A complete integrated NLG system using AI and NLU tools
Laurence Danlos, Adil El Ghali

To cite this version:
Laurence Danlos, Adil El Ghali. A complete integrated NLG system using AI and NLU tools. Proceedings of COLING’02, 2002, Taiwan. halshs-00082842

HAL Id: halshs-00082842
https://halshs.archives-ouvertes.fr/halshs-00082842
Submitted on 28 Jun 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A complete integrated NLG system using AI and NLU tools

Laurence Danlos
LATTICE
U. Paris 7, Case 7003
2, place Jussieu
75251 Paris Cedex 05
France
danlos@linguist.jussieu.fr

Adil El Ghali
LATTICE
U. Paris 7, Case 7003
2, place Jussieu
75251 Paris Cedex 05
France
adil@linguist.jussieu.fr

Abstract
A standard architecture for an NLG system has been defined in (Reiter and Dale, 2000). Their work describes the modularization of an NLG system and the tasks of each module. However, they do not indicate what kind of tools can be used by each module. Nevertheless, we believe that certain tools widely used by the AI or NLU community are appropriate for NLG tasks. This paper presents a complete integrated NLG system which uses a Description logic for the content determination module, Segmented Discourse Representation Theory for the document structuring module and a lexicalized formalism for the tactical component. The NLG system, which takes into account a user model, is illustrated with a generator which produces texts explaining the steps taken by a proof assistant.

1 Introduction
The standard architecture of an NLG system proposed in (Reiter and Dale, 2000) is represented schematically in Figure 1.1. The tool used by a module and the data structure of its output are not precisely defined. According to Reiter and Dale, they vary from one author to the other. However, we believe that certain tools widely used by the AI or NLU community are appropriate for NLG tasks. Therefore, we reformulate in more specific terms Figure 1.1 as Figure 1.2.

The paper describes the modules of Figure 1.2: Section 3 justifies the use of a description logic for the content determination task and its output, a “message”; Section 4 specifies the use of SDRT for the document structuring task and its output, a “document plan”; Section 5 exposes briefly the use of a lexicalized formalism in the tactical component. Each section is illustrated with GePhoX, a generator that produces texts which explain the steps taken by PhoX, a proof assistant (Raffalli and Roziere, 2002). GePhoX is presented in Section 2.

As this paper intends to present a complete NLG system, there is no room for explaining each module in detail. We refer the reader to (El Ghali, 2001) for the content determination module, to (Danlos et al., 2001) for the document structuring module and to Danlos (1998; 2000) for the lexicalized tactical component. These
modules have been built more or less independently from each other, but with the same underlying idea: adaptation of NLU/AI theories and tools to NLG. They are now integrated in a complete model, which is presented here and illustrated with GePhoX, a generator whose implementation is in progress.

2 GePhoX

PhoX is an extensible proof assistant based on higher order logic, which has been developed to help mathematicians building proofs and teaching mathematics. Like other proof assistants, PhoX works interactively. The user (a mathematician) first gives the theorem to be proven (a goal). PhoX returns a list of subgoals which should be easier to prove than the initial goal. The user enters a command to guide PhoX in choosing or achieving a subgoal. The proof is thus computed top-down from goals to evidences. The user’s commands form a Proof script. PhoX’s output is a list of successive goals equivalent to a Proof tree.

Both the Proof script and PhoX’s output are difficult to read (even for a mathematician), as the reader can see for himself in Table 1 and Table 2. Hence, the need of an NLG system in order to obtain an easy-to-read version of the proof.

GePhoX is given as input both the Proof script and the Proof tree. It is one of the main original proposals in our generator (similar generators, such as PROVERB (Huang and Fiedler, 1997), take as input only the Proof tree). It makes it possible for GePhoX to start from an incomplete proof and produce texts during the interactive session. These texts help the mathematician user: before entering a new command in the Proof script, he can read a text reminding himself what he has been doing so far. The Proof script is also useful for identifying the reasoning strategies that have been used (reasoning by contradiction or induction), while it is very hard (if not impossible) to retrieve this information from a Proof tree with its numerous deduction steps.

