Natural Language Generation Applied to Intelligent Query Interfaces

Paolo Dongilli

February 18, 2008

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

1.2 Main ideas

1.3 Overview of the thesis

Chapter 2

Ontologies

2.1 Introduction to ontologies

2.2 Languages for expressing ontologies

2.3 Description Logics

2.4 Reasoning over ontologies

2.5 $\mathcal{SHIQ}$ knowledge bases

We introduce the DL $\mathcal{SHIQ}$ along with its syntax and semantics. $\mathcal{SHIQ}$ is an extension of the DL $\mathcal{ALC}$ adding role transitivity ($\mathcal{S}$), inverse roles ($\mathcal{I}$), role hierarchies ($\mathcal{H}$), and qualified number restrictions ($\mathcal{Q}$).

Definition 1 ($\mathcal{SHIQ}$ knowledge base)**.** Let $\mathsf{N_C}$ be a set of concept names and $\mathsf{N_R}$ a set of role names with a subset $\mathsf{N_{R+}} \subseteq \mathsf{N_R}$ of transitive role names. The set of roles is $\mathsf{N_R} \cup \{R^- \mid R \in \mathsf{N_R}\}$. The two functions Inv and Trans defined on roles are introduced. Inv is defined as $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$ for any role name R. Trans is a boolean function, $\mathsf{Trans}(R) = \mathit{true}$ iff $R \in \mathsf{N_{R+}}$ or $\mathsf{Inv}(R) \in \mathsf{N_{R+}}$.

A *role inclusion axiom* is an expression of the form $R \sqsubseteq S$ where R and S are roles, each of which can be inverse. A *role hierarchy* is a set of role inclusion axioms. We define the relation $\sqsubseteq^*$ which denotes the transitive closure of $\sqsubseteq$ over a role hierarchy $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is a *sub-role* of S when $R \sqsubseteq^* S$, and a *super-role* of S when $S \sqsubseteq^* R$. A role is *simple* if its neither transitive nor has transitive sub-roles.

The set of $\mathcal{SHIQ}$ concepts is the smallest set such that:

- Every concept name is a concept,
- If C and D are concepts, R is a role, S is a simple role and n is a non-negative integer, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\geqslant nS.C$, $\leqslant nS.C$ are concepts.

A *concept inclusion axiom* is an expression of the form $C \sqsubseteq D$ for two concepts C and D. A *terminology* or T-Box is a set of concept inclusion axioms.

Let $\mathsf{N_I} = \{a, b, c \ldots\}$ be a set of individual names. An *assertion* is an expression that can have the form $C(a)$, $R(a, b)$ or $a \neq b$ where C is a concept, R is a role and $a, b \in \mathsf{N_I}$. An *A-Box* is a set of assertions.

A $\mathcal{SHIQ}$ knowledge base (KB) is a triple $K = \langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$, where $\mathcal{A}$ is an A-Box, $\mathcal{R}$ is role hierarchy and $\mathcal{T}$ is a terminology. The *statements* contained in $\mathcal{T}$ and $\mathcal{R}$ are called *terminological* while the ones in $\mathcal{A}$ are called *assertional*.

The semantics of $\mathcal{SHIQ}$ knowledge bases is given by means of interpretations.

Definition 2 (Interpretation)**.** An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is defined for a set of individual names $\mathsf{N_I}$, a set of concepts $\mathsf{N_C}$ and a set of roles $\mathsf{N_R}$. The set $\Delta^{\mathcal{I}}$ is called *domain* of $\mathcal{I}$. The valuation $\cdot^{\mathcal{I}}$ maps each individual name in $\mathsf{N_I}$ to an element in $\Delta^{\mathcal{I}}$, each concept in $\mathsf{N_C}$ to a subset of $\Delta^{\mathcal{I}}$, and each role in $\mathsf{N_R}$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Additionally, for any concepts C, D, any role R and any non-negative integer n, the valuation $\cdot^{\mathcal{I}}$ must satisfy the following equations, where :

$$
\begin{array}{rcll}
R^{\mathcal{I}} & = & (R^{\mathcal{I}})^{+} & \textit{for each role } R \in \mathsf{N_{R+}} \\
(R^{-})^{\mathcal{I}} & = & \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\} & \textit{(inverse roles)} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap C^{\mathcal{I}} & \textit{(conjunction)} \\
(C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup C^{\mathcal{I}} & \textit{(disjunction)} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & \textit{(negation)} \\
(\exists R.C)^{\mathcal{I}} & = & \{x \mid \textit{ for some } y, \langle x, y\rangle \in R^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\} & \textit{(exists restriction)} \\
(\forall R.C)^{\mathcal{I}} & = & \{x \mid \textit{ for all } y, \langle x, y\rangle \in R^{\mathcal{I}} \textit{ implies } y \in C^{\mathcal{I}}\} & \textit{(value restriction)} \\
(\geqslant n R.C)^{\mathcal{I}} & = & \{x \mid |\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\}| \geq n\} & \textit{(}\geqslant\textit{-number restriction)} \\
(\leqslant n R.C)^{\mathcal{I}} & = & \{x \mid |\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\}| \leq n\} & \textit{(}\leqslant\textit{-number restriction)}
\end{array}
$$

Definition 3 (Model of a knowledge base)**.** An interpretation $\mathcal{I}$ satisfies an assertion A iff:

$$
\begin{array}{rl}
a \in C^{\mathcal{I}} & \textit{if } A \textit{ is of the form } C(a) \\
\langle a, b\rangle \in R^{\mathcal{I}} & \textit{if } A \textit{ is of the form } R(a, b) \\
a^{\mathcal{I}} \neq b^{\mathcal{I}} & \textit{if } A \textit{ is of the form } a \neq b
\end{array}
$$

An interpretation $\mathcal{I}$ satisfies an A-Box $\mathcal{A}$ if it satisfies every assertion in $\mathcal{A}$. $\mathcal{I}$ satisfies a role hierarchy $\mathcal{R}$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for every $R \sqsubseteq S$ in $\mathcal{R}$. $\mathcal{I}$ satisfies a terminology $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every $C \sqsubseteq D$ in $\mathcal{T}$. $\mathcal{I}$ is a model of $K = \langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ if it satisfies $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{T}$.

The notation $\mathcal{I} \models \alpha$ is used to assert that an interpretation $\mathcal{I}$ satisfies a statement α. An interpretation $\mathcal{I}$ satisfies (or is a model for) a KB K iff it satisfies all the statements in K ($\mathcal{I} \models K$).

2.6 Introducing Concrete domains

(SHIQ(D)) For practical purposes in description logics we need to be able to represent *concrete* properties as e.g. name, age, height, with values taken from fixed domains as strings, integer or real numbers. Predicates on these domains are also needed in order to express, equality, inequality, or other complex predicates. The solution is to extend description logics with *concrete domains* [Baader and Hanschke, 1991].

Definition 4 (Concrete domain)**.** A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set, and $\Phi_\mathcal{D}$ is a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n, and an n-ary predicate $P^\mathcal{D} \subseteq \Delta^n_\mathcal{D}$.

Given m n-ary predicates P_i over $\mathcal{D}$ with $1 \leq i \leq m$, it is also possible to build a complex predicate derived from the conjunction ($\sqcap$) or disjunction ($\sqcup$) of the predicates, with the meaning that $(P_1 \sqcap/\sqcup P_2 \sqcap/\sqcup \cdots \sqcap/\sqcup P_n) = ((P_1)^\mathcal{D} \cap/\cup (P_2)^\mathcal{D} \cap/\cup \cdots \cap/\cup (P_n)^\mathcal{D}) \subseteq \Delta^n_\mathcal{D}$

A first example of concrete domain to illustrate Definition 4 is R, where as set Δ_R we use the real numbers $\mathbb{R}$. We consider the following predicates:

- unary predicates P_q for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and each $q \in \mathbb{R}$ with $(P_q)^\mathsf{R} = \{q' \in \mathbb{R} \mid q'Pq\}$;
- unary predicates $P'_qP''_r$ for each $P' \in \{>, \geq\}$, $P'' \in \{<, \leq\}$, and $q, r \in \mathbb{R}$ with $(P'_qP''_r)^\mathsf{R} = \{q' \in \mathbb{R} \mid q'P'q \wedge q'P''q\}$;

Example 1. *The concept representing all cars with a mileage lower than 70,000 would be:*

$$\mathsf{Car} \sqcap \exists \mathsf{mileage}.\, \leq_{70000}.$$

Example 2. *The cars with a price (in Euro) less than 10000, between 20000 and 40000, or greater than 60000 would be represented by this concept:*

$$\mathsf{Car} \sqcap \exists \mathsf{priceEUR}.(\leq_{10000} \sqcup \geq_{20000} \leq_{40000} \sqcup \geq_{60000}).$$

Another concrete domain is S, where $\Delta_\mathcal{S}$ is the set Σ^* of all finite strings of characters over some alphabet Σ. We just take into consideration two unary predicates:

- P_L for each $P \in \{\in, \notin\}$ and each $L \subseteq \Delta_\mathcal{S}$ with $(P_L)^\mathsf{S} = \{q' \in \Sigma^* \mid q'PL\}$;

Example 3. *Using the previous predicate, the off-roaders of make either "Land Rover", "Santana", or "Toyota" can be expressed with the concept:*

$$\mathit{Off\text{-}Roader} \sqcap \exists \mathit{make}.\, \in_{\{\text{"Land Rover"},\text{"Toyota"}\}}.$$

Now we give the formal definition of the description logic $\mathcal{SHIQ}(\mathcal{D})$, obtained extending the $\mathcal{SHIQ}$ DL described above with concrete datatypes. $\mathcal{SHIQ}$ will be augmented with

- *abstract features* which are roles interpreted as functional relations;
- *concrete features*, interpreted as partial functions from the logical domain into the concrete domain;
- a new concept constructor that allows to describe constraints on concrete values using predicates from the concrete domain.

Definition 5 ($\mathcal{SHIQ}(\mathcal{D})$ syntax)**.** Let $\mathsf{N_C}$, $\mathsf{N_R}$, $\mathsf{N_{cF}}$ be pairwise disjoint and countably infinite sets of *concept names*, *role names*, and *concrete features*. Furthermore, let $\mathsf{N_{aF}}$ be a countably infinite subset of $\mathsf{N_R}$. The elements of $\mathsf{N_{aF}}$ are called *abstract features*. A *path* u is a composition $f_1 \cdots f_n g$ of n abstract features $f_1, \ldots, f_n$ ($n \geq 0$) and a concrete feature g. For $\mathcal{D}$ a concrete domain, the set of $\mathcal{SHIQ}(\mathcal{D})$-concepts is the smallest set such that

- every concept name is a concept, and
- if C and D are concepts, R is a role name, g is a concrete feature, $u_1, \ldots, u_n$ are paths, and $P \in \Phi_{\mathcal{D}}$ is a predicate of arity n, then the following expressions are also concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\exists u_1, \ldots, u_n.P$, and $g\uparrow$.

$\top$ is used as abbreviation for an arbitrary propositional tautology and $\bot$ as abbreviation for $\neg\top$. Furthermore, if $u = f_1 \cdots f_k g$ is a path then $u\uparrow$ is used as abbreviation for $\forall f_1.\cdots.\forall f_k.g\uparrow$.

Let's extend now the $\mathcal{SHIQ}$ semantics presented in Definition 2.

Definition 6 ($\mathcal{SHIQ}(\mathcal{D})$ semantics)**.** The semantics of a $\mathcal{SHIQ}(\mathcal{D})$ knowledge base KB is given extending the interpretation function (or valuation) $\cdot^{\mathcal{I}}$ that has also to map

- each abstract feature f to a partial function $f^{\mathcal{I}}$ from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{I}}$, and
- each concrete feature g to a partial function $g^{\mathcal{I}}$ from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{D}}$.

If $u = f_1 \cdots f_k g$ is a path, then $u^{\mathcal{I}}(d)$ is defined as $g^{\mathcal{I}}(f_n^{\mathcal{I}} \cdots (f_1^{\mathcal{I}}(d)) \cdots)$. The interpretation function is extended to arbitrary concepts as:

$$\begin{aligned}(\exists u_1, \ldots, u_n.P)^{\mathcal{I}} &= \{d \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots x_n \in \Delta_{\mathcal{D}} : u_i^{\mathcal{I}}(d) = x_i \textit{ for } 1 \leq i \leq n \\ &\qquad \textit{and } (x_1, \ldots, x_n) \in P^{\mathcal{D}}\} \\ (g\uparrow)^{\mathcal{I}} &= \{d \in \Delta_{\mathcal{I}} \mid g^{\mathcal{I}}(d) \textit{ undefined }\}\end{aligned}$$

Chapter 3

Ontology-based querying

3.1 Conjunctive queries

Definition 7 (Querying a KB)**.** Querying a DL KB means verifying whether a given statement α (the query) is a logical consequence of the knowledge base (i.e. $K \models \alpha$). In other words the statement α is a logical consequence of K if it is satisfied in every interpretation $\mathcal{I}$ satisfying K ($K \models \alpha$ iff for any interpretation $\mathcal{I}$, $\mathcal{I} \models K$ implies $\mathcal{I} \models \alpha$).

Given a KB K, it is possible to query it using *conjunctive queries* which are defined below.

Definition 8 (Conjunctive Queries)**.** A *conjunctive query* can be represented as $\langle \vec{x} \rangle \leftarrow conj(\vec{x}, \vec{y})$, where $\vec{x}$ is the vector of so called *distinguished variables* that will be bound to individuals (single objects) of the knowledge base used to answer the query; $\vec{y}$ is the vector of non-distinguished variables (existentially quantified variables). $conj(\vec{x}, \vec{y})$ is a conjunction of terms of the form $v_1 : C$, $\langle v_2, v_3 \rangle : R$, or $v_4 : P$ where C is a concept name, R is a role name, P is a simple or complex predicate over a given concrete domain $\mathcal{D}$ and v_1, v_2, v_3, v_4 are variables from $\vec{x}$ or $\vec{y}$.

Example 4. *An example of conjunctive query in the automotive domain is the following:*

$$\begin{aligned}\langle \underline{x_3}, \underline{x_4}, \underline{x_8}, \underline{x_9} \rangle \leftarrow x1 : \mathit{Off\text{-}Roader} \sqcap \langle x_1, x_2 \rangle : \mathit{soldBy} \sqcap x_2 : \mathit{CarDealer} \sqcap \\ \sqcap \langle x_2, \underline{x_3} \rangle : \mathit{carDealerName} \sqcap \underline{x_3} : \mathit{String} \sqcap \langle x_2, \underline{x_4} \rangle : \mathit{carDealerAddress} \sqcap \\ \sqcap \underline{x_4} : \mathrm{String} \sqcap \langle x_2, x_5 \rangle : \mathit{locatedInCity} \sqcap x_5 : \mathit{City} \sqcap \langle x_5, x_6 \rangle : \mathit{cityName} \sqcap \\ \sqcap \underline{x_6} : \in_{\{\text{“Bolzano”}\}} \sqcap \langle x_1, x_7 \rangle : \mathit{hasMake} \sqcap x_7 : \mathit{Land\_Rover} \sqcap \langle x_1, \underline{x_8} \rangle : \mathit{hasMileage} \sqcap \\ \sqcap \underline{x_8} : \mathit{Float} \sqcap \underline{x_8} : \leq_{50000} \sqcap \langle x_1, \underline{x_9} \rangle : \mathit{hasPriceEUR} \sqcap \\ \sqcap \underline{x_9} : \mathit{Float} \sqcap \underline{x_9} : \geq_{10000} \leq_{20000} \end{aligned} \tag{3.1}$$

Underlined variables represent distinguished variables*.* String *and* Float *represent two concrete domains, while the others are abstract concepts.*

The query above is for an off-roader sold by a car-dealer located in a city whose name is “Bolzano”. The make must be Land_Rover, the mileage must be less

than 50000, and the price in Euro between 10000 and 20000. I want to know the off-roader's mileage and the price, the car dealer's name and address.

3.1.1 CQ answering

Definition 9 (Conjunctive Query Answering)**.** Given a query $q(\vec{x})$ where $\vec{x}$ are distinguished variables, and a KB K, answering $q(\vec{x})$ means returning all tuples $\vec{t}$ that substituted to $\vec{x}$ are such that $K \models q(\vec{t})$.

3.1.2 Query graphs

Definition 10 (Query Graphs)**.** A conjunctive query q can be represented by means of a directed labelled graph $G(q) := \langle V, E \rangle$ where V represents a set of vertices and E a set of edges. V is the union of the elements in $\vec{x}$, and $\vec{y}$; E is made up of all pairs $\langle v_1, v_2 \rangle$ where $v_1, v_2 \in V$ and $\langle v_1, v_2 \rangle : R$ is a term in q. A node $v \in V$ is labelled with a concept $C_1 \sqcap \cdots \sqcap C_n$ such that for every C_i, $v : C_i$ is a term of q. Optionally if node v is labelled with a concrete domain, it can also have an additional label containing the name of a (simple or complex) predicate. Every edge $e \in E$ is labelled with a set of role names $\{R \mid \langle v_1, v_2 \rangle : R \text{ is a term in } q\}$

We define the function $\mathcal{L}(v)$ which returns the label for $v \in V$. If $\mathcal{L}(v)$ is empty, the top concept ($\top$) is returned. The function $\mathcal{L}(e)$ ($e \in E$) returns a set of edge labels for e. The function $\mathcal{L}^-(e)$ ($e \in E$) returns instead, a set of inverted edge labels such that $\mathcal{L}^-(e) = \{R \mid R^- \in \mathcal{L}(e)\}$. The function $pred(v_1)$ ($v_1 \in V$) returns a set of vertices $\{v \mid v_1, v \in V \wedge \langle v, v_1 \rangle \in E\}$. Another function we define is $\mathcal{L}^P(v)$, which returns (if any) the predicate name associated to v, where $\mathcal{L}(v)$ must return a concrete domain name.

Two vertices $v_1, v_2 \in V$ are *adjacent*, if $\mathcal{L}(\langle v_1, v_2 \rangle) \neq \emptyset$ or $\mathcal{L}(\langle v_2, v_1 \rangle) \neq \emptyset$. The vertex $v_1 \in V$ is said to be *reachable* from $v_2 \in V$, if v_1 is adjacent to v_2 or if there exists another vertex $v_3 \in V$ such that v_3 is adjacent to v_1, and v_2 is reachable from v_3. A graph $G(q)$ is (cyclic), if there is a $v \in V$, such that $\mathcal{L}(\langle v, v \rangle) \neq \emptyset$ or if there is a $v' \in V$, such that v is adjacent to v' and if one element is removed from $\mathcal{L}(\langle v, v' \rangle)$, v' is still reachable from v.

q is an *acyclic conjunctive query* if $G(q)$ is not cyclic.

For a better graph readability, nodes are represented with a rectangle when $\mathcal{L}(v)$ is an abstract concept and with an oval when $\mathcal{L}(v)$ refers to a concrete domain.

Figure 3.1 shows the query graph corresponding to the query in Example 4.

Hereinafter we will use the terms query or conjunctive query referring always to *acyclic conjunctive queries*.

3.1.3 Query rolling-up and focused queries

We finally need to introduce a fundamental manipulation of the queries which enables us to exploit the reasoning services provided by a DL reasoner. This operation is the so called *rolling up* of an acyclic conjunctive query (see [Horrocks and Tessaris, 2002]).

Roughly speaking, the rolling up transforms an arbitrary query without cycles into an equivalent DL expression. The key idea behind is the fact that a (sub)query of the form $P(x, y), R(y)$ is equivalent to the DL expression $(\exists P.R)(x)$.

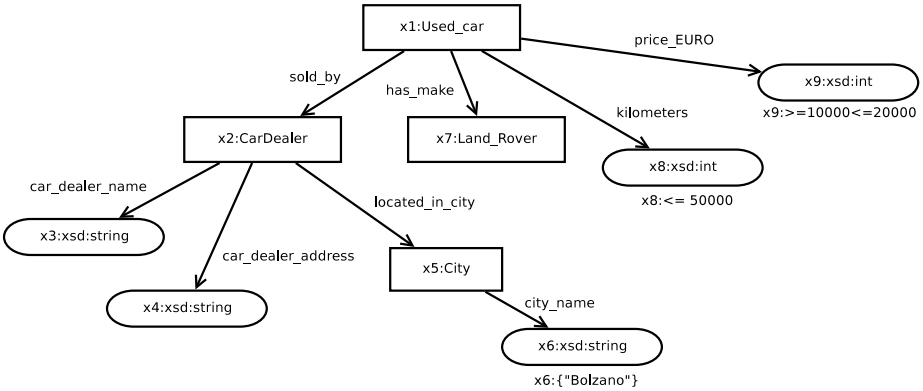


Figure 3.1: Example of query graph.

Any variable can be selected as the root of the tree (since we consider acyclic queries) and the rest of the query can be "rolled-up" starting from the leaves.[1]

To analyze the properties of a given query focused on a specific variable (a *focused query*), say q^x, we roll-up the query using the focus as the root (with variable x associated with concept F) and then we interrogate the reasoner using the resulting complex concept $Q^{F(x)}$.

In the following section and in [Zorzi *et al.*, 2007] we describe in detail how we implemented the previous ideas in an intelligent query interface that enables users to access heterogeneous data sources by means of an integrated ontology.

3.2 An intelligent Query Interface

We simply called it *Query Tool* and it was devised to enable users to access heterogeneous data sources by means of an integrated ontology. The Query Tool supports the users in formulating a precise conjunctive query, where the intelligence of the interface is driven by reasoning services running over a given logic-based ontology.

The ontology, which describes a given domain, defines a vocabulary which is richer than the logical schema of the underlying data, and it is meant to be closer to the user's wide vocabulary. The user can exploit the ontology's entities to formulate the query, and she is guided by such a richer vocabulary in order to understand how to express her information needs more precisely, given the knowledge of the system. This latter task —called intensional navigation— is the most innovative functional aspect of our interface. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related to the lack of schema comprehension and enabling her to easily formulate meaningful queries. Queries can be specified through an iterative refinement process supported by the ontology via intensional navigation. The user may specify her request using generic terms, refine

[1] Inverse roles provide the possibility of collapsing queries of the form $P(y, x), R(y)$ as well. If the ontology doesn't include transitive roles and nominals, cyclic queries can be handled in the same way (see [Horrocks *et al.*, 2000]). We're considering techniques to allow more expressive languages.

some terms of the query or introduce new terms, and iterate the process. All details are thoroughly described in the coming sections.

Furthermore we draw the attention of the reader towards the optimization techniques we are applying to the Query Tool in order to improve the usability of the system. Improvements are made working on three fronts: reducing as much as possible the calls to the reasoner, storing the taxonomy, and caching query information.

First of all we describe the technologies and techniques underlying the system, then we present the actual system (Query Tool) from the user perspective, with a complete exposition of the functionalities of the interface. Afterwards we illustrate the interaction with the reasoning services followed by a section on the optimizations of such a system.

3.2.1 Background

Ontology mediated access to data sources

The purpose of the presented Query Tool is to support query formulation in the context of information access mediated by ontologies. More specifically, the scenario in which we consider the deployment of the tool consists of one or more data sources providing their own query language (e.g. they can be relational sources). The information provided by the sources is described by means of a global ontology together with mappings relating the ontology vocabulary to the vocabulary of the data sources. We do not impose any constraint on the kind of mappings and/or architecture underlying the integration system.

The Query Tool relies on the availability of an ontology providing the vocabulary for the queries and a query engine capable to retrieve the data. These minimal requirements enable the Query Tool to be used in simple cases in which data are retrieved from a knowledge base (see [Sirin and Parsia, 2007]) as well as more complex architectures in which query answering requires complex processing (e.g. using rewriting [Calvanese *et al.*, 2007]).

The ontology language adopted by the tool is OWL-DL (see [Horrocks *et al.*, 2003]), therefore the conceptual model exposed to the user centers around the concept of classes and properties. While the user is guided to the construction of queries structured in terms and properties which can be refined (see the next sections for details), the system generates conjunctive queries composed by unary (classes) and binary (attribute and relation) predicates.

Queries

The Query Tool represents queries to the user as trees, in which nodes are labelled by classes and edges by properties. Each node of the tree correspond to a different variable and properties (edges) constitute the joins between a node and the rest of the query. In this way the conjunctive queries generated by the system are acyclic.[2]

Users interact with the system to refine the query by a set of operations which can be performed on nodes of the query tree. Once selected, a node

[2]From the technical point of view cyclic queries wouldn't pose any problem; however, usability tests conducted in the context of a previous project suggested that the users don't find co-references intuitive enough.

becomes the focus for the operations which can be divided into substitution (when a class is substituted by more general or specific one) and incremental refinement by addition of compatible classes or properties. Additionally, the system allows the deletion of part of the query.

For each focus the tool suggests the terms and/or properties which can be used to refine the query. This step requires the interaction with an OWL-DL reasoner in order to establish which properties or classes are "compatible" with the current query. This must be done in real time when the user interacts with the tool, since both the query and the focus affect the responses from the reasoner.

For more details on the query language and the user perspective over the tool, the reader is referred to [Dongilli *et al.*, 2004]; here we concentrate on showing how we increased the responsiveness of the system by optimizing the use of the OWL reasoner.

Reasoning services

An OWL reasoner is employed to derive the information required to drive the interface. These information range from the taxonomical position of an OWL expression w.r.t. the terms of the ontology, to the satisfiability of an expression.

To allow for the maximum flexibility, the tool communicates with the reasoner by means of the DIG API (see [Bechhofer *et al.*, 2003]). To one side this enables the possibility of using any compliant reasoner; but on the other side the use of HTTP as underlying transport introduces additional overhead in terms of network connections.

