Chapter 3

Overall FEMsum Architecture

As pointed out in Chapter 1, one of the main objectives of this thesis is to provide a general framework to allow the use of several Automatic Summarization (AS) techniques and approaches supplying capabilities to solve different summarization tasks. Each task is supposed to face specific user information needs, including: to process one single document vs. to process several documents; to take into account the information provided by a user query vs. to produce generic text-driven summaries; to assume a previous user knowledge giving only just-the-new information vs. the production of background summaries; and to produce summaries of different lengths (ranging, in our experiments, from 10 to 250 words). This general framework has to be able to deal with different kinds of digital documents such as news reports or scientific presentations. Moreover, this input can be of different media (e.g. text, speech), language (e.g. English, Spanish, Catalan) or domain (e.g. journalistic, scientific).

This chapter presents the Flexible Eclectic Multitask summarizer (FEMsum), a highly modular and parameterizable AS system able to face all the previously exposed requirements, choosing different techniques for each situation. A global view of the system is presented in Section 3.1, Section 3.2 presents some architecture design considerations. To introduce different instantiations[1] of each general architecture component, Section 3.3 presents the set of constituents of various approaches, and the following Sections detail different implementations of each possible constituent. To finish, Section 3.9 presents the implementation framework to integrate new components. In the following chapters, the FEMsum instantiations presented in this chapter are evaluated in different summarization tasks.

[1]The term "instantiation" refers to the implemented FEMsum components used to built each summarization approach.

3.1 Global Architecture

Figure 3.1 depicts the basic functionalities and the global architecture of FEMsum. The functional requirements are set by means of a number of parameters split into input, purpose and output settings (see left and right boxes in the Figure). As detailed in Chapter 1, input settings refer to the characteristics of the documents to be summarized, while purpose settings deal with the characteristics of the final utility of the summary and output settings refer to the content and presentation of the summary.

Input settings include the following items:

- Domain. The approach to be instantiated can be domain independent or of restricted domain. In the latter case, additional knowledge sources such as frequent terms (scientific domain[2]) or gazetteers (geographical domain[3]), can be included.
- Document structure. Information derived from the document structure (position, title, sections or available tags) can be considered. For instance, in the case of summarizing oral presentations, words appearing in each section can be contrasted with those in the slides in order to help in the identification of the voice segment to be reproduced as a summary[4].
- Language. All the implemented approaches are language dependent. Currently, texts in English or Spanish and in some cases in Catalan can be processed.
- Media. Depending on the scenario to be dealt with, different media (video, audio, well written text or any kind of digital document) are considered. However, FEMsum only processes the textual representation of documents, which means that, for instance, the conversion from an audio file to an automatic transcript must be previously carried out by an ASR.
- Unit. Both single documents and collections of related documents are processed for SDS and MDS respectively. The set of documents to be summarized has to be previously collected.
- Genre. Both documents independent of the style of writing and dependent ones are considered. An example of genre dependent approach is the one exploiting the fact that a journalistic document follows a pyramidal organization. Also the approach instantiated as genre dependent, applied to scientific documents, exploits the fact that in spontaneous speech documents relevant words have higher frequency than in written text.

[2]Both domain independent and restricted domain approaches have been instantiated. In the latter case, papers from the European Conference on Speech Communication and Technology have been used as instances for the scientific domain. However, moving to other domains is easy if there is a specific corpus available (detailed in Chapter 5).

[3]It has not been evaluated.

[4]It has not been evaluated

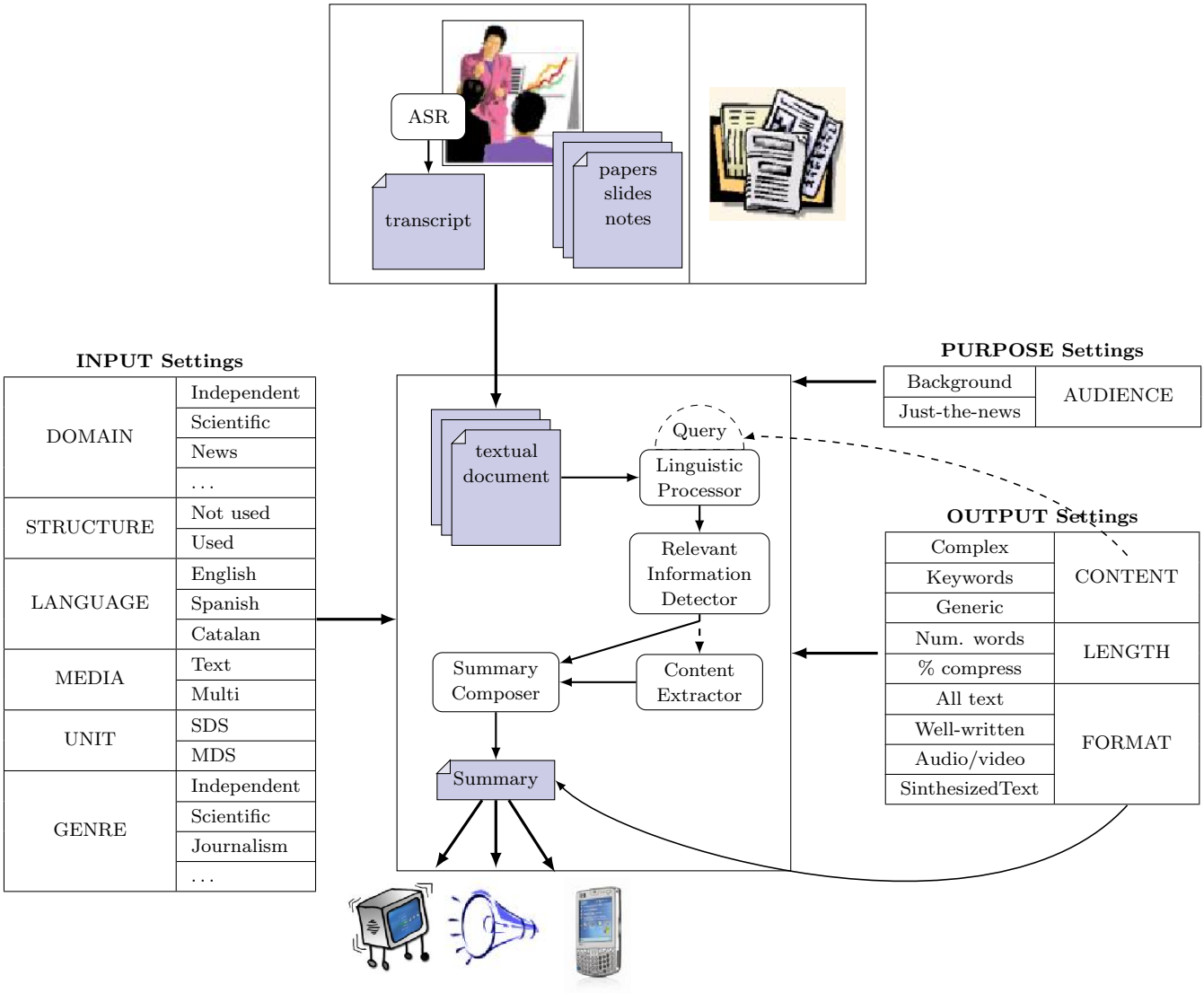

Figure 3.1: FEMsum Global Architecture

Purpose settings include:

- Audience. Summaries can be adapted to the needs of specific users, such as the user's prior knowledge on a determined subject. Background summaries assume that the reader's prior knowledge is poor, and so extensive information is supplied, while just-the-news are summaries conveying only the newest information on an already known subject, assuming the reader is familiar with the topic[5].

Output settings include:

- Content. The fragments of the documents to be included in the summary can be conditioned or not by the content of the user information need (question, topic description or list of words).
- Size. Summary length can be restricted in terms of document compression rate or number of summary words.
- Document restriction. Used for filtering out some type of documents, such as to allow parts of the transcript to appear in the final summary or not to allow it.
- Output format. The final summary can be presented to the user as text, voice synthesized text or audio/video stream. Although different techniques can be applied for each one, the FEMsum output is always textual and the final presentation is generated in a posterior phase.

In order to face the functional requirements described above, the system is organized in three language independent components (see the central box of Figure 3.1): *Relevant Information Detector (RID)*, *Content Extractor (CE)* and *Summary Composer (SC)*. In addition, there is a language dependent *Linguistic Processor (LP)* and a *Query Processor (QP)* component. As detailed in Section 3.3, not all the components are needed in all FEMsum instantiations, and different implementations of each component are proposed.

3.2 Component Architecture Design Considerations

Our work is based on the assumption that an AS system (as well as a manual one) follows a summarization process model. The aim of the FEMsum architecture is to allow the implementation of any summarization process. The first difference when analyzing the process models presented at the begining of the previous chapter is the fact that they differ in the number of suggested phases (stages or kind of rules):

[5]The approach developed to participate in the DUC 2007 update task produced several summaries for a same topic, each new summary was supposed to contain only just-the-news information with respect to the previous one.

- (Cremmins 1982) proposes 4 stages: identifying relevant information; extracting; organizing and reducing the relevant information into a coherent unit, usually one paragraph long; and refining the abstract through editing.

- (Brown and Day 1983) suggest to apply 5 different rules: to remove irrelevant information; to overcome redundancy; to produce concept superarrangement; to select the thematic sentence representing the text, if possible; and to construct the abstract.

- (Endres-Niggemeyer 1989) differentiates 2 kind of rules: analytical and synthetical. The first one includes reducing and selecting; while the synthesis rules consist in clarifying, reorganizing, and stylizing.

- (Pinto Molina 1995) proposes 3 operative stages: reading-understanding; selection and interpretation; and synthesis.

- (Sparck-Jones 1999; Mani and Maybury 1999; Hahn and Mani 2000; Hovy 2001; Mani 2001) distinguish 3 stages: analyzing the input text; transforming it into a summary representation; and synthesizing.

- (Sparck-Jones 2001) identifies 3 subproblems: locating the relevant fragments; ranking these fragments by relevance; and producing a summary.

- (Hovy 2005) also identifies 3 different stages: topic identification; interpretation and summary generation.

In order to validate the proposed architecture, in Table 3.1 each model phase has been associated to a FEMsum component. Each phase can involve several tasks that can be solved by different components, which is the case of the (Cremmins 1982) phase *organizing and reducing into a coherent unit refining by editing*. This phase can be decomposed in several tasks, and, for instance, the *organizing* and/or the *reducing* tasks can be carried out by the *Content Extractor* or the *Summary Composer* FEMsum components (see the last two columns of the first row in Table 3.1). When a phase can be addressed in different components, the symbol '*' is used to indicate the first and the last components involved. It is also possible that one phase is solved by several components, that is the case of the (Cremmins 1982) phase *identifying relevant information*, indicated by 'x'. This phase can be associated to the *Linguistic Processor*, *Query Processor* and *Relevant Information Detector* FEMsum components (see the first two columns of the first row of the Table 3.1).

Moreover, it can be observed that different tasks are placed at different stages by different models. An example of that is the reducing task, while (Cremmins 1982) places it as one of the last stages, the reducing (removing irrelevant information) task is one of the first tasks in (Brown and Day 1983) model.

Summarization Process Model	***L*inguistic and *Q*uery Processor (LP, QP)**	***R*elevant *I*nformation *D*etector (RID)**	***C*ontent *E*xtractor (CE)**	***S*ummary *C*omposer (SC)**
Cremmins (1982)	x – identifying relevant information - x		extracting * —— organizing and reducing —— *	into a coherent unit refining by editing
Brown and Day (1983)	x – removing irrelevant information - x		overcoming redundancy concept arrangement selecting	abstracting
Endres-Niggemeyer (1989)	x —— reducing and selecting —— *			clarifying reorganizing synthesizing
Pinto Molina (1995)	reading-understanding * —— selecting and interpreting —— *			synthesizing
Sparck-Jones (1999)	x —— analyzing —— x		transforming	synthesizing
Sparck-Jones (2001)	x —— locating —— x		ranking	producing
Hovy (2005)	x —— topic identifying —— x		interpreting	generating

Table 3.1: Association of summary production process model stages to the corresponent architecture component.

To allow the maximum flexibility, the main restriction of the proposed architecture is the input and the output expected by each component. As shown in Figure 3.2, the *LP* component enriches the original text (documents to be summarized or the user information need) with linguistic information. This component consists of a pipeline of more or less sophisticated language dependent linguistic tools (see details in Section 3.4). Different *LP* instantiations can be used depending on the requirements of the approach, as well as the language, media, genre or domain of the documents to be summarized. If a purely lexical FEMsum approach is instantiated, *LP* can be reduced to a segmentation task: splitting the input document into sentences or paragraph units, each composed by a sequence of words[6]. However, other approahes can require more sophisticated language dependent tools to enrich the text with syntactic or

[6]It is possible that the segmentation process only requires a simple word stemming or introduces some complexity detecting Multi-Word (MW)s, terms, NEs. However, straightforward methods, usually language independent, can be applied with a rather small decrease in accuracy.

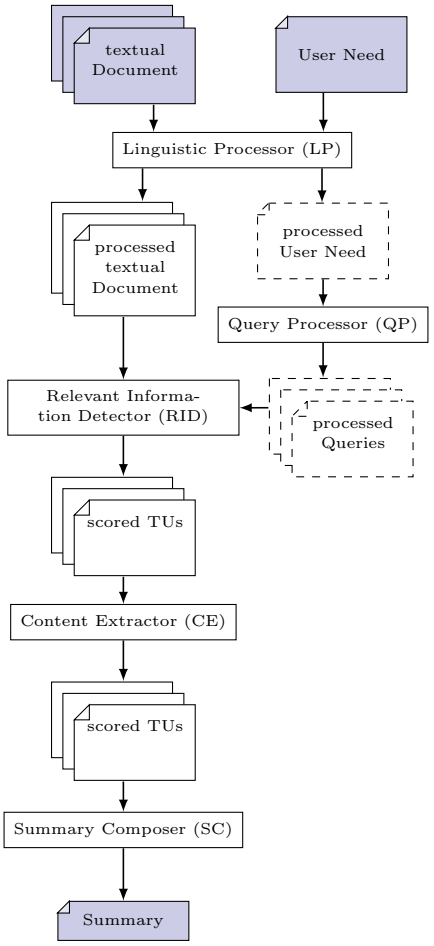

Figure 3.2: FEMsum data flow

semantic information.

