A practical application of computational humour

Graeme Ritchie
Computing Science
University of Aberdeen
Aberdeen AB24 3UE
gritchie@csd.abdn.ac.uk

Ruli Manurung[*] **Helen Pain**
School of Informatics
University of Edinburgh
Edinburgh EH9 8LW
ruli.manurung@ed.ac.uk
H.Pain@ed.ac.uk

Annalu Waller, Rolf Black, Dave O'Mara
School of Computing
University of Dundee
Dundee DD1 4HN
awaller@computing.dundee.ac.uk
rolfblack@computing.dundee.ac.uk
domara@computing.dundee.ac.uk

Abstract

The past 15 years has seen the development of a number of programs which perform tasks in the area of humour, but these have been exploratory research prototypes, usually on a very small scale, and none of them interacted with users. Amongst those which actually created humorous texts, the JAPE program was probably the most substantial, but even it was far from being useful for any practical purpose. We have developed a fully engineered riddle generator, inspired by the ideas in the JAPE system, which uses a large-scale multimedia lexicon and a set of symbolic rules to generate jokes. It has an interactive user interface, specially designed for children with complex communication needs (CCN), so that users can make choices to guide the riddle generator. The software is robust, stable, and responds sufficiently promptly that naive users can interact without difficulty. It has been tested over with real users (children with CCN), with highly positive results, and is publicly available for free download.

Keywords: Computational humour, riddles, AAC, joke generation

1 Introduction

Since 1992, research into computational humour has led to a number of exploratory implementations, including some joke-generation systems. However, these have generally been small exploratory research prototypes rather than full practical applications. We have developed a state of the art riddle-generation system which is completely usable by untrained and naive users, and which has been evaluated in a systematic manner. Our program (STANDUP - System To Augment Non-speakers Dialogue Using Puns) is aimed at young children, and lets them play with words and phrases by building punning riddles through a simple interactive user-interface.

A punning riddle is a question-answer joke in which the answer makes a play on words, as in (1).

(1) What kind of tree is nauseated?
A sick-amore.

What makes these jokes computationally manageable (and also makes them good illustrative examples of simple language mechanisms) is their reliance on simple linguistic relations such as homophony and synonymy.

Our target users are children with impaired speech and limited motor skills (as often results from cerebral palsy). Such *complex communication needs* (CCN) can result in lower levels of literacy than in typically-developing counterparts of the same age (Smith, 2005). This can prevent full involvement in normal forms of language play (Waller, 2006), leading to poorer skills in language, communication and social interaction (Lindsay and Dockrell, 2000). The STANDUP software is a "language playground", with which a child can explore sounds and meanings by making up jokes, with computer assistance. We conjecture that this will have a beneficial effect on literacy and communication skills, but our project addressed two more basic research questions:

(i) is it feasible to build an interactive riddle-generator which can be controlled by children with CCN?

(ii) if so, in what ways do such children use the software?

We have adopted the joke-construction mechanisms of the JAPE program (Binsted, 1996; Binsted and Ritchie, 1994, 1997) as the core of our stable, usable, robust, user-friendly, interactive system for children with CCN. In this paper, we will describe what was involved in developing a working application from the research ideas.

2 Computational Humour

At present, there is no theory of humour which is sufficiently precise, detailed and formal to be implementable. Hence computational humour has so far largely consisted of small research prototypes based on mechanisms designed specifically for whatever (narrow) problem was being tackled (for overviews, see Ritchie (2001b), Hulstijn and Nijholt (1996), Stock et al. (2002)). Many of these

[*] Now at Faculty of Computer Science, Universitas Indonesia, Depok 16424, Indonesia. maruli@cs.ui.ac.id

programs have been generators of simple verbal jokes, and have been very small studies (often student projects).

Lessard and Levison (1992) built a program which created a simple type of pun, the *Tom Swifty*. Lessard and Levison (1993) sketch the workings of a program which produced some basic forms of punning riddle. Venour (1999) built a small program which generated simple texts consisting of a one-sentence set-up and a punning punchline consisting of a head noun preceded by an adjective or a noun modifier. All these used an existing natural language generator, VINCI (Levison and Lessard, 1992). The WISCRAIC program (McKay, 2002) produced simple puns in three different linguistic forms (question-answer, single sentence, two-sentence sequence).

These systems operated with small amounts of hand-crafted data, and were not given much serious testing. The Lessard and Levison projects report no performance or evaluation, while Venour and Mackay report very small and not very systematic evaluations. (See Ritchie (2004, Chap. 10) for a fuller review of these systems.)

