XML-based Natural Language Generation

Graham Wilcock
Docent of Language Technology
University of Helsinki
graham.wilcock@helsinki.fi

Acknowledgements

• Part 1 (What is NLG?)
  is largely based on Stephan Busemann’s
  Natural Language Generation: An Overview
  (http://www.dfki.de/~busemann/VL-SS99/)

• Parts 2 (XML-based NLG) and 3 (demos)
  are based on work in the USIX-Interact project.
  (http://www.mlab.uiah.fi/interact/)
Requirements

• For this tutorial, you need:
  • a general knowledge of XML
  • a basic idea of XSLT transformations
• You don’t need:
  • any previous knowledge of natural language generation

Tutorial Outline

• Part 1: What is Natural Language Generation?
  • “Shallow” and “deep” NLG
• Part 2: XML-based NLG
  • Templates + trees + pipelines
  • A step-by-step example
• Part 3: Demos
  • Voice NLG with XML + Java speech
  • Web NLG with Cocoon servlets
Part 1: What is NLG?

Natural language generation is the process of deliberately constructing a natural language text in order to meet specified communicative goals. [McDonald 1992]

- Recommended textbook: 
  Ehud Reiter & Robert Dale, 
  *Building Natural Language Generation Systems*, 

NLG inputs and outputs

- **NLG input**
  - a communicative goal, including 
  - non-linguistic representation of information

- **NLG output**
  - a text, plain or formatted (HTML, JSML) 
  - may be combined with graphics, tables etc.

- Knowledge sources required
  - domain-specific knowledge 
  - language-specific knowledge 
  - knowledge about human communication
Template- v. plan-based NLG

- “Template-based” NLG
  - Canned texts with variable slots
  - String manipulation (often in Perl)
  - Single-shot processing
- “Plan-based” NLG
  - Text planning and sentence planning
  - Tree structure transforms (often in Java)
  - Multi-stage, multi-level processing

Tasks of plan-based NLG

- Content determination
- Discourse planning
- Lexicalization
- Referring expression generation
- Sentence aggregation
- Surface realization
NLG: Content Determination

- Deciding what to say
  - Construct a set of MESSAGES from the data source
  - A message may become a word, phrase or sentence
  - Messages are based on domain entities (concepts)

\[
\text{IDENTITY}(\text{NEXTSHIP}, \text{MS-LILLY}) \rightarrow \text{The next ship is the MS-LILLY.}
\]

\[
\text{DEPARTURETIME}(\text{MS-LILLY}, 1000) \rightarrow \text{The MS-LILLY departs at 10am.}
\]

\[
\text{COUNT}(\text{SHIP}, \text{SOURCE(HAMBURG)}, \text{DESTINATION(COPENHAGEN)}, 5, \text{PERDAY})
\rightarrow \text{There are 5 ships daily from Hamburg to Copenhagen}
\]

NLG: Discourse Planning

- Organize messages in a coherent text plan
  - A text is structured, not random sentences
  - Conceptual grouping, rhetorical relations

\[
\text{There are five ships daily from Hamburg to Copenhagen. The next ship is the MS-LILLY. It departs at 10am.}
\]
NLG: Lexicalization

- Mapping from domain concepts to lexemes
- Determines the content words to be used
  - discourse focus - buy vs sell
  - collocations - exert influence, administer punishment
  - lexical semantics - male unmarried adult vs bachelor
  - basic level categories - dog vs poodle
  - attitude - house vs home, father vs dad
  - idioms - kick the bucket
- Partly determines the syntactic structure

NLG: Referring Expressions

- Allow the hearer to identify discourse objects
- Kinds of referring expressions
  - Proper names - Hamburg, The United States
  - Definite descriptions - the ship that leaves at 10
  - Proforms - it, later, there
- Initial reference
  - use a full name - the MS-LILLY
  - relate to a salient object - the ship’s snack bar
  - specify physical location - the ship at pier 12
NLG: Sentence Aggregation

- Mapping from messages to sentences
  - One-to-one mapping results in poor text
  - Need to combine messages in complex sentences
- Without aggregation
  - The next ship is the MS-LILLY. It leaves Hamburg at 10am. It has a restaurant. It has a snack bar.
- With aggregation
  - The next ship, which leaves Hamburg at 10am, is the MS-LILLY. It has a snack bar and a restaurant.