The input to the *RID* module is the document or set of documents to be summarized with more or less linguistic information. The original user need linguistically processed is the input of the QP component. The output of this component, expressing the user need, can be taken into account by *RID* to score the set of relevant TUs (segments, phrases or sentences). The output of *RID* is a set of TU identifiers ranked by relevance. The linguistic information and the relevance score of the TUs is the input to the *CE* and the *SC* components.

The main function of the *CE* component is to extract and score by relevance summary candidate TUs. This is not always instantiated, but when it is, this component allows to apply different heuristics taking into account some input aspect, such as the genre of the document (journalistic vs. scientific); some purpose aspect, such as the type of audience of the produced

summaries (background vs. just-the-news); or some output aspect, such as the content of the summary (text-driven vs. query-driven.

The final text summary is the output of the *SC*. This component carries out the post-processing of the summary content. This post-processing can be more or less elaborated, taking into account the size or the format of the summary. In this, component the summary TUs can be simplified, paraphrased, reordered, or removed.

3.3 Components used by the instantiatiated FEMsum approaches

This section briefly exemplifies the implemented FEMsum components instantiated by several FEMsum approaches. Each approach has been evaluated when summarizing documents from different domain, genre, media, and language (see Chapter 4, 5, 6,and 7). A brief description of each approach is presented bellow:

- *LCsum* is based in shallow text-oriented techniques, mainly lexical chains. This approach is evaluated for textual Spanish and Catalan documents in Section 4.1 and for spontaneous speech transcripts in Chapter 5.

- *MLsum* applies a Machine Learning techniques in the detection of the most relevant sentences. A first approach, based on Decision Tree, participated in the DUC 2003 headline production task, see Section 4.2. A second approach, based on Support Vector Machine, is evaluated using DUC 2005 data, see Section 6.1.4.

- *QAsum* is a quick adaptation of TALP-QA factual Question & Answering system that participated in DUC 2005. The evaluation of this approach is reported in Section 6.1.2.

- *LEXsum* is based on the use of lexical features. This approach was evaluated in the DUC 2006 contest (see Section 6.1.3) as well as when dealing with scientific oral presentations (experiments reported in Chapter 7)

- *SEMsum* this approach uses a graph based algorithm that takes into account semantic information. This approach was also evaluated in DUC 2006 contest (see Section 6.1.3) and in the scientific oral presentation domain (see Chapter 7). In addition, participated in DUC 2007 tasks, see Section 6.1.5 and 6.2.

Table 3.2 summarizes the components used by each evaluated FEMsum approach. When an approach has several possible instantiations for one component, a parameter is used. *LP* is not specified in the table because the component to be instantiated depends on properties of the input document.

FEMsum approach	***Q*uery *P*rocessor (QP)**	***R*elevant *I*nformation *D*etector (RID)**	***C*ontent *E*xtractor (CE)**	***S*ummary *C*omposer (SC)**
LCsum	—	LC RID	LC CE (heuristic)	Basic SC
LCsum (DiscStr)	—	LC RID	LC CE (heuristic)	Complex SC (sentCompressor)
LCsum (PostSeg)	—	LC RID	—	Complex SC (chkExtractor)
MLsum	—	FE RID	ML CE (DT)	Basic SC
MLsum (HL)	—	FE RID	ML CE (DT)	Complex SC (sentCompressor)
MLsum (MDS,Q)	Splitting QP	FE RID (MDS,Q)	ML CE (SVM, sem)	Basic SC
QAsum (TALP_QA)	Generating QP	PR RID (TALP_QA)	QA CE	Complex SC (antiredundancy)
LEXsum	Splitting QP	PR RID	—	Basic SC
SEMsum	Splitting QP	PR RID	Sem CE	Basic SC
SEMsum (reRank)	Splitting QP	PR RID (reRank)	Sem CE	Basic SC
SEMsum (reRank,update)	Splitting QP	PR RID (reRank)	Sem CE (update)	Complex SC (reordering)

Table 3.2: Components of the instantiated FEMsum approaches

Some approaches can use less components than others, if a component can be ommitted it is indicated by '–'. Examples of that are text-driven approaches, which do not process any specific user information need (see the first column of the five first rows of Table 3.2).

All the *LCsum* approaches instantiate the *RID* component that uses the *LEXICAL CHAINER* described in Section 3.6.1, *Lexical Chain (LC) RID*. The main difference between them is the *SC* component. In the *PreSeg* (default option, first row in Table 3.2) and in the *DiscStr* (second row in Table 3.2), as in classical Text Summarization, segmentation is part of the *LP* component, while in the *PostSeg* approach (third row), summary text segment boundaries are defined in the *SC* component, after the detection of relevant elements in the document. That is especially useful when dealing with ASR output without punctuation marks or capitalization. What a TU is depends basically on the media of the input and the processing capabilities (e.g. sentences or clauses in well written text, sequences of words in spontaneous speech). The TU ranking takes into account the *LEXICAL CHAINER* output, and, as detailed in Section 3.8.1, the main characteristic of *LCsum(DiscStr)* is the use of rhetorical information in the *SC* component. The *LC*

CE allows the use of different heuristics to extract relevant TUs taking into account document properties (see Section 3.7.1).

MLsum instantiates the Feature Extractors (described in Section 3.6.2), *Feature Extractor (FE) RID*, and the *TU Classifiers* (described in Section 3.7.2), *ML CE*. Since *MLsum(HL)* was designed to produce headlines for the DUC 2003 task, *HL* (fifth row) uses the TU Full Parsing Sentence Compression module described in Section 3.8.1, although by default (fourth row) the *Basic SC* component extracts whole TUs as summary.

The rest of the table refers to MDS query-focused approaches. As can be observed looking at the second column of Table 3.2, with the exception of *MLsum(MDS,Q)*, the query-focused approaches instantiate a Passage Retrieval (PR) based RID component, PR RID.

QAsum uses part of a Question & Answering (QA) system in the *RID* and *CE* components (*PR RID* in Section 3.6.3 and *QA CE* in Section 3.7.3) and redundancy is eliminated in the *SC* component (see the Antiredundancy module described in Section 3.8.1). In contrast, *SEMsum* and *MLsum(MDS,Q)* deal with redundant information in the *CE* component (see Section 3.7.4).

The most important difference between *LEXsum* and *SEMsum* approaches is that while in *SEM CE*, described in Section 3.7.4, a rich semantic representation is needed, *LEXsum* only requires lexical information. In fact, *LEXsum* uses the same *RID* and *SC* components than *SEMsum*, but it does not instantiate the *CE* component.

In *SEMsum*, an optional reranking can be applied to the TUs detected as relevant. Moreover, in *SEM CE* some TUs can be discarded in the update option (see Section 3.7.4) to extract just-the-news information.

All the instantiated *SC* components produce extract-based summaries. However, an important difference is that while *QAsum*, *LEXsum* and *SEMsum* summaries are composed by complete TUs from the input documents, *LCsum* and *MLsum* allow the use of fragments of the original TU, by exploiting information associated to discourse markers. *LCsum* only requires partial chunking in *LCsum(PostSeg)* and partial parsing in *LCsum(DiscStr)*. In *MLsum(HL)*, full parsing is required.

Some post-editing is carried out in the *SC* component (considering referents instead of pronouns or organizing the summary content).

Each FEMsum approach can be mapped with one of the summarization models presented in Section 3.2. Relations can be established by analyzing the tasks proposed by the authors in each phase of the model. Table 3.3 synthesizes these relations: *QAsum* follows (Cremmins 1982); *SEMsum* (Brown and Day 1983); *LEXsum* (Endres-Niggemeyer 1989); *MLsum* (Sparck-Jones 1999; Mani and Maybury 1999; Hahn and Mani 2000; Hovy 2001; Mani 2001); and *LCsum* (Sparck-Jones 2001).

FEMsum approach components	**Summarization model stages**
QAsum	(Cremmins 1982)
Generating QP	
PR RID	1. relevent information detection
QA CE	2. extraction
Complex SC	3. organizing and reducing into a coherent unit
(antiredundancy)	4. refining by editing
SEMsum	(Brown and Day 1983)
Splitting QP	
PR RID	1. removing irrelevant information
SEM CE	2. overcoming redundancy
(update)	3. concept arrangement, 4. selecting
Complex SC	
(ordering)	5. abstracting
LEXsum	(Endres-Niggemeyer 1989)
Splitting QP	1. analytical rules:
PR RID	reducing and selecting
Basic SC	2. synthesis rules: reorganizing
	clarifying and stylizing
MLsum	(Sparck-Jones 1999)
Splitting QP	
FeatExtr RID	1. analyzing
ML CE	
(sem)	2. transforming
Complex SC	
(sentCompressor)	3. synthesizing
LCsum	(Sparck-Jones 2001)
LexChain RID	1. locating
LexChain CE	
(heuritic)	2. ranking
Complex SC	
(sentCompressor/chkExtractor)	3. producing

Table 3.3: FEMsum approaches related with a summarization model.

As the instantiated FEMsum approaches try to analyze similarities and differences between TUs, it can not be considered that a whole interpretation of the documents is carried out. That is why no correspondence has been established with the models proposed by (Pinto Molina 1995) or (Hovy 2005).

FEMsum architecture is extensible allowing the addition of new components and the refinement of the existing ones. In fact, the core of the system was designed after our participation in DUC 2005. The architecture was robust enough to include all the components previously designed for SDS prototypes as well as those included to participate in DUC 2006-2007 tasks and to summarizes different types of documents. Details about the implementation framework used to add new components is discussed in Section 3.9.

Following the notation described in Figure 3.3, a diagram models each general architecture component (see Figure 3.4 for *LP* in Section 3.4, Figure 3.6 for *QP* in Section 3.5, Figure 3.9 for *RID* in Section 3.6, Figure 3.13 for *CE* in Section 3.7, and 3.20 for *SC* in Section 3.8).

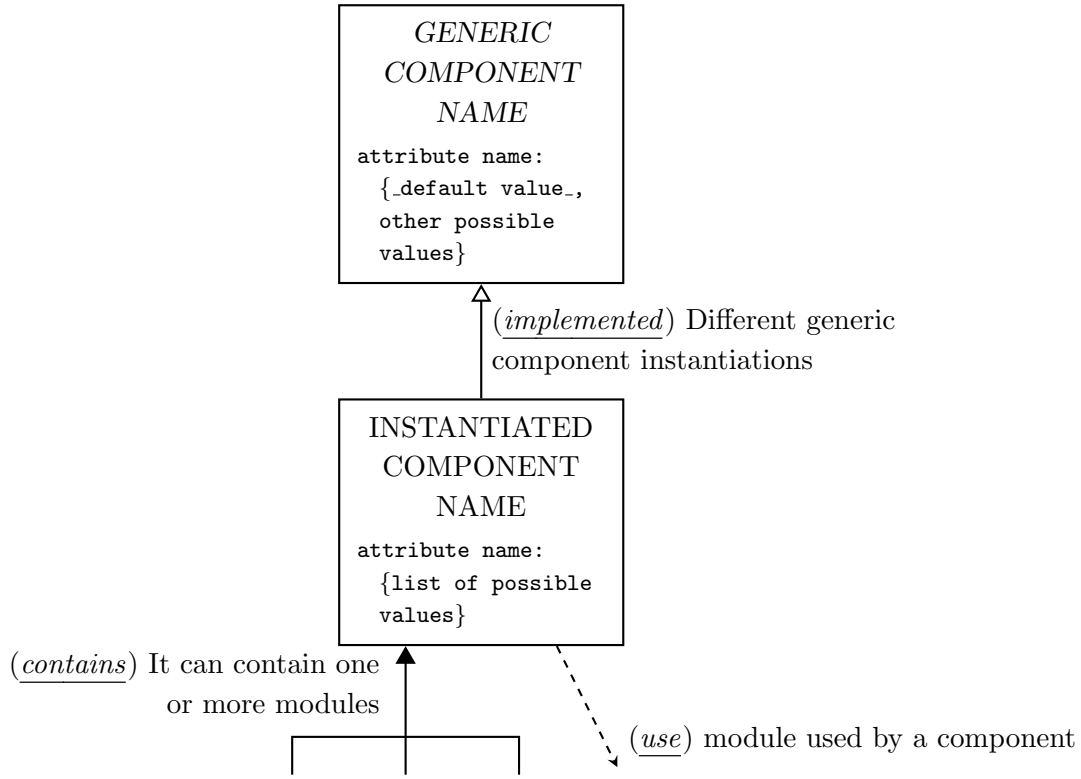

Figure 3.3: Notation used in the diagrams depicted in Figures 3.4, 3.6, 3.9, 3.13, 3.20

3.4 Linguistic Processor Component

This section describes the FEMsum component responsible for the linguistic processing of the documents to be summarized, the *LP*. As reflected in Figure 3.4, the *LP* can process four different generic representations: the *LexicoMorpho Representation*, the *Syntactic Representation*, the *Syntactic Phrase* and the *Semantic Representation*. As can be observed in Figure 3.4, each one is implemented with a language dependent module.

The specific tools to be used to produce the *LexicoMorpho Representation* can vary depending not only on the language but on the media (well-written text or spontaneous speech); the domain (independent, journalistic or scientific); or the type of TU involved (sentences, paragraphs, word segments). Furthermore, this TU segmentation can be carried out a posteriori, by another FEMsum component (*pos*) or the TUs can be previously manually established (*man*), otherwise *LP* will automatically (*auto*) segment the input in TUs. A specific multi-word repository can be used to segment lexical units.

Details about the specific NLP tools instantiated in the FEMsum approaches are presented in Section 3.4.1 and Section 3.4.2,

A generic *LP* component is illustrated in Figure 3.5. It consists of a pipeline of general purpose Natural Language (NL) processors performing: tokenization, Part Of Speech (POS) tagging, lemmatization, fine grained Named Entity Recognition and Classification (NERC), WordNet (WN) or EuroWordNet (EWN) based semantic tagging, collocation annotation, coreference resolution, discourse marker annotation, syntactic parsing (both full parsing and shallow parsing), and semantic role labeling. Figure 3.5 does not include TU segmentation, because this task is not always performed by the *LP* component.