As our program is a punning riddle generator, JAPE (Section 3 below) and the systems reviewed above are its antecedents. Other work in computational humour has included a program which could construct amusing acronyms (Stock and Strapparava, 2003, 2005), a recogniser for basic "knock-knock" jokes (Taylor and Mazlack, 2004), a study of how machine-learning techniques could separate joke texts from non-jokes (Mihalcea and Strapparava, 2006), a very preliminary generator for insults based on 'scalar humour' (Binsted et al., 2003), and a program which, for a particular class of jokes, selects a punchline for a joke set-up (Stark et al., 2005). Although some of these were on a slightly larger scale than the pun generators described above, all of them were research prototypes, with no claims to be usable applications.

3 The JAPE riddle generator

The JAPE program (Binsted and Ritchie, 1994, 1997; Binsted, 1996) generated certain classes of punning riddles. Some of the better examples were the following:

(2) How is a nice girl like a sugary bird?
Each is a sweet chick.

(3) What is the difference between leaves and a car? One you brush and rake, the other you rush and brake

(4) What is the difference between a pretty glove and a silent cat? One is a cute mitten, the other is a mute kitten.

(5) What do you call a strange market? A bizarre bazaar.

JAPE used three types of symbolic rules (*schemas*, *description rules*, *templates*) to characterise the possible linguistic structures (Ritchie, 2003). Our variants of these mechanisms are described in detail below (Section 4).

JAPE stands out from the other early pun-generators in two respects: it used a large, general-purpose lexicon, WordNet (Miller et al., 1990; Fellbaum, 1998), rather than a small hand-crafted one, and a properly controlled evaluation of the output was carried out (Binsted et al., 1997). The latter study showed that JAPE-generated jokes were reliably distinguished from non-jokes, human-generated jokes were more often deemed to be jokes than JAPE-generated jokes, JAPE-generated jokes were funnier than non-jokes, and human-generated jokes were funnier than JAPE-generated jokes. Also, a JAPE output ((3) above) was rated the funniest in the data set.

As well as developing the mechanisms for punning riddle generation, Binsted suggested, in passing, the idea of using such a program for interactive language teaching. However, the JAPE implementation was still just a research prototype, and there were certain aspects which would have to be altered or rebuilt if it was to be used for practical purposes. These limitations were roughly in the areas of *usability* and *output quality*. To be more precise:

(i) The only behaviour of the program was to create riddles, one after another, with very few parameters available for variation. Also, these parameters were internal to the mechanism (e.g. the choice of schema) and might not make sense to an ordinary user.

(ii) The program worked by exhaustively searching for words and phrases which would match its schemas and templates. There was no way to guide the software (e.g. to make a joke on a particular topic).

(iii) There was no real user interface – the user (always a knowledgeable researcher) would invoke the program from a simple command interface.

(iv) The search for suitable words, being unintelligent and exhaustive, could (with a large lexicon) be very slow; Binsted's test runs took hours. Hence, the response time was useless for interaction with a user.

(v) The jokes were of very variable quality, with the proportion of intelligible jokes being quite small; the proportion of *good* intelligible jokes was very small.

(vi) Facilities for comparing words for similarity of sound were quite primitive. In particular, there was no provision for approximate matches (near-homophony) and correspondences between written and phonetic forms of words were slightly ad hoc.

Of these, (iii) and (iv) had to be remedied for our application, and (i) and (ii) were serious drawbacks. The more we could do about (v), the better, and addressing (vi) would contribute to this.

4 The joke generator

The STANDUP generator consists of three stages, as in JAPE; all are implemented in Java. Each stage consists of instantiating a particular kind of rule: *schemas* (Section 4.1), *description rules* (Section 4.2) – both supported by a large dictionary– and *templates* (Section 4.3).

4.1 Schemas

A schema consists of 5 parts:

Header: This attaches a symbolic name to the schema, and lists its parameters.

Lexical preconditions: This is a collection of constraints specifying the core items needed for a particular subclass of riddle. Items can be either *lexemes* (lexical entries) or *word forms* (the orthographic textual representation of a word). Constraints can involve syntactic categorisation (e.g. a lexeme is a noun), phonetic relations (e.g. two items rhyme), structural relations (e.g. an item X is a compound noun made up of components Y and Z), and semantic relations (e.g. one lexeme is a hypernym of another).

Question specification: This specifies how certain variables in the schema are, once instantiated, to be described within the question part of the eventual riddle. This, and the answer specification, supply the input to the description rules (Section 4.2 below).

Answer specification: This is like the **Question specification**, but contributes to the answer in the riddle.