Another originality of GePhoX is that it takes into account the knowledge of the user who can be either a mathematician using PhoX or a person more or less novice in mathematics. For the same proof, GePhoX can generate several texts according to the user model.

3 Using a description logic (DL)

The knowledge representation system KL-ONE (Branchman et al., 1979) was the first DL. It was created to formalize semantic networks and frames (Minsky, 1974). It introduces the notions of TBoxes and ABoxes respectively for terminological and assertional knowledge. KL-ONE has been widely used in the NLG community to formalize the domain model. On the other hand, this is not the case for the more recent DLs. Nevertheless, they present at least two advantages compared to KL-ONE: 1) for a large variety of DLs, sound and complete algo-
rithms have been developed for main inference problems such as subsumption, concepts satisfiability and consistency (Donini et al., 1996); 2) the relations between instances and classes are well defined for all the constructors, and their mathematical and computational properties have been studied in detail (Horrocks et al., 2000). So we believe that DLs are appropriate for the content determination task as shown in 3.3. Let us first briefly present DLs.

3.1 A brief introduction to DL

The three fundamental notions of DLs are individuals (representing objects in the domain), concepts (describing sets of individuals), and roles (representing binary relations between individuals or concepts). A DL is characterized by a set of constructors that allow us to build concepts/roles from atomic ones.

Examples of constructors

<table>
<thead>
<tr>
<th>Constructor (abbreviation)</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>A</td>
</tr>
<tr>
<td>top</td>
<td>⊤</td>
</tr>
<tr>
<td>bottom</td>
<td>⊥</td>
</tr>
<tr>
<td>conjunction (U)</td>
<td>C ⊃ D</td>
</tr>
<tr>
<td>complement (C)</td>
<td>¬C</td>
</tr>
<tr>
<td>univ. quant.</td>
<td>∀R.C</td>
</tr>
<tr>
<td>exist. quant. (E)</td>
<td>∃R.C</td>
</tr>
<tr>
<td>numeral restrictions (N)</td>
<td>≥ n R.C</td>
</tr>
<tr>
<td>collection of individuals (O)</td>
<td>{a₁, ..., aₙ}</td>
</tr>
</tbody>
</table>

The choice of constructors is domain dependent. Constructors other than those used in GePhoX (e.g., temporal extension) can be used for other domains (e.g., domains with non-trivial temporal information), without altering the mathematical and computational properties.

3.2 Domain and user models in DL

The Domain model is the set of concepts and roles necessary to express the input of the generator. More formally, let \( T_D \) be a TBox, such that each input \( I \) can be described by means of an ABox \( A_D \) corresponding to \( T_D \). The knowledge base \( \Sigma_D = (T_D, A_D) \) is called knowledge base for the domain and noted dkb. The User model is a knowledge base \( \Sigma_U = (T_U, A_U) \) such that \( T_U \) and \( A_U \) are respectively subsets of \( T_D \) and \( A_D \). \( \Sigma_U \) is noted ukb. Table 4 shows a part of the dkb for GePhoX.

Table 3: Syntax of standard constructors

<table>
<thead>
<tr>
<th>Constructor (abbreviation)</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>A</td>
</tr>
<tr>
<td>top</td>
<td>⊤</td>
</tr>
<tr>
<td>bottom</td>
<td>⊥</td>
</tr>
<tr>
<td>conjunction (U)</td>
<td>C ⊃ D</td>
</tr>
<tr>
<td>complement (C)</td>
<td>¬C</td>
</tr>
<tr>
<td>univ. quant.</td>
<td>∀R.C</td>
</tr>
<tr>
<td>exist. quant. (E)</td>
<td>∃R.C</td>
</tr>
<tr>
<td>numeral restrictions (N)</td>
<td>≥ n R.C</td>
</tr>
<tr>
<td>collection of individuals (O)</td>
<td>{a₁, ..., aₙ}</td>
</tr>
</tbody>
</table>

Examples of concepts with their semantic

<table>
<thead>
<tr>
<th>IMPLIES, PROVES</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x,y / IMPLIES(x,y)} : x implies y</td>
</tr>
<tr>
<td>{x,y / PROVES(x,y)} : x proves y</td>
</tr>
</tbody>
</table>

The choice of constructors is domain dependent. Constructors other than those used in GePhoX (e.g., temporal extension) can be used for other domains (e.g., domains with non-trivial temporal information), without altering the mathematical and computational properties.