Different levels of linguistic enrichment are needed depending on the FEMsum approach; if not necessary, not all constituents are activated. For instance, the semantic analyzer is not used by *LCsum*, *MLsum* or *LEXsum*. In fact, *LP* can be reduced to a segmentation task: splitting the input document into sentences or paragraph units, each composed by a sequence of words (simple word stemming, multiwords or terms, NEs); or it can consist of a more sophisticated language dependent process.

Figure 3.5 shows several *LP* output types consisting of TUs enriched with *LexicoMorpho* (*sent*) and *Syntactic* (*sint*) language dependent representations. The *Syntactic Representation* includes syntactic constituent structure (including head specification) and the syntactic relations between constituents (subject, direct and indirect object, modifiers). The *Semantic Representation* of the TU, the environment (*env*), is composed by *sent* and *sint*. The three levels of a text representation produced by *LP* are:

- The *LexicoMorpho Representation*, *sent*, provides lexical information for each word: form, lemma, POS tag, semantic class of NE (and subclasses if available), list of WN or EWN

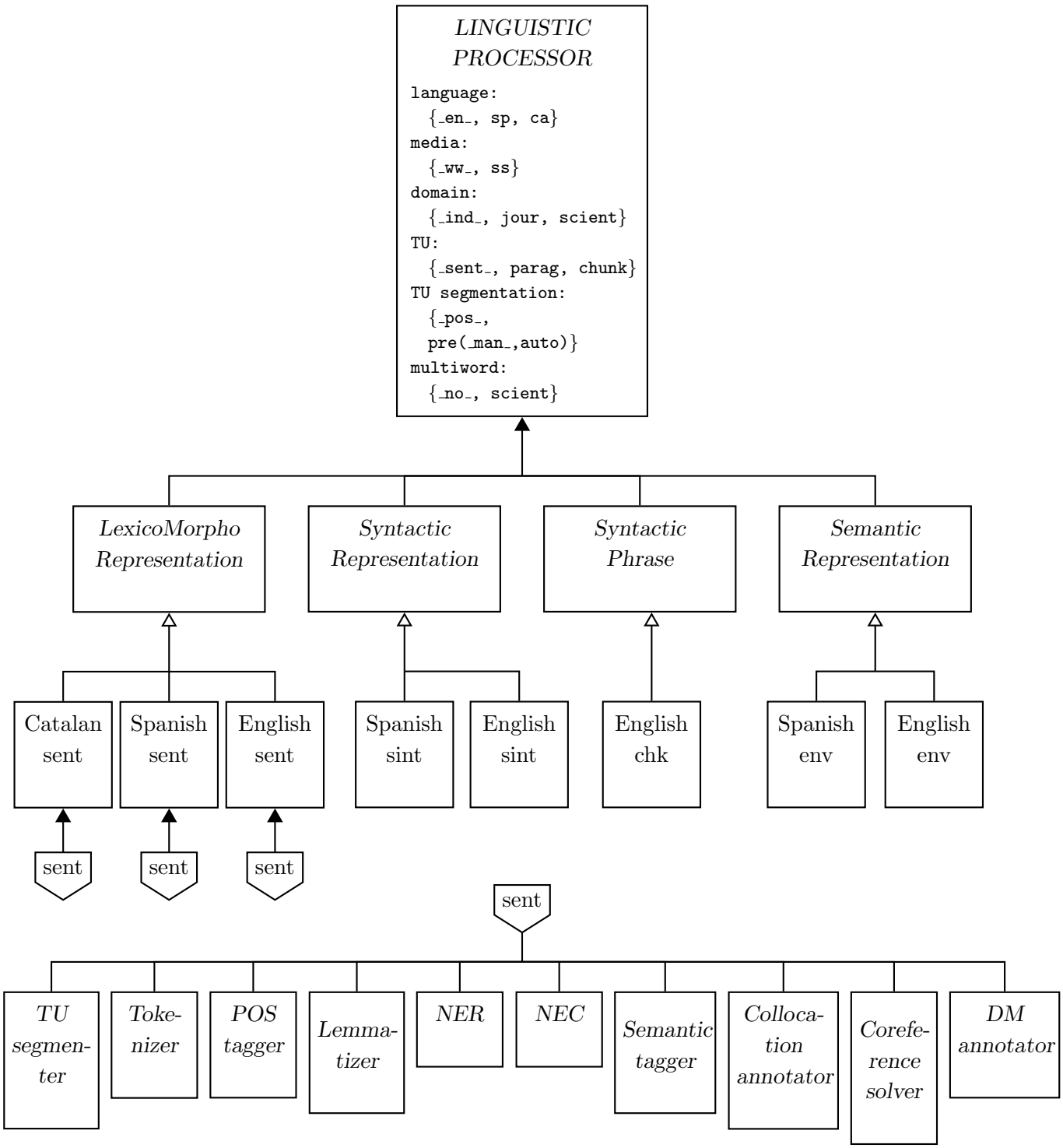

Figure 3.4: Linguistic Processor instantiated FEMsum component.

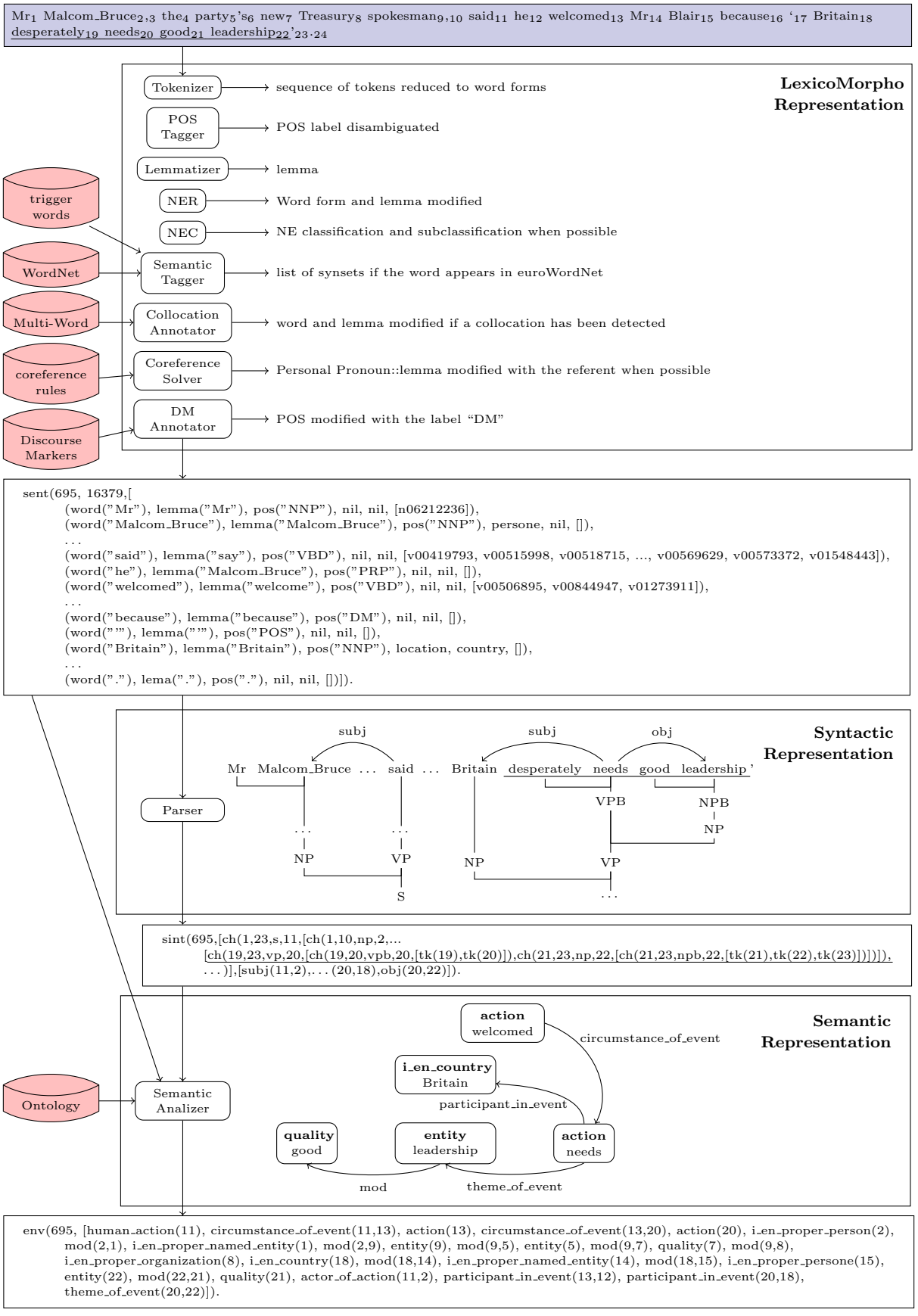

Figure 3.5: Linguistic Processor constituents and text representated by *sent*, *sint* and *env*.

synsets and, if available, derivational information. Details about each constituent output, when instantiated, are given in Figure 3.5:

  - The Tokenizer component produces sequence of tokens reduced to word forms.
  - The POS Tagger adds the information of the disambiguated POS label.
  - The lemmatizer adds the lemma of the word.
  - The NER modifies the word form and lemma field when a NE has been recognized.
  - The Named Entity Classification (NEC) gives a NE classification label when possible.
  - The Semantic Tagger produces a list of synsets if the word appears in WN or EWN.
  - The Collocation Annotator reagrup words and lemmas in order to create a multiword if a collocation has been detected.
  - The Coreference Solver substitutes the lemma of a Personal Pronoun with the referent when possible.
  - The Discourse Marker (DM) Annotator modifies the POS with the label "DM".

- The *Syntactic Representation*, *sint*, is composed by two lists, one recording the syntactic constituent structure (basically nominal, prepositional and verbal phrases) and the other collecting the information of dependencies between these constituents.

- The *Semantic Representation*, *env*, is a semantic-network-like representation of the semantic concepts (nodes) and the semantic relations (edges) holding between the different tokens in *sent*. Concepts and relation types belong to an ontology of about 100 semantic classes (like person, city, action, magnitude, etc.), and 25 relations between them (like time of event, actor of action, location of event, etc.). Both classes and relations are related by taxonomic links (see (Ferrés et al. 2005) and (Ferrés 2007) for details), allowing for inheritance. As can be observed in the example used in Figure 3.5, each term (token) of a sentence is identified by a number. Some of the tokens are assigned unary predicates: "Malcom Bruce" is a NE classified as a person, "Britain" is a country, and "needs" denotes and action. Binary predicates represent relations between tokens: in the example, "welcomed" is the action circumstance of the event "needs", "good" is the quality that modifies "leadership", being "good leadership" the theme of the event that reflects what is needed by the entity "Britain".

3.4.1 Catalan and Spanish LP

This section presents the specific NLP tools instantiated in the FEMsum approaches that can be used by the *CATALAN* and *SPANISH LP*.

- FreeLing[7] (Atserias et al. 2006), a parameterizable tool which performs tokenization, NERC (with classes of NEs like date, time, quantities, ratios, percentages, person, location, organization, others), morphological analysis, POS tagging, lemmatization, and shallow and dependency parsing.
- ABIONET (Carreras et al. 2002), a NERC on basic Message Understanding Conferences (MUC) categories (person, location, organization, others).
- EuroWordNet (Vossen 1998), used to obtain the list of synsets associated to each word, with no attempt to perform Word Sense Disambiguation (WSD), and, for each synset, a list of hyperonyms of each synset up to the top of the taxonomy, and the Top Concept Ontology class.
- Discourse Marker Annotator (Alonso 2005). The information stored in the DM lexicon[8] ((Alonso 2005), Appendix A) is used for identifying inter- and intra-sentential discourse segments and the discursive relations holding between them. Discourse segments are taken as TUs in the experiments reported in Section 4.1.2.

3.4.2 English LP

The specific NLP tools that can be used by the *ENGLISH LP* include:

- FreeLing.
- TnT (Brants 2000), a statistical POS tagger
- SVMTool, a SVM-based tagger [9](Giménez and Màrquez 2004).
- WordNet (Fellbaum 1998) lemmatizer 2.0.
- ABIONET
- (M. Surdeanu and Comelles 2005), a NERC for spontaneous speech.
- Alembic (Day et al. 1998), a NERC with MUC classes (used to complement ABIONET performance).
- WordNet, used to obtain the list of synsets and hypernyms.
- Discourse Marker Annotator ((Alonso 2005), Appendix A). The information stored in the DM lexicon is used to establish TU boundaries in the experiments reported in Section 5.3.

[7] `http://garraf.epsevg.upc.es/freeling/`
[8] `http://russell.famaf.unc.edu.ar/~laura/shallowdisc4summ/discmar/`
[9] `http://www.lsi.upc.edu/~nlp/SVMTool/`

- A modified version of Collin's parser which performs full parsing and robust detection of verbal predicate arguments (Ferrés et al. 2004).

- Dekang Lin's dependency parser, MINIPAR (MINIPAR 1998).

- YamCha chunker (Kudo and Matsumoto 2001).

3.5 Query Processor Component

The *QP* component is necessary for those query-focused FEMsum approaches that instantiate a *RID* component unable to process complex questions. As depicted in Figure 3.6, the *QP* component can be instantiated as the *SPLITTING QP*, presented in Section 3.5.1 or as the *GENERATING QP* in Section 3.5.2. The first one turns a complex question into a set of simpler ones by splitting the original text, while the latter requires a more complex linguistic process.

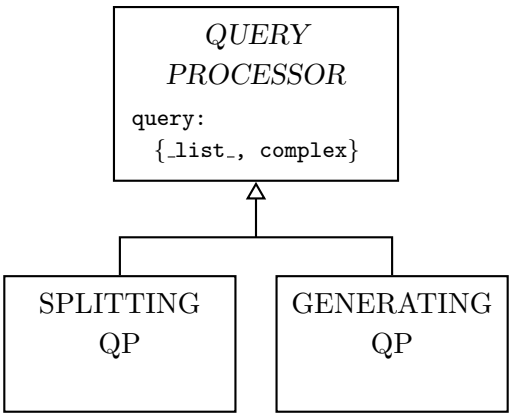

Figure 3.6: Query Processor instantiated FEMsum component.

