PLUTO: A Preprocessor for Multilingual Spoken Language Generation

by

Brooke A. Cowan

B.A., American Studies
Stanford University (1994)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2004

Author ....
Department of Electrical Engineering and Computer Science
December 22, 2003

Certified by ....
Stephanie Seneff
Principal Research Scientist
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

PLUTO: A Preprocessor for Multilingual Spoken Language Generation

by

Brooke A. Cowan

Submitted to the Department of Electrical Engineering and Computer Science
on December 22, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Surface realization, a subtask of natural language generation, maps a meaning representation to a natural language string. This thesis presents an architecture for a surface realization component in a spoken dialogue system. The architecture divides the surface realization task in two: (1) modification of the meaning representation to adhere to the constraints of the target language, and (2) string production. Each subtask is handled by a separate module. PLUTO is a new module, responsible for meaning representation modification, that has been added to the Spoken Language Systems group's surface realization component. PLUTO acts as a preprocessor to the main processor, GENESIS, which is responsible for string production. We show how this new, decoupled architecture is amenable to a hybrid approach to machine translation that combines transfer and interlingua. We also present a policy for generation that specifies the roles of PLUTO, GENESIS, and the lexicon they share. This policy formalizes a way of writing robust, reusable grammars. The primary contribution of this work is to simplify the development of such grammars in multilingual speech-based applications.

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

Acknowledgments

> *Pluto's symbolism is about change and transformation through the painful process of finding truth and meaning.*
>
> –Annabel Burton

In the spring of 2000, I went to have a reading with an astrologer. He told me that I was approaching my Saturn return: within the next couple of years, I would embark on a journey; I would not return until I had found myself. In the spring of 2001, I received an email informing me that my application to graduate school in computer science at MIT had been accepted. My journey was to commence in September.

This master's thesis commemorates the completion of the first stage of the journey.

***

To my best friend, Matto Marjanović: Thank you. Thank you. Thank you. You have made the process so much easier. You have helped translate the language I have had to learn at MIT in the past two and a half years into language I can understand. You have been my guide through tangled forests, my light in darkened recesses. You have challenged me and listened to me and discussed with me. The journey is much richer for you. Ja volim tebe.

***

To my advisor, Stephanie Seneff: It is because of the opportunity that you gave me that I have reached this place. Thank you for shepherding me here.

***

To my friends from MIT: You are so amazing — so curious, hard-working, talented, and kind — my kin, you are my family: Rodney Daughtrey, Ed Filisko, Tracy Hammond, Dave Huynh, Sayan Mitra, Louis-Philippe Morency, Mike Oltmans, Ali Rahimi, Luke Zettlemoyer. Thank you for making it fun.

***

To the Spoken Language Systems group: I have learned so much from the exposure I have had to the work that you do. Thank you especially to Jim Glass (for making me take 6.345, which I loved), TJ Hazen (for helping me with my project on automatic speech recognition for women and children), Karen Livescu (for authoring the master's thesis I read on-line while applying to MIT), Chao Wang (for explaining Chinese and GENESIS), Lauren Baptist, whom I would really like to meet (for writing a really helpful master's thesis), Scott Cyphers (for disentangling my CVS updates), Shinsuke Sakai (for making helpful suggestions about how to present this work), and Victor Zue (for supporting me from, and before, the start).

***

To Regina Barzilay: Thank you for thinking with me about generation. You helped me see what this work could be.

***

A Eduardo Torres-Jara: Muchísimas gracias por revisar las traducciones que el sistema producía. Deberíamos hablar más en español. Me encanta el español, y lo extraño mucho.

***

To my friends from home, Lisa Rapoport and Jocelyn Sperling: I miss you. Thank you for being like sisters and for never letting our friendship die, even when deserts and mountains and hours of air travel separate us.

***

To Momma Johanna: Thank you for feeding me both yummy food and wise stories.

***

To my family: Zach Cowan, my brother, born on Friday the 13th during a meteor shower, half man, half river. Yes, this is a wonderful place for me. And yes, that sounds like a wonderful place for you. Come see me play hockey; I will swing down your way next winter. Papa, my mentor, my inspiration: I know why you love MIT. Being here makes me even more like you. But no less like you, Mama (thank God). You are THE ARTIST. Thank you both for being my parents. I LOVE YOU!!!!!!

***

May the journey continue.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis focuses on recent changes to the Spoken Language System Group's surface realization component. Surface realization is a subtask of natural language generation (NLG), and NLG is, in turn, a subtask of many language-based computer systems. For example, in a dialogue system, NLG can be used to produce the system's response in the form of a string. NLG is also used to produce large quantities of computer-generated text, for instance a technical manual or a summary of news articles.

NLG is commonly decomposed into three subtasks: *text planning*, *sentence planning*, and *surface realization* [31]:

- the role of the **text planner** is to produce a structured set of communicative goals for the output;
- the **sentence planner** takes this structured set and groups the goals into sentence-size chunks, ordering them for coherence;
- the **surface (or string) realizer** maps each subset of communicative goals to a single sentence.

The output of the NLG system is a set of one or more language strings.

NLG systems are often characterized by the approach they take to generation. Our surface realization component blends two prominent approaches used in NLG: templates and linguistics. Templates are abstractions over strings; they contain variables that can be instantiated with particular values. Linguistics-based systems use formal representations of linguistic theories to represent a wide range of linguistic phenomena. The rules that we create to produce strings in our systems are template-like in that they abstract over strings; however, they also resemble the rules of a linguistic approach in that they exploit knowledge about syntactic structure and require the system to compute the correct forms of words (e.g., conjugations of verbs and inflections of nouns). For this reason, we call them *rule-templates*, emphasizing the hybrid nature of our surface realizer.

While the basic character of our surface realizer has not changed, we have made significant modifications to its structure. Specifically, we have added a new module, PLUTO, that acts as a *preprocessor* for the main processing engine, the string producer GENESIS (see [4] [5] [37] for more details on GENESIS). We aim to show in this thesis that the addition of PLUTO, and the major architectural changes it represents, greatly facilitate our ability to develop robust and reusable multilingual grammars for speech applications. We focus in particular on a machine translation (MT) application as we develop our argument.

This chapter first gives an overview of PLUTO and its role in the surface realizer. We then present related work in NLG and MT that will be important background information for the remainder of this thesis.

1.1 A New Module Called PLUTO

PLUTO is a new module in our surface realization subsystem, born of a desire to facilitate the development of multilingual grammars in speech-based applications. The input to our surface realizer is a *meaning representation*: a structure that captures the content of the system's output. The output of the surface realizer is a grammatically-correct language string that conveys this content. Within the surface realizer, PLUTO preprocesses the meaning representation before it is passed to the main processor, GENESIS, which turns the modified meaning representation into a string.

PLUTO is used primarily for two purposes, the first of which is to infer what features in the input meaning representation are (1) necessary for string production but missing and (2) superfluous but present. Because the new component both inserts and deletes features, we refer to this capability as *feature selection*. PLUTO can also be used to reorganize the structure of a meaning representation to better reflect a natural mode of expressing a concept in the target language. That is, PLUTO is also capable of *structural transformation*. Previously, GENESIS performed all surface realization tasks (i.e., feature selection, structural

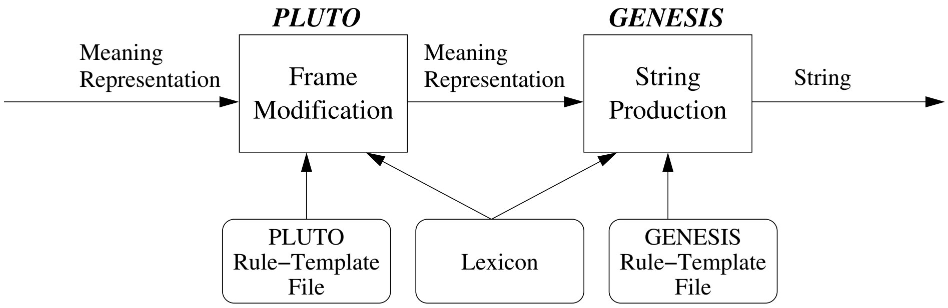

Figure 1-1: The new incarnation of the surface realizer includes two components instead of one: one is responsible for feature selection and structural transformation (frame modification) while the other performs word inflection, choice, and ordering (string production).

transformation, and *string production*, which involves the inflection, choice, and ordering of words). Figure 1-1 depicts the new architecture of the system.

The division of realization tasks in the system's new architecture has several advantages. First, the assumption that exactly the information necessary for string production exists in the meaning representation makes it easier to write rules-templates for GENESIS. Also, performing inference in a separate stage means that PLUTO can explore the meaning representation without actually having to generate strings. Finally, the decoupling allows us to define a policy for the surface realization components that brings us closer to the ideal of developing domain-independent language resources. These advantages make it much easier for us to develop multilingual generation grammars and will be elaborated on throughout this thesis.

1.1.1 Feature Selection

One aspect of NLG that can be particularly difficult is the production of missing features in the target language. In MT, which can be viewed as a special case of NLG in which the meaning representation input to the surface realizer is derived directly from the source text, some features may need to be ignored in the input meaning representation. Alternatively, some features may need to be hallucinated from the input meaning representation. This is because it is unlikely that the features in the source and target languages are identical. For example, English requires articles in places where Chinese does not. (In fact, Chinese does not use articles at all.)

In our system, two utterances e and f with the same meaning, where e is in language $L1$ and f is in language $L2$, will have similar but not necessarily identical meaning representations. They will be similar because, when parsed, they will be mapped to a common ontology and grammar. However, features like articles that exist in $L1$ but not $L2$ will, if present in e, appear in the meaning representation corresponding to e but not f. Figure 1-2 shows an example of a meaning representation derived from Chinese input that is lacking information needed to produce a string in English. In this case, these features need to be inserted in the meaning representation.

Alternatively, features sometimes need to be deleted from a meaning representation. If language $L1$ has a feature that language $L2$ does not ever exhibit, then there is no danger in allowing that feature to persist in the frame. Such is the case with English and Chinese with respect to articles: English uses them, whereas Chinese does not. If translating from English to Chinese, it may be acceptable to leave articular information in the meaning representation because Chinese string production rules will never name this feature, and it will not be spuriously generated.

However, it may be the case that $L1$ and $L2$ both exhibit some feature but use it differently. For example, both English and Spanish use articles. However, in Spanish the definite article is used before a noun to refer to something in a general way (mass or uncountable nouns) or to refer to all the members of a class. Thus, in Spanish, the sentence *I like fish* would be expressed roughly as *I like the fish*, since *fish* is used here to refer to all the members of its class. Figure 1-3 shows a meaning representation derived from the Spanish sentence *Me gusta el pescado* (*I like fish*, in English) and the transformation that produces the correct English translation.

Making appropriate insertions and deletions of features at the level of the meaning representation allows us to write simpler rule-templates in GENESIS. For instance, if a GENESIS rule-template says that an article should be produced in a noun phrase, that article will appear in the output only if it actually exists in the meaning representation. In other words, it is PLUTO's responsibility to define and assure correct usage. We present concrete examples of feature selection rule-templates in Chapter 2.

1.1.2 Structural Transformation

PLUTO can also alter the structure of a meaning representation to make it conform to the

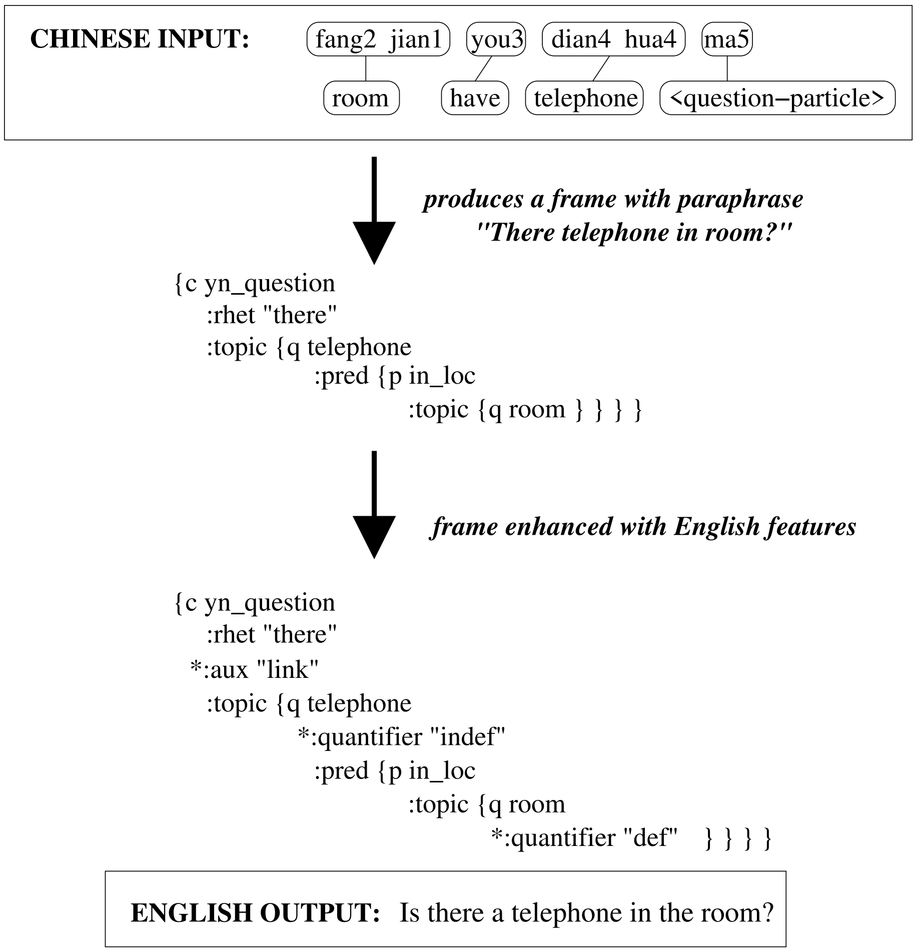

Figure 1-2: The Chinese sentence *Fang2 jian1 you3 dian4 hua4 ma5* generates a meaning representation that is lacking definite and indefinite articles as well as auxiliary verb information. As is, this meaning representation would produce a paraphrase such as *There telephone in room?* Adding appropriate articles where needed in the meaning representation yields a new meaning representation which can be paraphrased as *Is there a telephone in the room?* In this example, the portions of the meaning representation that have been modified during processing by PLUTO are prepended with an asterisk (*). This syntax is used throughout the thesis to highlight changes to the meaning representation.

rules of the target language. Again, we use MT by way of example: in English, the verb *to like* is used to express a person's preference for something. In Spanish, the same idea is conveyed with the verb *gustar*, corresponding most closely to the verb *to please*. Thus, a sentence such as *I like Indian food* becomes *Me gusta la comida india*, where the Spanish words have roughly the following English correspondences:

Me	gusta	la	comida india.
to me	is pleasing		Indian food

Figure 1-4 shows how the subject and object of the Spanish sentence are reversed with respect to the English equivalent.

Figure 1-5 shows that, given a frame derived from English input, it is possible to perform a simple structural transformation that swaps the subject and the object so that the meaning representation better reflects the structure of the Spanish translation. In our system, it is possible to write target-language-specific rule-templates to perform structural transformations in a manner independent of the source language. We give a concrete example of this in Chapter 2.

Like feature selection, we perform structural transformations in a preprocessing stage because it makes the string production rules simpler to write: when writing rule-templates for GENESIS, the developer can assume that the meaning representation has been structured appropriately with respect to the target language.

1.2 Related Work

At the most general level, PLUTO is a module for modifying meaning representations in the surface realization component of a spoken dialogue system. The work described in this thesis focuses in particular on using PLUTO and the new decoupled architecture it represents to develop high-quality multilingual grammars for limited domains. Because the context of this work is a dialogue system with an emphasis on multilingual applications, we explore here both spoken language generation and multilingual generation, subfields of natural language generation.

By way of example and so as to show how our new architecture facilitates the development of multilingual grammars and applications, this thesis describes a speech-to-speech

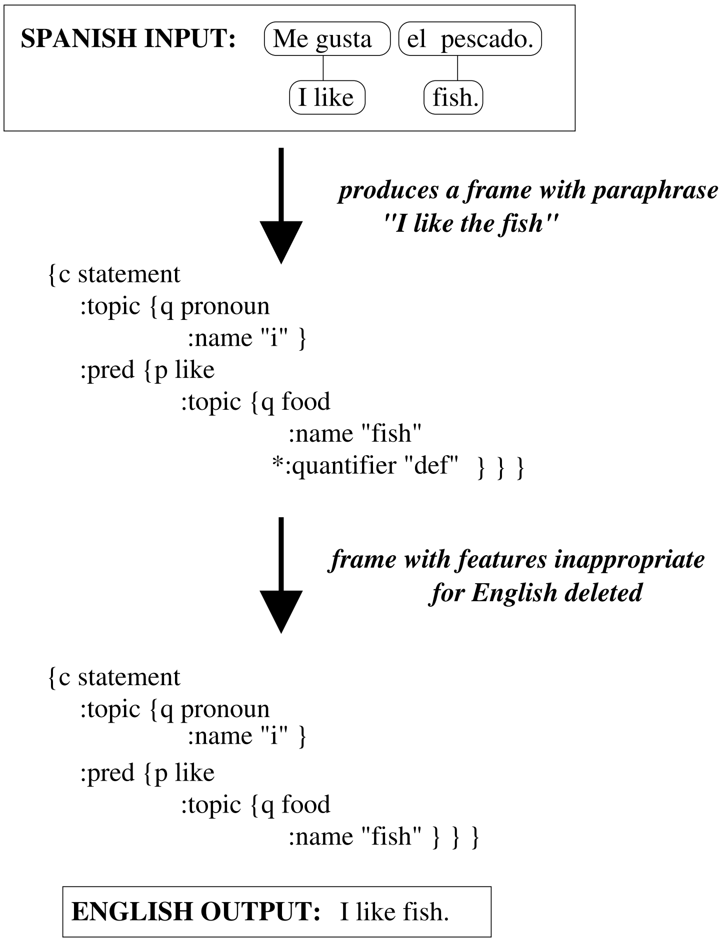

Figure 1-3: The Spanish sentence *Me gusta el pescado* generates a meaning representation with a superfluous definite article with respect to English. This input meaning representation would produce a paraphrase such as *I like the fish.* Deleting the article during preprocessing yields a new meaning representation which can be paraphrased correctly as *I like fish.*

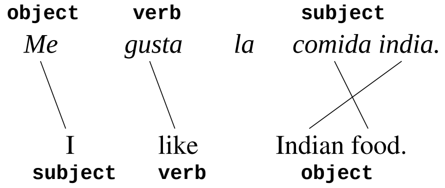

Figure 1-4: The *gustar* transformation reverses the roles (subject/object) of corresponding words. The lines in the figure connect words with roughly the same meaning and depict these correspondences.

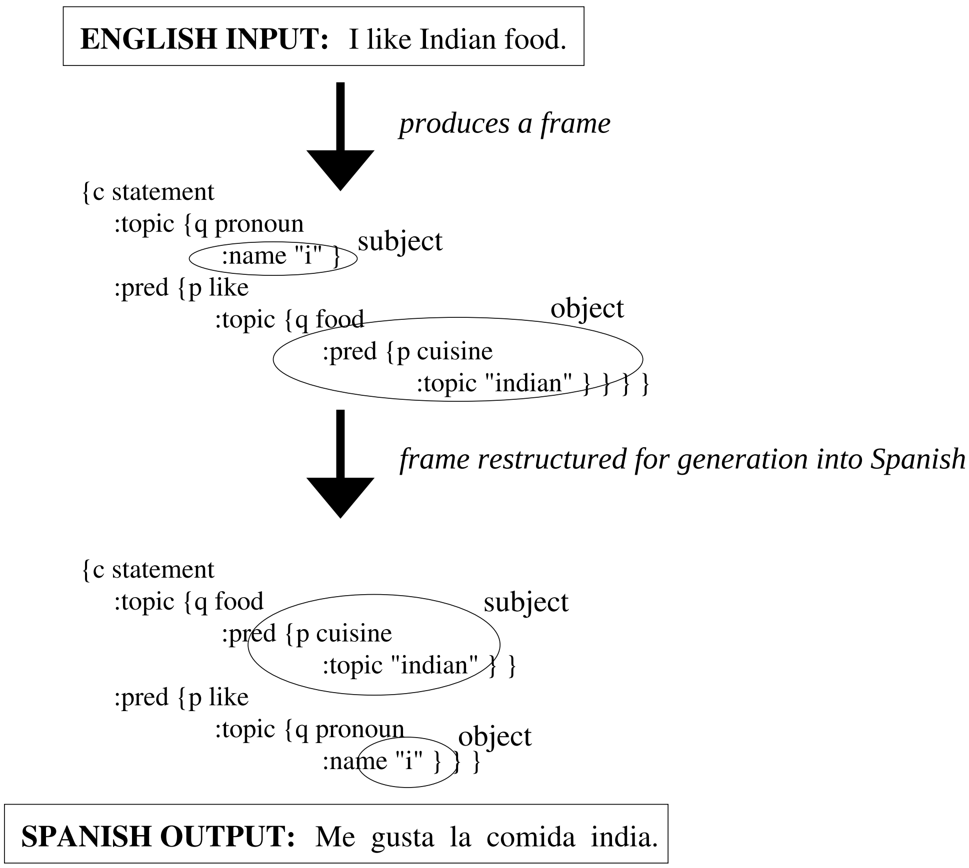

Figure 1-5: The English input *I like Indian food* is analyzed to produce a frame with subject "I" and object "Indian food." The frame then undergoes a structural transformation for generation into Spanish, and the subject and object are swapped.

translation system. For this reason, we also touch on some relevant ideas in MT that will prove useful in describing our approach.