Keywords: This lists the subset of the schema's variables which will be bound to lexemes. It is used to define a notion of *equivalence* between similar riddles: two riddles are deemed equivalent if they use the same schema with the same instantiation of the keyword variables. The generator tracks which instantiations have been used so far with a particular user, and does not offer 'equivalent' jokes again to the same user.

Informally, the schema's variables can be instantiated with values from the lexicon, providing they meet the constraints in the lexical preconditions.

There are 11 schemas (for 11 underlying kinds of joke). A typical schema is given in Figure 1. Following the practice of the JAPE system, relations and properties are expressed here in Prolog-style (logic-like) notation, with predicates applied to arguments. Although each schema was designed using this notation, in the actual implementation the lexical preconditions were compiled into an expression in the database query language SQL, to facilitate the finding of suitable variable values within the lexical database (implemented using the PostgreSQL software package[1]). This compilation was done by hand, although in principle the process could be fully automated.

```
Header: newelan2(NP, A, B, HomB)
Lexical preconditions:
  nouncompound(NP,A,B),
  homophone(B,HomB), noun(HomB)
Question specification:
  {shareproperties(NP, HomB)}
Answer specification: {phrase(A,HomB)}
Keywords: [NP, HomB]
```

Figure 1: A typical STANDUP schema

The `newelan2` schema given above could have an instantiation in which `NP`= *computer screen*, `A` = *computer*, `B` = *screen*, and `HomB` = *scream* (the relation `homophone` means that the two items lie within the current threshold for phonetic similarity; i.e. "homophones" can be approximate). This could give rise (after two further phases of processing — Sections 4.2, 4.3 below) to a riddle such as (6):

(6) What do you call a shout with a window?
A computer scream.

The question specification and the answer specification show how the instantiating values have to be passed on to the next phase (Section 4.2 below), by embedding the relevant variables within symbolic expressions which act as signals about what is to be done with these values. In this example, the question specification would be `shareproperties(computer screen, scream)` and the answer specification would be `phrase(computer, scream)`.

4.2 Constructing descriptions

The middle phase of joke generation – constructing descriptions – was not in JAPE-1 (Binsted and Ritchie, 1994, 1997), but was introduced in JAPE-2 (Binsted, 1996; Binsted et al., 1997). It encodes possible linguistic variations, given core values from the schema instantiation.

The question specification and answer specification are handled separately. Each is matched non-deterministically against a set of description rules. These rules have a structure roughly similar to schemas, in that they have a *header*, some *preconditions*, and an output expression, the *template specifier* (Figure 2).

```
Header: shareproperties(X,Y)
Preconditions:
  meronym(X, MerX), synonym(Y, SynY)
Template specifier: [merHyp, MerX, SynY]
```

Figure 2: A sample description rule

In the example above, the question specification `shareproperties(computer screen, scream)` would match the header for the rule in Figure 2. This matching causes the data values (`computer screen, scream`) to be bound to the local variables `X,Y` of the rule, ready for the preconditions of the rule to be tested. These preconditions check further lexical properties and relations, to determine whether this rule is actually applicable in this case. (As with schema preconditions, the implementation represents these expressions in SQL, to facilitate searching of the lexical database.) This may involve testing for the existence of further values (e.g. values for `MerX`, `SynY` in the example above), thereby resulting in the binding of more variables (local to the rule). For example, starting from `X` = *computer screen* and `Y` = *scream*, the precondition testing might find (in the lexicon) variable values `MerX` = *window* and `SynY` = *shout*, thereby satisfying all the conjuncts of the precondition. If the precondition testing succeeds, the template specifier is instantiated (using the current

[1] http://www.postgresql.org

variable values), and these expressions are passed on to the third stage of joke generation, *template filling*. Here, this expression would be `[merSyn, window, shout]`).

The answer specification `phrase(computer, scream)` matches a very basic description rule which passes both values on in an expression `[simple, computer, scream]`; this will later be interpreted (at the template phase) as a concatenation command.

However, the same schema instantiation could have led, if other values were found for `MerX` and `SynY` in the description rule used for the question, to a slightly different variant, such as (7).

(7) What do you call a cry that has pixels?
A computer scream.

Or if a different description rule had been chosen for the question specification, the joke might have been (8).

(8) What do you get when you cross a shout with a display?
A computer scream.

Here, the central idea of the joke (captured by the schema instantiation) is essentially the same in all three versions.

