USING CLASSIFICATION TO GENERATE TEXT

Ehud Reiter* and **Chris Mellish†**
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
BRITAIN

ABSTRACT

The IDAS natural-language generation system uses a KL-ONE type classifier to perform content determination, surface realisation, and part of text planning. Generation-by-classification allows IDAS to use a single representation and reasoning component for both domain and linguistic knowledge, which is difficult for systems based on unification or systemic generation techniques.

Introduction

Classification is the name for the procedure of automatically inserting new classes into the correct position in a KL-ONE type class taxonomy [Brachman and Schmolze, 1985]. When combined with an attribute inheritance system, classification provides a general pattern-matching and unification capability that can be used to do much of the processing needed by NL generation systems, including content-determination, surface-realisation, and portions of text planning. Classification and inheritance are used in this manner by the IDAS natural language generation system [Reiter *et al.*, 1992], and their use has allowed IDAS to use a single knowledge representation system for both linguistic and domain knowledge.

IDAS and I1

IDAS

IDAS is a natural-language generation system that generates on-line documentation and help messages for users of complex equipment. It supports user-tailoring and has a hypertext-like interface that allows users to pose follow-up questions.

The input to IDAS is a point in *question space*, which specifies a basic question type (e.g., What-is-it), a component the question is being asked about (e.g., **Computer23**), the user's task (e.g. Replace-Part), the user's expertise-level (e.g., Skilled), and the discourse in-focus list. The generation process in IDAS uses the three stages described in [Grosz *et al.*, 1986]:

- *Content Determination*: A content-determination rule is chosen based on the inputs; this rule specifies what information from the KB should be communicated to the user, and what overall format the response should use.
- *Text Planning*: An expression in the ISI Sentence Planning Language (SPL) [Kasper, 1989] is formed from the information specified in the content-determination rule.
- *Surface Realisation*: The SPL is converted into a surface form, i.e., actual words interspersed with text-formatting commands.

I1

I1 is the knowledge representation system used in IDAS to represent domain knowledge, grammar rules, lexicons, user tasks, user-expertise models, and content-determination rules. The I1 system includes:

- an automatic classifier;
- a default-inheritance system that inherits properties from superclass to subclass, using Touretsky's [1986] minimal inferential distance principle to resolve conflicts;
- various support tools, such as a graphical browser and editor.

An I1 knowledge base (KB) consists of classes, roles, and user-expertise models. User-expertise models are represented as KB overlays, in a similar fashion to the FN system [Reiter, 1990]. Roles are either *definitional* or *assertional*; only definitional roles are used in the classification process. Roles can be defined as having one filler or an arbitrary number of fillers, i.e., as having an inherent 'number restriction' of one or infinity.

An I1 class definition consists of at least one explicitly specified parent class, primitive? and individual? flags, value restrictions for definitional

*E-mail address is E.Reiter@ed.ac.uk
†E-mail address is C.Mellish@ed.ac.uk

roles, and value specifications for assertional roles. I1 does not support the more complex definitional constructs of KL-ONE, such as structural descriptions. The language for specifying assertional role values is richer than that for specifying definitional role value restrictions, and allows, for example: *measurements* that specify a quantity and a unit; *references* that specify the value of a role in terms of a KL-ONE type role chain; and *templates* that specify a parametrized class definition as a role value. The general design goal of I1 is to use a very simple definitional language, so that classification is computationally fast, but a rich assertional language, so that complex things can be stated about entities in the knowledge base.

An example I1 class definition is:

```
(define-class open-door
 :parent open
 :type defined
 :prop
  ((actor animate-object)
   (actee door)
   (decomposition
    ((*template*
      grasp
      (actor = actor *self*)
      (actee = (handle part) actee *self*))
     (*template*
      turn
      (actor = actor *self*)
      (actee = (handle part) actee *self*))
     (*template*
      pull
      (actor = actor *self*)
      (actee = (handle part) actee *self*))
    ))))
```

This defines the class **Open-door** to be a **defined** (non-primitive and non-individual) child of the class **Open**. **Actor** and **Actee** are definitional roles, so the values given for them in the above definition are treated as definitional value restrictions; i.e., an **Open-Door** entity is any **Open** entity whose **Actor** role has a filler subsumed by **Animate-Object**, and whose **Actee** role has a filler subsumed by **Door**.

