From Use Cases to UML Class Diagrams using Logic Grammars and Constraints

Henning Christiansen
Roskilde University
P.O. Box 260, DK-4000 Roskilde, Denmark
henning@ruc.dk

Christian Theil Have
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 København S, Denmark
cth@itu.dk

Knut Tveitane
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 København S, Denmark
knut@itu.dk

Abstract

We investigate the possibilities for automated transition from Use Cases in a restricted natural language syntax into UML class diagrams, by trying to capture the semantics of the natural language and map it into building blocks of the object oriented programming paradigm (classes, objects, methods, properties etc.). Syntax and semantic analysis is done in a framework of Definite Clause Grammars extended with Constraint Handling Rules, which generalizes previous approaches with a direct way to express domain knowledge utilized in the interpretation process as well as stating explicit rules for pronoun resolution. The latter involves an improvement of earlier work on assumptions with time stamps.

Keywords

Application of NLP; Logic grammars and constraints; Domain specific analysis; Abduction and assumptions.

1 Introduction

Use cases are widely used to map requirements during inception and elaboration of a software development project. Mapping requirements is an important but difficult task, that can be impaired by lack of understanding and communication difficulties. According to [33], known as the Chaos Report, imprecise and incomplete requirements is a prevalent cause of software project failures. Often it seems that the stake holders do not speak the same language. The engineer speaks in terms of design models while the domain expert defines the problem in domain specific, and often ambiguous, language within his frame of reference.

We suggest to bridge this gap by introducing an automatic and interactive system which translates a restricted, but naturally appearing, use case language into class diagrams in the UML notation [29].

The system maintains an up-to-date diagrammatical presentation of the current use case text in a window on the user's screen, cf. Fig. 1. This is intended to encourage an iterative mode of working, so as soon as the user adds a new or modifies an existing sentence,

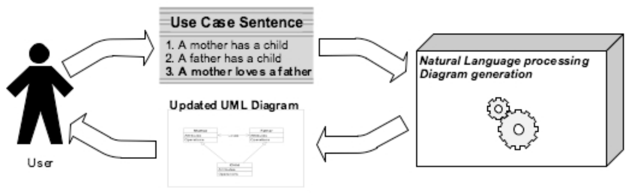

Fig. 1: *The system is used iteratively and interactively - each new sentence causes the UML diagram to be updated.*

the consequences in terms of the object model is displayed immediately. This can aid the user in the process of understanding the current domain, including identifying possible misconceptions at an early stage. Possible applications include requirement engineering, brainstorming, prototyping and as a tool for teaching UML. The current version is limited to generation of class diagrams but points forward to the goal, which is to include also the dynamic aspects of use cases with generation of process diagrams, etc.

In this work, we reconcile and promote different methods for discourse analysis founded on logic programming technology, more specifically the familiar Definite Clause Grammars [30] and Constraint Handling Rules [17] (CHR). CHR adds a global resource, in the shape of a constraint store, and makes a sort of production-like rules available for controlling and utilizing this store, while maintaining a declarative framework. We indicate an improvement of earlier work concerned with so-called assumptions by adding time stamps, which make it possible to state rules for anaphora resolution in an explicit and logical form.

2 Related Work

Similar Systems Several authors have attempted to automate translation from specifications in natural language to code or diagrams, using either essentially formal language with a "natural" appearance or opportunistic parsing. We have not seen this related directly to Use Cases and UML, which is our approach.

Attempto Controlled English [18] is a system that translates specification texts in a formal language of declarative sentences into first order logic discourse representation structures and optionally into Prolog. Sowa has defined a similar, but simpler, specification language called "Common Logic Controlled English" [31] which translates directly into first order logic (and vice versa).

The Metafor system [24] use opportunistic techniques and the semantically enriched lexicon *ConceptNet* [25] to derive and discover relations between classes and translate English sentences into code in Lisp or Python. Its input language supports a variety of narrative stances, past and present tense and anaphorical and indirect references. The authors [26] have coined the term *"programmatic semantics"* to describe the transliteration process: "Programmatic semantics is a mapping between natural linguistic structures and basic programming language structures" [26]. We have adopted this terminology.

Examples of other approaches using NLP for requirement analysis are described by [15, 16, 19, 23].