3.3 Content determination tasks

The content determination module performs four tasks, as shown in Figure 2.

**Translation:** The input of the generator (assertional information) is first translated into concepts of the TBox. For that purpose, a correspondence is established between the elements of the input and concepts and roles in the DKB. The \( O \) constructor is used to keep information about the individuals occurring in the input. For example, command 2 in Table 1 with individual H is translated into the concept \( C_0 = \exists EliminationWell_founded.Hypothese \)
$\{H\}$, and commands 8 to 11 are translated into $C_1 = \exists \text{ByInduction} \{p\}$.

**Selection:** The selection task consists of choosing the relevant concepts among those constructed in the translation phase with regard to the UKB. For example, if $C_0$ is an unknown concept for the user, a concept $C$ must be looked up in the UKB such as $C \text{ approximates}^1 C_0$.

**Verification:** At this point, the coherence of all the concepts of the selection is verified. For example, if the user tries to reason by induction on a real number, GePhoX tells him that it is not possible.

**Instanciation:** With the information about individuals, which have been kept in the translation phase (with the use of the $O$ constructor), the instanciation task is straightforward. Table 5 shows some instanciated concepts for the Euclidian division.

As is well known, designing knowledge bases (DKB and UKB) and translating the input of the generator into concepts and roles of the DL is a difficult task which has to be fulfilled for every generator. However, with a DL, the selection, verification and instanciation tasks are domain independent: algorithms and their implementation are reusable. Moreover, when using a DL for the content determination task, the “message” is a first order logic formula (a standard representation shared by a large community) which takes into account the user knowledge and whose coherence has been checked.

### 4 Using SDRT for document structuring

In (Danlos et al., 2001) we advocate using SDRT (Segmented Discourse Representation Theory (Asher, 1993; Asher and Lascarides, 1998)) as a discourse framework, since SDRT and DRT (Discourse Representation Theory, (Kamp and Reyle, 1993)) are the most popular frameworks for formal and computational semantics. Let us briefly present SDRT.

#### 4.1 A brief introduction to SDRT

SDRT, designed first for text understanding, was introduced as an extension of DRT in order to account for specific properties of discourse structure. SDRT can be viewed as a super-layer on DRT whose expressiveness is enhanced by the use of discourse relations. Thus the DRT structures (Discourse Representation Structures or DRS) are handled as basic discourse units in SDRT.

DRSs are “boxed” first order logic formulae. Formally, a DRS is a couple of sets $(U, Con)$. $U$ (the universe) is the set of discourse referents. $Con$ contains the truth conditions representing the meaning of the discourse.

A SDRS is a pair $(U,Con)$, see Figure 3. $U$ is a set of labels of DRS or SDRS which can be viewed as “speech act discourse referents” (Asher and Lascarides, 1998). $Con$ is a set of conditions on labels of the form:

- $\exists p_1 \in \text{Entier named}(p_1, p)$
  - $\text{choose}(\text{user}, p_1)$
- $\exists d_1 \in \text{EntierNonNul named}(d_1, d)$
  - $\text{choose}(\text{user}, d_1)$
- $\exists f_1 \in \text{Formula}$
  - $\text{constant}(f_1, \exists q,r: N \ (r < d \land p = q.d + r))$
  - $\text{prove}(\text{user}, f_1)$
  - $\text{induction}(f_1, p_1)$

... 

Table 5: DL-Message for Euclidian division

---

1Given two TBoxes $T$ and $T'$ with $T \subset T'$ and a concept $C \in T - T'$, $C' \in T'$ approximates $C$ if $C$ minimally subsumes $C'$ or $C'$ minimally subsumes $C$. 

...
- \(R(\pi_1, \pi_j)\), where \(\pi_i\) and \(\pi_j\) are labels and \(R\) a discourse relation. Discourse relations are inferred non-monotonically by means of a defeasible glue logic exploiting lexical and world knowledge.

