Flexibility, Configurability and Optimality in UNL Deconversion via Multiparadigm Programming

Jorge Marques Pelizzoni, Maria das Graças Volpe Nunes

Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação
Av. do Trabalhador São-Carlense, 400. CEP 13560-970. São Carlos – SP – Brasil
{jorgemp, gracan}@icmc.usp.br
http://www.nilc.icmc.usp.br

Abstract. The fulfillment of the UNL vision is primarily conditioned on the successful deployment of deconverters, each translating from the UNL into a target language. According to current practice, developing deconverters ultimately means configuring DeCo, the deconversion engine provided by the UNDL Foundation. However, DeCo has a number of limitations that hinder productivity and might even preclude quality deconversion. This paper discusses some of these shortcomings and introduces an alternative deconversion model – Manati, which is the result of work on UNL-mediated Portuguese-Brazilian Sign Language human-aided machine translation. With Manati we attempt to exemplify how multiparadigm – namely, constraint, object-oriented and higher-order – programming can be drawn upon not only to specify an open-architecture, optimum-searching deconversion engine but also and above all to rationalize its configuration into deconverters for target languages.

1 Introduction

The fulfillment of the UNL vision [10, 11, 18] is primarily conditioned on the successful deployment of deconverters, each translating from the UNL into a target language. UNL deconversion is actually an instance of Natural Language Generation (NLG), which refers to rendering linguistic form to input in a non-linguistic representation. As pointed out by e.g. Reiter & Dale [13], Cahill & Reape [3], and Paiva [12], NLG can be a very complex task involving processing both linguistic (e.g. lexicalization, aggregation and referring expression generation) and otherwise (e.g. content selection and layout planning). The good news is that UNL deconversion is in fact restricted to the linguistic aspect of NLG, which can be termed **linguistic realization** and comprises the usual macro-level tasks of microplanning and surface realization. Therefore, one should naturally expect UNL deconversion to benefit from recent advances in Natural Language Generation and software development practice, for which reason UNL developers may need to go beyond the model underlying the DeConverter – or simply DeCo, the generic deconversion engine provided by the UNDL foundation.

In this paper we analyze DeCo both as a formal object and a software product, with an emphasis on discussing DeCo's features that may hinder productivity. In this analysis we adopt configurability (i.e. ease of configuration into full-fledged decon-

Universal Network Language: Advances in Theory and Applications.
Research on Computing Science 12, 2005, pp. 175–194.

verters), flexibility to accommodate application-specificities and support for optimality (i.e. search for optimal solutions) as meta-requirements for an ideal deconversion model. As a first attempt to meet these requirements and overcome DeCo's limitations, we conceived Manati, an alternative linguistic realization/UNL deconversion model. Manati exemplifies how multiparadigm – namely, constraint, object-oriented and higher-order – programming can be drawn upon not only to specify an open-architecture, optimum-searching deconversion engine but also and above all to rationalize its configuration into actual deconverters for target languages. In order to present important aspects of Manati's rationale, we introduce LIBRAS, the Brazilian Sign Language, and PUL∅, a UNL-mediated Portuguese-LIBRAS machine translation project, as it was PUL∅ that provided (i) the opportunity to experiment with DeCo and (ii) specificities that promptly exposed its limitations.

The paper proceeds as follows. Section 2 shortly introduces DeCo to the unacquainted; Section 3 states the LIBRAS case and introduces LIST (LIBRAS Script for Translation), a notation employed in some examples; Section 4 discusses DeCo's limitations; and Section 5 briefly describes Manati.

2 Meet DeCo

In this section, we review only those features of DeCo's which are essential to our discussion, i.e. just enough to illustrate how most effort is expended in DeCo's application. This is a very simplified overview especially to cater for the unacquainted with DeCo's abstract machine. For a thorough description, please refer to the *DeConverter Specifications* document provided by the *UNL Centre/UNDL Foundation*. It is worth mentioning that the terminology used in this section slightly differs from that of the referred document.

2.1 Configuration

In order to configure DeCo, i.e. prepare it to translate UNL hypergraphs into text in a specific target language, one must feed it with at least two basic *language-specific* resources, namely a UNL-target language **dictionary** and an ordered set of **deconversion rules**.

In short, each dictionary entry has a twofold function: (i) to declare a possible mapping of a UW[1] *Src* into a target language word or morpheme *Target*[2] and (ii) to state a set of **atomic (i.e. non-structured) features**[3] that should be assumed for *Target* whenever the declared mapping happens to be used. For example, supposing one intended to state that the UW "I" should be translated into English "I", "me", "my" or

[1] UWs (Universal Words) are UNL words. Formally, they are possible node labels in UNL graphs.

[2] Though rather unusual, *Target* might also be an intermediate symbol later to be erased.

[3] The term "feature" is herein employed much in the grammatical sense. In computer jargon, "flag" would be more appropriate.

"mine" under mutually exclusive conditions, there would usually be at least four distinct dictionary entries, as shown in **Table 1**. It is worth mentioning that developers are free to design their own set of possible features, as well as their respective meanings. The developers of the entries in **Table 1** seem to have found it interesting to encode the grammatical cases each English pronoun can accept (by means of features SUBJ and OBJ), parts-of-speech (PRO and DET), and person-cum-number information when needed (1PS, 3PS, and 3PP). Nevertheless, they might as well have found it more convenient e.g. to split the latter into independent features, some for person (1P and 3P) and others for number (PLU and SING), the only actual requirement being consistence. Finally, it should be noticed that a feature set belongs to the entry/mapping, not to the target word proper, as one would expect e.g. English "mine" to have a rather different feature set were the source UW "mine(icl>source)".

Table 1. Example UNL-English dictionary entries mapping the UW "I" into "I", "me", "my" or "mine"

UW	English	Features
I	I	{SUBJ, PRO, 1PS …}
I	me	{OBJ, PRO, 1PS …}
I	my	{DET, 3PS, 3PP …}
I	mine	{SUBJ, OBJ, PRO, 3PS, 3PP …}

In turn, the set of deconversion rules specify exactly *how* deconversion should be carried out, including when to access the dictionary. These rules are intrinsically procedural and somehow encode the grammar of the target language in terms of operations (sensing/writing/erasing) on features. Deconversion rules can only be correctly understood with DeCo's abstract machine in mind.