1.2.1 Natural Language Generation

The task of natural language generation (NLG) is commonly formulated as the mapping from a representation that embodies the meaning of some information to a grammatical natural language string or set of strings that conveys the same information. In general, the string output may be either written or spoken text.

Reiter [28] surveys several prominent NLG systems and describes a common underlying "consensus architecture" among systems representing a wide range of linguistic and methodological theories. This modularized, pipeline architecture is broken into five stages. Reiter and Dale [31] later consolidate these five stages into three: content determination, sentence planning, and surface realization. The NLG literature since these articles appeared generally assumes this tripartite architecture. In this thesis, we focus on surface realization.

1.2.2 Templates, Linguistics, Statistics, and Hybrids

In general, researchers have used *shallow methods* (e.g., canned text and templates), *deep methods* (e.g., linguistic theories), and *statistical methods* to implement natural language generation in spoken dialogue systems. We review each of these approaches here and then describe some hybrid approaches.

Shallow Methods

Using *canned text* for NLG means outputting pre-composed strings. Canned text is guaranteed to be grammatically correct; however, it implies highly-constrained dialogue because the computer's language mechanism is static. *Templates* abstract away from canned text: they are patterns of strings containing variables that can be instantiated with values from the specific meaning representation being processed. For instance, the sentence *There are three Chinese restaurants on Massachusetts Avenue* can be abstracted as a template, with variables replacing actual values: *There are $NUMBER $CUISINE restaurants on $STREET-NAME.*

Most commercially-available systems use templates for NLG. Among the advantages of templates that Reiter [29] lists are simplicity and ease of debugging. Templates are simple to implement because they do not require syntactic or semantic analysis in the generation process. For example, the system need not be endowed with any linguistic knowledge for computing verb conjugations or number agreement phenomena. Templates are easy to debug because the developer can simply add new templates to handle ungrammatical cases (e.g., *There is one $CUISINE restaurant on $STREET-NAME*).

Most spoken language dialogue systems initially used or continue to use templates in some way (e.g., [26] [1] [40] [27]). There are at least two possible explanations for why templates have been so common in such systems: (1) the real-time requirement of spoken dialogue systems (templates are in general very computationally efficient), and (2) the constrained nature of spoken dialogue domains (a small number of templates is sufficient to cover the scope of system output).

In spite of the advantages of templates, researchers have generally not been content with them. Reiter [29] points out that they are inflexible and difficult to maintain and reuse; furthermore, he considers them insufficient for sophisticated tasks such as multilingual translation. Axelrod [1] and Oh and Rudnicky [26] mention the problem of combinatorial explosion in the number of templates: a "simple template" with n variables can actually require 2^n templates to ensure correct output when certain variables are not present in the input. Axelrod [1] circumvents this difficulty with a mechanism that causes subphrases to be generated solely when the variables they contain are to be instantiated and included in the output string. Our system uses branching mechanisms embodied in scripting languages to achieve a similar effect.

Deep Methods

Deep methods are theoretically-grounded and rule-based. For instance, *linguistic* approaches draw on linguistic theories to generate natural language: the system uses encoded rules to compute appropriate forms of words in grammatically-correct strings. Among the most common theories used are systemic-functional grammars [18] and meaning-text theory (MTT) grammars [25].

SURGE [13], based on the Functional Unification Formalism (FUF), is one system that has drawn heavily on systemic-functional linguistics; two others are the KPML (Komet-Penman MultiLingual) system [6] and PENMAN [17], both based on the NIGEL grammar [24]. Meaning-text theory defines seven canonical representation strata (one semantic, two syntactic, two morphological, and two phonological) and provides mappings between them. Systems that have drawn on the latter include RealPro [23] and FoG [16].

Linguistic techniques are appealing because they seem to come closer to building the ability to produce language into the computer. Practically speaking, they produce high-quality output and are easier to maintain than templates. However, they also require a deep understanding of the linguistic theories employed and handcrafting of extensive knowledge bases to develop [32].

While many off-the-shelf linguistic generators are available, a common complaint is that they "...require inputs with a daunting amount of linguistic detail" (Langkilde and Knight [21]). Also, for spoken dialogue systems, the generality inherent in linguistically-based systems can be a disadvantage: "...the quality of the output for a particular domain, or a particular situation in a dialogue, may be inferior to that of a template-based system without considerable investment in domain-specific rules or domain-tuning of general rules" (Walker et al. [40]). Notably, no spoken dialogue system that I know of has used a pure linguistic approach in the NLG component. Perhaps this is because, in addition to requiring substantial expertise, they are computationally expensive and generally do not work in real-time.

Statistical Methods

Many researchers believe that statistics promise to alleviate the burdens associated with the handcrafting of rules and templates. Recently, there have been many investigations into systems that integrate *statistical* methods into NLG. For instance, Knight and Hatzivassiloglou [20] present a surface realization system that uses rules to overgenerate candidate output strings. Their system searches the candidates for the highest probability string using a bigram model.[1] In a different approach, Ratnaparkhi [27] uses a large corpus of templates to build systems that learn to choose and order words appropriately.

Bangalore and Rambow [2] also develop a statistical surface realization component, using a wide-coverage tree-adjoining grammar and a probability model to annotate dependency trees and generate strings from them. Finally, Oh and Rudnicky [26] use corpus-based methods for surface realization, breaking utterances into classes and building 5-gram language models for each utterance class. These models are then used to predict the next word when creating a string from a meaning representation.

While the first foray into statistical NLG (Knight and Hatzivassiloglou [20]) was motivated by large-scale text-based MT, recently a lot of work in this area has been with spoken dialogue systems (e.g., [27] [2] [9] [26]). This may be due to the fact that off-the-shelf linguistic systems (1) cannot be used without extensive domain tuning and (2) compromise real-time requirements, while templates are too inflexible for modeling realistic dialogues. Statistical methods may be able to generate a wider range of output efficiently.

[1] A bigram model is based on conditional probabilities, where the probability of a particular word w in a sentence is simplified such that it depends only on the previous word in the sentence: $P(w_i|w_{(i-1)} \ldots w_1) = P(w_i|w_{(i-1)})$.

Hybrid

There has been some suggestion in the history of NLG that *hybrid* approaches could be effective. In 1999, for instance, there was a workshop on templates and linguistics (or *free choice*) held at the German Annual Conference on AI entitled *Between Templates and Free Choice in Natural Language Generation: What is the Right NLG Technology for my Application?* Many of the papers from this workshop note that using a pure linguistics-based system is impractical for a limited domain that does not need a full-coverage grammar. They suggest ways in which the relative merits of both shallow and deep approaches can be exploited.

For example, Reiter [30] describes the ways in which templates and linguistics are used in the STOP system,[2] noting that linguistic methods are often superior to templates but that lack of documentation can make existing linguistic systems prohibitively difficult to use. Hence, linguistics and other deep methods are applied in STOP solely to phenomena that can be explained using simple, easily-coded rules (e.g., constraining the length of the text).

van Deemter et al. [38] take a different approach to hybridization, folding linguistics into the templates themselves to fabricate "syntactically structured templates" which draw on well-known linguistic theories in the formulation of strings. The authors find that when developing a new domain, they are able to re-use much of the code for producing strings from templates, but that they have to restructure the templates themselves.

The Spoken Language Systems Group's generation system has been described as "...a set of fine-grained recursive templates, which represent general linguistic abstracts and domain-independent facts about language" by Walker and Rambow [41]. As a hybrid system, ours most resembles the type described by van Deemter et al. The *rule-templates* that we create compute the appropriate forms of most words that are included in a string. Hence, they are like abstractions of templates. For example, the template *There are $NUMBER $CUISINE restaurants on $STREET-NAME* can be abstracted as a rule that generates the appropriate form of the linking verb (*is* or *are*) and the head noun (e.g., *restaurant* or *restaurants*) according to the value of $NUMBER. Using this approach, many generalizations about language can be encapsulated in template-like rules.

[2] STOP is a system that automatically produces personalized smoking-cessation pamphlets.

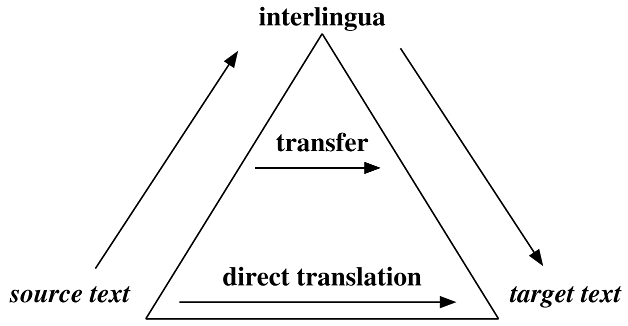

Figure 1-6: The MT pyramid.

1.2.3 Machine Translation

Approaches to MT are commonly summarized by a pyramid such as the one in Figure 1-6 (from [19]). At the bottom of the pyramid is *direct translation*, devoid of any intermediate analysis. For example, the following algorithm describes one possible manifestation of direct translation: the words in the source text are mapped to some root form, searched for in a bilingual dictionary, translated, and reordered.

The other two approaches represented by the pyramid, *transfer* and *interlingua*, depend on an increasing amount of analysis of the source text. Transfer imposes a language-dependent analysis of the source at the syntactic and/or semantic level. The resulting structure is then manipulated and rearranged to represent a structure conforming to the constraints of the target language. Verbmobil is a well-known MT system that has its roots in transfer [39]. An interlingual representation, in contrast, is language-independent and represents syntactic and semantic universals. Interlingual analyses of the source and target result in the same representation, based on these universals. Translation occurs immediately following the interlingual analysis. Dorr [12] describes an interlingual approach.

A common criticism of transfer-based systems is that the number of generation rule sets is $n(n-1)$, where n is the number of languages supported. This is because a separate rule set has to be developed for each source/target language pair. Using an interlingua reduces rule-set complexity to $O(n)$, where n is the number of target languages. This is because all source languages are mapped to a language-independent representation during analysis. Because an actual interlingua can be difficult to design, hybrid approaches have been suggested [8].

The approach to MT described in this paper is also hybrid, combining the transfer and interlingual approaches. However, in our approach we are able to successfully construct an interlingua because we are working within constrained domains. The analysis of the source (see Seneff [34] for more details on the analysis component, TINA) results in an interlingual representation we call *source-biased* because it preserves source-language features. That is, two sentences in different languages $L1$ and $L2$ that share the same meaning will also share the same meaning representation, with the following exception: if feature ϕ exists in language $L1$ but not in $L2$, ϕ will occur in the meaning representation derived from the sentence in $L1$ but not that derived from the sentence in $L2$. Hence, generation rule sets need only be developed for the n target languages supported in the translation application. PLUTO's role in translation can be viewed as a transfer step: feature selection and structural transformation modify the interlingua such that it conforms to the constraints of the target language. Finally, GENESIS decides which words to use to express the concepts in the meaning representation, computes their correct forms, and finds an appropriate ordering for them. We describe this hybrid approach to MT in detail in Chapter 3.

1.3 Outline

The remainder of this thesis is divided into four chapters:

- Chapter 2 focuses on the new architecture of the surface realization component,
- Chapter 3 shows how the new architecture can be used to implement a hybrid approach to MT in a constrained domain, and in particular how PLUTO may be used to meet many of the challenges of MT within this framework;
- Chapter 4 develops a policy for the use of the Spoken Language Systems group's generation subsystem in natural language applications;
- Chapter 5 discusses formal evaluation, conclusions, and future work.

Chapter 2

A New Module for Generation

PLUTO is a new module in the surface realization system used by the Spoken Language Systems group. Designed to carry out the tasks of feature selection and structural transformation, PLUTO represents a decoupling of the generation subtasks: previously, all such subtasks were relegated to GENESIS; now, GENESIS is responsible solely for string production.

2.1 The New Architecture

PLUTO has been designed in the context of the Spoken Language Systems group's GALAXY architecture [35] in which modules communicate with one another via a central hub. Figure 2-1 depicts an instantiation of GALAXY in which basic speech applications are broken into eight primary modules. The *audio* component captures audio information from the raw input signal (for example, from a telephone or a microphone). Audio information is then passed to the *speech recognizer*, which produces a set of textual utterance hypotheses. These hypotheses become the input to the *language understanding* component, which decides which hypothesis is the best and constructs a meaning representation for it. A *dialogue management* module receives this meaning representation and interprets it in relation to the dialogue history, calling on the *context resolution* component if necessary (to resolve referring expressions, for instance). The main task of the dialogue manager is to construct the system's response in an abstract form (i.e., meaning representation). In order to do so, it might also make use of the *database* (and even of the *surface realization* component, to form a database query). This meaning representation is then passed to the *surface realization* component, whose job is to produce a language string. Finally, the string is passed to

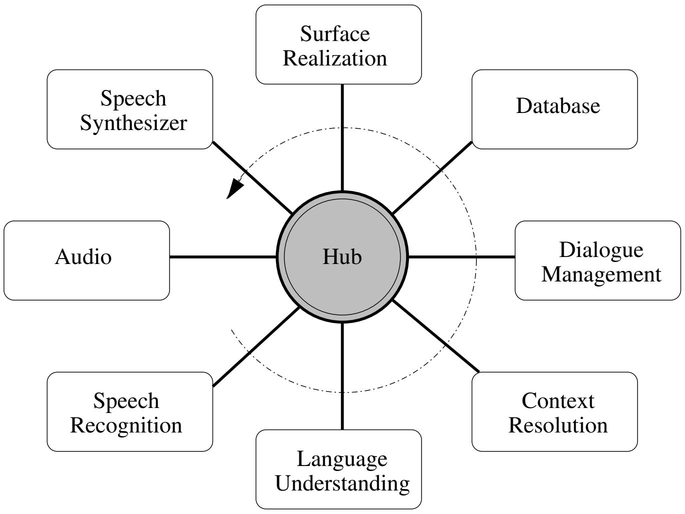

Figure 2-1: In the GALAXY architecture, modules communicate with one another via a central hub. The dashed arrow represents the flow of information through the system following a user query or utterance: first, the speech is processed by the audio/GUI and speech recognition modules. Next, the language understanding module constructs a meaning representation. The context resolution server and dialogue manager interpret the user utterance and produce an abstract response in the form of a meaning representation with information from the database. The surface realization component produces a string from the meaning representation, and the speech synthesizer outputs the string in spoken form.

the *speech synthesis* component and output in spoken form.

Figure 2-2 depicts the new architecture of the surface realization module. The role of the surface realization component is to map a meaning representation to a string. PLUTO takes a meaning representation and performs feature selection and structural transformation, producing a new meaning representation. GENESIS takes the preprocessed meaning representation and performs lexical selection, morphological inflection, and word ordering to produce a string. Both PLUTO and GENESIS have access to external rule-template files and a lexicon, resources that enable the processing of a meaning representation.

2.2 The Inner Workings of PLUTO

PLUTO carries out the tasks of feature selection and structural transformation via two mechanisms: (1) rule-templates specified with a scripting language in an external file and (2) algorithms for choosing appropriate rule-templates and for processing meaning representations.

PLUTO's core resembles that of a general-purpose interpreter and consists of three com-

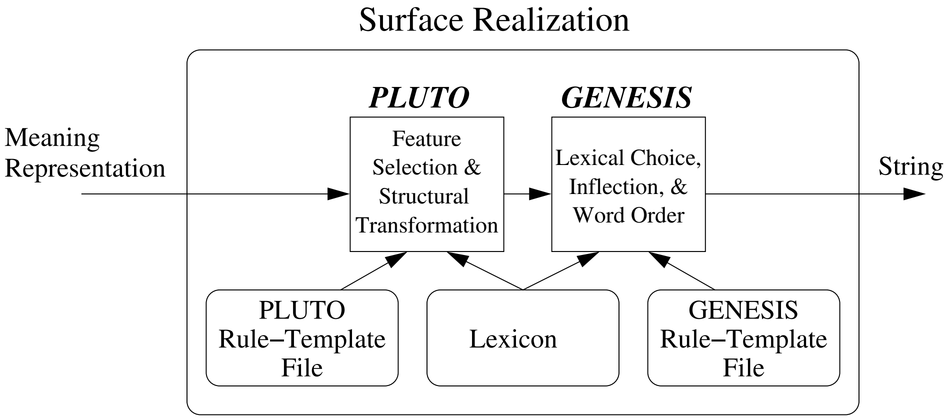

Figure 2-2: The surface realization module's new architecture. *Surface realization* is the process of mapping from a meaning representation to a string. In the new architecture, realization is conceived of as a two-stage process in which subtasks are divided between PLUTO (feature selection and structural transformation) and GENESIS (lexical selection, morphological inflection, and word ordering).

ponents: a parser, a command processor, and a lexical accessor (see Figure 2-3). The parser maps linguistic rule-templates written in the scripting language to an internal representation; the rule-templates are stored in a library that the command processor has access to while processing a meaning representation. The command processor also has access to the lexicon via the lexical accessor.

A PLUTO *rule-template file* is a hand-crafted set of rule-templates that dictate the specific feature selection and frame transformation actions to perform on meaning representations. Each instance of such a file may be tailored to the needs of a particular application and/or target language. The configurable nature of PLUTO makes it a flexible component for use in many settings.

The *lexicon* is also hand-crafted. It contains entries for the domain vocabulary items with their part-of-speech and default generation string. Particular entries may also have alternative generation strings and other linguistic information such as features of the target language (e.g., number and gender). PLUTO uses the lexicon to populate a meaning representation with features associated with relevant lexical entries. PLUTO can also use the values of these features to make other decisions concerning modifications to the meaning representation.

Meaning representations are implemented in our systems with hierarchical frames of *attribute:value* pairs. The meaning representations used throughout this thesis contain both

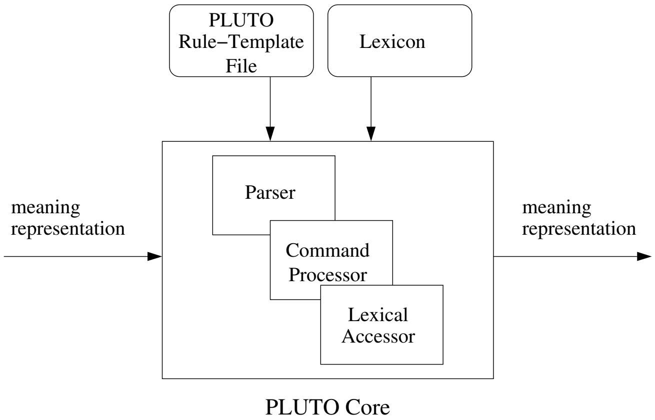

Figure 2-3: The basic architecture of the PLUTO system. PLUTO's core consists of a parser for parsing the syntax of the rule-template file, a command processor for choosing rule-templates and interpreting commands when processing frames, and a lexical accessor for moving information from the lexicon into frames.

semantic and syntactic information and are referred to as *linguistic frames*. The linguistic frame in Figure 2-4 was produced by the language understanding component TINA (Seneff [34]), a probabilistic natural language understanding component that builds parse trees using context-free grammar rules, a semantic network, and a training corpus. Linguistic frames are derived from parse trees. The parse tree from which the linguistic frame in the example was derived is shown in Figure 2-5.[1]

There are two principal algorithms that drive PLUTO's behavior in general: a processing algorithm and a rule-template-choosing algorithm. The processing algorithm may be characterized as *top-down*, *recursive*, and *deterministic*.[2] It is top-down because PLUTO always begins processing a frame at its top level and works down. It is recursive because the result of processing a frame f is a new frame f' that is the result of processing f's constituents (which may themselves be frames). It is deterministic because there is exactly one rule that will match and be used to process any frame.[3]

The pseudocode in Figure 2-6 gives an idea for how processing proceeds. The input to

[1]For more information on basic frame terminology, as well as the structure of linguistic frames, see Appendix A. Appendix A also describes the scripting language used to write rule-templates and PLUTO's use of the lexicon in more detail.

[2]The overall structure of the processing algorithm is very similar to the one used in GENESIS (see [5], pp. 52–57). This kinship makes system maintenance easier. It also makes it easier to become familiar with how the surface realization system operates.

[3]It would be relatively simple to modify this behavior. GENESIS actually has the capability to deal with rule "alternates" (see [5], pp. 80–81); defining a probability distribution over alternates in the context of PLUTO could allow the system to produce multiple strings from the set of all possible strings mapped to by a particular linguistic frame.

```
{c statement
   :topic {q pronoun
            :name "we" }
   :mode "past"
   :aux "link"
   :pred {p unable
           :aux "to_inf"
           :v_complement {p get
                            :topic {q ticket
                                      :number "pl"
                                      :pred {p for
                                               :topic {q event
                                                         :quantifier "def"
                                                         :name "show" } } } } } }
```

Figure 2-4: A linguistic frame representing an English string such as *We were unable to get tickets for the show.* This linguistic frame was derived from the parse tree in Figure 2-5.