Hence, there are two distinct mechanisms used to achieve textual variation. The variation illustrated above involves slightly different phrases which originate from the same semantic material expanded and realised in varying ways. These are constructed by this middle phase, which is a non-humorous set of linguistic rules about how to build descriptive phrases. (These variations usually occur in the riddle's question, but the data for the answer is also passed through this middle stage, so as to have a cleaner architecture, and also to allow for possible minor adjustments being needed in the linguistic form of the answer data, which does occur with some joke types.)

On the other hand, different stereotyped joke-framing phrases, such as *What do you get when you cross ____* or *What is the difference between ____* are handled by the third phase (Section 4.3) below.

4.3 Surface templates

A template is, loosely speaking, a fixed string of text with some blank slots available for other textual material to be inserted. Building text by filling data (e.g. phrases) into a template is a long-standing and much-used approach to the generation of simple natural language text (Reiter and Dale, 2000). It lacks linguistic subtlety and can be inflexible, but it is very convenient when a particular application (as here) needs to build a few stereotyped forms of text which vary only in a few well-defined places. We have three types of template: *phrasal*, *question* and *answer*. Phrasal templates put the finishing touches to phrases built by the previous stage (description construction), for example inserting articles or prepositions as needed. A question template has the broad outline of a riddle question (e.g. *What do you call a ____ ?*) with slots for phrases to be inserted, and an answer template has the broad structure of a riddle answer (e.g. *They're both ____*) with slots for phrases to be inserted.

A template has two parts: the *header* and the *body*. The expressions provided by the description rules, such as `[merSyn, window, shout]` and `[simple, computer, scream]` are non-deterministically matched against the headers of templates of the appropriate type (question or answer). This causes variables in the template header to be instantiated to the values (such as *window*, *shout*). These values are thereby passed into the body, which is a skeletal textual structure, such as `What do you call a NP(X,Y)`. Recursively, the template handler matches `NP(shout, window)` to a set of phrase templates, one of which yields *NP(shout) with a NP(window)*, and a further template match produces *a shout with a window*. The answer is also produced by the template module, but for an expression like `[simple, computer, scream]` there are no recursive calls – a single phrasal template produces *a computer scream*.

There are various restrictions about which question templates are compatible with which answer templates, and also which templates are viable for the values coming from a particular schema. These combinations are coded up in a table; these are known as the *joke types*, as we found it useful to characterise types of joke in terms of underlying rule combinations.

5 Going beyond JAPE

5.1 The lexicon

Using WordNet as its dictionary gave JAPE several benefits: many lexical entries (about 200,000), word-senses grouped into synonym sets, information about hyponym/hypernyms and meronyms; and the data is freely available in machine-manipulatable form. We therefore took WordNet as our starting point. However, before designing our system, we carried out consultations with relevant experts: adult users of software for augmentative and alternative communication (AAC), and speech/language therapists. This added further requirements to those needed just for joke generation.

5.1.1 Pictures

Our experts were adamant that children with limited literacy would need pictorial images to be displayed alongside words wherever possible. Preferably, these images should be familiar, and compatible with other uses of images that the children might have met. Fortunately, two companies who produce AAC software (Widgit Software Ltd and Mayer-Johnson LLC) kindly gave us permission to use their picture libraries. However, we had to expend a considerable amount of effort manually linking pictures to appropriate WordNet senses (not just to word forms).

5.1.2 Phonetic representation

In order to construct approximate puns (e.g. matching *rude* and *road*), we needed a representation of the phonetic form of each word (orthography, as in WordNet, can be misleading for punning). We used the Unisyn pronunciation dictionary[2] to add phonetic forms to more than 115,000 word forms in our lexicon. This allowed the implementation of a more subtle matching algorithm

[2] `http://www.cstr.ed.ac.uk/projects/unisyn`

for phonetic similarity (near-homophony), based on Ladefoged and Halle (1988) and minimum edit-cost.

5.1.3 Familiarity of words

It is essential to be able to restrict the available vocabulary to words which the intended users (young children, perhaps with poor literacy) are likely to know, as there will be no beneficial effect, and probably some demoralisation, if the software produces riddles with words which are totally incomprehensible. JAPE was liable to produce riddles using extremely obscure words, such as (9).

(9) What do you get when you cross a vitellus and a saddlery? A yolk yoke.

All the available sources of word-familiarity information manifested one (or more) of three problems: assigning ratings (e.g. corpus frequencies) to *word-forms*, not to *word-senses*; sparseness, i.e. covering only a few thousand words; unreliability or unsuitability for our purposes. To address this, we applied a hybrid strategy, which there is not space here to document. This involved assigning ratings of *priority* to several different resources and also scaling the ratings from each resource into a sub-interval of [0,1]. A word-sense was then assigned a rating (in [0,1]) by the highest priority source for which it had a rating.

