSpec>
```

Figure 7: Text Specification for Example 2

The text specification in Figure 7 is realized in Example 2 as the response *By bus - number 74 goes there*. This response is marked up in JSML as shown in Figure 8. The two phrase specifications are realized as two utterance divisions. The prosody element is realized as a `<break>` element telling the speech synthesizer that a pause is required before the second part of the response.

```
<jsml lang="en">
  <div type="sent"> By bus </div>
  <break size="large"/>
  <div type="sent"> Number
    <sayas class="number">74</sayas>
    goes there </div>
</jsml>
```

Figure 8: Speech Markup for Example 2

### 3. XML-based Generation

The starting point is the agenda of concepts specified in XML. The finishing point is the utterance to be passed to the speech synthesizer, specified in a speech mark-up language which is also XML. This is generating XML from XML. Moreover, XML is used for all the internal representations along the stages of the pipeline. One advantage of using XML is that the internal representations can be defined in DTDs or XML Schemas, and they can be checked by standard XML validation techniques.

In some stages a new XML tree is created. For example, in the text planning stage described in Section 3.1 a new text plan tree is created. In other stages information is added to the existing tree, or nodes are replaced with new nodes. For example, in the referring expression stage described in Section 3.2 domain concepts are replaced with linguistic referring expressions, within the existing text specification tree.

Whether creating a new XML tree or adding information to the existing XML tree, different processing options can be used. For example, the DOM document model can be used for explicit manipulation of the tree nodes by Java programs, or XSLT can be used to specify XML transformations. It is simple to set up a sequence of transformations, in which the output of one transformation is the input to the next transformation. This is a natural way to implement the NLG pipeline architecture.

#### 3.1. Template-based Text Planning

In the NewInfo-based model of generation described in Section 2.1, text planning selects those concepts marked as NewInfo as the basis for generation, and decides whether NewInfo will be the only output, or whether it will be preceded by the Topic linking concepts. In a less shallow approach, text planning combines messages to construct a text plan. In a more shallow approach, complete text plans are predefined by means of XSLT named templates, as illustrated in Figure 9.

```
<xsl:template name="TRANSPORT-PLUS-NUMDEST" >
<TextPlan>
  <Message>
    <type>TransportMsg</type>
    <xsl:copy-of select="./concept[type='transportation']"/>
  </Message>
  <prosody cat="pause"/>
  <Message>
    <type>NumDestMsg</type>
    <xsl:copy-of select="./concept[type='busnumber']"/>
    <xsl:copy-of select="./concept[type='bus']"/>
    <xsl:copy-of select="./concept[type='destination']"/>
  </Message>
</TextPlan>
</xsl:template>
```

Figure 9: Simplified Text Plan Template

The text plan template creates a new XML tree, with root node <TextPlan>. This contains two messages. The messages have variable slots, which will be filled in

later by the lexicalization and referring expression stages. In the text planning stage, the concepts from the agenda are copied directly into the appropriate slots by means of `<xsl:copy-of>` statements. The example in Figure 9 shows a simplified text plan for Example 2 *By bus - number 74 goes there*.

Selecting the appropriate text plan template is based on the agenda: which concept types are in the agenda, and whether their information status is Topic or NewInfo. The selection can be implemented by means of nested `<xsl:choose>` statements.

### 3.2. Transformation-based Microplanning

In the microplanning stages of the pipeline, further information needs to be added to the XML tree, or nodes in the tree need to be replaced by new nodes. This can be done either by explicit tree node manipulation using the DOM document model, or by specifying transformations in XSLT stylesheets. We have implemented prototype generators using both approaches, but suggest that these transformations can be most naturally expressed using XSLT, combining XPath expressions to find the relevant part of the tree and XSLT expressions to specify the transformations.

In the referring expression and lexicalization stages, the domain concepts in the text plan are replaced by noun phrases for referring expressions and by other language-specific lexical items. We illustrate how these stages can be implemented in XML by means of the following simplified examples of XSLT templates. The basic idea is that concepts which are marked as Topic are realized as pronouns, whereas concepts which are marked as NewInfo are realized as full descriptions.