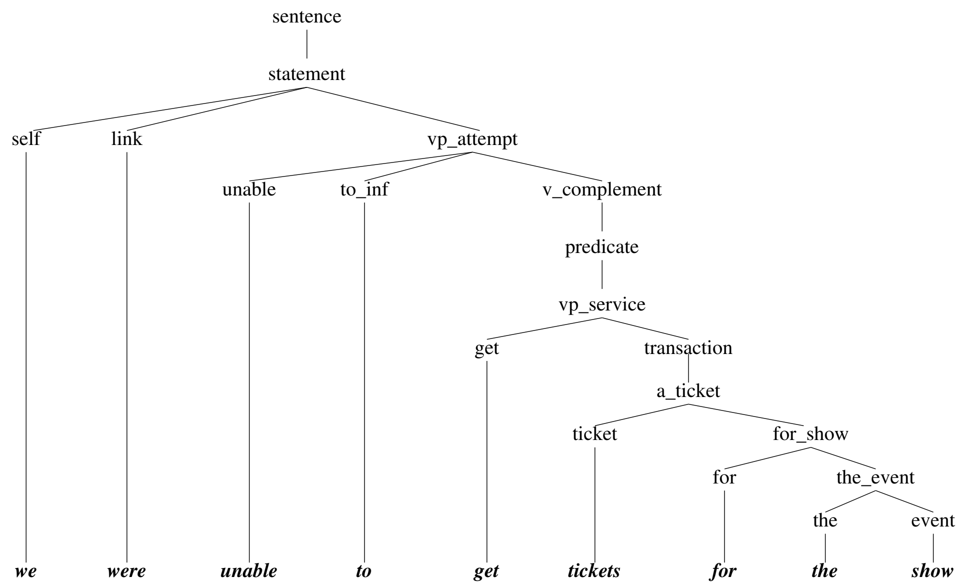

Figure 2-5: A parse tree for the string *We were unable to get tickets for the show*, produced by TINA. This parse tree forms the basis for the linguistic frame in Figure 2-4.

PLUTO is a linguistic frame. PLUTO first copies the linguistic frame and creates an empty *info-frame*. The info-frame is a structure used to propagate global frame information and to record information as it is learned.[4] Then, PLUTO finds a rule-template with which to process the frame, searching a library of rule-templates. Once an appropriate rule-template has been identified, each of its commands is examined for its type, and the frame, info-frame, and command are passed to an appropriate command-processing subroutine.

The pseudocode for the function `find_rule_template_for_frame()` in Figure 2-6 captures the essence of the rule-template-finding algorithm used by PLUTO. The algorithm looks first for a specific rule for a frame, then, failing that, backs off to progressively more general rule-templates. Specifically, given a frame, PLUTO looks in the rule-template library for a rule-template whose name matches the name of the frame. If there is none, then PLUTO checks to see if the frame name is in a group. If it is, PLUTO uses the group rule-template to process the frame. Finally, if there is no specific rule-template and no group rule-template for the frame, then PLUTO finds the default rule-template for the frame type.[5] The use of binary search to locate specific rule-templates in the library helps make the processing more efficient.

2.3 Frame Modification Examples

We demonstrate how PLUTO carries out frame modification tasks with three examples. The first two focus on feature selection (both adding and deleting features), while the third focuses on structural transformation. The three examples were first introduced in Section 1.1. Here we discuss how PLUTO chooses and interprets rule-templates to effect the desired changes.

2.3.1 Feature Selection

The first feature selection example involves the input linguistic frame in Figure 2-7. Using the rule-templates in Figure 2-8, PLUTO produces the frame in Figure 2-9, ready for string production in English. The rule-templates handle cases of the general form *There is/are a/some $NOUN1 in the $NOUN2.* The strategy embodied by the rule-templates is to (1)

[4]For more information on the info-frame, see Appendix A.

[5]Our linguistic framework currently distinguishes three frame types: clause, topic, and predicate, where predicate represents verb phrases, adjective phrases, and prepositional phrases.

add a linking verb to the top-level clause frame and an indefinite article to the topic frame based on the presence of the rhetorical *there*, and (2) add a definite article to the *in_loc* predicate.

The processing algorithm dictates that PLUTO begin processing the input at the top level. The name of the top-level frame is `yn_question` and its type is *clause*: since there is neither a specific rule-template named `yn_question` nor a group containing `yn_question` as a member, the generic rule-template *clause_template* is selected to process the frame.

The first command in the *clause_template* rule-template tests for the existence of a keyword called `:rhet`. This test succeeds and so the consequent is performed. The two commands of the *set_rhet_features* rule-template cause (1) an indefinite article to be set in the topic, and (2) an auxiliary verb with value *link* to be set in the top level of the frame.

The second command of the *clause_template* causes the processing of the `telephone` topic. The best rule-template is the generic *topic_template* with just one command that indicates that the `in_loc` predicate should be processed. There is a specific rule-template called `in_loc`, so this is selected. This rule-template sets a definite article in the `in_loc` frame.

Because the commands of all rule-templates selected to process the input have all been executed, processing is now complete, and PLUTO outputs the modified meaning representation in Figure 2-9. This meaning representation might be realized by GENESIS as the English string *Is there a telephone in the room?*

The second example of feature selection demonstrates that features may also be deleted from a linguistic frame. The frame in Figure 2-10 is processed by PLUTO using the rule-templates in Figure 2-11. The effect of the rule-templates is to delete the definite quantifier in the `food` topic frame. This alteration is based on the assumption that, in the domain for which these rule-templates have been written, statements of the form *I like the $NOUN* will never occur. The target language is, again, English.

PLUTO begins processing the frame at the top level and, as in the first example, selects the *clause_template*. The first command of this rule-template sets the mode in the info-frame to *present_indicative*. The details of this rule-template are omitted for brevity. The *:pred* command initiates the processing of the `like` predicate. Because `like` is a member of the clause group *like_preds*, the *like_preds_template* rule-template is selected. The single **if** command contained in this rule-template has two tests, both of which succeed. The **goto**

```
fill_in_features(orig_frame)
  frame <-- copy(orig_frame)
  info-frame <-- create_info_frame(frame)
  find_rule_template_for_frame(frame)
  process_with_rule_template(frame, info-frame, rule-template)
  return frame

find_rule_template_for_frame(frame)
  if there is a rule-template T for name(frame)
     return T
  else if there is a rule-template T' for group G and
         G contains name(frame)
     return T'
  t <-- type(frame)
  T" <-- generic rule-template for t
  return T"

process_with_rule_template(frame, info-frame, rule-template)
  for each command c in rule-template
     t <-- type(c)
     if t = goto
       return process_goto(frame, info-frame, c)
     else if t = if
       return process_if(frame, info-frame, c)

     ...    ;;; there is a processing subroutine for each command type

     else error(unknown command type)
```

Figure 2-6: Pseudocode for PLUTO processing algorithm.

```
{c yn_question
   :rhet "there"
   :topic {q telephone
            :pred {p in_loc
                     :topic {q room } } } }
```

Figure 2-7: This linguistic frame is input to PLUTO. The target language, English, requires that three features (an auxiliary verb and two articles) be added to the frame.

clause_template	($if :rhet >set_rhet_features) :topic
set_rhet_features	>softset_quant >soft_link
softset_quant	($softset :quantifier[:topic] "indef")
soft_link	($softset :aux "link")
topic_template	:pred
in_loc	($softset :quantifier[:topic] "def")

Figure 2-8: PLUTO rule-templates for effecting the necessary modifications to the linguistic frame in Figure 2-7.

```
{c yn_question
   :rhet "there"
  *:aux "link"
   :topic {q telephone
             *:quantifier "indef"
              :pred {p in_loc
                      :topic {q room
                               *:quantifier "def" } } } }
```

Figure 2-9: The linguistic frame output now contains an auxiliary verb as well as two articles, one definite and the other indefinite. This frame might be realized as the English string *Is there a telephone in the room?*

```
{c statement
   :topic {q pronoun
             :name "i" }
   :pred {p like
            :topic {q food
                      :name "fish"
                      :quantifier "def" } } }
```

Figure 2-10: This input linguistic frame, which is to be realized in English, contains a superfluous definite article. It is assumed that in this domain statements of the form *I like the $NOUN* never occur.

```
clause_template      >set_mode :pred
set_mode             ;; this command sets the mode in the info-frame
                     ;; to present_indicative
predicate_groups     like_preds
like_preds           like interest love
like_preds_template  ($if ^:mode == "present_indicative" && \
                     :quantifier[:topic] == "def" >del_def_quant)
del_def_quant        ($del :quantifier[:topic])
```

Figure 2-11: The rule-templates effecting the deletion of the definite article.

```
{c statement
   :topic {q pronoun
            :name "i" }
   :pred {p like
           :topic {q food
                    :name "fish" } } }
```

Figure 2-12: The output linguistic frame, ready for string production. This frame might be realized with the English string *I like fish.* The definite article has been removed.

```
{c statement
   :topic {q pronoun
            :name "i" }
   :pred {p like
           :topic {q food
                    :pred {p cuisine
                             :topic "indian" } } } }
```

Figure 2-13: This linguistic frame is structurally unsatisfactory for output in Spanish. In Spanish, the subject and object of the sentence *I like Indian food* are swapped.

consequent causes the quantifier to be deleted from the `food` topic frame.

2.3.2 Structural Transformation

The final example demonstrates how PLUTO can be used to carry out structural transformations on linguistic frames. The target language is Spanish, and the rule-templates in Figure 2-14 embody the principles related to the verb *gustar* (*to please*), introduced in Section 1.1.2. When applied to the linguistic frame in Figure 2-13, these rule-templates produce the frame in Figure 2-15, in which the subject and the object have been swapped. The strategy of the rule-templates is to process all predicates that are structurally similar to the Spanish verb *gustar* (*to please*) by swapping the predicate's topic and the top-level topic.

The *clause_template* rule-template initiates the processing of the frame. Its first command, a **goto**, executes because there exists a member of the group list *like_preds* in the frame (the predicate `like`). The *like_preds_template* stores the topic of the `like` frame in the info-frame. The second command of the *clause_template* tests for the existence of a *^:topic* info-key, which was just set in the info-frame. The *swap_subject_object2* rule-template executes next, carrying out the actual swapping of the top-level topic and the topic of the

clause_template	>like_preds ($if ^:topic >swap_subject_object2)
predicate_groups	like_preds
like_preds	like interest love
like_preds_template	>swap_subject_object1
swap_subject_object1	($set ^:topic :topic)
swap_subject_object2	($set :topic[:pred] :topic) ($set :topic ^:topic)

Figure 2-14: These rule-templates embody the necessary modifications to swap the subject and object of any verb like the Spanish *gustar*, (*to please*).

```
{c statement
   :topic {q food
             :pred {p cuisine
                      :topic "indian" } }
   :pred {p like
            :topic {q pronoun
                      :name "i" } } }
```

Figure 2-15: The output frame has the original subject and object swapped.

predicate.

2.4 Advantages of Decoupling the Generation Tasks

Introducing a decoupled architecture in the surface realization component has allowed developers working with our system to write cleaner rule-templates that are easier to read and debug. For example, the decoupled architecture has greatly simplified the development of PHRASEBOOK, a multilingual travel-domain translation system. The goal of PHRASEBOOK is to allow users to speak an utterance in a source language and have the system translate and repeat the utterance in a target language. We have developed rule-template sets for three languages (English, Spanish, and Mandarin) and are in the process of adding support for French. In PHRASEBOOK, PLUTO's frame modification capability is used to tailor the linguistic frame produced by the understanding component, TINA, to the syntactic constraints and appropriate word senses of the target language.

An example within this domain shows how the decoupled architecture has simplified the rule-templates for translation into Spanish. Figure 2-16 shows the single rule-template file and lexicon used to generate a string by GENESIS prior to the decoupling, while Figure 2-17 shows the two rule-template files used by PLUTO and GENESIS in addition to the shared lexicon. The rule-templates and lexica in both figures have been written to achieve the

```
;; GENESIS RULE-TEMPLATE FILE
;; clause frame rules
wh_question                     <==:trace > --wh_topic >set_aux \
                                :aux --wh_topic

;; topic frame rules
topic_template                  :name

;; auxiliary rules
--wh_topic                      :topic
set_aux                         ($if :aux == "link" >set_link)
set_link                        ($if at_age >set_link_tener)
set_link_tener                  ($set :aux "link_tener")
```

```
;; LEXICON
;; morphology
N                  PL "s"

;; entries
you           N  "usted" NUM "third"
how_old       O  "cuántos años"
link_tener    X  "tiene" ROOT "tener" FIRST "tengo" THIRD "tiene" \
                 PL_FIRST "tenemos" PL_THIRD "tienen"
```

Figure 2-16: Single rule-template file and lexicon used to generate the string *¿Cuántos años tiene usted?* from the frame in Figure 2-18.

same goal: generate a Spanish string from the meaning representation in Figure 2-18.

The words contained in the target Spanish string in this example have roughly the following English correspondences:

¿Cuántos	años	tiene	usted?
how many	years	have	you

The main verb of this sentence, *tener*, is conjugated in the third person singular, in agreement with the subject *usted*. In order to produce the Spanish string *¿Cuántos años tiene usted?*, the number information from the subject has to be made available to the auxiliary verb so that the appropriate verb form can be chosen.

In the old system, the only way to learn information such as the number of the subject was to *pregenerate* the string. The command > *--wh_topic* in Figure 2-16 executes this pregeneration. The result of this command is that the word *usted* is stored in the info-frame for later inclusion in the target string. The side effect is that the number information from the lexical entry for *you* in Figure 2-16 is also stored in the info-frame, where it can be accessed when the string *tiene* is produced. The command *--wh_question* discharges the

```
;; PLUTO RULE-TEMPLATE FILE
;; clause frame rules
clause_template             :topic >set_clause_number >set_aux

;; topic frame rules
topic_template              :name

;; auxiliary rules
set_clause_number           ($clone :number[:topic])
set_aux                     ($if :aux == "link" >set_link)
set_link                    ($if at_age >set_link_tener)
set_link_tener              ($set :aux "link_tener")
```

```
;; LEXICON
;; morphology
N                 PL "s"

;; entries
you            N  "usted" :number "third"
how_old        O  "cuántos años"
link_tener     X  "tiene" ROOT "tener" FIRST "tengo" THIRD "tiene" \
                  PL_FIRST "tenemos" PL_THIRD "tienen"
```

```
;; GENESIS RULE-TEMPLATE FILE
;; clause frame rules
wh_question                  <==:trace :aux :topic

;; topic frame rules
topic_template               :name
```

Figure 2-17: Two rule-template files and lexicon used to generate the string *¿Cuántos años tiene usted?* from the frame in Figure 2-18.

```
{c wh_question
   :aux "link"
   :topic {q pronoun
             :name "you" }
   :pred {p at_age
            :trace "how old" } }
```

Figure 2-18: Linguistic frame serving as input to the surface realizer. This frame represents, in English, a string such as *How old are you?*

```
{c wh_question
   :aux "have"
   :number "third"
   :topic {q pronoun
              :name "you"
            :number "third" }
   :pred {p at_age
             :trace "how old" } }
```

Figure 2-19: Linguistic frame serving as input to the surface realizer. This frame represents, in English, a string such as *How old are you?*

stored string *usted* at the moment it should be concatenated to the output string. The rule-templates in Figure 2-16 also set the value of the auxiliary verb, changing it from *link* to *have* based on the presence of the `at_age` predicate. This is because in Spanish the verb *tener* (*to have*) is used to talk about age, whereas English uses the verb *to be*.

Rule-templates carrying out tasks such these, now relegated to PLUTO, cluttered the GENESIS rule-template file in the old system, making it difficult to tell when constituents were actually being added to the target string. In contrast, the new system adds a stage in which this kind of processing is completed prior to generation of the target string. The rule-template files and lexicon in Figure 2-17 show the three files that realize the same string, *¿Cuántos años tiene usted?*, given the same frame as input, in the decoupled model. The PLUTO rule-templates, in conjunction with the lexicon, are used to preprocess the frame. Then, the GENESIS rule-templates, also with the lexicon, are used to generate the string. The GENESIS rule-template for generating the constituents of the `wh_question` frame is in particular much more transparent with respect to word ordering. The frame resulting from the PLUTO processing stage is shown in Figure 2-19.

2.5 Summary

Simplifying development is important in rule- and template-based systems, valued for the control they offer in the production of linguistically-sound output, but often associated with high development costs. In this chapter, we have shown how a decoupled architecture can help simplify development by making rule-templates easier to understand. We have also shown how PLUTO can be used to make changes to frame features and structure. In the next chapter, we develop a hybrid approach to MT that exploits the new architecture of the surface realization component and the capabilities of PLUTO.

Chapter 3

A Hybrid Approach to Translation

We have developed a hybrid approach to speech-based MT in a constrained domain that takes advantage of the new architecture of the surface realization component. The technique combines two prominent MT approaches: *interlingua* and *transfer*. The former converts the source string into a language-independent meaning representation before producing a target string (Figure 3-1a); the latter restructures a syntactic parse of the source string to make it conform to the constraints of the target language, producing a target string from the modified parse (Figure 3-1b).

The appeal of the interlingual approach is that, in theory, because source strings from any language are mapped to a canonical form, all that is needed to produce a target string is a mapping from the interlingua to the target language. Hence, only $O(n)$ mappings are needed – one from each source language to the interlingua, and one from the interlingua to each target language. However, it can be difficult to design an interlingua sufficiently universal to accommodate all languages. The appeal of the transfer approach is that it obviates the need to design an interlingua. The disadvantage is that $O(n^2)$ mappings are needed to support n languages: for each target language, mappings from parse trees of every possible source language to parse trees in the target language are needed.

Our hybrid approach to MT takes an interlingual meaning representation and manipulates it to conform to the requirements of the target language. We call the interlingual representation *source-biased* because it retains some source-language information (e.g., articles) while discarding other information (e.g., word order). We have utilized the natural language subsystem of GALAXY, comprising the language understanding and surface real-

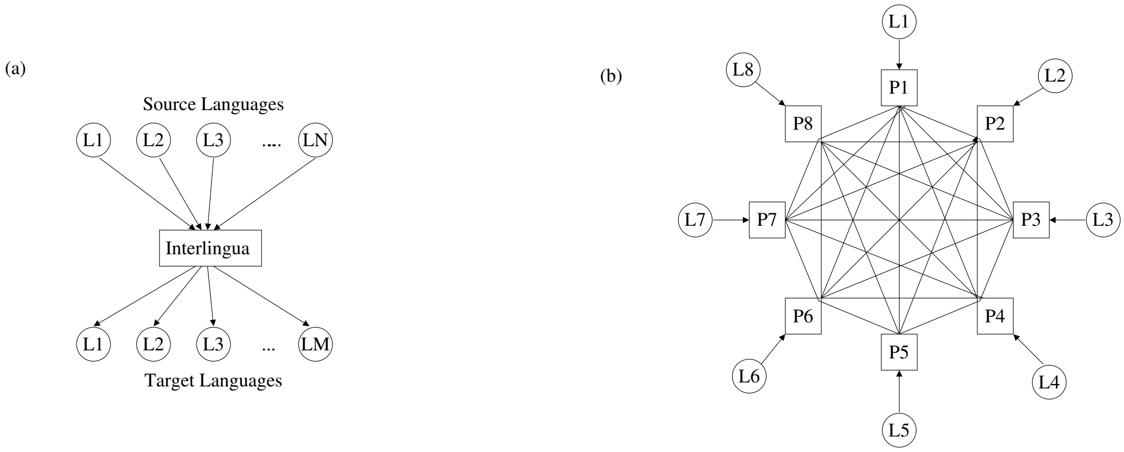

Figure 3-1: (a) In the interlingual approach to MT, all source inputs are mapped to a language independent meaning representation. To produce output in a target language, a mapping from the interlingua to the target language must be provided. (b) In the transfer approach, all source inputs pass through an analysis stage resulting in a source-specific parse ($P1..P8$). To produce output in a target language, a mapping from all possible source parses to a target parse must be provided.

ization components, to implement our hybrid technique.

3.1 Structural and Lexical Transfer

The components of the surface realization module perform three tasks: to modify the interlingual representation to adhere to the constraints of the target language, to choose appropriate words in the target language, and to place the words in a syntactically-appropriate order. PLUTO is responsible for the first task (*structural transfer*),[1] while GENESIS is responsible for the second two (*lexical transfer* and *word ordering*).

The goal of structural transfer is to map a source-biased interlingual representation to a target-specific one ready for generation into the target language. Most language-feature information is encoded in the lexicon, which PLUTO visits to populate the interlingual frame. PLUTO may also alter the structure of the frame, moving frame elements to positions more amenable to target string production. In this model, the string production component assumes that all words that are to appear in the final string exist in the frame. It also assumes that there is sufficient information in the frame to make lexical choices (i.e., to carry out lexical transfer). That is, all information necessary for contextual decision-making about

[1]The structural transfer stage is essentially what has been referred to as *frame modification* throughout this thesis.