Both *QP* instantiations where developed to deal with the query-focused MDS task proposed at DUC, where the user information need was expressed in terms of a title and a narrative. Figure 3.7 presents an example of a DUC 2007 Natural Language complex question.

The *SPLITTING QP* was first instantiated by *SEMsum* in the experiments reported in Section 6.1.3, and the *GENERATING QP* was instantiated by *QAsum* (see Section 6.1.2).

<topic>
<num> D0722E </num>
<title> US missile defense system </title>
<narr>
Discuss plans for a national missile defense system. Include information about system costs, treaty issues, and technical criticisms. Provide information about test results of the system.
</narr>
</topic>

Figure 3.7: An example of the information provided in one DUC 2007 topic.

Q1. discuss plan for a national missile defense system (US) .
Q2. include information about system cost , treaty issue , and technical criticism (US missile defense) .
Q3. provide information about test result of the system (US missile defense) .
Q4. include information about system cost (US missile defense) .
Q5. include information about system treaty issue (US missile defense) .
Q6. include information about system technical criticism (US missile defense) .

Figure 3.8: DUC 2007 complex natural language query processor output.

3.5.1 Splitting QP

The title and the narrative is first lemmatized, and then each sentence from the narrative is added to the output question set. In addition, all those sentences with the pattern $((POS_X)+, \textit{and } POS_X)$ or $(POS_X \textit{ or } POS_X)$ are split and also added to the output set. Finally, all those words from the title not present in the output question are added at the end. Figure 3.8 shows the *SPLITTING QP* output when the input is the complex question in Figure 3.7.

3.5.2 Generating QP

To create the output set of simpler questions, the title and the narrative of a topic are pre-processed to add linguistic information (as in Section 3.4). After that, each sentence from the

LOCATION	PERSON
Where is <*NE*>?	Who is <*NE*>?
	When <*NE*> was born?
ORGANIZATION	Where <*NE*> was born?
Where is <*NE*>?	When <*NE*> died?
Where is <*NE*> located?	Where <*NE*> died?
When was <*NE*> founded?	When <*NE*> lived?
Who is <*NE*> director?	Where <*NE*> lived?
OTHERS	ACTION
Where is <*NE*>?	Where <*chunkAction*>?
When was <*NE*>?	When <*chunkAction*>?
Who did <*NE*>?	Who <*chunkAction*>?

Table 3.4: Question Generation patterns by NE type.

narrative is considered as a question to be included in the set, as well as the ones produced by the splitting process described in the previous section.

The title of the topic, if available, is used to create a question following the pattern "What is <title>?". Moreover, questions are generated from NEs and actions occurring in the narrative. NE classification is taken into account to generate specific patterns (see Table 3.4). For instance, when a NE classified as PERSON appears in the narrative, questions as "Who is <NE>?" to "Where <NE> lived?", in Table 3.4, are generated.

To generate new questions, sentences in the narrative are previously processed by LP in order to obtain a *sent*, *sint*, and *env* representation; as described in Section 3.4, corresponding to the lexical, syntactic, and semantic levels.

Finally, to increase the accuracy and to avoid dispersion, as in *SPLITTING QP*, the title is added in round brackets at the end of each question.

3.6 Relevant Information Detector Component

The main function of the *RID* component is to provide a ranked set of n relevant TUs. The default value of n does not need to be fixed, but if fixed, the n best scoring TUs are taken into account. This section describes the *RID* components instantiated by the FEMsum approaches. Figure 3.9 shows the modules needed by each *RID* component.

The *LEXICAL CHAINER*, described in Section 3.6.1, is used by the *LC RID* component

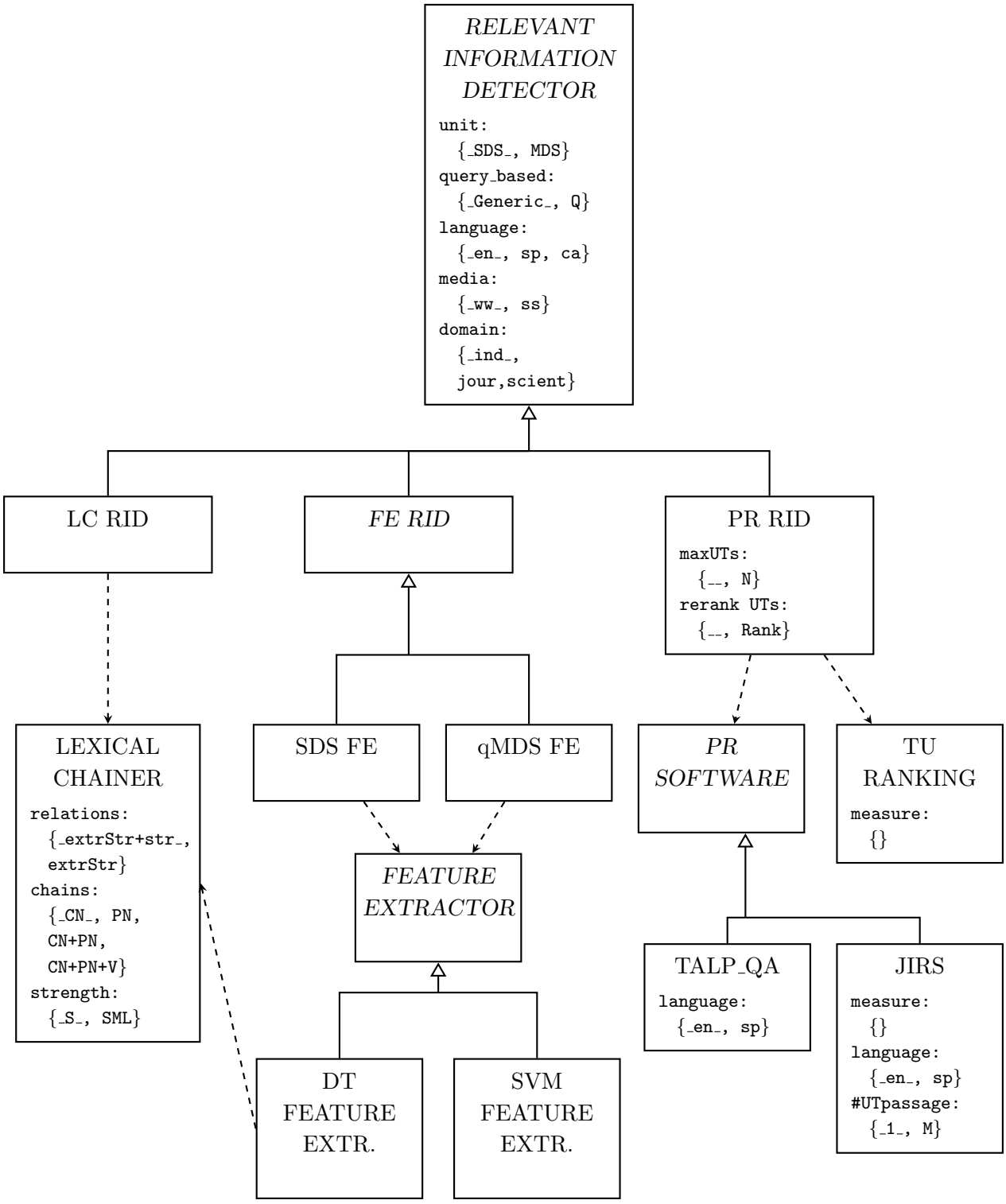

Figure 3.9: Relevant Information Detector instantiated FEMsum component.

instantiated by *LCsum* FEMsum approach. Moreover, the *LEXICAL CHAINER* is also used by the *DT FEATURE EXTR*, one of the two implementations of the *FEATURE EXTRACTOR*. This component is instantiated by both *FE RID* implementations, *SDS FE* and *qMDS FE*, presented in Section 3.6.2.

The *FE RID* component is instantiated by the *MLsum* FEMsum instantiation, but it is also used in the training process of the *Machine Learning (ML) CE* components described in Section 3.7.2. The *SDS FE*, implemented with the *DT FEATURE EXTR*, provides a set of features to represent documents for a generic SDS task, while the *qMDS FE*, implemented with the *SVM FEATURE EXTR*, is oriented to query-focused MDS. The first one is used in the training process of a Decision Tree (DT) TU classifier and the second one to train a Support Vector Machine (SVM) TU classifier.

Figure 3.9 also reflects the fact that the third type of *RID* component, *Passage Retrieval (PR) RID*, can be implemented with two different *PR SOFTWARE*: *TALP-QA PR* or *JIRS PR*. The first one has been instantiated by *QAsum* and the second one by *LEXsum* and *SEMsum* FEMsum instantiations. It can be observed that, optionally, in the *PR RID* a maximum number of relevant TUs can be fixed and a *TU RANKING* module can be used in addition to the *PR SOFTWARE* (see Section 3.6.3 for more details).

3.6.1 Lexical Chain based RID

Lexical Chains try to identify cohesion links between parts of text by identifying relations holding between their words. Two pieces of text are considered to be lexically related not only if they use the same words, but also if they use semantically related words. This is a way to obtain certain structure of a text based on the distribution of its content.

(Hasan 1984) establishes that *identity chains* contain terms that refer to the same object. They are created by pronominal cohesion, lexical repetition or instantial equivalence and are always text-bound, because the relation of coreference can be determined only in the context of a text. In contrast, *similarity chains* are not text-bound. Their elements are held together by semantic bonds obtained through a lexical resource outside the text. These bonds are supra-textual, with a language-wide validity. The two types of chains are important for cohesion analysis, however, one of the main advantages of similarity chains over identity ones is that their implementation is simpler, since they can be computed with no deep text understanding.

Lexical Chains provide a representation of text that has been widely used in a variety of IR and NLP tasks, from text segmentation (Morris and Hirst 1991; Hearst 1994) to WSD (Okumura and Honda 1994; Barzilay 1997; Galley and McKeown 2003); term weighting for IR tasks (Stairmand 1996), topic detection (Lin and Hovy 1997), detection of malapropisms (Hirst and St-Onge 1997), hypertext generation (Green 1997), detecting topic and sub-topic shifts in texts (Kozima 1993; Okumura and Honda 1994; Stairmand 1996; Hearst 1997; Ponte and Croft

1997; Reynar 1998; Kan et al. 1998; Beeferman et al. 1999; Choi 2000; Stokes 2004), analysis of the structure of texts to compute their similarity (Ellman 2000), PR (Mochizuki et al. 2000), QA (Moldovan and Novischi 2002) and AS (Barzilay 1997; Stokes et al. 2004). For AS, coreference chains have also been used (Bagga and Baldwin 1998; Baldwin and Morton 1998; Azzam et al. 1999). A detailed overview of approaches to lexical chain identification and use can be found in (Stokes 2004).

Figure 3.10 shows the different TUs of a Spanish agency news document. Those terms related with the concept "candidato"[10](see Figure 3.11) are emphasized, as well as those related with the concept "debate"[11](see Figure 3.12). These are two sets of lexically related words to be taken into account as lexical chain candidates when creating *similarity chains*. On the other hand, the NE "Vicente Fox" and the noun phrase "candidato del Partido Acción Nacional (PAN) de México"[12] are members of the same *identity chain* as "Fox". In case a personal pronoun is used with "Fox" as a referent, it will be also an identity chain candidate, what implies a previous step of coreference resolution.

The general procedure for constructing lexical chains usually follows three steps:

1. Select a set of candidate words. To detect chain candidates, the text is preprocessed with the *LP* component described in Section 3.4, taking into account the *sent* representation.
2. For each candidate word, find an appropriate chain relying on a relatedness criterion among members of the chains. Usually, relatedness of words is determined in terms of the distance of the path connecting them in WordNet (WN) (or EuroWordNet (EWN) for languages other than English).
3. If a chain is found, insert the word in the chain and update it accordingly.

Chain candidates are common nouns, proper nouns, named entities, definite noun phrases and pronouns. No previous WSD is performed. For each chain candidate, three kinds of relations are considered, as defined by (Barzilay 1997):

- **Extra-strong** between a word and its repetition.
- **Strong** between two words connected by a direct EWN relation.
- **Medium-strong** if the path length between the EWN synsets of the words is longer than one.

In contrast to (Stokes et al. 2004) and following Barzilay, our algorithm is non-greedy. The proper sense, or a reduction of the number of possible senses of ambiguous words, is determined

[10]candidate
[11]debate
[12]candidate of the National Action Party (NAP) of Mexico

El conservador Vicente Fox, candidato del Partido Acción Nacional (PAN) de México, cedió hoy ante sus rivales, el oficialista Francisco Labastida y el centroizquierdista Cuauhtémoc Cárdenas, en posponer para el próximo viernes el debate que estaba previsto para esta noche.

En un encuentro público en la casa de campaña de Cárdenas y frente a los representantes de los medios, los tres candidatos discutieron durante unas dos horas sus propuestas sobre el debate.

El candidato del PAN insistió reiteradamente en celebrar esta misma noche esta discusión, mientras que el candidato del Partido Revolucionario Institucional (PRI), Francisco Labastida, y Cuauhtémoc Cardenas, del Partido de la Revolución Democrática (PRD), pidieron posponerlo para el viernes a fin de garantizar las condiciones técnicas.

Cárdenas y Labastida calificaron de "superficial", "frívola", "caprichosa", "terquedad" y "ligereza", la insistencia de Fox en celebrar esta misma noche el debate, sin garantizar la neutralidad ni la capacidad de difusión a todos los medios.

En este minidebate, los candidatos evitaron ataques personales y se centraron en los puntos de procedimiento, el formato de la reunión, el tipo de moderadores y su papel, el sitio y la fecha.

Con la asistencia de más de un centenar de reporteros de diversos medios, los candidatos presidenciales presentaron sus propuestas y criticaron las de los contrarios.

Al concluir la reunión, en un discurso previamente escrito, Fox acusó a sus contrincantes de ponerse de acuerdo para boicotear el debate y reiteró su disposición a discutir abiertamente y negó ser el responsable de no cumplir con el acuerdo previsto desde marzo.

Cárdenas reiteró que no existían condiciones técnicas para celebrar el debate y "felicitó" a Fox por leer un discurso previamente elaborado dirigido a señalar la negativa de los candidatos del PRD y del PRI.