2.2 Abstract Machine

Roughly speaking, DeCo can be regarded as a non-deterministic Turing Machine fitted with some dictionary and graph lookup facilities. Its output is gradually built on an **extensible/retractable tape**, initially empty, by a pair of **ever-contiguous read/write (RW) heads**. At the end of deconversion, the sequence of tokens on the tape is printed out verbatim from left to right, and that should constitute the target text. However, if the tape were a mere string of tokens, it would have been of little use. As depicted in

Fig. 1, it is actually a string of cells each containing not only one single output token but also control data in the form of a rewritable feature set and *usually* a UW and a set of relations to nodes in the input UNL graph. The UW and relations can only be present if the cell results from the **transference** of an input node onto the tape, and those are precisely the UW of the referred node and all **unexplored** relations it has to other input nodes. A node is selected for transference by means of a relation it has to some **focused cell** (i.e. currently under one RW head); if the transference succeeds, then the referred relation is said to have been *explored.* Node transference is illustrated in **Fig. 2** and further explained below.

Much like bits of a Turing Machine's transition relation definition, deconversion rules are always relative to the pair of RW heads. Among other things, every rule may specify some of the following: (i) **preconditions** on the features, tokens or UWs of several existing cells (the two focused cells and variable-length sequences of cells on each side of and even *between*[4] the heads); (ii) similar preconditions on a new potential cell to be inserted; if this potential cell should result from the transference of a node, (iii) the label of the relation this node must have to one specific focused cell; (iv) features to be removed from or added to each focused cell (possibly a newly-inserted one); (v) whether to delete or replicate one specific focused cell; and, if the rule is not trying to insert or delete a cell, (vi) whether the heads should jointly slide one position to the left or right on completion. Naturally, a rule is applicable iff the preconditions (i) to (iii) are satisfied, and the actions of a rule are put into effect only if it is applicable and is actually selected for application.

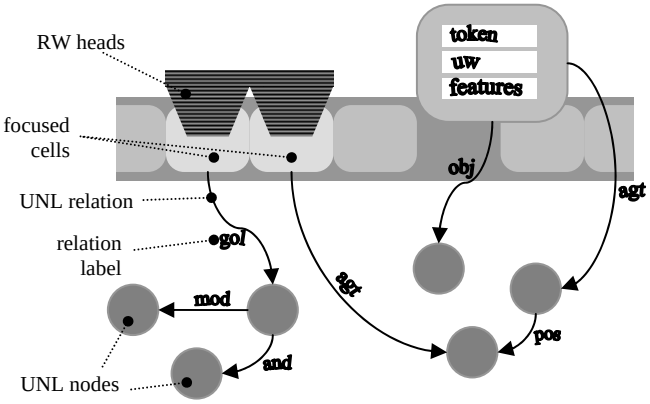

Fig. 1. Cells on DeCo's output tape

Two types of deconversion rules are of special interest to our discussion, namely:

- **feature-modifying rules**, which simply add or remove features to focused cells and optionally move the heads one position right or left on the tape. Features not only encode linguistic and conceptual information of the corresponding target language tokens, but also function as symbols in Turing Machines and much too often are used to simulate global states, implementing the rudiments of subroutines. For example, suppose there is the following top-priority set of rules (listed in order of priority. Notice this is not DeCo's actual notation):

```
if read(right, REWIND) and read(left, LEFT_DELIMITER)
   then erase(right, REWIND);
if read(right, REWIND)
   and not(read(left, LEFT_DELIMITER))
```

[4] This feature is only available to node-inserting rules and is explained later, when we tackle node-transferring rules, a specialization thereof.

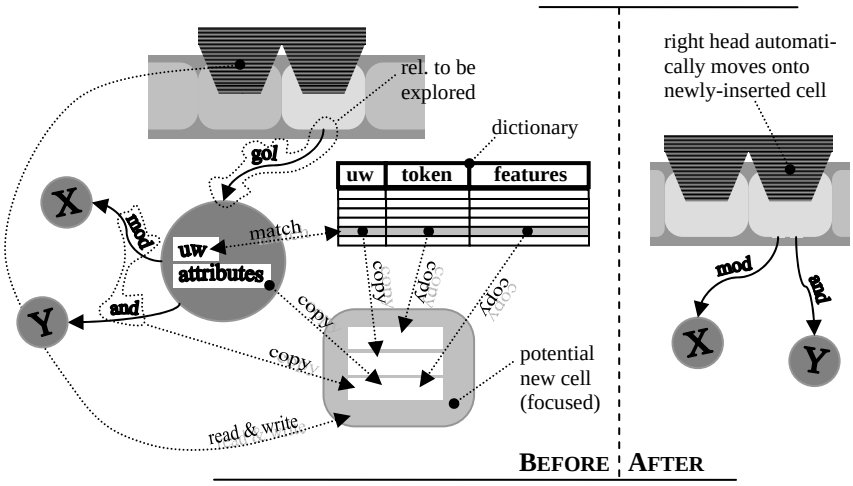

Fig. 2. Before and after node transference

```
then erase(right, REWIND),
     write(left, REWIND), move(left);
if read(left, REWIND) then move(left);
```

where `read(H,F)` is true iff head `H` can read `F` among the features of its respective focused cell, `write/erase(H,F)` adds/removes feature `F` to/from the cell focused by head `H`, and `move(D)` makes the heads move jointly one cell towards direction `D`. In this situation, a lower-priority rule writing REWIND roughly corresponds to calling a subroutine that will take the heads to the left end of the tape;

- **node-transferring rules**, which can also change the features of one focused cell, but whose most relevant effect is transferring one node from the UNL graph onto the tape and consequently expanding it. This is an all-important moment during deconversion as it is when the transferred node is "amalgamated" with one of its possible translations in the dictionary according to its UW, creating a new cell whose UW and output token are directly copied from the corresponding dictionary entry and whose feature set is initialized with (i) the features found in the entry, (ii) features homonymous to the UNL attributes found in the transferred node and (iii) features informative of the relations of the transferred node (e.g. a feature $>R$ or $<R$ indicates that the node plays respectively the left or right role in a relation labeled R). This process is depicted in Fig. 2. As an insertion rule may place preconditions on the feature set, token and UW of potential new cells, which are computed as just described, it follows that it is possible not only to state preconditions on the UNL attributes and relations of candidate nodes but also filter acceptable dictionary entries at all times.

One particularity of node-transferring rules, as well as node-inserting rules in general, is that the actual landing site of the new cell is not necessarily next to the **anchor cell** (i.e. the preexisting focused one, which provides the relation to be explored).