NLG: Surface Realization

- Converts text specifications into output text
- Generates grammatically correct text
  - insert function words - *he wants to book a ticket*
  - word inflection - *like + ed* → *liked*
  - grammatical word order
- Techniques for grammatical knowledge
  - declarative bidirectional grammars
  - grammars designed for generation
  - templates, easy and fast to implement
**Deep v. Shallow Generation**

- Deep generation tries to model everything
  - Research paradigm in NLG
  - Aims at general solutions, re-usable and scalable
  - Works for small domains, limited linguistic coverage
- Shallow generation aims at real applications
  - Commit to domain and task-specific choices
  - No general solutions attempted
  - Coverage based on domain corpus
  - Allow different methods within the same system
- Shallow generation: Stephan Busemann (DFKI)
Part 2: XML-based NLG

- Templates
  - Implement NLG template-based generation using predefined XSLT templates
- Trees
  - Implement NLG tree structures (text plans, text specifications) using XML tree structures
- Pipelines
  - Implement NLG pipeline architecture using piped sequences of XML-to-XML transformations

A Step-by-Step Example

- Simplified (rapid prototyping)
  - Quickly-developed “first prototype” generator for USIX-Interact spoken dialogue project
  - Dialogue response planning and generation
- XML + XSLT implementation
  - Agenda of domain concepts
  - Template-based text plans (“aggregations”)
  - Pipeline of XSLT transformations
Response planning

- New Information (NewInfo)
  - Response planning starts from NewInfo
  - Response always includes NewInfo

- Topic
  - NewInfo may be wrapped by Topic link
  - Polite response: Topic and NewInfo
  - Elliptical response: NewInfo only
Generation from NewInfo (Ex.1)

- *Which bus goes to Malmi?*
  - Topic: transportation: bus
  - Topic: destination: Malmi
  - NewInfo: bus number: 74
- *Number 74.*

---

Input: Agenda in XML

- Unordered set of domain concepts
  - Marked as “Topic” or “NewInfo”
- Specified by Dialogue Manager
- Starting point for Generator
  - Shared XML representation
Input: Agenda (Ex.1)

```xml
<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="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```

Template-based NLG in XSLT

- Create predefined XSLT named templates
  - Decide a set of predefined Text Plan tree structures (here, the structures are called “aggregations”)
  - Define one named template per Text Plan aggregation
- XSLT processing
  - Test the concept values given in Agenda
  - Select a Text Plan based on concept values
  - The named template for the Text Plan copies Agenda concepts into its slots using `<xsl:copy-of>`
Text Planning: Select the Plan

```
<xsl:template match="agenda">
  <xsl:choose>
    <xsl:when test="concept[@info='NewInfo']/type='transportation'>
      <xsl:call-template name="BY-TRANSPORT"/>
    </xsl:when>
    <xsl:when test="concept[@info='NewInfo']/type='bus'">
      <xsl:choose>
        <xsl:when test="concept[@info='NewInfo']/type='busnumber'">
          <xsl:call-template name="NUM-DEST-TIME"/>
        </xsl:when>
        <xsl:when test="concept[@info='NewInfo']/type='busnumber'">
          <xsl:call-template name="NUMBER-ONLY"/>
        </xsl:when>
      </xsl:choose>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

Text Planning: Insert Concepts

```
<xsl:template name="NUM-DEST-TIME">
  <aggregation>
    <subject cat="NP">
      <xsl:copy-of select="./concept[type='busnumber']"/>
    </subject>
    <predicate cat="VP">
      <xsl:copy-of select="./concept[type='bus']"/>
    </predicate>
    <complement cat="PP">
      <xsl:copy-of select="./concept[type='destination']"/>
    </complement>
    <adjunct cat="PP">
      <xsl:copy-of select="./concept[type='bustime']"/>
    </adjunct>
  </aggregation>
</xsl:template>
```
**Text Planning: Text Plan (Ex. 1)**

```xml
<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>
```

- Text Plan

```xml
<aggregation id="1">
  <subject cat="NP">
    <concept info="NewInfo">
      <type>busnumber</type>
      <value>74</value>
    </concept>
  </subject>
</aggregation>
```