```
<!-- REFERRING EXPRESSIONS: PRONOUNS -->
<xsl:template match="concept[@info='Topic']"
  mode="referring-expression">
  <xsl:choose>
    <xsl:when test="type='busnumber'">
      <xsl:text> it </xsl:text>
    </xsl:when>
    <xsl:when test="type='destination'">
      <xsl:text> there </xsl:text>
    </xsl:when>
    <xsl:when test="type='bustime'">
      <xsl:text> then </xsl:text>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

Figure 10: Referring Expressions: Pronouns

In Figure 10, a destination concept which is marked as Topic is pronominalized as *there*. By contrast, if the same destination concept were marked as NewInfo, it could be realized as a full description by the template in Figure 11, which generates a prepositional phrase with the preposition *to* followed by the actual text value of the destination placename, obtained by the `<xsl:value-of>` statement.

```

<!-- REFERRING EXPRESSIONS: DESCRIPTIONS -->
<xsl:template match="concept[@info='NewInfo']"
  mode="referring-expression">
  <xsl:choose>
  <xsl:when test="type='busnumber'">
    <xsl:text> number </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  <xsl:when test="type='destination'">
    <xsl:text> to </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  <xsl:when test="type='bustime'">
    <xsl:text> at </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  </xsl:choose>
</xsl:template>

```

Figure 11: Referring Expressions: Descriptions

These examples are simplified to show simple text output. In fact the generator performs further syntactic and morphological realization and produces output in a speech mark-up language, as described in Section 3.3.

### 3.3. Realization

The final stage of the pipeline for text generation listed in Section 1.1 renders the generated text in a specific output presentation format such as HTML. This kind of transformation to a presentation format is the task XSLT was originally designed for.

In spoken dialogue response generation, the output must be in the format required by the speech synthesizer. The implemented demonstration uses FreeTTS (Sun Microsystems 2002), a free, open-source speech synthesizer implemented entirely in Java. FreeTTS accepts input marked up in JSML, Java Speech Markup Language (Sun Microsystems 1999). As JSML is XML-based it can easily be produced by the XSLT pipelines.

However, XSLT is not suitable for all kinds of processing. The Finnish language has highly complex morphology, and a generator for Finnish must be able to handle complex morphological processing as part of the realization stage. XSLT would be unsuitable for this kind of processing, and in any case existing morphological generation software is available, which we wish to re-use. In such situations, XSLT can be combined with general purpose programming languages by embedding extension functions in XSLT templates. These functions can be written in Java (Apache XML Project).

## 4. Discussion and Related Work

It is easy to set up a pipeline architecture with XSLT. It is easy to perform template-based generation with XSLT. It is easy to perform tree-to-tree transformations with XSLT. This clearly raises the wider question of whether XSLT is really suitable as a general tool for building complete NLG systems.

This is discussed by Cawsey (Cawsey 2000), who concludes that XSLT transformations can be used for generation when the input is fairly constrained, but that XSLT is not suitable for less constrained input, when we need to turn to general purpose programming languages or NLG tools.

White and Caldwell (White and Caldwell 1999) compare their Java-based EXEMPLARS generator with XSLT and suggest that their system has advantages because it is more object-oriented. However, the problem with object-oriented systems and general purpose programming languages is precisely that they require the participation of skilled programmers. One of the major advantages of XSLT is that it is *not* a general-purpose programming language, but falls rather into the category of scripting languages which are (at least relatively) accessible and useable by non-programmers.

In any case, XSLT can be combined with general purpose programming languages by adding Java extension functions to XSLT templates. This means that even where general purpose programming languages are required for specific purposes, a pipeline of XSLT transformations can still be used as a uniform framework.