The STANDUP joke generator has an (adjustable) filter on the ratings of the lexemes used. This can be used to limit the unfamiliarity of words used.

5.1.4 Vocabulary restriction

It must be possible to avoid the use of words which are highly unsuitable for the user population (e.g. swear words, sexual terminology). JAPE was quite capable of producing jokes which, while semantically valid, were socially unacceptable for our target audience; e.g. (10).

(10) What do you call a capable seed?
An able semen.

We introduced a *blacklist* which contains words that must not be used anywhere by the system. It was populated by searching the Shorter Oxford English Dictionary for all entries tagged as either *coarse slang* or *racially offensive*; a few further entries were added to this list by the project members based on personal knowledge of words likely to be deemed unsuitable by teachers. (Despite this, a teacher objected to one riddle with quite innocent lexemes: *What do you call a queer rabbit? A funny bunny.*)

5.2 Avoiding simple faults

Although the JAPE riddle generator produced structurally correct texts, some of them were far from acceptable as jokes. We implemented various heterogeneous improvements, generally formal checks to eliminate configurations of lexemes which would lead to (intuitively speaking) poorer output; that is, we did not so much positively improve the jokes as selectively close off some of the more noticeable and formally definable routes to weak jokes.

5.2.1 Shared roots.

Early versions of STANDUP produced riddles in which the same word (or morphological variants of a word) appeared in both the question and the answer, which tended to spoil the joke: *What do you get when you cross a school principal with a rule? A principal principle.* Using information from the Unisyn dictionary we were able to associate a 'root' field with lexemes, and filter out riddles in which the same root appeared in question and in answer.

5.2.2 Excessive abstraction.

Many words in our lexicon were ultimately linked (via the WordNet hyponym/hypernym hierarchy), to very abstract entries such as *entity* or *human activity*. This could cause riddles to be excessively obscure; for example: *What do you get when you cross an aristocracy with a quality? A nobility mobility.* Here, *quality* is a hypernym of *mobility*, but this gives an excessively imprecise question. We therefore placed some of the roots of the hypernym forest in a further list of lexemes to be excluded from use. This was done by subjective judgement of the degree of abstraction, not by considering jokes which included the concepts. Although this removed many baffling riddles, the phenomenon of unworkable abstraction is more subtle. Example (11) is from an early version of STANDUP (before we improved the phonetic matching), and presumably puns on *double-decker* (a two-level bus):

(11) What do you call a cross between a coach and a trained worker?
A double baker.

The phrase *trained worker* is found by a description rule seeking hypernyms of *baker*. But *trained worker*, although not as wildly abstract as *entity* or *quality*, is still too vague to invoke the specific notion of *baker*. A hypernym should be used in a riddle question only if it is close enough in meaning to the target item (here, *baker*). It is hard to specify an appropriate criterion of "closeness".

5.3 Changes in coverage

STANDUP's set of schemas is slightly different from that in JAPE. Although we added one further schema (so that substitutions of a word into another word could happen at the end as well as at the start), there were fewer schemas (11 to JAPE's 15). This is due to two factors. Firstly, we were able to combine certain JAPE schemas which were very similar. Secondly, we had to omit some of the JAPE schema, for jokes such as (3). These schemas rely on information about what nouns are suitable subjects or objects for verbs, which, in the JAPE project, was compiled by hand in a relatively labour-intensive fashion. It was not clear how best to scale this up automatically (although it is conceivable that "Word Sketch" data (Kilgarriff et al., 2004) might help). Given the limited resources of our project, we had to do without these schemas. This highlights the difference in purpose between a research prototype and a working system. A prototype is often built to test whether some particular algorithm or design will work in principle – a proof of concept. This can be achieved if the necessary knowledge resources or environment can be created, even if only on a small scale. Thus, Binsted demonstrated a valid computational route to riddles such as (3) (and one or two other verb-based types), but this is very different from devising a practical means to make a large scale system which exploits this route.

5.4 User interaction

Perhaps the most significant advance in the STANDUP system is interaction between user and joke generator. There is a bright, colourful child-friendly GUI, using specially-designed graphic images, which allows the user to control the generator through a number of buttons. Thus a joke can be requested which contains a specified word, is on a given topic (e.g. *animals*) or is of a given type (e.g. *where the start of words are swapped round*). The user can browse through past jokes made at previous sessions, or save jokes to his/her own 'favourites'. When choosing a word for a joke, the user can browse through the lexicon.