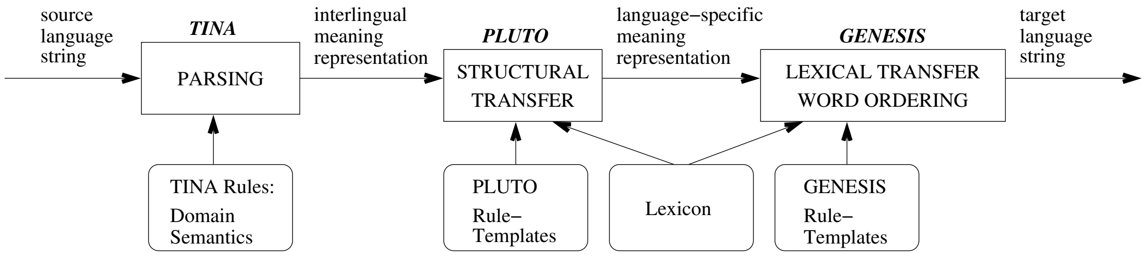

Figure 3-2: A hybrid approach to MT implemented using the Spoken Language Systems group's natural language subsystem. This approach combines the principles of interlingua and transfer.

```
{c yn_question
   :rhet "there"
   :topic {q telephone }
   :pred {p in_loc
            :topic {q room } } } }
```

Figure 3-3: An interlingual linguistic frame derived from a parse of the Chinese string *Fang2 jian1 you3 dian4 hua4 ma5*, or, in English, *Is there a telephone in the room?*

the words to include in the target string should already be explicitly manifest in the frame prior to string production. Figure 3-2 summarizes how our hybrid approach is implemented in the natural language subsystem.

3.2 An Interlingual Meaning Representation

In our model, the linguistic frame output from the language understanding module TINA (see Seneff [34]) serves as an interlingual representation. Our interlingual meaning representation is source-biased, meaning that it preserves source-specific language features. However, its core is language-independent because the analysis leading to its creation is guided by a language-independent ontology and grammar. Developing the interlingual ontology and grammar is made easier by working within constrained domains.

Figures 3-3 and 3-4 depict two interlingual frames that our system could produce. They represent the same concept; however, the first interlingual frame (Figure 3-3) is derived from a parse of the Chinese sentence *Fang2 jian1 you3 dian4 hua4 ma5*, whereas the second is derived from a parse of the English sentence *Is there a telephone in the room?* The core structure is identical in both frames, but the interlingual frame produced from Chinese is lacking some of the features in the interlingual frame produced from English. Specifically, the English frame contains auxiliary, number, person, and quantifier information.

```
{c yn_question
   :aux "link"
   :rhet "there"
   :number "sing_third"
   :topic {q telephone
            :number "sing"
            :quantifier "indef" }
   :pred {p in_loc
           :topic {q room
                    :number "sing"
                    :quantifier "def" } } } }
```

Figure 3-4: An interlingual linguistic frame derived from a parse of the English string *Is there a telephone in the room?*

3.3 Case Study: Using PLUTO for Structural Transfer

We now illustrate the phenomena in an MT task that can be handled by PLUTO in a structural transfer stage via rule-templates. The examples are drawn from a case study performed in the PHRASEBOOK domain for translation into Spanish.[2] The goal of this section is two-fold: first, to give examples of challenges particular to MT, and second, to demonstrate how PLUTO may be used to meet these challenges. Thus, this section also serves as an informal means of evaluating the power of the PLUTO scripting language and of the tool itself.

PHRASEBOOK is a travel-domain multilingual spoken-language translation system. The goal is to allow a traveler to speak an utterance in a source language and have the system translate and repeat the utterance in a target language. The domain comprises many contexts, including giving and getting directions, renting a hotel room, and ordering in a restaurant.

There are many challenges of MT that are manifest in the PHRASEBOOK domain. For example, it is important to structure sentences appropriately for the target language. We saw how PLUTO can be used to effect such structural transformations in the *gustar* example of Section 2.3.2. Here, we focus on other challenges:

- word-sense disambiguation and lexical selection,
- filling in function words, and

[2] The Spanish used is mostly Mexican Spanish, which is the Spanish I know. However, being a native speaker of English, not Spanish, I also had extensive help from Eduardo Torres-Jara, a native speaker of Ecuadorian Spanish, who provided, in many instances, alternative paraphrases for English source strings.

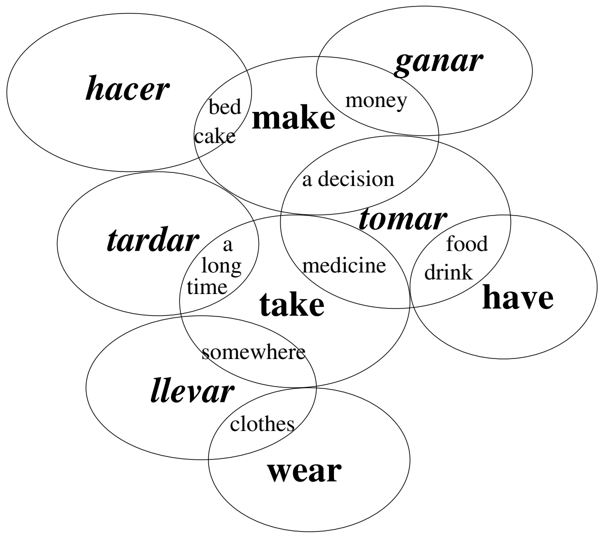

Figure 3-5: Many-to-many mappings between Spanish and English verbs.

- inflecting root forms and choosing appropriate forms for nominal modifiers.

PLUTO may be used in partnership with the lexicon to annotate the interlingual meaning representation with information that can help GENESIS make word choice and word ordering decisions. We explore each of these challenges in turn, focusing on PLUTO's role.

3.3.1 Word-Sense Disambiguation and Lexical Selection

It is a well-known characteristic of natural-language translation that the mappings between lexical items in different languages may be many-to-many. Figure 3-5 shows how two common English verbs – *make* and *take* – each map to several Spanish verbs depending on the context, while the same may be said of two common Spanish verbs – *llevar* and *tomar* – and their mappings to English.

The existence of such mappings leads to a dual problem in MT: word-sense disambiguation and lexical selection. Whenever a word from the source language may take on several meanings, it must be resolved, from the context, which meaning is being employed (*word-sense disambiguation*); similarly, whenever a concept may be expressed in several ways by the target language, it must be determined which way is most appropriate, again given the context (*lexical selection*). While it is, in theory, ideal to handle all word-sense disambiguation during analysis (i.e., while mapping to the interlingual form), in practice it may be the case that the semantic ontology is not sufficiently fine-grained to fully disambiguate every

Verb	Use	Examples
ser	origin, possession, material	I **am** from Canada. **Soy** *de Canadá.*
	inherent qualities: nationality, age, etc.	I **am** American. **Soy** *estadounidense.*
	profession	I **am** a doctor. **Soy** *doctora.*
	time, date, days of the week	What time **is** it? *¿Qué hora* **es***?*
estar	location or position	Where **is** the hotel? *¿Dónde* **está** *el hotel?*
	temporary qualities, subjective impressions	The soup **is** cold. *La sopa* **está** *fría.*
tener	hunger, thirst, age, etc.	I **am** 23. **Tengo** *23 años.*
hacer	weather	It **is** cold. **Hace** *frío.*
haber	rhetorical	**Is** there a television? *¿***Hay** *un televisor?*

Table 3.1: Spanish verbs used to translate the English verb *to be.*

language supported by the system.

For example, in our semantic ontology, any verb like the English verb *to be* is generalized as a *linking verb*. However, Spanish has at least four verbs that can be mapped to *to be*, and the auxiliary value *link* is not alone sufficient to decide which Spanish form to use. Table 3.1 presents some of the differences between five Spanish verbs used to translate *to be*: *ser*, *estar*, *tener*, *hacer*, and *haber*.

PLUTO can be used in partnership with the lexicon to set the linking verb appropriately in linguistic frames to be realized in Spanish. Figure 3-6 shows an excerpt from the Spanish PHRASEBOOK lexicon containing entries with pertinent linking verb information. The frame in Figure 3-8 shows a meaning representation that might be paraphrased as *You are right* in English. The interlingua represents the concept *to be* as a linking verb (the *key:value* pair <`aux:link`> in the frame). In Spanish, a translation of *You are right* is *Tiene razón*. *Tiene* is the third-person singular conjugation of the verb *tener*, or *to have*, and *tener razón* is the expression used in Spanish to say that a person is right. PLUTO has to examine the context of the meaning representation globally to decide which of the five possible linking verbs to use in this case.

The PLUTO rule-templates in Figure 3-7 are used to process the frame in Figure 3-8 and effect the changes in Figure 3-9. Processing begins with the first **goto** command of the *clause_template* rule-template. The *handle_link* rule-template checks to see if the auxiliary has the value `link`, which it does. Processing then proceeds to the *del_link* rule, which remembers the value of the auxiliary in the info-frame and deletes the auxiliary from the linguistic frame. The second command of the *clause_template* processes the topic with the *topic_template* rule-template. This causes number information to be set in the `pronoun` topic; it also causes a selector to be set in the info-frame. Processing of the `quality` predicate is next, handled by the *predicate_template* rule-template. The **$core** command causes an `:aux` key to be set in the predicate with value `link_estar` (from the `quality` entry in the lexicon). Then, the topic of the predicate is processed. The string value `right` causes PLUTO to visit the lexical entry with the same name, and since the selector `$:subj_person` was previously set in the info-frame, PLUTO looks up the `person_right` entry. This entry overwrites the auxiliary previously set in the frame by the `quality` entry, and now the value of the auxiliary becomes `link_tener`. Finally, the *<==:aux* command pulls the `:aux` key from the predicate up to the top level of the frame.

We have chosen in this example to place domain-specific information in the lexicon as opposed to in the PLUTO rule-templates. An alternative approach, which would put domain-specific information in the rule-templates, is to create a case-based rule-template for setting an appropriate auxiliary verb such as

```
set_aux_link  ($if at_age >set_link_tener \
               $else_if estar_preds >set_link_estar ... )
```

where *estar_preds* would explicitly list domain predicates taking the verb *estar*. We choose to avoid this alternative implementation. Doing so allows us to write PLUTO rule-templates that are as domain-independent as possible. Because we keep domain-specific information isolated in the lexicon, this example also illustrates the policy for generation that we outline in Chapter 4.

3.3.2 Filling in Function Words

PLUTO may be used to explicitly annotate a linguistic frame with function words, such as articles. For example, in Spanish, the definite article is required when the meaning of a

```
at_age             O   ""             :aux "link_tener"
at_loc             O   "en"           :aux "link_estar"
available          A1  "disponible"   :aux "link_estar"
body_view          O   ""             :aux "link_estar"
direction_right    N   "derecha"      :gender "f" :aux "link_estar"
how_far            O   "cuán lejos"   :aux "link_estar"
included           A   "incluid"      :aux "link_estar"
left               N   "izquierda"    :gender "f" :aux "link_estar"
lost               A   "perdid"       :aux "link_estar"
on                 O   "en"           :aux "link_estar"
over_there         O   "allá"         :aux "link_estar"
person_right       N   "razón"        :aux "link_tener"
quality            O   ""             :aux "link_estar"
ready              A   "list"         :aux "link_estar"
rhetorical_there   O   ""             :aux "link_haber"
right              O   "correcto"     $:direction "!direction_right" \
                                      $:subj_person "!person_right"
state_of_being     O   ""             :aux "link_estar"
straight           O   "derecho"      :aux "link_estar"
there              O   "allá"         $:rhetorical "!rhetorical_there"
where              O   "dónde"        :aux "link_estar"
you                N   ""             :number "third" ; $:subj_person
```

Figure 3-6: This excerpt from the Spanish lexicon shows entries with information about the kind of linking verb to set in the linguistic frame.

```
clause_template     >handle_link :topic :pred <==:aux
handle_link         ($if :aux == "link" >del_link)
del_link            ($set ^:aux "link") ($del :aux)
topic_template      :name
predicate_template  $core :topic
```

Figure 3-7: These PLUTO rule-templates set a linking verb in partnership with the lexicon.

```
{c statement
   :topic {q pronoun
             :name "you" }
   :mood "finite"
   :number "pl"
   :aux "link"
   :pred {p quality
            :topic "right" } }
```

Figure 3-8: The input frame to PLUTO contains an underspecified auxiliary verb with respect to Spanish.

```
{c statement
   :topic {q pronoun
            :name "you"
           *:number "third" }
   :mood "finite"
   :number "pl"
  *:aux "link_tener"
   :pred {p quality
           :topic "right" } }
```

Figure 3-9: The output frame from PLUTO contains a more specific auxiliary verb, appropriate for output in Spanish.

`like_preds`	`like interest love`
`like_preds_template`	`... >set_topic_quant_def ...`
`set_topic_quant_def`	`($set :quantifier[:topic] "def")`

Figure 3-10: Excerpt from rule-templates for setting a definite article for use with a noun in the general sense in Spanish.

sentence implies a general use of the noun.[3] Thus, the Spanish translation of the English phrase *I like fish* is *Me gusta el pescado.* The definite article *el* appears in the Spanish but not in the English sentence, although the sentences are equivalent in meaning. The rule-templates in Figure 3-10 implement this use of the definite article for verbs expressing *like*, *interest*, or *love* by setting a definite article in the predicate's topic. These rule-templates assume that in the PHRASEBOOK domain we will only encounter statements of general liking.

The essence of the approach embodied by these rules is to group predicates that exhibit a particular behavior and then to annotate the frame with the features associated with this group. Note that in this example, we allow lexical information to be expressed in the PLUTO rule-templates. An alternative approach would be to set a definite quantifier feature in the lexicon for each verb expressing *like*. However, if we did that, the quantifier would be set in the predicate frame as opposed to its topic. This introduces an added complexity that we choose to avoid. An alternative that would also be compatible with the policy of maintaining domain-specific information in the lexicon would be to implement more specific feature-placement mechanisms directly in the lexicon, i.e., to allow lexical entries such as those in Figure 3-11.

[3] This example is the inverse of the example from Section 2.3.1, in which the target was English and we wanted to delete a definite article.

```
interest  V1  "interes"  :quantifier[:topic] "def"
like      V1  "gust"     :quantifier[:topic] "def"
love      V1  "encant"   :quantifier[:topic] "def"
```

Figure 3-11: An imaginary lexicon with more specific feature-placement mechanisms. This would set a definite quantifier directly in a topic frame contained in any one of the three predicates listed, whenever PLUTO visited their lexical entry.

3.3.3 Inflecting Root Forms and Choosing Forms for Modifiers

PLUTO may also be used to set information, like gender and number, for inflecting root forms of verbs and choosing appropriate forms for the modifiers of nouns. In Spanish, there are many different tenses and moods. In PHRASEBOOK, unlike other domains we have developed, many of these tense/mood combinations occur in the kinds of sentences we would like to be able to support. For example, in our JUPITER weather information domain, the tense is generally either *present indicative* or *future indicative*. In contrast, PHRASEBOOK thus far (in the development phase) exhibits nine different tense/mood combinations, summarized in Table 3.2. Although the majority (72%) of the sentences in our development phase are in the present indicative tense, we want to be able to support inflection for a wide range of possible tenses and moods.

In Spanish, conjugated verbs agree in number with the subject of the sentence: there are six[4] conjugations for each tense/mood combination. Also, adjectives agree in number (singular or plural) and gender (male or female) with the nouns they modify.

The rule-templates in Figure 3-12 set number and gender information in a linguistic frame in a way appropriate to Spanish. The two basic strategies of the rule-templates are (1) get number and gender information from the top-level topic and propagate it to the top level of the frame and its predicate(s), and (2) set number and gender information in all topics and propagate that information to their modifier children.

An example will help elucidate these rules. Figure 3-13 depicts an input frame to PLUTO, and Figure 3-15 shows the same frame after being processed using the rule-templates in Figure 3-12 and the lexicon excerpt in Figure 3-14. Number and gender information is set in the `pronoun` topic by the **keyword** command *:topic* in the *clause_template* rule. This

[4]1st person, 2nd person, and 3rd person, both singular and plural.

Mood	**Tense**	**Examples from PHRASEBOOK**
Indicative	Present Simple	Can we have the check please? *Nos trae la cuenta, por favor?*
	Present Progressive	What are you trying to buy? *Qué está tratando de comprar?*
	Future: Ir + a	Is it going to rain today? *Va a llover hoy?*
	Future Simple	Hang on I'll ask someone. *Un momento preguntaré a alguien.*
	Preterite	I already reserved a room. *Ya reservé una habitación.*
	Perfect	My wallet has been stolen. *Mi billetera ha sido robada.*
	Perfect Progressive	We have been waiting for a long time. *Hemos estado esperando por mucho tiempo.*
Imperative		Please give it to me. *Démelo, por favor.*
Conditional		I would like to go to a Greek restaurant. *Me gustaría ir a un restaurante griego.*

Table 3.2: There are nine different tense/mood combinations manifest so far in our PHRASEBOOK domain.

```
clause_template      ... :topic >cls_propagate_num >cls_propagate_gen \
                     :pred ...

cls_propagate_num    ($clone :number[:topic]) \
                     ($softset :number "third") >set_child_num
cls_propagate_gen    ($clone :gender[:topic]) ($softset :gender "m") \
                     >set_child_gen

topic_template       ... (:name $core) \
                     >top_propagate_num >top_propagate_gen ...

top_propagate_num    ($softset :number "third") \
                     >set_child_num
top_propagate_gen    ($softset :gender "m") >set_child_gen

set_child_num        ($set :number[:pred] :number)
set_child_gen        ($set :gender[:pred] :gender)

predicate_template   ... :topic ...
```

Figure 3-12: PLUTO rule-templates for setting number and gender features in Spanish.

```
{c yn_question
   :aux "do"
   :topic {q pronoun
            :name "you" }
   :pred {p possess
           :topic {q size
                     :quantifier "indef"
                     :pred {p sized
                             :topic "smaller" } } } }
```

Figure 3-13: Frame representing a string such as *Do you have a smaller size?*, before being processed by PLUTO.

size	N	"talla" :gender "f"
you	N	"" :number "third"

Figure 3-14: Lexicon entries *size* and *you*.

rule looks up the entry for `you` in the lexicon and finds its number is `third`.[5] Next, the rule-template *cls_propagate_num* clones this information and places it in the top-level frame of the `possess` predicate. The rule-template *cls_propagate_gen* first sets a default value for gender (since there was no gender information set from the lexicon) and then propagates that information to the predicate.

When the `size` topic frame inside the `possess` predicate frame is processed by the *topic_template*, the lexicon is again visited, this time retrieving gender information from the `size` entry. Default number information (i.e., `third`, which is interpreted as singular) is set by the *top_propagate_num* rule, which also sets number information in the predicate `sized`. Finally, the gender information gleaned from the lexicon is propagated to the child predicate as well.

3.4 Summary

In this chapter, we have shown that our new architecture for the surface realization component may be used to establish a framework for MT that draws on two prominent approaches: interlingua and transfer. Furthermore, we have presented some in-depth examples of how PLUTO can be used to carry out structural transfer on source-biased interlingual frames. As we implied throughout the presentation of these examples, we have in mind a particular

[5]This is adhering to a convention that, unless otherwise indicated, number is singular.

```
{c yn_question
   :topic {q pronoun
             :name "you"
            *:number "third" }
  *:number "third"
  *:gender "m"
   :pred {p possess
            :topic {q size
                      :quantifier "indef"
                     *:gender "f"
                     *:number "third"
                      :pred {p sized
                               :topic "smaller"
                              *:gender "f"
                              *:number "third" } }
           *:number "third"
           *:gender "m" } }
```

Figure 3-15: Frame representing a string such as *Do you have a smaller size?*, ready for generation in Spanish. A possible output string is *¿Tiene una talla más pequeña?* The indicative verb form *tener* has been conjugated using the number (i.e., *third* information, and the adjective *pequeña* (*small*) is in the feminine singular form, like the noun (i.e., *talla*, or *size*) it modifies.

partnership between PLUTO, GENESIS, and the lexicon they share. The policy we develop in Chapter 4 advocates the localization of as much domain-specific information as possible in the lexicon, using PLUTO only to put that information into linguistic frames in the appropriate position. Our experimentation with PLUTO thus far has gradually led us to this use of the components of the surface realization subsystem. We find this approach beneficial because it allows for more general, domain-independent use of the PLUTO and GENESIS rule-templates. Thus, we find that the new architecture of the system helps bring us closer to one of our most important goals: making our rule- and template-based system more manageable from the point of view of development.

Chapter 4

A Policy for Multilingual Generation

There are many ways of utilizing the components of the NLG subsystem of the Spoken Language Systems group to build multilingual natural language applications. For instance, it is possible to imagine the PLUTO rule-templates encoding structural decision-making by directly testing for the existence of specific semantic entities. We saw an example of such an approach in Section 3.3.2: the presence of certain predicates (e.g., *like* and *interest*) triggered the setting of a quantifier in the topic from the PLUTO rule-templates. In general, however, we advocate a policy in which domain-dependent semantic information is contained entirely within the lexicon. Section 3.3.1, with its example for choosing an appropriate Spanish linking verb, adhered to this approach, as did the example for setting gender and number in Section 3.3.3. In this chapter, we develop a policy for generation that embodies this approach using PLUTO, GENESIS, and the lexicon in partnership.

