The Construction of a Pun Generator for Language Skills Development

Ruli Manurung*
School of Informatics, University of Edinburgh,
Edinburgh, United Kingdom EH8 9LW

Graeme Ritchie
Computing Science, University of Aberdeen,
Aberdeen, United Kingdom AB24 3UE

Helen Pain
School of Informatics, University of Edinburgh
Edinburgh, United Kingdom EH8 9LW

Annalu Waller, Dave O'Mara, Rolf Black
School of Computing, University of Dundee
Dundee, United Kingdom, DD1 4HN

Running head: Construction of a Pun Generator

Correspondence address:

Dr Graeme Ritchie
Computing Science
University of Aberdeen
Aberdeen AB24 3UE
United Kingdom.

g.ritchie@abdn.ac.uk

*Now at: Faculty of Computer Science, Universitas Indonesia, Depok 16424, Indonesia.

Abstract

Since the early 1990s, there have been a number of small-scale computer programs which automatically constructed simple verbal jokes (puns), but none of these were fully developed systems which could be used for a practical application. We describe the building and testing of the STANDUP program – a large-scale, robust, interactive, user-friendly pun-generator (inspired by Binsted's JAPE program) which is aimed at allowing children, particularly those with communication disabilities, to develop their linguistic skills. The STANDUP system was designed in consultation with potential users and suitable experts, was rigorously engineered using public domain linguistic data, and has a special-purpose child-friendly graphical user interface. The software was tested successfully with real users (children with complex communication needs).

1 Introduction

Although the young field of computational humour has been growing slowly since the early 1990s (see Section 4 below), most of the implementations have been small, exploratory research prototypes, with no practical applications of the ideas. We have developed a fully working joke-generation system (STANDUP – System To Augment Non-speakers' Dialogue Using Puns) which has been evaluated with real users and is freely available for use as an educational resource. Its humour-creating behaviour reflects the current state of the art in computational humour, and it is well in advance of other humour programs in being fully engineered to fulfil a practical purpose.

The central function of the software is to allow young children to explore language (pronunciation, ambiguity, etc.) by creating their own punning riddles through a simple interactive user-interface. A punning riddle is a simple 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.

Such jokes rely on phonetic similarity and basic semantic relationships (such as synonymy), which makes them computationally tractable and (potentially) illustrative of linguistic patterns.

The intended users are children with complex communication needs (CCN) such as impaired speech and limited motor skills (as often results from cerebral palsy); these individuals often have lower levels of literacy than their typically-developing counterparts of the same age (Smith 2005). The motivation for this application is explained more fully elsewhere (O'Mara et al. 2000; O'Mara et al. 2002; O'Mara and Waller 2003; O'Mara 2004; Waller 2006). In brief, the children in the target user population are often unable to engage in the normal forms of language play because of their disabilities. This can, in turn, impede the development of skills in language, communication and social interaction (Lindsay and Dockrell 2000). The STANDUP software is intended as a

"language playground", with which a child can explore sounds and meanings by making up jokes, with computer assistance. The hope is that this will have a beneficial effect on literacy and communication skills.

The core ideas for the joke-construction mechanisms are closely based on those in the JAPE program (Binsted 1996; Binsted and Ritchie 1994; Binsted and Ritchie 1997), but we had to invest considerable effort in transforming these research concepts into a stable, usable, robust, user-friendly, interactive system with acceptable output.

In this paper, we will describe what was involved in using the ideas at the centre of the JAPE program as the basis for a fully practical system. This includes gathering user requirements, designing the user-interface, preparing data resources, enhancing the joke-generation engine, and evaluating the software with real users. Finally, we shall reflect on what this experience illustrates about the development of computational humour applications from early research prototypes.

2 Education, Exploration and Humour

There have been several decades of research into interactive software to assist children in learning linguistic (or other) skills, and it is not feasible to review that whole field here (see (Wenger 1987) for discussion of the earlier years). The STANDUP project differs from much of that work in several ways:

- the software does not have some structured set of knowledge to be understood, nor any representation of skills to be acquired;
- consequently, the software is not constrained by any specific educational curriculum, and has few educational pre-requisites;
- if the child's skills improve while using the software, that does not automatically alter the possible interactions available with the software (although a teacher or therapist can set more advanced options as the child progresses);
- the software makes no attempt to assess the child's skills at any stage.
- the only user-model is the profile of basic settings for a user, which the STANDUP system does not alter.

While all this may suggest that STANDUP is a rather rudimentary system from a pedagogic perspective, with any educational effects being indirect, it emphasises that the system is primarily an environment which allows *exploration* and *play*, rather than a structured learning framework. This is far from being a new idea – using intelligent software to support a child's exploration of ideas is at the heart of the LOGO approach (Papert 1980), which has been around for over thirty years. STANDUP is close in spirit to some other 'learning through play' systems, such as certain story-writing software (Machado et al. 2001; Gjedde

2004; Gjedde 2006; Robertson and Oberlander 2002; Robertson and Good 2003), although some of these have a certain amount of pedagogic structuring (e.g. (Robertson and Cross 2004)).

Although Yuill's humour-based experiments (Yuill 1996; Yuill 1998) use joke material (predominantly riddles) with children, the items are carefully designed (not computer generated) and are used within a structured regime of training and testing. Interestingly, Yuill shows that understanding of simple riddles does relate to broader linguistic skills.