Response time ranges from under a second to several seconds. This has been achieved by efficient coding and by caching (as database tables) lexical information used by the joke generator, such as near-homophone lists, tuples of lexemes forming spoonerisms, etc., and also instantiations of schemas.

The software has a Control Panel through which a researcher, teacher or carer can customise the system's behaviour (what appears on the screen, what kinds of jokes are available, what input/output mechanisms are used, etc.) for individual users, in a very flexible manner.

5.5 Joke telling

Part of the motivation for this work came from the idea that a child who used a voice-output communication aid (VOCA) – i.e. using a speech synthesiser in order to "speak" – might like to incorporate jokes into their conversation. However, it would have been over-ambitious to attempt to incorporate the joke-building functionality into a VOCA at this stage of development, so we instead developed a stand-alone system which a child could experiment with. Our software had a built-in text-to-speech system (using FreeTTS[3]) for reading messages, button labels, etc. to the user. There was also a facility whereby the user could, having obtained a joke, "tell" it step-by-step (question, pause, answer) by getting the software to speak the text, with the user controlling this process through the pointing device. This proved to be highly popular with the users (Section 6 below), as the children could tell their newly-built jokes immediately without having to switch over to their VOCA and enter the text.

6 Evaluating the system

For our software, usability and effectiveness for our target group were central. We therefore evaluated STANDUP with a group of children with CCN (fuller details can be found elsewhere).

A single case-study methodology was used with nine pupils at a special-needs primary school. All had cerebral palsy, and were in the 8 - 13 year age group. Their literacy levels were rated as either *emerging* or *assisted*. Eight of the participants were users of various communication aids, and could interact with these via touch screens or, in four cases, head switches. Children were taken through five phases: *baseline testing, introductory training, intervention, evaluation, post-testing*, where the three central phases involved sessions with the software. *Introductory training* consisted of familiarisation with the system, aided by one of the project team. *Intervention* involved the child having a simple task (suggested by the researcher) to try with the software, such as finding a joke on a particular topic. The researcher also offered guidance as necessary. For the *evaluation* phase, tasks were again suggested, but no help was given unless absolutely essential. Sessions were video-taped for analysis, and the software logged user-interactions into a disk file. Follow-up interviews and questionnaires were conducted with school staff and the participants' parents.

In the *baseline testing*, two standard multiple-choice tests for facility with words were administered: Clinical Evaluation of Language Fundamentals, CELF, (Semel et al., 1995), in which 27 questions each ask for a choice of 2 semantically related words from a set of 4, and a rhyme-awareness test from the Preschool and Primary Inventory of Phonological Awareness, PIPA (Frederickson et al., 1997). We also tested each child's grasp of punning riddles, using the Keyword Manipulation Task (O'Mara, 2004), simply to check our assumptions about the level of the children's understanding.

The *post-testing* with PIPA (testing awareness of rhyme) showed no signs of improvement (although 6 of the 9 scored above 80% on both pre- and post-test, suggesting a possible ceiling effect). On the CELF post-test, all but one of the participants improved, the mean improvement being 4.1 out of 27 (paired t-test, two-tailed, yields $t = -3.742$, $df = 8$, $p = 0.006$). It is difficult to conduct randomised controlled trials in the AAC field, as the set of people who use AAC tends to be highly heterogeneous. In the absence of any comparison with a control group, it is hard to infer much from the scores.

All the children reacted very positively to their time with the STANDUP software. One of the older boys, who had good verbal abilities, complained about the quality of the jokes, but made insightful comments on possible improvements to the system. The pupils spontaneously used the software (some without need for prompting), enjoyed having the software tell the jokes to others, and re-told the jokes afterwards to parents and others. Children initiated interaction, some for the first time. This may be because they felt that the program provided them with novel language, and that they could truly control an interaction by telling a new joke, instead of repeating vocabulary stored in devices by their therapists/teachers. The computer-generated jokes became part of an existing class project, with pupils posting their favourite examples publicly.

Although this was a very small qualitative study, with no ambitions to show skill improvements over such a short term, there was anecdotal evidence (from parents and teachers) that children's attitudes to communication had improved. Since it was far from clear at the outset

[3] `http://freetts.sourceforge.net/docs/index.php`

of our project whether children with CCN would even be able to use the planned software, the results are not trivial.

