MUSTE, Appendix A: Resarch Program

1 Purpose and aims

In the last years several different modes of human computer interaction have emerged and are used by millions of people all around the world. The traditional way of using a physical keyboard and a mouse for writing and editing text has been challenged by new interfaces such as speech recognition, touch screens, eye tracking and even brain computer interfaces. New mobile devices have made it easier for people to interact with their computers, the internet, and people around the world.

Until around 10 years ago, the only way for people to enter text on a computer was by using a typewriter keyboard, and this mode of interaction has fostered a view of text authoring as an incremental left-to-right process. Over the last years, touch-screen phones and tables have become increasingly common, and other modalities such as speech, eye tracking or even brain computer interfaces have emerged. Still this incremental view of text authoring has by and large continued. The amount of text interactions has increase enormously over the years – e.g., over 50 billion text messages are sent every day, only counting SMS and chat clients [27]. But still the full potential of the new modalities remains largely unexploited.

There are several problems with the incremental left-to-right view, especially when it comes to new modalities such as touch screens:

- A virtual touch-screen keyboard is cognitively demanding, since the user cannot get haptic feedback but instead have to constantly look at the virtual keys. This is worsened if the screen is small, making it more difficult to find the correct key to touch.
- Letter-by-letter text authoring is by itself demanding for cognitively disabled users, since it requires so many interactions with the device. By cognitively disabled, we do not only mean people with communicative or physical disabilities, but also normal-developed people doing something cognitively difficult such as driving a car.
- The incremental view focuses on how to enter new text, and does not give much help when it comes to editing existing text. This is standardly done by first positioning a cursor at the correct location, and then use the keyboard again to delete the existing text and add new. This has the effect that it is even more cognitively demanding to modify existing words than it is to enter new words.

Most solutions to these problems rely on using a built-in lexicon to do automatic spelling correction and predictive suggestions, but the available techniques still rely on the same conceptual idea that text is written incrementally left-to-right, using a typewriter keyboard.

1.1 Purpose of the project

The overall research problem that we want to solve in this project is to reduce the cognitive load when authoring and editing text on devices with non-traditional input modalities. We do this by changing the conceptual view of how text is authored. The traditional view is that a text is built by adding new words at the end until it is complete. Our view is instead that a text is authored by starting with an existing text and modifying it until it is satisfactory. The user can choose to modify any word or phrase in the text, not just at the end. In this sense, text authoring can be viewed as a two-dimensional process as opposed to the one-dimensional adding of words.

Our interpretation is therefore that text editing is a dialogue between the user and the system. The user selects a word or a phrase that she wants to modify, and the system suggests alternative formulations, by inflecting, replacing or deleting existing words, or by adding new words. The user selects an alternative formulation, the text gets modified, and the dialogue can continue with another step. The more specific research problem that we need to solve is then how to suggest good alternative formulations which help the user accomplish her task.

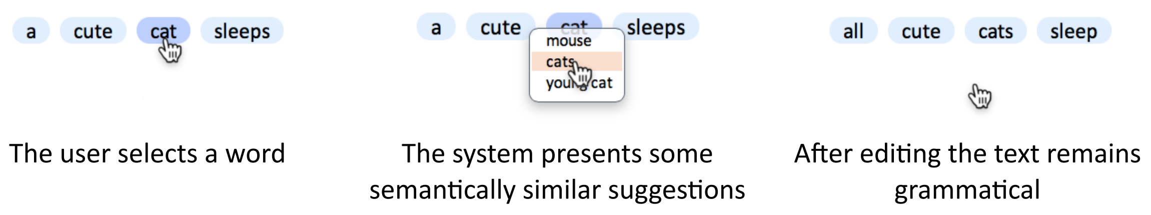
a
cute
cat
sleeps
a
cute
cat
sleeps
mouse
cats
young cat
all
cute
cats
sleep
The user selects a word
The system presents some semantically similar suggestions
After editing the text remains grammatical

Figure 1: A simple example of multimodal semantic text editing

1.2 Multimodal semantic text editing

Our idea of text editing is exemplified in figure 1. In this example, the user selects a word (*"cat"*) that she wants to edit, after which the system presents a pop-up menu of possible choices. The user now selects one of the alternatives (*"cats"*, in plural), and the sentence is changed. One of our important goals is to reduce the cognitive load on the user by reducing the number of interactions with the system. Therefore the system must be helpful and give plausible suggestions. This is the reason why the system should use grammatical knowledge to inflect the words in the rest of the sentence so that it stays grammatical. In the example, the word *"sleeps"* is inflected, and the word *"a"* is replaced by *"all"*, to keep the sentence grammatical.