For this reason, one of the first goal we wanted to achieve is to minimize the number of calls to the reasoner (see Section 3.2.4).

An Example

Now we want to present an example that will be referred throughout the rest of chapter to better understand the operations involving the reasoner. To do so, we employ an excerpt of the Wine Ontology which is shown in Fig. 3.2. We adopted the UML notation to represent the *is-a* relationships among terms and we introduced constraints of disjointness where needed.

We have *Wine_Descriptor* as root concept and *Wine_Taste* and *Wine_Colour* as specializations of Wine_Descriptor.The concept *Wine* has a property *has_Colour* towards concept Wine_Colour; the inverse of this property is *colour_Of* when seen from concept Wine_Colour. Wine_Colour specializes in *Red* and *White* while Wine in *Red_Wine*, *White_Wine*, and *Table_Wine* respectively.

The axioms of this sample ontology are represented below: [MISSING]

3.2.2 Query Tool

In this section we present a brief description of the end user system. It is a Java-based application adopting the Standard Widget Toolkit (SWT) [SWT, 2007] for creating the graphical user interface. The system requires at least JRE 1.4 and a DL reasoner providing a DIG 1.x interface.

The query interface is provided with four Tabs:

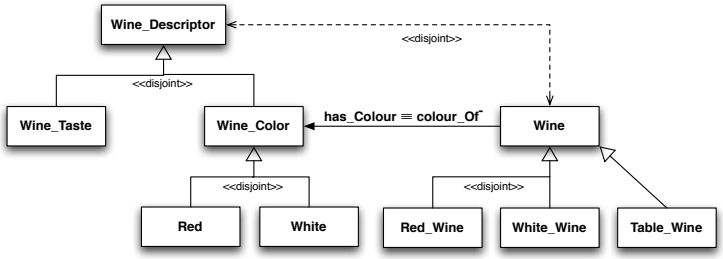


Figure 3.2: An excerpt of the Wine Ontology.

- Admin: administrative interface used to load the ontology and connect to the reasoner.
- Compose: main query composition interface.
- Query: displays the actual query, mainly for debugging purposes.
- Results: displays the results of the query evaluation.

Initially the user is presented with the *Admin* Tab (see Figure 3.3). Here, some preliminary operations necessary for the query formulation have to be executed:

1. **Connection setup**: one of the operations the user has to carry out consists in testing the reasoner connection. A reasoner with reference to the ontology is used by the system to drive the query interface: in particular, it is used to discover the terms and properties which are proposed to the user to manipulate the query.
2. **Loading and managing ontology files**: all the operations the system provides cannot be accomplished without loading an ontology. The interface allows the user to specify an ontology in DIG 1.x format to be loaded into the system. Once the ontology is loaded into the system, the user has also the possibility to adjust the content of that ontology, depending on her needs; if the user wants that the modifications take a permanent effect, she can save them back to the file. As a matter of fact, users might frequently have the necessity to extend an ontology in order to obtain different results or to correct it as a consequence of unexpected behaviour.
3. **Loading a metadata file**: the interface gives the possibility to the user to specify a metadata file to be loaded into the system. Metadata files contain valuable information about the terms in the ontology; that information concerns essentially the lexicalizations of those terms. Actually, as the terms contained in the ontology could be expressed by a sort of shorthand, their lexicalizations are provided so that the user can deal with clearly understandable terms.
4. **Customising lexicalizations**: given the metadata file, the interface offers to the user the opportunity to apply desired variations to the lexical information of the terms. Those variations can be saved back to a metadata

file or just saved temporarily in the system. The query to be generated should be as unambiguous as possible: if the user can assign to the terms the lexicalizations which best give significant importance to her, the query formulation will be transparent and therefore the really intended result will be retrieved.

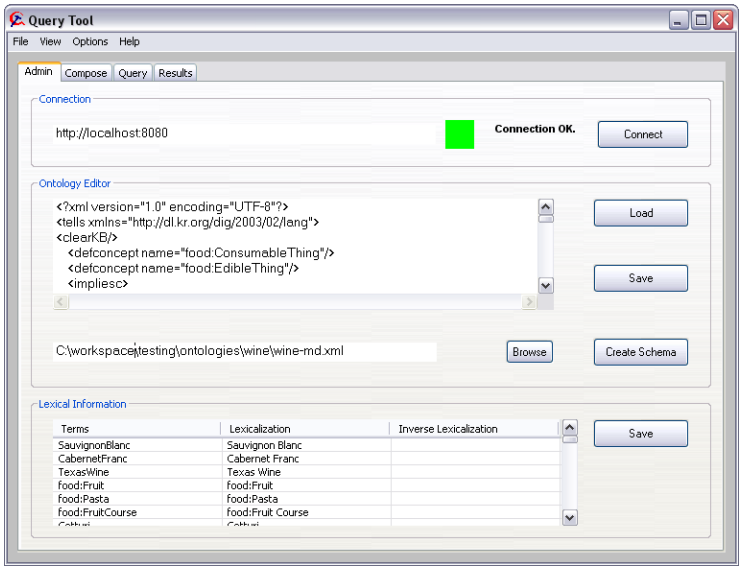


Figure 3.3: Administrative interface of the Query Tool.

As you can see in Figure 3.3, the reasoner connection has been tested by means of the "Connect" button. An ontology has been loaded ("Load" button) and also a metadata file ("Browse" button). Subsequently, the "Create Schema" button has been clicked and all the lexicalizations of the ontology terms are presented in the "Lexical Information" table. Here the user can change the lexicalizations by clicking on the cell corresponding to the lexicalization she wants to modify.

In the *Compose* Tab (see Figure 3.4) the user can formulate the query by means of pop-up menus presenting the possible operations. Initially the user is presented with a choice of different starting terms (all the concepts in the ontology or a subset defined by means of the metadata file): she selects the first term to be added in the query. Subsequently, the interface gives the possibility to perform the following operations:

- **Add compatible terms**: other terms specified in the ontology can be added to the query. The compatible terms are automatically suggested to the user by means of appropriate reasoning tasks on the ontology describing the data sources. Indeed, the system suggests only the operations which are compatible with the current query expression.

- **Substitute terms**: the system gives the opportunity of substituting the selected term of the query with a more specific or more general term. It can also be the case that in the ontology there are terms which are

equivalent to the selected one: in this case the user is offered to replace the selection with an equivalent term.

- **Delete terms**: as the query is specified through an iterative refinement process, it could be the case that the user needs to delete some terms from the query.
- **Add or delete properties**: analogously, the user can add properties to the query. A property can be a relation or an attribute. The interface suggests a list with the possible alternatives. The user can specify some restriction values to attributes.

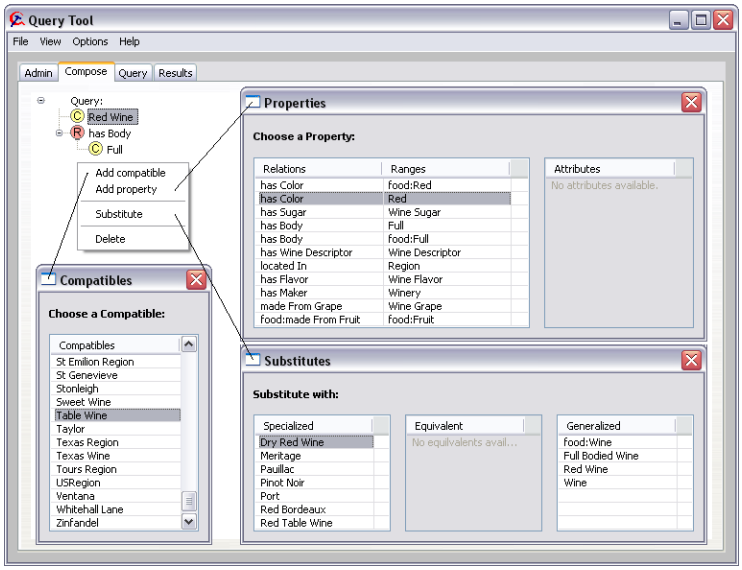


Figure 3.4: Query composition interface.

The first operation to compose a query consists in selecting the starting term. By clicking on a pop-up menu ("Choose starting term") the user is presented with a windows showing all the terms that can be used as starting term.

Once the user has selected the starting term, it is possible to refine the query using again the pop-up menu. The operations allowed are listed in the pop-up menu; the user can add a compatible term, add a property (relation or attribute), substitute the term or delete it.

If the user selects an attribute, it is possible to set it as distinguished variable or to add a restriction to the attribute. The user can also delete properties or terms from the query and select new ones.

Once the user has formulated the query, the *Query* tab shows the query in XML and DIG formats (the menu bar "Options" allows also to view the query in the corresponding SQL code). Finally, in the *Results* tab the user can retrieve the results (if any) corresponding with the formulated query.

In the menu bar, by clicking on "View" menu, the user can have a look to the log file ("View" log) of the application and also a concise description of the schema with all the taxonomy ("View" schema).

3.2.3 Reasoner interaction

In this section we describe all the operations (w.r.t. the reasoner) that users can perform during the query formulation process. Refinement of the query expression can be done by the following operations:

- addition of a compatible term;
- addition of a property;
- substitution of a term with an equivalent, more specific or more general term.

In primis we present the approach which enables the system to interact with the reasoner and then the formalization of the above operations.

Addition of a compatible term

This operation requires the list of terms "compatible" with the given query. In terms of conjunctive queries, it corresponds to add a new term to the query. The term is compatible and can be added to the query if the resulting query is satisfiable. Let us formally define a compatible term w.r.t. a query. Given an ontology Σ and a focused query $Q^{F(x)}$ we want to find all the terms $Y \in \mathbb{C}$ (where $\mathbb{C}$ are all the unary atomic terms) such that:

$\Sigma \not\models Q^{F(x)} \sqcap Y \sqsubseteq \bot$	Y is not disjoint with $Q^{F(x)}$
$\Sigma \not\models Q^{F(x)} \sqsubseteq Y$	Y is not among ancestors of $Q^{F(x)}$
$\Sigma \not\models Y \sqsubseteq Q^{F(x)}$	Y is not among descendants of $Q^{F(x)}$

The reasoning service makes use of satisfiability to check which predicates in the ontology are compatible with the current focused query. This check corresponds simply to the addition of the term Y to the focused query $Q^{F(x)}$, and verify that the resulting query is satisfiable. Actually, this operation is very expensive because the number of reasoner calls matches the number of unary predicates.

Going back to the example of Sec. 3.2.1, if we have a query with concept Red_Wine and we want to find all the concepts which are compatible with it, it will turn out that concept Table_Wine is compatible with the query while concept White_Wine is incompatible since it is disjoint with Red_Wine.

Addition of a property

The addition of a property requires the discovery of both a binary term and its restriction (or range). The system should check all the different binary predicates from the ontology for their compatibility. Formally, a property P is compatible with a focused query $Q^{F(x)}$ if

$$\Sigma \not\models Q^{F(x)} \sqcap \exists P.\top \sqsubseteq \bot,$$

where $\top$ represents any possible concept of the domain.

This is practically performed by verifying the satisfiability of the query $Q^{F(x)} \sqcap \exists P.\top$, for all atomic binary predicates P in the signature. Once a binary predicate is found to be compatible with $Q^{F(x)}$, repeated satisfiability is used to

select the least generic unary predicate $Y \in \mathbb{C}$ such that the query $Q^{F(x)} \sqcap \exists P.Y$ is satisfiable. In other terms, the operation would consist in determining which are the compatible properties first, and then establishing which are the restrictions applicable to P. To discover all compatible properties, we need a number of reasoner calls equal to the number of properties in Σ. In addition, for each property found, to determine its restriction, we need as many reasoner calls as the number of unary predicates.

Again, returning to the example (see Sec. 3.2.1), this time we want to discover the properties which are compatible with the query Wine_Descriptor. As compatible properties propagate upwards in the hierarchy, property colour_of would be among the compatible properties of Wine_Descriptor. If the user instead composes a query with the concept Wine_Taste, the property colour_of would be incompatible because the concept Wine_Taste is disjoint with the concept Wine_Colour.

Substitution of a term

Here we want to substitute a focused term of the query with an equivalent, more specific or more general term. Let us examine the substitution with a more specific term. In this case we need to perform a containment test of two conjunctive queries. Given a query focussed on concept F ($Q^{F(x)}$), we are interested in the unary terms Y subsumed by $Q^{F(x)}$, where Y must be the most general concept among the terms found (i.e. there is no other concept Y subsumed by $Q^{F(x)}$ and containing Y). Formally, given an ontology Σ and a query $Q^{F(x)}$, we want to find all the terms $Y \in \mathbb{C}$ such that:

$$\Sigma \models Y \sqsubseteq F, \neg\exists Z \in \mathbb{C} \,|\, (Z \sqsubseteq F, Y \sqsubseteq Z, Z \neq Y).$$

$$\Sigma \not\models F \sqcap Q^{F(x)} \sqcap Y \sqsubseteq \bot.$$

From Figure 3.2 it is possible to see that if the query is composed by concept Wine and we want to substitute it with a more specific term, we would get Red_Wine, White_Wine, and Table_Wine as candidates for the substitution since they are direct children of concept Wine.

The cases of substitution with more general and equivalent terms are analogous.

For the sake of clarity we report the sequence of operations needed to retrieve the substituting terms:

- query rolling-up;
- retrieval of incompatible classes: the descendants of negation of the query;
- retrieval of parents and children of the substituting term;
- filtering using incompatible terms.

We will see in Section 3.2.4 that a similar procedure is adopted to reduce the calls to the reasoner when looking for compatible terms.

3.2.4 Optimization

As discussed in Section 3.2.1, the system relies on a DL reasoner to drive the query interface. If on one hand reasoning services with satisfiability and classification allow only to formulate consistent queries, on the other hand they introduce performance issues. Reasoner calls are expensive and should be therefore minimized as much as possible. The expensiveness of reasoner calls depends both on complexity and the fact that DL reasoners exploit the HTTP protocol to communicate.

In the following we present some optimization techniques which can improve the usability of the system via a more responsive interface. Aim of the optimization is to reduce the transitions between the query interface and the reasoner. Some techniques have already been exploited to reduce the number of reasoner calls especially in the retrieval of compatible terms to be added to the query. Another important improvement comes from the storage of the taxonomy. Finally, information concerning the query can be cached during the query formulation process in order to extract some deductions to reduce reasoner calls.

Reducing reasoner calls

Concerning the refinement of the query by compatible terms, the basic policy to retrieve the compatible terms is to use the satisfiability reasoning service to check which unary predicates in the ontology are compatible with the current query. This check corresponds simply to the addition of the term to the current query, and to verify that the resulting query is satisfiable. Actually, this kind of operation is very expensive because the number of reasoner calls corresponds to the number of unary predicates in the ontology.

We adopted a different implementation in the current system. We classify the query and retrieve the its equivalents, ancestors, and descendants; then we classify the negation of the query and retrieve the descendants which are incompatible. The remaining unary predicates are the compatibles (see Section 3.2.3).

In reference to the addition of a property, as we discussed in Section 3.2.3, this operation requires the discovery of both a binary term and its restriction. One of the advantages of OWL-DL is the possibility of expressing the inverse of a role which is extremely useful for determining compatibility of binary terms. Hence, to discover the restriction of a property we use classification instead of repeated (and expensive) satisfiability. The idea is to classify the inverse of the property restricted to the query.
For example, to discover the restriction of property has_Colour applied to the query expression

$$\{x_1 \mid \text{Red\_Wine}(x_1), \text{Table\_Wine}(x_1)\},$$

we classify the expression $\exists \text{has\_Colour}^{-}(\text{Red\_Wine} \sqcap \text{Table\_Wine})$.
The reasoner returns the list of concept names more general and equivalent as range candidates of the relation has_Colour, when restricted to the domain (Red_Wine $\sqcap$ Table_Wine). This method, not only lets us discover the least general predicate(s) which can be applied to the property in the given context, but also allows us to discard those properties which are incompatible with the query, i.e. bottom ($\bot$) is returned as range whenever a given property is

incompatible with the query. Summarizing, we are able to both check the compatibility of a property with the query and find out the property's range by means of one single reasoner call.

Taxonomy storage

The taxonomy of the ontology provides static information concerning primitive concepts. If we store the taxonomy before starting to compose a query, initial operations like substituting a concept, would not involve the reasoner, thus improving efficiency.
The taxonomy is actually a partial order '$<$' from *Top* ($\top$), the whole domain, to *Bottom* ($\bot$), the empty set, where the partial order relation is subsumption. The partial order can be represented by a *directed acyclic graph (DAG)*, i.e. a directed graph that contains no cycles. An edge is drawn from a to b whenever $a < b$. A partial order satisfies the following properties:

- transitivity, $a < b$ and $b < c$ implies $a < c$;
- non-reflexive, not($a < a$).

These condition prevent cycles because $a < b < ... < z < a$ would imply that $a < a$, which is false. The only exception where the property $a < a$ holds is for the equivalent concepts.

The idea is to save not only the taxonomy but also other information pertaining each concept such as e.g. its incompatible classes and the list of incompatible properties.

Caching query information

The *focus* plays an important role during the query formulation process; in particular the system proposes the available operations on the query w.r.t. the current focus (i.e. the variable which is currently selected). The focus is crucial also for caching dynamic information concerning the query and the idea is to cache both the query and its actual classification at the focus level. In other words, we want to associate to each single variable which gets the focus the overall status of the query. Of course, cache at the single node level would be invalidated as soon as the user further refines the query.

An intuitive approach to exploit the cache would consist in modifying the system in a way that the user can only remove terms by following the exact inverse order of the one which has been used to formulate the query. This means that only the last operation can be undone. In this way we could reduce or even avoid reasoner calls because the information we need has already been cached at the node.

We know that refinement of the query is monotonic and therefore whenever the user adds new terms to the query, the domain is going to be restricted. This property can also lead us to some conclusions for reducing reasoner calls in further refinements of the query. This property does not hold when the user deletes a term from the query; in this case all the cache has to be removed.

3.2.5 Discussion

Optimizing the communication and the quantity of exchanged messages behind the scenes between the Query Tool and the reasoner is only the first action we are taking to make the use of the interface more comfortable for the user.

The interaction time we are able to save with these enhancements is *partially* re-invested in the demands of a new and more complex interface we are building, based on state-of-the-art natural language generation (NLG) technologies.

The main challenge is that the query (now with partial verbalization of single concepts and roles) is to be presented to the user in natural language with full verbalization, and stepwise refinements of the query composed by the user are presented as natural language refinements that maintain the grammaticality of the sentences representing the query. Our solution adopts the paradigm called WYSIWYM ('What You See Is What You Meant'), a user-interface technique which uses (NLG) technology to provide feedback for user interaction [Power and Scott, 1998]; it will be presented in Section 4.1.1. The differences between our approach and that used by available systems employing WYSIWYM were published in [Dongilli *et al.*, 2006], and are explained here in Section 4.2. Chapter 5 will be entirely devoted to the detailed analysis of the problem of rendering a conjunctive query in English.

Chapter 4

Querying with natural language support

4.1 Related work

4.1.1 WYSIWYM and available implementations

4.2 A novel implementation of the WYSIWYM paradigm

Chapter 5

Natural language rendering of a conjunctive query

.....

5.1 An Introduction to Natural Language Generation

Communication by means of natural language involves two fundamental skills: Producing text and understanding it. These tasks are the subject of study of two big areas of research in computational linguistics, which are *natural language generation* and *natural language understanding* respectively; the former will be dealt with hereinafter.

Natural language generation (NLG) is seen in general as the sequence of operations needed to map information from some *non-linguistic* (e.g. raw data) into *linguistic* form (either oral or written). These operations are not at all straightforward, because the task of bridging the gap between non-linguistic and linguistic representations requires several non-trivial decisions or choices which include *content determination*, choice of *rhetorical structures* at various levels (text, paragraph, sentence), choice of *words* and *syntactic structures*, and finally the determination of the *text layout* (or acoustic patterns if we intend to generate spoken text). One of the main challenges of NLG is devising modular architectures able to make the previous choices coexist. At least three kinds of expertise are needed: *application domain knowledge*, *knowledge of the language* (grammar, lexicon, and semantics), and *strategic rhetorical knowledge* (i.e. how to achieve communicative goals, text types, style).

NLG system architectures need to include various levels of planning and merging of information in a way that generated text looks natural and not repetitive. Typical tasks we find are [Reiter and Dale, 2000]:

- **Text planning**
  - **Content determination**: Determination of the salient features that are worth being said.

  - **Discourse planning**: Overall organization of the information to convey.

- **Sentence planning**
  - **Lexicalization**: Putting words to the concepts.
  - **Sentence aggregation**: Merging of similar sentences to improve readability and naturalness. For example, the sentences "The car is equipped with a diesel engine" and "The engine's power is 140 HP" can be aggregated to form "The car is equipped with a diesel engine whose power is 140 HP".
  - **Referring expression generation**: Linking words in the sentences by introducing pronouns and other types of means of reference.

- **Linguistic realization**
  - **Syntactic and morphological realization**: This stage is the inverse of parsing: given all the information collected above, syntactic and morphological rules are applied to produce the surface string.
  - **Orthographic realization**: Matters like casing, punctuation, and formatting are resolved

In the coming sections we will tackle each one of the mentioned tasks, describing a pipeline of steps which will build up our own generation architecture, able to map a conjunctive query (over a given domain ontology) into its corresponding textual form.

5.2 Text Planning

Thinking what to write or say, and organizing the constituents of our idea in one of the possible manners that once verbalized will best convey our thought is what we could define as the *text planning* capability of a human being. We want to mimic this human behavior with the first module of our generation system, where the content (a query), determined by the user by means of an intelligent query composition interface, is internally reorganized in order to obtain the possibly most coherent sequence of its constituents. Since the query is isomorphic to a tree, the job of the discourse planner is to find the best topological sorting according to some objective function. In Section 5.2.2 we present and compare six discourse planning strategies indicating one of them as the best suited for this task. These results were also published in [Dongilli, 2007a] and [Dongilli, 2007b].

5.2.1 Content determination

In our specific context, the content is represented by the query formulated by the user. Given the specific domain of interest chosen (read *ontology*), the query built using the ontology's (unary and binary) predicates corresponds to a simple or complex concept (namely a conjunction of predicates) whose constituents need to be organized and explained in a coherent discourse using one or more natural language sentences,

5.2.2 Discourse planning

Planning a discourse means in general finding the best way of representing an idea in an organized, specific, and coherent manner. In our case the idea is a query, a complex concept that the user is thinking and building. We start by defining the main components of a discourse that are called *discourse units*.

Definition 11 (Discourse unit)**.** A discourse unit $u_i(c_j, c_k)$ is the atomic component of a discourse. In our setting a discourse unit is represented by a *role* r_l between two entities c_j and c_k (using the terminology introduced by def. [insert ref. to cq definition], r_l can be either a relation or an attribute having c_j and c_k as domain and range respectively; c_j is always a concept, and c_k is a concept if r_l is a relation, or a predicate over a concrete domain if r_l is an attribute).

A discourse unit, once verbalized, can be seen as a stand-alone sentence, or as a clause in a longer sentence.

We start with a query tree and we map it into another tree we call *discourse tree*, created by collapsing a relation between two concepts into a single node.

Definition 12 (Discourse tree)**.** A discourse tree is a directed tree whose nodes are discourse units. The nodes are tagged with the domain and range entities of the corresponding role. The edges connect two nodes where the second entity of the start node and the first entity of the end node are the same. The root node of the discourse tree is an additional node which introduces the main concept (c_1) the user is looking for, i.e. the root concept of the query tree. The first entity (c_0) of this node is a new entity that will be verbalized as the subject of this first discourse unit.

Figures 5.1 and 5.2 show a starting query tree and the derived discourse tree. The sequence order of indexes assigned to concepts and relations in the query tree, and as consequence the indexes assigned to discourse units, respects the order of insertion followed by the user while creating the query.

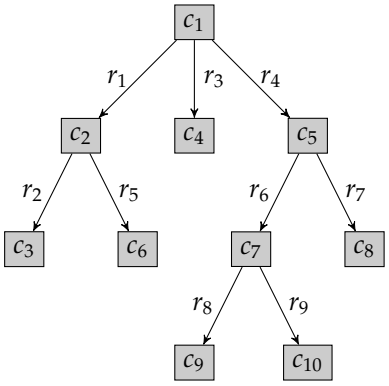


Figure 5.1: A query tree

Our starting point for discourse planning is the generated discourse tree which is a directed tree as mentioned in def. 12. The problem of finding a linear sequence of the discourse units in a discourse tree can be translated into a problem which in graph theory is called *topological sorting*.

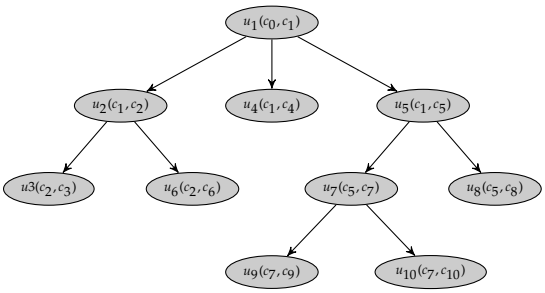


Figure 5.2: A discourse tree

Definition 13 (Topological sort)**.** A *topological sort* of a directed acyclic graph (DAG) $G(V, E)$ is a linear ordering of its nodes which is compatible with the partial order R induced on the nodes, where x comes before y (xRy) if there's a directed path from x to y in the DAG.