These rules may also specify a sequence of cells that should be present between the anchor cell and the cell to be inserted. Any such sequence is said to be *between the heads*, which is a very ephemeral state, as the heads again become contiguous right after insertion. It is exactly the mentioned particularity that allows e.g. the generation of discontinuous constituents, like the underlined subject in "A law was enacted during the previous administration that would require factories to reduce their emission of air pollutants by 70% over the next 3 years."

2.3 Nondeterministic Execution

DeCo executes nondeterministically in that it often reaches **choice points**, at each of which it has to choose from a priority-ordered set of alternative execution branches. DeCo then selects the highest-priority one to continue, but keeps track of the alternatives so that it may **backtrack** on failure. Backtracking consists of rolling execution back to the latest non-exhausted choice point, taking the highest-priority alternative branch not yet attempted and thus resuming execution. Unrecoverable failure, i.e. output consisting of an error message, arises from the exhaustion of all combinations of choices or usually timeout, due to combinatorial explosion. Success is restricted to the first good guess in depth-first search. Choice points are created whenever (i) there is more than one applicable rule or, during application of an insertion rule, (ii) there is more than one acceptable dictionary entry and/or eligible node for transference. It is worth mentioning that, whether implicitly or explicitly, developers statically stipulate the priority of every dictionary entry and deconversion rule.

Given one UNL graph as input, a configuration of DeCo tries to produce text in the target language of choice as follows: (i) DeCo starts with a tape containing only two predefined delimiter cells; (ii) it regularly transfers the entry node to the input graph onto the tape, between the delimiters; (iii) the right head is placed over the newly-inserted cell; (ii) DeCo iteratively applies rules until the right head tries to trespass the right end of the tape, the only sign of success; (iii) whenever DeCo gets stuck for lack of applicable rules, it tries to backtrack, unrecoverable failure arising from lack of non-exhausted choice points.

3 LIBRAS Testifies

This paper is one by-product of a very first attempt at Portuguese-LIBRAS[5] machine translation. This project is still under development but has provided enough opportunity to experiment with DeCo and put forth and implement the first draft of Manati, our alternative deconversion model. Naturally, neither DeCo nor Manati is ever intended to produce actual LIBRAS speech[6], but a script thereof – LIST (LIBRAS

[5] LIBRAS is an acronym for "LÍngua BRAsileira de Sinais", which is Portuguese for "Brazilian Sign Language".

[6] The words "spoken", "speech", etc. are employed here especially as opposed to "written", "writing", etc. Specifically, those words should not be regarded as necessarily implying *oral-*

Script for Translation) – to feed an eventual speech synthesizer. LIST is still in its infancy and shall be the result of a compromise between simplification of the translation apparatus and sufficiency for final synthesis.

To avoid some frequent misconceptions, it is worth reminding that sign languages are full-fledged languages on their own and usually rather dissimilar to their national oral counterparts. In fact, oral languages usually regarded as very different and thus translation–hard become in many respects closely related when sign languages come into scope. For the specific pair at issue, Portuguese and LIBRAS, we find that translation can be at times much harder than between Portuguese and English, for example. We shall present some evidence of this later, but plenty can be found elsewhere, as in Speers [15] and Brito [2]. Another point worth mentioning is that Linguistics have lately dedicated more and more attention to the subject, and most concepts and terms originally coined for the analysis of oral languages – such as "word", "phonetics", "phonology", "morphology", "syntax", and "prosody" – naturally apply and have been applied to sign languages as well.

LIBRAS is profligate in especially challenging problems for translation/deconversion and thus compelling examples, as several of Manati's features are thereby motivated. Consequently, an informal introduction to LIST is in order, so that LIBRAS examples can be presented.

First of all, LIST should not be expected to be readable by end-users. It is an interface protocol between two software modules: a translator and a speech synthesizer. If humans are ever to understand it, those must be the developers of those modules. Currently, LIST is biased towards ease of translation and, as much as possible, tries to approximate LIBRAS sentences with lists of **logograms**[7]. For the sake of readability, each logogram will herein be represented by a blank-delimited English string suggestive of its meaning in LIBRAS. For example, we show three LIST logograms below:

```
cut-with-a-knife old-man closed
```

LIST allows for tree-like structuring by means of **prosodic groups**, which are square-bracketed lists of logograms or other prosodic groups. This construct is to be used wisely and is just meant to include annotations strictly required by synthesis. The current prescriptions for prosodic group usage are beyond the scope of this paper, it sufficing to mention that every LIBRAS sentence is itself a prosodic group. Therefore, the following are examples of LISTified LIBRAS sentences: `[water still mosquito be-born grow]` ("Mosquitoes are born and grow in still water.") and `[I say water dangerous]` ("I told you water is dangerous.").

Finally, both logograms and prosodic groups as a whole may have associated **attribute-value matrices (AVMs)**, which allow e.g. (i) adding inflectional data to the logograms of the few inflected words of LIBRAS and (ii) annotating prosodic groups with the relevant prosodic information. An AVM is a curly-bracketed list of *Attrib-*

---

ity. In the case of sign languages, for example, speech synthesis involves actually moving an artificial communication actor, either by means of computer graphics or robotics.

[7] Each logogram is an atomic (i.e. non-analysable), strictly non-phonologically motivated symbol standing for a word. Chinese characters are examples of logograms; and, much though one can identify smaller component logograms inside a bigger Chinese character, the meaning of the latter can never be deduced from the former.

ute:Value pairs and is attached to the adjacent logogram or group to its left. When an *Attribute* is given without a value, *Attribute:true* is implied. For example,

```
[ ask{subjpers:2ps objpers:3pp} ]{imperative}
```

represents a one-word LIBRAS sentence meaning "Ask them!" and implies that LIBRAS `ask` agrees in person with its subject and object simultaneously, which are lexically absent in this sentence. Again, there are strict specifications ruling AVM usage, but they do not need to be covered here. It suffices to mention that, in addition to `subjpers` and `objpers`, attributes `subjgend`, `objgend` and `objlidgend` shall be used in examples and imply gender agreement of a verb with its subject, object and lid of its object (!), respectively.