Decomposition is an assertional role, whose value is a list of three templates. Each template defines a class whose ancestor is an action (**Grasp**, **Turn**, **Pull**) that has the same **Actor** as the **Open-Door** action and that has an **Actee** that is the filler of the **Part** role of the **Actee** of the **Open-Door** action which is subsumed by **Handle** (i.e., **(handle part)** is a differentiation of **Part** onto **Handle**).

For example, if **Open-12** was defined as an **Open** action with role fillers **Actor**:**Sam** and **Actee**:**Door-6**, then **Open-12** would be classified beneath **Open-Door** by the classifier on the basis of its **Actor** and **Actee** values. If an inquiry was issued for the value of **Decomposition** for **Open-12**, the above definition from **Open-Door** would be inherited, and, if **Door-6** had **Handle-6** as one of its fillers for **Part**, the templates would be expanded into a list of three actions, (**Grasp-12** **Turn-12** **Pull-12**), each of which had an **Actor** of **Sam** and an **Actee** of **Handle-6**.

Using Classification in Generation

Content Determination

The input to IDAS is a point in question space, which specifies a basic question, component, user-task, user-expertise model, and discourse in-focus list. The first three members of this tuple are used to pick a *content-determination rule*, which specifies the information the generated response should communicate. This is done by forming a rule-instance with fillers that specify the basic-question, component, and user-task; classifying this rule-instance into a taxonomy of content-rule classes, and reading off inherited values for various attributive roles. A (simplified) example of a content-rule class definition is:

```
(define-class what-operations-rule
 :parent content-rule
 :type defined
 :prop
  ((rule-question what)
   (rule-task operations)
   (rule-rolegroup
    (manufacturer model-number colour))
   (rule-function
     '(identify-schema :bullet? nil))))
```

Rule-Question and **Rule-Task** are definitional roles that specify which queries a content rule applies to; **What-Operations-Rule** is used for "What" questions issued under an Operations task (for any component). **Rule-Rolegroup** specifies the role fillers of the target component that the response should communicate to the user; **What-Operations-Rule** specifies that the **manufacturer**, **model-number**, and **colour** of the target component should be communicated to the user. **Rule-Function** specifies a Lisp text-planning function that is called with these role fillers in order to generate SPL. Content-rule class definitions can also contain attributive roles that specify a human-readable title for the query; followup queries that will be presented as hypertext clickable buttons in the response window; objects to be added to the discourse in-focus list; and a testing function that determines if a query is answerable.

Content-determination in IDAS is therefore done entirely by classification and feature inheritance;

once the rule-instance has been formed from the input query, the classifier is used to find the most specific content-rule which applies to the rule-instance, and the inheritance mechanism is then used to obtain a specification for the KB information that the response should communicate, the text-planning function to be used, and other relevant information.

IDAS's content-determination system is primarily designed to allow human domain experts to relatively easily specify the desired contents of short (paragraph or smaller) responses. As such, it is quite different from systems that depend on deeper plan-based reasoning (e.g. [Wahlster *et al.*, 1991; Moore and Paris, 1989]). Authorability is stressed in IDAS because we believe this is the best way to achieve IDAS's goal of fairly broad, but not necessarily deep, domain coverage; short responses are stressed because IDAS's hypertext interface should allow users to dynamically choose the paragraphs they wish to read, i.e., perform their own high-level text-planning [Reiter *et al.*, 1992].