3 Augmented and Alternative Communication

Voice output communication aids (VOCAs) generally allow the users with limited literacy skills to retrieve prestored linguistic items (e.g. sentences, phrases, words) using picture keyboards or dynamic touch screens. Switch access is used by individuals who are unable to use a keyboard or touch screen. Items selected are "spoken" using a speech synthesiser. Software and hardware for chidren with complex communication needs have generally emphasised transactional aspects of communication, i.e. needs-based communication – making requests and giving short responses. The focus has been on making it easier and more efficient for the user to indicate their needs and express their emotions. Engaging in free narrative and topic-based conversation is more difficult, and some of our past work has considered the need for story-telling within computer-assisted conversations, and the effect of providing this to users of communication aids (Waller and O'Mara 2003; Waller 2006). The playful aspect of language use has not been considered to any great extent, although some clinicians (Musselwhite and Burkhart 2002; King-DeBaun 1997) have reported on the use of verbal humour as a support for communication skills such as turn-taking.

The STANDUP system is not intended as a communication aid: it is a standalone system for language exploration. However, as noted in Section 9.1 below, it proved to be usable as a specialised tool for telling computer-generated jokes (and could in the future be combined in some way with a VOCA).

4 Computational Humour

To get a general overview of work in computational humour during the first ten years of the field, see (Ritchie 2001), (Hulstijn and Nijholt 1996), (Stock et al. 2002). The area is still in its infancy. There are no real theories of humour to guide computational work, so projects tend to be narrowly defined, on a small scale, using mechanisms designed for specific tasks, and generally exploratory.

The STANDUP project has taken one of the dominant strands of the past work – pun generation – and successfully subjected it to thorough and substantial engineering effort. Also, even those humour-generating programs which have been evaluated were not tested as actual working systems with real users, whereas the STANDUP trials (Section 10) placed the software in a real setting

with typical users.

For our purposes, we can review past work under two headings: small pun generators, and other systems; JAPE will be dealt with separately in Section 4.3.

4.1 Early pun generators

Since 1992, a number of computer programs have been built which construct simple (usually novel) verbal jokes. These were generally very small studies, often carried out as student projects.

(Lessard and Levison 1992) built a program which, using an existing natural language generator, VINCI (Levison and Lessard 1992), created a simple variety of pun, the *Tom Swifty*. (Lessard and Levison 1993) sketch the workings of a program which, again using the VINCI generator, produces some basic forms of punning riddle. (Venour 1999) built a small program which could generate 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. The WISCRAIC program (McKay 2000; McKay 2002) could produce simple puns in three different linguistic forms (question-answer, single sentence, two-sentence sequence).

These systems operated with quite 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.)

4.2 Other types of system

As the STANDUP program is a punning riddle generator, JAPE and the systems reviewed in Section 4.1 are its antecedents. Other work in computational humour has included a program which could construct amusing acronyms (Stock and Strapparava 2002; Stock and Strapparava 2003; Stock and Strapparava 2005), a recogniser for basic "knock-knock" jokes (Taylor and Mazlack 2004), and some studies of how machine-learning techniques could separate joke texts from non-jokes (Mihalcea and Strapparava 2006; Mihalcea and Pulman 2007). Binsted and Bergen have been involved with a number of small prototypes: a very preliminary generator for insults based on *scalar humour* (Binsted et al. 2003), and a program which selects punchlines for a joke set-up for a particular class of jokes (Stark et al. 2005).

Although some of these were on a slightly larger scale than the pun generators in Section 4.1 above, all of them were research prototypes, with no claims to be usable applications.

4.3 The JAPE riddle generator

The JAPE program (Binsted and Ritchie 1994; Binsted and Ritchie 1997; Binsted 1996) (devised completely independently of the similar and roughly simul-

taneous work by Lessard and Levison, Section 4.1) 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.

The mechanisms used in JAPE (see (Ritchie 2003) for full technical details) are very similar to those in the STANDUP program, which are described in more detail in Section 6 below: three types of rules (*schemas*, *description rules*, *templates*) manipulated words and phrases from a large dictionary.

The two aspects of JAPE which make it stand out from the other early pun-generators is that it used a large, general-purpose lexicon, WordNet (Miller et al. 1990; Fellbaum 1998), rather than a small hand-crafted one, and that 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. However, the joke rated the funniest in the data set was a JAPE creation ((3) above).

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 (or any of its subsequent enhanced versions (Griffiths 2000; Ritchie 2003)) 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:

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

2. 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 use a particular word, or to make a joke on a particular topic).

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

4. The search for suitable words, being unintelligent and exhaustive, could (if the lexicon was suitably large) be very slow; Binsted's test runs took hours. This meant that the response time for joke production was very far from usable for interaction with a user.

5. 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.

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

We endeavoured to address these deficiencies in our design and implementation of the STANDUP system.

5 The user interface

The deficiencies 1, 2 and 3 (Section 4.3 above) were addressed by creating a flexible user interface.

5.1 Determining requirements

Designing the user interface consisted of two questions: what functionality should be available to the user, and how should these functions be presented and controlled? To answer both of these, we conducted a number of requirements-gathering exercises, with some "expert" users (i.e. adults with complex communication needs (CCN)) and with a number of domain experts (teachers and therapists familiar with issues in augmentative and alternative communication (AAC)). This involved focus groups and semi-structured questionnaires to elicit and develop ideas about what an interactive joke-building system should provide for children with CCN. Paper prototypes of screens were devised, and feedback was gathered on these.

Fuller details of the requirements-gathering stage are given elsewhere (Waller et al. 2005).

5.2 Initial steps

Based on the findings described in Section 5.1, we developed a semi-formal specification of the possible user interactions with the system, in the form of a *use cases* document (Cockburn 2000). On the basis of this, a software mock-up was created, in which there was no joke-generator, just a small set of (human-generated) punning riddles, but where the user's access to the jokes was through a prototype of the interactive interface. This mock-up was tested for general usability with three potential users, all with cerebral palsy (typical of the type

of disability the project was focussing on). Some minor changes were needed to the user interface, but no significant problems were found in these trials, so we proceeded to implement a full system.