Use Cases and Unified Modelling Language Abbott [1] introduced a (manual) methodology for object-oriented program design that derives candidate classes, objects and operators from the syntactical elements of English sentences. Abbotts method, became an integral part of Booch's "object-oriented design" [5] process, where an informal problem description is formalized through definition of objects and their attributes and operations. Jacobsen introduced the concept of Use Cases [21], which resembled Booch's problem descriptions. Use Cases describe a users view of the system which is useful for "gaining an understanding of the problem" and "identifying candidate objects" [4]. Rumbaugh et. all [27] described the OMT notation including the "Class Association Diagram", the precursor of the UML class diagram. Booch, Jacobson and Rumbaugh later defined the first draft for a "Unified Modeling Language" [7].

Use cases model the actors of a system and the flow of events between them. They describe *what* a system does without specifying how [6]. Even though use cases are written in natural language, only a subset of English is normally used. The UML User Guide [6] provides examples but no clear guidelines.

Cockburn [12] has suggested a semi-formal approach where each action description has a certain structured format. The CREWS guidelines for use case writing in [3] provides insight wrt. to the linguistic structures of use cases. The CP guidelines [13] was proposed as simpler set guidelines with similar expressiveness. The language suggested by the guidelines includes present tense *subject-verb-object* like sentences with no adverbs or adjectives. In essence the guidelines advocate avoiding all ambiguous language constructs. Our grammar is inspired by the CP guidelines, but allows more advanced pronoun usage.

NLP using Logic Grammars As shown in previous work [2, 8, 9, 10], CHR provides a straightforward implementation of abduction, and here we follow the principle of discourse analysis as abduction, introduced by [20] and now widely accepted: the meaning of a discourse is taken to be the set of "hidden" facts over which the discourse is faithfully created.

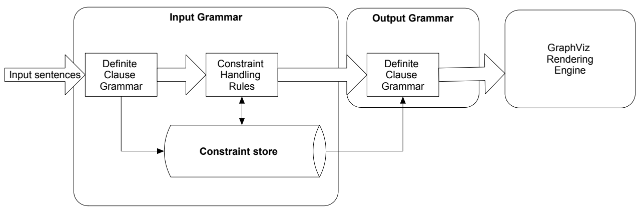

Fig. 2: *Architecture Overview. The thick lines illustrate program flow and the thin lines illustrate the use of the constraint store.*

Our work follows the tradition of logic programming based grammars, but extends previous work in different ways. Assumption Grammars [14] (AG) provide mechanisms that may cope with pronouns. Inspired by the work of [9, 10] that realizes AGs in CHR, we have extended with time stamps to make it possible to specify detailed scope and preference rules in CHR, which otherwise is a shortcoming of AGs. Creation of data and knowledge bases from text using logic grammars have been pursued by a variety of authors, e.g., [32, 18]. The model of "Meaning in Context" formalized by [11], which is based on CHR shows how domain knowledge utilized in the interpretation process can be expressed directly in CHR.

3 System Overview

3.1 Architecture

The system is implemented in a combination of Definite Clause Grammars (DCG) and CHR, with sentence meaning added gradually to the constraint store. This converted into the GraphViz language using a second DCG, and rendered as a UML class diagram using a GraphViz engine, and displayed to the user; see Fig. 2.Incrementality is simulated by parsing the entire text and drawing new diagram when a period which is added or changed. Only a rudimentary user interface exists at the moment, but the current prototype qualifies as proof of concept for our ideas.

3.2 Supported Language Constructs

The subset of natural English supported by our system needs to be sufficiently expressive as to cover the important entities and relations in an object oriented system description. Fig. 3 below shows the diagram for the example sentences.