- or it is expressed through a discourse relation; that is the case for \(condition_3\) with \(R_1(\pi_1, \pi_2) \rightarrow condition_3\), which means that \(R_1(\pi_1, \pi_2)\) must have \(condition_3\) among its consequences: no other element is in charge of expressing \(condition_3\).

To establish discourse relations, the SDRT conditions are reversed. As an illustration, in SDRT for text understanding, there is the Axiom (1) for Narration. This axiom states that if \(Narration\) holds between two SDRSs \(\pi_1\) and \(\pi_2\), then the main event (\(me\)) of \(\pi_1\) happens before the main event of \(\pi_2\).

\[\square(Narration(\pi_1, \pi_2) \rightarrow me(\pi_1) < me(\pi_2))\]

For text generation, this axiom is reversed as shown below (Roussarie, 2000, p. 154):

- If \(k_1\) and \(k_2\) are DRSS whose main eventualities are not states,
- and if the main event of \(k_1\) occurs before the main event of \(k_2\),
- then \(Narration(\pi_1, \pi_2)\) is valid when \(\pi_1\) and \(\pi_2\) respectively label \(k_1\) and \(k_2\).

As another example, the condition \(cause(e_1, e_2)\) can be expressed through Result(\(\pi_1, \pi_2\)) or Explanation(\(\pi_2, \pi_1\)) when \(\pi_1\) and \(\pi_2\) label the sub-DRSS that contain the descriptions of \(e_1\) and \(e_2\) respectively.

Let us now examine how we determine the universes of sub-DRSS, i.e. discourse referents, while observing two technical constraints, namely:

- the arguments of any condition in a sub-DRS must appear in the universe of this DRS;
- the universes of all the sub-DRSSs have to be disjoint. This constraint is the counterpart of the following constraint in understanding: “partial DRSS introduce new discourse referents” (Asher, 1993, p. 71).

These two constraints are not independent. Assuming that the first constraint is respected, the second one can be respected with the following mechanism: if a variable \(x\) already appears in a preceding sub-DRS labelled \(\pi_x\), then a new variable \(y\) is created in the universe of

---

**Figure 3:** SDRS for Max fell. John pushed him.

**4.2 Building a SDRS**

Starting from a “message” encoded into a logical form, the document structuring module builds a SDRS. On a first step, the logical form is translated into a DRS. In the case of a purely existential formula\(^2\), this amounts to putting all the variables into the universe of the DRS and splitting the formula into elementary conjoined conditions.

After this first step, the document structuring task amounts to building a SDRS from a DRS and to go on recursively on each embedded (s)DRS. This process is schematized below.

Let us first examine the principles governing the splitting of the conditions. All the conditions in the DRS have to be expressed in the SDRS. Two cases arise:

- either a condition in the DRS appears as a condition in one of the sub-DRS; that is the case for \(condition_1\) which appears in the sub-DRS labelled \(\pi_1\);
- or it is expressed through a discourse relation; that is the case for \(condition_3\) with \(R_1(\pi_1, \pi_2) \rightarrow condition_3\), which means that \(R_1(\pi_1, \pi_2)\) must have \(condition_3\) among its consequences: no other element is in charge of expressing \(condition_3\).

---

\(^2\)More complex formulas are not considered here.
the current sub-DRS labelled \( \pi_y \) and the condition \( y = x \) is added to the conditions of \( \pi_y \). The discourse referent \( y \) will be generated as an anaphora if \( \pi_y \) is available to \( \pi_y \) (Asher, 1993), otherwise it will be generated as a definite or demonstrative NP.

A document structuring module la SDRT based on the principles we have just exposed can be used for any generator (whose “message” is first order logic formula). The algorithm and the rules establish discourse relations (obtained by reversing the rules in NLU) are generic. See below an example of sDRS in GePhoX, the sDRS built from Table 5.

![Table 6: sDRS for Euclidian division](image)

### 5 Using a lexicalized grammar for the tactical component

Lexicalized grammars are commonly used in NLU and also in NLG (Stede, 1996). In Danelos (1998; 2000) we propose a lexicalized formalism, called G-TAG, for the tactical component of an NLG system. It is modularized into a microplanner which produces a semantic dependency tree and a surface realizer which produces the text (see Figure 1.2).