Text Planning

Text planning is the only part of the generation process that is not entirely done by classification in IDAS. The job of IDAS's text-planning system is to produce an SPL expression that communicates the information specified by the content-determination system. This involves, in particular:

- Determining how many sentences to use, and what information each sentence should communicate (text structuring).
- Generating referring expressions that identify domain entities to the user.
- Choosing lexical units (words) to express domain concepts to the user.

Classification is currently used only in the lexical-choice portion of the text-planning process, and even there it only performs part of this task.

Text structuring in IDAS is currently done in a fairly trivial way; this could perhaps be implemented with classification, but this would not demonstrate anything interesting about the capabilities of classification by generation. More sophisticated text-structuring techniques have been discussed by, among others, Mann and Moore [1981], who used a hill-climbing algorithm based on an explicit preference function. We have not to date investigated whether classification could be used to implement this or other such text-structuring algorithms.

Referring expressions in IDAS are generated by the algorithm described in [Reiter and Dale, 1992]. This algorithm is most naturally stated iteratively in a conventional programming language; there does not seem to be much point in attempting to re-express it in terms of classification.

Lexical choice in IDAS is based on the ideas presented in [Reiter, 1991]. When an entity needs to be lexicalized, it is classified into the main domain taxonomy, and all ancestors of the class that have lexical realisations in the current user-expertise model are retrieved. Classes that are too general to fulfill the system's communicative goal are rejected, and preference criteria (largely based on lexical preferences recorded in the user-expertise model) are then used to choose between the remaining lexicalizable ancestors.

For example, to lexicalize the action (**Activate** with role fillers **Actor:Sam** and **Actee:Toggle-Switch-23**) under the Skilled user-expertise model, the classifier is called to place this action in the taxonomy. In the current IDAS knowledge base, this action would have have two realisable ancestors that are sufficiently informative to meet an instructional communicative goal,[1] **Activate** (realisation "activate") and (**Activate** with role filler **Actee:Switch**) (realisation "flip"). Preference criteria would pick the second ancestor, because it is marked as basic-level [Rosch, 1978] in the Skilled user-expertise model. Hence, if "the switch" is a valid referring expression for **Toggle-Switch-23**, the entire action will be realised as "Flip the switch".

In short, lexical-choice in IDAS uses classification to produce a set of possible lexicalizations, but other considerations are used to choose the most appropriate member of this set. The lexical-choice system could be made entirely classification-based if it was acceptable to always use the most specific realisable class that subsumed an entity, but ignoring communicative goals and the user's preferences in this way can cause inappropriate text to be generated [Reiter, 1991].

In general, it may be the case that an entirely classification-based approach is not appropriate for tasks which require taking into consideration complex pragmatic criteria, such as the user's lexical preferences or the current discourse context (classification may still be usefully used to perform part of these tasks, however, as is the case in IDAS's lexical-choice module). It is not clear to the authors how the user's lexical preferences or the discourse context could even be encoded in a manner that would make them easily accessible to a classifier-based generation algorithm, although perhaps this simply means that more research needs to be done on this issue.

[1] The general class **Action** is an example of an ancestor class that is too general to meet the communicative goal; if the user is simply told "Perform an action on the switch", he will not know that he is supposed to activate the switch.

Surface Realisation

Surface realisation is performed entirely by classification in IDAS. The SPL input to the surface realisation system is interpreted as an I1 class definition, and is classified beneath an *upper model* [Bateman *et al.*, 1990]. The upper model distinguishes, for example, between **Relational** and **Nonrelational** propositions, and **Animate** and **Inanimate** objects.[2] A new class is then created whose parent is the desired grammatical unit (typically **Complete-Phrase**), and which has the SPL class as a filler for the definitional **Semantics** role. This class is classified, and the realisation of the sentence is obtained by requesting the value of its **Realisation** role (an attributive role).