3.2.1 Basic Sentences

The basic sentence consist of a noun phrase followed by a simple verb phrase that contains an intransitive or transitive verb. We consider first verbs that imply an action to be performed by or on the subject of the sentence. The subject maps to a *class definition* in the object oriented programming paradigm. The verb maps to a *method* of the class represented by the subject. For a transitive verb, the object defines another class that serves as *argument* to the method. Example: *"The professor teaches. A student reads, writes projects and takes exams."*

3.2.2 Property Sentences

Property sentences imply an ownership or containment relation, and are similar to the transitive sentences above, but use specific verbs such as "have". The object may be plural and quantified. The quantification may a numeral or a linguistic quantifier such as "some", "most" or "every". These sentences associate properties expressed by their object with the class(es) indicated by their subject. There are different approaches for representing these in object oriented programming languages. We have tried not to limit the flexibility, by maintaining as precise information as possible about the cardinality of the property. When exact number is given, this is preserved and a quantifier such as "some" is mapped into an undetermined cardinality denoted as "`n`". When alternatives are given for the same property, the different cardinalities are aggregated into an interval; the details are spelled out in section 4.2 below. Example: *"A professor has an office. The university has five study lines."*

3.2.3 Inheritance and Instantiation Sentences

Sentences formed with the verb "to be" relate single objects or classes to other classes. In *"A student is a (kind of) person"*, "person" is a *superclass* of "student", and "student" a subclass of "person". Multiple inheritance, where the subject is "a kind of" more than one class is also possible. In "John is a student", the subject is a proper noun indicating a named entity (John) of the class "student". In object oriented terminology, John is an *object* of class student. After an individual has been introduced with a designated class membership, it can be used as a *prototype* member of that class C in sentence forms above. Consider a sentence such as "John reads". It looks straightforward at first glance, but its programmatic semantics is bit more complex. "John" maps to an object, "reads" to a method, but objects don't define methods (or properties). The sentence must be understood to mean that the method belongs to the *class* the object is an instance of. Prototypes have also natural usages as arguments in basic sentences that introduces method In a suitable context, "Mary interviews Peter" carries the same programmatic semantics as "Teachers interview students". Example: *'Students are a kind of persons and professors are a kind of persons."*

3.2.4 Adjectives

The implied programmatic semantics of adjectives limits or specializes the meaning of a noun which may be reflected in the class diagrams in different ways. The term *"large car"* may imply a subclass of car, a boolean property "large", or a property which may take different values, one of which is "large". A nontrivial semantic analysis needs to be made, and we currently provide no solution.

3.2.5 Pronouns

We require that pronouns can be resolved in a unique way which is intelligible to the user; at the same time we should also, for the acceptance of the tool, allow a variety of natural patterns. As is well-known, pronoun (and anaphora) resolution is one of the most difficult tasks in computational linguistics, cf. [28], and we have decided to use a simple heuristic, and reject any sentence for which it does not apply. Resolving, say, "he" considers the most recent occurrence of a male object which is found in a previous sentence S; however, if S contains two candidates, the pronoun application is claimed ambiguous. This excludes usages such as "Peter and Paul ... . He ..." and "Women are persons. They have two legs." Example: *"The professor writes papers and he supervises students."*

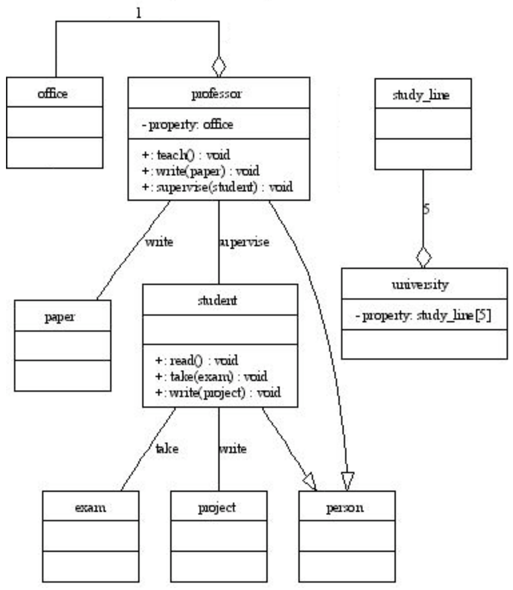

Fig. 3: *Results from the collected sentences of sec. 3*

4 NLP Methods Applied

In the following, we assume a basic knowledge of DCG, and explain the CHR specifics that are used.