7 Discussion

7.1 The outcome

We have designed and built a fully working, large-scale, robust, interactive, user-friendly riddle generator, with a number of auxiliary facilities such as speech output and adjustable user profiles, remedying the limitations of JAPE listed in Section 3. It can create millions of jokes, has been evaluated in a non-trivial way, and is available for download over the WWW. Although we started from the JAPE ideas, our design and implementation effort was probably around four to five person-years of full-time work (excluding the evaluation). The area where further improvement is most needed is joke quality.

7.2 Creativity

7.2.1 Is the software creative?

Binsted did not claim that JAPE was creative, but Boden (1998, 2003) discusses it as an example of a creative program. Given STANDUP's relationship to JAPE, the question of creativity again arises. Anecdotal evidence suggests that the program produces novel and acceptable jokes, but, in the absence of a formal evaluation of the output (cf. Binsted et al. (1997)), no solid claim can be made. All of these jokes are based on hand-crafted schemas, so there is no creation of novel *types* of joke. (In the sense of Boden (1998), such innovation would be *transformational* rather than *exploratory* creativity.)

Is STANDUP "more creative" than JAPE? The devices described in Section 5 above alter the set of output items (compared to JAPE's, or – more realistically – to an earlier version of STANDUP). The modifications in Section 5.2 eliminate poorer items, thereby enhancing the overall output quality. By the formal criteria 1 to 4 (and possibly 5) in Ritchie (2001a, forthcoming), the elimination of faulty items would improve the program's ratings, as these criteria assess the *proportion* of the output items which are categorisable as jokes, or which are classed as *good* jokes. Some other changes (Section 5.3) eliminate certain classes, making the output set less varied. Ritchie's proposed criteria do not assess output set variety, so this would not affect the rating of the STANDUP program, but Pereira et al. (2005) hint that less variety in output is a sign of lower creativity. It is hard to draw firm conclusions here (except perhaps that these criteria are insufficiently subtle for making fine comparisons of creativity).

7.2.2 Supporting creativity

Given that the whole project was intended to give support to the users' skills development, perhaps a relevant viewpoint to consider is the extent to which the software supports or encourages *human* creativity. This is very hard to assess. If there were a fuller study to determine the effect of software usage on a child's skills (including social actions), perhaps some educational tests of creative thinking could be used to assess this aspect.

7.3 Future directions

Further studies. It would be very illuminating to carry out a long term study of the use of the software by children, to obtain some idea of the effects such language play has on linguistic, communicative or social skills. Comparisons with other "language play" educational software would be interesting, as would studies with other user populations (e.g. children with autism, second-language learners).

Improving the system. Because of our requirements studies (and the limited time available), we implemented relatively simple facilities for user interaction. These could be extended, to allow greater participation by the user in the joke-building process. It would also be interesting to handle other joke types (e.g. 'knock-knock' jokes (Taylor and Mazlack, 2004)).

A testbed for humour. The STANDUP software could become a framework to test ideas about humour, in limited ways. Allowing users to record reactions to jokes would allow the collection of data about which generated items work best, a resource for researchers. Alternatively, it might be possible to embed, in a future version, some conjecture about factor(s) which affect funniness, and then determine the empirical effectiveness of this.

7.4 Conclusions

Computational humour may still be at a basic level, but the work here represents an significant milestone in its development: a complete working system that addresses a practical application and is accessible for ordinary users.

Acknowledgements

This work was supported by the UK's Engineering and Physical Sciences Research Council. The Widgit Rebus symbols are the property of Widgit Software Ltd and are used under licence. The Picture Communication Symbols are the property of Mayer-Johnson LLC and are used under licence. We are extremely grateful to Capability Scotland and the staff and pupils at Corseford School for their help with the evaluation sessions.

References

Binsted, K. (1996). *Machine humour: An implemented model of puns*. PhD thesis, University of Edinburgh, Edinburgh, Scotland.

Binsted, K., Bergen, B., and McKay, J. (2003). Pun and non-pun humour in second-language learning. In *Workshop Proceedings, CHI 2003*, Fort Lauderdale, Florida.

Binsted, K., Pain, H., and Ritchie, G. (1997). Children's evaluation of computer-generated punning riddles. *Pragmatics and Cognition*, 5(2):305–354.

Binsted, K. and Ritchie, G. (1994). An implemented model of punning riddles. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, USA.

Binsted, K. and Ritchie, G. (1997). Computational rules for generating punning riddles. *Humor: International Journal of Humor Research*, 10(1):25–76.

Boden, M. A. (1998). Creativity and Artificial Intelligence. *Artificial Intelligence*, 103:347–356.

Boden, M. A. (2003). *The Creative Mind*. Routledge, London, 2nd edition. First edition 1990.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Mass.