Note that we do not prohibit other modalities for text authoring. The editing process must start with an initial text somehow, which could be something returned by a speech recognizer, a prototypical utterance that is designed to be edited, or even a text that is entered using a keyboard.

More specifically, our aims in this project are as follows.

A computational theory for semantic suggestions When the user has selected a word or a phrase, the system needs to come up with plausible alternatives. The main problem here is to reduce the number of alternatives so that it can be presented to the user. For this we need a theory for calculating grammatically and semantically similar words and phrases.

A Grammar library for semantic text editing To be able to give grammatical and semantical suggestions, the system needs a notion of grammaticality. This will be done by developing a grammar library that can be used when creating applications for semantic text editing. The library will be based on an existing multilingual phrase-structure grammar [18].

An interpretation of editing gestures The text needs to be stored internally as some kind of semantical term, from which the system can calcluate new suggestions. This means that the we need to be able to interpret gestures by the user into editing operations on the unerlying semantic term.

Functioning prototypes that can be evaluated We will develop two prototypes for text editing applications. The prototypes will function as proof-of-concept that our theory is feasible, but we will also use them for evaluating the feasibility of our approach on users.

2 Survey of the field

The conceptual idea described in section 1.2, to edit an underlying abstract term by operating on the surface words, has not been described before in the literature. There has been lot of research on text authoring in different modalities, but most of it is focused on left-to-right text input, and there is not much work done on editing existing text. There has been some research on using a grammar for conceptual authoring, but either the user have to edit the abstract term directly, or it has been focused on left-to-right authoring.

Text input on mobile devices There are still mobile phones which do not use a touch screen, and instead use the numeric keypad for entering text. Usually they use predictive input which is based on a built-in lexicon and/or morphological rules [12]. For touch screen mobile devices, the standard way of editing text is to use a virtual keyboard, usually with a typical QWERTY layout. Since the keys are so small, the devices incorporate intelligent prediction and tries to remedy when you happen to touch a neighbouring key.

There have been lot of research on different alternatives to inputting letters, from swiping over the keyboard so that you don't have to lift your finger [7, 26], to using different layouts for the fingers [1, 4, 12]. Some devices have support for handwriting recognition, either via a stylus or directly by your finger [3], and there are also research that incorporates haptic or auditory feedback so that you don't have to look at the screen [25]

Almost all research are focused on incremental left-to-right authoring of letters, possibly augmented with prediction and error correction. The usual way of editing text is to move the cursor and selecting existing words by means of arrow keys or different kinds of touch screen gestures [5].

Alternative and augmentative communication (AAC) Communication tools that are designed for people with communicative disabilities, such as Autism Spectrum Disorder (ASD) or Cerebral Palsy (CP), usually have words or symbols as the basic entity, instead of letters. The reasons for this is to decrease the number of interactions with the device, and to reduce the cognitive load for people with communicative or cognitive disabilities.

The most common AAC communication tool is a grid with different words and/or symbols, which the user can select in sequence to author a text. Selection can be done using pointing, or by switch access scanning [22], or even eye-tracking [13]. Some devices have prediction for suggesting the next word and/or error correction for changing the order between the words, and in some cases grammatical information is included so that the device can inflect the words by grammatical function such as number or gender [14, 15], and there has been recent work on allowing the user to input the words/symbols in a non-grammatical order, and the system automatically selects the most probable sequence [28].

Syntax-based text editing The idea of using grammars to guide text editing is not new [2, 17], but it has not been considered much in the scope of mobile devices or AAC tools. There have been some attemps at multilingual authoring on mobile devices [6, 21], but the work that has been done has either required the user to edit the abstract grammar term directly or it has focused on left-to-right authoring.

We have developed the basic theory of syntax editing that was described in section 1.2 [10, 11]. Currently there is a working online prototype which uses a simple non-heuristic search strategy to find new replacement suggestions.[1] The current theory depends on the grammar formalism GF [20], which has its roots in type theory. In [9] we also showed that type theory can be used as the basis of a theory of human-computer dialogue, which could prove useful since in this project we view the process of text editing as a dialogue between the user and the computer. To be able to find similar suggestions, we need a notion of similarity between grammatical trees [24].

3 Project description

In section 3.1 we give a detailed description of the current status of our editing theory [11]. Sections 3.2, 3.3 and 3.5 explains how the theory can be used and evaluated in different applications, and section 3.4 describes the work plan of the project.