The surface realizer is designed to use the syntactic and lexical information of a lexicalized TAG grammar. The TAG grammar is extended to handle multi-sentential texts and not only isolated sentences.

The microplanner is based on a lexicalized conceptual-semantic interface. This interface is made up of concepts; each concept is associated with a lexical database. In our model, a concept is either a term in the TBox or a discourse relation. A lexical database for a given concept records the lexemes lexicalizing it with their argument structure, and the mappings between the conceptual and semantic arguments. The process of generating a semantic dependency tree from a sDRS \( \langle U, \text{Con} \rangle \) is recursive:

- An element \( \pi_i \) in \( U \) is generated as a clause if \( \pi_i \) labels a sDRS and recursively as a text (possibly a complex sentence) if \( \pi_i \) labels a sDRS.
- A condition \( R(\pi_i, \pi_j) \) in \( \text{Con} \) is generated as a text “\( S_i \), Cue \( S_j \)” or as a complex sentence “\( S_i \) Cue \( S_j \)” where \( S_i \) generates \( \pi_i \), \( S_j \) \( \pi_j \), and Cue is a cue phrase which is encoded in the lexical database associated with \( R \) (Cue may be empty).
- A condition \( \pi : K \) in \( \text{Con} \) where \( K \) is a sDRS \( \langle U, \text{Con} \rangle \) is generated as a clause according to the following constraints (which are the counterparts of constraints in understanding):
  - A discourse referent in \( U \) is generated as an NP or a tensed verb.
  - Conditions guide lexical choices. Conditions such as \( x = \text{John} \) correspond to proper nouns. Equality conditions between discourse referents (e.g. \( x = y \)) give rise to (pronominal or nominal) anaphora. The other conditions, e.g. \( \text{prove}(e_1, x, y) \), are lexicalized through the lexical data base associated with the concept (prove).

The surface realizer, based on a TAG grammar, is a set of lexical data bases. A data base for a given lexical entry encodes the syntactic structures realizing it with their syntactic arguments. With such a TAG grammar and a morphological module, the text is computed in a deterministic way from the semantic dependency tree.

### 6 Conclusion

Since NLG is a subfield of NLP, which is itself a subfield of AI, it seems to be a good idea to reuse tools developed by the NLP or AI community. We have shown in this paper how to integrate DL, SDRT, and a lexicalized grammar into an NLG system, while following the standard pipelined architecture.

---

3 Some authors (de Smedt et al., 1996) have made justified criticisms of the pipelined architecture. However, we decided to keep it for the time being.
Table 7: A Text of proof for Euclidian division

Theorem.
\[ \forall p, d \in \mathbb{N} (d \neq 0 \rightarrow \exists q, r \in \mathbb{N} (r < d \land p = q \cdot d + r)) \]

Proof. Let us choose \( p, d \) two natural numbers with \( d \neq 0 \). By induction on \( p \) we prove \( \exists q, r \in \mathbb{N} (r < d \land p = q \cdot d + r) \). Let us prove a \( a \) strictly positive natural. We assume

\[ \forall b \in \mathbb{N} (b < a \rightarrow \exists q, r \in \mathbb{N} (r < d \land b = q \cdot d + r)) \]

and we must prove \( \exists q, r \in \mathbb{N} (r < d \land a = q \cdot d + r) \).

We distinguish two cases: \( a < d \) and \( d \leq a \). In the first case, we choose \( q = 0 \) and \( r = a \). In the second case, we take \( a' = a - d \). Using the induction hypothesis on \( a' \), we find two naturals \( q, r \) such that \( r < d \) and \( a' = q \cdot d + r \). We take \( S \) and \( r \) as quotient and remaining for the division of \( a \). We must prove \( a = S \cdot q \cdot d + r \) which is immediate.

Table 7: A Text of proof for Euclidian division

GePhoX illustrates the applicability of our system. It is currently being implemented in Java. The development of the document planner of GePhoX is work in progress. The goal is to interface this module with CLEF (Meunier and Reyes, 1999), an implementation of g-TAG. We intend to produce a text as shown in Table 7.

References