En esta discusión participó el presidente de la Cámara de la Industria de Radio y Televisión (CIRT), Joaquín Vargas, quien argumentó la necesidad de contar con 48 horas para preparar las condiciones técnicas en la mayoría de los medios de difusión, principalmente la televisión y la radio.

El debate previsto para hoy se canceló el domingo después de que Fox —quien según la mayoría de los sondeos supera en unos cinco puntos a Labastida— propuso a última hora la participación de tres personas que hicieran preguntas a los candidatos sobre temas específicos, pero la iniciativa la rechazaron sus adversarios.

El 25 de abril pasado, los seis aspirantes a la presidencia participaron en el primer debate y acordaron que para el segundo sólo acudirían los tres con más posibilidades de ganar las elecciones.

La mayor parte de los especialistas han destacado que las elecciones del 2 de julio serán las más reñidas y han señalado la posibilidad de un triunfo de la oposición en la historia de México.

candidato, debate Similarity Chains (terms related by EuroWorldNet).
Fox Named Entity Chain. sus Co-Reference Chain.

Figure 3.10: Terms related in an agency news document.

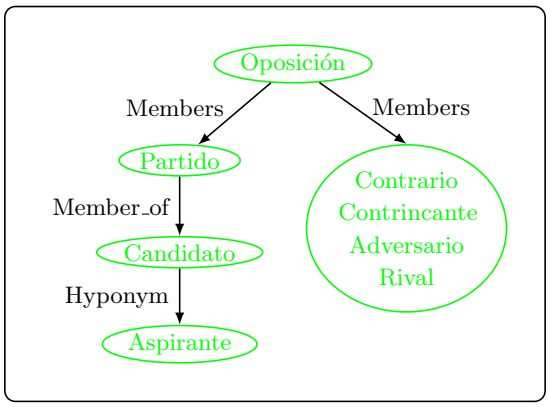

Figure 3.11: Terms related with the concept "candidato."

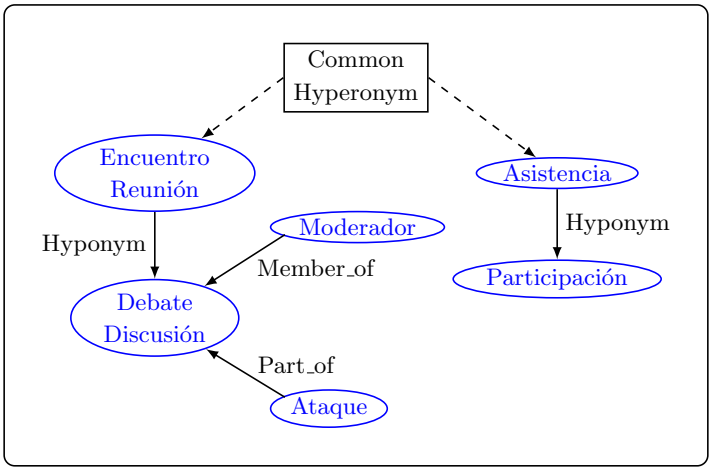

Figure 3.12: Terms related with the concept "debate."

when all words in the document have been processed. So, lexical chain identification performs a reductive WSD process as well. Once chains are identified, they are scored according to a number of heuristics: their length, the kind of relation between their words, their starting position within the text, etc. In our case, the measure of strength can be defined in a way slightly different from Barzilay's, chains are classified into *Strong*, *Medium* and *Light*, depending on their score:

$$\begin{aligned} \tau &= \mu_s + 2 \cdot \sigma_s \\ Strong &= \{c \mid score_c \geq \tau\} \\ Medium &= \{c \mid \tau > score_c \geq \tau/2\} \\ Light &= \{c \mid \tau/2 > score_c\} \end{aligned}$$

where μ_s and σ_s are the average of the scores of all the chains and the corresponding standard deviation. In contrast to other approaches where lexical chains are used, if necessary, we consider *Medium* and *Light* chains in addition to the typical strong ones. That is specially useful when dealing with spontaneous speech, due to the fact that *Strong* chains tend to provide a misrepresentation of the information in a text, because the distribution of the frequency of words is rather squewed, and only few strong chains are found (see details in Section 5.2.2).

As reflected in Figure 3.9, the *LEXICAL CHAINER* module can be parameterized to take into account different WN or EWN relations (**Extra-strong** and **Strong** relations or only **Extra-strong**). The second parameter indicates the type of chain to be produced: only common name chains (CN); only proper name chains (PN); CN and PN chains; and CN, PN and V (verb) chains. And the third one set up the strength measure (only *Strong* chains (S) or *Strong*, *Medium* and *Light* (SML)).

The *LEXICAL CHAINER* adds to each input TU the score of the lexical chains crossing it as well as the type of lexical chain (*Strong*, *Medium* or *Light*). A new field is added to the lexical representation of each term to indicate the kind and the score of the corresponding lexical chain.

3.6.2 Feature Extractor RID

The aim of this *FE RID* instantiation is to represent the text with a set of features. As reflected in Figure 3.9, two sets of features have been designed, one for generic SDS, *SDS FEATURE EXTR*, and the other for query-focused MDS, *qMDS FEATURE EXTR*. Besides its use in the *RID* component, the first set of features, *DT FEATURE EXTR*, was also used to train a DT TU classifier and the second one, *SVM FEATURE EXTR*, to train a SVM *TU CLASSIFIER* (see 3.7.2).

The *SDS Feature Extr* is instantiated as a *RID* component by *MLsum(HL)* (see Section 4.2.2), while *qMDS FEATURE EXTR* is the *RID* component of *MLsum(MDS,Q)* (see Section 6.1.4). The features extracted by each *FE RID* are detailed in the following sections.

SDS Feature Extractor

The *SDS FEATURE EXTR* component has been implemented as a *DT FEATURE EXTR* to participate in the DUC 2003 contest with the *MLsum(HL)* FEMsum approach. The input of the *DT FEATURE EXTR* is a set of sentences previously processed by the *LP* component in order to obtain the lexical representation described in Section 3.4, *sent*. The output consists in the same sentences, now represented as a set of features. As these features have to be used in a DT algorithm, continuous predictor values were discretized prior to learning.

The discretizing process requires the analysis of the distribution of the TUs of a set of documents similar to the ones to be processed in the future. As this component was first implemented to process DUC documents, the 6933 sentences from the 147 DUC 2001 documents of the corpus created by (Conroy et al. 2001) were used as training data.

To represent a single-document, this *RID* component computes two types of features. The first type extracts internal TU properties and the second type extracts similarities between the TU and the rest of the TUs from the input document. Both of them are described below.

TU internal features include:

- Two five-valued features to classify the TU into a qualitative length value: extra-short, short, mean, long and extra-long. The first feature measures the TU length in number of words, while the second takes into account the number of characters.
- A numeric value indicates the number of strong lexical chains crossing the TU. To compute this feature, the *LEXICAL CHAINER* described in the previous section is used.

TU similarity features with the rest of TUs include:

- The qualitative length relative to the other TUs of the same document (discretized into 5 intervals, denoting: extra-short, short, mean, long, extra-long TU).
- The TU position in document (discretized into 6 intervals).
- The number of TUs in the document with null bigram overlap with current TU.
- The number of TUs in the document with not null bigram overlap with current TU.
- Five numeric features to count the number of TUs in the document that have a degree of unigram overlap with the current TU within interval Xi–Yi; where i ranges from 1 to 5 (one interval for each feature).
- Five numeric features counting the number of TUs in the document with a cosine similarity with the current TU within interval Xi–Yi; where i ranges from 1 to 5 (one interval for feature).

- Five numeric features counting the number of TUs in the document with a weighted cosine similarity with the current TU within interval Xi–Yi; where i ranges from 1 to 5 (one interval for feature).

For the three last features, the training corpus distribution has been studied to establish the limits of five intervals, containing each interval a similar number of elements.

qMDS Feature Extractor

The *qMDS FEATURE EXTR* component has been implemented as a *SVM FEATURE EXTR* to deal with the query-focused MDS task proposed at DUC 2005.

The objective of the *SVM FEATURE EXTR* is to represent the TU content from the input set of documents with respect to the rest of TUs in the set and with respect to a query or a user information need. Each TU is previously processed using the *LP* component to be enriched with lexical, *sent*, syntactic, *sint*, and semantic, *env*, representations.

The features extracted by this *FE RID* component can be classified into three groups: those internal to the TUs (sentences in this case), those that capture a similarity metric between the sentence and the user need, and those that try to relate the cohesion between the sentence and all the other sentences in the same document or set of documents.

Internal features, those calculated from attributes of the sentences themselves, are the following:

- The position of the sentence inside its document.
- $\frac{1}{N_d}$, where N_d is the number of sentences in the document.
- $\frac{1}{N_c}$, where N_c is the number of sentences in the set of documents.
- Three boolean features indicating whether the sentence contains positive, negative or neutral discourse markers, respectively. For instance, *what's more* or *above all* are positive discourse markers, indicating relevance, while *for example* or *incidentally* indicate lack of relevance.
- Two boolean features indicating whether the sentence contains *right-directed* discourse markers (those that affect the relevance of the sentence fragment after the marker, such as *first of all*), or discourse markers affecting both the fragment at the right-hand side and at the left-hand side, such as *that's why* (Alonso 2005).
- Several boolean features that are evaluated to true if the sentence starts with or contains a particular word (*startsQuote, containsQuote, containsSayVerb*) or POS (*startsPersonalPronoun, startsConjunction, startsDemonstrative, startsDiscourseMarker*). This set of features is intended to detect not relevant sentences.

- The total number of NEs included in the sentence, and the number of NEs of each kind considered (people, organizations, locations and miscellaneous entities).
- Two features based on *SumBasic score* (Nenkova and Vanderwende 2005). This metric is based on the observation that high-frequency words in document clusters tend to occur as well in human summaries. Therefore, each sentence receives a weight equal to the average probability of its words in the cluster. The original SumBasic algorithm provides an iterative procedure that updates word probabilities as sentences are selected for the summary. However, as we are evaluating separate sentences but not selecting them for the summary yet, weights are not updated in our case.

  Two different scores are calculated, by estimating word probabilities using only the set of words in the current document, and using all the words in the cluster.

The following features try to capture the similarity between the sentence and the query:

- The percentage of word-stem overlap between each sentence and the query.
- Three boolean features indicating whether the sentence contains a subject, object or indirect object dependency in common with the query.
- The overlap between the *env* predicates in the sentence and those in the query.
- Two similarity metrics calculated by expanding the query words using the Google search engine, as described by (Alfonseca et al. 2006) (the two weight functions applied are χ^2 and TFIDF). A query is sent to Google search engine as a list of keywords. For each query, a maximum of one thousand document snippets are downloaded. Then, each snippet is modeled as a word vector of co-occurrence frequencies. For each one of the themes to be answered there is a separate vector that contains some relevant words. Next, to obtain representative co-occurrences, based on a χ^2 test performed against the British National Corpus, only words with χ^2 over 0 are considered to be included in the vector-space model of the main topic of the question. At this point, we have a vector-space model of the main topic of each one of the questions. Therefore, the cosine similarity is calculated between each of the sentences in the original documents and the model vectors.
- *Modified SumFocus score* (Vanderwende et al. 2006): this score extends the SumBasic score (mentioned above) in order to capture the similarity of a sentence to the query. Because SumBasic is already a feature, only the score obtained by estimating word frequencies from the topic description is included in this feature.

The third set of features tries to capture the relationships between a sentence and the remaining by computing four overlap measures. For each one, the mean, the median, the standard deviation and the histogram of the overlap distribution are calculated and included as features.

To compute the histogram, first we measure the % of elements shared by the analyzed sentence and the rest of sentences. The histogram contains four values [13] indicating the percentage of sentences overlapping with the characterized one.

- Word-stem overlap between the sentence and the other sentences in the same document.
- Word-stem overlap between the sentence and the other sentences in the same cluster.
- Synset overlap between the sentence and the other sentences in the same document.
- Synset overlap with other sentences in the same collection.

3.6.3 Passage Retrieval RID

In the *Passage Retrieval (PR) RID* component, the most relevant TUs are obtained by using a PR software. As depicted in Figure 3.9, the *PR RID* can be implemented using two different *PR SOFTWARE*, optionally a maximum number of relevant TU can be fixed and a *TU RANKING* module can be used in addition to the *PR SOFTWARE*. The following sections present the *PR SOFTWARE* instantiated by *QAsum*, and the one used by *LEXsum* and *SEMsum*, as well as the *TU RANKING* module used by *SEMsum(reRank)*.

PR SOFTWARE

Two *PR SOFTWARE* have been instantiated by several query-focused MDS FEMsum approaches:

TALP-QA PR is used by *QAsum*, the first query-focused FEMsum prototype, designed to participate in DUC 2005 (see the experiments reported in Section 6.1.2). The objective was to use part of the TALP-QA QA system. This system is in continuous evolution trying to adapt itself to the specific characteristics of Text REtrieval Conference (TREC) and Cross-Language Evaluation Forum (CLEF) contests. When *QAsum* was designed, the last TALP-QA English prototype had participated in TREC 2005 QA track (Ferrés et al. 2005). This system can process Spanish and English documents.

The information needed for the *TALP-QA PR* is basically lexical and syntactic. This implies that documents and user needs must have been previously processed by a *LP* component to obtain the *sent* and *sint* representations described in Section 3.4.

The main function of the *TALP-QA PR* is to extract small text passages that are likely to contain the correct answer. Document retrieval is performed using the *Lucene*[14] Information

[13] [0%,25%], [25%, 50%], [50%,75%], [75%, 100%]
[14] http://jakarta.apache.org/lucene

Retrieval system. Each keyword is assigned a score using a series of heuristics. For example, a proper noun is assigned a score higher than a common noun, the question focus word (e.g. "state" in the question "What state has the most Indians?") is assigned the lowest score, and stop words are removed. The passage retrieval algorithm uses a data-driven query relaxation technique: if too few passages are retrieved, the query is relaxed first by increasing the accepted keyword proximity and then by discarding the keywords with the lowest score. The contrary happens when too many passages are extracted.