Seki (2001) has demonstrated an XML-based generation system for Japanese. His work combines Java and XSLT processing in a pipeline architecture similar to the approach discussed here.

Foster and White (2004) describe the use of XSLT for text planning. They combine XML-based generation with an open-source surface realizer implemented in Java.

## References

- Apache Cocoon Project. Apache Cocoon. <http://cocoon.apache.org/>
- Apache XML Project. Xalan-Java. <http://xml.apache.org/xalan-j/>
- Becker, Tilman; Busemann, Stephan (eds.) 1999. May I Speak Freely? Between Templates and Free Choice in Natural Language Generation. Proceedings of the KI-99 Workshop. DFKI, Saarbrücken
- Cawsey, Alison 2000. Presenting tailored resource descriptions: Will XSLT do the job?. In: *9th International World Wide Web Conference*. <http://www9.org/w9cdrom/>
- Foster, Mary Ellen; White, Michael 2004. Techniques for text planning with XSLT. In: *RDF/RDFS and OWL in Language Technology: Proceedings of the 4th Workshop on NLP and XML (NLPXML-2004)*, Barcelona. 1–8
- Jokinen, Kristiina; Kerminen, Antti; Kaipainen, Mauri; Jauhiainen, Tommi; Wilcock, Graham; Turunen, Markku; Hakulinen, Jaakko; Kuusisto, Jukka; Lagus, Krista 2002. Adaptive dialogue systems - Interaction with Interact. In: *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, Philadelphia. 64–73

- Jokinen, Kristiina; Tanaka, Hideki; Yokoo, Akio 1998. Planning dialogue contributions with new information. In: *Proceedings of the Ninth International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario. 158–167
- Jokinen, Kristiina; Wilcock, Graham 2003. Adaptivity and response generation in a spoken dialogue system. In: van Kuppevelt, J.; Smith, R. (eds.), *Current and New Directions in Discourse and Dialogue*, Kluwer Academic Publishers. 213–234
- Reiter, Ehud; Dale, Robert 2000. *Building Natural Language Generation Systems*. Cambridge University Press
- Seki, Yohei 2001. XML transformation-based three-stage pipelined natural language generation system. In: *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium (NLPRS-2001)*, Tokyo. 767–768
- Sun Microsystems 1999. Java Speech Markup Language Specification, version 0.6. <http://java.sun.com/products/java-media/speech/>
- Sun Microsystems 2002. FreeTTS: A speech synthesizer written entirely in the Java programming language. <http://freetts.sourceforge.net/>
- van Deemter, Kees; Krahmer, Emiel; Theune, Mariët 1999. Plan-based vs. template-based NLG: A false opposition?. In: Becker and Busemann (1999). 1–5
- White, Michael; Caldwell, Ted 1999. Beyond XSL: Generating XML-Annotated Texts with EXEMPLARS. CoGenTex, Inc.
- Wilcock, Graham 2001. Pipelines, templates and transformations: XML for natural language generation. In: *Proceedings of the 1st NLP and XML Workshop*, Tokyo. 1–8
- Wilcock, Graham 2002. XML-based Natural Language Generation. In: *Towards the Semantic Web and Web Services: XML Finland 2002 - Slide Presentations*, Helsinki. 40–63
- Wilcock, Graham 2003. Integrating Natural Language Generation with XML Web Technology. In: *Proceedings of the Demo Sessions of EACL-2003*, Budapest. 247–250

GRAHAM WILCOCK is Docent of Language Technology at University of Helsinki. He is co-organiser and co-chair of the NLPXML series of international workshops on natural language processing and XML. He has given invited talks and courses on XML-based natural language generation in UK, Japan, Finland and Estonia. He is co-chair of the 10th European Workshop on Natural Language Generation (2005). E-mail: [graham.wilcock@helsinki.fi](mailto:graham.wilcock@helsinki.fi).