4.1 The Partnership

The partnership we advocate between PLUTO, GENESIS, and the lexicon establishes a contract: the role of the lexicon is to encode domain-specific lexical information; the role of PLUTO is to populate meaning representations with this information; and the role of GENESIS is to form a string from the meaning representation in the context of domain-specific lexical information. This partnership is summarized in Figure 4-1.

An ever-present goal driving our language generation research is to enable the devel-

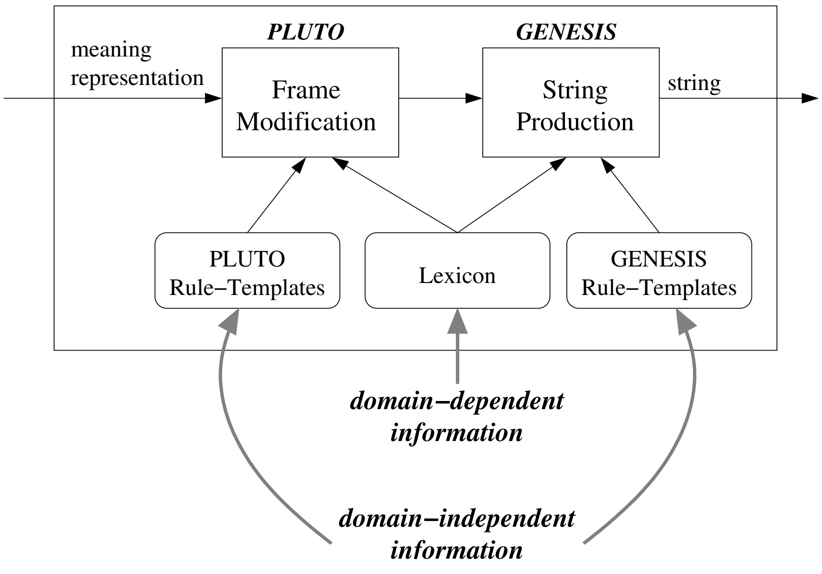

Figure 4-1: PLUTO, GENESIS, and the lexicon are partners in a contract: the lexicon encodes domain-specific information that PLUTO uses to perform frame modification; GENESIS uses the enhanced frame to produce an appropriate string.

opment of domain-independent linguistic resources. The ability to encode such resources allows for reuse across disparate domains. Research in spoken dialogue systems is often carried out in multiple constrained domains. For example, the Spoken Language Systems group has built systems in the weather information domain (JUPITER) [43], the air-travel reservation domain (MERCURY) [33] [36], the flight status domain (PEGASUS) [45], and the urban navigation domain (VOYAGER) [14], among many others. One reason for working in limited domains is to restrict the vocabulary to a reasonable subset of all possible words in a given natural language: automatic speech recognition relies heavily on statistical models, and its performance degrades rapidly in the presence of out-of-vocabulary words [7] (i.e., words seen in testing but not in training). Developing new systems in novel domains helps push our research forward because it encourages us to test new ideas and introduces new challenges. It also links our research to practical problems. If the surface realization subsystem is easy to develop, this facilitates the rapid deployment of systems in new domains.

Research in multilingual spoken dialogue systems also benefits greatly from domain-independent linguistic resources. The Spoken Language Systems group has traditionally been very interested in multilingual research (e.g., [22] [10] [42] [44] [15]). The more domain-independent our linguistic resources can be, the easier it is to make new systems multilingual by adapting the existing resources. The combination of the new architecture of our surface realization subsystem and the policy we define helps to make our linguistic resources more domain-independent, thereby facilitating the development of multilingual systems.

4.2 How to Use the New System: Examples

We present three examples – from French, Chinese, and English – to demonstrate the policy we wish to enforce. In contrast to the examples from Chapter 3, the examples in this chapter showcase the workings of the whole generation subsystem and not only PLUTO.

4.2.1 French

In this example, we demonstrate how our policy works with a translation of the phrases *there is* or *there are*, and *is there* or *are there* into French. We start with the statement forms (*there is* and *there are*) and show how a simple modification to the word ordering rules in GENESIS is sufficient to translate the interrogative forms. In French, the translation of both *there is* and *there are* is *il y a*, where the French words have roughly the following English correspondences:

il	y	a
he	there	has

Figure 4-2 shows an input frame to the surface realization system. In English, this frame might be paraphrased with the sentence *There is a telephone in the room.* However, for output in French, this frame is incomplete: it is missing explicit subject information (the word *il*, or *he*); its auxiliary verb *link* is underspecified; and it is missing number and gender information. In this example, we focus on the first two problems and assume that other rules in the PLUTO rule-template file fill in the number and gender features shown in the PLUTO output frame of Figure 4-5.

The role of PLUTO in this example is simply to visit the lexicon and insert linguistic information into the frame. The rule-template file in Figure 4-3 shows a *clause_template* rule that processes the input frame of type *clause* from Figure 4-2. Among its commands is a **keyword** command that causes PLUTO to visit the lexical entry for `there`, the value of the `:rhet` key. This entry, shown in Figure 4-4, contains four pieces of information: the default generation string, an alternative generation string, a pronoun, and a selector. The

```
{c statement
   :rhet "there"
   :aux "link"
   :topic {q telephone
            :quantifier "indef"
            :pred {p in
                     :topic {q room
                               :quantifier "def" } } }
```

Figure 4-2: An input frame to the surface realization system. A possible paraphrase, in English, is *There is a telephone in the room.* This frame is underspecified with respect to output in French.

clause_template ... :rhet ...

Figure 4-3: An excerpt from a French PLUTO rule-template file.

first two pieces of information have no effect on PLUTO's behavior, but the third piece causes PLUTO to add the *key:value* pair <`:pro, il`> to the input frame, and the fourth piece of information causes PLUTO to set a vocabulary selector in the info-frame.[1]

The frame in Figure 4-5 is now ready for processing by GENESIS.[2] The *statement* rule-template in the GENESIS file in Figure 4-6 is responsible for generating the phrase *il y a* from the frame. First, it generates the value of the keyword `:pro`, which is `il`. Since there is no lexical entry for this word, the sub-string *il* is inserted into the target string. Next, GENESIS looks up the lexical entry for `there`, the value of the keyword `:rhet`. It chooses the default string *y* to add to the target string because there is no selector called `$:prepobj` that has been set; it also sets the selector at the end of the entry (`$:have`) in the info-frame.[3] Finally, it goes to the lexical entry for `link`, where it is redirected to the *have* entry by the "!" syntax. This redirection is due to the `$:have` selector. The number information set in the top level of the frame causes GENESIS to select the correct conjugated form (*a*) to add to the target string. We assume other rules in the GENESIS rule-template file that generate the remainder of the target string: *Il y a une téléphone dans la chambre.* Note that all lexical information in this example is contained in the lexicon, delegating to PLUTO the role of inserting features into the frame, and to GENESIS the role of using this

[1]This last bit of information is not used by PLUTO in this example; rather, it is used in the string production stage by GENESIS.

[2]We assume the existence of PLUTO rule-templates that visit the lexical entries for the words *he*, *telephone*, and *room* to supply appropriate number and gender information.

[3]Note that the info-frame used by GENESIS is distinct from the one used by PLUTO.

```
have   X  "avoir" FIRST "ai" SECOND "as" THIRD "a" \
          PL_FIST "avons" PL_SECOND "avez" PL_THIRD "ont" \
          FUTURE_FIRST "aurai" FUTURE_SECOND "aura"
link   X  "être" $:money "!cost" $:have "!have" \
          FIRST "suis" SECOND "es" THIRD "est" PL_FIRST "sommes" \
          PL_SECOND" "êtes" PL_THIRD "sont"
there  0  "y" $:prepobj "la bas" :pro "il" ; $:have
```

Figure 4-4: An excerpt from a French lexicon.

```
{c statement
   :rhet "there"
  *:pro "il"
   :aux "link"
  *:number "third"
   :topic {q telephone
             :quantifier "indef"
            *:gender "f"
             :pred {p in
                      :topic {q room
                                :quantifier "def"
                               *:gender "f" } } } }
```

Figure 4-5: The frame from Figure 4-2 after processing by PLUTO. The new information in the frame is the (*:pro, il*) *key:value* pair and number information at the top-level, and the gender of *telephone* and *room* in nested frames.

information to produce a string.

A reordering of the commands in the GENESIS rule-template file allows the system to produce the interrogative forms *is there* and *are there*, translated into French as *y a-t-il.* The *t* that appears between the verb (*a*) and subject (*il*) reflects the pronunciation of the phrase and is linguistically non-functional. We will see how a postprocessing stage is used to make this superficial modification to the target string.

The frame in Figure 4-7 is the input to the system. This frame is like the one in Figure 4-2 except that it represents a *yes-no question*, rather than a *statement*. An English paraphrase of this frame could be *Is there a telephone in the room?* Like the first frame, this frame is lacking a subject pronoun, sufficient auxiliary information, and number and gender information. It also undergoes processing by PLUTO, which uses the same rule-template (from Figure 4-3) to produce the modified frame of Figure 4-8.

The GENESIS rule-templates of Figure 4-6 handle the frame in Figure 4-8; this time, the rule-template named *yn_question*, matching the frame name, is selected to generate a string.

`yn_question`	`... :rhet :aux :pro ...`
`statement`	`... :pro :rhet :aux ...`

Figure 4-6: GENESIS rule-templates for producing strings *il y a* (*there is* and *there are*) and *y a-t-il* (*is there* and *are there*).

```
{c yn_question
   :aux "link"
   :rhet "there"
   :topic {q telephone
            :quantifier "indef"
            :pred {p in
                     :topic {q room
                               :quantifier "def" } } } }
```

Figure 4-7: This frame, input to PLUTO, is underspecified with respect to French. PLUTO uses its rule-templates and the lexicon to add a subject pronoun as well as number and gender information.

The word ordering specified by this rule-template causes the production of the target string *y a il.* The rewrite rules of Figure 4-9 cause this output to be rewritten with the correct surface form (*y a-t-il*). These rules are contained in a separate file and are executed in a separate postprocessing stage, following GENESIS string production. The final target string is *Y a-t-il une téléphone dans la chambre?*

4.2.2 Chinese

Our previous example showed how we could produce French phrases using the lexicon to encode linguistic information. Our next example uses the lexicon to encode word-class information for surface realization of predicates in Chinese. In Chinese, noun phrases may be associated with a particular *word-class context* which depends on the main noun being modified. For example, in the noun phrase *Yi1 ben3 shu1*, or *a book* in English, the main noun *shu1* (*book*) sets the word-class context for the phrase. This word-class determines the form of the particle *ben3*. More generally, the word-class context can determine the form of certain other words within the scope of the noun phrase, such as predicates containing the noun phrase. Here, we see how we can encode this information in our surface realization system while adhering to our policy of isolating domain-specific information in the lexicon.

We use the frame in Figure 4-10 to show how we achieve our goal. In English, this frame could be paraphrased as *Please take me to the airport.* However, the input frame

```
{c yn_question
   :aux "link"
  *:number "third"
   :rhet "there"
  *:pro "il"
   :topic {q telephone
             :quantifier "indef"
            *:gender "f"
             :pred {p in
                      :topic {q room
                                :quantifier "def"
                               *:gender "f" } } } }
```

Figure 4-8: The frame output by PLUTO contains a subject pronoun (*il*) as well as number and gender information.

```
"y a il"  "y a-t-il"
```

Figure 4-9: Rewrite rules executed in a postprocessing stage account for non-linguistic surface phenomena.

does not contain any of the word-class context information necessary for string production in Chinese. Specifically, the correct form of the predicate *take*, *song4*, needs to be chosen. In this case, it is the direct object pronoun *me* that establishes the word-class context. PLUTO's role is to fill in this missing word-class contextual information. In the example, we say that the word *me* establishes a *person* word class. It is the lexicon that stores this information. The *topic_template* rule-template in Figure 4-11 tells PLUTO that, in each topic frame, it should search the lexicon for an entry matching the value of the `:name` key, if it exists, or the frame name, otherwise. Doing so sets the word-class context, if one exists, for each noun phrase.

There are two nested topic frames in the input frame, both with a `:name` key. The `pronoun` topic frame contains a `:name` key whose value is `me`; the corresponding lexical entry (in Figure 4-12) causes PLUTO to insert a *key:value* pair <`:word-class, *person*`> into the frame. This defines the word-class context for the `pronoun` frame; this word-class context will be used by GENESIS (in conjunction with the lexicon) to produce an appropriate string. The second topic frame, named `a_location`, contains a `:name` key whose value is `airport`. Since the lexical entry corresponding to `airport` does not contain word-class information, none is added to this topic frame. The frame in Figure 4-13 shows the modification made by PLUTO during the preprocessing stage.

```
{c request
   :politeness "please"
   :pred {p take
            :topic {q pronoun
                      :name "me" }
            :pred {p to_place
                     :prep "to"
                     :topic {q a_location
                               :name "airport"
                               :quantifier "def" } } } }
```

Figure 4-10: The PLUTO input frame contains no word-class context information and is not ready for string production in Chinese. PLUTO visits the head noun of each topic frame, which results in the appropriate word-class context being filled in for each noun phrase.

```
topic_template  ... (:name $core) ...
```

Figure 4-11: An excerpt from a Chinese PLUTO rule-template file.

```
;; word-class features
*person*              O   "" ; $:person
*self*                O   "" ; $:person $:self
*money*               O   "" ; $:money
*space*               O   "" ; $:space
*food*                O   "" ; $:food
*loc*                 O   "" ; $:loc
*weather*             O   "" ; $:weather
*utensil*             O   "" ; $:utensil
*vehicle*             O   "" ; $:vehicle
*tea*                 O   "" ; $:tea
*move*                O   "" ; $:move
*act*                 O   "" ; $:act
*item*                O   "" ; $:item

;; lexical entries
airport               N   "ji1_chang3"
credit_card           O   "xin4_yong4_ka3" :word-class "*money*"
me                    O   "wo3" :word-class "*person*"
take                  O   "yao4" $:vehicle "zuo4" $:person "song4" \
                          $:money "shou1" $:move ""
```

Figure 4-12: An excerpt from a Chinese lexicon. The key `:word-class` represents word-class information for noun phrases.

```
{c request
   :politeness "please"
   :pred {p take
            :topic {q pronoun
                      :name "me"
                     *:word-class "*person*" }
            :pred {p to_place
                     :prep "to"
                     :topic {q a_location
                               :name "airport"
                               :quantifier "def" } } } }
```

Figure 4-13: The modified output frame from the PLUTO preprocessing stage. Word-class information has been added to the `pronoun` topic frame.

clause_template	... :pred ...
predicate_template	:word-class <==:word-class $core

Figure 4-14: An excerpt from a Chinese GENESIS rule-template file.

The frame output by PLUTO is next processed by GENESIS, using the rule-templates in Figure 4-14 and the lexicon (Figure 4-12). In each predicate, GENESIS pulls any word-class context information in the predicate's descendants up to the level of the predicate so that it can be used there to select appropriate lexical forms. The important rule-template for the purposes of this example is the *predicate_template*, which processes the `take` predicate. The first two commands of this rule-template allow GENESIS to use the word-class context set during preprocessing to select an appropriate realization of the verb *take*. Since there is no `:word-class` key at the top level of the predicate frame, it is the second command that accesses this information. The **pull** command reaches down into the topic frame to find the `:word-class` key there and searches the lexicon for an entry matching its value (`*person*`). GENESIS finds this entry among several such entries in the lexicon. The `*person*` entry sets a selector `$:person` in the info-frame. Then, when the **$core** command executes and GENESIS has to choose an appropriate realization of this word, the `$:person` selector causes the selection of a form appropriate to the word-class context (*song4*), overriding the default string, *yao4*.

One more example shows how the word-class contexts determined by noun phrases can be used to produce appropriate Chinese strings in our surface realization system, using the policy we have been advocating throughout this chapter. The PLUTO output frame in Figure 4-16 is the modified version of the input frame in Figure 4-15 in which the word-

```
{c yn_question
   :aux "do"
   :topic {q pronoun
            :name "you" }
   :pred {p take
           :topic {q credit_card
                     :number "pl" } } }
```

Figure 4-15: An input frame to PLUTO containing no word-class context information. This frame can be paraphrased *Do you take credit cards?* in English.

```
{c yn_question
   :aux "do"
   :topic {q pronoun
            :name "you"
           *:word-class "*person*" }
   :pred {p take
           :topic {q credit_card
                     :number "pl"
                    *:word-class "*money*" } } }
```

Figure 4-16: The output frame contains word-class information sufficient for selecting a form of the word *take* appropriate to a *money* word-class context.

class context for each topic frame has been filled in using the rule-template and lexicon (Figures 4-11 and 4-12, respectively). This frame can be paraphrased in English with the sentence *Do you take credit cards?* In this case, the form of *take* chosen by GENESIS in the lexicon is *shou1*, appropriate in a *money* word-class context.

4.2.3 English

In Section 3.3.2, we presented an example for Spanish generation that deviated from our policy of maintaining lexical information strictly in the lexicon. Here, we present an example involving generation into English that is very similar to that example. We contrast these examples to emphasize when it might be appropriate to break from the policy.

The example we develop here demonstrates how we can fill in missing auxiliary information based on the value of the frame's predicate, once again using the lexicon to encode linguistic information. The frame in Figure 4-17 is missing auxiliary information with respect to English and might produce the incorrect form *When will it ready?* We want to write some general rules that will fill in a missing linking verb *to be* when the auxiliary *will* is present and the predicate is an adjective that we know exists in the domain lexicon.

The rule-templates in Figure 4-18, use the lexicon to fill in the missing auxiliary infor-

```
{c wh_question
   :aux "will"
   :topic {q pronoun
            :name "it" }
   :mode "root"
   :pred {p ready
           :trace "when" } }
```

Figure 4-17: This PLUTO input frame is missing auxiliary information. PLUTO fills it in using its rule-templates in conjunction with the lexicon.

`clause_template`	`... :pred >soft_aux ...`
`predicate_template`	`$core`
`soft_aux`	`($if :aux == "will" >set_aux2 >set_aux)`
`set_aux`	`($softset :aux ^:aux)`
`set_aux2`	`($softset :aux2 ^:aux)`

Figure 4-18: An excerpt from an English PLUTO rule-template file.

mation appropriately. The *clause_template* triggers the processing of the `ready` predicate frame, which in turn causes PLUTO to visit the lexical entry for `ready`. This entry, along with other domain adjectives, causes an info-key to be set in the info-frame. The value of this info-key is `link` and suggests that a linking verb should be used, if needed, with such adjectives.

The second command in the *clause_template* causes the rule-template *soft_aux* to execute. Since the test of the single command contained in this rule-template succeeds, the rule-template called *set_aux2* is executed. This rule-template causes a *key:value* pair <`:aux2, link`> to be set in the frame. It is now trivial for the GENESIS rule-template *wh_question*, shown in Figure 4-21 to generate the string *When will it be ready?*

The frame in Figure 4-23 is the modified version of the frame in Figure 4-22, subsequent to PLUTO processing. In this example, the same rules as before (i.e., the PLUTO rule-templates in Figure 4-18) are used to fill in auxiliary information, only this time, because there is no auxiliary `will` in the frame, the test in the *soft_aux* rule fails, and it is an `:aux` key rather than an `:aux2` key that is set in the frame with the information gleaned from the lexicon. The frame, when processed by GENESIS, has paraphrase *When is room 245 available?*

This example contrasts with the example presented in Section 3.3.2. In that example, we put lexical information into the PLUTO rule-template file because it was difficult to place

allergic	A	"allergic"	^:aux "link"
available	A	"available"	^:aux "link"
closest	A	"closest"	^:aux "link"
ready	A	"ready"	^:aux "link"
vegetarian	A	"vegetarian"	^:aux "link"

Figure 4-19: An excerpt from an English lexicon. Domain adjectives set an info-key in the info-frame which then causes PLUTO to set a second auxiliary in the frame in the presence of an auxiliary *will.*

```
{c wh_question
   :aux "will"
  *:aux2 "link"
   :topic {q pronoun
             :name "it" }
   :mode "root"
   :number "third"
   :pred {p ready
            :trace "when" } }
```

Figure 4-20: The PLUTO output frame contains a second auxiliary. The frame can now easily be paraphrased as *When will it be ready?*

```
wh_question  <==:trace :aux :topic :aux2 :pred
```

Figure 4-21: An excerpt from a GENESIS rule-template file.

```
{c wh_question
   :topic {q room
             :pred {p post_number
                      :topic 245 } }
   :pred {p available
            :trace "when" } }
```

Figure 4-22: An input frame to PLUTO, missing auxiliary information.

```
{c wh_question
   :topic {q room
             :pred {p post_number
                      :topic 245 } }
  *:aux "link"
   :pred {p available
            :trace "when" } }
```

Figure 4-23: PLUTO picks up the appropriate auxiliary information from the lexicon and then uses it to fill in a missing auxiliary.

it from the predicate, which contained the semantically-relevant information, into the topic directly from the lexicon. Hence, we resorted to grouping these predicates in PLUTO, where we have more control over placement of keys. In the example we just developed, we also could have grouped the domain adjectives in the PLUTO rule-template file. However, we chose to put this information in the lexicon, thereby adhering to our stated principle of encoding as much domain-specific lexical information as possible in the lexicon, leaving the PLUTO and GENESIS rule-templates as domain-independent as possible.