4.1 Overall Implementation Principles

We consider sentences that describe class hierarchy, e.g., "A dog is an animal". The following grammar rule gives the overall structure.

```
sentence -->
    fc_noun_phrase(Number, _, subj, IdSub),
    subord_verb(Number,_),
    fc_noun_phrase(_, Number, obj, IdObj),
    {extends(IdSub,IdObj)}.
```

Notice that it produces no explicit output via arguments, but updates the constraint store by the calls to constraints, i.e., extends by abductive reasoning the constraint store with those facts that seem to be the reason, why the sentence can be stated. In this example, the rule depends on `extends(dog,animal)` and, since `extends` is declared as a constraints, it will be added to the constraint store if it was not there already. The grammar rules for noun phrases may, in a similar way, create the facts `class(dog)` and `class(animal)`. Thus the analysis of this sample sentence produces a store of three constraints that can be converted in a straightforward way into the input language of Graphviz, which in turn produces a class diagram showing that the class of dogs is a subclass of animals.

The second attribute of noun phrases is called CollectiveNumber; for “a cat and a dog” it evaluates to `sing` and for “cats and dogs” to `plur`. Taking the collective number for the subject as the number for the object allows “A dog is an animal and a pet” and excludes “A dog is animals and pets”. Noun phrases are divided into different categories with particular restrictions; for example, `fc_noun_phrase` (for fully specified class) used above is a category which forbids pronouns and quantified expressions like “two cats” in this particular sort of sentences. Similar categories are defined for `indiv_noun_phrase` referring to particular objects (“She, Peter, and Paul”), `rc_noun_phrase` for restricted class (“Mary and the boys” assuming that Mary is a prototype for some class), and `q_noun_phrase` for quantified expressions such as “four legs and a tail”.

Noun phrases generate a representation of their contents, using a plus to combine conjunctive phrases. Here are some examples, assuming that Mary is a prototype woman and that “she” refers to Mary.

`fc_`	cats and dogs	`cat+dog`
`indiv_`	her, Peter, and Paul	`mary+peter+paul`
`rc_`	Mary and the boys	`woman+boy`
`q_`	a tail and some legs	`tail:1+legs:n`

The following CHR rules unroll constraints with composite arguments into individual constraints.

```
extends(A+B,C) <=> extends(A,C), extends(B,C).
extends(A,B+C) <=> extends(A,B), extends(A,C).
```

These are so-called simplification rules, triggered each time an `extends` constraint with a plus in one of its arguments appear in the store. They delete the constraint(s) matched by the left-hand side, the head, and add those on the right, the body.

4.2 Expressing Knowledge About Use Case Modeling

CHR can be used to express knowledge about the domain in question. We can illustrate this by the way we aggregate the constraints emerging from different statements about the same property. We use `property(car,wheels:4)` to express that a car has four wheels and `property(car,doors:(2..5))` to say that it has between 2 and 5 doors. Consider the following CHR rule which is part of the implementation.

```
property(C,P:N), property(C,P:M) <=>
   q_count(N), q_count(M), q_less_eq(N,M)
   | property(C,P:(N..M)).
```

It applies when the store contains two `property` constraints for the same class and property, provided the guard is true. The guard is an optional part between the arrow and the vertical bar which here refers to Prolog predicates written specifically for the purpose, so that, say, `q_count(5)`, `q_count(n)`, and `q_less_eq(2,n)` are true. So “Peter has a dog. Paul has five dogs” yields `property(man,dog:1)` and `property(man,dog:5)` which by the rule above get replaced by `property(man,dog:(1..5))`. Another rule (not shown) combines different intervals for multiplicity into one.

4.3 Pronoun Resolution

Here we sketch briefly the approach inspired by the assumption principle of [14] but extended with a time stamp (here, sentence number) to realize the indicated principle. When, say, “Peter” is mentioned in sentence no. 7, a constraint `referent(sing,masc,peter,7)` is emitted, and an occurrence of “him” in sentence no. 10 gives rise to `expect_referent(sing, masc,X)`; the following rule attempts to bind `X` to the suitable value.

```
sentence_no(Now), referent(No,G,Id,T) \
                        expect_referent(No,G,X) <=>
   T < Now,
   \+ ( find_constraint( referent(No,G,_,TMoreRecent),_),
        T < TMoreRecent, TMoreRecent\==Now)
    | ( find_constraint( referent(No,G,Id1,T), _),
        Id1\=Id -> X = error:pronoun:ambiguous(No,G,Now)
          ; X=Id ).
```