In other terms, topological sorting is a way to extend a *partial order* relation into a *total order*. We can state that every DAG has at least one topological sort, because of the following

Theorem 1. *Every partial order can be extended to a total order. That is: Suppose* $\rightarrow$ *is a partial order on a set X. Then there exists a total order* $\Rightarrow$ *on X that extends* $\rightarrow$ *as a relation: If* $x, y \in X$ *and* $x \rightarrow y$*, then* $x \Rightarrow y$*.*

Typical algorithms for topological sorting have running time linear in the number of nodes plus the number of edges ($\Theta(|V| + |E|)$). Since in our setting we are working with a DAG where $|E| = |V| - 1$, the complexity is $\Theta(|V|)$. A possible algorithm is the following:

Algorithm 1 Generation of a topological sort

```
Q ← Set of all nodes with no incoming edges
while Q is not empty do
  remove a node n from Q
  output n
  for all nodes m with an edge e from n to m do
    remove edge e from the graph
    if m has no other incoming edges then
      insert m into Q
    end if
  end for
end while
if graph has edges then
  output error (the graph has a cycle)
end if
```

If we want to find all possible topological sorts, algorithm 1 needs the following modifications that lead to algorithm 2:

- the *while* loop must be implemented by a recursive function;

- bag Q has to be global to the recursive function;
- an array must be defined to hold the current ranking and that must be output when Q is found to be empty;
- the currently removed item is to be kept locally;
- when returning from the recursive call, the current item has to be put back into Q and another node must be picked from Q.

Algorithm 2 Generation of all topological sorts

```
Q ← Set of all nodes with no incoming edges
D ← ∅ {array containing temporary linear sort}
call CalculateSorts()

procedure CalculateSorts()
if Q is not empty then
  for i = 1 to size(Q) do
    remove node n_i from Q
    add n_i to D
    for all nodes m with an edge e from n_i to m do
      remove edge e from the graph
      if m has no other incoming edges then
        insert m into Q
      end if
    end for
    call CalculateSorts()
    restore n_i into Q
    restore previously removed edges outgoing from n_i
    remove last element from D
  end for
else
  output D
end if
if graph has edges then
  output error (the graph has a cycle)
end if
end procedure
```

We implemented this algorithm (see algorithm 2) whose running time depends on the topology of the tree. It is easy to see that the number of topological sorts varies from 1 to $|E|!$; these two extreme cases are shown in Figure 5.3. The former tree already represents a linear order, while the latter has $n-1$ possible topological sorts.

In our context, finding a topological sort of a discourse tree can be defined this way:

Definition 14 (Topological sort of a discourse tree)**.** Given a discourse tree with n discourse units $u_1, u_2, \ldots, u_n$ containing $n+1$ discourse entities $c_1, c_2, \ldots, c_n$, a *topological sort* can be obtained with a permutation π of $\{1, 2, \ldots, n\}$, where the

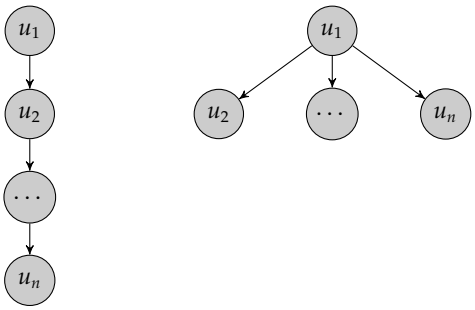


Figure 5.3: Best and worst cases for topological sorting

sequence of discourse units $(u_{\pi(1)}, u_{\pi(2)}, \ldots, u_{\pi(n)})$ is compatible with the partial order induced by the discourse tree.

Hereinafter a generic topological sort will be associated and identified with a permutation π.

Given all possible topological sorts of our discourse tree, we need now to find some constraints with the intent to keep only those orderings that maximize/minimize some objective function. Given an objective function, the aim is to find some common properties of the best orderings, in a way to be able to infer an algorithm that is able to calculate just one of the best topological sorts.

Centering-Theory-based planning

The constraints we use in this first attempt are borrowed from Centering Theory which gives us the means to find all possible sequences of discourse units that maximize coherence.

Centering theory (CT) finds its origins within the theory of discourse structure that was first developed by [Grosz and Sidner, 1986]. A draft manuscript describing the centering framework and the first theoretical claims appeared in 1986 [Grosz *et al.*, 1986], and the authors were then urged to publish a more detailed description which came into light in 1995 [Grosz *et al.*, 1995]. This, along with a previous contribution from [Brennan *et al.*, 1987], contains the main claims of this theory, which are:

1. for each discourse unit, there is exactly one entity which is the *center* of attention;
2. there is a preference for consecutive discourse units that keep the same entity as center, and for the most salient entity in a discourse unit to be realized as the center of the next utterance;
3. the center is the entity with the highest probability to be pronominalized.

The assumptions of CT are formalized in terms of C_f, C_b, and C_p. Given two consecutive discourse units $u_{\pi(i-1)}$ and $u_{\pi(i)}$,

- $C_f(u_{\pi(i)})$ (*forward looking centers*) is a list of all discourse entities contained in u_i;

- $C_b(u_{\pi(i)})$ (*backward looking center*) is the most highly *ranked* entity realized in $u_{\pi(i-1)}$ which is also realized in $u_{\pi(i)}$; If $u_{\pi(i-1)}$ does not exist, there is no $C_b(u_{\pi(i)})$;
- $C_p(u_{\pi(i)})$ (*preferred center*) is the highest *ranked* entity of $u_{\pi(i)}$.

[Brennan *et al.*, 1987] define *ranking* of an entity in a discourse unit as *the likelihood that it will be the primary focus of subsequent discourse*. It is more common now defining the rank in terms of grammatical roles (obliqueness), where subject > direct object > indirect object > others.

With the abovementioned parameters, we list now the following constraints, whose violations will build-up the cost function we are going to use.

cohesion: $C_b(u_{\pi(i)}) = C_b(u_{\pi(i-1)})$ (checks if the center of the current discourse unit is the same as the preceding one);

salience: $C_b(u_{\pi(i)}) = C_p(u_{\pi(i)})$ (checks if the center is realized as subject);

cheapness: $C_b(u_{\pi(i)}) = C_p(u_{\pi(i-1)})$ (checks if the current center was a subject in the previous discourse unit);

continuity: $C_f(u_{\pi(i)}) \cap C_f(u_{\pi(i-1)}) \neq \emptyset$ (checks if there are no entities in common between the previous and the current discourse unit).

We say that there is a *violation* to one of these constraints, if the corresponding condition does not hold. If there exists no $C_b(u_{\pi(i)})$, cohesion, salience, and cheapness are not violated. No violation is accounted for the first discourse unit $u_{\pi(1)}$.

The cost function we want to minimize in order to maximize local coherence is defined as follows:

Definition 15 (Centering-theory-based cost function)**.** Given the setting of definition 14, we define this cost function:

$$\phi_{CT}(\pi) = \sum_{i=1}^{n} [coh(u_{\pi(i)}) + sal(u_{\pi(i)}) + che(u_{\pi(i)}) + con(u_{\pi(i)})]$$

where:

$$coh(u_{\pi(i)}) = \begin{cases} 0 & \text{if } i \in \{1,2\} \text{ or} \\ & i > 2 \text{ and } C_b(u_{\pi(i)}) = C_b(u_{\pi(i-1)}) \\ K_{\text{coh}} & \text{if } i > 2 \text{ and } C_b(u_{\pi(i)}) \neq C_b(u_{\pi(i-1)}) \end{cases}$$

$$sal(u_{\pi(i)}) = \begin{cases} 0 & \text{if } i = 1 \text{ or} \\ & i > 1 \text{ and } C_b(u_{\pi(i)}) = C_p(u_{\pi(i)}) \\ K_{\text{sal}} & \text{if } i > 1 \text{ and } C_b(u_{\pi(i)}) \neq C_p(u_{\pi(i)}) \end{cases}$$

$$che(u_{\pi(i)}) = \begin{cases} 0 & \text{if } i = 1 \text{ or} \\ & i > 1 \text{ and } C_b(u_{\pi(i)}) = C_p(u_{\pi(i-1)}) \\ K_{\text{che}} & \text{if } i > 1 \text{ and } C_b(u_{\pi(i)}) \neq C_p(u_{\pi(i-1)}) \end{cases}$$

$$con(u_{\pi(i)}) = \begin{cases} 0 & \text{if } i = 1 \text{ or} \\ & i > 1 \text{ and } C_f(u_{\pi(i)}) \cap C_f(u_{\pi(i-1)}) \neq \emptyset \\ K_{\text{con}} & \text{if } i > 1 \text{ and } C_f(u_{\pi(i)}) \cap C_f(u_{\pi(i-1)}) \neq \emptyset \end{cases}$$

K_{coh}, K_{sal}, K_{che}, and K_{con} represent the weights assigned to each constraint violation.

We can now use this cost function to discern, among all orderings, the ones that minimize violations to local coherence in terms of cohesion, salience, cheapness, and continuity, where the respective weights are assigned according to the proposal of [Kibble and Power, 2004]: $K_{\text{coh}} = K_{\text{sal}} = K_{\text{che}} = 1$, and $K_{\text{con}} = 3$. The assumption we make is that for every discourse unit $u_i(c_k, c_l)$, the preferred center $C_p(u_i)$ (subject) will always be the first discourse entity i.e. c_k.

We implemented algorithm 2, and conducted experiments over several tree topologies, isolating all sortings that minimized the given cost function. The results obtained, quite unexpected, are reported in the next section.

The first result we observed is that salience is never violated. This can be formalized in the following theorem:

Theorem 2. *Given the setting of definition 14, none of the topological sorts generated from the discourse tree violates the salience constraint.*

Proof. In order to check salience in any discourse unit $u_{\pi(i)}(c_l, c_m)$ of a generic topological sort π, we have to check that $C_b(u_{\pi(i)}(c_l, c_m)) = C_p(u_{\pi(i)}(c_l, c_m))$ where $C_p(u_{\pi(i)}(c_l, c_m)) = c_l$ as we assumed above. To identify the C_b we need the previous discourse unit $u_{\pi(i-1)}(c_j, c_k)$. If it does not exist, this means that $u_{\pi(i)}$ is the first discourse unit ($u_{\pi(1)}$) and there is no salience violation. If we have a previous utterance, we distinguish two cases:

1. $\{c_l, c_m\} \cap \{c_j, c_k\} = \emptyset$: this means that there is no C_b in unit $u_{\pi(i)}$, therefore salience is not violated;

2. $\{c_l, c_m\} \cap \{c_j, c_k\} \neq \emptyset$; the units cannot have two discourse entities in common because this would mean that either the two entities are the same ($c_j = c_l \wedge c_k = c_m$) or that we have a cycle in our tree ($c_j = c_m \wedge c_k = c_l$) which is impossible. We can have only one entity in common, i.e. the following four cases:

   (a) $c_j = c_l$: This is a valid case; it implies that $C_b(u_{\pi(i)}(c_l, c_m)) = c_l$ which is equal to $C_p(u_{\pi(i)}(c_l, c_m))$. The salience constraint is attended.

   (b) $c_j = c_m$: This case is not valid, because this would imply that $u_{\pi(i-1)}$ should occur after before $u_{\pi(i)}$, contradicting our hypothesis.

   (c) $c_k = c_l$: This is a valid case; it implies that $C_b(u_{\pi(i)}(c_l, c_m)) = c_l$ which is equal to $C_p(u_{\pi(i)}(c_l, c_m))$. The salience constraint is attended.

   (d) $c_k = c_m$: This is not a valid case, since it would imply coreference and therefore a cycle in our tree which is impossible.

□

The second result we obtained pertains a common property shown by all best topological sorts, i.e. the ones that minimize the cost function. This result is expressed in the following proposition.

Proposition 1. *Given a discourse tree with n discourse units $u_1, u_2, \ldots, u_n$ containing $n + 1$ discourse entities $c_0, c_1, c_2, \ldots, c_n$, the topological sorts that minimize the cost function expressed in def. 15 are all and only the ones returned by algorithm 3.*

Algorithm 3 Generation of the best topological sorts (Centering Theory)

$D \leftarrow u_1$ {array D stores one by one all best linear sorts; here it is initialized with the first discourse unit}
$n \leftarrow$ number of nodes in tree
$count \leftarrow 1$ {current size of D}
call *CalculateBestSorts*(u_1)

procedure *CalculateBestSorts*(u)
$L \leftarrow$ all children of u
$L_d \leftarrow$ children of u having at least one descendant
if $L_d \neq \emptyset$ **then**
  $P_L \leftarrow$ list of all permutations of L where last node of each permutation is in L_d
else
  $P_L \leftarrow$ list of all permutations of L
end if
for all $p \in P_L$ **do**
  append array p to D
  for $i = size(p)$ **downto** 1 **do**
    call *CalculateBestSorts*(p_i)
  end for
  $count \leftarrow count + size(p)$
  if $count = n$ **then**
    output D
  end if
  remove array p from D
  $count \leftarrow count - size(p)$
end for
end procedure

In plain words, the best topological sorts are the ones for which every discourse unit is followed by its remaining siblings, where the last sibling must be one with descendants (to allow a continuity in the discourse); for each sibling then, starting from the last one, the list of its children (if any) is output. E.g. one of the best topological sorts from the discourse tree of figure 5.2 is $(u_1, u_2, u_4, u_5, u_8, u_7, u_9, u_{10}, u_3, u_6)$.

Minimal conceptual distance

The second constraint we decided to experiment with, calculates the sum over each discourse entity of the distances among discourse units where each entity is referenced.

This optimization problem can be described as follows:

Definition 16 (Conceptual distance minimization)**.** Given n discourse units, $(u_1, u_2, \ldots, u_n)$ embedding $n+1$ discourse entities $(c_1, c_2, \ldots, c_{n+1})$; given a permutation π of $\{1, 2, \ldots, n\}$ where the sequence of discourse units $(u_{\pi(1)}, u_{\pi(2)}, \ldots, u_{\pi(n)})$ is compatible with the partial order induced by the tree, we create a hash table $H\pi$ where the keys correspond to the discourse entities $(c_1, c_2, \ldots, c_{n+1})$, and each value $H\pi(c_i)$ is a sorted list of indexes taken from $\{1, 2, \ldots, n\}$ and referring to some positions of the permutation.

We want to

$$min \sum_{i=1}^{n+1} \delta_\pi(c_i)$$

where

$$\delta_\pi(c_i) = \begin{cases} \sum_{k=1}^{|H_\pi(c_i)|-1} (H_\pi(c_i)[k+1] - H_\pi(c_i)[k]) & \text{if } |H_\pi(c_i)| > 1 \\ 0 & \text{if } |H_\pi(c_i)| = 1 \end{cases}$$

We were able to find a common property of all topological sorts that minimize the newly introduced constraint; it is expressed by the following proposition:

Proposition 2. *Given a discourse tree with n discourse units $u_1, u_2, \ldots, u_n$ containing $n+1$ discourse entities $c_0, c_1, c_2, \ldots, c_n$, the topological sorts that minimize the measure of* conceptual distance *as of definition 16 are those that derive from a depth-first-like visit of the tree, where at any point, given an output node, the valid visits of its subtrees are all the ones where the biggest (in terms of number of nodes) subtree comes last.*

Hybrid approach

The next step was to run topological sorting using a hybrid approach with both of the two previous constraints. We ran several tests on different tree topologies.

We first applied the constraints based on centering theory (CT-approach) followed by the calculation of the minimal conceptual distance (mCD-approach), i.e. after minimizing the cost function of the CT-approach, we applied the mCD-approach on the best ordering. The best results coming out are obviously a subset of the orderings found by algorithm 3. The best topological sorts are the ones for which every discourse unit is followed by its remaining siblings, starting from the ones with no children (if any) and continuing with the siblings in decreasing order of their respective subtree dimensions; for each sibling then, starting from the last one (LIFO), the list of its children (if any) is output. E.g. one of the best topological sorts from the discourse tree of figure 5.2 is $(u_1, u_4, u_5, u_2, u_3, u_6, u_8, u_7, u_9, u_{10})$. This result is expressed in the following proposition, where algorithm 4 is a slight variation of algorithm 3.

Proposition 3. *Given a discourse tree with n discourse units $u_1, u_2, \ldots, u_n$ containing $n+1$ discourse entities $c_0, c_1, c_2, \ldots, c_n$, the topological sorts that minimize the cost function expressed in def. 15 first, and then the cost function of def. 16 next, are all and only the ones returned by algorithm 4.*

Algorithm 4 Generation of the best topological sorts (Hybrid approach #1 (CT-mCD))

$D \leftarrow u_1$ {array D stores one by one all best linear sorts; here it is initialized with the first discourse unit}
$n \leftarrow$ number of nodes in tree
$count \leftarrow 1$ {current size of D}
call *CalculateBestSorts*(u_1)

procedure *CalculateBestSorts*(u)

$L \leftarrow$ all children of u
$L_d \leftarrow$ children of u having at least one descendant
if $L_d \neq \emptyset$ **then**
  $P_L \leftarrow$ list of all permutations of L where nodes in L_d come last, in decreasing order of their respective subtree dimensions;
else
  $P_L \leftarrow$ list of all permutations of L
end if
for all $p \in P_L$ **do**
  append array p to D
  for $i = size(p)$ **downto** 1 **do**
    call *CalculateBestSorts*(p_i)
  end for
  $count \leftarrow count + size(p)$
  if $count = n$ **then**
    output D
  end if
  remove array p from D
  $count \leftarrow count - size(p)$
end for
end procedure

We tried then to apply in sequence the mCD-approach first, and the CT-approach next. We found out that the sequence of discourse units in each best topological sort follow the rule expressed by proposition 4.

Proposition 4. *Given a discourse tree with n discourse units $u_1, u_2, \ldots, u_n$ containing $n + 1$ discourse entities $c_0, c_1, c_2, \ldots, c_n$, the topological sorts that minimize the cost function expressed in def. 16 first, and the cost function of def. 15 next, are all and only the ones that derive from a depth-first-like visit of the tree, where at any point, given an output node, we visit its subtrees ordering them by increasing size.*

The interesting feature of the outcoming orderings is that we leave long elaboration chains at the end, planning the short ones first. If we see it from the point of view of the reader, this is what she usually expects from a text describing an object: Immediate characteristics/attributes of the described object come first, and relations of this object with further entities (possibly nested) are left at the end.

We propose an algorithm that calculates only one of the best orderings, since there could be more than one.

User-driven planning

If on one side the previous approaches try to minimize some cost functions in order to have a higher local coherence and/or a better distribution of entities within discourse units in the text plan, on the other side they don't take into consideration how much change is involved in the text plan whenever the user modifies the query (adding or removing branches to the query tree). The idea would be to minimize the changes in the order of the discourse units in the text plan when the user edits the query.

Algorithm 5 Generation of one of the best topological sorts (Hybrid approach #2 (mCD-CT))

$n \leftarrow$ number of nodes in tree
call *CalculateSort*(u_1)

procedure *CalculateSort*(u)
output u
$L \leftarrow$ all children of u sorted by increasing size of respective subtrees
for all $p \in L$ **do**
  call *CalculateSort*(p)
end for
end procedure

We could think of a text plan where the discourse units have the same order of insertion followed by the user. In this case any addition to the query reflects in a new discourse unit appended to the text plan. E.g. if a new relation is added to concept c_5 in the query tree of figure 5.1, let's say r_{10} along with the range concept c_{11}, this would generate the query tree of figure 5.4, and the discourse tree of figure 5.5 with the new discourse unit u_{11}.

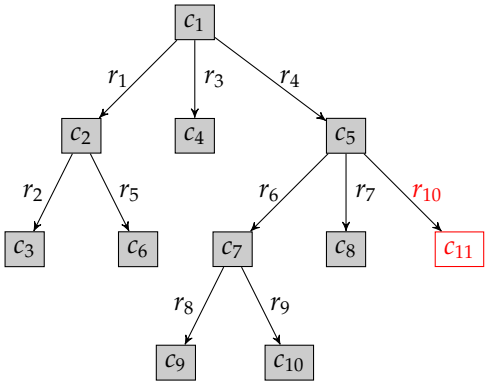


Figure 5.4: Adding a new relation to the query tree of fig. 5.1

The chosen topological sort, according to the planning strategy proposed, would simply be:

$$u_1(c_0, c_1), u_2(c_1, c_2), u_3(c_2, c_3), u_4(c_1, c_4), u_5(c_1, c_5), u_6(c_2, c_6), u_7(c_5, c_7),$$
$$u_8(c_5, c_8), u_9(c_7, c_9), u_{10}(c_7, c_{10}), u_{11}(c_5, c_{11}).$$

A newly inserted discourse unit (as $u_{11}(c_5, c_{11})$ in the example) is always appended to the text plan, possibly far away from the latest previous discourse unit ($u_8(c_5, c_8)$,) having a common discourse referent (c_5), and farther from the unit ($u_5(c_1, c_5)$) where this referent was first introduced.

Hence this kind of strategy yields on average pretty bad values in terms of local coherence and overall conceptual distance according to the measures of definitions 15 and 16. In fact we cannot expect that the user edits the query in a coherent way, having the query tree already in mind before typing, and reproducing it immediately afterwards with a clean depth-first traversal.

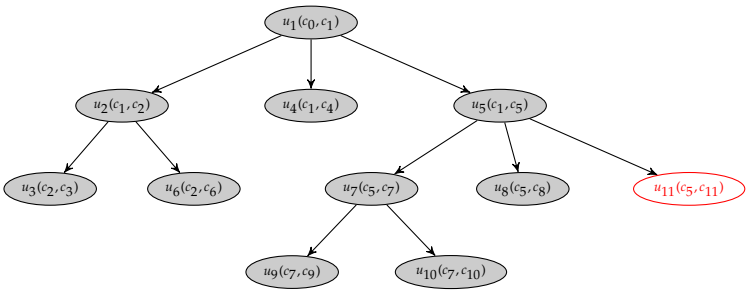


Figure 5.5: Discourse tree derived from the query tree of fig. 5.4

The regularity of a user-driven order of discourse units, where the last inserted unit is always appended to the text plan, doesn't pay-off the bad average performance in terms of local coherence or conceptual distance.

It is evident that we need to find an appropriate trade-off between coherence/distance criteria and minimal change in the text plan after each query editing operation.

Depth-first planning

A good trade-off, slightly unbalanced towards the minimization of the change in the text plan after a query tree edit, is the easiest-to-obtain topological ordering, merely a *depth-first serialization* of the tree. Considering the ordering strategies discussed above, this is among the ones that can be obtained with the lowest time complexity, i.e. $O(n)$ where n is the number of the discourse tree nodes.

Given the tree of figure 5.5, the topological sort according to this method would be:

$$u_1(c_0, c_1), u_2(c_1, c_2), u_3(c_2, c_3), u_6(c_2, c_6), u_4(c_1, c_4), u_5(c_1, c_5), u_7(c_5, c_7),$$
$$u_9(c_7, c_9), u_{10}(c_7, c_{10}), u_8(c_5, c_8), u_{11}(c_5, c_{11}).$$

Algorithm 6 shows how to obtain such simple ordering.

Algorithm 6 Generation of a topological ordering using depth-first search

```
n ← number of nodes in tree
call CalculateSort(u_1)

procedure CalculateSort(u)
output u
L ← all children of u from left to right
for all p ∈ L do
   call CalculateSort(p)
end for
end procedure
```

To complete the list of possible planning strategies we tested, we would like to mention a further one that could be boiled down to a depth-first planning

provided we change the tree topology according to further constraints. It's presented in the next subsection.

Relation-priority depth-first planning

This planning strategy requires additional information that must be pre-specified in the ontology and attached to each relation. These information consist of ordering annotations assigned to relations, and specifying a partial order among those relations having the same domain concept. This translates into a priority value assigned to each discourse unit, and valid only locally within each set of discourse units which are siblings in the discourse tree.

The idea is to keep the query tree edges (relations) that exit from each node, sorted from left to right according to their order. In other words the sequence of relations having the same domain concept are not sorted according to the insertion order followed while creating the query, but respecting the priority value associated with each relation. It could happen though, that two or more relations have the same priority; in this case they are sorted by creation order.

The resulting discourse tree is then ready to be linearized with the simple depth-first traversal proposed above (algorithm 6).

Figures 5.6 and 5.7 show the previous examples of query and discourse tree whose topology is modified according to the additional ordering annotations.

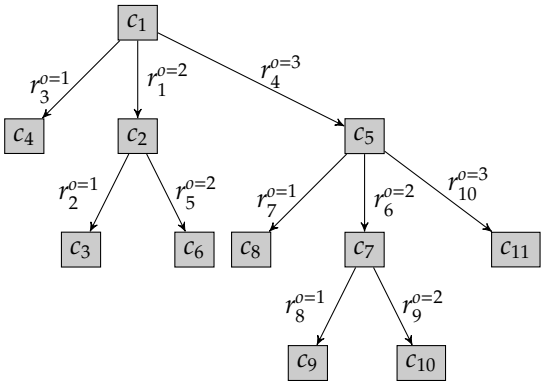


Figure 5.6: Query tree with ordering annotations attached to relations

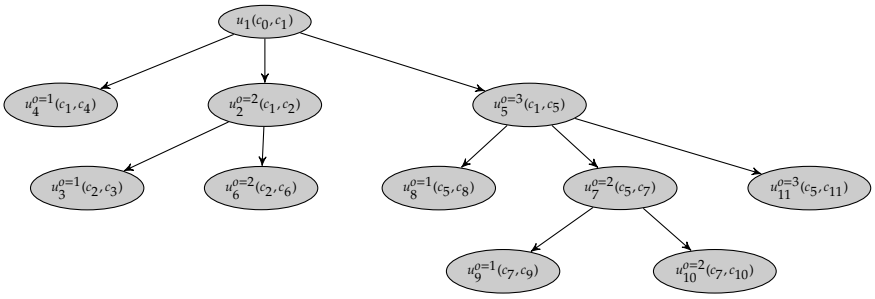


Figure 5.7: Discourse tree with ordering annotation attached to discourse units

Following this planning strategy, from the discourse tree of figure 5.7 we obtain the following topological sort:

$$u_1(c_0, c_1), u_4(c_1, c_4), u_2(c_1, c_2), u_3(c_2, c_3), u_6(c_2, c_6), u_5(c_1, c_5), u_8(c_5, c_8), \\ u_7(c_5, c_7), u_9(c_7, c_9), u_{10}(c_7, c_{10}), u_{11}(c_5, c_{11}).$$

5.2.3 Summary

We presented six possible strategies for discourse planning of a given complex concept description. We concentrated on three different goals

1. maximization of local referential-coherence (CT);
2. minimization of overall conceptual distance (mCD);
3. minimization of change in the discourse plan between consecutive edits (user-driven, depth-first, relation-priority depth-first).