In the next and final chapter, we give a formal evaluation of PLUTO and present our conclusions and future directions for this work.

Chapter 5

Conclusions

5.1 Evaluation

The formal evaluation of NLG systems has been the subject of much discussion in the literature. For instance, Mellish and Dale [11] offer an overview of the problem. First, they note that NLG encompasses a wide range of subtasks, including content determination, sentence planning, and surface realization. Hence, one of the most important decisions to be made when evaluating NLG is whether to evaluate the entire system (*black box evaluation*) or the individual components (*glass box evaluation*). In the case of the Spoken Language Systems group's GALAXY architecture, a black box evaluation of the NLG system would include the dialogue manager, context resolution server, and surface realization component. But because this thesis focuses on the surface realization component, we have looked for ways to formally evaluate this component alone.

According to Mellish and Dale, another important question to ask concerning evaluation is *What should be measured?* We have decided to evaluate the speed of our surface realization component. In a spoken dialogue system, it is essential that the surface realization component work in a fraction of real time, since it is the system as a whole that must operate in real time. In addition to attaining an absolute measure of the speed of this component, we aim to measure the effect of the new architecture on the speed of the system.

In the next three sections, we first report the results of our timing experiments. Next, we discuss some remaining aspects of our system that we would like to evaluate more formally, including the coverage of our grammars and the quality of the system's output in unseen circumstances. Finally, we give an informal evaluation of our new system, emphasizing the ways in which PLUTO has already been employed in several applications including machine translation and query-response.

5.1.1 Timing

Processing time is a valuable, shared commodity in any real-time system. One goal in spoken dialogue systems is to minimize the time it takes for the system to respond to user queries. Thus, it is important to analyze the effect on processing time when a change to the system is introduced. Adding a new component to a system may increase the processing time of that system. Our experiments reveal that this is the case in our new system: in adding a second pass through the frame, processing time has increased.

One question to ask is *By how much has it increased?* To answer this question, we gathered a set of 355 English sentences with adequate (as judged by the author) Spanish translations in both the old and new systems. We then generated a linguistic frame for each sentence and ran each frame through the surface realization subsystem to estimate the processing time in each system configuration. Since the granularity of the system clock was too coarse to measure the processing time of single frames, we measured the total time it took the system to process each frame 100 times,[1] and then divided by 100 to estimate the average time for each frame. The experiment was conducted on an 850-MHz Pentium III machine.

Figure 5-1 shows the results of this experiment. The histogram plots the log of the ratio (new:old) of the times for corresponding sentences in the new and old systems. The median of this distribution is 0.2159, which, when exponentiated, is 1.2410. This indicates that our current implementation of the new system takes about 1.24 times longer to process sentences than our implementation of the old system.

Another question to ask about timing is *Approximately how much processing time does the surface realization subsystem take in the new system?* We use the same set of 355 sentence used in the first timing experiment to answer this question. On average, the new system processes a sentence of between 1 and 13 words in length (an average length of 5 words) in 10.8 ms. Our search of the literature has found only two other sources

[1]We used a call to `clock()`, a C function that estimates the processor time used by the program.

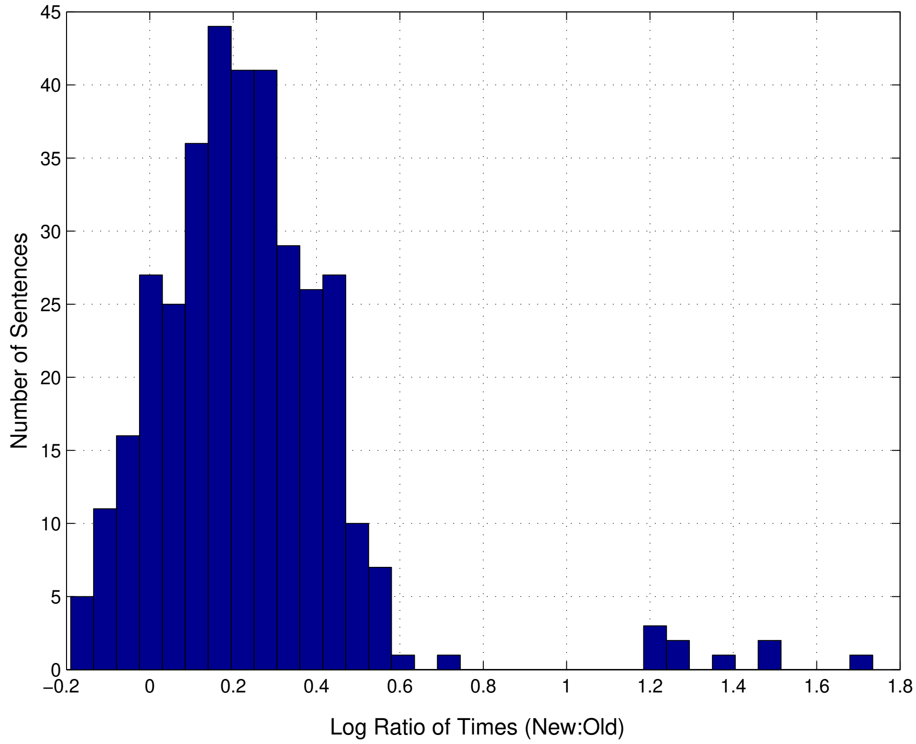

Figure 5-1: The graph plots the log ratio of times (in milliseconds) in the new system vs. the old system against the number of sentences. The new system takes an average of 1.24 times longer than the old system.

of comparison. Oh and Rudnicky [26] report an average generation time in their hybrid (template-stochastic) system of about 200 ms for utterances of around 10-20 words long on a 400-MHz Pentium machine. Bangalore and Rambow [2] report an average time per sentence of 517 ms in their stochastic tree-based generator FERGUS. The authors do not state the type of machine they used to conduct their experiments.

5.1.2 Coverage and Quality

There are some remaining aspects of our system that we would like to evaluate more formally, including the coverage of our grammars and the quality of the system's output. One way to test these aspects of the system is to run it on a set of novel test sentences and evaluate the results. Mellish and Dale [11] suggest two metrics for measuring the quality of such output: *accuracy* (how well the generated utterance conveys the desired meaning) and *fluency* (the readability of the output). Bangalore, Rambow, and Whittaker [3] suggest some automatically-derived metrics that correlate with human judgments of quality and understandability. However, their metrics only work for generators with syntactic input, and the linguistic frames used in this thesis are peppered with both syntactic and semantic information.[2] Alternatively, it would be possible to ask human judges to rate the quality of the output.

The reason we do not include a formal evaluation of these aspects of the system in this work is that we have not yet implemented several comprehensive grammars that adhere to the grammar-writing policy we developed in Chapter 4 for using our surface realization component.[3] We advocate this policy because we believe it will lead to robust and reusable grammars. Hence, any test of the coverage of our grammars and of the quality of the system's output should also test this approach to grammar-writing, since the system itself does not sufficiently constrain grammar-writing. Also, it is important that when we do formally evaluate the coverage and quality of the grammars, we have several grammars available to us: one must keep in mind that an evaluation of our system via an evaluation of the grammars is inherently subject to the idiosyncrasies of the grammar-writer. Having several grammars written by different people all following the same principles of grammar writing will help to normalize the results of such an evaluation.

5.1.3 Informal Evaluation

Although this work has not yet been fully evaluated in a formal manner, there is much informal evidence that it has been and will continue to be useful in the context of the Spoken Language Systems group's GALAXY architecture. We have been using the new system for several months now and are finding it very amenable to the development of multilingual applications. Throughout this thesis, we have given several examples from our pilot PHRASEBOOK application, a speech-based, travel-domain translation system. Several developers have used PLUTO to create multilingual grammars for this application. The Spanish grammar I wrote can handle a set of 500 sentences that were solicited from colleagues, prompted for phrases a traveler might want to say in a foreign language while abroad. While this grammar does not adhere strictly to the policy we formulated in Chapter 4,[4] its

[2] We are currently in the process of moving toward a syntactic representation, so this may be a useful metric in future work.

[3] This is in part due to the fact that we are also in the process of updating our linguistic representation to be more syntactic, which impacts our grammars.

[4] In fact, it was in part the process of writing this grammar that helped us conceive of this policy: it made us realize that there were many ways of using the surface realization components, some with more promise

existence is proof that the new incarnation of the surface realization system is at least as powerful as the old one. Having written a Spanish grammar using the old system, I can also say that the new system is much easier to use. In addition to Spanish, we have also implemented grammars for both English and Chinese, having translated the same set of 500 sentences from English to Chinese and back to English.

The thrust of this thesis has been on the use of PLUTO for multilingual applications. However, we have also employed PLUTO as a preprocessor in query-response applications. For instance, in the last year, our group has developed new applications in the HOTEL and RESTAURANT domains. In these domains, PLUTO has been used to annotate meaning representations with information pertinent to summarization of a database lookup. For example, PLUTO can process a long list of restaurants to figure out where the majority of them are located. PLUTO can also decide when there are few enough entries in a list of items to be explicitly enumerated. We have developed some special commands for using PLUTO in query-response applications. These are explained further in Appendix A.

5.2 Summary

This work has presented an architecture and a grammar-writing policy for a surface realization component embedded in a spoken dialogue system. The architecture decouples two subtasks in surface realization: (1) the modification of meaning representations to perform feature selection and structural transformations, and (2) lexicalization, morphological inflection, and word ordering in the production of strings. This decoupling has many advantages. For instance, it simplifies the system's rules, making them easier to read and develop. Additionally, making two passes over the meaning representation allows the exploration and analysis of the meaning representation without actually having to generate strings. Making two passes has some potential disadvantages — forcing a developer to learn two distinct formalisms, and increasing processing time. However, we have shown that the latter problem is negligible since the system is already quite fast, and we have mitigated the former: by respecting the formalism previously developed in our realization system, we have developed a new grammar-scripting language that utilizes almost the same command names and syntax, with an associated semantics appropriate to frame modification. Finally, we have

for domain independence than others.

showcased our decoupled architecture in a hybrid approach to MT combining structural transfer and interlingua.

In spite of the fact that the decoupling imposes a logical division in the surface realization component, the system is not sufficiently constrained by itself to achieve our goal of writing robust, reusable, domain-independent grammars. To compensate for this, we have presented a policy for grammar-writing. Our policy is to encode as much domain-dependent and linguistic information as possible in the lexicon, allowing such information to spill into the grammars only when doing so greatly simplifies the writing of the grammars. We have offered several examples for generation into four languages that illustrate the partnership among the components of the surface realization system.

5.3 Future Work

There are a few changes to the current implementation of the surface realization system that will likely need to be made in the future. Most importantly, there exists functionality in PLUTO and GENESIS that should be identical but is not. This is due in part to the fact that I developed the PLUTO code largely independently of the GENESIS code.[5] While I endeavored to adhere exactly to the descriptions of the system painstakingly supplied by Lauren Baptist in her thesis [5], there were adjustments made to certain algorithms that resulted in misalignments between the two components. For example, PLUTO and GENESIS seem to have different policies for selectors set from the vocabulary. This may be confusing to a developer and should be standardized in the future. Another source of misalignment is functionality that was added to PLUTO but not to GENESIS. For example, the ability to test the equality of values in **if** commands was not available in the native implementation of GENESIS. This has been found to be very useful in PLUTO, and GENESIS needs to be brought fully up-to-date. It would, I think, be very beneficial to re-work the realization code to break down the component functions so that it is clearer what functionality PLUTO and GENESIS share and where they diverge.

There are several project currently underway that will need and use the capability for fast and easy, reusable generation grammars. For example, we are working toward

[5] I did not build a common library of functions shared by the two components because the GENESIS code does not lend itself easily to such modularization.

the induction of parsing grammars from generation grammars. This work depends on generation grammars that are simple to write, understand, and analyze. In addition, we are developing a spoken language learning system that serves as a personal tutor for a foreign language learner. The success of this application will also benefit from the ability to create multilingual grammars quickly and easily. We are certain that in the future, as world-wide communication continues to increase as ever, and as computers have a larger and larger role in facilitating that communication, that our work will have an increasingly important place.

Appendix A

PLUTO Reference Manual

PLUTO is a frame-manipulation component designed for use in a natural language generation system. A developer can use PLUTO by creating a rule-template file using the commands of the PLUTO scripting language. This reference manual provides information about these commands in four sections:

- **Frame usage and terminology**: PLUTO has been developed in the context of the Spoken Language Group's GALAXY architecture which uses a linguistic frame consisting of *keywords* and *attributes* to represent meaning. This section provides details about GALAXY frames necessary for understanding the workings of the PLUTO scripting language.

- **The PLUTO rule-template file and the lexicon**: The second section describes the format of the rule-template file and the lexicon. It also explains how PLUTO rule-templates can be grouped to form classes.

- **The PLUTO scripting language**: The third section describes the syntax and semantics of the scripting language commands.

- **Examples**: The last section provides examples of several of the commands. The primary purpose of the examples is to illustrate the syntax and semantics of the PLUTO scripting language. Although they are motivated by linguistic concerns, in general they are insufficient for actual generation purposes.

This appendix adheres to the following typeface conventions: `teletype` is used for frame

```
{c yn_question
   :aux "do"
   :topic {q pronoun
            :name "you" }
   :pred {p possess
           :topic {q food
                     :name "apple"
                     :number "pl" } } }
```

Figure A-1: Linguistic frame representing a string such as *Do you have any apples?*

constituents, **bold** for general command names, and *italics* for rule-template and (specific) command names.

A.1 GALAXY Frame Usage and Terminology

A.1.1 Introduction to Linguistic Frames

Linguistic frames are used to represent meaning in the GALAXY system. They are hierarchical, tree-like structures containing (*keyword:attribute*) pairs. Attributes can be strings, integers, floats, frames, or lists. The recursive structure of frames allows for the encoding of sentences of arbitrary length and complexity.

Figure A-1 depicts a linguistic frame. Any particular linguistic frame corresponds to a set of language strings. For instance, two English strings in the set corresponding to the frame in Figure A-1 are *Do you have apples?* and *Do you have any apples?* Similarly, there may be more than one linguistic frame that represents the same language string. For instance, a modified version of the frame in Figure A-1 that included a (quantifier:any) pair in the `food` topic frame could also represent the string *Do you have any apples?* Hence, there is a many-to-many mapping between linguistic frames and strings.

The frame from Figure A-1 is depicted in tree form in Figure A-2. In general, the top level frame and its immediate constituents correspond to the root of the tree and its children; nested frames and their immediate constituents correspond to the internal nodes of the tree and their children; and the leaves of the tree are all strings, integers, or floats.

Frames come in three types, designed to be the major elements of linguistic structures: *clause*, *topic*, and *predicate* (Figure A-3). Clause frames represent sentences and complements. Topic frames are primarily for noun phrases, while predicate frames are primarily for prepositional, adjective, and verb phrases.

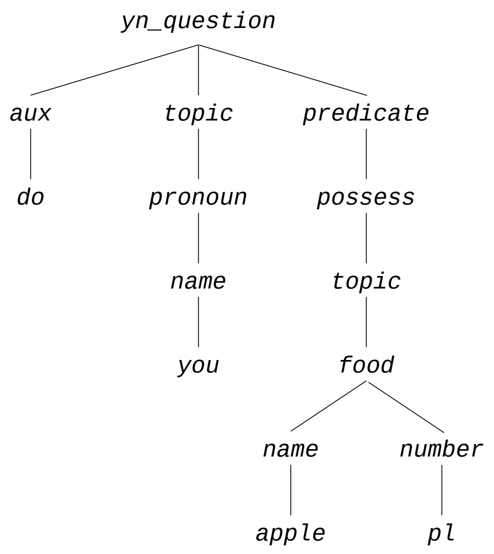

Figure A-2: Linguistic frame as a hierarchical tree.

Linguistic frame names (Figure A-4) and keywords (Figure A-5) can relay semantic or syntactic information. The hierarchical structure of a linguistic frame suggests syntactic relationships among constituents. For instance, the frame in Figure A-6, with a topic, auxiliary, and predicate at the top-level, and a topic frame nested in the top-level predicate implies a *subject-verb-object* relationship among the frame constituents.

Frames as Generic Data Structures: The Info-Frame

Frames may also be used as generic data structures. For example, PLUTO uses frames to encode information global to a linguistic frame in a structure called the *info-frame*. Because child frames do not have access to their parents in GALAXY frames (i.e., there are no back-pointers from children to parents), the info-frame also provides a mechanism for propagating information from higher level frame constituents to lower-level ones.

Info-frames in PLUTO contain both keys (called *info-keys*) and selectors. The primary difference between info-keys and selectors is that the former can take on string values and the latter are binary flags. Both info-keys and selectors are set from either the PLUTO rule-template file or the lexicon. Figure A-7 shows that there is a distinction between selectors set from the rule-template file (*rule selectors*) and the lexicon (*vocabulary selectors*). Vocabulary selectors are stronger than rule selectors because they persist in the info-frame until explicitly removed, whereas rule selectors have an inherently limited lifetime. In contrast, info-keys persist in the info-frame until explicitly deleted, whether set from the

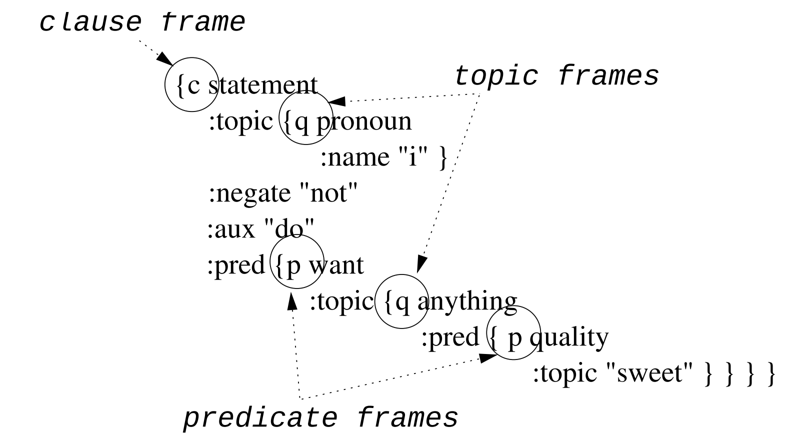

Figure A-3: Basic frame types correspond to the categories *clause* (denoted by a *c*), *topic* (denoted by a *q*), and *predicate* (denoted by a *p*).

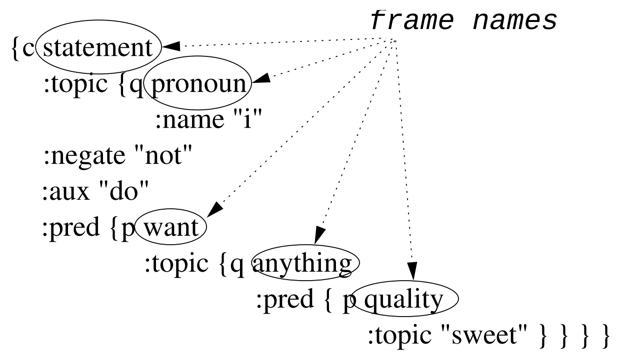

Figure A-4: Linguistic frame names relay semantic and syntactic information. The frame name `pronoun` provides syntactic information, while the other frame names provide semantic information.

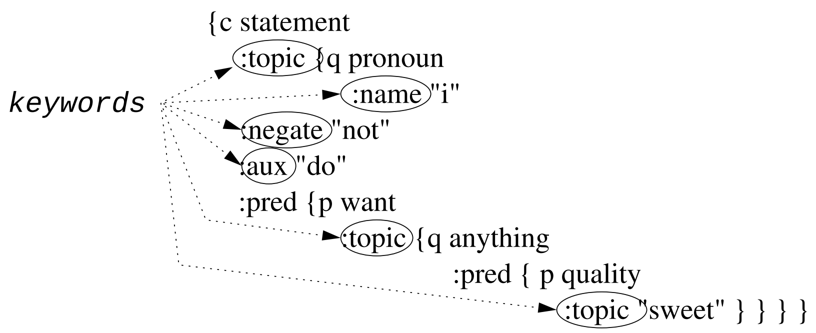

Figure A-5: Linguistic frame keys relay more semantic and syntactic information: the keys `:topic` and `:aux` provide syntactic information, while `:negate` provides semantic information.

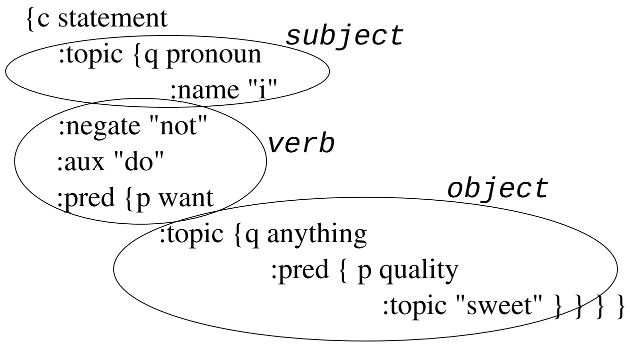

Figure A-6: Linguistic frame hierarchy suggests syntactic relations. This linguistic frame represents a string such as *I do not want any sweets.*

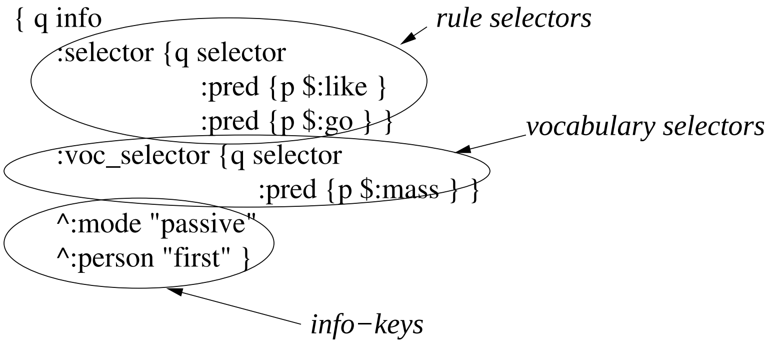

Figure A-7: An info-frame contains info-keys that can take on values, as well as selectors, set either from the vocabulary file or the rule-templates file, that act as flags.

rule-template file or the lexicon.