5.3 The appearance of the interface

The user interface offers the user, at any stage, a relatively small number of choices, each represented as a large button (depicted as clouds against a sky). As well as the various choice buttons (which include general navigational ones such as *Exit*, as well as the central choices relevant to the joke building task), there is a *progress map*, which is a rough pictorial indication of how far through the joke-building process the user is (see Figure 1).

The overall appearance of the interface was created by a professional graphic designer, who also provided specialised icons and a cartoon character (a jester-robot) to act as the persona of the system. Feedback on the icons and jester robot from several children helped to choose the final designs. The general style is colourful and lively.

There are a wide range of options (specific to the current user) which can be set via a control panel (invoked by an escape key). For example, the user interface can be set for use with a conventional mouse, with a touchscreen, or via a single-switch; selecting the latter invokes a scanning regime, a technique used in assistive technology, by which options are highlighted and chosen when the switch is activated. The user's menus for word-choice are modified automatically depending on whether a text input device is available. Speech output can be switched on selectively for different types of text, such as system prompts or jokes when displayed. The presence of pictorial symbols accompanying words can be turned on or off, and there are further options to fine-tune how information is to be displayed and which joke-construction resources are to be used.

This control panel is not intended for the actual end user, who might have difficulty in grasping some of the more technical options, or with selecting options using a mouse (the control panel is not within the flexible AAC-oriented user interface); instead, this facility is aimed at the researcher, teacher, therapist, or carer, who can modify the behaviour of the STANDUP software to suit a particular child or group of children.

6 The joke generator

Riddle generation in STANDUP consists of the same three stages as in JAPE. Each stage consists of instantiating a particular kind of rule: *schemas* (Section 6.1), *description rules* (Section 6.2), and *templates* (Section 6.3).

6.1 Schemas

A schema consists of 5 parts:

Header: As well as attaching a symbolic name to the schema, this lists the schema's variables, which can be thought of as its parameters.

Lexical preconditions: This is the heart of the schema, containing a collection of constraints which specify the core data items needed for a particular subclass of riddle. The types of core items involved are *lexemes* (entries from the STANDUP lexicon) and *word forms* (the orthographic textual representation of a particular word). The constraints can involve syntactic categorisation (e.g. that a lexeme is a noun), phonetic relations (e.g. that two items rhyme), structural relations (e.g. that an item X is a compound noun made up of components Y and Z), and semantic relations (e.g. that 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 processing of description rules (Section 6.2 below).

Answer specification: This specifies how certain variables in the schema are, once instantiated, to be described within the answer part of the eventual 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 *equivalent* 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 these (or equivalent) jokes again to the same user.

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

STANDUP uses 11 schemas (for 11 underlying kinds of joke). A typical schema is shown here:

```
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]
```

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 hand-compiled into an expression in the database query language SQL (Elmasri and Navathe 2000), to facilitate the finding of suitable variable values within the lexical database (Section 7 below). This coding as SQL could in principle have been automated, but it was

not worth investing time in devising a schema-compiler, given the small number of schemas involved.

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 the two further phases of processing) to a riddle such as (6):

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

Instantiating a schema assigns values to schema-variables. Some of these values have to be passed on to the next phase of joke generation, and these are indicated in further components of the schema, the *question specification* and the *answer specification*. These specifications embed the relevant variables within symbolic expressions which act as signals to the next phase (Section 6.2 below) 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)`.

6.2 Constructing descriptions

The middle phase of joke generation – constructing descriptions – was originally introduced in JAPE-2 (Binsted et al. 1997), having not been present in JAPE-1 (Binsted and Ritchie 1994; Binsted and Ritchie 1997). The rationale for this stage is to encode certain linguistic variations which may be possible, given particular core values from the schema instantiation.

The question specification and answer specification are handled separately. Each of these expressions 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*. For example, one description rule is:

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

In the example joke given above for the `newelan2` schema, the question specification `shareproperties(computer screen, scream)` would match the header for this rule. 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 whatever variable values are now current), and these expressions are passed on to the third stage of joke generation, *template filling*. In the example 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 eventually be interpreted (at the template phase, below) 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.

In STANDUP (as in JAPE), 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 generally 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 which might be needed in the linguistic form of the answer data, which does occur when using the two schemas in which one word is substituted for a part of another.)

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 6.3) below.

6.3 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. There are three types of template in STANDUP: *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 appropriate (question or answer) templates. 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.

7 The lexicon

The JAPE lexicon consisted of the standard WordNet lexical database (Miller et al. 1990; Fellbaum 1998), together with a small number of auxiliary knowledge bases: an explicit list of homophones based on Evan Antworth's online version of (Townsend 1975); the British English Example Pronunciation (BEEP) dictionary; some hand-crafted rules for estimating which portions of two words corresponded phonetically when they were similar orthographically; a total of about 1000 *typical verb-subject pairings* and *typical verb-object pairings*, indicating which nouns were suitable subjects or objects for some verbs, created by hand from a children's dictionary; the MRC Psycholinguistic Database (Coltheart 1981), for rating the readability of jokes.

The auxiliary knowledge bases had, from our point of view, a number of problems. There was inconsistency in the phonetic relations represented, some being based on US English and some on British English. Although BEEP contains about 100,000 entries, the resources were all smaller than WordNet (to varying degrees) and in most cases were constructed in ways which were hard to scale up.

As a result of our consultations with users and experts, and consideration

of the needs of the joke generator, we concluded that it would be highly desirable (in some cases essential) for our system to have, in addition to the existing WordNet provisions of *part of speech*, *noun compounds*, *multiple senses for words*, *synonym sets*, *hyponym* and *meronym links*, a number of other facilities.

