Mapping knowledge to language with LOOM

Manfred Stede
(University of Toronto and)
FAW Ulm
Helmholtzstr. 16
89081 Ulm, Germany
stede@faw.uni-ulm.de

Dietmar Rösner
FAW Ulm
Helmholtzstr. 16
89081 Ulm, Germany
roesner@faw.uni-ulm.de

Abstract

In language generation, one of the central tasks is to map a "deep" representation of some kind to lexical units, from which a grammatical utterance can be produced. Unless a strict one-to-one relationship between concepts and words is assumed, which, we argue, is too simplistic for the general case, some sort of structure-mapping process is required for accomplishing this step. We investigate the mechanisms that one particular knowledge representation language of the KL-ONE family has to offer for this purpose, and compare their relative merits.

1 Overview

The question of associating words with semantic or conceptual entities arises in language analysis and generation alike. In this paper, we are concerned with *generation* from a KL-ONE style knowledge representation and argue that the organization of a domain model can in general not be expected to correspond to linguistic distinctions required for verbalization. Specifically, for the link between concepts and lexical items a one-to-one mapping between the two cannot be assumed, and therefore some kind of mapping process is needed in the verbalization step. Having identified the major desiderata for such a mapping scheme, we explore the various possibilities that the KR language we are using, Loom [MacGregor and Bates, 1987] has to offer for these purposes and compare their suitability. This work is part of the TECHDOC project at the Research Center for Applied Knowledge Processing (FAW) Ulm, where a multilingual generation system for technical instruction manuals is being built [Rösner and Stede, 1992].

2 "Deep" representations and their verbalization

2.1 Concepts are not the same as words

It has often been lamented (e.g., [Marcus, 1987; McDonald, 1991]) that language generators do not perform genuine lexical *choice*, due to the fact that concepts and words are supposed to be in a one-to-one correspondence. That is, the input structure to a generator fully or almost fully determines the lexical outcomes. On the one hand, such a correspondence precludes the selection of words from sets of synonyms that have the same denotation but differ in terms of connotations or stylistic color (*die, pass away, kick the bucket*). Enabling this choice amounts to a 1:n mapping from concepts to words.

Furthermore, a simple correspondence assumes the same *granularity* of the concept base and the lexicon. Quite often, though, specific lexical items make distinctions that are of no relevance for the underlying reasoning component and thus do not warrant the existence of such specific concepts in the KB. In the terminology of KL-ONE languages, one wants to associate words with concepts plus particular role-filler restrictions. For example, if the word *eat* is linked to the concept EAT, then *gobble* might be associated with EAT plus a role MANNER and its filler FAST. This way, we avoid the introduction of concepts for the sole purpose of adequate verbalization; the issue becomes even more important in multilingual settings where different languages provide lexemes of different specificity. Separating lexical from conceptual distinctions is crucial here, and by "conceptual" we mean those distinctions that are geared towards the *purpose* of the KB and the reasoning system, for which verbalization is (usually) only a secondary aspect.

Thus, verbalizing a conceptual representation is more than a mere lookup of one word per concept; besides the choice among synonyms it involves a *matching* step where the meaning of lexical items is compared to the parts of the conceptual structure, and a set of items is sought that together expresses the complete structure (and no more than that). Computationally speaking,

the problem is one of subgraph matching.

Several proposals for achieving this task have been made in the literature. Miezitis [Miezitis, 1988] suggests a variant of *marker passing* along the network representation, an essentially procedural approach. Working with the *conceptual graphs* formalism, Nogier and Zock [Nogier and Zock, 1992] describe a way of successively replacing semantic sub-graphs with others representing syntactic objects, thereby treating the whole generation process as a matching task. In the KL-ONE framework, Horacek [Horacek, 1990] suggested four "zoom" schemata that associate lexical items with various types of concept/role configurations, but does not discuss implementational issues. The present paper deals with the "how to do it" question and investigates what mechanisms supplied by one particular KL-ONE-like language, Loom, can be used to perform the matching step, so that additional programming can be reduced to a minimum (in particular, one need not implement a graph matcher on one's own). We address only the retrieval task, i.e., the step of finding all candidate words that could possibly participate in a verbalization. For brevity, we will henceforth call the structure to be verbalized the *event representation* ER and the entities representing lexical items *verbalization objects* VOs.