A.2 The PLUTO Rule-Template File and the Lexicon

The PLUTO rule-template file contains rule-template names and commands. Each rule-template is associated with a set of commands that are executed when the rule-template executes. The rule-template file has the following general format:[1]

$$\boxed{\textit{rule-template-name} \quad \textit{command}_1 \ \textit{command}_2 \ \ldots \textit{command}_n}$$

In general, every time PLUTO needs to process a frame (either the top-level frame or a nested frame), it looks for a rule-template with which to process it. PLUTO adheres to the following *rule-template-finding algorithm* to search for such a rule-template:

[1] When a rule-template is too long to fit on a single line, a backslash character ('\') may be used to indicate that the rule-template continues onto the subsequent line.

1. if there is a rule-template that matches the **frame name** of the current frame, use that rule-template;

2. if there is a **group** that contains the frame as a member (i.e., the name of the frame is on its membership list), then use the group's rule-template;

3. use a **default template** for the type of the current frame (i.e., clause, topic, or predicate).

When a match is made in one of the steps in the algorithm, the rule-template executes. An alternative way for a rule-template to execute is when control is explicitly transfered to that rule-template via a **goto** command.

Groups

Groups serve two purposes: the first is to abstract over frames that have similar processing requirements so as to avoid an explosion of rule-templates (see [5], pp. 81–87); the other is to refer to a set of semantic concepts as a class and avoid having to name each member separately.

Grouping rule-templates have the following format:

*type*_`groups`	$\textit{group-name}_1$ $\textit{group-name}_2$ ... $\textit{group-name}_n$
$\textit{group-name}_1$	*group-member-list*
$\textit{group-name}_1$_`template`	*commands*
	...
$\textit{group-name}_n$	*group-member-list*
$\textit{group-name}_n$_`template`	*commands*

All of the members of a particular group are constrained to have the same frame type. A special rule-template called *type*_groups lists the groups for each type. There is another special rule-template for each *group-name* in this list. The body of each such rule-template consists of a list of the members of the group. Members are identified by frame name. Finally, there is a rule-template for each group, called *group-name*_template.

Once grouped, group names may be used as primitives in other commands. For instance, an **if** command can test on the existence of a semantic class by referring to the group name.

Default Rule-Templates

To ensure that there is a rule-template that will handle every frame, it is possible to provide a default rule-template for each frame type. If there is no rule-template matching the specific frame name, and if the frame is not a member of any group, then the default rule-template will be used to process the frame. Default rule-templates have the following format:

*frame-type*_template   *command*$_1$ *command*$_2$ … *command*$_n$

The Lexicon

The lexicon, shared by PLUTO and GENESIS, has the following general format:

entry-name   *part-of-speech*   *"default"* *other-lexical-information*

The *part-of-speech* can be anything but is typically one of N (noun), V (verb), A (adjective), or O (other). The first quoted string following the part of speech is the default, although there may be other possible realizations for the entry in *other-lexical-information*. *other-lexical-information* may also include information such as number and gender, which PLUTO can then make explicit in the frame when it visits the vocabulary file.

A.3 The PLUTO Scripting Language

Table A.1 gives a brief description of each command in the PLUTO scripting language. The remainder of this section describes the syntax and semantics of each command in more detail.

A.3.1 Selector

Lexical and Global Selectors

$: *selector-name*

Local Selectors

Command Name	**Purpose**
selector	Set a flag in the info-frame.
keyword	Process the value of a key in the current frame.
info-key	Process the value of an info-key in the current info-frame.
single predicate	Process the value of a predicate in the current frame.
:pred	Process the values of all predicates in the current frame.
lexical lookup	Insert information from a particular lexical entry into the frame.
core	Insert information from the lexical entry associated with the frame name into the frame.
child	Process a key, a predicate, or all predicates in one or more child frames.
tug	Process a key, a predicate, or all predicates in all child frames.
yank	Process a key, a predicate, or all predicates in all descendant frames.
yanked child	Process a key, a predicate, or all predicates in the descendants of one of more child frames.
goto	Transfer control from the current rule-template to a new rule-template.
special form	Perform some specific computation.
or	Execute a list of primitive commands from left to right and stop upon the first success.
set	Set a (*keyword:attribute*) pair in the linguistic frame or a non-empty set of its children, or set an (*info-key:attribute*) pair in the info-frame, overwriting the current value of the *keyword* or *info-key*, if it exists.
softset	Set a (*keyword:attribute*) pair in the linguistic frame or a non-empty set of its children, or set an (*info-key:attribute*) pair in the info-frame, if there is no current value of the *keyword* or *info-key*.
clone, softclone	Syntactic sugar for a particular type of **set** and **softset** command, respectively.
del	Delete a (*keyword:attribute*) pair from the linguistic frame or a non-empty set of its children, or an *info-key:attribute* pair or selector from the info-frame.
if, if/else_if	Execute a command based on the result of a test.
suspend	Conceal a set of info-keys for the duration of the current rule-template.

Table A.1: The PLUTO scripting language commands in a nutshell.

($:*selector-name* `target`) *where* `target` *is a* **keyword** *or* **predicate** *command.*

Selector commands are used by PLUTO to set a binary flag in the info-frame. They may be issued from either the PLUTO rule-template file (*rule selector*) or the lexicon (*lexical selector*). There are two types of rule selectors: *local selectors* and *global selectors*.

The lifetime of a selector depends on its type:

- a **lexical selector** is set in the info-frame and remains there until explicitly deleted;
- a **local selector** appears with an associated target and remains in the info-frame only as long as the target is being processed;
- a **global selector** is set in the info-frame while PLUTO is processing some frame and remains in the info-frame as long as that frame is being processed.

A.3.2 Keyword

:*keyword-name*

Keyword commands are used to reference the value of a key in the current frame. The value of a key is processed according to its type, assumed to be one of *string*, *integer*, *frame*, or *list*:

- **strings and integers**: PLUTO searches the lexicon for an entry matching the string or integer. If such an entry is found, PLUTO adds all keywords in the entry to the frame and all info-keys and vocabulary selectors to the info-frame.
- **frames**: PLUTO searches for a rule-template with which to preprocess the frame using the rule-template-finding algorithm of Section A.2.
- **lists**: each list element is processed according to its value (string, integer, frame, or list).

A.3.3 Info-Key

^:*info-key–name*

Info-key commands are used to reference the value of an info-key in the current info-frame. The value of an info-key is assumed to be a string.

A.3.4 Single Predicate

predicate-name

Single predicate commands are used to reference the value of a predicate in the current frame. The value of a predicate is always a frame, and it is processed in the same way as in a **keyword** command.

A.3.5 Any Predicate

`:pred`

A **:pred** command is a wildcard used to reference the values of all of the predicates in the current frame (in an arbitrary order).

A.3.6 Lexical Lookup

! *lexical-item*

Lexical lookup commands are used to insert information from a particular lexical entry into the current frame. PLUTO looks for a lexical entry that matches the *lexical item* and adds all keywords in the entry to the frame and all info-keys and vocabulary selectors to the info-frame.

A.3.7 Core

`$core`

Core commands are used to insert information from the lexical entry associated with the frame name into the frame (keywords) or info frame (info-keys and vocabulary selectors).

A.3.8 Child

outer-expr [$inner\text{-}expr_1$ $inner\text{-}expr_2$ ... $inner\text{-}expr_n$]

Child commands are used to reference a key, a predicate, or all predicates in one or more child frames. Each *inner-expression* is a keyword name, a predicate name, or a **:pred** command. Each *outer expression* is a **keyword** command, a **single predicate** command, or a **:pred** command. PLUTO evaluates each *inner-expression* and searches for matching subframes of the current frame. Then, it executes the *outer-expression* in the context of each of these frames.

A.3.9 Tug

`<--`*expr*

Tug commands are used to process a key, a predicate, or all predicates in all child frames. In other words, the expression can be a **keyword** command, a **single predicate** command, or a **:pred** command, which is executed in the context of all child frames.

A.3.10 Yank

`<==`*expr*

Yank commands are used to process a key, a predicate, or all predicates in all descendant frames. Again, the expression can be a **keyword** command, a **single predicate** command, or a **:pred** command. This command is executed in the context of any descendant of the current frame with a frame value.

A.3.11 Yanked Child

`<==`*outer-expr* [*inner-expr*$_1$ *inner-expr*$_2$ ... *inner-expr*$_n$]

Yanked child commands are used to process a key, a predicate, or all predicates in the descendants of one of more child frames. PLUTO first finds the *inner expressions* with a frame value in the current frame. Then, the *outer expression* is executed in the context of all frame descendants of these child frames.

A.3.12 Goto

`>`*expression*

Goto commands are used to transfer control from the current rule-template to a new rule-template. When control is transferred, PLUTO continues working on whatever frame it was working on before the transfer occurred. The *expression* can be a rule-template name or a group name. PLUTO will search for a rule-template whose name matches the expression. If there is one, PLUTO checks to see if it is a group list. If it is, PLUTO searches for frame children that are members of the group and processes each one. If the rule-template is not a group name, PLUTO simply executes its commands.

A.3.13 Special Forms

Special forms are used to perform some specific computation or inference related to the current frame. For instance, there is a special form for determining the most commonly occurring elements in a list of values (*majority*). It is appropriate for PLUTO to perform this kind of computation because the resulting information can be inserted into the frame, where it is available for components that receive the frame (such as GENESIS) to make use of it.

Majority Command

```
$majority
```

A **majority** command causes PLUTO to create and insert into the frame a list of values that constitute the majority of a list in the frame. It makes use of two arguments that can be set in the rule-template or supplied by the dialogue manager: *:maxn* and *percent*. It will only create a *:majority* list if fewer than *:maxn* elements of the list consume more than *:percent* percent of the total number.

List Items Command

```
$list_items
```

A **list_items** command causes PLUTO to add a list to the frame only if that list does not exceed some predetermined size, which can be specified directly in the rule-template.

Singleton Command

```
$singleton
```

Given a list of values, the **singleton** command adds information to the frame to note whether there is only a single value present (*singleton*) or whether there is only a single value absent (*singleton-out*) in a list.

Time Command

```
$time
```

A **time** command may be used to convert military time to a standard format. This is used primarily in conjunction with a database that stores time in military format.

Filter Command

`$filter`

A **filter** command may be used to silence elements in a list not containing a designated pattern.

A.3.14 Or

($expr_1$ $expr_2$ $expr_3$ ... $expr_n$)

Or commands are used to execute a list of primitive commands from left to right and stop upon the first success, where success is considered a change to the linguistic or info-frame.

A.3.15 Set and Softset

(`$set` *target source*)

(`$softset` *target source*)

A **set** command is used to

- set a (*keyword:attribute*) pair in the linguistic frame or one or more of its children, or
- set an (*info-key:attribute*) pair in the info-frame,

overwriting the current value of the *keyword* or *info-key*, if it exists.

A **softset** command performs the same function unless the *keyword* or *info-key* already exists.

The *target* of a **set** or **softset** command may take the form of a keyword, an info-key, or a **child** command:

- if the target is a keyword, then a **set** inserts the keyword whose value is the value of the source into the frame, and a **softset** makes the insertion only if the keyword does not already exist;
- if the target is an info-key, then the results of **set** and **softset** are the same as for a keyword, but instead of setting keys in the linguistic frame, PLUTO sets info-keys in the info-frame;

- if the target has the form of a **child** command, then its *inner-strings-list* is constrained to have a single value (which may be a key, predicate, or **:pred** command) and the *outer-string* must name a key and may not be yanked or tugged. Thus, the target will either be a single keyword in (1) a particular child frame (keyword or predicate) or (2) every predicate child.

The *source* of a **set** or **softset** command may take the form of a string, keyword, info-key, GENESIS **lexical lookup** command, a **core** command, or a **child** command:

- If the source of the **set** command is a string, PLUTO sets the *target* to that string.
- If the source of the **set** command is a keyword that exists in the linguistic frame, PLUTO sets the target to the associated value (a string, integer, frame, or list).
- If the source of the **set** command is an info-key that exists in the info-frame, PLUTO sets the target to the associated value (a string).
- If the source of the **set** command is a GENESIS **lexical lookup** command, PLUTO asks GENESIS to find a string for the source (as described in [5], pp. 63–64) and sets the target to the result.
- If the source of the **set** command is a **core** command, PLUTO sets the target to the name of the current frame.
- When the source of the **set** command is a **child** command, PLUTO sets the target to the value of the keyword or predicate of the child frame.

A.3.16 Clone and Softclone

`($clone` *child-statement*`)`

`($softclone` *child-statement*`)`

The **clone** and **softclone** commands are syntactic sugar for special cases of the **set** and **softset** commands, respectively: when the source is a **child** command and the target matches its *outer-string*, then the target may be dropped and **clone** or **softclone** used instead. Hence, the following two commands are semantically equivalent:

`($set :name :name[:topic])` ≡ `($clone :name[:topic])`

A.3.17 Del

`($del` *target*`)`

Del commands are used to delete a (*keyword:attribute*) pair from the linguistic frame or a non-empty set of its children, or an *info-key:attribute* pair or selector from the info-frame.

The *target* may take the form of a keyword, predicate, info-key, selector, or **child** command. In all cases, PLUTO searches for the target in the linguistic frame (keywords, predicates, and **child** commands) or the info-frame (info-keys and selectors), and, if it is found, deletes the target and its value from the frame.

A.3.18 If

`($if` *test then* `[`*else*`])`

If commands execute the *then* consequent if the *test* evaluates to *true*; otherwise, if there is an *else* consequent, it is executed. The *then* consequent and *else* consequents may be any command that contains no whitespace.

The **if** *test* may take any of the following forms:

- a keyword, predicate, info-key, or selector
- a **:pred** command,
- a group name,
- a **child** command,
- a test of equality or inequality,
- or a combination of any of the above tests, connected by a logical *and* or a logical *or*.

The first items in the above list evaluate to *true* if they exist in the current frame (keywords and predicates) or info-frame (info-keys and selectors). A test in the form of a **:pred** command evaluates to true if there exists at least one predicate in the current frame. A group name test evaluates to *true* if one of the group's members exists in the current frame. A test in the form of a **child** command evaluates to *true* if the *outer-expression* is found in at least one member of the *inner-expression-list*. A test of equality or inequality compares the value of two string-valued operands.[2] A test of equality evaluates to *true* if both operands are the same; a test of inequality evaluates to *true* if both operands are different.

Any number of the aforementioned tests may be connected by logical *and* (using the symbols "&&") or logical *or* (using the symbols "||"). Any such agglomerated test may only use one logical connective or the other. An *and*-connected test is *true* iff each constituent test is *true*, while an *or*-connected test is*true* iff at least one constituent test is *true*.

Tests of Equality and Inequality

Tests of equality or inequality have the following form:

- **Equality:** *operand1* `==` *operand2*
- **Inequality:** *operand1* `!=` *operand2*

Operand1 may take the form of a keyword, an info-key, a predicate, a **tug**, a **yank**, a **yanked child**, or a **child** command, while *operand2* may be any of these or a string. Both operands are reduced to strings or lists of strings so as to compare their values:

- A keyword is converted to a string according to the value of the keyword in the current frame. If the keyword does not exist in the current frame, then the operand value is *null*. Otherwise, if the value of the keyword is a string, then the operand value is that string. If the value is an integer, then the operand value is that integer, converted to a decimal string. If the value is a frame, then the operand value is the frame name.
- The string value of an info-key is its value in the info-frame, if such an info-key exists. If the info-key does not exist, the operand value is *null*.
- The string value of a predicate is the frame name. If the predicate does not exist in the frame, the operand value is *null*.
- An operand in the form of a **tug** (or **yank**) command is converted to a string by searching for the target among child (or descendant) frames of the current frame. The algorithm for converting keywords and predicates to strings is used to convert

[2] All string comparisons are case sensitive.

the target to a string. If more than one instance of the target is found, then the operand reduces to a list of strings.

- An operand in the form of a **yanked child** command is converted to a string by searching for the target among descendants of the frames in the *inner-expressions-list*. The algorithm for converting keywords and predicates to string is used to convert the target to a string. If more than one instance of the target is found, then the operand reduces to a list of strings.

- An operand in the form of a **child** command is converted to a string by searching the *inner-expressions-list* for frames, and searching those frames for the *outer-expression*. The *outer-expression*, be it a keyword or predicate, is then converted to a string using the algorithm for converting keywords and predicates.

Two single strings are compared to one another directly. A single string p is compared to each member of a list of strings ℓ. An equality test evaluates to *true* iff $\exists s \in \ell$ s.t. $p = s$. An inequality test evaluates to *true* iff $\forall s \in \ell$ s.t. $p \neq s$. Two lists of strings ℓ and ℓ' are compared to one another by comparing each member of one list to all of the members of the other list. An equality test evaluates to *true* iff $\exists s \in \ell, s' \in \ell'$ s.t. $s = s'$. An inequality test evaluates to *true* iff $\forall s \in \ell, s' \in \ell'$ s.t. $s \neq s'$.

A.3.19 If/Else_if

($if *test*$_1$ *then*$_1$ $else_if *test*$_2$ *then*$_2$ ... $else_if *test*$_n$ *then*$_n$ [*else*])

An **if/else_if** command allows chaining of **if** commands. A series of *test*s and *then* consequents describe different conditions and associated *then* consequents to execute. Execution of the **if/else_if** command stops when the first test evaluates to *true*. If all tests fail, a final *else* consequent is executed, if it exists.

A.3.20 Suspend

$suspend

($suspend *info-key–name*)

A **suspend** command is used to conceal a particular info-key (when *info-key–name* is provided) or all info-keys for the duration of a command.

```
{c statement
   :topic {q pronoun
            :name "i" }
   :pred {p understand } }
```

Figure A-8: Frame before **keyword** and **predicate** commands have been executed. An English string realization of this frame might be *I understand.*

`clause_template`	`:topic :pred`
`pronoun`	`:name`
`predicate_template`	`$core`

Figure A-9: **Keyword** and **predicate** commands.

A.4 Examples

In the following examples of the PLUTO scripting language commands, frame elements that have been inserted or altered during processing are prepended with an asterisk in the output to highlight any changes due to processing. All examples use a Spanish lexicon but should be understandable to any English speaker.

A.4.1 Example Keyword, :pred, and Core Commands

Given the frame in Figure A-8, and using the rule-templates in Figure A-9, PLUTO selects the *clause_template* (because it matches the type of the input frame) and begins executing its commands. The first is the **keyword** command *:topic*. The value of the topic is a frame, and so PLUTO searches for a rule-template with which to process it. The `pronoun` rule-template is used because it matches the frame name.

The *pronoun* template has one **keyword** command, :*name*. The value of this keyword is the string "i." PLUTO next searches the lexicon for a matching entry (Figure A-10). The entry found contains number information, expressed in the form of a (*key:value*) pair. PLUTO sets this number information as a (*key:value*) pair in the frame, as show in Figure A-11.

Since the *pronoun* command has now terminated, control is transferred back to the *clause_template*, where the **all predicates** command is processed. There is only one child predicate, `understand`. The *predicate_template* is chosen to process the frame since the frame type is predicate. The **core** command causes PLUTO to search the lexicon for an entry matching the frame name, "understand." The entry found in Figure A-10 contains a **lexical selector** command, which causes PLUTO to insert the selector `$stem_changing`

```
i           N   "" :number "first"
understand  V   "entender" ; $stem_changing
```

Figure A-10: Lexicon entries matching "i" and "understand".

```
{c statement
   :topic {q pronoun
             :name "i"
            *:number "first" }
   :pred {p understand } }
```

Figure A-11: Frame after **keyword** and **predicate** commands have been executed. Number information has been added to the `pronoun` frame. In Spanish, this frame might be realized as *Entiendo.*

into the info frame.[3] Processing ends with the frame in Figure A-11.

A.4.2 Example Keyword and Child Commands

Given the frame in Figure A-12 and the rule-templates in Figure A-13, PLUTO selects the *clause_template* and begins executing its **child** command.

It first looks one-by-one through the inner expressions in the list, which in this case are all predicate names. It has no success with the first predicate name and moves on to the second. This predicate does exist in the frame, and so PLUTO searches the `like` frame for the outer expression, in this case the keyword *:topic.* Since there is a `:topic` keyword in the `like` frame, and since its value is a frame, PLUTO finds a rule-template (*topic_template*) with which to process it. This rule-template contains a single **keyword** command, *:name.* At this point, PLUTO visits the lexicon and finds an entry for "fish" (Figure A-14), producing the frame in Figure A-15.