If on one side *maximizing the referential coherence* (CT) among discourse units seemed to be a good planning strategy, on the other side we noticed that the overall conceptual proximity in the generated plans for several tree topologies was not satisfactory. We introduced then the measure of *conceptual distance* (mCD) applying it separately in a first attempt, and in hybrid approaches next. In *hybrid planning* we tried two strategies: seeking goal 1 and 2 in sequence (CT-mCD), and in reverse order (mCD-CT). This second hybrid strategy gave an interesting result, reported in proposition 4.

While these approaches work well when we consider the complex concept description as a static input, they fail when we want that the input be created incrementally by a user requesting a sequence of plan generations. From a human-machine interaction viewpoint we would like to minimize the changes in the plan between consecutive edits. This is achieved introducing the third goal. *User-driven planning* answers this purpose but doesn't pay-off the bad average performance in terms of local coherence or conceptual distance. *Depth-first planning* although very simple, yields a better trade-off of the three goals.

We proposed then a last strategy (*relation-priority depth-first*) which requires that roles be augmented in the domain ontology with ordering annotations, specifying a partial order valid among those roles having the same domain concept. The natural rationale behind this is that when describing a concept, relations and attributes that better characterize the concept under exam should be planned first, leaving secondary roles for subsequent positions in the plan.

5.3 Sentence Planning

The module that usually comes next to a text planner in most NLG systems is a *sentence planner* (otherwise called *microplanner*). It is widely recognized (even if there still is considerable debate in the NLG research community) that the main tasks of a sentence planner are

- lexicalization
- aggregation

- referring expression generation

Lexicalization means choosing the right words and syntactic structures to effectively communicate the message encoded in a text plan. Given the NLG system we are going to use (KPML), this process is part of the linguistic realization module, and it will be described in section 5.4.

Aggregation deals with the quantity of information that each sentence in the text must contain.

Referring expression generation suggests which phrases should be used to refer to each domain entity found in the text plan.

Given these three phases, systems available to-date employ one of two possible solutions as described in [Reiter and Dale, 2000]:

- a blackboard architecture, where no specific ordering is imposed over the abovementioned phases;
- a pipelined architecture, which the various phases come in a pre-specified order.

In our system we start with sentence aggregation, followed by referring expressions generation, and finally we generate a sentence plan.

5.3.1 Sentence aggregation

Aggregation can be seen as the task of combining several input elements into a more complex structure for the sake of conciseness, coherence, fluency and conciseness.

[Cheng *et al.*, 1997] give an excellent definition of aggregation that reads as follows:

> *Functioning as one or a set of processes acting on some intermediate text structures in text planning, aggregation decides which pieces of structures can be combined together to be realized as complex sentences later on so that a concise and cohesive text can be generated while the meaning of the text is kept almost the same as that without aggregation.*

Skipping sentence aggregation would lead to stilted texts composed by simple subject-verb-object sentences. Starting from the concept expressed in section ..., and using the relation-priority depth-first planning strategy without aggregation we would obtain the following text after linguistic realization:

> u_1I'm looking for a car. u_4The car is made by Land Rover. u_2The car is equipped with an engine. u_3The engine runs on diesel. u_6The engine's displacement size is [...]. u_5The car is sold by a car dealer. u_8The car dealer's name is [...]. u_7The car dealer is located in a city. u_9The city is in Italy. u_{10}The city is in the province of Trento.

Types of Aggregation

In the past twelve years, researchers working on aggregation mainly elaborated on Wilkinson's [Wilkinson, 1995] classification which is based on *locus of process*: *"Something which the various treatments of aggregation have differed on or left vague*

is at exactly what levels of language generation aggregation may take place. . . . In fact, aggregation-like phenomena can occur at such a variety of stages and in such a variety of ways that the term begins to seem stretched beyond its capacity." [Wilkinson, 1995]

The typologies are six:

- **Conceptual aggregation**: this is the deepest locus of aggregation, where a complex concept can possibly be reduced to a simpler equivalent one by means of an inference.
- **Discourse (rhetorical) aggregation**: any operation that applies to a *discourse structure*, *rhetorical structure*, or *text plan* and maps it to a better structure or plan (how "better" must be defined by a metric).
- **Semantic aggregation**: the combination of two or more semantic entities into one by means of semantic grouping; it takes place at a level which is abstracted from syntax, but is language-dependent. [Reape and Mellish, 1999] note that they *"could not find no clear examples of semantic aggregation in the literature which couldn't alternatively be classified as either conceptual, syntactic or lexical aggregation."*
- **Syntactic aggregation**: the most frequent form of aggregation. Aggregation rules that are commonly specified are (a) *subject grouping* rules and (b) *predicate grouping* rules.
- **Lexical aggregation**: includes three types of aggregation: (a) the mapping of more lexical predicates to fewer lexemes, (b) the mapping of (more) lexical predicates to (fewer) lexical predicates and (c) the mapping of (more) lexemes to (fewer) lexemes.
- **Referential aggregation**: this was introduced by [Reape and Mellish, 1999] and is not covered by [Wilkinson, 1995]; it refers to aggregation by means of referring expression generation.

In our setting, aggregation is meant to detect shared concepts and roles, and to combine them in order to reduce redundancies and repetitions in the resulting text. Given a text plan, as sequence of discourse units, we try to aggregate them according to a set of *aggregation template structures* which can be reduced to three types:

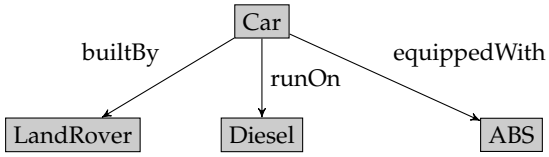
- simple conjunction structure,
- shared subject-predicate,
- syntactic embedding,

which are a subset of the aggregation roles foreseen and described in [Melengoglou, 2002] for the M-PIRO project.

The three abomementioned types mainly fall under the syntactic aggregation typology, and they are described below.

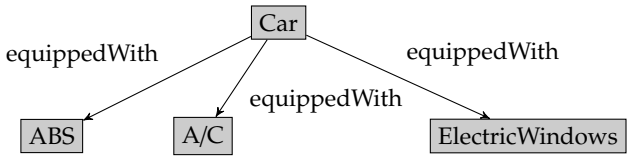
Simple conjunction Simple conjunction can be employed whenever we want to aggregate several roles of the same concept, and the result is the aggregation of several propositions with the same subject.

Let's suppose we have the following relational structure:



Without aggregation the three discourse units would generate three separate sentences: *The car is built by Land Rover. The car runs on diesel. The car is equipped with ABS.* Using simple conjunction we would obtain: *The car is built by Land Rover, it runs on diesel, and it is equipped with ABS*, where for the sake of readability we pronominalized the subjects of the second and third clause.

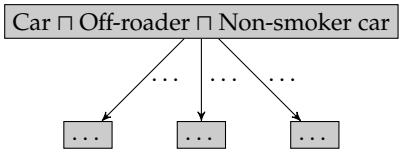
Shared subject-predicate There are cases where two or more consecutive discourse units sharing the same domain concept also have the same role name. This is a case of conjunction with shared subject-predicate, as e.g. in the following relational structure:



Without aggregation we would have: *The car is equipped with ABS. The car is equipped with A/C. The car is equipped with electric windows.*

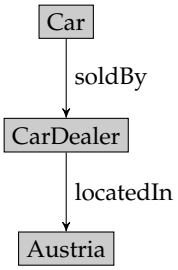
Using simple conjunction we obtain: *The car is equipped with ABS, A/C, and electric windows.*

We can also use shared subject-predicate aggregation when we need to express identity among concepts. Given a relational tree this can happen when a concept is followed by one or more compatibles as in this example:



In aggregated form we have *The car is an off-roader and a non-smoker car.*

Syntactic embedding With this kind of aggregation we have a dominant proposition a secondary proposition which is realized as a subordinated constituent as e.g. a non-defining relative clause. Starting from the following relational tree



we could obtain this aggregated form: *The car is sold by a car dealer who is located in Austria.*

Aggregation Template Structures

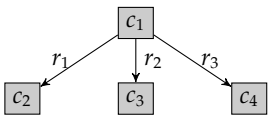
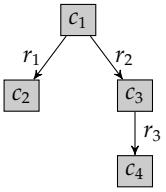
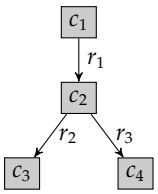
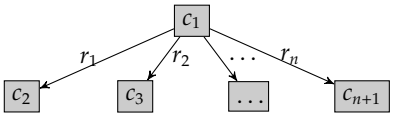
The aggregation template structures we mentioned above are now formally listed in table 5.1 according to the number of discourse units we want to aggregate.

Table 5.1 takes into consideration all subtree patterns we try to recognize in a given relational tree. A maximum number of aggregatable units must be defined and it represents the maximum value that can be assigned to variable n. We define with n_u the number of unique roles with the same domain concept (c_1 as shown in templates n.2 and n.3).

Table 5.1: Aggregation template structures

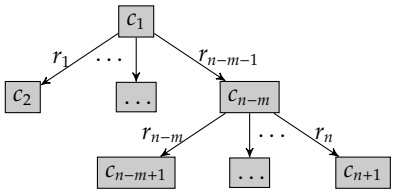
Units	ID	Template	Aggregation
2	2.1	$c_1 \sqcap c_2 \sqcap c_3$	**shared subject-predicate**
	2.2	c_1 with r_1 to c_2 and r_2 to c_3	a) **simple conjunction** b) **shared subject-predicate** (if $r_1 = r_2$)
	2.3	c_1 — r_1 — c_2 — r_2 — c_3	**syntactic embedding**
3	3.1	$c_1 \sqcap c_2 \sqcap c_3 \sqcap c_4$	**shared subject-predicate**

continued on next page

Units	ID	Template	Aggregation
	3.2		a) **simple conjunction** b) **shared subject-predicate** (if $r_1 = r_2 = r_3$) c) **simple conjunction** (between different roles) + **simple conjunction** (for roles that are equal, if either $r_1 = r_2$ or $r_2 = r_3$)
	3.3		a) **simple conjunction** + **syntactic embedding** b) **shared subject-predicate** + **syntactic embedding** (if $r_1 = r_2$)
	3.4		**syntactic embedding** + **shared subject-predicate** ($r_2 = r_3$)
n	n.1	$c_1 \sqcap c_2 \sqcap c_3 \sqcap \ldots \sqcap c_{n+1}$	**shared subject-predicate**
	n.2		a) **simple conjunction** (for $n \leq 3$ and $r_1 \neq r_2 \neq \cdots \neq r_n$) b) **shared subject-predicate** (if $r_1 = r_2 = \cdots = r_n$) c) **simple conjunction** (between different roles, if $2 \leq n_u \leq 3$) + **shared subject-predicate** (for roles that are equal[1])

continued on next page

[1]Note that the planning algorithm we have chosen (relation-priority depth-first) keeps the roles (descending from the same concept) ordered according to their priority, where multiple instances of the same role are always consecutive in the text plan, and never mixed-up with other roles having their same priority.

Units	ID	Template	Aggregation
	n.3		a) if $n = m + 2$, **syntactic embedding** + **shared subject-predicate** (if $m \geq 1$) b) **simple conjunction** (if $m + 2 < n \leq m + 3$ and $r_1 \neq r_2 \neq \cdots \neq r_{n-m-1}$) + **syntactic embedding** + **shared subject-predicate** (if $m \geq 1$) c) **shared subject-predicate** (if $n > m + 2$ and $r_1 = r_2 = \cdots = r_{n-m-1}$) + **syntactic embedding** + **shared subject-predicate** (if $m \geq 1$) d) **simple conjunction** (between different roles $\in \{r_1, \ldots, r_{n-m-1}\}$ if $2 \leq n_u \leq 3$) + **shared subject-predicate** (for roles $\in \{r_1, \ldots, r_{n-m-1}\}$ that are equal) + **syntactic embedding** + **shared subject-predicate** (if $m \geq 1$) For all n.3 templates these conditions must hold: $m \geq 0$ and $r_{n-m} = \cdots = r_n$.

For the patterns expressed above and in particular for $n \in \{2, 3, 5, 6\}$ we show in table 5.2 several examples of aggregation. For $n = 6$ and $n = 7$ in some cases we exceed the limit of maximum number of clauses (with different roles) aggregatable in one sentence by means of simple conjunction ($n_u = 4$), therefore we need to use two sentences. On the opposite, in 8.2.c even if we have 8 propositions, we are able to aggregate them all into one single sentence because $n_u = 3$.

Note that for the sake of readability the last column of table 5.2 shows the final surface form after aggregation and pronominalization.

Table 5.2: Aggregation examples

ID	Propositions	Aggregated form
2.1	The car is an off-roader. The car is a non-smoker car.	The car is an off-roader and a non-smoker car.
2.2.a	The car is a Land Rover. It is equipped with ABS.	The car is a Land Rover and it's equipped with ABS.
2.2.b	The car is equipped with air conditioning. The car is equipped with electric windows.	The car is equipped with air conditioning and electric windows.
2.3.a	The car is equipped with an engine. The engine runs on diesel.	The car is quipped with an engine that runs on diesel.
3.1	The car is an off-roader. The car is a demonstration car. The car is a non-smoker car.	The car is an off-roader, a demonstration car, and a non-smoker car.

continued on next page

ID	Propositions	Aggregated form
3.2.a	The car is a Land Rover. Its model is Defender. It is equipped with a traction control system.	The car is a Land Rover, its model is Defender, and it is equipped with a traction control system.
3.2.b	The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with electric windows.	The car is equipped with ABS, air conditioning, and electric windows.
3.2.c	The car is a Land Rover. The car is equipped with ABS. The car is equipped with air conditioning.	The car is a Land Rover and it's equipped with ABS and air conditioning.
3.3.a	The car is an off-roader. It is equipped with an engine. The engine runs on diesel.	The car is an off-roader and it's equipped with and engine that runs on diesel.
3.3.b	It is equipped with ABS. The car is equipped with an engine. The engine runs on diesel.	The car is quipped with ABS and an engine that runs on diesel.
3.4	The car is equipped with an engine. The engine runs on gasoline. The engine runs on methane.	The car is quipped with an engine that runs on gasoline and methane.
5.1	*similar to 3.1*	
5.2.a	*similar to 3.2.a*	
5.2.b	*similar to 3.2.b*	
5.2.c	The car is a Land Rover. The car's model is Defender. The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with electric windows.	The car is a Land Rover, its model is Defender and it's equipped with ABS and air conditioning.
5.3.b	The car is a Land Rover. The car's model is Defender. The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with an engine. The engine runs on diesel.	The car is a Land Rover, its model is Defender and it's equipped with ABS and an engine that runs on diesel.
5.3.c	*similar to 3.3.b*	
5.3.d	The car is a Land Rover. The car's model is Defender. The car is equipped with an engine. The engine runs on gasoline. The engine runs on methane.	The car is a Land Rover, it's model is Defender, and it's equipped with an engine that runs on gasoline and methane.
6.2.d $n_u = 4$	The car is a Land Rover. The car's model is Defender. The car's color is yellow. The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with electric windows.	The car is a Land Rover, its model is Defender and it's color is yellow. It's equipped with ABS, air conditioning, and electric windows.

continued on next page

ID	Propositions	Aggregated form
7.2.c $n_u = 4$	The car is a Land Rover. The car's model is Defender. The car's color is yellow. The car's color is blue. The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with electric windows.	The car is a Land Rover, its model is Defender and it's color is yellow and blue. It's equipped with ABS, air conditioning, and electric windows.
8.2.c	The car is a Land Rover. The car's color is yellow. The car's color is blue. The car's color is red. The car is equipped with a traction control system. The car is equipped with ABS. The car is equipped with air conditioning. The car is equipped with electric windows.	The car is a Land Rover, it's color is yellow, blue, and red, and it's equipped with a traction control system, ABS, and air conditioning.

The next step is meant to find a way to compute the best match of the patterns of table 5.1 for a given relational tree taking into consideration that the text planning algorithm we choose is the *relation-priority depth-first*.

Best template structure matching

The idea behind is to linearize both the relational tree (according to the chosen planning algorithm) and the template structures, seeking the best covering match of the linearized templates in the plan that minimizes the number of sentences in the outcoming sentence plan.

The linearized templates we foresee are listed in table 5.3. Concepts are represented as `Ci`, `Ci+1`, `Ci+2`, etc. where the emphasized index stands for the level in the tree where the concept (as node) is situated. Roles are represented as `Rj`, `Rj+1`, `Rj+2`, etc. where the emphasized index is the same for edges of the tree that represent the same role.

#	Linearized template	Original ID
1	`Ci+,`	n.1
2	`(CiRjCi+1,)+`	n.2.b
3	`(CiRjCi+1,)+(Ci+1Rj+1Ci+2,)+`	n.3.a, n.3.b, n.3.c
4	`(CiRjCi+1,)+(CiRj+1Ci+1,)+`	n.2.c
5	`(CiRjCi+1,)+(CiRj+1Ci+1,)+(Ci+1Rj+2Ci+2,)+`	n.3.d
6	`(CiRjCi+1,)+(CiRj+1Ci+1,)+(CiRj+2Ci+1,)+`	n.2.a, n.2.c
7	`(CiRjCi+1,)+(CiRj+1Ci+1,)+(CiRj+2Ci+1,)+(Ci+1Rj+3Ci+2,)+`	n.3.d

Table 5.3: Linearized templates

Table 5.3 groups the linearization of the allowed tree templates ordered by increasing number of constituents; it also shows that each linearization corresponds to one or more tree templates.

In order to find the best template structure match, we need to convert the text plan output by the previous phase using the same notation employed for the linearizations above.

Let's suppose we have the tree of figure 5.8. Provided we use the planning algorithm proposed in section 5.2.2 (relation-priority depth-first planning), the text plan with the notation introduced above would be:

```
C1C1C1,C1R1C2,C1R2C2,C2R3C3,C2R3C3,C1R4C2,C2R5C3,C2R6C3,
                                        C3R7C4,C3R8C4,C2R9C3,    (5.1)
```

The list of templates needs to be instantiated: indexes `i` and `j` must be initialized according to the index (level) of the first concept and the first role in the text plan respectively. We obtain a list of regular expressions we call *aggregation patterns* or simply *patterns*.

```
1  C1(C1)+,
2  (C1R1C2,)+
3  (C1R1C2,)+(C2R2C3,)+
4  (C1R1C2,)+(C1R2C2,)+
5  (C1R1C2,)+(C1R2C2,)+(C2R3C3,)+
6  (C1R1C2,)+(C1R2C2,)+(C1R3C2,)+
7  (C1R1C2,)+(C1R2C2,)+(C1R3C2,)+(C2R4C3,)+
```

The pattern matching is done starting from the longest pattern #7 to the shortest one #1. The first pattern that matches is #1 followed by #5. We are able to aggregate the first three compatible concepts (`C1C1C1,`) in a first sentence, followed by another sentence that aggregates four units (`C1R1C2,C1R2C2, C2R3C3,C2R3C3,`). After these first two hits, no other pattern matches the remaining part of the plan:

```
C1R4C2,C2R5C3,C2R6C3,C3R7C4,C3R8C4,C2R9C3,
```

We need to re-instantiate the list of templates setting $i = 1$ and $j = 4$. The patterns become then:

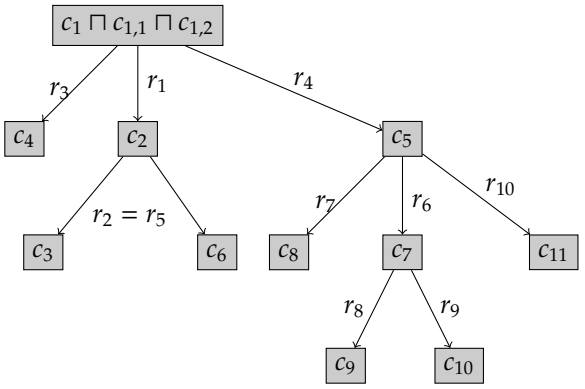


Figure 5.8: A query tree waiting to be linearized

1	(C1,)+
2	(C1R4C2,)+
3	(C1R4C2,)+(C2R5C3,)+
4	(C1R4C2,)+(C1R5C2,)+
5	(C1R4C2,)+(C1R5C2,)+(C2R6C3,)+
6	(C1R4C2,)+(C1R5C2,)+(C1R6C2,)+
7	(C1R4C2,)+(C1R5C2,)+(C1R6C2,)+(C2R8C3,)+

This time there is only one pattern that matches, namely #3. The part of the text plan that remains to be matched is

C2R6C3,C3R7C4,C3R8C4,C2R9C3,

The templates need to be instantiated again with $i = 2$ and $j = 6$ yielding these patterns:

1	(C2(C2)+,)+
2	(C2R6C3,)+
3	(C2R6C3,)+(C3R7C4,)+
4	(C2R6C3,)+(C2R7C3,)+
5	(C2R6C3,)+(C2R7C3,)+(C3R8C4,)+
6	(C2R6C3,)+(C2R7C3,)+(C2R8C3,)+
7	(C2R6C3,)+(C2R7C3,)+(C2R8C3,)+(C3R10C4,)+

Again, pattern #3 is the only one that matches. There are two more units to be matched:

C3R8C4,C2R9C3,

This time we need to instantiate just the first four templates (whose minimum length doesn't exceed the remaining two units), with $i = 3$ and $j = 8$:

1	C3(C3)+,
2	(C3R8C4,)+
3	(C3R8C4,)+(C4R9C5,)+
4	(C3R8C4,)+(C3R9C4,)+

where only pattern #2 matches. Finally the last set of patterns is generated setting $i = 2$ and $j = 9$, in order to match the very last unit with pattern #2 shown below.

C2R9C3,

1	C2(C2)+,
2	(C2R9C3,)+

Summarizing, given the query tree of figure 5.8, the text plan resulting from it is composed by 11 clauses which, according to the proposed templates, can be joined to form 6 sentences ($S_1 \dots S_6$) as reported below:

Sentences		**Patterns**	
S_1	$c_1 \sqcap c_{1,1} \sqcap c_{1,2}$	C1C1C1,	1
S_2	$c_1 r_3 c_4 + c_1 r_1 c_2 + c_2 r_2 c_3 + c_2 r_5 c_6$	C1R1C2,C1R2C2,C2R3C3,C2R3C3,	5
S_3	$c_1 r_4 c_5 + c_5 r_7 c_8$	C1R4C2,C2R5C3,	3
S_4	$c_5 r_6 c_7 r_8 c_9$	C2R6C3,C3R7C4,	3
S_5	$c_7 r_9 c_{10}$	C3R8C4,	2
S_6	$c_5 r_{10} c_{11}$	C2R9C3,	2

Algorithms

In this section we describe two algorithms: The first one (algorithm 7) is needed to obtain the text plan with the notation introduced in the previous section; the second one (algorithm 8) finds the best covering match of the text plan using the aggregation templates of table 5.3.

Algorithm 7 Generation of the text plan ...

```
u_1 ← root node
p ← "" {text plan as string}
P ← empty vector {text plan as vector of pointers to query entities}
i_r ← 1 {role counter}
r_-1 ← null {previous role}
call CalculateTextPlan(u_1)

procedure CalculateTextPlan(u)
append "C" & level(u) to p
append main concept of u to P;
C_u ← list of compatible concepts in node u
for all c ∈ C_u do
   append "C" & level(u) to p
   append c to P
end for
append "," to p
R_u ← all edges (roles) from u {left to right}
for all r ∈ R_u do
   v ← target node of edge r
   {same roles (r = r_-1) keep the same index i_r in the new notation}
   if r ≠ r_-1 then
      i_r ← i_r + 1
   end if
   append "C" & level(u) & "R" & i_r & "C" & level(u) + 1 & "," to p
   append main concept of u to P
   append r to P
   append main concept of v to P
   r_-1 ← r
   call CalculateTextPlan(v)
end for
end procedure
```

The outputs of this algorithm both represent the text plan as serialization of a given query tree:

- a string p with the notation shown above;
- a vector P of pointers to the entities composing the query.

Algorithm 8, instead, takes the output string p and calculates the best covering match of the text plan using the aggregation templates of table 5.3. The templates are instantiated at the beginning according to the indexes assigned to the first concept and the first role of p. The resulting patterns are matched

against p starting from the longest to the shortest one. The matching template number and the match are saved; then the match is removed from p. As soon as none of the patterns matches, the templates are instantiated again as above. The process stops when p is empty. The output consists of two lists: a list containing the sequence of matches, and a list of the template IDs corresponding to each match.