We therefore started from WordNet, but added various resources, as follows.

7.1 Phonetic similarity

It was essential to be able to compare words and parts of words for phonetic similarity and identity in a uniform manner, preferably on the basis of an English accent which would be compatible with our intended users' intuitions. JAPE compared words for homophony in a rather *ad hoc* way, and had no mechanism for approximate matching (to allow, for example, *road* to pun with *rude*).

The Unisyn dictionary (`http://www.cstr.ed.ac.uk/projects/unisyn`) was used to assign pronunciations to the words in the WordNet lexicon. That is, each lexical entry was allocated a string in the phonetic alphabet used by Unisyn. It was important that the assignments of phonetic form to words were sensitive to the *sense* of a word, in order that puns were internally consistent. For example, if the program is making a pun which relies (semantically) on the word *bass* meaning *singer* (rather than *fish*), then it should not, in the same joke, assume that *bass* is pronounced as in the *fish* sense. For a small number of word senses, there was ambiguity about the correct pronunciation of a word (e.g. *bass*). These were disambiguated by hand.

These phonetic strings were used by the punning mechanisms to determine phonetic similarity, using a least-edit-cost algorithm (Jurafsky and Martin 2000). (See (Manurung et al. forthcoming) for fuller details.)

7.2 Speech Output

As our users might have limited literacy, it was essential to have speech output. The facility to pronounce words and phrases was provided by the FreeTTS text-to-speech system (`http://freetts.sourceforge.net/docs/index.php`), a program which processes (orthographic) texts produced by the joke generator or the user interface. That is, this requirement was not met by encoding data in the lexicon. (Ideally, the accent used by the speech output should be similar to that assumed for phonetic matching, for consistency. We were not in a position to enforce that, but the generality of our approach should make it straightforward to achieve in a future version, subject to processing power and available voice files.)

7.3 Picture support

A further highly desirable support for limited literacy users is that, when displaying a lexical item, a pictorial symbol should, if possible, accompany it, preferably from a standard symbol-library used in AAC. We considered various sources of word-to-picture mappings, such as children's illustrated dictionaries.

The main problem with these (leaving aside availability in machine-readable form, or licensing of proprietary material) was that they linked pictures to *word-forms* (strings of letters) not to *word-senses* (meanings). For STANDUP, the intended use of the symbols was to assist the users in understanding the meaning of words, so it was crucial that where a word-form could have more than one meaning, the appropriate picture would be displayed for the sense intended in context. For example, the word *match* could mean a sporting event or an item for producing a flame, and different pictures are needed for these two senses.

Two widely-used proprietary libraries of pictorial symbols used in AAC devices (i.e. communication aids) are the Widgit Rebus set (by Widgit Software Ltd) and the Picture Communication Symbols (PCS) (by Mayer-Johnson LLC). Both these companies kindly agreed to let us use their symbol sets within our software for the duration of our research, including our trials with users.* Although Widgit Software supplied data linking each of their pictorial symbols to one or more *conceptcodes* (abstract identifiers for semantic concepts) and also to one or more English words, there was still the need to link the pictures to the word *senses* used within our lexicon, for the reason explained earlier. In the absence of any way of automating this, we manually linked the Widgit conceptcodes to WordNet concepts. The resulting mapping between conceptcodes and WordNet identifiers means that there is a subset (about 9000) of our WordNet-derived senses which have associated pictorial symbols. As Widgit Software had already linked their own conceptcodes to PCS symbols as well, our mapping connected our WordNet senses to both Widgit Rebus and PCS libraries.

7.4 Topics

Our experts suggested that word-senses should be grouped into subject-areas (topics) to facilitate access by the user; for example, so that they could build jokes on "sport" or about "food". Also, these topics should be structured, probably into a hierarchy.

Although WordNet has a hypernym hierarchy, it is unsuitable for our purpose, being an abstract ontology rather than a classification of a child's everyday world into recognisable categories. However, the Widgit conceptcodes had been clustered, by Widgit Software Ltd, into more recognisable topics, which are hierarchically grouped. Hence, once the WordNet senses were linked to Widgit conceptcodes, this automatically connected them to the Widgit topic sets. It was convenient for us to adopt this topic hierarchy (which may be familiar to users who have encountered Widgit AAC products).

*For contractual reasons, in released versions of our software after the project period, the use of PCS symbols is limited to messages from the system, without coverage of words in jokes.

7.5 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.

We were faced with three problems in creating a "familiarity score" for the entries in our lexicon. The first was the disambiguation problem: most available resources assigned ratings (e.g. corpus frequencies) to *word-forms*, not to *word-senses*. As in the case of pronunciation (Section 7.1) and pictorial images (Section 7.3), it was essential that the data be linked to word-senses. For example, the WordNet senses for the word *argument* include 'a discussion in which reasons are advanced for and against some proposition or proposal' and also 'a variable in a logical or mathematical expression whose value determines the dependent variable'. These may have quite different levels of familiarity to young children.

A second problem was the sparseness of data: most available sources covered only a few thousand words. Thirdly, the information in existing resources might be unreliable or unsuitable for our purposes.

WordNet has corpus-frequency data for every entry, based on the SemCor sense-annotated corpus (Miller et al. 1993), and this would seem at first to be ideal. However, we found that it was not very suitable for our target user group: some very simple everyday words (e.g. *baker*, *onion*, *sleepy*) scored 0 (i.e. did not appear in the corpus), many others (e.g. *milk*, *nail*) appeared only rarely. On the other hand, some more obscure terms (e.g. *vocational*, *polynomial*) had quite high frequencies. This was, however, the only sense-disambiguated resource we had – all others failed on both the sparseness and ambiguity factors.