A simplified example of an I1 class that defines a grammatical unit is:

```
(define-class sentence
 :parent complete-phrase
 :type defined
 :prop
 ((semantics predication)
  (realisation
   ((*reference*
      realisation subject *self*)
    (*reference*
      realisation predicate *self*)))
  (number
   (*reference* number subject *self*))
  (subject
   (*template*
    noun-phrase
    (semantics = actor semantics *self*)))
  (predicate ...)
  ...))
```

Semantics is a definitional role, so the above definition is for children of **Complete-Phrase** whose **Semantics** role is filled by something classified beneath **Predication** in the upper model. It states that

- the **Realisation** of the class is formed by concatenating the realisation of the **Subject** of the class with the realisation of the **Predicate** of the class;
- the **Number** of the class is the **Number** of the **Subject** of the class;
- the **Subject** of the class is obtained by creating a new class beneath **Noun-Phrase** whose semantics is the **Actor** of the **Semantics** of the class; this in essence is a recursive call to realise a semantic constituent.

If some specialized types of **Sentence** need different values for **Realisation**, **Number**, **Subject**, or another attributive role value, this can be specified by creating a child of **Sentence** that uses I1's default inheritance mechanism to selectively override the relevant role fillers. For example,

```
(define-class imperative
 :parent sentence
 :type defined
 :prop
  ((semantics command)
   (realisation
    (*reference*
      realisation predicate *self*))))
```

This defines a new class **Imperative** that applies to **Sentences** whose **Semantics** filler is classified beneath **Command** in the upper model (**Command** is a child of **Predication**). This class inherits the values of the **Number** and **Subject** fillers from **Sentence**, but specifies a new filler for **Realisation**, which is just the **Realisation** of the **Predicate** of the class. In other words, the above class informs the generation system of the grammatical fact that imperative sentences do not contain surface subjects. The classification system places classes beneath their most specific parent in the taxonomy, so to-be-realised classes always inherit realisation information from the most specific grammatical-unit class that applies to them.

The Role of Conflict Resolution

In general terms, a classification system can be thought of as supporting a pattern-matching process, in which the definitional role fillers of a class represent the pattern (e.g. `(semantics command)` in **Imperative**), and the attributive roles (e.g., **Realisation**) specify some sort of action. In other words, a classification system is in essence a way of encoding pattern-action rules of the form:

$$\alpha_1 \rightarrow \beta_1$$
$$\alpha_2 \rightarrow \beta_2$$
$$\ldots$$

If several classes subsume an input, then classification systems use the attributive roles specified (or inherited by) the most specific subsuming class; in production rule terminology, this means that if several α_i's match an input, only the β_i associated with the most specific matching α_i is triggered. In other words, classification systems use the *conflict resolution* principle of always choosing the most specific matching pattern-action rule.

This conflict-resolution principle is used in different ways by different parts of IDAS. The content-determination system uses it as a preference mechanism; if several content-determination rules subsume an input query, any of these rules can be used to generate a response, but presumably the most appropriate response will be generated by the most specific subsuming rule. The

[2] The IDAS upper model is similar to a subset of the PENMAN upper model.

lexical-choice system, in contrast, effectively ignores the 'prefer most specific' principle, and instead uses its own preference criteria to choose among the lexemes that subsume an entity. The surface-generation system is different yet again, in that it uses the conflict-resolution mechanism to exclude inapplicable grammar rules. If a particular term is classified beneath **Imperative**, for example, it also must be subsumed by **Sentence**, but using the **Realisation** specified in **Sentence** to realise this term would result in text that is incorrect, not just stylistically inferior.

The 'use most specific matching rule' conflict-resolution principle is thus just a tool that can be used by the system designer. In some cases it can be used to implement preferences (as in IDAS's content-determination system); in some cases it can be used to exclude incorrect rules which would cause an error if they were used (as in IDAS's surface-generation system); and in some cases it needs to be overridden by a more appropriate choice mechanism (as in IDAS's lexical choice system).