3.1 A theory of semantic text editing

The system consists of three implementation layers. At the bottom is the grammar layer, in which the domain grammar is defined. The only requirements on the grammar formalism are that 1) it is able to distinguish grammatical and ungrammatical semantic trees, 2) it should have a way of transforming trees into sequences of surface words, and 3) each word should have a backpointer saying which node in the tree is responsible for introducing that word.

Any grammar formalism with these properties should be usable in our approach. We use the Grammatical Framework [19, 20], which in addition has good support for multilingual grammars and comes with a useful resource grammar API for more than 20 different languages [18]. In GF, the semantic trees

[1] Online prototype: http://www.grammaticalframework.org/~peter/grasp/

$$\begin{array}{rcl}
\mathit{cat} & : & \mathrm{N} \\
\mathit{mouse} & : & \mathrm{N} \\
\mathit{young} & : & \mathrm{N} \to \mathrm{N} \\
\mathit{cute} & : & \mathrm{N} \to \mathrm{N} \\
\mathit{a} & : & \mathrm{N} \to \mathrm{NP} \\
\mathit{all} & : & \mathrm{N} \to \mathrm{NP} \\
\mathit{sleep} & : & \mathrm{VP} \\
\mathit{chase} & : & \mathrm{NP} \to \mathrm{VP} \\
\mathit{sentence} & : & \mathrm{NP}, \mathrm{VP} \to \mathrm{S}
\end{array}$$

Figure 2: Example GF grammar

$$\begin{array}{rcl}
\mathit{cat}^\circ & = & \{\mathsf{sg} : \text{“}\mathit{cat}\text{”} ; \mathsf{pl} : \text{“}\mathit{cats}\text{”}\} \\
\mathit{mouse}^\circ & = & \{\mathsf{sg} : \text{“}\mathit{mouse}\text{”} ; \mathsf{pl} : \text{“}\mathit{mice}\text{”}\} \\
\mathit{young}^\circ(x) & = & \{\mathsf{sg} : \text{“}\mathit{young}\text{”} + x \,!\, \mathsf{sg} ; \mathsf{pl} : \text{“}\mathit{young}\text{”} + x \,!\, \mathsf{pl}\} \\
\mathit{cute}^\circ(x) & = & \{\mathsf{sg} : \text{“}\mathit{cute}\text{”} + x \,!\, \mathsf{sg} ; \mathsf{pl} : \text{“}\mathit{cute}\text{”} + x \,!\, \mathsf{pl}\} \\
\mathit{a}^\circ(x) & = & \{\mathsf{s} : \text{“}\mathit{a}\text{”} + x \,!\, \mathsf{sg} ; \mathsf{num} : \mathsf{sg}\} \\
\mathit{all}^\circ(x) & = & \{\mathsf{s} : \text{“}\mathit{all}\text{”} + x \,!\, \mathsf{pl} ; \mathsf{num} : \mathsf{pl}\} \\
\mathit{sleep}^\circ & = & \{\mathsf{sg} : \text{“}\mathit{sleeps}\text{”} ; \mathsf{pl} : \text{“}\mathit{sleep}\text{”}\} \\
\mathit{chase}^\circ(x) & = & \{\mathsf{sg} : \text{“}\mathit{chases}\text{”} + x \,!\, \mathsf{s} ; \mathsf{pl} : \text{“}\mathit{chase}\text{”} + x \,!\, \mathsf{s}\} \\
\mathit{sentence}^\circ(x, y) & = & \{\mathsf{s} : x \,!\, \mathsf{s} + y \,!(x \,!\, \mathsf{num})\}
\end{array}$$

Figure 3: Concrete syntax for the grammar

are called the *abstract syntax* and the surface words are called the *concrete syntax*. The translation from abstract syntax trees to concrete syntax words is called *linearisation*.

The middle layer consists of an API for modifying abstract syntax trees by specifying constraints on the tree and on its linearisation. The operations in the API tries to transform the given tree to obey the constraints, still keeping the new tree as semantically similar as possible. An example of a constraint can be that the linearisation of a given tree node must be different from the current linearisation.

The final layer is the graphical user interface, which communicates with the API to decide what alterantive suggestions should be displayed to the user. Internally, the text is *not* stored as the sequence of words that the user sees, but instead as semantic trees. The linearisation algorithm is used for displaying the sentences to the user, and everything that the user tries to do with a word is translated into a corresponding operation on the underlying tree.

3.1.1 Grammatical Framework

We start with some background on GF before we dwell into the the underlying editing theory on a more formal level.