Algorithm 8 Calculation of the best covering match

input T {list of templates to be instantiated}
input p {text plan as string}
input P {text plan as vector of pointers to query entities}
$p_{tmp} \leftarrow p$ {tmp copy of text plan}
$T_{inst} \leftarrow$ *empty list* {list of patterns as instances of T}
$L_m \leftarrow$ *empty list* {list of string matches}
$L_t \leftarrow$ *empty list* {list of template IDs that matched}
$M \leftarrow$ *empty list* {array containing in M[0] the match, in M[1] the ID of the matching pattern}
$i \leftarrow$ index (level) of first concept in p_{tmp}
$j \leftarrow$ index of first role in p_{tmp}
$T_{inst} \leftarrow$ *generatePatterns*(T, i, j)
while p_{tmp} not empty **do**
  $M \leftarrow$ *getMatch*$(T_{inst}, \&p_{tmp})$
  if M = *null* **then**
    $i \leftarrow$ index (level) of first concept in p_{tmp}
    $j \leftarrow$ index of first role in p_{tmp}
    $T_{inst} \leftarrow$ *generatePatterns*(T, i, j)
    $M \leftarrow$ *getMatch*$(T_{inst}, \&p_{tmp})$
  end if
  append M[0] to L_m {match}
  append M[1] to L_t {template ID}
end while
output L_m
output L_t

function *generatePatterns*(T, i, j)
for all $t \in T$ **do**
  substitute i, j in t
  calculate indexes of concepts `C` and roles `R` in T
  append t to T_{inst}
end for
return T_{inst}
end function

function *getMatch*(T_{inst}, p_{tmp})
$M \leftarrow$ *null*
for i = *length*(T_{inst}) **downto** 1 **do**
  if *existsMatch*$(T_{inst}[i], p_{tmp})$ **then**
    M[0] $\leftarrow$ match $T_{inst}[i]$ in p_{tmp}
    M[1] $\leftarrow i$
    remove M[0] from p_{tmp}

```
      break
    end if
  end for
  return M
  end function
```

Some clarifications about algorithm 8 are necessary.. . . . . .

5.3.2 Referring expressions generation

Referring expressions represent the ways we can consider to refer to discourse entities in a message or text in general. As [Reiter and Dale, 2000] clearly explain, *the symbolic names of knowledge base entities within these messages need to be replaced by the semantic content of noun phrase referring expressions that will be sufficient to identify the intended referents to the hearer*. The reference to a discourse entity can be done by means of a noun phrase in several ways:

1. **definite noun phrases** (as e.g. '*the car*'): these are used when referring to an entity that has already been introduced before, or when the entity is assumed to be known or inferable by the hearer;
2. **indefinite noun phrases** (as e.g. '*a car*'): this is the case when we refer to a new discourse entity that hasn't been previously mentioned;
3. **definite pronouns** (he, she, it, . . .) usually anaphoric[2], and typically referring to entities mentioned in the same or the previous sentence;
4. **indefinite pronouns** (one, as in 'the regular *one*');
5. **relative pronouns as subject** (who, that, which), referring to an entity contained in the previous clause;
6. **names**, where named entities can be referred to using portions of their name ('The writer Richard Wright' → 'Richard Wright')

Of the above categories, we restrict the generation of referring expressions to definite noun phrases (as subject), indefinite noun phrases (as direct or indirect object), definite pronouns (as subject), and relative pronouns (as subject).

We also report the use of

- **possessive pronouns** when referring to one of the attributes of a previously mentioned entity (e.g. *The engine's displacement size is 2500 cc, and **its** weight is 250 kg*);
- **relative pronouns** used as possessives (like *whose*), to incorporate a reference to the possessor of an attribute following the pronoun. The possessor is usually introduced in the previous clause within the same sentence (e.g. *I'm looking for a car **whose** make is Lada*).

[2] A reference is said to be *anaphoric* if its interpretation depends on a preceding entity in the discourse, which is called the *antecedent*.

We start by listing some constraints we have to take into account during this phase.

The first and most general constraint is that all entities of the text plan (except the subject of the first unit which is in first person singular form) will be rendered in third person singular form.

Moreover, for each one of the referring expressions we use, there are certain constraints we have to stick to that limit the position that the expression can occupy within a sentence:

indefinite noun phrases (R-INP) are always in (direct or indirect) object position, and they are used the first time an entity appears in the text;

definite noun phrases (R-DNP) are always in subject position; otherwise this would mean that the entity, being also in object position the first time it was mentioned in the text, is co-referenced by two roles, which is impossible for our definition of conjunctive query (see Section . . .).

definite pronouns (R-DP) are always in subject position; in this case we must be careful to respect the gender of the referent;

relative pronouns as subject (R-RPS) which must be the same as the object of the previous unit;

possessive pronouns (R-PP) can only precede a subject; they must refer to the subject of the previous unit, not to the object, otherwise

relative pronouns as possessives (R-RPP) would be the right choice.

Given these constraints it turns out to be very easy to assign the first two referring expressions: for each discourse unit, the first entity is tagged as a *definite noun phrase* (R-DNP) and the second as an *indefinite noun phrase* (R-INP). At this point we have to note that this pre-assignment of a definite or indefinite status to entities will not affect those entities that will be lexicalized either as proper nouns or uncountable nouns. We will see this further on, when we handle the generation of sentence plans.

From this point on, the task is to deal with the pronominalization of the first entity of each unit. We could easily borrow the idea of the local focus of attention, in particular the pronominalization strategy proposed by Centering Theory [Grosz *et al.*, 1995], which states in **Rule 1** that

> If any element of $C_f(U_n)$ is realized by a pronoun in U_{n-1}, then the $C_b(U_{n+1})$ must be realized by a pronoun also.

In other terms, citing again the authors, this means that

> [. . .] no element in an utterance can be realized as a pronoun unless the backward-looking center of the utterance is realized as a pronoun also.

where utterance (U_n) is what we call discourse unit or simply unit (with a lower-case notation u_n).

This rule, though, does not discern among the four categories of pronouns we have, indicating which one we should use. In principle we could simply

use *definite pronouns* (R-DP), but we want to go beyond the simple achievement of grammatical sentences, having a higher degree of fluency, conciseness, and avoiding repetitions.

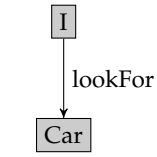
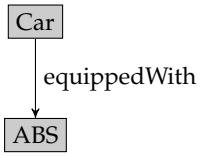
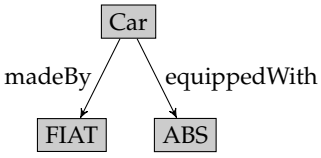
The previous phase (see Section 5.3.1) yielded the aggregation (where possible) of several discourse units into what will become multi-clausal sentences. Within the same sentence we can have clauses (units) whose first entity (subject) is the same as the second entity (object) of the previous unit. This is a case in which the pronoun of the latter unit is a *relative pronoun as subject* (R-RPS), of what will become a relative clause.

If the role expressed in a unit is concrete (i.e. an attribute of the first entity), and the first entity of the current unit is the same as the first entity of the previous unit, the role will be the subject, prepended by a *possessive pronoun* (R-PP). The two consecutive units don't need to be part of the same sentence.

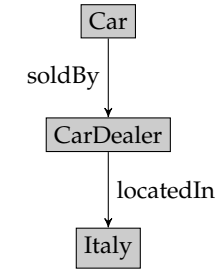
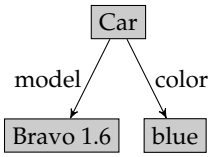
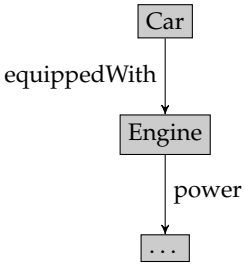
Finally, in the same setting as the previous paragraph, where instead the first entity of the current unit has to correspond to the second entity of the previous unit, we would prepend the role (subject) of the present unit with a *relative pronoun used as possessive* (R-RPP).

Table 5.4 reports an example for each one of the referring expressions we took into consideration.

Table 5.4: Examples of usage of referring expressions

Ref. expr.	Query tree	Sentence
R-INP		I'm looking for a car.
R-DNP		The car is equipped with ABS.
R-DP		The car is made by FIAT and it is equipped with ABS.

continued on next page

Ref. expr.	Query tree	Sentence
R-RPS		The car is sold by a car dealer who is located in Italy.
R-PP		The car's model is Bravo 1.6 and its color is blue.
R-RPS		The car is equipped with an engine whose power is ....

We condense now all the previous considerations, rules, and constraints into an algorithm for the generation of referring expressions. The input of the algorithm is an ordered list of all entities we have in our text plan, with the additional aggregation information obtained from the aggregation algorithm.

The output will be the same list of entities, where each entity will be completed with additional information about the referring expression to be used.

To accomplish this task we need a few functions:

- `getUnit`(c_i) returns the discourse unit where entity c_i is to be found;
- `getPreviousUnit`(u_k) returns the unit preceding u_k;
- `getPreviousEntity`(c_i) returns the entity preceding c_i;
- `getFirstEntity`(u_k) returns the first entity in u_k;
- `getNextEntity`(c_i) returns the next (to the current one) entity in u_k;
- `getLastEntity`(u_k) returns the last entity u_k;
- `getEntityPosition`(c_i) returns the relative position of entity c_k within its discourse unit; the position is a positive integer in $\{1, 2\}$ for units like $c_j r_k c_l$ or an integer in a bigger set $\{1, 2, 3, \ldots\}$ for units such as $c_i \sqcap c_{i,1} \sqcap c_{i,2} \sqcap \ldots$ which represent the conjunction of two or more compatible concepts;

- `getSentence(u_k)` returns the sentence to which unit u_k has been assigned after aggregation;
- `inSameSentence(u_k, u_l)` which returns `true` if the two discourse units u_k and u_l are part of the same sentence after aggregation, otherwise it returns `false`;
- `sameConcept(c_i, c_j)` returns `true` if the two entities refer to the same concept;
- `setRefExpr(c_i, refExpr)` sets the given referring expression `refExpr` in c_i;
- `existsRole(c_i, c_j)` returns `true` if c_i and c_j are connected by a role, `false` otherwise;
- `isConcreteRole(c_i, c_j)` returns `true` if the role having c_i as domain and c_j as range is a *concrete role* (attribute), `false` if it is an *abstract role* (relation).

Algorithm 9 Generation of appropriate referring expressions for each entity present in a given text plan

```
input P {text plan as vector of discourse entities, which are uniquely identified, even
though it can happen that two entities refer to the same KB concept}
for all c ∈ P do
  u_cur ← getUnit(c)
  u_prev ← getPreviousUnit(u);
  if getEntityPosition(c) = 1 then
    c_next = getNextEntity(c)
    setRefExpr(c, R-DNP)
    if u_prev ≠ NULL then
      if sameConcept(getFirstEntity(u_prev), c) then
        setRefExpr(c, R-DP)
        if existsRole(c, c_next) then
          if isConcreteRole(c, c_next) then
            setRefExpr(c, R-PP)
          end if
        end if
      else if inSameSentence(u_prev, u_cur) then
        if sameConcept(getLastEntity((u_prev), c)) then
          setRefExpr(c, R-RPS)
          if existsRole(c, c_next) then
            if isConcreteRole(c, c_next) then
              setRefExpr(c, R-RPP)
            end if
          end if
        end if
      end if
    else
      setRefExpr(c, R-INP)
    end if
  end for
```

A few comments at this point are necessary. First of all, the use of possessive pronouns (R-PP) and relative pronouns as possessives (R-RPP) is not restricted to entities connected with a concrete role: We can have cases where an entity is followed by an abstract role which behaves as an attribute and is therefore rendered as a substantive instead of a predicate, as abstract roles usually are.

For example, the query of figure 5.9 would be rendered as "I am looking for a car whose make is Santana", where the abstract role *make* is rendered as the subject of the second unit, and the reference to car is incorporated into the relative pronoun (*whose*).

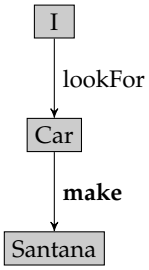


Figure 5.9: Query with abstract role (*make*) that is rendered as substantive.

This is to be considered an exception, since the rule is that abstract roles are usually rendered as a predicate (as e.g. the abstract role *lookFor*).

Another issue regards the correct choice of a pronoun according to the gender of the *referent* (third person singular), and the fact that *they* are either human or non human entities. The problem arises when we want to refer to a single definite person androgynously, i.e. with a gender-neutral pronoun. There are various viable solutions. We could try to **avoid using the pronoun**, but this would lead to annoying repetitions of the name that should have been pronominalized. In order to avoid sexist writing we could **alternate male and female pronouns**: in this case this would be pretty confusing for the user. We very often see people using **both pronouns together** but this is considered by readers and writers stylistically inelegant. Excluding the possibility of **inventing a new pronoun**, what remains —and this is the solution we adopt— is resorting to plural pronouns such as *they*, and *their* for singular uses. This is called the **singular they**.

singular they

Singular they is a popular, non-technical expression for uses of the pronoun they (and its inflected forms) when plurality is not required by the context. Singular they remains morphologically and syntactically plural, and its use as pronoun of indefinite gender and indefinite number is well established in speech and writing, even in literary and formal contexts [Merriam-Webster, 2007]. We weaved an example of *singular they* usage in the previous paragraph, in correspondence of the margin note.

The assignment of the correct pronoun to each pronominalizable entity will be dealt in detail in the following section. We only anticipate in Table 5.5 the set of all pronouns we are going to use.

Type	Non-human	Human		
	Indefinite	Masculine	Feminine	Indefinite
R-DP	it	he	she	they
R-RPS	that	who/that		
R-PP	its	his	her	their
R-RPP	whose			

Table 5.5: Complete set of singular pronouns used

5.3.3 Generation of a Sentence Plan in SPL

With the outputs obtained from the discourse planning, sentence aggregation, and referring expression generation phases, we are ready to generate the input for the linguistic realizer. The input is called *sentence plan* and the language used is the *Sentence Plan Language* or simply SPL, a language devised by Robert Kasper [Kasper, 1989]. The details of this formalism are thoroughly explained in Section 5.5.4.

In short, SPL is the form of non-linguistic input adoptet by several linguistic realizers, among which we mention KPML [Bateman, 1997a], the one we adopted. In a more general way we can say that an SPL is the semantic specification of a sentence.

We start with some examples, where for each one we show the query, the sentence plan it is mapped into, and the KPML-generated text.

Example 1 In this first example the query is a conjunction of three compatible concepts: `Used-car`, `Off-Roader`, and `Non-smoker-car`.

Query

Used-car ⊓ Off-roader ⊓ Non-smoker-car

Sentence Plan

```
(s1 / class-ascription
  :modality must
  :domain
    (c1 / used-car
      :determiner the)
  :range
    (:and
      (c2 / off-roader)
      (c3 / non-smoker-car))
)
```

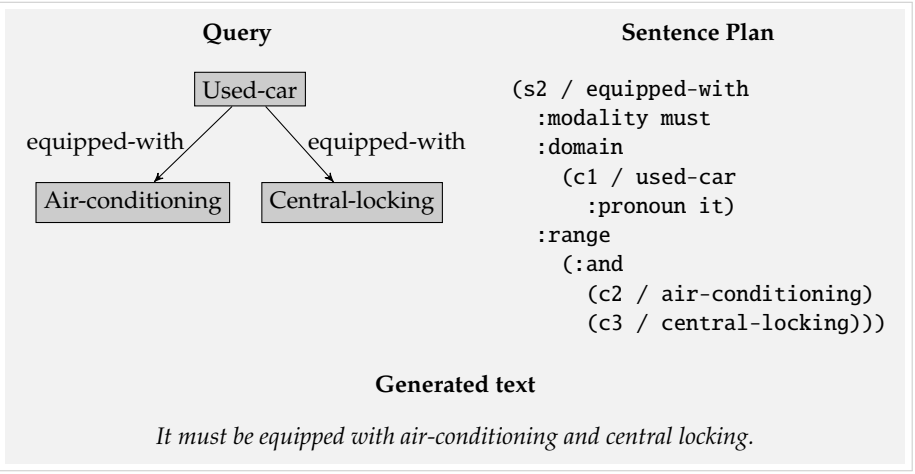
Generated text

The used-car must be an off-roader and a non-smoker car.

These three concepts are represented in the sentence plan by three variables `c1`, `c2`, and `c3`; `s1` instead, is the variable representing the *relational process* we use in order to verbalize our input query in a descriptive way. The process is a `class-ascription`, one of the process types defined in the *Merged Upper*

Model (see Section 5.5.3), a general task- and domain-independent linguistically-motivated ontology used for mediating between domain knowledge and the linguistic realizer. A `class-ascription` process must have at least two participants, which are called `:domain` and `:range`: The domain is the first concept (`Used-car`), and the range is the conjunction of all other concepts (in this case `Off-roader` and `Non-smoker-car`). KPML generates the class ascription as a copula that relates domain and range as subsets i.e. the used car we are looking for is contained in the intersection of the sets of all off-roaders and non-smoker cars. We also added the `:modality` property to the class ascription process, in order to emphasize that this is a query expressing user requirements.

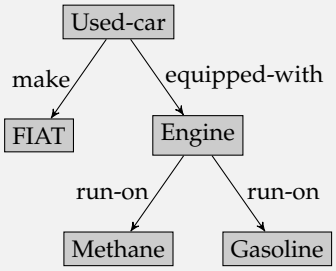
Example 2 Here the query is composed by three concepts (`Used-car`, `Air-conditioning`, and `Central-locking`) and two instances of the same role (`equipped-with`).



The derived sentence plan contains a process named `equipped-with` which is subsumed by the more general Upper-Model (UM) concept called `generalized-possession`. The participants are the used car as `:domain` and both air-conditioning and central-locking as `:range`. We decided to pronominalize the subject of the sentence.

Example 3 We show here a more complex query, containing five concepts and three roles.

Query



Sentence Plan

```
((s3 s4)
 (s3 / property-ascription
   :modality must
     :domain
       (c1 / make
         :lex make
         :determiner the
         :owned-by
           (c2 / used-car
             :pronoun it
             :determiner the))
     :range
       (c3 / template
         :pattern "FIAT"))
 (s4 / equipped-with
   :lex equipped-with
   :modality must
   :domain
     (c2 / used-car
       :determiner the
       :pronoun it)
   :range
     (c4 / engine
       :process
         (s5 / run-on
           :tense present
           :actor c4
           :actee
             (:and
               (c5 / methane)
               (c6 / gasoline))))))
```

Generated text

Its make must be FIAT, and it must be equipped with an engine that runs on methane and gasoline.

The sentence plan is made up of two main coordinate clauses, `s3` and `s4`, which are associated to two processes: a `property-ascription` and `equipped-with` (seen in the previous example). The latter contains a further process (`run-on`) that gives additional information about the engine's fuel (`methane` and `gasoline`): This sub-process is realized as a relative clause (*. . . that runs on methane and gasoline*). Since `run-on` is subsumed by the UM-process `dispositive-material-action`, the participants of this process have to be named `:actor` and `:actee`. We also want to remark the use of a possessive (*its*) and a definite pronoun (*it*) referring to used-car, along with the relative pronoun (*that*) referring to the engine, automatically generated by the realizer as subject of the sub-process.

We proceed now formally by describing how to map each one of the templates we listed in Table 5.3 into a corresponding text plan. Indexes of concepts

in that table are equal for concepts on the same level, while here indexes are numbered differently for different concepts. For each template, we show the corresponding generic query, its linearization and the generated sentence plan.

We draw the attention of the reader to the fact that all concepts included in the following sentence plans may seem confusing because of a name duplication. When we write `(ci / ci)`, the first c_i is a variable, the second one is the name of the concept assigned to the variable, also called *type* as we will see in Section 5.5.4.

Template 1.

Query

$c_1 \sqcap c_2 \sqcap \ldots \sqcap c_n$

Linearization

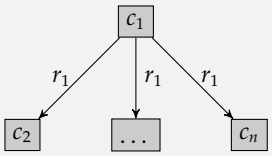
$(c_1, c_2, \ldots, c_n)$

Sentence Plan

```
(s1 / class-ascription
  :modality must
  :domain
    (c1 / c1
      :determiner ???
      :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn)))
```

Template 2. r_1 is an *abstract role* in the domain ontology:

Query



$r1$: abstract role

Linearization

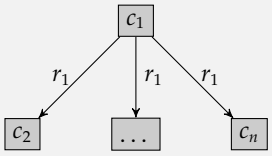
$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n)$

Sentence Plan

```
(s1 / r1
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn)))
```

or, if r_1 is a *concrete role* in the domain ontology:

Query



$r1$: concrete role

Linearization

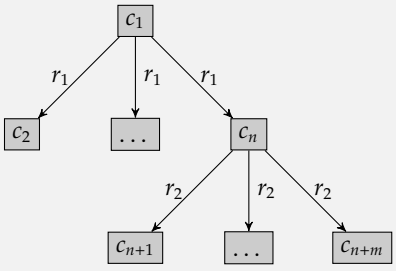
$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n)$

Sentence Plan

```
(s1 / property-ascription
  :modality must
  :domain
    (r1 / r1
      :owned-by
        (c1 / c1
          :determiner ???
          :pronoun ???))
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn)))
```

Template 3. If r_1 and r_2 are both abstract roles:

Query



r_1, r_2: abstract roles

Linearization

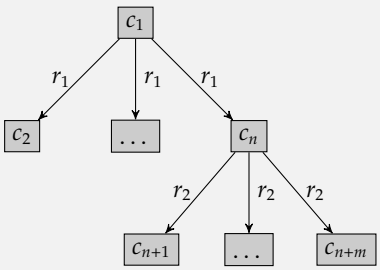
$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n, c_n r_1 c_{n+1}, c_{n+1} r_2 c_{n+2},$
$c_{n+1} r_2 c_{n+3}, \ldots c_{n+1} r_2 c_{n+m})$

Sentence Plan

```
(s1 / r1
  :modality must
  :domain
    (c1 / c1
      :determiner ???
      :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn
        :process
          (r2 / r2
            :tense present
            :actor cn
            :actee
              (:and
                (cn+1 / cn+1)
                (... / ...)
                (cn+m / cn+m))))))
```

otherwise, if r_2 is a concrete role we have:

Query



r_1: abstract role
r_2: concrete role

Linearization

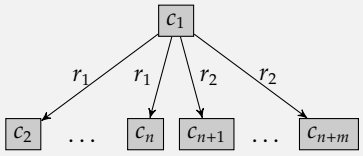
$$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n, c_n r_1 c_{n+1}, c_{n+1} r_2 c_{n+2}, c_{n+1} r_2 c_{n+3}, \ldots c_{n+1} r_2 c_{n+m})$$

Sentence Plan

```
(s1 / r1
  :modality must
  :domain
    (c1 / c1
      :determiner ???
      :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn
        :process
          (r2 / property-ascription
            :tense present
            :extracted-variable-id r2
            :domain
              (r2 / r2
                :relations
                  (o1 / ownership
                    :domain cn
                    :range r2))
            :range
              (:and
                (cn+1 / cn+1)
                (.../...)
                (cn+m / cn+m))))))
```

Template 4. Here we have four possibilities, depending on the fact that r_1 and r_2 can either be abstract or concrete roles. We start with the case that r_1 and r_2 are abstract roles.

Query



r_1, r_2: abstract roles

Linearization

$$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n, c_1 r_2 c_{n+1}, c_1 r_2 c_{n+2}, \ldots, c_1 r_2 c_{n+m})$$

Sentence Plan

```
((s1 s2)
 (s1 / r1
   :modality must
   :domain
     (c1 / c1
       :determiner ???
       :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn)))
 (s2 / r2
  :modality must
  :domain
    (c1 / c1
      :determiner ???
      :pronoun ???)
  :range
    (:and
      (cn+1 / cn+2)
      (.. / ..)
      (cn+m / cn+m))))
```

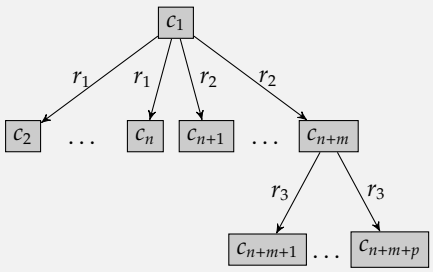
If either one of r_1 or r_2 (or both) is a concrete role (say r_y), the previous sentence plan is no more valid. We need to replace the sentence plan chunk containing r_y (left box below) with the one on the right.

```
(sx / ry
  :modality must
  :domain
    (c1 / c1
      :determiner ???
      :pronoun ???)
```

```
(sx / property-ascription
  :modality must
  :domain
    (ry / ry
      :owned-by
        (c1 / c1
          :determiner ???
          :pronoun ???))
```

Template 5. If $r1$, r_2, and r_3 are abstract roles:

Query



r_1, r_2, r_3: abstract roles

Linearization

$$(c_1 r_1 c_2, \ldots, c_1 r_1 c_n, c_1 r_2 c_{n+1}, \ldots, c_1 r_2 c_{n+m}, c_{n+m} r_3 c_{n+m+1}, \ldots, c_{n+m} r_3 c_{n+m+p})$$

Sentence Plan

```
((s1 s2)
 (s1 / r1
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (... / ...)
      (cn / cn)))
 (s2 / r2
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (cn+1 / cn+1)
      (... / ...)
      (cn+m / cn+m
        :process
          (r3 / r3
            :tense present
            :actor cn+m
            :actee
              (:and
                (cn+m+1 / cn+m+1)
                (... / ...)
                (cn+m+p / cn+m+p))
          )))))
```

If r_1 is a concrete role we should substitute in the previous plan the chunk reported in the left box below with the one on the right box.

```
(s1 / r1
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
```

```
(s1 / property-ascription
  :modality must
  :domain
    (r1 / r1
      :owned-by
        (c1 / c1
          :determiner ???
          :pronoun ???))
```

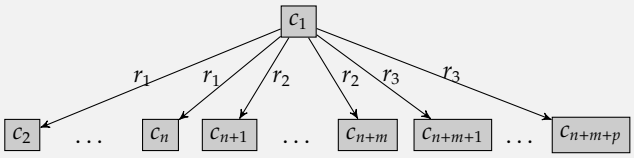
Since r_2 cannot be a concrete role (otherwise c_{n+m} would be a concrete data type, which is not possible because it should be a leaf), the last variant to this sentence plan is that r_3 is a concrete role. The substitution we perform in this case is:

```
(r3 / r3
  :tense present
  :actor cn
  :actee
    (:and
      (cn+m+1 / cn+m+1)
      (... / ...)
      (cn+m+p / cn+m+p)))
```

```
(s3 / property-ascription
  :tense present
  :extracted-variable-id r3
  :domain
    (r3 / r3
      :relations
        (o1 / ownership
          :domain cn
          :range r3))
      :range
        (:and
        (cn+m+1 / cn+m+1)
        (... / ...)
        (cn+m+p / cn+m+p)))
```

Template 6. If $r1$, r_2, and r_3 are abstract roles:

Query



r_1, r_2, r_3: abstract roles

Linearization

$$(c_1r_1c_2, c_1r_1c_3, \ldots, c_1r_1c_n, c_1r_2c_{n+1}, c_1r_2c_{n+2}, \ldots, c_1r_2c_{n+m}, c_{n+m}r_3c_{n+m+1}, \ldots, c_{n+m}r_3c_{n+m+p})$$

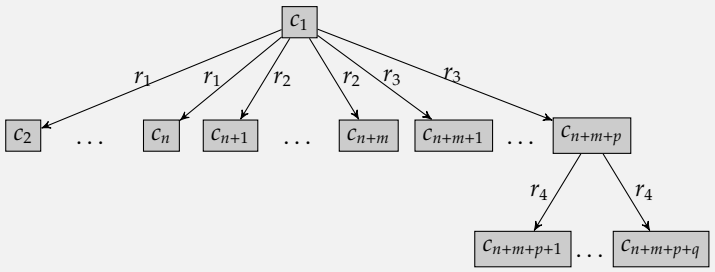
Sentence Plan

```
((s1 s2 s3)
 (s1 / r1
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (.. / ..)
      (cn / cn)))
 (s2 / r2
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (cn+1 / cn+2)
      (... / ...)
      (cn+m / cn+m)))
 (s3 / r3
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
    (cn+m+1 / cn+m+1)
    (... / ...)
    (cn+m+p / cn+m+p))))
```

Otherwise, if any of r_1, r_2, or r_3 is a concrete role, we perform a substitution as we explained for Template 4.