**NLG Pipeline in XML**

- Pipeline architecture
  - XML input (agenda), XML output (JSML)
  - Sequence of XML-to-XML transformations
  - agenda → aggregation → speech markup
- Tree transformations in DOM or XSLT
  - Modify tree structures and nodes in them
  - For example (in Referring Expressions stage): replace domain concepts in aggregation tree by linguistic referring expressions
Microplanning: NewInfo

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

Microplanning: Pronouns

<!-- REFERRING EXPRESSIONS: PRONOUNS -->
<xsl:template match="concept[@info= 'Topic']">
  <xsl:choose>
    <xsl:when test="type= 'busnumber' ">
      <word>it</word>
    </xsl:when>
    <xsl:when test="type= 'destination' ">
      <word>there</word>
    </xsl:when>
    <xsl:when test="type= 'bustime' ">
      <word>then</word>
    </xsl:when>
  </xsl:choose>
</xsl:template>
Microplanning: Text Spec (Ex.1)

- Text Plan

  `<aggregation id="1">
  <subject cat="NP">
    <concept info="NewInfo">
      <type>busnumber</type>
      <value>74</value>
    </concept>
  </subject>
  </aggregation>`

- Text Specification

  `<aggregation id="1">
  <subject cat="NP">
    <word>number</word>
    <word>74</word>
  </subject>
  </aggregation>`

Realization: Output (Ex.1)

- Text Specification

  `<aggregation id="1">
  <subject cat="NP">
    <word>number</word>
    <word>74</word>
  </subject>
  </aggregation>`

- Output

  `<jsml lang="en">
  <div type="sent">
    number 74
  </div>
  </jsml>`

  (Java Speech Markup Language)
Another Example (Ex.2)

- *How do I get to Malmi?*
  - Topic: destination: Malmi
  - NewInfo: transportation: bus
  - NewInfo: bus number: 74
- *By bus – number 74 goes there.*

Input: Agenda (Ex.2)

```xml
<agenda id="2">
  <concept info="NewInfo">
    <type>transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="NewInfo">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```
Text Planning: Text Plan (Ex.2)

```xml
<agenda id="2">
  <concept info="NewInfo">
    <type>transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="NewInfo">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>bustype</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>bustype</type>
    <value>busnumber</value>
    <value>74</value>
  </concept>
</agenda>
```

Microplanning: Text Spec (Ex.2)

```xml
<aggregation id="2">
  <adjunct cat="PP">
    <concept info="NewInfo">
      <type>transportation</type>
      <value>bus</value>
    </concept>
  </adjunct>
  <subject cat="NP">
    <concept info="NewInfo">
      <type>busnumber</type>
      <value>74</value>
    </concept>
  </subject>
  <predicate cat="VP">
    <concept info="NewInfo">
      <type>bus</type>
      <value>exists</value>
    </concept>
    <complement cat="PP">
      <concept info="Topic">
        <type>destination</type>
        <value>malmi</value>
      </concept>
    </complement>
  </predicate>
</aggregation>
```
Realization: Output (Ex.2)

```
<aggregation id="2">
  <adjunct cat="PP">
    <word>by</word>
  </adjunct>
  <prosody cat="pause"/>
  <subject cat="NP">
    <word>number</word>
    <word>74</word>
  </subject>
  <predicate cat="VP">
    <word cat="V">
      <lexeme>go</lexeme>
      <features>3sg</features>
    </word>
  </predicate>
  <complement cat="PP">
    <word>there</word>
  </complement>
</aggregation>
```

Output

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

Part 3: Demos

- Voice NLG
  - Bilingual Finnish/English generator
  - FreeTTS Java speech synthesizer
  - Currently has only English voice

- Web NLG
  - Bilingual Finnish/English generator
  - Cocoon XML server implementation
  - Towards the Semantic Web
Demo: Voice NLG

- Implemented in Java
  - Java API for XML (JAXP) to execute XSLT
  - Java Speech API to execute speech synthesis
- XML-based NLG
  - Input: Annotation Graph in XML
  - Output: bilingual Finnish and English in JSML
- Java speech synthesis
  - Input: (currently) English in JSML
  - Output: speech

Annotation Graphs

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<AGSet id="AGSet" version="1.0"
   xmlns="http://www.ldc.upenn.edu/atlas/ag/
   xmlns:xlink="http://www.w3.org/1999/xlink"
<AG id="AGSet_AG1" type="transcription" timeline="AGSet_Timeline1">
<Anchor id="AGSet_AG1_anchor1" offset="1" unit="index"/>
<Anchor id="AGSet_AG1_anchor2" offset="2" unit="index"/>
<Annotation type="DAct" id="AGSet1_AG1_annotation1"
   start="AGSet1_AG1_anchor2" end="AGSet1_AG1_anchor2">
   <Feature name="score">1.0</Feature>
   <Feature name="name">ALOITUS</Feature>
</Annotation>
</AGSet_AG1>
</AGSet>
```
FreeTTS