JIRS PR (Gómez et al. 2005) is used by *LEXsum* and *SEMsum*. Both query-focused MDS FEMsum approaches instantiate Java Information Retrieval System (JIRS)[15] as a PR software to obtain the most relevant TUs with respect to the user need. The *JIRS PR* only needs lexical information. The input has to be previously tokenized and lemmatized. Additionally, the pronominal anaphora has been solved by a *LP* component to obtain the definitive *sent* representation (see Section 3.4).

The *JIRS PR* software can be instantiated to process Spanish or English documents and several models can be used to get passages with a high similarity between the largest n-grams of the question and the ones in the passage: simple n-gram model, term weight n-gram model, and distance n-gram model.

Apart from the TALP-QA PR or the JIRS PR other IR software can be integrated. IR-n (Llopis 2003) and an adaptation of (Rojas et al. 2005) has been analyzed. However, those IR systems has not been instantiated by any of the FEMsum approaches presented in this thesis.

TU RANKING

In addition to the *PR SOFTWARE*, *SEMsum(reRank)* and *SEMsum(reRank, Update)* use a TU Reranking algorithm to try to give answer to the multiple queries that can be included in a complex question.

To participate in DUC 2007, we implemented a sentence re-ranking algorithm using the DUC 2006 corpus (Copeck et al. 2006) to empirically tune JIRS options. Precision, Recall, and F-1 measures were used, giving preference to the options with best F-1 measure.

The following options were set:

- Retrieval Model. JIRS modes to get passages with a high similarity between the largest n-grams of the question and the ones in the passage are: simple n-gram model, term weight n-gram model, and distance n-gram model. The best retrieval model was the JIRS Distance model with the soft-Inverse Document Frequency (IDF) term weighting (distance of 0.1). In this model, the weight of a passage is computed using the larger n-gram structure of

[15] `http://jirs.dsic.upv.es/`

the question that can be found in the passage itself and the distances among the different n-grams of the question found in the passage.

- Number of sentences per passage. We experimented with configurations of 1, 3, and 5 sentences per passage, and we obtained the best results with the option of 1 sentence per passage.
- Number of total sentences to retrieve. We tested empirically that the best number of sentences to retrieve was between 100 and 120 sentences.
- Topic fields used to compose the JIRS questions. A retrieval mode that consists in attaching the title at the end of each narrative sentence to compose the queries has achieved better results than the one that uses narrative sentences alone.
- To filter "say" sentences. Better results were obtained when applying a filter to remove all the sentences that had a form of the verb say after a quoted expression. The other tested filters include: no filtering, filtering sentences with the verb "say", filtering sentences with "say" before a quoted expression, and filtering sentences with "say" and a quoted expression.

The designed sentence ranking algorithm uses as input the retrieved sentences for each query from JIRS and a threshold N that indicates the maximum number of final sentences to retrieve. Originally, a sentence pool containing all retrieved sentences is created after removing duplicates. Each sentence in the pool is scored adding the weight of all the passages in which it appears. Then, a refinement of the sentence's score is applied: if two or more sentences are consecutive in the original document their score is changed by the sum of their original scores.

At this point we want to obtain a balanced set of sentences for each query. For that purpose, a half of the final N sentences must be selected from the top ranked sentences of each query. If the number of queries is Q, we will obtain the N/(2*Q) top-ranked sentences for each query, using as a score the weights computed in the sentence pool.

Finally, the sentences not selected for each query are put in a common pool without adding repeated sentences. From this pool, the remaining half of N sentences are obtained by selecting the top-ranked ones, using the weights computed in the previous sentence pool.

3.7 Content Extractor Component

This section presents the different FEMsum *CE* components. As can be observed in Figure 3.13, this component has been instantiated as: *LC CE*, *ML CE*, *QA CE*, and *SEM CE*. Each instantiation is described in the following sections.

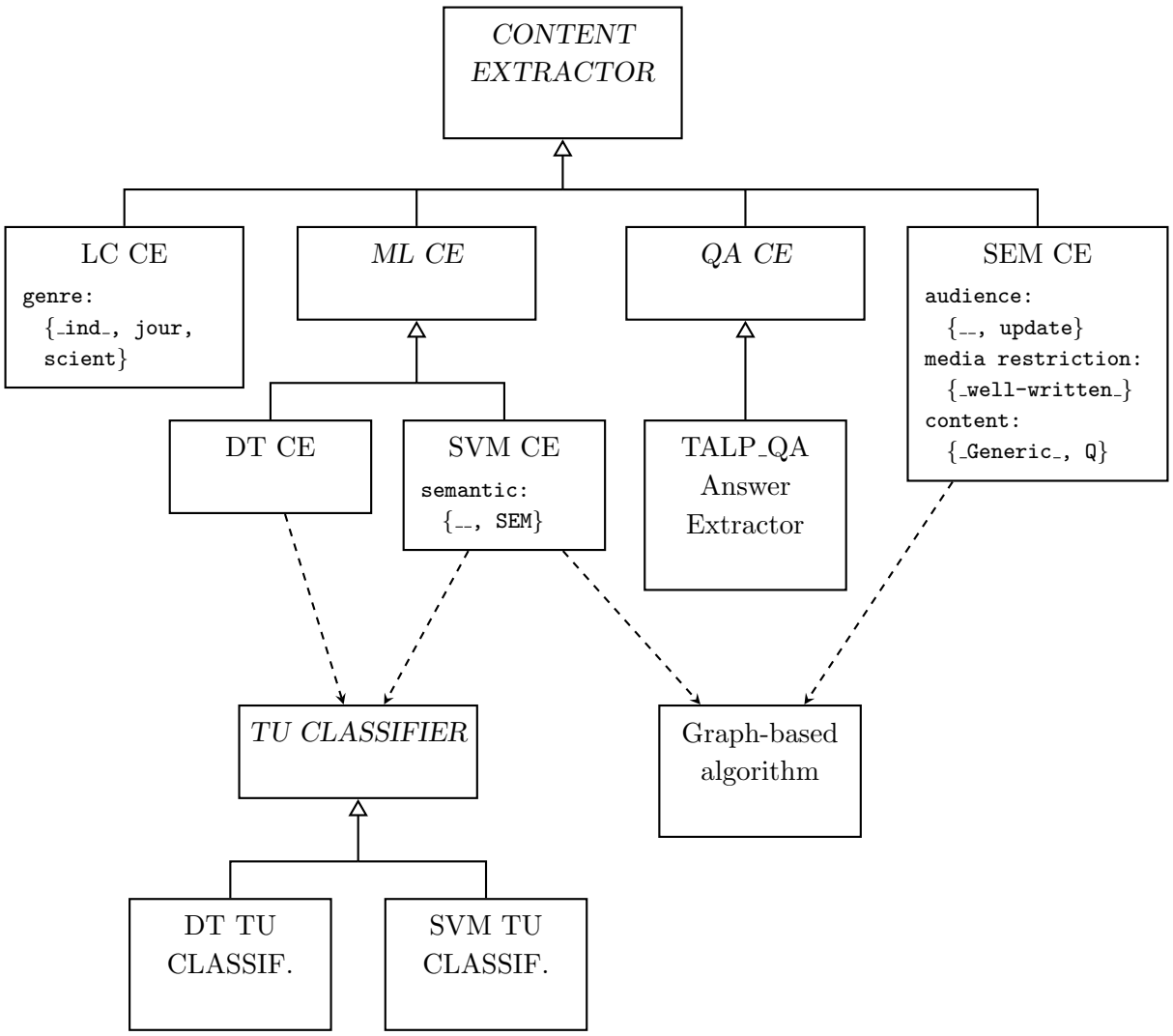

Figure 3.13: Content Extractor instantiated FEMsum component.

Section 3.7.1 presents the *LC CE*. The two different Machine Learning (ML) techniques used to implement the *ML CE* component are detailed in Section 3.7.2. The first one, *DT CE*, uses a *DT TU CLASSIF*, while the second one, *SVM CE*, uses a *SVM TU CLASSIF*.

As reflected in Figure 3.13, the *QA CE* is implemented with a component of a QA system, the TALP-QA Answer Extractor (see Section 3.7.3). The *SEM CE* and the *SVM CE* use the same *Graph-based algorithm* to process TUs semantic information, a component detailed in Section 3.7.4.

The input of the *CE* component are the set of document TUs previously processed by the *LP* and the *RID* component in order to obtain the text representation required by the instantiated *CE* component.

3.7.1 Lexical Chain based CE

The *LC CE* component extracts the set of summary candidate TUs from the input data. The input is the *sent* lexical representation of the TUs enriched with the score of the lexical chains crossing them as well as the type of lexical chain (*Strong*, *Medium* or *Light*).

Candidate TUs are scored by applying certain heuristic, weighting some aspects of lexical chains. Following (Barzilay 1997)'s work, two heuristic schemata have been implemented:

- *Heuristic 1* ranks as most relevant the first TU crossed by a *Strong* chain.
- *Heuristic 2* ranks highest the TU crossed by the maximum of *Strong* chains.

While to exploit the pyramidal organization of newspaper articles, in the journalistic genre *Heuristic 1* is applied by default. In the scientific domain *Heuristic 2* is applied by default.

3.7.2 Machine Learning based CE

This *CE* component envisions the extraction of important TUs as a classification problem, where a sentence TU is either apt or not suitable for inclusion in a summary. The two implementations of the *ML CE* are described below. The first implementation, the *DT CE*, uses a binary *TU CLASSIFIER*, *DT TU CLASSIF*. The second one, the *SVM CE*, uses a SVM trained to rank candidate sentences in order of relevance, *SVM TU CLASSIF*. As both methods imply supervised learning a training corpus is required.

Decision Tree CE

The *DT CE* is implemented by a *TU CLASSIFIER*, the *DT TU CLASSIF*. The objective of the *DT TU CLASSIF* is to classify TUs as apt to be summary candidates or not, by taking into ac-

count other types of relevant information different from lexical dependency. In the implemented classifier, this is done by applying Decision Tree (DT).

In the training corpus each of the TUs have been previously classified as belonging to a summary of the text or not, so that the fact that a TU belongs to a summary can be learned as a combination of relevant features.

The training corpus consisted of a set of 147 documents with human built extracts, obtained from the data in DUC 2001. Extractive summaries, covering approximately half of the original non-extractive summaries distributed as training data, were contributed by (Conroy et al. 2001). A total of 6,933 sentences were used as examples for training.

Each sentence in this training corpus was represented by using the features presented in Section 3.6.2 plus the additional feature of belonging or not to the extract (the classification task is binary). The learning algorithm used was C4.5.

This *CE* component is instantiated by *MLsum(HL)*. The performance of this FEMsum approach, when producing 10-word generic SDS, is analyzed in Section 4.2.

Support Vector Machine CE

The *SVM CE* combines the ranking given by *SVM TU CLASSIF* with the *SEM graph-based algorithm*. As detailed in Section 3.7.4, the *SEM graph-based algorithm* uses semantic similarities between TUs to avoid redundancy and to improve cohesion.

The input to the *SVM TU CLASSIF* are TUs represented by the three groups of features described in Section 3.6.2. These three types of features are: those extracted from the sentences, those that capture a similarity metric between the sentence and the topic description, and those that try to identify cohesive properties between a sentence and all the other sentences in the same document or set of documents.

To train the Support Vector Machine (SVM), we built a training corpus using the DUC 2006 dataset, including topic descriptions, document clusters, peer and manual summaries, and the Pyramid evaluations as annotated during the DUC-2006 manual evaluation. From all these data, the training set is generated in the following way: Sentences in the original documents are matched with sentences in the summaries as proposed by (Copeck and Szpakowicz 2005). With this method a certain number of the sentences in a collection of source documents are characterized according to the Pyramid measure on their suitability to be in a summary of the collection topic. As detailed in Section 2.3, a set of Summary Content Unit (SCU)s and the manual summaries on which it is based together constitute a Pyramid. All document sentences that match a summary sentence containing at least one SCU are extracted and considered as a positive example. Each sentence is represented with the features detailed in Section 3.6.2. Thus, the training set contains only positive examples.

In order to train a traditional SVM, both positive and negative examples are necessary. A possible procedure to train on just positive instances is a One-Class Support Vector Machine (OCSVM) (Manevitz and Yousef 2001), that calculates a boundary around positive instances. However, according to (Yu et al. 2002), OCSVMs are prone to underfitting or overfitting when data is scant (which is the case here), and a simple iterative procedure called Mapping-Convergence (MC) algorithm can greatly outperform OCSVM (see pseudocode in Figure 3.14). It starts by identifying a small set of instances that are very dissimilar to positive examples, called "strong negatives". Next, at each iteration, a new SVM h'_i is trained using the original positive examples, and negative examples found so far. The set of negative instances is then extended with the unlabeled instances classified as negative by h'_i.

Input: positive examples, POS, unlabeled examples U
Output: hypothesis at each iteration $h'_1, h'_2, ..., h'_k$

1. Train h to identify "strong negatives" in U:
   N_1 := examples from U classified as negative by h
   P_1 := examples from U classified as positive by h
2. Set $NEG := \emptyset$ and $i := 1$
3. Loop until $N_i = \emptyset$,
   3.1. $NEG := NEG \cup N_i$
   3.2. Train h'_i from POS and NEG
   3.3. Classify P_i by h'_i:
        N_{i+1} = examples from P_i classified as negative
        P_{i+1} = examples from P_i classified as positive
5. Return $\{h'_1, h'_2, ..., h'_k\}$

Figure 3.14: Mapping-Convergence algorithm.

3.7.3 Question & Answering System based CE

The QA CE is implemented by some of the modules of the TALP-QA system (Ferrés et al. 2005). This component is instantiated by the *QAsum* (see Section 6.1.2).

The TALP-QA module used to extract candidate answer TUs is the Answer Extractor. The input TU information required by the TALP-QA Answer Extractor is: lexical (*sent*), syntactic (*sint*) and semantic (*env*).

This *QA CE* implementation consists of two tasks performed in sequence: Candidate Extraction and Answer Selection. In the first component, all candidate answers, those TUs containing at least one component of the expected answer type, are extracted from the highest scoring sentences of the selected passages. The Question Type has been previously extracted from the question text. Part of the actual TALP-QA categories (see Figure 3.15) were first used in TREC 2003 (Massot et al. 2003). In the second component, the best answer is chosen from the set of candidates.

```
abreviation          abreviation_expansion
definee              definition
event_related_to     feature_of_person
howlong_event        howlong_object
howmany_objects      howmany_people
howmuch_action       non_human_actor_of_action
subclass_of          synonymous
theme_of_event       translation
when_action          when_begins
when_person_died     where_action
where_location       where_organization
where_person_died    where_quality
who_action           who_person_quality
```

Figure 3.15: TALP-QA Question Types.