Classification vs. Other Approaches

Perhaps the most popular alternative approaches to generation are unification (especially functional unification) and systemic grammars. As with classification, the unification and systemic approaches can be applied to all phases of the generation process [McKeown *et al.*, 1990; Patten, 1988].[3] However, most of the published work on unification and systemic systems deals with surface realisation, so it is easiest to focus on this task when making a comparison with classification systems.

Like classification, unification and systemic systems can be thought of as supporting a recursive pattern-matching process. All three frameworks allow grammar rules to be written declaratively. They also all support unrestricted recursion, i.e., they all allow a grammar rule to specify that a constituent of the input should be recursively processed by the grammar (IDAS does this with I1's template mechanism). In particular, this means that all three approaches are Turing-equivalent. There are differences in how patterns and actions are specified in the three formalisms, but it is probably fair to say that all three approaches are sufficiently flexible to be able to encode most desirable grammars. The choice between them must therefore be made on the basis of which is easiest to incorporate into a real NL generation system.

[3]Although it is unclear whether unification or systemic systems can do any better at the text-planning tasks that are difficult for classification systems, such as generating referring expressions.

We believe that classification has a significant advantage here because many generation systems already include a classifier to support reasoning on a domain knowledge base; hence, using classification for generation means the same knowledge representation (KR) system can be used to support both domain and linguistic knowledge. Thus, IDAS uses only one KR system — I1 — whereas systems such as COMET (unification) [McKeown *et al.*, 1990] and PENMAN (systemic) [Penman Natural Language Group, 1989] use two different KR systems: a classifier-based system for domain knowledge, and a unification or systemic system for grammatical knowledge.

Unification Systems

The most popular unification formalism for generation up to now has probably been functional unification (FUG) [Kay, 1979]. FUG systems work by searching for patterns (alternations) in the grammar that unify with the system's input (i.e., unification is used for pattern-matching); inheriting syntactic (output) feature values from the grammar patterns (the actions); and recursively processing members of the *constituent set* (the recursion). That is, pattern-action rules of the above kind are encoded as something like:

$$(\alpha_1 \wedge \beta_1) \vee (\alpha_2 \wedge \beta_2) \vee \ldots$$

If a unification system is based on a typed feature logic, then its grammar can include classification-like subsumption tests [Elhadad, 1990], and thus be as expressive in specifying patterns as a classification system.

An initial formal comparison of unification with classification is given in the Appendix. Perhaps the most important practical differences are:

- Classification grammars cannot be used bidirectionally, while unification grammars can [Sheiber, 1988].
- Unification systems produce (at least in principle) all surface forms that agree (unify) with the semantic input; classification systems produce a single surface form output.

These differences are in a sense a result of the fact that unification grammars represent general mappings between semantic and surface forms (and hence can be used bidirectionally, and produce all compatible surface forms), while classification systems generate a single surface form from a semantic input. In McDonald's [1983] terminology, classification-based generation systems deterministically and indelibly make choices about alternate surface-form constructs as the choices arise, with no backtracking;[4] unification-based systems,

[4]McDonald claims, incidentally, that indelible decision-making is more plausible than backtracking from a psycholinguistic perspective.

in contrast, produce the set of all syntactically correct surface-forms that are compatible with the semantic input.[5]

In practice, all generation systems must possess a 'preference filter' of some kind that chooses a single output surface-form from the set of possibilities. In unification approaches, choosing a particular surface form to output tends to be regarded (at least theoretically) as a separate task from generating the set of syntactically and semantically correct surface forms; in classification approaches, in contrast, the process of making choices between possible surface forms is interwoven with the main generation algorithm.

Systemic approaches

Systemic grammars [Halliday, 1985] are another popular formalism for generation systems. Systemic systems vary substantially in the input language they accept; we will here focus on the NIGEL system [Mann, 1983], since it uses the same input language (SPL) as IDAS's surface realisation system.[6] Other systemic systems (e.g., [Patten, 1988]) tend to use systemic features as their input language (i.e., they don't have an equivalent of NIGEL's chooser mechanism), which makes comparisons more difficult.