Template 7. If $r1$, r_2, r_3, and r_4 are abstract roles:

Query



r_1, r_2, r_3, r_4: abstract roles

Linearization

$$(c_1 r_1 c_2, c_1 r_1 c_3, \ldots, c_1 r_1 c_n, c_1 r_2 c_{n+1}, c_1 r_2 c_{n+2}, \ldots, c_1 r_2 c_{n+m}, c_{n+m} r_3 c_{n+m+1}, \ldots, c_{n+m} r_3 c_{n+m+p}, c_{n+m+p} r_4 c_{n+m+p+1}, \ldots, c_{n+m+p} r_4 c_{n+m+p+q})$$

Sentence Plan

```
((s1 s2 s3)
 (s1 / r1
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (c2 / c2)
      (.. / ..)
      (cn / cn)))
 (s2 / r2
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
      (cn+1 / cn+2)
      (.. / ..)
      (cn+m / cn+m)))
 (s3 / r3
  :modality must
  :domain (c1 / c1
    :determiner ???
    :pronoun ???)
  :range
    (:and
    (cn+m+1 / cn+m+1)
    (... / ...)
    (cn+m+p / cn+m+p
      :process
        (r4 / r4
          :tense present
          :actor cn+m+p
          :actee
            (:and
              (cn+m+p+1 / cn+m+p+1)
              (... / ...)
              (cn+m+p+q / cn+m+p+q)))))))
```

If any of r_1, r_2, or r_4 is a concrete role, we perform a substitution as explained for Template 4.

Fine tuning of the sentence plans Given the text plan augmented with aggregation information (i.e. the matching templates from Table 5.3) and the referring expression types (Table 5.5) associated to each concept, obtaining the sentence plan(s) is just a matter of mapping the list of aggregation patterns into the corresponding SPL chunks as specified above.

It was not mentioned above that the sentence plan(s) must be preceded by an introductory one, which declares what the user is looking for (root concept, say c_1), at least in our setting where the complex concept description represents

introductory sentence plan

a user query.

One possible introductory sentence could be "I'm looking for c_1", where c_1 is the root concept of the query, and the corresponding sentence plan would be the following:

```
(s0 / look-for
  :actor speaker
  :actee (c1 / c1)
  :tense present-continuous
)
```

The type `look-for` is subsumed by the UM-process `dispositive-material-action`, the participants of this process are `:actor` and `:actee`; the actor in this case is of type `speaker` and will be realized as a first person singular (I), and the actee will be the c_1. The verb describing the process `look-for` will be rendered as a present continuous.

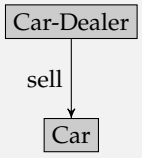
With this initial SPL chunk, we have all needed sentence plans to be passed to the linguistic realizer. They need to be finalized though.

First of all we need to check if any relation (abstract role) has to be realized as a passive voice verb. In order to test this, we need a way to directly recognize from the role name if the domain concept (subject) will act as source (actor) or recipient (actee) of the action represented by the role. We will call the former kind of role *active role* and the latter as *passive role*. We adopted a simple naming convention that helps us recognize passive roles, which is a *minus sign* (−) put as suffix of the corresponding active role. E.g. if the active role is `sell`, the passive role will be `sell-`. Therefore, whenever a query contains a passive role, its minus sign is deleted, and the active role is used in the sentence plan with an additional SPL line (`:pp-theme`) specifying that the grammatical subject is the recipient of the action denoted by the verb.

passive voice

We show below two sentence plans involving an active role and the corresponding passive role.

Query



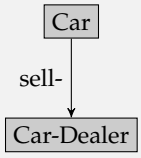
Sentence plan

```
(s1 / sell
    :actor (c1 / car-dealer
             :determiner the)
    :actee (c2 / car))
```

Generated text

The car dealer sells a car.

Query



Sentence plan

```
(s1 / sell
    :pp-theme c2
    :actor (c1 / car-dealer)
    :actee (c2 / car
             :determiner the))
```

Generated text

The car is sold by a car dealer.

A further adjustment on the generated sentence plans is the correct assignment of values to the parameters `:determiner` and `:pronoun` according to the output of Algorithm 9 together with Table 5.5. For each concept, the output of the algorithm can be one of R-INP, R-DNP, R-DP, R-RPS, R-PP, R-RPP. For each one we detail now what the effects on the sentence plan (hereinafter SP) are.

referring expressions

R-INP in this case nothing needs to be done because if no `:determiner` attribute is specified in the SP, the linguistic realizer automatically assigns an indefinite article (*a*, *an*) to the entity;

R-DNP this triggers the assignment of the article *the* to the `:determiner` attribute in the SP;

R-DP here we must consider first if the concept in question is *human* or *non-human*. This information must be available in the ontology, e.g. under the form of concepts as `Non-Human-Entity` and `Human-Entity` directly or indirectly subsuming the given concept. If human, we need to check if it's either `Male` or `Female` using the pronoun *he* or *she* respectively, or *they* if undefined. For non-human entities we use *it*. In the SP we assign one of {`it`, `he`, `she`, `they`} to the attribute `:pronoun`;

R-RPS nothing needs to be done here, because the SP structure will already lead the linguistic realizer to render the concept as a relative pronoun (see e.g. the process that involves actor c_n in Template 3;

R-PP the same considerations made for **R-DP** are valid here; the pronoun values of the set {`it`, `he`, `she`, `they`} are used and associated via the `:pronoun` attribute to the owner of a given concrete role; the linguistic realizer translates then the definite pronoun into the right possessive pronoun;

R-RPP for the generation of relative pronouns as possessives, nothing needs to be added to the SP.

The following last consideration shortly discusses the inclusion of a modal verb in the clauses, needed to emphasize the requirements of the user in terms of relations among concepts and their attributes in a query. The modal we already introduced before is *must* as e.g. in the sentence *The car **must** be an off-roader and a non-smoker car.* Naively putting it in each sentence would be quite annoying for the same user rereading the query, but on the other side this would be the right way of referring to an object we are looking for and precisely describing.

modal verb

An option would be the one of rendering the query without modal auxiliaries (in our case only *must*), and the user describes the object how it *is* instead of how it *must be*. The introductory sentence could be extxsended e.g. as: *I'm looking for a THING that is described as follows*, where THING stands for the starting concept chosen by the user. The SP generating this sentence is:

```
(s0 / look-for
  :tense present-continuous
  :actor speaker
  :actee
    (c1 / THING
      :process
        (s1 / be-described-as
          :tense present
          :actor c1
          :actee
            (c2 / template
              :pattern "follows"))))
```

5.4 Linguistic Realization

Linguistic realization is the last operation of the NLG-pipeline we have described so far. The task of a linguistic realizer is to convert sentence-sized chunks of a suitable input representation (sentence plan) into grammatically correct sentences.

5.4.1 Approaches to LR

Four main approaches to text realization are available [Hovy, 1997], differing in terms of sophistication/expressive power and flexibility. They are listed below, ordered from the simplest (and less flexible) to the most sophisticated ones:

- canned text systems,
- template systems,
- phrase-based systems,
- feature-based systems.

We will describe in turn each one of these categories.

Canned Text Systems

Whenever we want to generate a piece of text for a very specific purpose, without the need of modifying it according to some parameters, canned text is the easiest solution. It has been used by almost every application to convey a message (warning, error, help, etc.) to the user, a message which is simply associated to a given code produced by an application event. No syntactic or morphological process is involved, except, in some cases, capitalizing the first word in the sentence, and putting a full stop at the end.

Template-based Systems

Slightly more sophisticated, these systems provide template texts containing a certain number of placeholders, which at runtime will be substituted with

strings depending on the context (a title, a name, an address, some numbers, etc.). One typical example is represented by mail-merge applications, where the same letter with a few variations (receiver, salutation, closing, ...) needs to be created in multiple copies for different receivers.

Phrase-based Systems

In these systems, templates are more general, and resemble phrase structure grammar rules. They represent the various typologies of phrases we have in natural language (noun phrases, verb phrases, etc.) along with a set of rules specifying how phrases can be combined together to form grammatical sentences. E.g. we could have a pattern like `[subject verb object]` where each one of its components can be further decomposed into one of other possible phrasal patterns as `[subject]` → `[determiner adjectives head-noun modifiers]`. The generation process starts with a top-level sentence pattern matching the sentence plan, and stops when all pattern constituents have been replaced by one or more words.

The phrase-based approach is quite flexible in comparison to the ones seen before, and it is rather simple to implement such a system for a grammar of limited size; beyond a certain limit though, it is hard to keep track of all phrasal interrelationships in order to avoid wrong phrase expansions.

Feature-based Systems

These represent the highest level of sophistication and flexibility available for the generation of sentences. Here every possible alternative for expressing a sentence or part of it can be chosen by means of features: we can say if a sentence is *positive* or *negative*, if it is *declarative*, *imperative*, or a *question*, which are the tenses used in its clauses, etc. Generation in this case is accomplished by incrementally collecting features for each part of the input sentence plan until the sentence is complete: this can be done either by traversing a *feature selection network* (see Sec. 5.5.1) or via *unification* (see Sec. 5.4.2).

The strength of a feature-based approach is that any distinction in language can be encoded as a feature in the system. On the other side, cons of this approach are that also in these systems —as in the previous ones— maintenance of feature interrelationships tends to be quite hard; moreover (but this is no more an issue nowadays) some authors [McRoy *et al.*, 2001] report that quite often the entire grammar needs to be traversed, such systems tend to be too slow for real-time applications.

5.4.2 Overview of Three Feature-based realizers

We present in this section three widely used feature-based linguistic realizers, and we will show the different input representations they require. KPML is the system we employed and we just provide a short description of it in the next section, leaving all details for Section 5.5.

KPML

KPML (KOMET-Pennman-multilingual) is a grammar development environment from the University of Bremen [Bateman, 1997a]. KPML is a complex application, well known for extensive multilingual systemic-functional grammar (SFG) development and maintenance as well as for NL generation. For the sake of preciseness, as described in [Bateman, 1997b], the intended purposes of KPML are:

- to offer generation projects large-scale, general linguistic resources (at the time of writing available resources include English, Chinese, Czech, Greek, Japanese, Russian, German, and Spanish in varying stages of development);
- to offer generation projects an engine for using such resources for generation;
- to encourage the development of similarly structured resources for languages where they do not already exist;
- to provide optimal user-support for undertaking such development and refining general resources to specific needs;
- to minimize the overhead of providing text im multiple languages;
- to encourage contrastive functional linguistic work;
- to raise awareness and acceptance of text generation as a useful endeavor.

KPML can be used as fully featured grammar development environment, but it is also available as a simple blackbox linguistic realizer. The environment offered by the system takes over and extends the functionality of its predecessor, the Penman text generation system [Mann, 1983a; 1983b] outperforming it in terms of ease of use, development support, and multilingual design.

The input required by KPML is an annotated semantic specification (sentence plan) expressed using the *Sentence Plan Language* (SPL). Our sentence planner, as shown above, adopts this language which will be formally described in Sec. 5.5.4.

SURGE (FUF)

FUF/SURGE [Elhadad, 1992; 1993] is a text realization system which implements an extension of a functional unification grammar[3] (FUG). FUF is a declarative formalism for which exists an interpreter written in CommonLisp for a functional

[3] FUG [Kay, 1979] is a formalism for describing grammars of natural languages. It shares some similarities with Tree Adjoining Grammar, and allows the expression of the idiosyncratic syntactic constraints of a given language. FUG does not subscribe to any specific grammar model, and is not a theory of grammar itself. As clearly described by [Kasper, 1988], *it is a framework that represents a grammar in a form that is simultaneously readable by linguists and suitable for use by computer programs that generate or analyze text. Unlike many previous grammatical formalisms, which required a different representation for analysis than for generation, FUG is intended to be neutral in this respect.' In earlier work, FUG was also called Functional Grammar or Unification Grammar. The notation of FUG looks very different from that of Systemic Functional Gram-mar (SFG); but if we look beyond the differences of format, we find that FUG and SFG make many of the same assumptions about language and grammar. The similarities are due, in part, to the fact that when Martin Kay [Kay, 1979] formulated FUG he was responding to many of Michael Halliday's ideas.*

unification based language specifically designed to develop text generation applications. SURGE [Elhadad and Robin, 1996] is a comprehensive generation grammar of English written in FUF.

The input of FUF/SURGE is a functional description (FD) which is a type of feature structure. FDs include thematic structures, syntactic categories, and content words. Function words, as articles, pronouns, an conjunctions can be automatically generated by the system. Moreover SURGE provides defaults (as most other systems do) in a way that a shorter partial specification is possible. Figure 5.10 shows an example of FD which generates the sentence "Ray sends a nice letter to Sandra."

```
((cat clause)
 (process ((type composite)
           (relation-type possessive)
           (lex "send")))
 (participants ((agent ((cat proper)
                        (lex "Ray")))
                (affected ((cat proper)
                           (lex "Sandra")))
                (possessor (^ affected))
                (possessed ((cat common)
                            (lex "letter")
                            (definite no)
                            (describer ((lex "nice"))))) ))
```

Figure 5.10: Example of FUF/SURGE input for the sentence "Ray sends a nice letter to Sandra."

RealPro (MTT)

Finally we introduce RealPro [Lavoie and Rambow, 1997], described in [RealPro, 2008] as a text generation `engine` that performs syntactic realization — i.e., the transformation of abstract syntactic specifications of natural language sentences (or phrases) into their corresponding surface forms. It supports multiple languages and levels of linguistic representation, with performance suitable for real-world applications. In applications such as machine translation, text generators need to be able to handle a wide variety of different inputs, and produce fluent text for each one. Whether the input is at the semantic, conceptual, or phrasal level, there are often simply too many syntactic rules that need to be taken into account for a simple phrase concatenation-based approach to be feasible. This is in contrast to applications such as data summarization, where the system designer has more control over the syntactic variety that will be required in generated text, and a template-based approach is often practical. RealPro provides a grammar rule engine that can generate text from sophisticated, multi-level linguistic representations. The abstraction it provides makes it easy to generate many syntactic variants of the same semantic content on demand — unlike with template-based approaches, where the combinatorics of generating multiple syntactic variants quickly becomes unmanageable.

RealPro's interesting aspect is that its syntactic specifications are based on the deep-syntactic structures of the Meaning-Text linguistic theory (MTT) [Žolkovskij and Mel'čuk, 1967; Mel'čuk and Žolkovskij, 1970], a theoretical framework for the construction of models of natural languages, called Meaning-Text Models. Meaning-Text theory emphasizes on semantics and considers natural language primarily as a tool for expressing meaning. MTT is based on the following three postulates:

Postulate 1. *Natural language is (considered as) a many-to-many correspondence between an infinite denumerable set of meanings and an infinite denumerable set of texts.*

Postulate 2. *The Meaning-Text correspondence is described by a formal device which simulates the linguistic activity of the native speaker—a Meaning-Text Model.*

Postulate 3. *Given the complexity of the Meaning-Text correspondence, intermediate levels of (utterance) representation have to be distinguished: more specifically, a Syntactic and a Morphological level.*

Methodological Principle *The Meaning-Text correspondence should be described in the direction of synthesis, i.e., from Meaning to Text (rather than in that of analysis, i.e., from Text to Meaning).*

This is why MTT is well suited for linguistic synthesis (rather than analysis), and paraphrasing (synonymy production of linguistic expressions, and full sentences). MTT considers relations (rather than classes) as the main organizing factor in language and makes an extensive use of the concept of linguistic dependency, in particular of syntactic dependency (vs. constituency).

As far as input to RealPro is concerned, Figure 5.11 shows a sample input for the sentence "The lady gave a letter to the postman." With an input of this form, RealPro adds then the missing function words (as e.g. auxiliaries, prepositions, and articles) and all the necessary morphological features. In order to use RealPro, applications need not specify all syntactic objects that appear in the output but just the sufficient knowledge of the target language to render the necessary syntactic relations.

```
give [ class:verb tense:past ]
(
   I   lady    [ class:common_noun article:def gender:fem ]
   II  letter  [ class:common_noun article:undef ]
   III postman [ class:common_noun article:def ]
)
```

Figure 5.11: Example of RealPro input to generate the sentence "The lady gave a letter to the postman."

5.5 Linguistic Realization with Systemic Functional Grammar

In the previous section we introduced three feature-based realizers along with the different linguistic theories or formalisms employed. Hereinafter we concentrate on one of them, a famous and fascinating linguistic theory, Systemic Functional Linguistics (SFL), which was leveraged for the purpose of natural language generation giving rise to what is called *computational SFL*. We present Systemic Functional Grammar (SFG) and a computational implementation called the Nigel systemic grammar of English. We will see what the *Upper Model*, a linguistic ontology is used for, and what is the input specification to the chosen realizer we chose (KPML). Finally we will see how KPML really works and how we are using it for our purpose.

5.5.1 Systemic Functional Grammar

History

Systemic Functional Grammar (SFG) is a grammar model and major influential linguistic theory developed by Michael Alexander Kirkwood (M. A. K.) Halliday and grown out of the work of John Rupert Firth, a British linguist who influenced a whole generation of linguists for more than twenty years in the University of London. Firth was an important figure in the foundation of linguistics as an autonomous discipline in Britain, and the popularity of his ideas among contemporaries gave rise to what was known as the 'London School' of linguistics. Among Firth's students, the so-called *neo-Firthians* were exemplified by Michael Halliday, Professor of General Linguistics in the University of London from 1965 until 1970 when he moved to Australia, establishing the department of linguistics at the University of Sydney. Through his teaching there, SFL has spread to a number of institutions throughout Australia, and around the world.

Theory

Systemic-functional grammar is concerned primarily with the *choices* that are made available to speakers of a language by their grammatical systems. These choices are assumed to be meaningful and relate speakers' intentions to the concrete forms of a language.

Language is considered a social resource by means of which speakers and hearers act meaningfully. Meanings are in systemic functional grammar divided into three broad areas, called *metafunctions*: the *ideational*, the *interpersonal* and the *textual*, as extensively described in the SFL literature, in particular [Halliday and Matthiessen, 2004].

- The *ideational* is grammar for representing the world. That is, the propositional content, which is concerned with ideation providing the speaker with the resources for interpreting and representing reality. It is divided into two subtypes, the *experiential* and the *logical* metafunctions: The former is reflected in terms of configurations of processes and participants. We could name e.g. the TRANSITIVITY structure of the clause, which

describes what in other theories are known as semantic relations. The experiential part of the ideational metafunction also includes systems for circumstantials, types of prepositional phrase, tense, noun-types, etc. The logical part, instead, is the mode for creating various kinds of complexes that are hypotactically or paratactically related.

- The *interpersonal* is grammar for enacting social relationships such as asking, requests, asserting control, or ordering. Thus the interpersonally is very much about interaction between human beings, society and culture.
- Finally, the *textual* is grammar for binding linguistic elements together into broader texts (via pronominalizations, grammatical topicalization, thematization, expressing the newsworthiness of information, etc.), or more simply, the rhetorical structure of a text. What is a subordinate clause? What is an independent clause? These are the kinds of questions that deal with the textual element of meaning.

Systemic functional grammar deals with all of these areas of meaning equally and within the grammatical system itself.

From a higher level perspective, as clearly explained in [Teich, 1999, pages 8–9], SFL view of language rests upon four main considerations:

- language is behaviour potential, realized by systems that support the theoretical notion of choice;
- language construes meaning which is realized by stratification (phonology, grammar, semantics), represented in SFL as paradigmatically organized resources;
- language is multifunctional, where functional diversification is represented by the metafunctions descripted above;
- using language is choice in the potential and ultimately actualization of the potential, by means of realization statements.

In the following subsections we will see in more detail what a system and a system network are, along with an explanation on how to specify linguistic structure by means of realization.

The System Network A *system network* is a directed graph whose nodes are choice points called *systems*. Each system consists of *entry conditions* and *output features*. A small section of systemic network for the English grammar is shown in Figure 5.12, where system names are capitalized.

The "MOOD TYPE" system e.g. has an entry condition which is "clause", and two alternative output features which are "indicative" and "imperative". Entry conditions can be conjunctions or disjunctions of output features of other systems. The "TAGGING" system for example has a disjunctive entry condition, which can be either the "declarative" or "imperative" feature. There can be *symultaneous systems* that share entry conditions, such as "PROCESS TYPE" and "MOOD TYPE"; this means that both are relevant in the paradigmatic context described by the entry condition "clause" and both must be entered as soon as the system "RANK" outputs "clause".

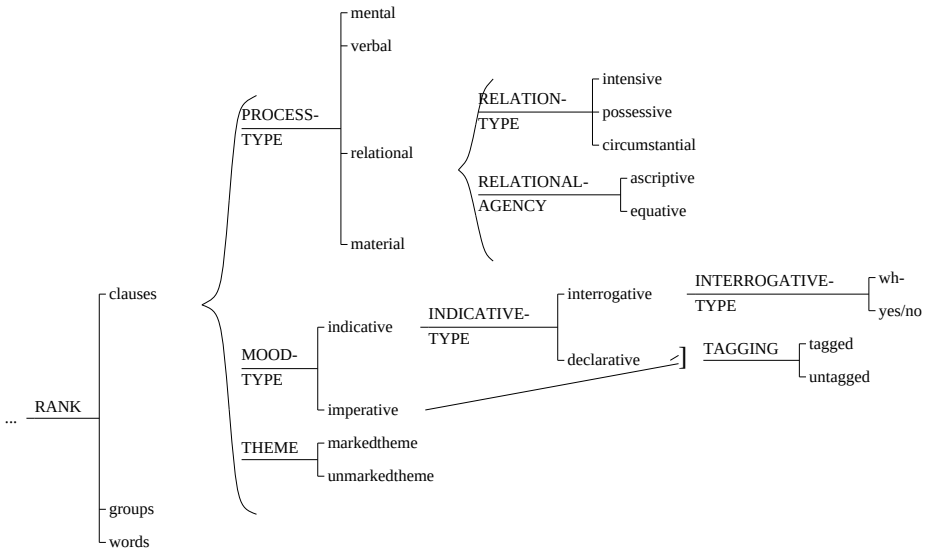


Figure 5.12: Example of system network fragment

Connections among systems define a partial ordering that spans, if we consider the graphical representation, from least *delicate* (most general) systems on the left to most delicate (most specific) systems on the right. We have an *incremental description refinement* as discussed in [Mellish, 1988], a scale of delicacy representing a left-to-right dimension. An interesting aspect is that paradigmatic choices in systems take place not only between grammatical alternatives, but also between lexical alternatives. In fact Halliday introduces the term *lexico-grammar* to include both of them, meaning that there is no clear division between grammar and lexicon, and if on the left part of the network we have grammatical choices, towards the right side of it lexical choices take place. This is summarized in Halliday's expression of *lexis as the most delicate grammar*.

We still need to see how systems are related to the functional side of SFG, in particular with metafunctions. The relation is that each system pertains to one and only one metafunction. Moreover, systems of the same metafunction are strictly connected, in a measure that they are largely independent from systems of other metafunctions. If we refer to Figure 5.12, "PROCESS-TYPE" and the systems depending on it are in the "TRANSITIVITY" region of the ideational metafunction; "MOOD TYPE" and its successors are in the "MOOD" region[4] of the interpersonal metafunction, while "THEME" and other systems connected to it are in the "MOOD" region of the textual metafunction. Table 5.6 shows the main systems in SFG according to some high-level entry conditions and the three metafunction.