The rule is a so-called simpagation rule which, when applied, keeps the constraints before “\” in the store and removes the remaining ones up until the arrow. CHR does not allow negations in the head, so the test that time `T` designates the most recent `referent` (i.e., there is no other such with a more recent time stamp) is done in less elegant way in the guard. The body tests for ambiguity and may generate an error code. Finally, the following rule catches unresolved pronouns if, e.g., the whole text start with “He”.

```
sentence_no(Now) \ expect_referent(No,G,X) <=>
    X=error:pronoun:unresolved(No,G,Now).
```

The following grammar rule for using pronouns shows how the implemented `expect_referent` constraint can be used.

```
indiv_simple_noun_phrase(Num,Case,G,Id) -->
   pronoun(Num,Case,G),
   {expect_referent(Num,G,Id)}.
```

This example illustrates how relatively complicated contextual dependencies in logic grammars can be modeled in a fairly concise way using CHR. The use of prototypical individuals for classes is realized in a similar way. In “Mary is the boss. She manages the employees.”, the pronoun is resolved to `mary`, and a call to a constraint `expect_class(···mary···)` locates the class `boss` (i.e., if it is unique, otherwise an error code) and the constraint `method(boss,manage,employees)` is created.

4.4 From Constraint Stores to Diagrams

Another DCG is defined for the GraphViz input language. This grammar references the constraint store and generates a phrase as long as possible, thereby converting constraints into phrases to be given as input to GraphViz. This is straightforward and not described further. If, for example, the constraint store contains `class(man)` and `method(man,walk)`, the nonterminal `class_node` generates the phrase `man[ label = "{man||: walk(): void\l}"]`.

5 Conclusions and Future Work

We presented a system for analyzing a restricted natural language for use case writing, based on Definite Clause Grammars extended with Constraint Handling Rules. Our grammar captures candidate domain classes and their relations and visualize these using an UML class diagram. The syntax of the language is simple, but expressive enough to model a given domain. The language seems natural and expressive but avoids inherently ambiguous sentence elements such as adverbs and adjectives. By extending our grammar with Constraint Handling Rules, we are able to handle pronoun resolution with ambiguity detection, prototypical references (e.g. names) and allow the user to express knowledge about the domain, such as multiplicity, using simple prototypical sentences.

Our grammar captures information about the static world. This is precisely what is reflected in the UML class diagram. However, Use Cases normally also contain sentences about event flows as such *"If the light is red then the cars stop"* and *"They wait until the light is green"*. These sentences contain information of a dynamic nature that would typically be depicted in state or flow charts, or in UML, sequence diagrams. It would be a natural next step to extend the grammar to support such sentences. As we have indicated, CHR rules which include more "expert knowledge" about use case modeling can be added to the analysis, and this potential should be investigated further. However, this must be done with care: adding more intelligence to the system may help the user to realize properties of the world he is describing, but it may also destroy the transparency and incrementality exposed in the current prototype.

Acknowledgement: This work is supported by the CONTROL project, funded by Danish Natural Science Research Council.

References

[1] R. J. Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[2] S. Abdennadher and H. Christiansen. An experimental CLP platform for integrity constraints and abduction. In *Proceedings of FQAS2000, Flexible Query Answering Systems: Advances in Soft Computing series*, pages 141–152. Physica-Verlag (Springer), 2000.

[3] C. B. Achour. Guiding scenario authoring. In *EJC*, pages 152–171, 1998.

[4] E. V. Berard. *Be Careful With 'Use Cases'*. The Object Agency, Inc., August 1998.

[5] G. Booch. Object-oriented development. *IEEE Transactions on Software Engineering*, 12(2):211–220, Feb. 1986.

[6] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[7] G. Booch and J. Rumbaugh. *Unified Method for Object-Oriented Development Version 1.0*. Rational Software Corporation, 1997.

[8] H. Christiansen. A constraint-based bottom-up counterpart to definite clause grammars. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *RANLP*, volume 260 of *Current Issues in Linguistic Theory (CILT)*, pages 227–236. John Benjamins, Amsterdam/Philadelphia, 2004.

[9] H. Christiansen. CHR Grammars. *Int'l Journal on Theory and Practice of Logic Programming*, 5(4-5):467–501, 2005.