3.7.4 Semantic information based CE

The aim of this component is to use semantic similarities between TUs as input of an iterative graph-based algorithm to avoid redundancy and obtain a cohesioned text.

The *SEM CE* instantiation is divided into two modules, the Similarity Matrix Generator (SMG) and the Candidates Selector (CS). As seen in Figure 3.16, this instantiation requires document TUs enriched with the *env* semantic representation. The SMG computes lexico-semantic similarities between TUs. Then, the CS uses these similarities to prevent redundancy and to propose the most appropriate TUs to be part of the summary. Next, we describe those two components.

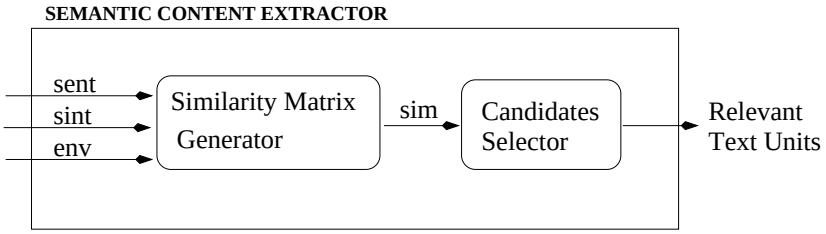

Figure 3.16: Semantic Content Extractor modules

Similarity Matrix Generator

SEMsum instantiations use similarity or distance between TUs (see Section 6.1.3 and 6.1.5). The goal of the SMG is to generate a similarity matrix between candidate TUs. The similarity matrix *sim* is a real valued $n * n$ matrix where n is the number of TUs and $sim(i, j)$ is the similarity between TUs i and j. Values range from 0 (absolutely dissimilar) to 1 (identical TUs). To obtain *sim*, each TU *env* is transformed into a labeled directed graph representation with nodes assigned to positions in the TU, labeled with the corresponding token, and edges to predicates (a dummy node, 0, is used for representing unary predicates). Only unary and binary predicates are used. Figure 3.17 is the graph representation of the *env* in Figure 3.18.

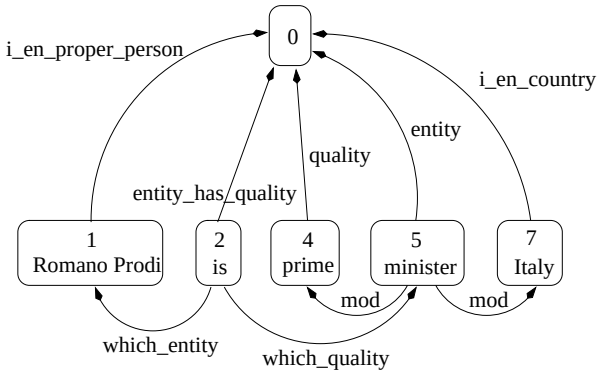

Figure 3.17: Example of the graph of an *env* representation

Over this representation, several lexico-semantic similarity measures between TUs have been built. Each measure combines two components:

- The lexical component considers the set of common tokens, i.e. those occurring in both TUs. The size of this set and the strength of the compatibility links between its members

Romano_Prodi$_1$ is$_2$ the$_3$ prime$_4$ minister$_5$ of$_6$ Italy$_7$

i_en_proper_person(1)	entity_has_quality(2)	quality(4)
entity(5)	i_en_country(7)	mod(5,7)
which_entity(2,1)	which_quality(2,5)	mod(5,4)

Figure 3.18: Environment (*env*) representation of a sentence.

are used for defining the measure. A flexible way of measuring token-level compatibility has been empirically set, ranging from word-form identity, lemma identity, overlap of EWN synsets, and approximate string match between NEs.

Maximum compatibility (1) is assigned to identical TUs; lemma identity (e.g. minister vs. ministers) has a score of 0.8; synonymy has a score computed taking into account EWN overlap. As no WSD is performed the scores depend on the number of overlapping synsets and the degree of polysemy of both TUs. For instance, "Romano Prodi" is lexically compatible with "R. Prodi" with a score of 0.5 and with "Prodi" with a score of 0.4. The same score is assigned to action-action and location-gentile such as the case of "Italy" and "Italian", which are compatible with a score of 0.7. A minimum compatibility threshold has been defined in order to limit the number of compatible TUs. For our purposes, the threshold has been empirically set to 0.6.

- The semantic component computed over the subgraphs corresponding to the set of lexically compatible nodes. Four different measures have been defined:
  - Strict overlap of unary predicates.
  - Strict overlap of binary predicates.
  - Loose overlap of unary predicates.
  - Loose overlap of binary predicates.

The loose versions allow a relaxed match of predicates by climbing up in the ontology of predicates described in (Ferrés 2007), e.g. provided that A and B are lexically compatible, i_en_city(A) can match i_en_proper_place(B), location(B) or entity(B). Loose overlap produces a penalty on the score.

Candidate Selector

In order to select summary candidates, three criteria are taken into account: relevance (with respect to the query or any other previously established criteria), density and antiredundancy.

Input: *Sim* be the similarity matrix,
Candidates a list of candidate TUs,
Output: *Summary* an ordered list of TUs to be included in the summary.

1. Set *Candidates* to the list provided by the *RID* component.
2. Set *Summary* to the empty list.
3. Set *Sim* to the matrix containing the similarity values between members from *Candidates* provided by the SMG.
4. Compute for each candidate in *Candidates* a score that takes into account the initial relevance score and the values in *Sim*. The score used is based on PageRank, as used by (Mihalcea and Tarau 2005), but without making the distinction between input and output links.
5. Sort *Candidates* by this score.
6. Append the highest scoring candidate (the head of the list) to the *Summary* and remove it from *Candidates*.
7. In order to prevent content redundancy, the S% most similar TUs to the selected one (using *Sim*) are removed as well from *Candidates* and the R% least scored are also removed from *Candidates* to reduce the search space (1,5% is the default value of both thresholds: S and R).
8. If *Candidates* is not empty go to 4, otherwise exit.

Figure 3.19: Candidates Selector procedure.

The general CS procedure is described in Figure 3.19. However, to produce update (or just-the-news) summaries, the *SEMsum(Update)* approach instantiates this module with a modified antiredundancy process. A first *SEMsum(Update)* prototype was designed to participate in the DUC 2007 update pilot task (see experiments reported in Section 6.2).

The process to produce just-the-news summaries is performed in N iterations, being N the number of set of documents to be summarized. The set of documents are supposed to be ordered chronologically. The initial set of sentences are processed in a first iteration following the general CS procedure. The rest of iterations follow a new procedure assuming that a previous set of sentences is already know by the user and this set is supposed to be less relevant in the production of the actual summary.

In addition, after the first iteration an additional antiredundancy step is performed to prevent the duplication of information. In this process sentences having a high overlapping with the content of previous summaries are removed. To maintain a minimum number of candidate sentences for performing the *CE* process two parameters have been defined: the minimum number of sentences to be selected from each set and a relative threshold defining the percentage of sentences provided by *RID* to be selected. These parameters have been empirically set to

10 and 0.5. The minimum number of sentences for *CE* is set to the maximum of these two thresholds. The antiredundancy process, thus, removes the redundant sentences, according to its own threshold, but leaving at least this minimum. The selection process is the same used as in the general process.

3.8 Summary Composer Component

The input to the *SC* component is a set of TUs either provided by the *RID* or the *CE* component. Each TU is scored by relevance, so the input is a list of ranked TUs. As represented in Figure 3.20, the length in words or the compression degree of the final summary can be set as parameters.

This section describes the two FEMsum *SC* components: the *BASIC SC* and the *COMPLEX SC*. The *BASIC SC* consists in adding the input TUs to the final summary text until reaching the desired length. So, summary content is selected from a list of TUs ordered by relevance, until the desired summary size is achieved. The *COMPLEX SC* is detailed in next section.

3.8.1 Complex SC

As reflected in Figure 3.20, in addition to the *BASIC SC*, the *COMPLEX SC* can instantiate one or more of the modules described in this section: the *Sentence Compressor*, the *Chunk Extractor*, the *Antiredundancy* or the *Reordering*.

The main difference among the instantiated *COMPLEX SC* is the granularity of the units of the original text to be included in the final summary. For instance, to produce long summaries from well-written text, *QAsum*, *LEXsum* and *SEMsum* use as TUs complete sentences extracted from some document of the collection to be summarized. It is possible that TUs suffer some small changes, that is the case when the anaphora was solved by the *LP* component (see DUC 2005 participation (Fuentes et al. 2005) as an example in Section 6.1.2). Instead of using complete sentences, *LCsum(DiscStr)*, *LCsum(PostSeg)* and *MLsum(HL)* use chunks of words or clauses.

Sentence Compression

As reflected in Figure 3.20, the *Sentence Compression* component has been implemented as the *Rhetorical Argumentative Compressor* and as the *Full Parsing Compressor*, both detailed in this section. The first one requires partial parsing, while full parsing of the TUs is required by the second one. The *Rethorical Argumentative Compressor* has been instantiated by *LCsum(DiscStr)*, analyzed in Section 4.1.2, and the *Full Parsing Compressor* by *MLsum(HL)* (see Section 4.2).

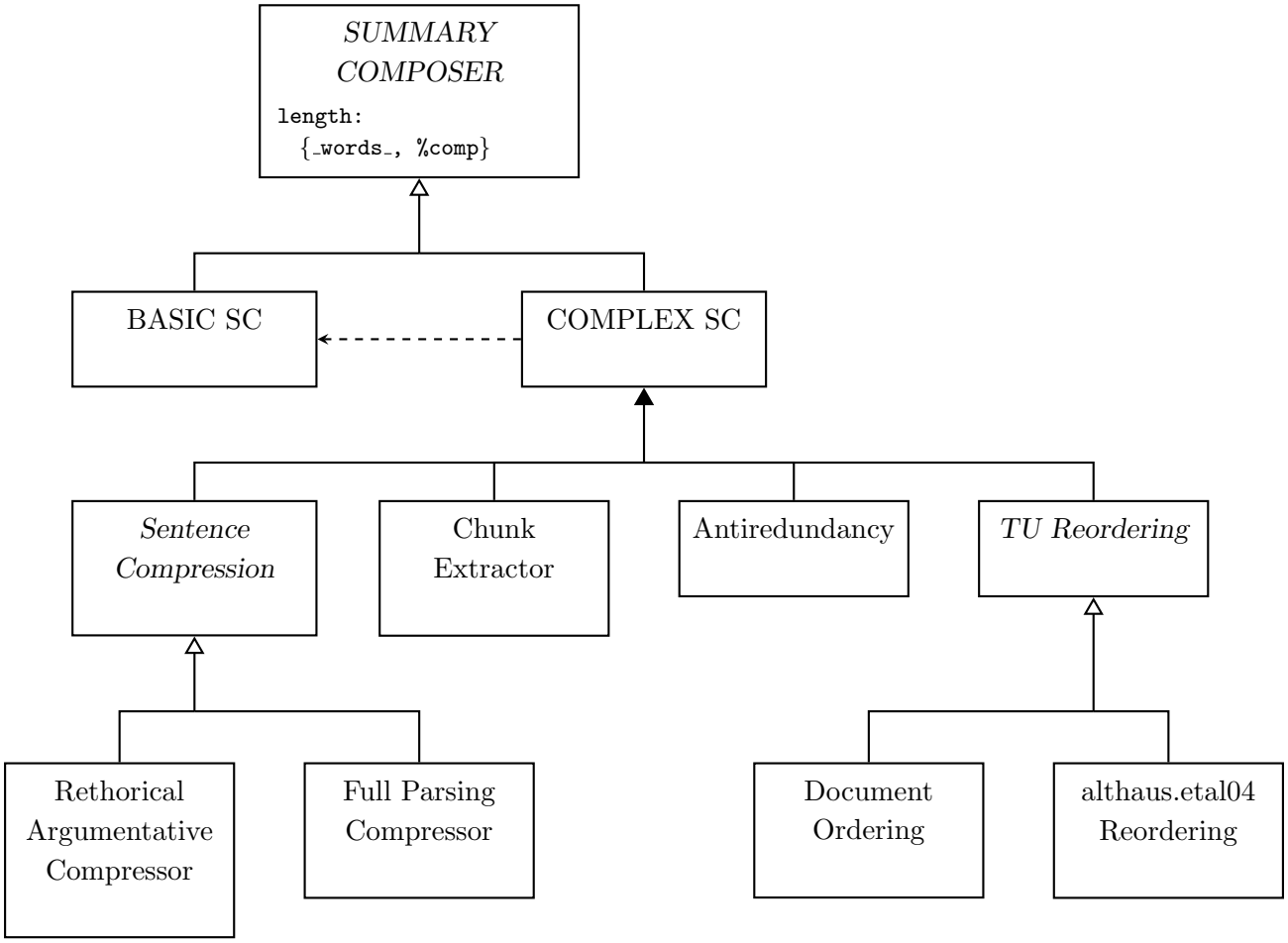

Figure 3.20: Summary Composer instantiated FEMsum component.

Rhetorical Argumentative Compressor

Following the approach of Marcu (Marcu 1997c), a partial representation of discourse structure was obtained by means of the information associated to a DM lexicon for English, Spanish and Catalan (see (Alonso 2005) Appendix A). In this lexicon, discourse markers are described in four dimensions: matter, argumentation, structure and syntax. The information stored in this lexicon was used for identifying inter- and intra-sentential discourse segments and the discursive relations holding between them.

Two combinations of the descriptive features of discourse markers were exploited:

rhetorical *nucleus-satellite* relations were identified by the combination of *matter* and *structure* dimensions of discourse markers. This rhetorical information yields a hierarchical structure of text, so that satellites are subordinated to nuclei and they are accordingly considered less relevant.

argumentative segments were tagged with their contribution to the progression of the argumentation via the information in the dimensions of *argumentation* and *structure* of discourse markers.

These features provide information about the structure of discourse that interacts with the information provided by lexical chains, see Section 4.1.2 for more details.