A.4.3 Example Tug, If, and Yank Commands

Figure A-16 shows a frame before the rule-templates in Figure A-17 are applied.

The first command of the *clause_template* is a **tug** that tells PLUTO to process all predicates found in the child frames of the current frame. There are two child frames: the value of the `:topic` keyword and the `awaken` predicate frame. PLUTO then searches these frames

[3]Although this selector is not used by PLUTO in this example, it would be used by GENESIS to choose an appropriate conjugation of the verb. This example thus illustrates the way in which the two surface realization components share the lexicon.

```
{c statement
   :topic {q pronoun
            :name "i" }
   :pred {p like
           :topic {q food
                    :name "fish" } } }
```

Figure A-12: Frame before **child** command has been executed. A possible string realization of this frame in English is *I like fish.*

`clause_template`	`:topic[love like interest]`
`topic_template`	`:name`

Figure A-13: **Child** command.

`fish`	`N`	`"pescado"`	`:number "third" :gender "m"`

Figure A-14: Lexicon entry matching "fish".

```
{c statement
   :topic {q pronoun
            :name "i" }
   :pred {p like
           :topic {q food
                    :name "fish"
                   *:gender "m"
                   *:number "third" } } }
```

Figure A-15: Frame after **child** command has been executed. Number and gender information has been added to the `food` frame. A Spanish paraphrase of this frame is *Me gusta el pescado.*

```
{c yn_question
   :aux "can"
   :topic {q pronoun
            :name "you" }
   :please "please"
   :pred {p awaken
           :topic {q pronoun
                    :name "me" }
           :pred {p at_time
                   :pred {p clock_hour
                           :topic 6 }
                   :pred {p minutes
                           :topic 30 } } } }
```

Figure A-16: Frame before **yank** and **tug** commands have been executed. In English, this frame might represent the string *Can you please wake me up at six thirty?*

```
clause_template      <--:pred ($if $:reflexive <==:topic)
predicate_template   $core
topic_template       :name
```

Figure A-17: **Yank** and **tug** commands.

```
6            O   "seis" :number "pl"
30           O   "treinta" :number "pl"
at_time      O   "a" :quantifier "def"
awaken       V   "despertar" ; $:reflexive
clock_hour   N   "hora"
minutes      N   "minuto" :conj "and"
```

Figure A-18: Lexicon entries matching topics and predicates in Figure A-16.

for all predicates they contain. The `pronoun` frame contains no predicates; however, the `awaken` frame contains one predicate, `at_time`.

This frame is processed using the *predicate_template* rule-template. Its single **core** command tells PLUTO to search for a lexical entry matching the frame name. The entry in Figure A-18 causes PLUTO to insert the vocabulary selector `$:reflexive` into the info-frame.

The **tug** command is now complete, and PLUTO continues with the **if** command. Since the `$reflexive` selector was just set in the info-frame, the **if** test evaluates to *true*, and the consequent is evaluated.

In order to process the **yank** command, PLUTO first finds all children whose value is a frame, following the same protocol as with the **tug** command. Again, the only such children are the top-level `pronoun` topic and the `awaken` predicate. However, this time PLUTO looks for *all* :*topic* descendants. During the search, PLUTO finds three such descendants: the `pronoun` topic inside the `awaken` predicate, and the `6` and `30` topics inside the `at_time` predicate.

For the `pronoun` topic, PLUTO uses the *topic_template* to process the frame and looks up the value of the keyword :*name* in the lexicon. For the `6` and `30` topics, since their values are integers, PLUTO simply searches the lexicon for an entry that matches these values as strings (Figure A-18). In each case, PLUTO inserts the appropriate information, producing the frame in Figure A-19.

```
{c yn_question
   :aux "can"
   :topic {q pronoun
             :name "you" }
   :please "please"
   :pred {p awaken
            :topic {q pronoun
                       :name "me" }
            :mode "root"
            :pred {p at_time
                    *:quantifier "def"
                     :pred {p clock_hour
                              :topic 6
                             *:number "pl" }
                     :pred {p minutes
                              :topic 30
                             *:number "pl" } } } }
```

Figure A-19: Frame after **yank** and **tug** commands have been executed. Quantifier information has been added to the `at_time` frame, and number information to the `clock_hour` and `minutes` frames. In Spanish, this frame could be realized as the string *¿Puede Ud. despertarme a las seis y media por favor?*

A.4.4 Example Keyword, Goto, and Core Commands

PLUTO begins processing the frame in Figure A-20 with the *clause_template* (Figure A-21). The first command is the :*topic* **keyword** command. Using the *topic_template* and the lexicon in Figure A-22, PLUTO adds the number information for the "you" entry to the `pronoun` frame[4] (as shown in Figure A-23).

The next command in the *clause_template* is a **goto** command. PLUTO finds a rule-template that matches the target of the **goto**, "verb_preds"; it then checks the *predicate_groups* list and discovers that "verb_preds" is the name of a group. It searches the list contained in the *verb_preds* special membership list rule-template to see if there are any child predicates on the list. Since the predicate `know` is on the list, PLUTO processes this child frame, selecting the *verb_preds_template* to do so.

The first command of the *verb_preds_template* is a **keyword** command. PLUTO finds a :`rel_clause` key in the `know` frame and processes its frame value (the `exist` frame) with the *clause_template*. PLUTO processes the topic of the `exist` frame with the *topic_template* (which, together with the lexicon, adds number and gender information to the `bank_machine` frame). The **goto** command of the *clause_template* in this case fails because there are no

[4]Note that in Spanish, the formal *you* (*usted*) is conjugated in the third person, singular form.

```
{c yn_question
   :aux "do"
   :topic {q pronoun
            :name "you" }
   :pred {p know
           :rel_clause {c exist
                         :rhet "there"
                         :aux "link"
                         :trace "where"
                         :topic {q bank_machine
                                  :quantifier "indef" } } } }
```

Figure A-20: Frame before **goto** commands have been executed. In English, this frame could represent the string *Do you know where there is a bank machine?*

clause_template	:topic >verb_preds
topic_template	:name $core
predicate_groups	verb_preds
verb_preds	go listen know see understand
verb_preds_template	:rel_clause

Figure A-21: **Goto** commands.

children in the `exist` frame that are members of the *verb_preds* group.

Having completed the :*rel_clause* rule-template, outputs the frame in Figure A-23.

A.4.5 Example Goto, Or, Clone, and Softset Commands

The frame in Figure A-24 is processed using the rule-templates in Figure A-25.

Processing begins using the *clause_template* rule-template and its first command, the **keyword** command *:topic*. The :*name* command embedded in the first command – an **or** command – of the *topic_template* executes successfully, setting the number information in the lexical entry for "we" in the frame (see Figure A-27).

The next command in the *topic_template* is a **goto** command. The target is a rule-template that is not a group name, so control is transferred to it. This rule-template, *set_default_gender*, then causes default gender information (in this case the default is male) to be set in the `pronoun` frame, due to its **softset** command. In contrast, the second **goto** command of the *topic_template* transfers control to a command that has no effect since a key called *number* information already exists in the frame.

Now finished with the :*topic* command of the *clause_template*, PLUTO proceeds to the second command, another **goto** command. This command causes PLUTO to transfer control

bank_machine	N	"ATM" :number "third" :gender "m"
where	0	"dónde"
you	0	"" :number "third"

Figure A-22: Lexicon entries matching topics and predicates in Figure A-20.

```
{c yn_question
   :topic {q pronoun
            :name "you"
           *:number "third"}
   :pred {p know
           :rel_clause {c exist
                        *:aux "there_is"
                         :trace "where"
                         :topic {q bank_machine
                                  :quantifier "indef"
                                 *:number "third"
                                 *:gender "m" } } }
```

Figure A-23: Frame after **goto** commands have been executed. Number information has been added to the pronoun frame, while :rhet key has been deleted from the exist frame. The :aux key in the same frame has been altered to reflect the fact that the auxiliary *hay* in Spanish means *there is* in English. Finally, number and gender information have been added to the bank_machine frame. In Spanish, this frame might be realized as *¿Sabe Ud. dónde hay un cajero automático?*

```
{c statement
   :topic {q pronoun
             :name "we" }
   :pred {p speak
            :topic {q language
                      :name "chinese" } } }
```

Figure A-24: Frame before **or**, **clone**, and **softset** commands have been executed. This frame might represent the English string *We speak Chinese.*

clause_template	:topic >propagate_number >propagate_gender :pred
topic_template	(:name $core) >set_default_gender >set_default_number
propagate_number	($clone :number[:topic]) \
	($softset :number[:pred] :number)
propagate_gender	($clone :gender[:topic]) \
	($softset :gender[:pred] :gender)
set_default_number	($softset :number "third")
set_default_gender	($softset :gender "m")
predicate_template	:topic

Figure A-25: **Or**, **set**, and **softset** commands.

```
chinese  N   "chino" :gender "m"
we       N   "" :number "pl_third"
```

Figure A-26: Lexicon entries matching topics in Figure A-24.

to the *propagate_number* rule-template. This rule-template consists of a series of commands from the set family that cause number information to be propagated to the top-level frame and its children. The first of these commands, a **clone** command, causes the target keyword (*:number*) to be set in the frame with the value taken from the number of the topic, recently set in the `pronoun` frame. The second command propagates the number information in the top-level frame to all predicates.

The next **goto** command in the *clause_template* causes PLUTO to transfer control to the *propagate_gender* rule-template. Its commands behave similarly to those of the *propagate_number* rule-template.

The last command in the *clause_template* causes PLUTO to process the predicate `speak` with the *predicate_template*. The only command in this rule-template is the **keyword** command *:topic*; PLUTO can once again use the *topic_template* to process the topic frame. The first command of the **or** command is successful because PLUTO modifies the frame with the gender information in the lexical entry for `chinese`. Finally, default number information (in this case third person singular[5]) is set in the `language` frame. The *set_default_gender* rule-template has no effect in this case.

A.4.6 Example Del and If Commands

PLUTO takes as input the frame in Figure A-28 and outputs the frame in Figure A-30 after processing with the rule-templates in Figure A-29.

The first rule-template is the *clause_template*, and its first command says to go to a rule-template called *decide_del_topic* containing a single **if** command. Its test has two subparts connected with a logical *and*. The first part of the test says that the topic must evaluate to the string "thing." Since the name of the `:topic` frame is "thing," this test of equality evaluates to *true*. The second part of the test says that the quantifier of the topic must evaluate to the string "this." Since it does, this equality test is also *true*, and the *then*

[5]By convention, if number is not explicitly plural (e.g., "pl_third"), it is considered singular.

```
{c statement
   :topic {q pronoun
             :name "we"
            *:number "pl_first" }
  *:number "pl_first"
  *:gender "m"
   :pred {p speak
            :topic {q language
                      :name "chinese"
                     *:gender "m"
                     *:number "third" }
           *:number "pl_first"
           *:gender "" } }
```

Figure A-27: Frame after **or**, **clone**, and **softset** commands have been executed. Number and gender information has been acquired for the `pronoun` frame and then propagated from there to the top-level frame and its predicate. Number and gender information has also been added to the `language` frame. In Spanish, this frame might represent *Hablamos chino.*

```
{c statement
   :topic {q thing
             :quantifier "this" }
   :number "third"
   :gender "m"
   :aux "link"
   :pred {p quality
            :topic "too salty"
            :number "third"
            :gender "m" } }
```

Figure A-28: Frame before **del** and **if** commands have been executed. In English this frame might represent the string *This is too salty.*

`clause_template`	`>decide_del_topic >set_aux`
`decide_del_topic`	`$if :topic == "thing" && :quantifier[:topic] == "this" \`
	`>del_topic`
`del_topic`	`($del :topic)`
`set_aux`	`($if :aux == "would" >set_conditional \`
	`$else_if :aux == "link" || :aux == "will_be" >set_link \`
`set_link`	`($if at_age >set_tener \`
	`$else_if :trace == "where" || <==:trace == "where" \`
	`>set_estar \`
	`$else_if estar_preds >set_estar >set_ser)`
`predicate_groups`	`estar_preds`
`estar_preds`	`on state_of_being available lost ready quality over_there`
`set_estar`	`($set :aux "link_estar")`

Figure A-29: **Del** and **if** commands.

```
{c statement
   :number "third"
   :gender "m"
  *:aux "link_estar"
   :pred {p quality
            :topic "too salty"
            :number "third"
            :gender "m" } }
```

Figure A-30: Frame after **del** and **if** commands have been executed. The topic frame `thing` has been deleted, and the `:aux` key has been refined. In Spanish, there are several ways to express the English verb *to be.* This frame might be realized in Spanish with the string *Está demasiado salado.*

consequent is executed. The *then* consequent is a **goto** command. Control is transferred to the *del_topic* rule-template, which tells PLUTO to delete the topic (see Figure A-30).

The second command of the *clause_template* transfers control to a rule-template for setting the auxiliary verb in the frame. The *set_aux* rule-template consists of a single **if/else_if** command with two clauses. The test of the first clause fails because the value of `:aux` in the frame is not equal to "would." The test of the second clause succeeds because one of the subtests (the first one) evaluates to *true*: `:aux` has the value "link." The *then* consequent of this clause transfers control to a different rule-template, *set_link.*

This rule-template also consists of a single **if/else_if** command, this time with three clauses. The test of the first clause fails because the predicate *at_age* does not exist in the frame. The test of the second clause asks two questions: first, if there is a key called *:trace* in the current frame (i.e., the top level), and if so, if its value is the string "where;" and second, if there is a key called *:trace* in any of the descendants of the current frame, and if so, if one of these descendent's value is the string "where." Since the answer to both of these questions is "no," the second test fails. Finally, the test of the third clause asks whether there are any members of the group *estar_preds* in the current frame. The members of the group are listed in the rule-template file, and indeed there is such a member in the frame: the predicate `quality`. Hence, this test succeeds, and its *then* consequent is executed. The consequent is a **goto** rule-template that transfers control to a rule-template called *set_link_estar* that causes the key :*aux* to be set in the frame with the value "link_estar."

Bibliography

[1] Scott Axelrod. Natural language generation in the IBM flight information system. In *Proceedings of First Language Technology Joint Conference of Applied Natural Language Processing and the North American Chapter of the Association for Computational Linguistics*, pages 21–26, Seattle, WA, May 2000.

[2] Srinivas Bangalore and Owen Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics*, Saarbrucken, Germany, 2000.

[3] Srinivas Bangalore, Owen Rambow, and Steve Whittaker. Evaluation metrics for generation. In *Proceedings of the First International Natural Language Generation Conference*, Mitzpe Ramon, Israel, June 2000.

[4] Lauren Baptist and Stephanie Seneff. GENESIS-II: A versatile system for language generation in conversational system applications. In *Proceedings of the International Conference on Spoken Language Processing*, pages 271–274, Beijing, China, October 2000.

[5] Lauren M. Baptist. GENESIS-II: A language generation module for conversational systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, September 2000.

[6] John Bateman. KPML development environment – multilingual linguistic resource development and sentence generation. Technical report, German Centre for Information Technology, Institute for Integrated Information and Publication Systems, Darmstadt, Germany, 1996.

[7] Issam Bazzi. *Modelling Out-of-Vocabulary Words for Robust Speech Recognition.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2002.

[8] Jaime G. Carbonell, Teruko Mitamura, and Eric H. Nyberg. The KANT perspective: A critique of pure transfer (and pure interlingua, pure statistics, ...). In *Proceedings of the Fourth International Conference on Theoretical and Methodological Issues in Machine Translation*, Montreal, Canada, June 1992.

[9] John Chen, Srinivas Bangalore, Owen Rambow, and Marilyn A. Walker. Towards automatic generation of natural language generation systems. In *International Conference on Computational Linguistics*, Tapei, Taiwan, 2002.

[10] Chian Chuu. LIESHOU: A Mandarin conversational task agent for the GALAXY-II architecture. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[11] Robert Dale and Chris Mellish. Towards the evaluation of natural language generation. In *Proceedings of the First International Conference on Language Resources and Evaluation*, pages 555–562, Granada, Spain, May 1998.

[12] Bonnie Jean Dorr. *Machine Translation: A View from the Lexicon.* The MIT Press, Cambridge, Massachusetts, 1993.

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

[14] J. Glass, G. Flammia, D. Goodine, M. Phillips, J. Polifroni, S. Sakai, S. Seneff, and V. Zue. Multilingual spoken-language understanding in the mit voyager system. *Speech Communication*, 17:1–18, 1995.

[15] James Glass, Joseph Polifroni, and Stephanie Seneff. Multilingual language generation across multiple domains. In *Proceedings of the International Conference on Spoken Language Processing*, pages 983–986, Yokohama, Japan, September 1994.

[16] E. Goldberg and N. Driedger. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53, 1994.

[17] Penman Natural Language Group. The PENMAN user guide. Technical report, Information Sciences Institute, Marina Del Rey, CA, 1989.

[18] Michael Halliday. *System and Function in Language.* Oxford University Press, 1976.

[19] W. John Hutchins and Harold L. Somers. *An Introduction to Machine Translation.* Academic Press, 1992.

[20] Kevin Knight and Vasileios Hatzivassiloglou. Two-level, many-paths generation. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 252–260, 1995.

[21] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the ACL/COLING-98*, pages 704–710, Montréal, Québec, Canada, 1998.

[22] Tien-Lok Jonathan Lau. SLLS: An online conversational spoken language learning system. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[23] B. Lavoie and O. Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 265–268, 1997.

[24] William C. Mann. An overview of the Nigel text generation grammar. Technical Report RR-83-113, USC Information Sciences Institute, Marina del Rey, CA, 1983.

[25] Igor A. Mel'čuk. *Dependency Syntax: Theory and Practice.* State University of New York Press, Albany, NY, 1988.

[26] Alice H. Oh and Alexander I. Rudnicky. Stochastic natural language generation for spoken dialog systems. *Computer Speech and Language*, 16(3–4):387–407, July–October 2002.

[27] A. Ratnaparkhi. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech and Language*, 16(3–4):435–455, Jul–Oct 2002.

[28] Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Kennebunkport, ME, 1994.

[29] Ehud Reiter. NLG vs. templates. In *Natural Language Generation*, Leiden, The Netherlands, 1995.

[30] Ehud Reiter. Shallow vs. deep techniques for handling linguistic constraints and optimisations. In *Proceedings of the Workshop on Natural Language Systems at the German Annual Conference on Artificial Intelligence*, Bonn, Germany, sep 1999.

[31] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3:57–87, 1997.

[32] Ehud Reiter and Chris Mellish. Optimizing the costs and benefits of natural language generation. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 1164–1171, 1993.

[33] S. Seneff. Response planning and generation in the MERCURY flight reservation system. *Computer Speech and Language*, 16(3–4):283–312, Jul–Oct 2002.

[34] Stephanie Seneff. TINA: A natural language system for spoken language applications. *Computational Linguistics*, 18(1):61–86, 1992.

[35] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. GALAXY-II: A reference architecture for conversational systems. In *Proceedings of the International Conference on Spoken Language Processing*, Sydney, Australia, September 1998.

[36] Stephanie Seneff and Joseph Polifroni. Dialogue management in the mercury flight reservation system. In *Proceedings of the conference of Applied Natural Language Processing and the North American Chapter of the Association for Computational Linguistics*, Seattle, WA, April 2000.

[37] Stephanie Seneff and Joseph Polifroni. Formal and natural language generation in the MERCURY conversational system. In *Proceedings of the 6th International Conference on Spoken Language Processing*, Beijing, China, October 2000.

[38] K. van Deemter, E. Krahmer, and M. Theune. Plan-based vs. template-based NLG: A false opposition? In *Proceedings of the Workshop on Natural Language Systems at the German Annual Conference on Artificial Intelligence*, Bonn, Germany, September 1999.

[39] Wolfgang Wahlster. Verbmobil: Translation of face-to-face dialogs. In *The Fourth Machine Translation Summit: MT Summit IV*, pages 127–135, Kobe, Japan, July 1993.

[40] M. Walker, O. Rambow, and M. Rogati. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*, 16(3–4):409–433, Jul–Oct 2002.

[41] Marilyn Walker and Owen Rambow. Spoken language generation. *Computer Speech and Language*, 16(3–4):273–281, Jul–Oct 2002.

[42] C. Wang, S. Cyphers, X. Mou, J. Polifroni, S. Seneff, J. Yi, and V. Zue. MUXING: A telephone-access Mandarin conversational system. In *Proceedings of the 6th International Conference on Spoken Language Processing*, Beijing, China, October 2000.

[43] V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. J. Hazen, and L. Hetherington. Jupiter: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85–96, 2000.

[44] V. Zue, S. Seneff, J. Polifroni, M. Nakano, Y. Minami, T. Hazen, and J. Glass. From JUPITER to MOKUSEI: Multilingual conversational systems in the weather domain. In *Proceedings of the Workshop on Multilingual Speech Communications*, Kyoto, Japan, October 2000.

[45] V. Zue, S. Seneff, J. Polifroni, M. Phillips, C. Pao, D. Goddeau, J. Glass, and E. Brill. Pegasus: A spoken language interface for on-line air travel planning. *Speech Communication*, 15:331–340, 1994.