Frederickson, N., Frith, U., and Reason, R. (1997). *The Phonological Assessment Battery*. NFER-Nelson, Windsor.

Hulstijn, J. and Nijholt, A., editors (1996). *Proceedings of the International Workshop on Computational Humor*, number 12 in Twente Workshops on Language Technology, Enschede, Netherlands. University of Twente.

Kilgarriff, A., Rychly, P., Smrz, P., and Tugwell, D. (2004). The Sketch Engine. In *Proceedings of EURALEX 2004*, pages 105–116, Lorient, France.

Ladefoged, P. and Halle, M. (1988). Some major features of the international phonetic alphabet. *Language*, 64(3):577–582.

Lessard, G. and Levison, M. (1992). Computational modelling of linguistic humour: Tom Swifties. In *ALLC/ACH Joint Annual Conference, Oxford*, pages 175–178.

Lessard, G. and Levison, M. (1993). Computational modelling of riddle strategies. In *ALLC/ACH Joint Annual Conference, Georgetown University, Washington, DC*, pages 120–122.

Levison, M. and Lessard, G. (1992). A system for natural language generation. *Computers and the Humanities*, 26:43–58.

Lindsay, G. and Dockrell, J. (2000). The behaviour and self-esteem of children with specific speech and language difficulties. *British Journal of Educational Psychology*, (70):583–601.

McKay, J. (2002). Generation of idiom-based witticisms to aid second language learning. In Stock et al. (2002), pages 77–87.

Mihalcea, R. and Strapparava, C. (2006). Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence*, 22(2):126–142.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1990). Five papers on WordNet. *International Journal of Lexicography*, 3(4). Revised March 1993.

O'Mara, D. (2004). *Providing access to verbal humour play for children with severe language impairment*. PhD thesis, Applied Computing, University of Dundee, Dundee, Scotland.

Pereira, F. C., Mendes, M., Gervás, P., and Cardoso, A. (2005). Experiments with assessment of creative systems: an application of Ritchie's criteria. In Gervás, P., Veale, T., and Pease, A., editors, *Proceedings of the Workshop on Computational Creativity, 19th International Joint Conference on Artificial Intelligence*, volume 5-05 of *Technical Report*, pages 37–44. Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid.

Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK.

Ritchie, G. (2001a). Assessing creativity. In *Proceedings of the AISB Symposium on Artificial Intelligence and Creativity in Arts and Science*, pages 3–11, York, England.

Ritchie, G. (2001b). Current directions in computational humour. *Artificial Intelligence Review*, 16(2):119–135.

Ritchie, G. (2003). The JAPE riddle generator: technical specification. Informatics Research Report EDI-INF-RR-0158, School of Informatics, University of Edinburgh, Edinburgh.

Ritchie, G. (2004). *The Linguistic Analysis of Jokes*. Routledge, London.

Ritchie, G. (forthcoming). Some empirical criteria for attributing creativity to a computer program. *Minds and Machines*. To appear.

Semel, E., Wiig, E. H., and Secord, W. A. (1995). *Clinical Evaluation of Language Fundamentals 3*. The Psychological Corporation, San Antonio, Texas.

Smith, M. (2005). *Literacy and Augmentative and Alternative Communication*. Elsevier Academic Press, Burlington.

Stark, J., Binsted, K., and Bergen, B. (2005). Disjunctor selection for one-line jokes. In Maybury, M. T., Stock, O., and Wahlster, W., editors, *Proceedings of First International Conference on Intelligent Technologies for Interactive Entertainment*, volume 3814 of *Lecture Notes in Computer Science*, pages 174–182. Springer.

Stock, O. and Strapparava, C. (2003). HAHAcronym: Humorous agents for humorous acronyms. *Humor : International Journal of Humor Research*, 16(3):297–314.

Stock, O. and Strapparava, C. (2005). The act of creating humorous acronyms. *Applied Artificial Intelligence*, 19(2):137–151.

Stock, O., Strapparava, C., and Nijholt, A., editors (2002). *Proceedings of the April Fools' Day Workshop on Computational Humor*, number 20 in Twente Workshops on Language Technology, Enschede, Netherlands. University of Twente.

Taylor, J. M. and Mazlack, L. J. (2004). Computationally recognizing wordplay in jokes. In *Proceedings of Cognitive Science Conference*, pages 2166–2171, Stresa, Italy.

Venour, C. (1999). The computational generation of a class of puns. Master's thesis, Queen's University, Kingston, Ontario.

Waller, A. (2006). Communication access to conversational narrative. *Topics in Language Disorders*, 26(3):221–239.