[10] H. Christiansen and V. Dahl. HYPROLOG: A new logic programming language with assumptions and abduction. In M. Gabbrielli and G. Gupta, editors, *ICLP*, volume 3668 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2005.

[11] H. Christiansen and V. Dahl. Meaning in Context. In A. Dey, B. Kokinov, D. Leake, and R. Turner, editors, *Proceedings of Fifth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-05)*, volume 3554 of *Lecture Notes in Artificial Intelligence*, pages 97–111, 2005.

[12] A. Cockburn. Structuring use cases with goals. *Journal of Object-Oriented Programming*, Sept.Oct. 1997.

[13] K. Cox and K. Phalp. Replicating the CREWS use case authoring guidelines experiment. *Empirical Software Engineering*, 5(3):245–267, 2000.

[14] V. Dahl, P. Tarau, and R. Li. Assumption grammars for processing natural language. In *ICLP*, pages 256–270, 1997.

[15] J. Drazan and V. Mencl. Improved processing of textual use cases: Deriving behavior specifications. In *Proceedings of SOFSEM 2007*, volume 4362 of *Lecture Notes in Computer Science*. Springer, 2007.

[16] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of linguistic techniques for use case analysis. In *RE*, pages 157–164. IEEE Computer Society, 2002.

[17] T. Frühwirth. Theory and practice of constraint handling rules, special issue on constraint logic programming. *Journal of Logic Programming*, 37(1–3):95–138, Oct. 1998.

[18] N. E. Fuchs. Attempto controlled english. In *WLP*, pages 211–218, 2000.

[19] H. M. Harmain and R. J. Gaizauskas. Cm-builder: A natural language-based case tool for object-oriented analysis. *Autom. Softw. Eng.*, 10(2):157–181, 2003.

[20] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. A. Martin. Interpretation as abduction. *Artif. Intell.*, 63(1-2):69–142, 1993.

[21] I. Jacobson. Object oriented development in an industrial environment. In *OOPSLA*, pages 183–191, 1987.

[22] K. P. Karl Cox and M. Shepperd. Comparing use case writing guidelines. In *Seventh International Workshop on Requirements Engineering (RE'01)*, June 2001.

[23] N. Kiyavitskaya, N. Zeni, L. Mich, and J. Mylopoulos. Experimenting with linguistic tools for conceptual modelling: Quality of the models and critical features. In F. Meziane and E. Métais, editors, *NLDB*, volume 3136 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2004.

[24] H. Liu and H. Lieberman. Toward a programmatic semantics of natural language. In *VL/HCC*, pages 281–282. IEEE Computer Society, 2004.

[25] H. Liu and P. Singh. Conceptnet: A practical commonsense reasoning toolkit, May 02 2004.

[26] Liu, Hugo and Lieberman, Henry. Programmatic semantics for natural language interfaces. In *Proceedings of ACM CHI 2005 Conference on Human Factors in Computing Systems*, volume 2 of *Late breaking results: short papers*, pages 1597–1600, 2005.

[27] M. E. S. Loomis, A. V. Shah, and J. E. Rumbaugh. An object modeling technique for conceptual design. In J. Bézivin, J.-M. Hullot, P. Cointe, and H. Lieberman, editors, *ECOOP '87, European Conference on Object-Oriented Programming*, volume 276 of *Lecture Notes in Computer Science*, pages 192–202. Springer-Verlag, 1987.

[28] R. Mitkov. *Anaphora Resolution*. Longman (Pearson Education), 2002.

[29] Object Management Group. *Unified Modeling Language (UML), version 2.0*. Object Management Group, Framingham, Massachusetts, Oct. 2004.

[30] F. C. N. Pereira and D. H. D. Warren. Definite clause grammars for language analysis—A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278, 1980.

[31] J. F. Sowa. *Common Logic Controlled English*, 2004. Draft, http://www.jfsowa.com/clce/specs.htm.

[32] P. Tarau, K. D. Bosschere, V. Dahl, and S. Rochefort. Logimoo: An extensible multi-user virtual world with natural language control. *J. Log. Program.*, 38(3):331–353, 1999.

[33] The Standish Group. The CHAOS report, 1994.