2.2 A specific example

In our system, we aim at being able to generate a fairly wide range of paraphrases from the same underlying meaning representation. For the rest of the paper, we will use as an example an event of a person named Bob putting water into some tank, possibly up to a certain mark imprinted on it. Such an event can be expressed by utterances like the following, which can focus on different aspects of the situation:
(1) *Water filled the tank.*
(2) *The tank filled with water.*
(3) *Bob filled the tank to the MAX mark with water.*
(4) *Bob added water to the tank until the level reached the MAX mark.*
These and other sentences are all to be derived from the same abstract representation, possibly with varying parameters, which therefore needs to be fairly fine-grained. The situation can be described as follows: There is a transition from a particular fill-state of the tank to another one, where the second is characterized by the level being at MAX, and the first by its being somewhere below MAX. The transition is brought about by the fact that the fluid level in the tank moves upward, which in turn is caused by Bob pouring water into the tank. — As we focus solely on the technical aspects of word–concept linking in this paper, we will not discuss our ontological/representational choices here. Thus, we simply take our ER, written in Loom and illustrated in figure 1, as given. Boxes contain instance names and concept identifiers (in capital letters), or only a name if it is just a symbol.

Word meanings correspond to subgraphs, and five relevant ones are shown in figure 2. In argument lists (whose theoretical status is, again, not our topic here), optional arguments are enclosed in square brackets. An indentation amounts to following an arc, i.e., a relation in the graph; upper-case letters denote concept names (types), and x^Y is a restriction that the filler of role x has to be of type Y. Entities in angle brackets are default fillers that are taken whenever no actual value is instantiated in ER. For example, the inchoative reading of FILL has one obligatory and two optional arguments. It denotes a transition between two fill-states of some tank. If the content of the tank is not instantiated, the verbalization *The tank filled* is appropriate (the representation ignores tense and aspect). If the content is present, it can be expressed in a *with*-PP: *The tank filled with water* (which needs to be specified in the syntax part of the lexical entry). The default goal-state of the tank is 'full', but it can be overwritten, as in *The tank filled with water, up to the second mark.* The other lexeme-meanings are to be read in the same way. The first step of verbalization is to notice which lexemes are candidates for expressing parts of the message: there is a *pouring*, a *rising*, a *reaching*, an inchoative *filling*, a causative *filling*, an *until* relationship involved, and so forth. The first prerequisite is encoding the VOs in Loom.

2.3 Desiderata for encoding VOs

A central task of verbalization is to map ER entities to argument positions, in particular for verbs. The representation should be expressive enough to account for filler-type restrictions, for the optionality of arguments, and for defaults. On the other hand, the items thus represented need to be retrieved (efficiently, if possible!) when the ER is verbalized. This process should account for inheritance, and hence find VOs representing specific as well as more general lexical items.

As a consequence of the fine-grained representation, VOs can be rather complex and extend over paths of several connected instances: in FILL_INCH, for example, the y argument is the filler of the content role of the object role of the pre-state role of the transition. Not every encoding can cope with such paths, as we will see.

For many applications it can be desirable to keep the "proper" knowledge base separate from the VOs, so that the KB is kept conceptually clean and VOs don't disturb any reasoning processes. This requirement is especially important in multilingual settings, where one would end up with a wealth of language-specific entities that in effect denote the same conceptual object. Finally, an obvious target is to avoid redundant storage of information, and to minimize extra programming, i.e., to explore the built-in facilities of Loom as much as possible. And, as a related goal, representations should be declarative, readable and changeable; the best ones (from this viewpoint)