4 DeCo Exposed

The craft of programming DeCo requires clockwork precision. Correct deconversion can be summed up as scheduling the transference of nodes with accuracy and handling cell features at the right times, since all things are global, flat, transient and public. Precise prioritizing of rules is the key. For example, supposing a verb must be preceded by its subject and object in that order, it follows that:

1. *subject insertion rules* must have priority over those for *object insertion*, as subject and object source nodes usually have direct UNL relations to verb cells;
2. the moment just after the insertion of a subject, in which it is adjacent to its verb, is the opportunity to solve whatever matters of agreement between them by means of rules that add specific features to the cell of the verb according to features they read in the cell of the subject. These *agreement rules* must thus have priority over object insertion;
3. agreement rules must read the right subject features; therefore, just in case e.g. we are dealing with a compound subject, there are likely to be rules computing the *sum* of agreement features (e.g. 1PS + 3PS = 1PP). These *agreement sum rules* must thus have priority over agreement;
4. subject insertion as a whole cannot simply have priority over agreement sum, as the insertion of nested modifying noun phrases may hinder the sum. Agreement sum must thus be *interleaved* with subject insertion;
5. sometimes and especially for languages with more than two number features (e.g. LIBRAS, the Brazilian Sign Language), the exact number of a noun phrase is not given by the source node of its nucleus (e.g. "those two girls"). Thus, at least some noun modifier insertion rules must have priority over agreement sum.

Obviously, the tasks above are error-prone, since each of these rule subsets (subject/object insertion, agreement, etc.) usually contains numerous low-level, hardly readable rules involving various artificial control features to keep DeCo's abstract machine on track. Furthermore, priority can be implemented not only by rule ordering but also – and often – implicitly, by careful positioning of the heads, which is always a must, anyway. Frequently a whole process is triggered by one single rule waiting on a certain feature/command under e.g. the left head only. The developer

then choreographs the heads ingeniously so that the left head will only pass over the trigger at the right time. This is known to be a major source of unmanageability but is hard to avoid in real DeCo programming.

In the following sections, we discuss DeCo's limitations from two perspectives: firstly, as a formal object and, finally, as a software product.

4.1 Formal Limitations

In this section, we demonstrate some undesirable consequences of DeCo's formal specifications, which may hinder if not preclude operations necessary to quality deconversion.

Precondition Language

As routine a phenomenon as verb agreement suffices to demonstrate maybe the greatest among DeCo's limitations, namely the absolute simplicity of the precondition specification language. Cells do not hold attribute-value matrices, just plain feature sets; each feature is literally atomic; and the precondition language is equivalent to predicate logic. This means that, even though features like (i) *pers=1ps* and (ii) *pers=3pp* are possible, they appear to preconditions as if unrelated. Therefore, for each and every possible person feature a subject may assume, there must be a distinct rule to generate the corresponding information in the verb. It is simply *not possible* to express something like:

```
if read(left,SUJ) and read(right,V) and
   read(left,pers=$X) and not(read(right,pers=$_))
then write(right,subjpers=$X)
```

with `$X` and `$_` as variables. Neither would `read($X,SUJ)` be possible, implying that, if subjects were to be generated now to the right, now to the left of verbs, then there would have to be distinct rules to deal with each side, doubling the number of agreement rules.

In general, implementing any n-ary function (i.e. one head writing a specific value whenever n features of the form $Param_i=Val_i$ are read) takes as many rules as the cardinality of its domain, or rather, the *product* of the cardinalities of the domains of its parameters. If exactly the same function should yield its result now in the left, now in the right focused cell, taking its parameters from the other focused cell, then that number of rules doubles.

Linear, Non-Structured Output

Much of the awkwardness exemplified in the previous topic is due to the fact that DeCo gradually builds a linear tape of otherwise formally unrelated cells, which ultimately – and implicitly – stands for a highly structured entity, i.e. some sentence/text in a target language. Surely deconversion rules are designed to impose e.g. agreement and positioning constraints between syntactic constituents. The nuisance is that these constraints can never be expressed in terms of real syntactic structure. Configuration would be more natural if deconversion could be analyzed as if including two decoupled steps as follows:

- **syntactic mapping,** in which real syntactic nodes were created and explicitly related to each other as if, given e.g. a verb node and a noun phrase root node, the developer could simply say "Verb, this is your subject!"; and
- **governor-governee constraining**, in which one could simply state "this class of verbs agrees in person with its subject and object" or "this other class agrees in gender with its subject" and rely on the deconversion engine to impose these constraints during syntactic mapping implicitly.

Subgraph Matching

Deconversion would simply not be possible if rules were not able to sense the input UNL graph, even if only from the limited point of view of a focused cell. In fact, DeCo allows node-transferring rules to inspect no further than exactly one node directly linked to one focused cell (the other head is over the inspected node, so to speak). It is left to the other rules at most to sense relation-related features, i.e. those of the form $>R$ or $<R$ indicating that the inspected cell plays the left or right role in some relation labeled *R*.

Therefore, if ever a higher-level translation step requires inspecting/matching a less limited subgraph as a purely *semantic* precondition, then a cumbersome routine is in order of transferring the whole subgraph onto the tape and next deleting the undesired cells. This has an extreme side-effect: if the precondition succeeds, then the relations of the subgraph will have been explored and thus can never be traversed again by other translation steps. Unless the referred step coincidentally consumes the whole subgraph, that side-effect is unacceptable. In short, DeCo does not support general subgraph matching, which represents a heavy constraint on the expression of semantic preconditions.

Even if such a transfer-inspect-delete routine happens to be acceptable in a particular case, not only will it be difficult to choreograph, but also it will entail the creation of several undesired lexical bindings and related choice points, as the dictionary is necessarily accessed. Moreover, for reasons explained in the previous topic, the implementation of one such routine can seldom be reused by similar semantic preconditions on subgraphs.