To address this, we applied a hybrid strategy. The set of word-senses which have associated pictures in either the PCS or Widgit Rebus symbol libraries (after our hand-disambiguation – Section 7.3) were assumed to be moderately familiar. We took all the word-forms which appeared in a published set of graded words for testing children's spelling (Schonell 2003) and manually attached WordNet senses to them, using our own judgement as to the most suitable (and omitting those where WordNet supplied no candidate senses). This gave us two small, but relatively reliable, hand-disambiguated lists. We used these to assign senses to word-forms appearing in other, more principled lists, particular some of the rated lists in the MRC psycholinguistic lexical database (Coltheart 1981).

Finally, we assigned (again intuitively) a *priority ordering* to these sources, and a *numerical range* to each source. The *FAM-score* for a word-sense was then whatever was assigned by the highest priority source in which that word-sense

appeared, with the value being computed from that source's numerical range and whatever rating the source itself assigned.

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

7.6 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; for example, (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.[†]

As a result of all the above resource development, we had a lexicon with heterogeneous information, derived from a number of sources. It was highly desirable to integrate the various knowledge into a uniform lexical database, to facilitate systematic access. We therefore created a relational database, so that all the relevant properties and links (e.g. lexeme to word-form, hyponym to hypernym, word-sense to FAM-score) were represented as relations between columns in these tables. In this way, a standard SQL query could be used to extract all items or sequences of items which met an arbitrary combination of conditions (as required for instantiating schemas and description rules – Section 6).

Further details of the lexicon can be found elsewhere (Manurung et al. 2006; Manurung et al. 2006; Manurung et al. forthcoming).

8 Improving joke quality

It was fairly clear from the JAPE project that although the riddle generator did indeed produce structurally correct texts, some of them were being very far from acceptable as jokes. If we leave aside the fundamental limitation that we had no theory of humour to guide the joke generator (not something that could be easily remedied in a single project), it appeared, from inspecting the output of JAPE (and of early versions of STANDUP), that there were a number of very different, and often quite trivial, factors undermining the system's efforts at humour. To tackle this, we implemented various heterogeneous improvements. On the whole, these could be classed as inserting formal checks to eliminate configurations of

[†]Despite this, one teacher objected to a STANDUP riddle built from quite innocent lexemes: *What do you call a queer rabbit? A funny bunny.*

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 noticeably poorer (and formally definable) routes to weak jokes.

Avoiding shared roots. Early versions of STANDUP produced riddles in which the same word (or two morphological variants of a word) appeared both in the question and in the answer, which tended to spoil the joke: *What do you get when you cross a school principal with a rule? A principal principle.* Although WordNet has no notion of a word's root, the Unisyn dictionary which we used to allocate phonetic representations did have this information, so we were able to associate a 'root' field with lexemes in our dictionary, and have a check that filtered out riddles in which the same root appeared in question and in answer.

Excessive abstraction. As the lexicon used the WordNet hyponym/hypernym hierarchy, many words were ultimately linked 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, rather than by considering actual jokes which included the concepts. Although this removed many baffling riddles, the phenomenon of unworkable abstraction is more subtle. Consider the rather poor STANDUP example (11).

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

The pun is presumably on *double decker* (a two-level bus), but that is not the point of interest here. The phrase *trained worker* is a hypernym of *baker*, and so is found by a description rule seeking hypernyms. 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 like this only if it is close enough in meaning to the target item (here, *baker*). Since it is hard to specify an appropriate criterion of "closeness", an alternative solution would be to remove the description rules which rely on hypernyms; unfortunately, this would remove half of the description rules used for questions.

9 Other facilities

9.1 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 STANDUP functionality into a VOCA at this stage of development, so we instead developed STANDUP as a stand-alone system which a child could experiment with. As noted earlier, STANDUP had a built-in text-to-speech system for reading messages, button labels, etc. to the user. At a relatively late stage in the design, we incorporated a facility whereby the user could, having obtained a joke, "tell" it step-by-step (question, pause, answer) by having the text spoken by the software, with the user controlling this process by the pointing device. This proved to be highly popular with the users (Section 10 below), as the children could tell their STANDUP jokes immediately without having to switch over to their VOCA and enter the text.

9.2 User profiles

Each user of the STANDUP system chooses a username (entered via the keyboard by the researcher/teacher), and the system thereafter maintains information about that user. This includes two kinds of data: *option settings* (e.g. what kind of input device is to be used, which classes of text are to be spoken by the software, what level of word familiarity is to be used), and *personal data* (which jokes that user has already been shown, and that user's 'Favourites' list.) The STANDUP control panel allows option settings to be altered at any time during a session, and values are retained between sessions.

In order that the various parameters can be set up easily for new users, option settings can be exported to and imported from disc files. Hence, some standard packages such as *beginner*, or *highly-literate-touchscreen-user* can be saved in these files, and either explicitly imported at the start of a STANDUP session, or set to be the default profile for new users. To assist with this, there is a simple *options authoring tool*, separate from the STANDUP system, which mimicks the STANDUP control panel but with the added facility of saving/restoring option files.

9.3 Logging

Optionally, the user's interactions with the STANDUP system can be logged (in a basic XML-based notation) in a disc file. This allows researchers to study usage as required. To assist in such analysis, there is a STANDUP *log player* which drives the STANDUP user interface (without the joke generator) through the steps listed in the log, thereby showing on the screen what would have been appearing during the session.

We also used standard software to dump the simulated re-runs into a video file, so that when analysing the sessions, our researchers could load the video data into any standard video-playing software and have all the benefits of fast-forward, rewind, pause, etc.