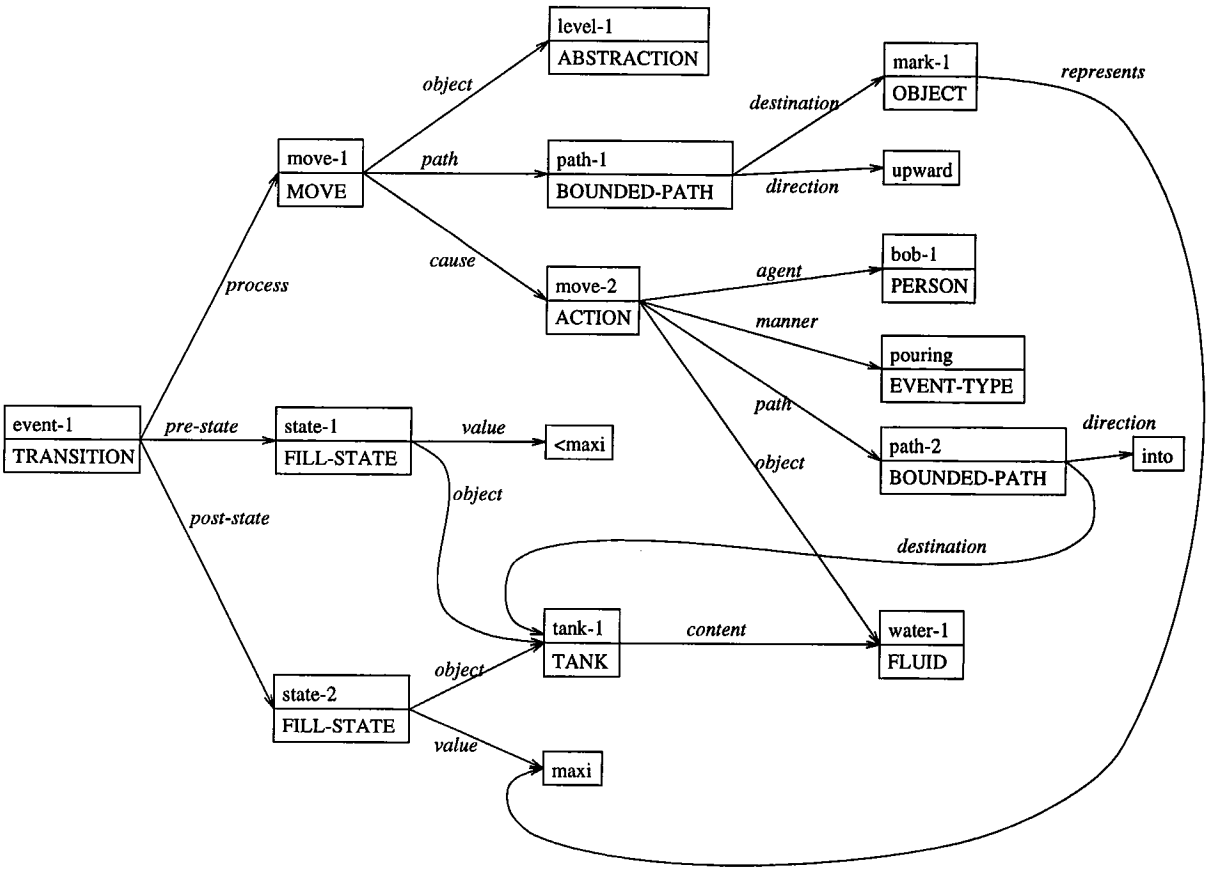

Figure 1: Representation of tank-fill event

```
pour (x y z)
ACTION
  agent x
  object y
  path z
  manner pouring

rise (x [y])
TRANSITION
  process^MOVE
    object x
    path
      direction upward
      destination y

reach (x y)
TRANSITION
  process^MOVE
    object x
    path^BOUNDED-PATH
      destination y
  post-state
    value y

fill_caus (x y [w] [z])
TRANSITION
  pre-state^FILL-STATE
    object^TANK y
      content w
  post-state^FILL-STATE
    object^TANK y
      content w
    value z <maxi>
  process^MOVE
    cause^ACTION
      agent x

fill_inch (x [y] [z])
TRANSITION
  pre-state^FILL-STATE
    object^TANK x
      content y
  post-state^FILL-STATE
    object^TANK x
      content y
    value z <maxi>
```

Figure 2: Representations of a few word meanings

would differ as little as possible from those in figure 2.

3 Possible solutions using LOOM

An ER is a number of instances of concepts that stand in relations to one another. In Loom (version 2.0), VOs can, in principle, be encoded as either concepts, relations, instances, production rules, or methods. We will review these possibilities in turn and see how well they meet the aforementioned desiderata: complex VOs, argument mapping, optional arguments, defaults, retrieval of VOs, inheritance, separation of conceptual KB and VOs, storage efficiency, extra programming, declarativeness. We will refer to the sample codings for the VO *rise* given in figure 3. Obviously, we cannot explain the syntax or the functionality of Loom in this abstract, but readers familiar with KL-ONE languages should be able to follow the discussion.

3.1 VOs as concepts

VO-concepts can either form a separate taxonomy dedicated to language-specific lexicalization, or mix with the standard KB. They are sub-concepts of the VO's "root" type (e.g., for *rise* it is TRANSITION) and add the specific conditions that render the VO applicable for verbalizing the ER. Thereby, this approach is limited to non-complex VOs: while it is possible to *check* for the presence of particular arguments on some path of instances (in example [1] the test for the movement being upward), one cannot "carry over" arguments from distant instances: if 'upward' were not a fixed object but an instance that needed to be mapped to an argument position, this would not be possible. For mapping "local" objects to argument positions, the device concept-annotation can be used, which Loom offers for purposes that can't be achieved effectively within the language proper.