NIGEL works by traversing a network of *systems*, each with an associated *chooser*. The choosers determine *features*, by performing tests on the semantic input. Choosers can be arbitrary Lisp code, which means that NIGEL can in principle use more general 'patterns' in its rules than IDAS can; in practice it is not clear to what extent this extra expressive power is used in NIGEL, since many choosers seem to be based on subsumption tests between semantic components and the system's *upper model*. In any case, once a set of features has been chosen, these features trigger *gates* and their associated *realisation rules*; these rules assert information about the output text. From the pattern-matching perspective, choosers and gates provide the patterns α_i of rules, while realisation rules specify the actions β_i to be performed on the output text.

Like classification systems (but unlike unification systems), systemic generation systems are, in McDonald's terminology, deterministic and indelible choice-makers; NIGEL makes choices about alternative surface-form constructs as they arise during the generation process, and does not backtrack. Systemic generation systems are thus probably closer to classification systems than unification systems are; indeed, in a sense the biggest difference between systemic and classification systems is that systemic systems use a notation and inference system that was developed by the linguistic community, while classification systems use a notation and inference system that was developed by the AI community.

Other Related Work

Rösner [1986] describes a generation system that uses object-oriented techniques. SPL-like input specifications are converted into objects, and then realised by activating their To-Realise methods. Rösner does not use a declarative grammar; his grammar rules are implicitly encoded in his Lisp methods. He also does not use classification as an inference technique (his taxonomy is hand-built).

DATR [Evans and Gazdar, 1989] is a system that declaratively represents morphological rules, using a representation that in some ways is similar to I1. In particular, DATR allows default inheritance and supports role-chain-like constructs. DATR does not include a classifier, and also has no equivalent of I1's template mechanism for specifying recursion.

PSI-KLONE [Brachman and Schmolze, 1985, appendix] is an NL understanding system that makes some use of classification, in particular to map surface cases onto semantic cases. Syntactic forms are classified into an appropriate taxonomy, and by virtue of their position inherit semantic rules that state which semantic cases (e.g., **Actee**) correspond to which surface cases (e.g., **Object**).

Conclusion

In summary, classification can be used to perform much of the necessary processing in natural-language generation, including content-determination, surface-realisation, and part of text-planning. Classification-based generation allows a single knowledge representation system to be used for both domain and linguistic knowledge; this means that a classification-based generation system can have a significantly simpler overall architecture than a unification or systemic generation system, and thus be easier to build and maintain.

Acknowledgements

The IDAS project is partially funded by UK SERC grant GR/F/36750 and UK DTI grant IED 4/1/1072, and we are grateful to SERC and DTI for their support of this work. We would also like

[5] Of course these differences are in a sense more theoretical than practical, since one can design a unification system to only return a single surface form instead of a set of surface forms, and one can include backtracking-like mechanisms in a classification-based system.

[6] Strictly speaking, SPL is an input language to PENMAN, not NIGEL; we will here ignore the difference between PENMAN and NIGEL.

to thank the IDAS industrial collaborators — Inference Europe, Ltd.; Racal Instruments, Ltd.; and Racal Research, Ltd. — for all the help they have given us in performing this research.

Appendix: A Comparison of Classification and Unification

FUG is only one of a number of grammar formalisms based on feature logics. The logic underlying FUG is relatively simple, but much more expressive logics are now being implemented [Emele and Zajac, 1990; Dörre and Seiffert, 1991; Dörre and Eisele, 1991]. Here we provide an initial formal characterisation of the relation between classification and unification, but abstracting away from the differences between the different unification systems.

Crucial to all approaches in unification-based generation (or parsing) is the idea that at every level an input description (i.e. logical form or similar) γ is combined with a set of axioms (type specifications, grammar functional descriptions, rules) and the resulting logical expression is then reduced to a normal form that can be used straightforwardly to construct the set of models for the combined axioms and description.