Graph Editing and Nontrivial Maneuvers

If graph sensing is as restrained as described in the previous topic, graph editing follows closely, receiving the status of a mere side-effect. In fact, a relation can be considered erased once it has been explored in node transference, as it can never be explored again. However, more sophisticated operations are usually most welcome. Consider, for example, a real LIBRAS generation case in which informers seemed to neutralize the difference between English (i) "to keep something Xed" and (ii) "to X something", producing one single LIBRAS version reflecting (ii) more closely. One actual translation pair was the following:[8]

[8] All real-case source sentences are originally in Portuguese. However, whenever the differences between English and Portuguese are not relevant, we show only English translations to improve readability.

source: We must keep water tanks closed!
target: `[water tank must close-with-lid{objlidgend:flat}]{excl}`

The problem here is that, in order to produce the same target sentence, the source could as well be “We must close water tanks!”, which would actually bear a more direct structural relation to the target. We managed to tackle this neutralization with DeCo at the expense of the following rather awkward ad-hoc strategy:

1. add as many entries under UW “closed” as to replicate all the possible mappings of UW “close”;
2. make sure that the features of these new entries would avoid their application on trivial deconversion of UW “closed”;
3. make the entry under UW “keep” map into an empty token and contain some common verbal features that would trigger subject insertion and aggreement;
4. add one special feature to that entry triggering a complex procedure as follows:
5. explore the obj relation to insert the root cell of the object;
6. copy subject agreement information from the “keep” cell into the object cell;
7. prior to full object development, use the object cell as an anchor to transfer the node accessible through the aoj relation, requiring the inserted cell to be a verb. This new verb cell will be inserted either to the left or right according to its own feature set, which should inform the required relative position of an object;
8. copy subject agreement information from the object cell into the verb cell;
9. develop the object fully;
10. copy object agreement information into the verb cell and so on.

Complex though it may seem, the description above is much simpler than the actual implementation, which involves subtle rule prioritizing and choreographing. It is worth noticing that we needed to implement and activate a whole rule set alternative to regular verb insertion rules. Were graph editing operations available in a decoupled form, it would have been much neater to perform a purely semantic transformation as depicted in Fig. 3 and next apply regular rules to the new root node. No ad-hoc entries would have to be added to the dictionary then; on the other hand, an additional semantic resource would be needed to map UW “closed” into “close”.

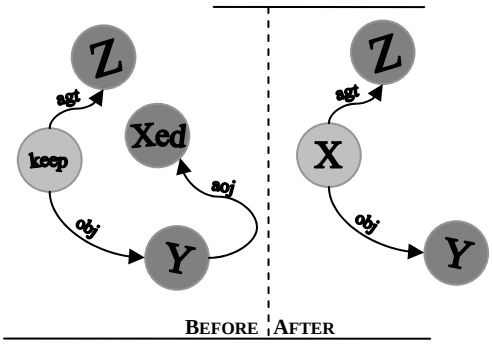

Fig. 3. Graph editing operation implementing the neutralization between “to keep something Xed” and “to X something”

Optimality

DeCo is strict in that (i) a solution is depth-first searched for and (ii) the first solution found is the only one to be outputted. This is built in DeCo's design and is acknowledged to rule out any chance of defining a quality measure to optimize. Even if it were possible to relax (ii) and have some external device eventually rank all the solutions, DeCo would probably timeout when searching for alternatives, since failure by timeout should always be enabled even for toy configurations, so many the created choice points usually are.

Therefore, all such concepts as models of conciseness/readability/etc. (see e.g. Eddy [8]) or Optimality-Theoretical soft constraints are rendered inapplicable. However, it is possible to implement simple rules of thumb locally to decide whether a constituent (e.g. a relative clause) should be generated now in a position, now in another based e.g. on its length. Unfortunately, this requires some nontrivial programming and risks so much backtracking as to lead to timeout. DeCo would have to be instructed to (i) generate the whole constituent first in one position, suitably delimited by special markers; (ii) check the size constraint on the basis of those markers; and, if the constraint were not satisfied, (iii) force failure in order eventually to try alternative rules starting generation in a different position.

4.2 Architectural Limitations

As a piece of software design, DeCo's architecture is perfectly closed in that at no time can user-defined modules aid deconversion. In other words, neither dictionary entries nor rules can refer or resort to any entity whatsoever outside the standard system, which can be a serious limitation to some applications. For a start, even if graph editing facilities were available, the semantic neutralization scheme exemplified in the previous section could not be implemented in this scenario, as it requires a special semantic resource to map e.g. UW "closed" into "close".

However, the need for interoperability becomes patent when one take into account that at times the source UNL graph may lack pieces of information essential to quality or even grammatical deconversion. This situation is frequent when LIBRAS is the target language. Sometimes a UW is just too general, like "cut", which simply has no direct translation. LIBRAS "cut" signs necessarily incorporate an instrument; therefore, there are only specific signs such as `cut-with-a-knife`, `cut-with-scissors`, `cut-with-a-saw` and so on. It is worth noticing that mistranslation would lead then to ungrammatical, unintelligible or at least seriously misleading sentences. Hence the need for a semantic resource to answer such queries as "With which instrument is X usually cut?"

In some cases, even a knowledge base, which could suffice for answering most such queries, is not enough. Take grammatical number in LIBRAS for example, which may assume as many as five values, namely singular, dual, trial, quadral and plural. According to current UNL codification standards, there is rather likely to be a simple node with attributes `@def` (definite) and `@pl` (plural) actually referring to a group of entities already mentioned by some preceding UNL graph in UNL-encoded text. Moreover, it may well be the case that the actual cardinality of the group has been made explicit in that or yet another previous reference. If so and the current

sentence presents any kind of person agreement, then this cardinality – in the form of a corresponding number value – will be paramount to producing a sound LIBRAS version. Due to DeCo's closed architecture, there is no chance it would do anything but blindly guess in such a situation.

It is actually a non-issue here what the exact nature of such external devices should be. What is required is that a deconversion engine should be able to query them, and developers should be free to develop and employ any number of devices necessary in a specific application. To mention one less usual example, in our Portuguese-LIBRAS translation project, named PUL∅ (Portuguese-UNL-LIST DeOralizer), we intend to overcome such accuracy-demanding challenges as mentioned above by resorting to **human aid**, though strictly non-specialized in that only required to be proficient in the source-language (Portuguese). This means PUL∅ withdraws from real-time translation and resigns its operation to what we call edition (i.e. pre-publishing) time.[9] Therefore, in addition to a minimized knowledge base, PUL∅'s deconversion engine shall query an interactive device which, whenever necessary, should reply on the basis of a human editor's answers to questions elaborated on the fly.

5 Meet Manati

Manati is the linguistic realization engine all UNL-LIST conversion in PUL∅ is based on. In other words, PUL∅ includes a configuration of Manati, i.e. a module obtained by fixing Manati's parameters. It should be clarified at once that Manati, much unlike DeCo, is not an application, but a **software framework** or simply a **library** to serve as a foundation for UNL deconversion modules/systems. This should not be regarded as a disadvantage, actually being particularly favorable to **interoperability**. For example, auxiliary devices external to Manati, such as user prompts or knowledge bases, can be directly built in the final application; and the power of a full-fledged programming language is available to help handle complex translation procedures. The framework is fully implemented in Oz (www.mozart-oz.org [14, 17]) and heavily draws upon the expressiveness and elegant, seamless multiparadigm integration of this language to meet its requirements. The following description assumes some familiarity with the terminology of higher-order, constraint and especially object-oriented programming.