10 Evaluating the system

Although one of the notable aspects of the JAPE project was its thorough evaluation (Binsted et al. 1997), that trial tested only the quality of the output. For STANDUP, we were interested in evaluating the effectiveness of the software for its particular application. It was not feasible, within the timescale of the project, to carry out a proper longitudinal study of the effects of long-term use of STANDUP on a child's linguistic or communicative skills. However, we wanted to at least determine that the STANDUP software was usable by the target user group, and that it was potentially beneficial. As well as one or two small pilot studies, we carried out a substantial evaluation with a group of children with CCN (see (Black et al. 2007) for fuller details).

Nine pupils at Capability Scotland's Corseford School[‡] participated in a trial using a single case-study methodology. The children, all with cerebral palsy, ranged from 8 years 4 months to 12 years 9 months, and had literacy levels rated as either *emerging* or *assisted*. Eight of the participants were users of various communication aids (e.g. Dynavox DV4), and could interact with these via touch screens or, in four cases, head switches.

Over a period of ten weeks, there were five phases: *baseline testing, introductory training, intervention, evaluation, post-testing*. 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 administered a locally developed assessment of a child's grasp of punning riddles, the Keyword Manipulation Task (O'Mara 2004). The introductory training stage allowed the children to familiarise themselves with the system with the help of one of the project's researchers. In the intervention phase, the researcher suggested simple tasks that the children could try with the software (such as finding a joke on a particular topic), and offered guidance as necessary. (It had become clear during the training phase that it was not feasible to have a rigid schedule of tasks the completion of which could be measured – it was unrealistic to attempt to override a child's own desires and inclinations.) For the evaluation phase, tasks were again suggested, but no help was given unless absolutely essential. All of the sessions were video-taped for later analysis, and STANDUP's own logging system recorded all user-interactions into a disk file. Afterwards, follow-up interviews and questionnaires were used to get feedback from school staff and from the participants' parents.

All but one of the children reacted very positively to their time with the software – one of the older boys, who had good verbal abilities, complained about the quality of the jokes, but provided insightful feedback on how the system might be improved. Pupils spontaneously used the software (without need for prompting), took pleasure in having the software tell the jokes to others, and

[‡] `http://schools.capability-scotland.org.uk/`.

re-told the jokes afterwards to parents and others. The STANDUP-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, it is interesting to note that there was anecdotal evidence (from parents and teachers) that children's attitude to communication had improved. The parents of one child had previously not wished their child to use a VOCA, as they were concerned that it might prevent the child from developing natural speech. However, on seeing the enlivening effect on their child of being able to tell jokes using the STANDUP software, they changed their decision.

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 the participants improved except the pupil who complained about the jokes; the mean improvement was 4.1 (out of 27).[§] It is difficult to conduct randomised controlled trials in the AAC field as the set of people who use AAC tends to be heterogeneous. In the absence of any comparison with a control group, it is hard to infer much from the scores.

Since it was unclear at the outset of the STANDUP project whether children with CCN would even be able to use the planned software, the level of positive reaction came as a considerable – and welcome – surprise.

11 Discussion

11.1 The outcome

By the end of the project, we had succeeded in our aim of translating the research prototype ideas of JAPE into 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. The software has been used not only in our evaluations, but was used by participants (teachers and therapists) at a two-day workshop on Language Play and Computers (August 2006) and has been made available to a number of other researchers on a trial basis. The software is available free of charge, and can be downloaded from the STANDUP website, `http://www.csd.abdn.ac.uk/research/standup`.

What was clear from this project was the considerable amount of work required to achieve this transformation. The JAPE system was a better starting point than many programs produced during doctoral work, as it was well documented, had an explicit (and accurate) abstract model (as distinct from merely being source code) and had already been explored in a couple of student projects (Griffiths 2000; Low 2003). Also, the two original supervisors of the

[§]It is probably not appropriate to apply serious statistical analysis to data from such a preliminary exploratory study, but, for those who are interested, applying the paired t-test, two-tailed, to the CELF scores yields $t = -3.742$, $df = 8$, $p = 0.006$.

JAPE project were involved in STANDUP. Nevertheless, the design and implementation effort involved in building (from scratch) the STANDUP system was considerable, being probably around four to five person-years of full-time work (without including the further labour involved in the evaluation phase). ¶

11.2 Changes in coverage

The set of schemas in STANDUP 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), STANDUP had 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, those involving verbs, for jokes such as (3) and (12).

(12) What's the difference between a sea and a sale?
You can sail a sea but you can't see a sale.

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. As we could not see a way to scale this up automatically, 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 done satisfactorily if the necessary knowledge resources or environment can be created, even if only on a small scale. Thus Binsted had shown that there was a valid computational route to riddles such as (3) and (12), but this is very different from devising a practical means to make a large scale system which exploits this route.

11.3 Future directions

In view of the success of the STANDUP project, there are a number of ways in which this work could be taken further:

Longitudinal studies. It would be very illuminating to carry out a long term study of the use of STANDUP by children, to determine (as rigorously as can be managed) what effects this language play might have on linguistic, communicative or social skills. Comparisons with other educational software, particularly of a "language play" nature, would be interesting.

¶Three project leaders (HP, GR, AW) contributed to the design to varying degrees, one research assistant (DOM) carried out a large portion of the gathering of requirements and the design of the user interface, and one research assistant (RM) carried out *all* the implementation as well as contributing to the whole system design, the gathering of requirements and the evaluation sessions. RB took a leading role in the evaluations, having replaced DOM in the final year of the project.

Other user populations. Although the software was primarily designed for children with the types of disabilities typified by our evaluation group at Corseford School, it would be interesting to explore use by other educational groups, such as children with autism, or second-language learners.