On the other hand, the elegance of this approach stems from exploiting the Loom classifier for finding applicable VOs: ER instances are just being re-classified in a taxonomy of VOs, whereby we are given the most specific VO automatically. More general ones can be determined by moving upward in the taxonomy and collecting the VOs, which is not difficult.

If the VO taxonomy is kept separate from the "ordinary" KB, one needs to duplicate many concepts and relations, but the reasoning KB is kept clean. If they are mixed together, one loses clarity, which might be problematic in certain applications.

3.2 VOs as relations

Coding a VO as relation [2] makes argument mapping very straightforward, as the call `(retrieve (?x ?y) (rise ?x ?y))` will return them: `(level-1 maxi)`. VOs can be complex because paths of arbitrary length can be built—but a variable bound by `:for-some` is

	Concepts	Relations	Instances	Productions	Methods
complex VOs	3	1	3	1	1
arg. mapping	2	1	2	1	1
optional args	3	3	2	1	1
defaults	3	3	2	1	1
retrieval	1	2	1	1 (speed?)	1 (speed?)
inheritance	1	3	3	3	2
KB/VOs separate	1/3	1	(1)	3	1
storage	3/1	1	1	1	1
programming	1	2	2	1	1
declarative	1	1	2	2	2

Figure 4: Rating of LOOM's verbalization methods

needed at every step, and this retrieval is costly. Optional arguments are difficult to realize, as every variable in the query needs to be bound and it is not possible to perform arbitrary Lisp function calls within a relation definition. For the same reason, defaults are problematic. The major shortcoming of this approach is that *finding* VOs needs to be programmed, i.e., one has to determine oneself when to execute what `retrieve` commands. Consequently, inheritance of more general items is not provided. A further, significant weakness lies in the fact that Loom does not offer relation retrieval when their arity is >2; but more than two arguments to a VO is a common occurrence. This can be finessed by embedding several binary relations, but then the representation becomes rather clumsy.

3.3 VOs as instances

VOs can be encoded as instances of the concepts they serve to verbalize [3]. The primary advantages are that the conceptual KB remains unaffected by the VOs and that retrieval of candidate VOs is trivial: just query the database for all instances of the concept in question that are recognizable as VOs (for example, by having a pointer to a lexicon entry). On the other hand, the VO instances mix with the "ordinary" KB instances, which can be undesirable for reasoning and retrieval; but future versions of Loom promise to offer a context mechanism where sets of instances can be kept separate from one another, which would solve this problem.

Inheritance of more general VOs does not come for free; one has to program an "instance subsumption checker" oneself. This is not difficult as long as only one instance and the types of its fillers are considered (it has been implemented, see [Stede, 1993]), but with more complex VOs that encompass paths of instances, much more programming effort is required. Besides, the VO representation is split over several instances, as illustrated in figure 3. Optional arguments and defaults can be accounted for, since the subsumption checker has to be hand-coded anyway, or they can be represented by using more than one instance—one with the optional argument present, the other without it.

3.4 VOs as production rules

When writing VOs as productions [4], Loom's built-in matcher will find the applicable VOs for an ER automatically. The `:detects` part of a rule describes the situation necessary for it to fire, and the `:do` part gives the action to be executed (arbitrary Lisp code). While in `:detects` there is no disjunction of conditions allowed, considerable flexibility is at hand in the `do` part. Argument mapping is done via co-referential variables, much as in the relation case [2]. In example [4], the presence of an optional argument is checked for and returned only if present. Defaults can be handled in the same way. There are no principled restrictions on the complexity of VOs, but we have not yet tested the efficiency of the matcher when a large KB and a large number of productions are used.

As for inheritance, rules representing more general VOs will fire together with the specific one, but the subsumption relation holding between them is not transparent: the VOs produced by the rules do not indicate whether they subsume one another or not, hence this information needs to be maintained separately.

The problem with productions that needs to be finessed somehow is their permanent residence in the system: as soon as they are loaded, Loom's matcher will constantly check their applicability, which is clearly unacceptable in a somewhat larger system.