Manati[10] is undoubtedly DeCo's child. The parentage is not only historical – as it was only after experimenting with DeCo that Manati could be conceived, and it is in DeCo's shortcomings that one can find much of Manati's rationale – but also conceptual. Several features embryonic in DeCo have been generalized and above all *de-*

[9] Taking into account how rare and costly bilingual human translators are in this case, one can easily understand how reasonable this tradeoff is.

[10] Manati is named in honor of its idol and definitive evolution-perfected form, the legendary Babel Fish [1], which feeds upon mixed-up brain-wave energy and absorbs all but intentional linguistic thought. Just as manatis are not really fish, Manati is not a Babel Fish and strives to digest the UNL into one target language at a time.

coupled in Manati. Manati's lexicon-driven **delegation model** is perhaps the most outstanding of its resemblances to DeCo, although each lexicon entry now states not simply a mapping, but rather a **translation rule** triggered by a UW. Each rule covers an arbitrary subgraph, inspects and changes the input graph at will, builds an arbitrary portion of the output and eventually delegates the translation of other adjacent subgraphs by invoking translation rules for boundary nodes via the lexicon.

Manati takes **decoupling** seriously. The very concept of a translation rule is not atomic, being the crossing of four orthogonal concepts – semantic precondition, syntactic mapping, governor-governee constraints and linear precedence constraints – each of which are implemented separately by four distinct class hierarchies. Rules are obtained by combining classes from these hierarchies interchangeably. At all times, class definition is supported by high-level constructs, e.g. syntactic dependency trees [7] and morphosyntactic feature structures in syntactic nodes. It follows that Manati produces **highly-structured output**, which, nonetheless, can straightforwardly be printed out as a sentence.

Efficiency and **optimality** are also major concerns. The ultimate goal of configuring Manati is instruct it to derive a low-level constraint satisfaction problem (CSP) description, effectively exploiting propagation, when it is fed with input. Naturally, a quality measure should integrate the derived CSP; and search for an optimum solution is carried out as usual in constraint programming. This programming paradigm was chosen due to its potential to reduce search dramatically. Its application in conjunction with the dependency tree formalism follows work by Duchier [4][5] & Debusmann [7], which focused on parsing. Their research also inspired Koller & Striegnitz's generation work [9], which is, however, fundamentally distinct from ours in that it strictly focuses on taming flat semantics, a non-issue here.

5.1 Parameters

Manati currently allows the rationalized configuration of ten *orthogonal* parameters in that independently and modularly defined, namely:

1. **input formalism**, which, even though restricted to hypergraph types, is free to accept any open set of UWs (node labels) and closed set of relations (edge labels);
2. **morphosyntax:** each part of speech (POS) in the target language must be defined as a record with arity *{avm, constr}*, where feature *avm* is an attribute-value matrix (AVM) type, and *constr*, a constraint on instances of *avm*. Whenever a morpheme *M* is generated with part of speech *P*, then *M.feats* denotes a unique morphosyntactic feature structure for which:

   $$M.feats \in P.avm \wedge P.constr(M.feats)$$

   holds. Furthermore, given that *M.roles* denotes the actual label set of all syntactic relations having *M* as a governor, the invariants:

   $$M.feats.reqComps \subseteq M.roles$$
   $$M.roles \subseteq M.feats.reqComps \cup M.feats.optComps$$

also hold, meaning that POSs must necessarily define at least features *reqComps*, specifying required syntactic relations (as to complements), and *optComps*, specifying optional ones (as to adjuncts).

POS declaration in Manati is extremely user-friendly, allowing inheritance hierarchies and expressiveness in defining AVM types building on work by Duchier et al. [6]. Attribute types can be any of (i) atom from a finite domain, (ii) set of atoms from a finite domain, (iii) the cartesian product of such sets or (iv) nested AVM. The *contsr* features of POSs have intuitive notational support (also due to Duchier et al.) and are useful when stating e.g. that a given set of nouns/verbs imply specific gender/tense;

3. **syntactic mapping:** a specific mapper class hierarchy must be provided in order exclusively to specify the mapping of UNL (hyper)graphs onto syntactic dependency trees in the target language. Roughly speaking, mappers simply convert (i) semantic nodes into lexemes (classes of lexical items) and (ii) semantic relations into syntactic roles; or, in Natural Language Generation jargon, they are responsible for *lexical choice* and *aggregation* [13]. It is worth noticing that mappers are not interested either in morphosyntatic constraints, such as agreement, or in final linear ordering of morphemes.

   During generation, according to information in the lexicon (see below), a set of *mutually exclusive* mappers are instantiated for the global UNL entry node *Src*. Each mapper (i) tries to recognize a specific subgraph of its own starting at *Src*, performing whatever necessary semantic checking on candidate nodes, (ii) creates a set of syntactic nodes (usually corresponding to target language words) and (iii) establishes binary syntactic relations between them. Some of the nodes in (ii) may be created by means of recursively applying the same process to some of the source nodes in the subgraph recognized in (i), as mappers always yield exactly one syntactic root node. In time, the root *received* by a mapper as a result of recursion might actually be a *selector node* choosing from the root set of several mutually exclusive subtrees produced by alternative mappers.