Richer interactions. At the start of the work, we considered a number of ways in which a user could potentially interact with a joke-generator, collaborating to choose words or joke patterns. However, our requirements studies (and the limited time available) indicated that we should initially aim for a relatively simple set of possible interactions. Now that we have a stable software platform, we would like to return to this question, by incorporating further ways in which users could guide, or be guided by, the software. This could provide a richer educational environment.

Other joke types. The project was limited to punning riddles because there was proof (via JAPE) that such jokes could be handled computationally in a systematic manner using just lexical resources. If further joke types could be implemented (e.g. 'knock-knock' jokes (Taylor and Mazlack 2004)), this would provide a richer range of linguistic play for the users.

A testbed for humour. The STANDUP software could become a framework to test ideas about humour, at least in limited ways. If users were given a facility to record their opinions of a joke (using a simple slider, or even a button labelled '*I don't understand!*'), it would be possible to gather a large amount of data about which punning riddles work best. This could be used by researchers looking for regularities in what children find funny. Alternatively, it might be possible to embed, in a future version of STANDUP, some conjecture about factor(s) which affect funniness, and then determine the empirical effectiveness of this.

11.4 Conclusions

As noted above, the evaluations were not intended to prove conclusively that using STANDUP has a beneficial long term effect on children's skills. However, they were enough to demonstrate that the software is very definitely usable by the intended user group, and there is strong anecdotal evidence to suggest that, in more protracted use, it could well have the desired effects, even if only in a small way.

We have also shown that even a relatively clean research prototype like JAPE may require considerable labour to transform its concept into a complete application. However, we believe that the effort has been well worthwhile, demonstrating both the viability of the ideas for this educational purpose and the practicality of taking computational humour research to this further level.

Acknowledgements

This work was supported by grants GR/S15402/01 and GR/S15419/01, and the preparation of this paper was supported by grant EP/E011764/1, all from the UK 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. Thanks are also due to the Aberdeen NLG group for comments on an earlier draft of this paper.

References

Binsted, K. 1996. *Machine humour: An implemented model of puns*. Ph. D. thesis, University of Edinburgh, Edinburgh, Scotland.

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

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

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

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

Black, R., A. Waller, G. Ritchie, H. Pain, and R. Manurung. 2007. Evaluation of joke-creation software with children with complex communication needs. *Communication Matters 21*(1), 23–27.

Cockburn, A. 2000. *Writing Effective Use Cases*. Boston, MA: Addison-Wesley Professional.

Coltheart, M. 1981. The MRC psycholinguistic database. *The Quarterly Journal of Experimental Psychology 33(A)*, 497–505.

Elmasri, R. and S. B. Navathe. 2000. *Fundamentals of Database Systems* (Third ed.). Reading, Mass.: Addison-Wesley.

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

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

Gjedde, L. 2004. Designing for learning in narrative multimedia environments. In S. Mishra and R. C. Sharma (Eds.)., *Interactive Multimedia in Education and Training*, Chapter 6. Hershey, PA: Idea Group Publishing.

Gjedde, L. 2006. Story-based e-learning as a vehicle for inclusive education. In L. Gjedde and A.Méndez-Vilas (Eds.)., *Current Developments in Technology-Assisted Education*. Madrid: Formatex.

Griffiths, P. 2000. Lexical support for joke generation. Master's thesis, Division of Informatics, University of Edinburgh, Edinburgh, Scotland.

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

Jurafsky, D. and J. H. Martin. 2000. *Speech and Language Processing: an Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. New Jersey: Prentice-Hall.

King-DeBaun, P. 1997. Computer fun and adapted play: strategies for cognitively or chronologically young children with severe disabilities; Part I and II. In *Proceedings of Technology And Persons With Disabilities Conference*, California State University, Northridge, CA. http://www.csun.edu/cod/conf/1997/proceedings/csun97.htm (last visited February 2007).

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

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

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

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

Low, A. 2003. Software support for joke creation. Honours dissertation, School of Informatics, University of Edinburgh, Edinburgh, UK.

Machado, I., P. Brna, and A. Paiva. 2001. Learning by playing: supporting and guiding story-creation activities. In J. D. Moore, C. L. Redfield, and W. L. Johnson (Eds.)., *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future*, pp. 334–342. Amsterdam: IOS Press.

Manurung, R., D. O'Mara, H. Pain, G. Ritchie, and A. Waller. 2006. Building a lexical database for an interactive joke-generator. In *Proceedings of Language Resources and Evaluation Conference*, Genoa.

Manurung, R., G. Ritchie, D. O'Mara, A. Waller, and H. Pain. 2006. Combining lexical resources for an interactive language tool. In *Proceedings of the 12th Biennial International Conference of the International Society for Augmentative and Alternative Communication*, Düsseldorf, Germany. CD.

Manurung, R., G. Ritchie, H. Pain, A. Waller, R. Black, and D. O'Mara. Forthcoming. Adding phonetic similarity data to a lexical database. *Language Resources and Evaluation*.

McKay, J. 2000. Generation of idiom-based witticisms to aid second language learning. Master's thesis, Division of Informatics, University of Edinburgh, Edinburgh, Scotland.

McKay, J. 2002. Generation of idiom-based witticisms to aid second language learning. See Stock, Strapparava, and Nijholt (2002), pp. 77–87.

Mihalcea, R. and S. Pulman. 2007. Characterizing humour: An exploration of features in humorous texts. In *Proceedings of the Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*.

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

Miller, G., C. Leacock, T. Randee, and R. Bunker.. 1993. A semantic concordance. In *Proc. 3rd DARPA Workshop on Human Language Technology*, Princeton, USA.