Classification is an appropriate operation to use in normal form construction when the axioms take the form $\alpha_i \rightarrow \beta_i$, with $\rightarrow$ interpreted as logical implication, and where each α_i and β_i is a term in a feature logic. If the input description is 'complete' with respect to the conditions of these axioms (that is, if $\gamma \wedge \alpha_i \neq \bot$ exactly when $\gamma \sqsubseteq \alpha_i$, where $\sqsubseteq$ is subsumption), then it follows that for every model $\mathcal{M}$:

$$\mathcal{M} \models \{\alpha_i \supset \beta_i\} \cup \{\gamma\} \text{ iff}$$
$$\mathcal{M} \models \{\beta_i | \gamma \sqsubseteq \alpha_i\} \cup \{\gamma\}$$

(the relationship is more complex if the grammar is recursive, though the same basic principle holds). The first step of the computation of the models of γ and the axioms then just needs quick access to $\{\beta_i | \gamma \sqsubseteq \alpha_i\}$. The classification approach is to have the different α_i ordered in a subsumption taxonomy. An input description γ is placed in this taxonomy and the β_i corresponding to its ancestors are collected.

Input descriptions are 'complete' if every input description is fully specified as regards the conditions that will be tested on it. This implies a rigid distinction between 'input' and 'output' information which, in particular, means that classification will not be able to implement bidirectional grammars. If all the axioms are of the above form, input descriptions are complete and conjunctive, and the β_i's are conjunctive (as is the case in IDAS) then there will always only be a single model.

The above assumption about the form of axioms is clearly very restrictive compared to what is allowed in many modern unification formalisms. In IDAS, the notation is restricted even further by requiring the α_i and β_i to be purely conjunctive. In spite of these restrictions, the system is still in some respects more expressive than the simpler unification formalisms. In Definite Clause Grammars (DCGs) [Pereira and Warren, 1980], for instance, it is not possible to specify $\alpha_1 \rightarrow \beta_1$ and also $\alpha_2 \rightarrow \beta_2$, whilst allowing that $(\alpha_1 \wedge \alpha_2) \rightarrow (\beta_1 \wedge \beta 2)$ (unless α_1 and α_2 are related by subsumption) [Mellish, 1991].

The comparison between unification and classification is, unfortunately, made more complex when default inheritance is allowed in the classification system (as it is in IDAS). Partly, the use of defaults may be viewed formally as simply a mechanism to make it easier to specify 'complete' input descriptions. The extent to which defaults are used in an essential way in IDAS still remains to be investigated. Certainly for the grammar writer the ability to specify defaults is very valuable, and this has been widely acknowledged in grammar frameworks and implementations.

References

[Bateman *et al.*, 1990] John Bateman, Robert Kasper, Johanna Moore, and Richard Whitney. A general organization of knowledge for natural language processing: the Penman upper model. Technical report, Information Sciences Institute, Marina del Rey, CA 90292, 1990.

[Brachman and Schmolze, 1985] Ronald Brachman and James Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[Dörre and Eisele, 1991] Jochen Dörre and Anreas Eisele. A comprehensive unification formalism, 1991. Deliverable R3.1.B, DYANA - ESPRIT Basic Research Action BR3175.

[Dörre and Seiffert, 1991] Jochen Dörre and Roland Seiffert. Sorted feature terms and relational dependencies. IWBS Report 153, IBM Deutschland, 1991.

[Elhadad, 1990] Michael Elhadad. Types in functional unification grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL-1990)*, pages 157–164, 1990.

[Emele and Zajac, 1990] Martin Emele and Rémi Zajac. Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-1990)*, volume 3, pages 293–298, 1990.

[Evans and Gazdar, 1989] Roger Evans and Gerald Gazdar. Inference in DATR. In *Proceedings of Fourth Meeting of the European Chapter of the Association for Computational Linguistics (EACL-1989)*, pages 66–71, 1989.