- A speech synthesizer in Java
  - Free open source from Sun
  - Based on Flite and Festival (Edinburgh)
  - Portable (needs Java 1.4)
  - JSAPI interface
- Current limitations (FreeTTS 1.1)
  - Supplied voices: English
  - JSML accepted but not applied

Demo: Web NLG

- NLG web server
  - XML-based NLG
- Cocoon implementation
  - Cocoon web publishing framework (Java)
  - Runs in a servlet engine (Tomcat: Java)
  - Uses XSLT transformers (Xalan: Java)
  - Configurable XSLT pipelines
What is Cocoon?

“Apache Cocoon is an XML publishing framework that raises the usage of XML and XSLT technologies for server applications to a new level. Designed for performance and scalability around pipelined SAX processing, Cocoon offers a flexible environment based on a separation of concerns between content, logic, and style. To top this all off, Cocoon's centralized configuration system and sophisticated caching help you to create, deploy, and maintain rock-solid XML server applications.”

- Apache Cocoon website

Configurable pipelines

- Pipelines specified in Cocoon configuration file

```xml
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <!-- English examples: Jokinen & Wilcock (SIGDIAL-2001) -->
  <!-- Get an agenda, apply XSLT transforms, serialize as HTML -->
  <map:pipeline>
    <map:match pattern="generate(agendas/*.xml)">
      <map:generate src="agendas/{1}.xml"/>
      <map:transform src="transforms/SIGDIAL-2001/aggregation.xsl"/>
      <map:transform src="transforms/SIGDIAL-2001/lexicalization.xsl"/>
      <map:transform src="transforms/SIGDIAL-2001/realization.xsl"/>
      <map:serialize type="html"/>
    </map:match>
  </map:pipeline>
</map:sitemap>
```
Adding a Finnish pipeline

<! -- Finnish pipeline: Interact Demo 2002 -->
<! -- Get an A-Graph, apply XSLT transforms, serialize as HTML -->
<map:pipeline>
  <map:match pattern="suomeksi(agraphs/*.*xml)"/>
  <map:generate src="agraphs/\{1}.xml"/>
  <map:transform src="transforms/InteractDemo/AG-GetContent.xsl"/>
  <map:transform src="transforms/InteractDemo/AG-ResponsePlanner.xsl"/>
  <map:transform src="transforms/InteractDemo/FI-Lexicalization.xsl"/>
  <map:transform src="transforms/InteractDemo/FI-ReferringExps.xsl"/>
  <map:transform src="transforms/InteractDemo/FI-SurfaceRealizer.xsl"/>
  <map:transform src="transforms/displayverbatim.xsl"/>
  <map:serialize type="html"/>
</map:match>
</map:pipeline>

Adding an English pipeline

<! -- English pipeline: Interact Demo 2002 -->
<! -- Get an A-Graph, apply XSLT transforms, serialize as HTML -->
<map:pipeline>
  <map:match pattern="english(agraphs/*.*xml)"/>
  <map:generate src="agraphs/\{1}.xml"/>
  <map:transform src="transforms/InteractDemo/AG-GetContent.xsl"/>
  <map:transform src="transforms/InteractDemo/AG-ResponsePlanner.xsl"/>
  <map:transform src="transforms/InteractDemo/EN-Lexicalization.xsl"/>
  <map:transform src="transforms/InteractDemo/EN-ReferringExps.xsl"/>
  <map:transform src="transforms/InteractDemo/EN-SurfaceRealizer.xsl"/>
  <map:transform src="transforms/displayverbatim.xsl"/>
  <map:serialize type="html"/>
</map:match>
</map:pipeline>
From Domains to Languages

- 1 Domain, 2 Languages

- 2 Domains, 1 Language

Towards the Semantic Web

- 2 Ontologies, 1 Language