GF abstract syntax The abstract syntax of a GF grammar consists of a finite number of typed functions, $f : A_1 \dots A_n \to A$ (where n can be 0). Given a function f we can create a tree $f(t_1 \dots t_n)$ of type A whenever $t_1, \dots, t_n$ are terms of types $A_1, \dots, A_n$, respectively. This makes the abstract syntax equivalent to a context-free grammar without terminal symbols, where the nonterminals correspond to GF types, and where the grammar rules have names. A simple example grammar is shown in Figure 2, and an example tree of type S licensed by this grammar is $\mathit{sentence}(\mathit{a}(\mathit{cute}(\mathit{cat})), \mathit{sleep})$.

GF concrete syntax The concrete syntax of a GF grammar is a compositional mapping from semantic trees to concrete terms, called the *linearisation*. The concrete terms can be quite complex and consist of strings, finite parameters, recursive records and inflection tables. However, we do not describe the concrete syntax in more detail, since it is not important for the later discussion. For more detailed information about the concrete syntax, we refer to the literature [19, 20] or to the GF online documentation.[2]

What is necessary to understand for the discussion in this section is that each word in a linearisation is introduced by exactly one node in the semantic tree. This makes it possible to translate an editing operation on a specific word into an editing operation on the corresponding node of the underlying tree. Note that GF does not make any distinction between leaf nodes and internal nodes, which means that any node in the tree can be responsible for a word.

Let's write t° for the linearisation of t. Compositionality can then be formulated as $f(t_1 \dots t_n)^\circ = f^\circ(t_1^\circ \dots t_n^\circ)$, where f° is the n-ary linearisation function corresponding to the n-ary abstract function f.

[2]Grammatical Framework: http://www.grammaticalframework.org/

The linearisation does not have to be a single string, but its result type can be different depending on the context. As an example, the linearisation of cat is an inflection table, which says that the corresponding strings should be *"cat"* in a singular context and *"cats"* in a plural context. The context is determined by the determiner, so the linearisation of a is a string that selects the singular value of its child, whereas $some$ specifies a plural child.

The concrete syntax of our example grammar is shown in Figure 3, and here are the linearisations of the example tree and a similar tree where the determiner a is replaced by all:

$$\begin{aligned} sentence(a(cute(cat)), sleep)^\circ &= \text{"}a\ cute\ cat\ sleeps\text{"} \\ sentence(all(cute(cat)), sleep)^\circ &= \text{"}all\ cute\ cats\ sleep\text{"} \end{aligned}$$

Note that the number of the determiner determines the inflection of both the noun and the verb, and that this inflection dependency is independent of the distance between the determiner and the verb. That the grammar formalism can handle long-distance dependencies is crucial if we want to write grammars for languages such as Swedish, German or Finnish.

3.1.2 Trees and tree editing

The *tree edit distance* is a distance measure between trees [24], which is a modification of the well-known Levenshtein string edit distance, where the distance between two trees is the number of editing operations required to transform one into the other. The allowed operations are 1) to *insert* a new node as a child of an existing node, 2) to *delete* a node, and 3) to *replace* an existing node with a new.

In our theory we use a variant of tree edit distance where nodes are ordered by semantic similarity, which is a notion that will be defined by the grammar. One possibility is to use Wordnet [16] for calculating the semantic similarity between two concepts.

Constrained linearisation In GF, not all strings in a linearisation of a subtree node have to be used in the linearisation of the full tree. In the example grammar, cat° contains two strings (*"cat"* and *"cats"*), but only one of them is used in $t_a = sentence(a(cute(cat)), sleep)^\circ$. We need to be able to talk about only the parts of a linearisation that are used, and for this purpose we define the *constrained linearisation* $[\![v]\!]_t$ of a subtree node v in a tree t. The intuition is that $[\![v]\!]_t$ consists of the strings in v° that are actually used when calculating t°. For the example tree t_a we get the constrained linearisations $[\![cat]\!]_{t_a} = \text{"}cat\text{"}$ and $[\![sleep]\!]_{t_a} = \text{"}sleeps\text{"}$.

Constraints for tree editing Each GF grammar rule $f : A_1 \ldots A_n \to A$ can be seen as a constraint on f-labeled nodes and its children. Checking that a tree is licensed by a grammar, which in GF is the same as checking that the tree is type-correct, can then be implemented as a constraint satisfaction problem [23]. Furthermore, when we formulate the grammar as constraints on trees, we can add additional constraints for specifying in more detail how our intended tree should look like.