[Grosz *et al.*, 1986] Barbara Grosz, Karen Sparck Jones, and Bonnie Webber, editors. *Readings in Natural Language Processing.* Morgan Kaufmann, Los Altos, California, 1986.

[Halliday, 1985] M. A. K. Halliday. *An Introduction to Functional Grammar.* Edward Arnold, London, 1985.

[Kasper, 1989] Robert Kasper. A flexible interface for linking applications to Penman's sentence generator. In *Proceedings of the 1989 DARPA Speech and Natural Language Workshop*, pages 153–158, Philadelphia, 1989.

[Kay, 1979] Martin Kay. Functional grammar. In *Proceedings of the Fifth Meeting of the Berkeley Linguistics Society*, pages 142–158, Berkeley, CA, 17-19 Febuary 1979.

[Mann, 1983] William Mann. An overview of the NIGEL text generation grammar. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL-1983)*, pages 79–84, 1983.

[Mann and Moore, 1981] William Mann and James Moore. Computer generation of multiparagraph English text. *American Journal of Computational Linguistics*, 7:17–29, 1981.

[McDonald, 1983] David McDonald. Description directed control. *Computers and Mathematics*, 9:111–130, 1983.

[McKeown *et al.*, 1990] Kathleen McKeown, Michael Elhadad, Yumiko Fukumoto, Jong Lim, Christine Lombardi, Jacques Robin, and Frank Smadja. Natural language generation in COMET. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 103–139. Academic Press, London, 1990.

[Mellish, 1991] Chris Mellish. Approaches to realisation in natural language generation. In E. Klein and F. Veltman, editors, *Natural Language and Speech.* Springer-Verlag, 1991.

[Moore and Paris, 1989] Johanna Moore and Cecile Paris. Planning text for advisory dialogues. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL-1989)*, pages 203–211, 1989.

[Patten, 1988] Terry Patten. *Systemic Text Generation as Problem Solving.* Cambridge University Press, 1988.

[Penman Natural Language Group, 1989] Penman Natural Language Group. The Penman user guide. Technical report, Information Sciences Institute, Marina del Rey, CA 90292, 1989.

[Pereira and Warren, 1980] Fernando Pereira and David Warren. Definite clause grammars for language analysis. *Artificial Intelligence*, 13:231–278, 1980.

[Reiter, 1990] Ehud Reiter. Generating descriptions that exploit a user's domain knowledge. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*, pages 257–285. Academic Press, London, 1990.

[Reiter, 1991] Ehud Reiter. A new model of lexical choice for nouns. *Computational Intelligence*, 7(4), 1991.

[Reiter and Dale, 1992] Ehud Reiter and Robert Dale. A fast algorithm for the generation of referring expressions. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING-1992)*, 1992.

[Reiter *et al.*, 1992] Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of on-line documentation in the IDAS project. In *Proceedings of the Third Conference on Applied Natural Language Processing (ANLP-1992)*, pages 64–71, 1992.

[Rosch, 1978] Eleanor Rosch. Principles of categorization. In E. Rosch and B. Lloyd, editors, *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum, Hillsdale, NJ, 1978.

[Rösner, 1986] Dietmar Rösner. *Ein System zur Generierung von deutschen Texten aus semantischen Repräsentationen.* PhD thesis, Institut für Informatik, University of Stuttgart, 1986.

[Sheiber, 1988] Stuart Sheiber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*, pages 614–619, 1988.

[Touretzky, 1986] David Touretzky. *The Mathematics of Inheritance Systems.* Morgan Kaufmann, Los Altos, California, 1986.

[Wahlster *et al.*, 1991] Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, and Thomas Rist. WIP: The coordinated generation of multimodal presentations from a common representation. In Oliverio Stock, John Slack, and Andrew Ortony, editors, *Computational Theories of Communication and their Applications.* Springer-Verlag, 1991.