3.5 VOs as methods

Loom also offers aspects of the object-oriented programming paradigm: it is possible to encode VOs as methods that respond to a "verbalize yourself" message sent to the root-instance of the ER. This approach is very similar to the production rule one; Loom will automatically find those methods whose `situation` description matches the instance that the message was sent to. Again, we have not yet determined the efficiency of the

```
[1]
(defconcept rise :is
  (:and transition
        (:the has-process move_event)
        (:filled-by (:compose has-process
                              has-path has-direction)
                    'upward)))
%-----------------------------------------------
[2]
(defrelation rise :is
  (:satisfies (?x ?y)
    (:for-some (?t ?pr ?pa)
      (:and (transition ?t)
            (has-process ?t ?pr)
            (move_event ?pr)
            (has-object ?pr ?x)
            (has-path ?pr ?pa)
            (:about ?pa
              (:filled-by has-direction 'upward))
            (has-destination ?pa ?y)))))
%-----------------------------------------------
[3]
(tell (:about rise transition
      (has-process move-rise)))
(tell (:about move-rise
      (has-object x)
      (has-path path-rise)))
(tell (:about path-rise
      (:filled-by has-direction 'upward)
      (has-destination y)))
%-----------------------------------------------
[4]
(defproduction rise
  :when (:detects (:and (transition ?t)
                        (has-process ?t ?pr)
                        (move_event ?pr)
                        (has-object ?pr ?x)
                        (has-path ?pr ?pa)
                        (has-direction ?pa
                           'upward)))
  :do ((if (get-value ?pa 'has-destination)
           (list 'rise ?x (get-value ?pa
                           'has-destination))
           (list 'rise ?x))))
%-----------------------------------------------
[5]
(defmethod verbalize (?t)
  :title "rise"
  :situation (:and (transition ?t)
                   (:for-some (?x ?pr ?pa)
                      (:and (has-process ?t ?pr)
                            (move_event ?pr)
                            (has-object ?pr ?x)
                            (has-path ?pr ?pa)
                            (has-direction ?pa
                              'upward))))
:response ((if (get-value ?pa 'has-destination)
               (list 'rise ?x
                 (get-value ?pa 'has-destination))
               (list 'rise ?x))))
```

Figure 3: Alternative Loom codings for a VO

matcher when working in large KBs. As a possible advantage over production rules, with methods it is possible to tell the matcher to find only the most specific applicable method, so with respect to inheritance the situation is a little better. Also, the verbalization process is started explicitly by sending the message to the ER—we do not have the problem of activating and de-activating, as with production rules.

4 Summary

Different applications and different styles of doing knowledge representation and lexical semantics will emphasize the various factors dicussed above to different degrees. Hence, none of the schemes for encoding VOs is universally the best. But it is important to know the differences, and the table in figure 4 sums them up by assigning grades between 1 (best) and 3 (worst) to each criterion. Note, however, that this method yields a somewhat simplified picture, as often the factors depend on one another.

In the column 'concepts', sometimes two grades are listed: the first applies to an approach using a separate taxonomy for VOs, the second for the one that mixes them with the standard KB (i.e., one trades storage efficiency against conceptual clarity). The parentheses in the 'instances' column indicate that the current version of Loom does not offer the facility, but a future version will.

References

[Horacek, 1990] Helmut Horacek. The Architecture of a Generation Component in a Complete Natural Language Dialogue System. In R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*, chapter 8, pages 193–228. Academic Press, London, 1990.

[MacGregor and Bates, 1987] Robert MacGregor and Robert Bates. The Loom Knowledge Representation Language. Technical Report ISI/RS-87-188, USC/Information Sciences Institute, 1987.

[Marcus, 1987] Mitchell Marcus. Generation Systems Should Choose Their Words. In Yorick Wilks, editor, *Theoretical Issues in Natural Language Processing*, pages 211–214. New Mexico State University, Las Cruces, 1987.

[McDonald, 1991] David D. McDonald. On the Place of Words in the Generation Process. In C. L. Paris, W. R. Swartout, and W. C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 227–248. Kluwer, Dordrecht, 1991.

[Miezitis, 1988] Mara Anita Miezitis. Generating Lexical Options by Matching in a Knowledge Base. Technical Report CSRI-217, University of Toronto, 1988.

[Nogier and Zock, 1992] Jean-Francois Nogier and Michael Zock. Lexical Choice as Pattern Matching. In T.E. Nagle, J.A. Nagle, L.L. Gerholz, and P.W. Eklund, editors, *Conceptual Structures: Current Research and Practice.* Ellis Horwood, New York, 1992.

[Rösner and Stede, 1992] Dietmar Rösner and Manfred Stede. TECHDOC: A System for the Automatic Production of Multilingual Technical Documents. In G. Görz, editor, *Proceedings of KONVENS-92, First German Conference on Natural Language Processing.* Springer, Berlin/Heidelberg, 1992. Also as FAW-TR.

[Stede, 1993] Manfred Stede. Lexical Options in Multilingual Generation from a Knowledge Base. In *Working notes of the Fifth European Workshop on Natural Language Generation*, Pisa, 1993.