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

Musselwhite, C. and L. Burkhart. 2002. Social scripts: co-planned sequenced scripts for AAC users. In *Proceedings of Technology And Persons With Disabilities Conference*, California State University, Northridge, CA. http://www.csun.edu/cod/conf/2002/proceedings/csun02.htm (last visited February 2007).

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

O'Mara, D., A. Waller, and J. Todman. 2002. Linguistic-humour and the development of language skills in AAC. In *Proceedings of 10th Biennial Conference of the International Society for Augmentative and Alternative Communication*, Odense, Denmark.

O'Mara, D. A. and A. Waller. 2003. What do you get when you cross a communication aid with a riddle? *The Psychologist 16*(2), 78–80.

O'Mara, D. A., A. Waller, and G. Ritchie. 2000. Joke telling as an introduction and a motivator to a narrative-based communication system for people with severe communication disorders. *Interfaces 42*(13).

Papert, S. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

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

Ritchie, G. 2001. 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*. London: Routledge.

Robertson, J. and B. Cross. 2004. Children's perceptions about writing with their teacher and the storystation learning environment. *International Journal of Continuing Engineering Education and Lifelong Learning 14*(6), 454–471.

Robertson, J. and J. Good. 2003. Using a collaborative virtual role-play environment to foster characterisation in stories. *Journal of Interactive Learning Research 14*(1), 5–29.

Robertson, J. and J. Oberlander. 2002. Ghostwriter: Educational drama and presence in a virtual environment. *Journal of Computer Mediated Communication 8*(1).

Schonell, F. J. 2003. *The Essential Spelling List*. Cheltenham, UK: Nelson. First published 1932.

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

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

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

Stock, O. and C. Strapparava. 2002. Humorous agent for humorous acronyms: the HAHAcronym project. See Stock, Strapparava, and Nijholt (2002), pp. 125–135.

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

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

Stock, O., C. Strapparava, and A. Nijholt. (Eds.) 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 L. J. Mazlack. 2004. Computationally recognizing wordplay in jokes. In *Proceedings of Cognitive Science Conference*, Stresa, Italy, pp. 2166–2171.

Townsend, W. C. 1975. *A handbook of homophones of General American English*. Waxhaw, N.C.: International Friendship.

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

Walker, W., P. Lamere, and P. Kwok. 2002. FreeTTS - a performance case study. Technical Report TR-2002-114, Sun Microsystems, Menlo Park, CA. `http://research.sun.com/techrep/2002/abstract-114.html`.

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

Waller, A. and D. O'Mara. 2003. Aided communication and the development of personal storytelling. In S. von Tetzchner and N. Grove (Eds.)., *Augmentative and Alternative Communication: Developmental Issues*, Volume 11, pp. 256–271. London: Whurr Pub. Ltd.

Waller, A., D. O'Mara, R. Manurung, H. Pain, and G. Ritchie. 2005. Facilitating user feedback in the design of a novel joke generation system for people with severe communication impairment. In *Proceedings of 11th International Conference on Human-Computer Interaction*, Las Vegas.

Wenger, E. 1987. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Los Altos, CA: Morgan Kaufmann.

Yuill, N. 1996. A funny thing happened on the way to the classroom: Jokes, riddles, and metalinguistic awareness in understanding and improving poor comprehension in children. In C. Cornoldi and J. Oakhill (Eds.)., *Reading Comprehension Difficulties: Processes and Intervention*, pp. 193–220. Mahwah, NJ: Lawrence Erlbaum Associates.

Yuill, N. 1998. Reading and riddling: The role of riddle appreciation in understanding and improving poor text comprehension in children. *Cahiers de Psychologie Cognitive 17*(2), 313–342.

Appendix: Sample output

Unlike the JAPE project, STANDUP included no evaluation of the humorous quality of its constructed jokes. The categorisation below (into good and bad) is informal and subjective.

These are some of the STANDUP system's better jokes.

What do you call a fish tank that has a horn?
A goldfish bull.

What kind of a temperature is a son?
A boy-ling point.

Why is a bronzed handle different from a fringe benefit that is lordly?
One is a tanned grip, the other is a grand tip.

What do you call a heavenly body with an assembly line?
A manufacturing planet.

What do you get when you cross a degree celsius with a water?
A high c.

What is the difference between a desolate amusement and a smart impact?
One is a bleak show, the other is a chic blow.

What do you get when you cross a frog with a road??
A main toad.

What do you get when you cross a kerb with a preserve?
A lemon curb.

What do you call a cross between a bun and a character?
A minor roll.

What do you call a washing machine with a september?
An autumn-atic washer.

What do you get when you cross a choice with a meal?
A pick-nic.

How is an unclean tinned meat different from a pampered sacred writing?
One is a soiled spam, the other is a spoiled psalm.

What do you call a cross between a cereal and a vegetable?
A grain green.

What do you call an oxen fight?
A cattle battle.

What do you get when you cross a buffalo with a sink?
A bison basin.

What do you get when you cross a fragance with an actor?
A smell gibson.

What do you call an author with a zipper?
A skirt vonnegut.

The following are some of STANDUP's weaker efforts.

What do you call a dull whole lot?
A gray deal.

How is a jolly hearth like a gay occurrence?
They are both merry fires.

What kind of jug has a feather?
A long bird.

What do you call a paw with a state?
A right land.

What do you call a preference check?
A stress taste.

What do you get when you cross a superiority with a district?
A favorable possession.

What do you call a banquet control?
An iron feast.

What kind of a limb has a Bahrain?
An arab leg.

What do you call a bread that has a ewe?
A gold toast.

Figure 1: A screenshot of the main menu display.