Full Parsing Compressor

This module applies compression rules to the input TU until achieving the targeted summary length, trying to preserve informativity. The full parsing of the TU has been previously performed by the *LP* component. Compression rules proceed as follows:

1. Find the main verb(s).
2. Take syntactically required arguments of main verb(s): subject and objects, but not lexically required ones, like collocative or semantic arguments.
3. Take complements of main verb(s) that were necessary from the point of view of truth value, for example negative particles.
4. Take complements of verbal arguments that may specify their truth value, like lexical modifiers.
5. Take discursively salient sentence constituents, namely, adjuncts marked by a discursive particle signaling relevance.
6. Fulfill well-formedness requirements.
7. Discard unused text.

Chunk Extractor

Taking into account the lexical chains found by the *LC RID* component, this module extracts windows of n contiguous words (chunks) to form a summary of the targeted size. This *SC* instantiation is used in the *LCsum(PostSeg)* approach.

Chunks are included in the summary using a priority ranking function that tries to capture both relevance and well-formedness. First, the algorithm selects the chunk with highest priority that covers the first occurrence of the highest scoring chain. This policy tries to capture the intuition that the first occurrence of a relevant word introduces the corresponding concept. In the following steps, the algorithm selects the following candidate chunk with highest priority whose content does not overlap with that of the already selected chunks, until the number of selected words is equal or greater than the targeted summary size.

Chunks are ranked by the following criteria (from most to least discriminating):

1. Least internal repetitions (items belonging to the same chain inside a chunk).
2. Highest score summing all covered chains.
3. Highest number of strong lexical chains.
4. Highest number of new chains (not present in already selected chunks).
5. Lowest collocation breakage (sum of the weight, statistically determined with χ^2, of all the collocations the chunk boundaries break).
6. Lowest number of violations on the conditions of well-formedness of chunks: restrictions on the POS of the starting and final words (it can not start or end with a DM or a conjuction; it can not end with a determinant or a preposition), breakage of syntactic chunks (as in item **1a.**).
7. Highest rhetorical relevance (sum of the rhetorical relevance of all words in the chunk, determined by the presence of discourse markers).
8. Highest size suitability (1 for chunks whose addition would give a summary with exactly the desired size, -1 for those whose addition would leave a gap smaller than the minimum chunk size).
9. Earliest Starting Document Position.
10. Earliest Ending Document Position.

These criteria are applied in a sequential order trying to determine the priority of the chunks in a several step strategy: criteria 1 to 4 try to determine regions of relevance within the

document, finding a set of chunks covering relevant text fragments. Criteria 5 and 6 are intended to refine the selection and to find, among the set of relevant chunks, those which are also syntactically better-formed. Criteria 7 tries to exploit text coherence. The last three criteria are aimed to break ties between any remaining candidates, taking into account chunk size and position.

When one chunk is added to the summary, the score of the lexical chains occurring in the chunk is reduced to its half. As a result, the scores used in criterion 2 are readjusted for the following calculation of segments with highest priority, making it more robust against the redundancy of spontaneous speech.

Antiredundancy

This component is used by the *QAsum* instantiation to take into account the information already included in the summary (see details in Section 6.1.2).

The final summary content is selected from the set of candidate TUs, sentences in this case, by applying the following greedy algorithm:

- Firstly, sentences containing answers to the generated questions are considered, sorted by their score.
- Then, the rest of sentences are considered, giving priority at each step to those sentences that precede or follow a previously selected one.

At each step, the redundancy of the candidate sentence with respect to the previously selected ones is measured, taking into account the *env* semantic representation of all sentences. If this redundancy exceeds a threshold, the sentence is discarded. The redundancy measure currently used is the fraction of *env* predicates of the candidate sentence not present in the *env*s of the previously selected ones, with respect to the total size of the *env* of the candidate. The threshold has been set to 0.5.

This redundancy measure may be modified by another factor, according to the specificity[16] of the desired summary: in the case the summary is asked to be particularly specific, the similarity between NEs in the sentences is also taken into account. When the summary is required to be generic, this similarity is not considered in the measure.

Sentences are added until the desired summary size is reached. If all retrieved sentences have been considered and the size has not been reached, sentences from a back-off summary, generated using lexical chain information, are incorporated into the summary to complete it. To create this back-off *LC* summary, the lexical chains from each document are previously computed using

[16]DUC 2005 task asked for two different types of summaries: specific and generic

the module described in Section 3.6.1. After that, the first sentence of each document crossed by a strong lexical chain is taken, until the desired size is reached.

Reordering

The default TU reordering algorithm used by *SEMsum* consists in selecting the summary TUs by relevance until reaching the desired summary size. For each selected TU, in our experiments sentence, it is checked whether the previous sentence in the original document is also a candidate, in that case, both are added to the summary in the order they appear in the original document. But any other algorithm could be used, for instance ordering TUs by increasing date of the original document if available or, as in the case of *SEMsum(Update)*, using the (Althaus et al. 2004) reordering algorithm (see Section 6.2).

The (Althaus et al. 2004) algorithm computes optimal locally coherent discourses, and approaches the discourse ordering problem as an instance of the Travelling Salesman Problem and solves this known NP-complete problem efficiently in cases similar to ours using a branch-and-cut algorithm based on linear programming.

For reordering TUs the algorithm starts with a set of selected TUs, each one owning a score. The space of the possible solutions is composed by different orderings of this set. So if the size of the set is n there are permutations $(n) = n!$ different states. Each state consists of a sequence of TUs: u_1, u_2, ..., u_n.

The global coherence of the state can be measured in terms of the coherence of the first TU, u_1, and in terms of the coherence of the transitions from this first TU, u_1, to u_2, from u_2 to u_3 and so on. We can define two costs: the *initial cost*, i.e. the cost of beginning with a specific TU, u_1, and the *transition cost*, i.e. the cost of following u_i by u_j. The objective is to get the sequence u_1, u_2, ..., u_n that minimize the following expression:

$$initial\_cost(u_1) = \prod_{i=2}^{n} transition\_cost(u_{i-1}, u_i) \tag{3.1}$$

So we had to provide the costs of transitions between units (i.e. the cost assigned in terms of lack of cohesion when a sentence i is followed by a sentence j. We have computed such costs as the inverse of similarities between the corresponding sentences. The system needs too a cost of initial position, i.e. the cost of placing sentence i in the first place of the summary. We have used in this case the inverse of the score assigned to each sentence.

3.9 Implementation FEMsum framework

Once the best configuration for each approach has been studied, the FEMsum instantiation can be integrated in the general platform for NLP first presented in (Gonzàlez 2004). This platform uses a client/server architecture. Communication between servers and clients is managed by a specialized process called MetaServer. The task of the MetaServer is that depicted in Figure 3.21. This process is responsible for: finding the proper server given a request from a client; activating the server if there is none of its kind already running; coordinating the sharing of resources (which becomes sharing of servers); and deactivating servers which are no longer being used.

Figure 3.22 shows a Dialog between a Client that wants to use a *LP* server and the MetaServer. The data to process is stored in files in disk, and what is sent in the requests is the location of this data (file names, see line 6 and 7 in Figure 3.22). The amount of data to process in NLP applications can be quite big, so working with all data in memory is not always possible. Intermediate files can be used as traces of the application process.

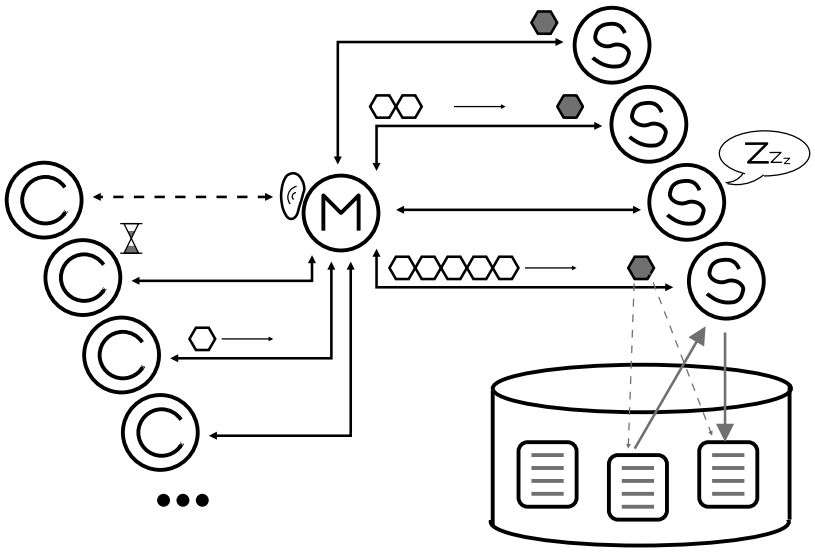

Figure 3.21: The MetaServer tasks.

This platform includes a set of servers for processing text depending on the language. They are reusable servers which apply generic NLP procedures to an input text, and which, in turn, depend on even simpler servers. For instance, as said in previous sections, all FEMsum instantiations need a linguistic processing server suitable for the type of document. The set of attributes used in the initialization of the *LP* server (media, domain, TU segmentation, MW level in Figure 3.23) will be taken into account to request the corresponding servers: Tokenizer, Tagger,

Client Request	C: REQ
Type of requested Service	C: TASK: LinguisticProcessor
Sevice Initialitation ...	C:
MetaServer Request OK	M: REQ OK 1
	M: PROC 1
Input data file location	C: INPUT: /tmp/question.lp
Output data file location	C: OUTPUT: /tmp/question.qp
	C:
	M: PROC WAIT
	M: PROC START
	M: PROC OK
	M:
	C: REL 1
Release Service Pipe ...	C:
Pipe Released OK	M: REL OK

Figure 3.22: Client MetaServer Dialog.

Lemmatizer, Named Entity Recognition and Classification, Semantic Annotator, Segmenter, ...

Every FEMsum instantiation component and component module is converted into a server in the MetaServer platform, and clients are applications asking servers to process text. Figure 3.23 shows an example of the first level of servers requested to summarize a scientific oral presentation in English.

The MetaServer has a configuration file where new services have to be included using the notation in Figure 3.24. Figure 3.25 is an example of the server definitions related with the *LP* component designed in Figure 3.4.

The MetaServer architecture allows an easy integration of previously implemented components, both if their source code is available or if only a binary is available. Details about this integration are given in the following section.

3.9.1 Integration from Source Code

If the source code of the application is available, the first step of the adaptation process consists in identifying those parts of the code that allocate shareable resources, those that process the input data and generate the output, and those that free the resources.

Then, a new main procedure has to be written, which starts allocating the resources and then waits for requests from the MetaServer. Every time a request is received, the data processing

```
my @servers;

push(@servers, $pms->newServer('LP','en',
                               PROGR   => 'LP',
                               MEDIA   => 'Voice',
                               DOMAIN  => 'Scien',
                               TUSEGM  => 'PostSeg'
                               MWLEVEL => 'Scien'));

push(@servers, $pms->newServer('RID','en'
                               PROGR   => 'LCRID'));

push(@servers, $pms->newServer('SC','en'
                               PROGR   => 'LCSC'
                               LENGTH  => '10'));
```

Figure 3.23: Servers to summarize a scientific oral presentation in English.

```
[ATTRIBUTE] == [Value]  : Required Attribute, and the value has
                          to be equal (idem for  <> != <= >= < >)

[ATTRIBUTE] ?= [Value]  : Default Attribute Value.

[ATTRIBUTE] :  [Value]  : Forced Attribute Value. Accepted only petitions
                          with this value. Default Value, if this attribute
                          does not appear in the request.
```

Figure 3.24: Condition Notation.

part of the program must be run, until the MetaServer requests the server for finalization, in which case the resources must be freed and the program ended.

The main advantage of this approach is that resource allocation is done only once for all possible requests, which can improve efficiency. The main difficulty lies in the identification of relevant parts of the source code and the rewriting of the main program. However, often only outer levels of the code need to be examined.

```
<package id="esca-preprocess-1.5" name="esca-preprocess" version="1.5">

<description>
A PreProcessing pipeline for Spanish and Catalan
<person name="Edgar Gonzalez i Pellicer" task="packager"/>
</description>

<dependencies>
<package name="freeling"/>
<package name="abionet"/>
<service>
    <feature name="TASK" value="Semantic Annotator"/>
    <feature name="LANG" value="es"/>
</service>
<service>
    <feature name="TASK" value="Semantic Annotator"/>
    <feature name="LANG" value="ca"/>
</service>
</dependencies>

<!-- No properties -->

<services>
<!-- It is a preprocessor for Spanish and Catalan -->
<service>
<conditions>
<feature cond="TASK == LinguisticProcessor"/>
<feature cond="PROGR : esca-preprocess"/>
<or>
<feature cond="LANG == es"/>
<feature cond="LANG == ca"/>
</or>
</conditions>
<exec command="perl preProcess.pl localhost ${System::port} ${LANG}"/>
</service>
</services>

</package>
```

Figure 3.25: Configuration of a *LP* server used to process documents in Spanish or Catalan.

3.9.2 Integration from Binaries

If only a binary of the component is available, the only solution is to create a wrapper program. This program will simply wait for requests from the MetaServer. When it receives one, it will use the binary to start a new child process of the component, forwarding it the request parameters. It will then wait for the completion of the child process, to send the response back to the MetaServer, and lastly start waiting again for the next request.

As these processes will deal mostly with input/output and child process related tasks, scripting languages such as Perl are a sensible choice for their writing.

The main drawback of this approach is that, as a new child process has to be spawned for each request, we cannot fully benefit from the resource reuse and sharing capabilities offered by the MetaServer. For this reason, even if the same approach can also be used to integrate components for which the source code is available, adaptation of the source code is more advantageous in those cases to improve efficiency.

As an example of use, this approach has been used to successfully create servers for the TnT tagger (Brants 2000) and the Yamcha chunker (Kudo and Matsumoto 2001).

Given that both the new main programs and the wrappers will usually involve a series of common tasks (such as managing communication with the MetaServer, or obtaining requests from the input), a library has been created to simplify their development. Ports to several common programming languages are available, and there are also classes to simplify cross programming language interfaces (for instance, to encapsulate Prolog code inside a C++ main program, given that input/output handling is much simpler and more efficient in the latter, while the program logic may be easier to express in the former).