[4] *Regions* are groups of systems within the same metafunction, possessing strong intra-region dependencies, and weak inter-region dependencies, creating a modularity that is beneficial for grammar design, maintenance and development.

	ideational	**interpersonal**	**textual**
clause	TRANSITIVITY	MOOD	THEME
verbal group	TENSE	MODALITY	VOICE
nominal group	MODIFICATION	PERSON	DETERMINATION

Table 5.6: Main systems in SFG

Specifications of linguistic structure The way syntactic structure is created in SFG is by means of *realization statements* which are associated with the output features of systems, and show how the paradigmatic choices in the systems are expressed as syntagmatic chains in the language structures. In Figure 5.13

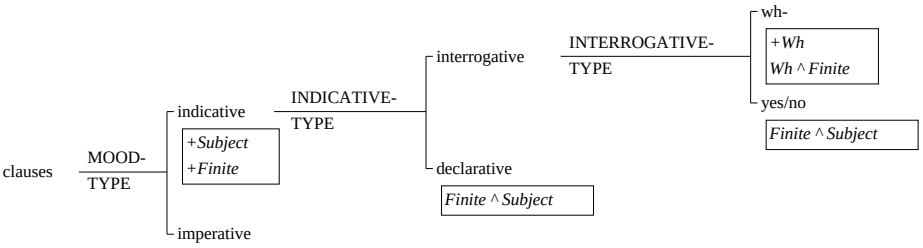


Figure 5.13: Example of system network fragment with realization statements

The "indicative" feature e.g.. embeds two realization statements, "+Subject" and "+Finite" which are *insertion* realizations. The "yes/no" feature instead has just one, "Subject ∧ Finite", which is an *ordering* realization. Table 5.7 summarizes the realization statements of SFG.

Name	Notation and example	Description
insert	+Subject	this statement requires the presence of this function as constituent
order	Subject ∧ Finite	this requires that the two functions must be ordered one after the other
conflate	Subject / Agent	requires that the two functions are realized by the same element of structure
expand	Mood (Finite)	the first function is expanded to have the one in brackets as constituent
preselect	Subject : singular	this constrains the realization of the function to display the given feature

Table 5.7: Realization statements used in SFG

We terminate with a simplified example (Figure 5.14) of the kind of information that is specified in an SFG syntagmatic unit.

In your car	*Paul*	*you*	*will*	*find*	*a navigation system*	
Theme		Rheme				**textual**
		Mood		Residue		**interpersonal**
	Vocative	Subject	Finite			
Locative		Actor	Process		Complement	**ideational**

Figure 5.14: Simplified example of metafunctional layering

5.5.2 The Nigel systemic grammar of English

Nigel represents the biggest computational systemic functional grammar for English available to-date. Nigel has been under development since the early 1980s [Matthiessen, 1981; 1983; Mann and Matthiessen, 1983], when it was used within the Penman project for English generation. It was mainly developed by Christian Matthiessen on the foundation of work by Michael Halliday. Since then many people have contributed to various parts of its coverage.

The latest version consists of around 765 systems[5], where the first one to be entered is the "RANK" system reported below, whose output features are the items of the rank scale: clauses, group-phrases, words or morphemes.

```
(SYSTEM
    :NAME   RANK
    :INPUTS   START
    :OUTPUTS
      ((0.2 CLAUSES)
       (0.2 GROUPS-PHRASES)
       (0.2 WORDS
         (INSERT STEM)
         (PRESELECT STEM MORPHEMES))
       (0.2 MORPHEMES
         (INSERT HEAD)))
    :CHOOSER   RANK-CHOOSER
    :REGION   RANKING
    :METAFUNCTION   LOGICAL
)
```

At word and morpheme level, the Nigel grammar does not provide a unified lexicogrammar of SFG as in the theory; lexis and morphology are treated apart in an external lexicon. At the clause level the grammar can generate clause complexes of two clauses in paratactic or hypotactic relation. In order to generate, the system network is traversed starting from the "RANK" system; the rule is that whenever an output feature is chosen, the next step is to collect all systems having the same entry conditions as the preceding output feature, and to enter each one of them on turn in random order. Every time an output feature is chosen, the realization statements attached to it are immediately executed, except the ordering realizations which are collected and executed later.

[5]The total count of systems includes 324 *gates* which are symplified system having only one output feature.

The choice among output features is done by means of choosers and inquiries, an explicit formalization developed by William C. Mann under the name of *inquiry semantics* or *chooser/inquiry interface* [Mann, 1983a]. Each system with more than one output feature specifies a *chooser*, a small "choice expert" that knows how to make appropriate choices among the grammatical features available. This is done by traversing a decision tree from the root to one of the leaf nodes which represents the chosen feature. *Inquiries* are oracles which can be relied on to motivate grammatical alternations for the current communicative goals being pursued. Figure 5.15 shows the same network of figure 5.13 augmented with choosers.

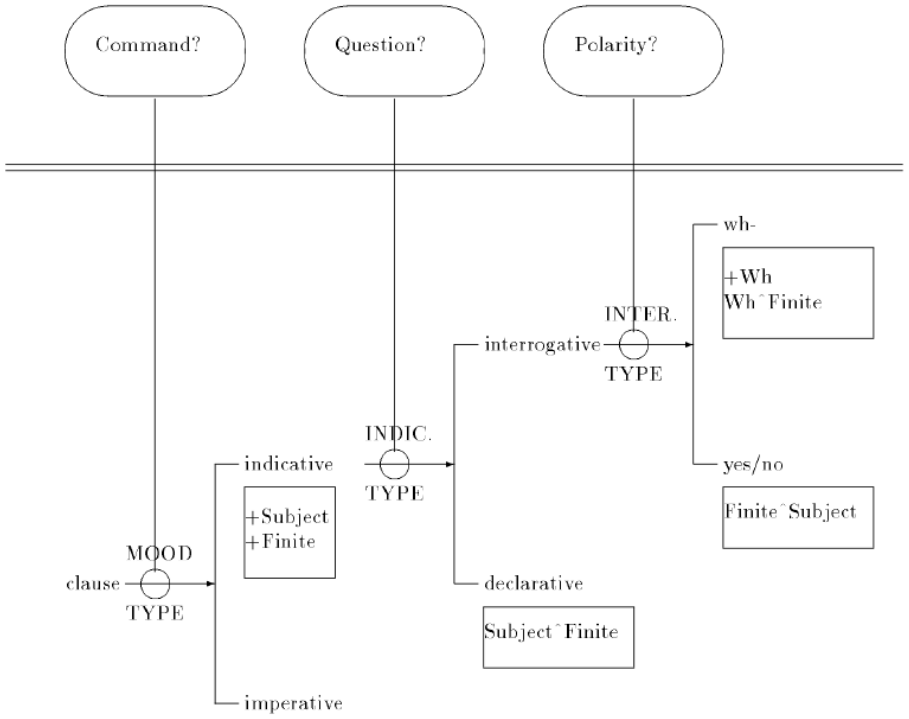


Figure 5.15: Example of system network with choosers

5.5.3 The Upper Model

The Upper Model, also known as *linguistic ontology* was born within the Penman project [Matthiessen, 1987] as a fundamental resource for organizing domain knowledge appropriately for linguistic realization. It is a domain- and task-independent ontology meant to support and simplify the interface between domain knowledge and linguistic resources [Bateman, 1990]. The importance of this interface is clear if we think that most ideational inquiries ask questions regarding the classification of an input category in terms of abstract semantic categories. The Upper Model is based on the *Bloomington Lattice* [Matthiessen, 2005, page 168], an *ideational* grammatical semantic typology for English started by Michael Halliday and Christian Matthiessen during the summer of 1986. It

reflects the English lexicogrammatical semantics, the *ideational* metafunction only, and it is called the *ideation base*. Figure 5.16 shows an excerpt of the higher level classes of the Penman Upper model and their taxonomical relations.

Figure 5.16: Excerpt of the Penman Upper Model taxonomy

The Penman Upper Model was augmented to account for the grammar of German in the 1990's and it became the *Merged Upper Model* [Henschel, 1993; Henschel and Bateman, 1994] for use in the KOMET-Penman Multilingual Development Environment (KPML) system (see Section 5.5.5 below).

In order to provide more linguistic coverage, both in terms of the generation ability in a given language, but also in various other languages, and to bring the Merged Upper Model more in line with the systemic work of Halliday and Matthiessen [Halliday and Matthiessen, 1999], the *Generalized Upper Model* (GUM) [Bateman *et al.*, 1995] was created. At the time of writing, the latest version of GUM is 3.0[6], and Figure 5.17 shows its higher level classes along with their taxonomical relations. In terms of representation format, the GUM, originally written in LOOM [MacGregor and Bates, 1987], has been made available as OWL-DL[7] file.

The contents of the Penman Upper Model were used in conjunction with the Sentence Plan Language (SPL) [Kasper, 1989] (presented below) as input to the Penman generation system. The KPML system, instead, employs the Merged Upper Model (LOOM format), hereinafter referred to as the Upper Model..

5.5.4 Input specification: the Sentence Plan Language

An SPL representation is a list of terms which describe the entities that need to be expressed in NL along with the particular attributes of those entities.

[6] `http://www.fb10.uni-bremen.de/anglistik/langpro/kpml/README.html`
[7] `http://www.w3.org/TR/owl-guide`

Figure 5.17: Excerpt of the Generalized Upper Model taxonomy (v3.0)

Attributes may specify semantic relations that are to be expressed from the domain model or they may specify responses to inquiries about grammatical features of sentences. The syntax of SPL, specified in [Kasper, 1989], is reported here:

$$
\begin{aligned}
\textit{Plan} &\rightarrow \textit{Term}^{+} \\
\textit{Term} &\rightarrow (\textit{Variable} \ / \ \textit{Type Attribute}^{*}) \mid \textit{Variable} \mid \textit{Constant} \mid \\
&\quad (\textit{Term}^{+}) \mid (:\textit{and Term}^{+}) \mid (:\textit{or Term}^{+}) \\
\textit{Type} &\rightarrow \textit{ConceptName} \mid (\textit{ConceptName}^{+}) \\
\textit{Attribute} &\rightarrow \textit{Keyword} \ \ \textit{Term} \\
\textit{Keyword} &\rightarrow \textit{RelationName} \mid \textit{MacroName} \mid \textit{InquiryName} \ (\textit{Variable}^{+}) \mid \\
&\quad \textit{SpecialKeyword}
\end{aligned}
$$

Example 5. *A sentence plan for "*The car is equipped with a service booklet.*" could be:*

```
(e1 / be-equipped-with
  :actor (e2 / car
     :determiner the)
  :actee (e3 / service-booklet))
```

`e1` represents the main term of this plan, and it denotes an entity of the domain model. The type of `e1` is `be-equipped-with` which is defined as specialization of `generalized-possession`, a reified relation from the upper model. It has two main attributes named `:actor` and `:actee`. The actor is denoted by the keyboard `e2` (referring to the concept `car`) and the actee by `e3` (referring to the concept `service-booklet`).

The syntax of SPL permits the use of macros as keywords also. With a macro we can express in a succint manner a set of delicate features to generate some

specific grammatical phenomenon. E.g. if we want to express English tense in general terms, we should provide precise inquiry responses, setting three times and the ordering relations among them:

- the actual speaking time
- the event time, and
- the time of reference with which the event is contrasted.

The `:tense` macro was created to simply avoid specifying these temporal relations, simply distinguishing the English tenses using values that are expanded into the appropriate inquiry responses.

A system as KPML contains a package of macro keywords that greatly help in simplifying the creation of a sentence plan.

Example 6. *To modify the previous sentence plan in order to generate the sentence "*The car was equipped with a service booklet*", we can use the* `:tense` *macro as follows:*

```
(e1 / be-equipped-with
  :actor (e2 / car
     :determiner the)
  :actee (e3 / service-booklet)
  :tense past)
```

Another facility provided by KPML is a way of defining *default values* for inquiries in order to predefine sentence features that do not change frequently in a given application domain. The SP given above doesn't specify if the sentence to be generated must be a statement, a question, or a command, nor does it say if it should have positive or negative polarity. This is because the KPML system generates by default statements with positive polarity.

The interpretation of a SP is done by KPML in two phases:

1. the plan is first transformed into an internal representation, where all macros are expanded, and type information is distributed to variable terms whose consistency is also checked. The first term of the plan is treated as the initial unit to be expressed (main clause of the sentence).
2. given this representation, the generation process is guided by means of a series of inquiries to the sentence plan and the available knowledge sources according to this order:
   (a) SPL keyword: The SP is searched first for a keyword matching the name of the inquiry, and the value is returned;
   (b) knowledge sources: inquiries may have a function associated with them called *inquiry implementation*, which searches the domain and upper model for the type or the attributes of the SPL terms;
   (c) active default value: If an undefined answer is supplied by the inquiry implementation, or if there is no inquiry implementation, then the current active default value is used.

5.5.5 The KPML System

In this section, proceeding from the introduction of Section 5.4.2, we present the architecture of the KPML system along with on overview of the generation process (based on [Bateman, 1997b]) and the resources that the system uses for this purpose.

The KPML Generation Process

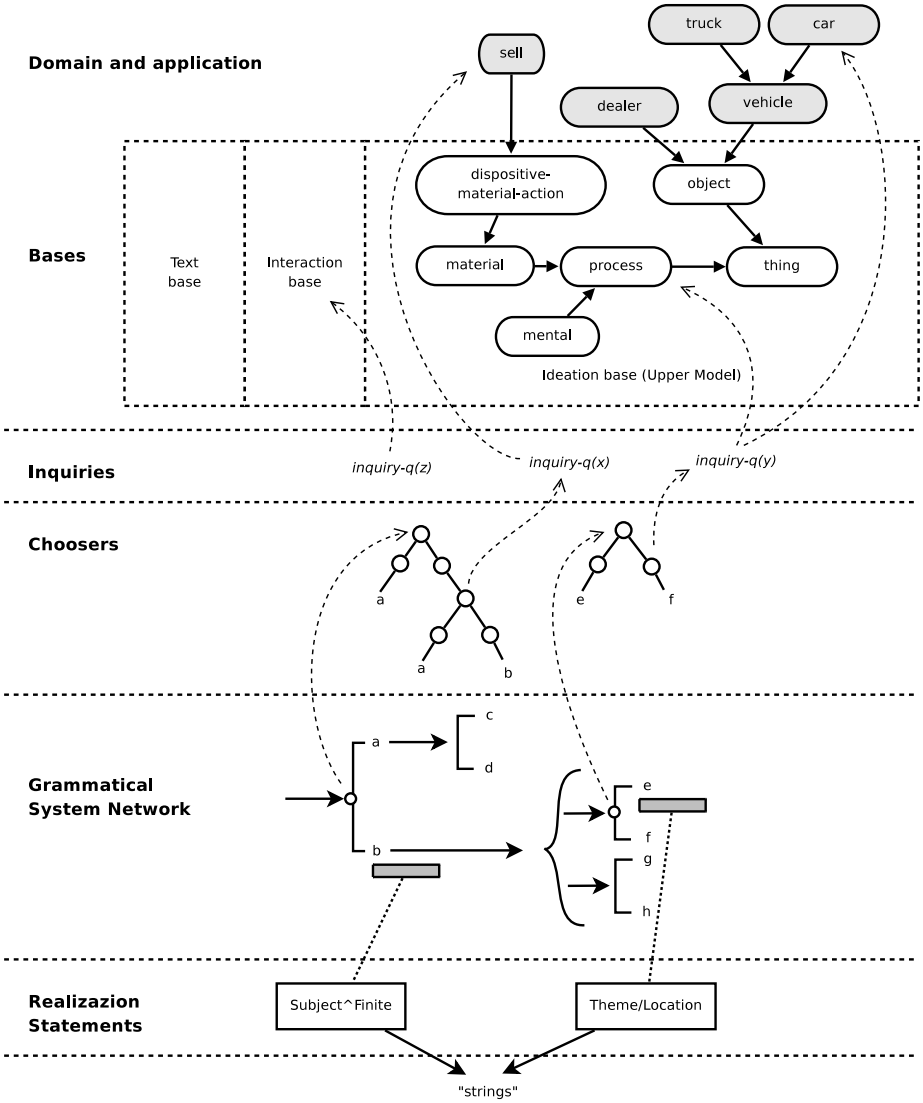


Figure 5.18: KPML Pennman-style generation architecture (based on [Bateman, 1997b])

KPML uses a Pennman-style generation architecture that is depicted in Figure 5.18. Generation in KPML proceeds in cycles of traversal through the system

network. The outcome of this traversal is a set of grammatical features called *selection expressions* and a resulting grammatical structure. It is by resolving grammatical constituents associated with features of the selection expression that the grammatical structure is created. Features chosen during network traversal are selected according to the semantic input that needs to be expressed, operation that is mediated by the chooser and inquiry framework (see Section 5.5.2): Choosers organize inquiries into "decision trees", and inquiries are responsible for (a) inspecting the semantic specification that is being expressed in order to classify it and (b) providing access to particular portions of the semantic specification in order to trigger further realization. The connection between the systemic grammar and the semantic input is made via a *function association table* that relates grammatical functions (labels for grammatical constituents) and semantic "hubs" (labels for the semantic input chunks that need to be expressed). The input arguments for inquiries are grammatical functions.

Cycles of generation will continue for all sub-constituents of a grammatical unit until all sub-constituents are filled by some linguistic substance, usually *lexemes* or *morphemes*. One thing that has to be avoided is underconstraining grammatical constituents, which would cause infinite regression. There are four ways in KPML to correctly specify a grammatical constituent so that it receives lexical material and doesn't trigger another cycle through the grammar:

1. an explicit lexical entry can be selected with the realization statement `lexify`;

2. a set of lexical features can be associated with a grammatical constituent using the `classify` realization statement; on completion of a traversal through the grammar, the complete collection of lexical features of lexical features for a grammatical constituent is used to pick a matching lexical item (i.e. a lexical item whose features unify);

3. the inquiry `term-resolve-id` can be invoked to ask for an explicit lexicalization on semantic grounds;

4. an explicit selection of a morpheme can be made with the morphological realization operators, which are: `preselect-substance`, `preselect-substance-as-stem`, or `preselect-substance-as-property`.

It must be noted that if a constituent has been classified, the selection of a lexical item as shown in (2) will not respect any additional information since it follows a purely lexicogrammar internal selection. This means that no semantic information or SPL input will be consulted. If we need to take into account semantic information also, option (3) must be chosen by including the `term-resolve-id` inquiry in some chooser that is activated at an appropriate point during generation.

The semantic organization adopted by KPML foresees first of all a linguistic ontology called the *Upper Model* that was presented in Section 5.5.3. All of the KPML resources are defined in a way that generation is possible with respect to a single Upper Model, as concrete instantiation of the *ideation base*. The domain model representing the universe of discourse we want to generate natural language about, must be connected with the upper model. This way we can directly use entities from the domain model to formulate SPL inputs for the

generator. Two other "bases" are needed (as shown in Figure 5.18): Interaction and text base. The *interaction base* represents the knowledge that the system has about the social and epistemic relationship between speaker (machine) and the hearer; this can be instantiated as a *user model*. The *text base* instead, is concerned with the system's knowledge about which discourse structures, coherence relations, and cohesive ties need to be used, which grammatically are interpreted as theme-rheme structure, conjunctions, referring-expressions, etc.

KPML Input Resources

In order to be able to generate, the KPML system needs the following linguistic resources:

- a domain model,
- a grammar,
- and a lexicon.

We introduce them briefly hereinafter, suggesting the reader to refer to [Bateman, 1997b, Section 12.2] for an in-depth description of resource organization and definition formats.

Domain Model Given a domain model on which we want to generate, its concepts and properties (relations and attributes) must be subordinated to the Upper Model entities by means of LOOM axioms. This means we have to rewrite the original domain ontology in the input format required by KPML (LOOM), subordinating it to the Upper Model. For generation purposes, not all axioms of the original ontology need to be translated, but just concept and role (abstract and concrete) definitions, and subsumption relations. The mapping is pretty simple, since all source entities (both concepts and roles) will be translated into LOOM concepts, and either subordinated to an UM `Object` or a `Process` or one of their descendants (see Table 5.8).

Domain Ontology Entity	. . . mapped into a	. . . subordinated by an UM
concept	concept	Object
relation_v	concept	Process
relation_a	concept	Object
attribute	concept	Object

Table 5.8: Mapping of domain ontology entities and subordination to UM entitities

Since attribute descriptions will be rendered using a copula (e.g. "the engine's power *is* 250 HP", "the car's weight *is* 1500 kg"), an UM Property Ascription (Process) needs to be used in the sentence plan to describe this process. Furthermore the reified attribute (see Table 5.8) will most probably become the subject of the clause, and the domain concept of the attribute will be used as the subject's modifier.

Relations instead, have been distinguished into two categories: relation$_v$ and relation$_a$. The former represents relations which will be expressed as verbs describing the respective processes as in "the car *runs* on gasoline". The latter refers to those relations that act as attributes (but have a concept as range instead of a concrete datatype) and are treated the same way as attributes are (e.g. "the car's *make* is VW", "the car's *model* is Golf GTD").

```
(defconcept Vehicle
  :is (:and Penman-kb::Decomposable-Object :primitive))
(kpml::annotate-concept Vehicle :lex-items (vehicle))

(defconcept Car
  :is (:and Vehicle :primitive))
(kpml::annotate-concept Car :lex-items (car))

(defconcept Car-Dealer
  :is (:and Penman-kb::Object :primitive))
(kpml::annotate-concept Car-Dealer :lex-items (car-dealer))

(defconcept Make
  :is (:and Penman-kb::Object :primitive))
(kpml::annotate-concept Car :lex-items (make))

(defconcept Model
  :is (:and Penman-kb::Object :primitive))
(kpml::annotate-concept Model :lex-items (model))

(defconcept Sell
  :is (:and Penman-kb::Dispositive-Material-Action :primitive))
(kpml::annotate-concept Sell :lex-items (sell))
```

Figure 5.19: Excerpt of LOOM ontology from the automotive domain

Some example concept definitions in LOOM format regarding the automotive domain are reported in Figure 5.19. The `:primitive` predicate means that the concept it refers to is incompletely specified, i.e. there are hidden attributes about objects of that type that are not represented and the concept is thus considered as a 'primitive'. *v*-type relations of the original domain ontology must be reified and represented as concepts subsumed by one of the "processes" of the Upper Model (see Figure 5.16). The remaining *a*-type relations are reified and classified under the `Object` sub-hierarchy (see concepts `Make` or `Model` above) and, as stated above, will be usually rendered as subject of the generated clause (e.g. "the car's *make* is Opel"). The domain concept of the relation will be used as subject modifier (as *car's* in the previous example).

The `kpml::annotate-concept` lines following concept definitions are necessary to create a link between the defined concept and the respective lexical items contained in the Lexicon.

Grammar Although in our work we currently generate in English using the latest Nigel Grammar for English (Section 5.5.2), it is good to know that there are other resources available for a range of languages (including resources for Chinese, Czech, Greek, Japanese, Russian, German, and Spanish in varying stages of development)[8]. The Nigel Grammar for English consists of 42 functional regions (see Table 5.9), each one giving its name to three files, one containing the systems, one the choosers and the third one the enquiries for that region. E.g. the `RANKING` region is covered by the three files which are `RANKING.systems`, `RANKING.choosers`, and `RANKING.inquiries`.

ADJECTIVAL-COMPARISON	EPITHET	PROCESSUALTHINGTYPE
ADJECTIVAL-GROUP	GATES	PRONOUN
ADVERBIAL	MOOD	QUALIFICATION
ATTITUDE	NOMINALGROUPCOMPLEXITY	QUANTIFICATION
CIRCUMSTANTIAL	NOMINAL-PERSON	QUANTITY-GROUP
CLASSIFICATION	NONRELATIONALTRANSIT.	RANKING
CLAUSECOMPLEX	NOUNTYPE	RELATIONALTRANSITIVITY
CONJUNCTION	ORDINALITY	SELECTION
COUNTNUMBER	PHRASAL-MOOD	TAG
CULMINATION	POLARITY	TENSE
DEPENDENCY	POST-DEICTICITY	THEME
DETERMINATION	PPCOMPLEXITY	UNIFYINGGATES
ELABORATION	PPOTHER	VOICE
ELLIPSIS	PPSPATIOTEMPORAL	WORD-FORMS

Table 5.9: Functional regions in the Nigel Grammar for English

The grammar files are written using LISP-like syntax, as shown in the following triplet of system, chooser, and inquiry taken from the `MOOD` region.

```
(SYSTEM
    :NAME   MOOD-TYPE
    :INPUTS   INDEPENDENT-CLAUSE-SIMPLEX
    :OUTPUTS
      ((0.5 INDICATIVE)
       (0.5 IMPERATIVE
         (INSERT NONFINITIVE)
         (INFLECTIFY NONFINITIVE STEM)))
    :CHOOSER   MOOD-TYPE-CHOOSER
    :REGION   MOOD
    :METAFUNCTION   INTERPERSONAL
)
```

[8] A collection of systemic-functional grammars for natural language generation can be found in the Generation Bank of the University of Bremen. The generation bank is a website that is being constructed to contain lexicogrammars for tactical generation in a variety of languages. All grammars have the same form and can be used by the same generator i.e. the KPML system. When complete, each grammar fragment will contain a complete grammar definition in KPML-standard format, including example sets ('target suites') that provide a summary of coverage and corresponding semantic inputs. It's available here: `http://www.fb10.uni-bremen.de/anglistik/langpro/kpml/genbank/generation-bank.html`.

```
(CHOOSER
    :NAME   MOOD-TYPE-CHOOSER
    :DEFINITION
      ((ASK (COMMAND-Q SPEECHACT)
              (COMMAND
                (IDENTIFY SUBJECT
                  (COMMAND-RESPONSIBLE-ID SPEECHACT))
                (CHOOSE IMPERATIVE))
              (NOCOMMAND
                (CHOOSE INDICATIVE)))
)
```

```
(ASKOPERATOR
    :NAME   COMMAND-Q
    :DOMAIN   TP
    :PARAMETERS   (ACT1)
    :ENGLISH   (
          "   Is the illocutionary point of the surface level"
          "speech act represented by"
          ACT1
          "    a command, i.e. a request of an action by the"
          "hearer?"
          )
    :OPERATORCODE   KPML::COMMAND-Q-CODE
    :PARAMETERASSOCIATIONTYPES   (CONCEPT)
    :ANSWERSET   (COMMAND NOCOMMAND)
)
```

Finally we see how lexical items are stored and described in KPML.