   If the mapper class hierarchy is well-defined and correctly employed in the lexicon, the process sketched above will traverse the source UNL graph tree-wise from its global entry. For more complex operations such as generating relative clauses and dealing with coordination – or sometimes simply to avoid infinite cycling – some input formalisms (and UNL flavours) require that mappers be able to edit source hypergraphs. The edit operations available are node insertion and edge deletion and insertion. In any case, however, changes by a mapper are only visible to itself and the mappers it creates recursively;

4. **mapping preconditions:** in order to optimize resource usage during search, part of if not all precondition checking in mappers can optionally be delegated to a specific class hierarchy. Such so-called *precond* classes are associated with mappers by lexicon data. See “lexicon” below for details;

5. **governor-governee constraints:** a specific class hierarchy must be provided in order exclusively to tell morphosyntactic constraints on each pair of syntactically related target nodes (i.e. words or morphemes). Such so-called *gamma* classes are associated with mappers by lexicon data and have methods of the signature

Role(Parent.feats Child.feats) invoked for each syntactic relation *Role* their corresponding mappers establish between any target nodes *Parent* (governor) and *Child* (governee);

6. **linear precedence:** a specific class hierarchy must be provided in order exclusively to determine the final ordering of target nodes and carry out whatever further tasks that might occasionally be required on mapper completion, when all direct child nodes are accessible – though not as yet fully determined – for e.g. telling further constraints. Such so-called *finishUp* classes tackle linear precedence by telling constraints relating target nodes to each of their children and siblings to each other. Order constraints, though definable at various levels of abstraction, ultimately operate on features *roots* and *yield* of nodes or *role bundles* – a simplified interface to all siblings filling one same syntactic role. Feature *roots* denotes a set containing either the absolute position of a node within the generated text or the union of all *roots* features of the siblings in a role bundle. Feature *yield* denotes the union of either all *roots* in the subtree rooted at a node or all *yield* features of the siblings in a role bundle. If needed, role bundles also give access to each "bundled" sibling individually.

   Following Duchier & Debusmann [7], ultimate control over linear precedence is provided by constraints operating also on *topological fields*. The concept involves axiomatizing a *topology* of the *yield* of a syntactic tree, i.e. a *partition P[i]* such that:

$$\forall x,y,i,j\ (x \in P[i] \wedge y \in P[j] \wedge i < j \rightarrow x < y).$$

   Each partition element *P[i]* is said a *topological field*. Manati allows absolute flexibility in axiomatizing topologies, including the possibility of nesting, one topology holding for an entire tree unless a mapper overrides it for subtrees. Nested topological fields can be unified with those in an overridden topology granting finer-grained overriding control.

   Topologies provide for long-distance "movements", topicalization, nested clauses and the generation of multiple sentences from a single source graph[11], which is essential for Libras generation;

7. any number of **oracles** – e.g. user prompts, knowledge bases, etc. – to resort to at virtually any generation stage. Oracles are services running concurrently and accepting asynchronous requests. The sole requirements on oracles are (i) requests must be ground, i.e. involving no unbound variables, and (ii) responses must be either ground or finite domain variables (in some commonly agreed protocol, e.g. 0/1 meaning true/false) eventually to be determined by oracles themselves;

8. **lexicon:** Manati's lexicon is more of a **translation rule base**, each of whose entries is a tuple *(UW, TransList, POS, Precond, Mapper, Gamma, FinishUp)*, where *UW* is a source node label; *TransList*, a character string list of possible target lan-

[11] Notice that it is always possible to define a rightmost/leftmost topological field to contain trailing/preceding text.

guage translations; *POS*, the part of speech of the elements of *TransList*; and *Precond*, *Mapper*, *Gamma* and *FinishUp*, classes of the homonymous types.

When the translation of a source node is required, its label is used to search the lexicon for eligible transfer rules. For each rule, *Precond* is activated, performs specified checking and, iff *TransList* has more than one element, must select exactly one of its elements to be the translation word of the rule. *TransList* may as well be empty, but then the definition of a translation word is left to *Mapper*, which hinders code reuse and search efficiency. The *null* word is also possible, creating an invisible syntactic node and enabling null categories.

From this point on, Manati attempts to optimize the application of constraint programming by instantiating one single mapper for each set of so far successful rules sharing the same *Mapper* class. Each mapper receives a default syntactic target node constructed from the data remaining in its originating set of rules, i.e. translation words, *POS*s, *Gamma*s and *FinishUp*s. This is actually a complex two-level selector node built with the powerful Oz selection constraints. It is up to each mapper to decide what to do with its default target node: (i) simply ignore it (not wasteful due to lazy evaluation) or (ii) use it as a final target to receive children or likewise (iii) as part of any arbitrary syntactic structure it may build;

9. **output formalism,** i.e. how the resulting syntactic trees are to be printed out. This is highly configurable ranging smoothly from raw lists of target language words to fully structured trees by means of user-defined bracketing. Words and bracketed groups may be associated with arbitrary *Output AVMs* (OAVMs) created by *FinishUp* classes. OAVMs may be useful to add syntactic and prosodic annotations (as required by PUL∅) or even to output morphologic features, leaving full inflection of words to dedicated modules and thus downsizing the lexicon. These are the facilities that allow the generation of prosodic groups and AVMs in LIST;

10. **quality measure**, in the form of a binary constraint *Q* that, during search, is iteratively imposed on pairs *(CurBest, Wannabe)*, where *CurBest* is the best fully determined solution so far, and *Wannabe* is a partially determined solution which will attempt to be even better than *CurBest*. In fact, it is exactly *Q* that should give *Wannabe* a drive to supersede by strictly constraining it to be better. As *CurBest* and *Wannabe* are given as the roots of their respective syntactic tree solutions, and *Wannabe* is not yet fully determined, which rules out direct access to its subtrees, constraint *Q* should relate both solutions solely on the basis of their roots. Therefore, for complex quality measures, developers are expected to include quality-related features in POSs (see "morphosyntax" above) and have them propagate up the tree by means of special constraints in *gamma* and *finishUp* classes.

    So far we have only experimented with minimizing output length in words. This is especially interesting to LIBRAS generation because it is rather often the case that two or more distinct words can actually be combined into a single preferred one. As Manati provides every node with feature `yieldCard`, denoting the cardinality of its *word yield*,[12] this quality measure is optimized by a simple procedure `LengthOrder` as follows:

[12] The *word yield* of a node is the subset of its *yield* that actually corresponds to target language words, which excludes null categories and bracketed block nodes.

```
proc {LengthOrder CurBest Wannabe}
    Wannabe.yieldCard <: CurBest.yieldCard
end
```

where `X <: Y` is not the usual comparison operation, but rather a constraint, telling that `X < Y` should always hold. Other measures are usually of interest to systems relying on e.g. heavy content selection and advanced referring expression generation, which are almost absent in PUL∅ since these tasks are satisfactorily performed by source text authors to current standards. Alternatively, if one is interested in the very first solution only, it suffices to provide the following even simpler constraint:

```
proc {FirstWillDo _ _}
    fail
end
```

5.4 Searching for a Global Optimum

Deconversion starts by applying translation rules to the global UNL entry node. In spite of involving some pattern matching and search, this corresponds to model creation only and yields a complex partially determined syntactic tree, whose distinct potentialities are modelled by occasional **higher-order selector nodes** choosing from (the roots of) a set of subtrees. In addition to `yieldCard`, every node has two further important features *affected by constraint propagation*, namely: **`id`**, denoting its absolute position in the generated text, and **`active`**, denoting an encoded boolean telling whether the node actually takes part in the current solution or is discarded. Every higher-order selector node has at most one active selectable root at a time and is active iff it has exactly one such root. If so, that root becomes selected, which makes its features (`id`, `active`, `yieldCard`, etc.) and those of the selector coincide. Finally, all other nodes are actually **first-order selectors** choosing from a list of alternative target language words, for which reason they have an additional **`lexI`** feature, denoting the index of the word of choice.