Lexicon Three lexical items taken from the automotive domain are presented in Figure 5.20.

```
(lexical-item
    :name   car
    :spelling   "car"
    :sample-sentence   "This car is a Land Rover."

    :features   (common-noun noun)
    :editor   "PAOLO DONGILLI"
    :date   "Mon Sep 19 15:26:39 CEST 2006"
)

(lexical-item
    :name   sell
    :spelling   "sell"
    :sample-sentence   "This car is sold by a car dealer."
    :features (disposal-verb do-verb effective-verb s-irr)
    :properties   ((pastform "sold")
                   (edparticipleform "sold"))
    :editor   "PAOLO DONGILLI"
    :date   "Wed Sep 14 11:11:39 CEST 2006"
)

(lexical-item
    :name   car-dealer
    :spelling   "car dealer"
    :sample-sentence   "The car dealer is located in Germany."
    :features   (common-noun noun)
    :editor   "PAOLO DONGILLI"
    :date   "Wed Sep 14 10:59:45 CEST 2006"
)
```

Figure 5.20: Lexical items from the automotive domain

The features that appear under the `features` slot depend on the concrete linguistic resources defined to the system. The `properties` slot, instead, is used for holding idiosyncratic exceptions to general morphological processes. A resource-external morphology handling is adopted in KPML, i.e. the resource definitions assume that the morphological features that they use are interpreted by some non-systemic component of KPML. One example of such a resource definition is the Nigel grammar of English, for which the Penman system provided hardcoded English morphology. This hardcoded morphology is inherited by KPML[9].

[9]The current version of the Nigel grammar released as a KPML-resource set, does however include systemic resources for morphology. This provides a more flexible and transparent representation of the linguistic resources at word and morpheme rank, but increases the generation time a little since further cycles through the grammar are required.

Chapter 6

Evaluation

Chapter 7

Discussion and future work

Chapter 8

Conclusions

Appendix

English semantics for generation with KPML

Process types

The following tree shows the process types defined in the Merged Upper Model and usable in SPL input specifications for generating natural language with the KPML system. The most common process types are shown in boldface, and followed by the participants (in brackets) that are inherent to them. An example sentence illustrates an instance of that process type.

1. RELATIONAL-PROCESS
    1.1. ONE-PLACE-RELATION
        1.1.1. **EXISTENCE** (:domain) *There is a book on the table.*
    1.2. TWO-PLACE-RELATION
        1.2.1. **GENERALIZED-POSSESSION** (:domain, :range) *I've got two brothers.*
        1.2.2. **CIRCUMSTANTIAL** (:domain, :range) *The stone weighs eight kilos.*
        1.2.3. INTENSIVE
            1.2.3.1. ASCRIPTION
                1.2.3.1.1. **PROPERTY-ASCRIPTION** (:domain, :range) *My tailor is rich.*
                1.2.3.1.2. **CLASS-ASCRIPTION** (:domain, :range) *My father is a teacher.*
                1.2.3.1.3. QUANTITY-ASCRIPTION
            1.2.3.2. UM-IDENTITY
            1.2.3.3. SYMBOLIZATION
2. MENTAL-PROCESS
    2.1. MENTAL-ACTIVE
    2.2. MENTAL-INACTIVE
        2.2.1. **COGNITION** (:senser, :phenomenon) *I know the answer.*
            2.2.1.1. BELIEVE
            2.2.1.2. KNOW
            2.2.1.3. THINK

2.2.2. **REACTION** (:senser, :phenomenon) *I don't like tea.*

2.2.2.1. LIKING

2.2.2.2. STRIVING

2.2.2.3. WANTING

2.2.2.4. DISLIKING

2.2.2.5. FEARING

2.2.3. **PERCEPTION** (:senser, :phenomenon) *Nobody saw the accident.*

3. VERBAL-PROCESS

3.1. **ADDRESSEE-ORIENTED-VERBAL-PROCESS** (:sayer, :saying, :recipient) *I told her the news.*

3.2. **NON-ADDRESSEE-ORIENTED-VERBAL-PROCESS** (:sayer, :saying) *I didn't say that.*

4. MATERIAL-PROCESS

4.1. DIRECTED-ACTION

4.1.1. **CREATIVE-MATERIAL-ACTION** (:actor, :actee, :beneficiary) *My brother has written a book.*

4.1.2. **DISPOSITIVE-MATERIAL-ACTION** (:actor, :actee, :beneficiary) *We have changed the first chapter.*

4.2. **NONDIRECTED-ACTION** (:actor) *He died.*

4.2.1. **AMBIENT-PROCESS** (usually no participant involved) *It's raining.*

Circumstances

This is a list of the circumstances recognized by the semantic organization built into KPML. Most circumstances types are defined just like participants: first you type the circumstance type after a colon (shown in blue below), and then in brackets you write the name of the circumstance, the semantics of the constituent that comes with the preposition and the rest of information (which is the same as for participants, because you always have participants or processes after prepositions). Other circumstances however have a more complex formalism, including two names and two places for semantics. NAME1 refers to the name given to the circumstance relation, while NAME2 refers to the name of the participant that comes with the preposition. Logically, the semantics relative to that participant are placed next to NAME2. Sometimes, however, it is important to specify the precise semantics to obtain the right generation. In these cases the semantics appears in bold. Examples, again drawing on the results that would be produced by the Nigel grammar of English, are:

```
:inclusive (accom-1 / object :lex money)
:destination (Fr / object :lex France)
:absolute-temporal-extent (tempo / object :lex day :number plural)
:matter-q matter
:matter-id
  (abo1 / empty :domain x :range (book / object :lex book :determiner the))
```

In red you can find the result of generation with these commands.

ACCOMPANIMENT

- **with**
  :inclusive ([name] / [semantics] :lex [item])
- **as well as**
  :additive ([name] / [semantics] :lex [item])
- **instead of**
  alternative ([name] / [semantics] :lex [item])
- **without**
  :exclusive ([name] / [semantics] :lex [item])

CAUSE

- **because of**
  :reason ([name] / [semantics] :lex [item])
- **for (purpose)**
  :purpose ([name] / [semantics] :lex [item])
- **for (client)**
  :client ([name] / [semantics] :lex [item])
- **in spite of**
  :causal-relation ([name] / [semantics] :lex [item])

COMPARISON

- **like**
  :similarity ([name] / [semantics] :lex [item])
- **similar to**
  :know-manner-q known
  :process-manner-id ([name1] / [semantics]
  :resemblance-q resemblance
  :formal-register-q formal
  :concrete-comparison-q concrete
  :domain x
  :range ([name2] / [semantics of the participant that comes with the preposition]))
- **different from**
  :know-manner-q known
  :process-manner-id ([name1] / [semantics]
  :resemblance-q resemblance
  :resemblance-type-q difference
  :domain x
  :range ([name2] / [semantics of the participant that comes with the preposition]))

MEANS

- **Adverbial Group**
  :manner ([name] / [semantics] :lex [item])

- **by (generalized means)**
  :generalized-means ([name] / [semantics] :lex [item])
- **by (enablement)**
  :enablement ([name] / [semantics] :lex [item])
- **by (agentive)**
  :agentive ([name] / [semantics] :lex [item])
- **by means of**
  :know-manner-q known
  :process-manner-id ([name1] / enablement
    :explicit-means-q explicit
    :domain x
    :range ([name2] / [semantics of the participant that comes with
      the preposition]))
- **with (instrumental)**
  :instrumental ([name] / [semantics] :lex [item])

SUBJECT-MATTER

- **concerning**
  :specific-matter ([name] / [semantics] :lex [item])
- **in the case of**
  :matter-q matter
  :matter-id ([name1] / specific-matter
    :matter-coverage-q clause
    :domain x
    :range ([name2] / [semantics of the participant that comes with
      the preposition]))
- about :diffuse-matter ([name] / [semantics] :lex [item])
- **as to**
  :matter-q matter
  :matter-id ([name1] / diffuse-matter
  :matter-coverage-q clause
  :domain x
  :range ([name2] / [semantics of the participant that comes with
    the preposition]))
- **of**
  :matter-q matter
  :matter-id ([name1] / diffuse-matter
    :formal-register-q formal
    :domain x
    :range ([name2] / [semantics of the participant that comes with
      the preposition]))

ROLE-PLAYING

- as :role-playing ([name] / [semantics] :lex [item])

TEMPORAL EXTENT

- **for (temporal extent)**
  :absolute-temporal-extent ([name] / [semantics] :lex [item])
- **in (temporal extent)**
  :relative-temporal-extent ([name] / [semantics] :lex [item])
- **during**
  :exhaustive-duration ([name] / [semantics] :lex [item])

SPATIAL EXTENT

- **for (spatial extent)**
  :absolute-spatial-extent ([name] / [semantics] :lex [item])
- **along**
  :parallel-extent ([name] / [semantics] :lex [item])
- **across**
  :nonparallel-extent ([name] / [semantics] :lex [item])

SPATIAL LOCATION

- **Adverbial Group**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / [semantics]
    :identifiability-q identifiable
    :location-relation-specificity-q unspecified
    :lex [item])
- **at (spatial location)**
  :spatial-locating ([name] / space-point :lex [item])
- **in (spatial location)**
  :spatial-locating ([name] / three-d-location :lex [item])
- **outside**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / [semantics]
    :containment-q noncontainment
    :domain x
    :range ([name2] / three-d-location :lex [item]))
- **inside**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / [semantics]
    :explicit-containment-q explicit
    :domain x
    :range ([name2] / three-d-location :lex [item]))
- **on**
  :spatial-locating ([name] / one-or-two-d-location :lex [item])
- **beside**
  :horizontal ([name] / [semantics] :lex [item])
- **next to**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / horizontal
    :immediate-adjacency-q adjacent

:specify-adjacency-q specified
:domain x
:range ([name2] / [semantics] :lex [item]))

- **between**
  :between ([name] / [semantics] :lex [item])
- **behind**
  :behind ([name] / [semantics] :lex [item])
- **below**
  :below ([name] / [semantics] :lex [item])
- **underneath**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / below
  :area-of-coverage-q partial
  :domain x
  :range ([name2] / [semantics] :lex [item]))
- **under**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / below
  :area-of-coverage-q partial
  :surface-contact-q noncontact
  :domain x
  :range ([name2] / [semantics] :lex [item]))
- **above**
  :above ([name] / [semantics] :lex [item])
- **over**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / above
  :area-of-coverage-q partial
  :surface-contact-q noncontact
  :domain x
  :range ([name2] / [semantics] :lex [item]))
- **on top of**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / above
  :area-of-coverage-q partial
  :domain x
  :range ([name2] / [semantics] :lex [item]))
- **in front of**
  :facing ([name] / [semantics] :lex [item]) to (destination)
  :destination ([name] / [semantics] :lex [item])
- **onto**
  :spatial-location-specification-q spatiallocation
  :spatial-location-id ([name1] / destination
  :domain x
  :range ([name2] / one-or-two-d-location :lex [item]))
- **into**
  :spatial-location-specification-q spatiallocation

:spatial-location-id ([name1] / destination
:domain x
:range ([name2] / three-d-location :lex [item]))

- **towards**
:spatial-location-specification-q spatiallocation
:spatial-location-id ([name1] / destination
:orientation-q oriented
:domain x
:range ([name2] / [semantics] :lex [item]))
- **from**
:source ([name] / [semantics] :lex [item])
- **off**
:spatial-location-specification-q spatiallocation
:spatial-location-id ([name1] / source
:domain x
:range ([name2] / one-or-two-d-location :lex [item]))
- **out of**
:spatial-location-specification-q spatiallocation
:spatial-location-id ([name1] / source
:domain x
:range ([name2] / three-d-location :lex [item]))
- **away from**
:spatial-location-specification-q spatiallocation
:spatial-location-id ([name1] / source
:orientation-q oriented
:domain x
:range ([name2] / [semantics] :lex [item]))

TEMPORAL LOCATION

- **Adverbial Group**
:temporal-location-specification-q temporallocation
:temporal-location-id ([name1] / [semantics]
:identifiability-q identifiable
:location-relation-specificity-q unspecified
:lex [item])
- **at**
:temporal-locating ([name] / time-point :lex [item])
- **in**
:temporal-locating ([name] / three-d-time :lex [item])
- **on**
:temporal-locating ([name] / one-or-two-d-time :lex [item])
- **by (temporal location)**
:temporal-ordering ([name] / [semantics] :lex [item])
- **before**
:anterior ([name] / [semantics] :lex [item])

- **until**
  :temporal-location-specification-q temporallocation
  :temporal-location-id ([name1] / anterior
  :period-extremal-q periodextremal
  :domain x :range ([name2] / [semantics] :lex [item]))
- **after**
  :posterior ([name] / [semantics] :lex [item])
- **since**
  :temporal-location-specification-q temporallocation
  :temporal-location-id ([name1] / posterior
  :period-extremal-q periodextremal :domain x :range ([name2] / [semantics] :lex [item]))
- **from**
  :temporal-location-specification-q temporallocation
  :temporal-location-id ([name1] / posterior
  :period-extremal-q periodextremal
  :period-time-or-state-q state-or-activity
  :domain x
  :range ([name2] / [semantics] :lex [item]))

Sentence Types

Many of these specifications are actually 'macros', which means that they are shorthand for something more complex. They allow to write simple SPLs without worrying about the semantic specification that is behind them. For example, ":tense present" produces a sentence in the simple present tense without requiring that you know that this shorthand for a specific set of temporal relations between the time of speaking and the time of the event described. There are occasions when you need to delve more deeply, but for a beginning you can often get by without.

- **WH-QUESTIONS**
  :speech-act-id (q / question :polarity positive)
  :question-item-id [name of the participant or circumstance asked about]
- **YES/NO-QUESTIONS**
  :speech-act-id (q / question :polarity variable)
- **TENSE**
  :tense [present, past, future, present-continuous (or present-progressive), past-continuous (or past-progressive), future-continuous (or future-progressive), present-perfect, past-perfect, future-perfect, future-in-present (going to), present-perfect-continuous]
- **VOICE**
  :voice [active/passive]
- **POLARITY**
  :polarity [positive/negative]

- **MODALITY**
  :modality [can, cant, could, couldnt, may, might, must, neednt, should, shouldnt, will, wont, would, wouldnt]

Bibliography

[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 452–457, Sydney (Australia), 1991.

[Bateman *et al.*, 1995] John A. Bateman, Bernardo Magnini, and Giovanni Fabris. The generalized upper model knowledge base: Organization and use. In N. J. I. Mars, editor, *Towards very large knowledge bases: knowledge building and knowle dge sharing*, pages 60–72, Amsterdam, 1995. IOS Press.

[Bateman, 1990] John A. Bateman. Upper modeling: organizing knowledge for natural language processing. In *Proceedings of the Fifth International Natural Language Generation Workshop*, pages 54–60, Pittsburgh, PA., 1990. Organized by Kathleen R. McKeown (Columbia University), Johanna D. Moore (University of Pittsburgh) and Sergei Nirenburg (Carnegie Mellon University). Held 3-6 June 1990, Dawson, PA.

[Bateman, 1997a] John A. Bateman. Enabling technology for multilingual natural language generation: the KPML development environment. *Journal of Natural Language Engineering*, 3(1):15–55, 1997.

[Bateman, 1997b] John A. Bateman. *KPML Development Environment: multilingual linguistic resource development and sentence generation*. German National Center for Information Technology (GMD), Inst itute for integrated publication and information systems (IPSI), Darmstadt, Germany, January 1997. (Release 1.1).

[Bechhofer *et al.*, 2003] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.

[Brennan *et al.*, 1987] Susan E. Brennan, Marilyn W. Friedman, and Carl J. Pollard. A centering approach to pronouns. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics*, pages 155–162, Morristown, NJ, USA, 1987. Association for Computational Linguistics.

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. Ontology-based database access. In *Proc. of the 15th Italian Conf. on Database Systems (SEBD 2007)*, 2007.

[Cheng *et al.*, 1997] Hua Cheng, Chris Mellish, and Michael O'Donnell. Aggregation based on text structure for descriptive text generation. In *Proceedings of the PhD Workshop on Natural Language Generation, 9th European Summer School in Logic, Language and Information (ESSLLI97)*, Aix-en-Provence, France, 1997.

[Dongilli *et al.*, 2004] Paolo Dongilli, Enrico Franconi, and Sergio Tessaris. Semantics driven support for query formulation. In *Description Logics*, 2004.

[Dongilli *et al.*, 2006] Paolo Dongilli, Sergio Tessaris, and John Bateman. Leveraging Systemic-Functional Linguistics to Enhance Intelligent Database Querying. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, Jinan, China, October 2006.

[Dongilli, 2007a] Paolo Dongilli. Centering-theory-based text planning of a conjunctive query. In *Proceedings of the 3rd Language & Technology Conference*, Lecture Notes in AI (LNAI), Poznan, Poland, October 2007. Springer Verlag.

[Dongilli, 2007b] Paolo Dongilli. Discourse planning strategies for complex concept descriptions. In *Proceedings of the 7th International Symposium on Natural Language Processing (SNLP-2007)*, Pattaya, Chonburi, Thailand, December 2007.

[Elhadad and Robin, 1996] Michael Elhadad and Jacques Robin. An overview of SURGE: a reusable comprehensive syntactic realization component. Technical Report Technical Report 96-03, Computer Science, Ben Gurion University of the Negev, Beer Sheva, Israel, 1996.

[Elhadad, 1992] Michael Elhadad. *Using argumentation to control lexical choice: a functional unification-based approach*. Phd dissertation, Department of Computer Science, Columbia University, 1992.

[Elhadad, 1993] Michael Elhadad. FUF: the Universal Unifier. User Manual Version 5.2. Technical Report Technical Report CUCS-038-91, Department of Computer Science, Columbia University, New York, 1993.

[Grosz and Sidner, 1986] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[Grosz *et al.*, 1986] Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Towards a computational theory of discourse interpretation. Unpublished ms., 1986.

[Grosz *et al.*, 1995] Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225, 1995.

[Halliday and Matthiessen, 1999] Michael A. K. Halliday and Christian M. I. M. Matthiessen. *Construing experience through meaning: a language-based approach to cognition*. Cassell, London, 1999.

[Halliday and Matthiessen, 2004] Michael A. K. Halliday and Christian M.I.M. Matthiessen. *An Introduction to Functional Grammar*. Edward Arnold, London, 3rd edition, 2004.

[Henschel and Bateman, 1994] Renate Henschel and John A. Bateman. The merged upper model: a linguistic ontology for German and English. In *Proceedings of COLING '94*, volume II, pages 803–809, Kyoto, Japan, August 1994.

[Henschel, 1993] Renate Henschel. Merging the English and the German Upper Model. Technical report, GMD/Institut für Integrierte Publikations- und Informationssysteme, Darmstadt, Germany, 1993. Appears as: *Arbeitspapiere der GMD*, **848**, June 1994. GMD, Sankt Augustin.

[Horrocks and Tessaris, 2002] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer-Verlag, 2002.

[Horrocks *et al.*, 2000] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[Horrocks *et al.*, 2003] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[Hovy, 1997] Eduard Hovy. *Survey of the state of the art in human language technology*, chapter 4.1, Language generation: overview, pages 139–146. Cambridge University Press, New York, NY, USA, 1997.

[Kasper, 1988] Robert T. Kasper. Systemic grammar and functional unification grammar. In James D. Benson and William S. Greaves, editors, *Systemic Functional Approaches to Discourse*, pages 176–199. Ablex, Norwood, New Jersey, 1988. Also available as USC/Information Sciences Institute, Reprint Report ISI/RS-87-179, 1987.

[Kasper, 1989] Robert T. Kasper. A flexible interface for linking applications to PENMAN's sentence generator. In *Proceedings of the DARPA Workshop on Speech and Natural Language*, 1989.

[Kay, 1979] Martin Kay. Functional grammar. In Berkeley Linguistics Society, editor, *Proceedings of the 5th Meeting of the Berkeley Linguistics Society*, 1979.

[Kibble and Power, 2004] Rodger Kibble and Richard Power. Optimizing referential coherence in text generation. *Computational Linguistics*, 30(4):401–416, 2004.

[Lavoie and Rambow, 1997] Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the 5th. Conference on Applied Natural Language Processing*, pages 265–268, Washington, D.C., 1997. Association for Computational Linguistics.

[MacGregor and Bates, 1987] Robert MacGregor and Raymond Bates. The LOOM knowledge representation language. In *Proceedings of the Knowledge-Based Systems Workshop*, 1987. Held in St. Louis, Missouri, April 21-23, 1987. Also available as ISI reprint series report, RS-87-188, USC/Information Sciences Institute, Marina del Rey, CA.

[Mann and Matthiessen, 1983] William C. Mann and C. M. I. M. Matthiessen. Nigel: A systemic grammar for text generation. Technical Report RR-83-105, USC/Information Sciences Institute, February 1983. This paper also appears in a volume of the *Advances in Discourse Processes Series*, R. Freedle (ed.): *Systemic Perspectives on Discourse: Volume I.* published by Ablex.

[Mann, 1983a] William C. Mann. The anatomy of a systemic choice. *Discourse Processes*, 1983. Also available as USC/Information Sciences Institute, Research Report ISI/RR-82-104, 1982.

[Mann, 1983b] William C. Mann. An overview of the penman text generation system. In *AAAI*, pages 261–265, 1983.

[Matthiessen, 1981] C. M. I. M. Matthiessen. A grammar and a lexicon for a text-production system. In *The Nineteenth Annual Meeting of the Association for Computational Linguistics*. Sperry Univac, 1981.

[Matthiessen, 1983] Christian M. I. M. Matthiessen. The systemic framework in text generation: Nigel. In W. Greaves & J. Benson, editor, *Systemic Perspectives on Discourse*. Ablex, 1983.

[Matthiessen, 1987] Christian M. I. M. Matthiessen. Notes on the organization of the environment of a text generation grammar. In Gerard Kempen, editor, *Natural Language Generation: Recent Advances in Artificial Intelligence, Psychology, and Linguistics*. Kluwer Academic Publishers, Boston/Dordrecht, 1987. Paper presented at the Third International Workshop on Natural Language Generation, August 1986, Nijmegen, The Netherlands.

[Matthiessen, 2005] Christian M. I. M. Matthiessen. Remembering bill mann. *Comput. Linguist.*, 31(2):161–172, 2005.

[McRoy *et al.*, 2001] Susan W. McRoy, Songsak Channarukul, and Syed S. Ali. Creating natural language ouput for real-time applications. *Intelligence*, 12(2):21–34, 2001.

[Melengoglou, 2002] Alexander Melengoglou. Multilingual aggregation in the m-piro system. Master's thesis, University of Edinburgh, 2002.

[Mellish, 1988] Christopher S. Mellish. Implementing systemic classification by unification. *Journal of Computational Linguistics*, 14(1):40–51, 1988.

[Mel'čuk and Žolkovskij, 1970] Igor A. Mel'čuk and Alexander K. Žolkovskij. Towards a Functioning Meaning-Text Model of Language. *Linguistics*, 57:10–47, 1970.

[Merriam-Webster, 2007] Merriam-Webster. Merriam-Webster's Online Dictionary. `http://www.merriam-webster.com`, 2007.

[Power and Scott, 1998] Richard Power and Donia Scott. Multilingual Authoring Using Feedback Texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98)*, pages 1053–1059, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[RealPro, 2008] RealPro. The RealPro text generation engine. `http://www.cogentex.com/technology/realpro/index.shtml`, 2008.

[Reape and Mellish, 1999] Mike Reape and Chris Mellish. Just what *is* aggregation anyway? In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 20–29, Toulouse, 1999.

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

[Sirin and Parsia, 2007] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.

[SWT, 2007] SWT. The Standard Widget Toolkit. `http://www.eclipse.org/swt`, 2007.

[Teich, 1999] Elke Teich. *Systemic Functional Grammar in Natural Language Generation: Linguistic Description and Computational Representation*. Cassell, London, 1999.

[Žolkovskij and Mel'čuk, 1967] Alexander K. Žolkovskij and Igor A. Mel'čuk. O semantičeskom sinteze. *Problemy kibernetiki*, 19(?):177–238, 1967.

[Wilkinson, 1995] John Wilkinson. Aggregation in natural language generation: Another look. Co-op work term report, September 1995.

[Zorzi *et al.*, 2007] Ivan Zorzi, Sergio Tessaris, and Paolo Dongilli. Improving Responsiveness of Ontology-Based Query Formulation. In *Proceedings of the 4th Italian Workshop on Semantic Web Applications and Perspectives (SWAP 2007)*, Bari, Italy, December 2007.