Manati's search script is just like any ordinary Oz script and is executed in cycles of constraint propagation followed by domain distribution until a solution is found. It reads as follows:

1. distribute over the vector of all *active* features, prioritizing (i) activation over deactivation and (ii) elements in order of appearance, which roughly corresponds to the order in which translation rules appear in the lexicon;
2. *ActiveNds* ← list of all active first-order selector nodes. The notation *List.Feature* used in subsequent steps denote the vector obtained by selecting *Feature* for each item in *List*;
3. tell $\forall Id \in ActiveNds.id: dom(Id) \subseteq \{1 \ldots length(ActiveNds)\}$;
4. distribute naïvely over *ActiveNds.lexI*, i.e. trying lower values first;
5. distribute over *ActiveNds.id* using a first-fail strategy, i.e. prioritizing the distribution of the most constrained `id`s as an heuristic to rule out failed choices first and thus minimize their impact on search;

6. if a fully determined solution *CurBest* is found, try to improve it by starting over with a fresh model *Wannabe* and ensuring that *Q(CurBest,Wannabe)* should hold, for a given quality constraint *Q*.

6 Conclusions and Future Work

We have scrutinized DeCo and demonstrated that some of its features are likely to hinder productivity and quality in deconverter development. These features can be summarized as strong coupling of concepts, lack of generality, low level of abstraction and no support for modularity, abstraction, optimality or interoperability.

As a first attempt to overcome these shortcomings, we have presented the first draft of Manati, an alternative linguistic realization/UNL deconversion engine. Manati heavily draws on constraint programming as a means to reduce search; while object-oriented and higher-order programming provides a basis for defining friendly primitives with which (i) to fill the blanks (i.e. parameters) of a configuration at appropriate levels of abstraction and (ii) automatically to derive a low-level constraint satisfaction problem (CSP) description, effectively exploiting propagation, when a configuration is eventually fed with input.

Manati is currently being configured to generate the Brazilian Sign Language and shall be evaluated against other linguistic realization engines in the near future. Scheduled further work on Manati includes full coverage of generation tasks [13] – e.g. content selection – and experiments with different quality measures. Additionally, as our experience of applying Manati in real-case scenarios increases, we expect to produce even higher-level abstractions building on Manati's current facilities.

Acknowledgements This project has been partially funded by MEC (the Brazilian Ministry of Education). We would like to thank Prof. Dr. Tanya Amara Felipe (UPE – Universidade de Pernambuco) and her team for providing the LIBRAS versions of this article.

References

1. Adams, D. *The Hitchhiker's Guide to the Galaxy*. Ballantine Books, 1995 (reissue edition).
2. Brito, L. F. Por uma Gramática de Línguas de Sinais. Tempo Brasileiro Ed., Departamento de Lingüística e Filologia, Universidade Federal do Rio de Janeiro, 1995.
3. Cahill, L. and Reape, M. *Component tasks in applied NLG systems*. Technical Report ITRI-99-05, Information Technology Research Institute (ITRI), University of Brighton, 1998. http://www.itri.brighton.ac.uk/projects/rags.
4. Duchier, D. Configuration of labeled trees under lexicalized constraints and principles, *Journal of Language and Computation*, 2002.
5. Duchier, D. Axiomatizing dependency parsing using set constraints. In *Proceedings of the 6th Meeting on the Mathematics of Language*, USA, 1999.
6. Duchier, D., Gardent C., and Niehren J. *Concurrent Constraint Programming in Oz for Natural Language Generation*. http://www.ps.uni-sb.de/~niehren/Web/Vorlesungen/Oz-NL-SS01/vorlesung (accessed on Aug. 2004).

7. Duchier, D. and Debusmann, R. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the Association for Computational Linguistics (ACL)*, France, 2001.
8. Eddy, B. Toward balancing conciseness, readability and salience: an integrated architecture. In *Proceedings of the International Natural Language Generation Conference (INLG'02)*, 2002.
9. Koller, A. and Striegnitz, K. Generation as Dependency Parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2002, 17-24.
10. Martins, R. T., Rino, L. H. M., Nunes, M. G. V., and Oliveira Jr., O. N. The UNL distinctive features: evidences through a NL-UNL encoding task. In *Proceedings of the First International Workshop on the UNL, other Interlinguas and their Applications*, 2002, 08-13.
11. Martins, R. T. *A Língua Nova do Imperador*. Ph.D. Thesis, Instituto de Estudos da Linguagem, Universidade Estadual de Campinas (UNICAMP), 2004.
12. Paiva, D. *A survey of applied natural language generation systems*. Technical Report ITRI-98-03, Information Technology Research Institute (ITRI), University of Brighton, 1998. http://www.itri.brighton.ac.uk/techreports.
13. Reiter, E. and Dale, R. *Building Natural Language Generation Systems*. Cambridge University Press, 2000.
14. Schulte, C. *Programming Constraint Services: High-Level Programming of Standard and New Constraint Services*, Lecture Notes in Computer Science Series, Springer-Verlag, 2002.
15. Speers, d'A. L. *Representation of American Sign Language for Machine Translation.* PhD. dissertation, Graduate School of Arts and Sciences, Georgetown University, 2001.
16. Stone, M. and Doran, C. Sentence planning as description using tree-adjoining grammar. In *Proceedings of the Association for Computational Linguistics*, 1997, 198-205
17. Van Roy, P. and Haridi, S. *Concepts, Techniques, and Models of Computer Programming*, MIT Press, 2004.
18. Uchida, H., Zhu, M. and Della Santa, T. *UNL: A Gift for a Millennium.* Institute of Advanced Studies, University of the United Nations, 1999. http://www.undl.org/publications.