By using tree constraints and a suitable semantic edit distance, we can describe a system for interactive tree editing. The system starts with a grammatical tree, and the user specifies additional constraints on the tree. The system then searches for the closest grammatical tree (in terms of semantic distance) that meets the constraints. This continues until the user is satisfied.

This approach lifts the level of tree editing from procedural to declarative: the user does not have to think about how to modify the tree, but instead she specifies what the tree should look like. We use two kinds of constraints:

- Structural constraints on the tree: We can specify whether a node should be or not be in the tree: $v \in V$ and $v \notin V$, respectively.
- Linearisation constraints: We can specify whether the (constrained) linearisation of a node shoule be or not be a given string: $[\![v]\!] = s$ and $[\![v]\!] \neq s$, respectively. There is also a linear precedence constraint: $[\![v]\!] \prec [\![v']\!]$, which means that $[\![v]\!]$ comes before $[\![v']\!]$ and that they are adjacent.

Example: Modifying a phrase The context-menu example shown in Figure 1, can be explained like this. Assume that we start with the following tree t_{a}:

$$t_{\mathsf{a}} = sentence(a(cute(cat)), sleep) \qquad t^{\circ}_{\mathsf{a}} = \text{“}a\ cute\ cat\ sleeps\text{”}$$

This tree has the nodes $sentence$, a, $cute$, cat and $sleep$. Now we want to say that the third word (whose corresponding node is cat) should be in plural form. This can be specified by the constraint $[\![cat]\!] = \text{“}cats\text{”}$. The system can then apply the tree editing operations to search for the most similar type-correct tree t'_{a} meeting the constraint. In this case it suffices to replace the determiner a by all:

$$t'_{\mathsf{a}} = sentence(all(cute(cat)), sleep) \qquad t'^{\circ}_{a} = \text{“}all\ cute\ cats\ sleep\text{”}$$

Semantic distance When the grammar is large, there might be several possible syntax trees that are equally close to the original tree (in terms of number of tree editing operations), making it difficult to decide which one would be a good alternative to suggest to the user. Our solution is to use a more fine-grained distance measure, where the cost of the editing operations depend on the specific values of the nodes that are involved .

If the example grammar contains the plural determiner $some$ in addition to all, there will be two possible solutions to the example problem. This is why we have to augment the grammar with a semantic distance measure between different abstract functions. In this case the grammar might state that $some$ is semantically more similar to a than all is, which therefore will suggest that the most similar resulting tree would be be $sentence(some(cute(cat)), sleep)$.

In a much the same way we can introduce similarity costs for deleting and inserting nodes; so that some functions prefer some other functions as parents, or siblings. This could be used for deciding which tree node a new phrase should attach to.

3.1.3 Syntactic editing of the surface string

Now we are ready to hide the semantic trees from the user altogether, and introduce semantic editing operations directly on the surface string. Our final goal is to implement a text editor where the user does not need any knowledge of formal grammars or semantic trees. Instead the text is presented to the user as a sequence of words, and in this section we define two intuitive editing operations on the text.

To implement these operations, we only make use of one GUI “gesture”, to *select* an object (other names are to *click* or *point*). The user can either select a word or the empty space between two words. This makes it possible to implement the operations in a very limited interface, such as switch access scanning [22].

Since the user only modifies the surface string, we need a way of translating surface editing operations onto the underying semantic tree. We use the fact that in GF, each surface word belongs to one and only one node in the tree. So, when the user makes a gesture on a word $w \in [\![v]\!]_t$, we interpret it as a gesture on the underlying node v.

Modifying or deleting a phrase When the user selects a word $w \in [\![v]\!]_t$, the system first highlights the whole phrase $s = [\![v]\!]_t$ (which can contain more words than w), and then displays an editing menu for that phrase. To calculate the menu, we search for similar trees t' satisfying the constraint $[\![v]\!] \neq s$, i.e., so that v is linearised differently from the current linearisation. For each of these trees, we create a menu item consisting of the constrained linearisation $[\![v]\!]_{t'}$. A special case is when the node v is deleted from the tree, which we handle in the same way as the empty linearisation $[\![v]\!]_{t'} = \epsilon$.

If there are no matching similar trees, we increase the selection by moving up to the parent node v'. Then we try again to find similar trees satisfying the constraint $[\![v']\!] \neq s$, and so on.

When the user selects a menu item, the current tree t is replaced by the new tree t'. The selected node v remains highlighted. The example in Figure 1 shows what happens when the user selects the menu item “$cats$” for the selected word “cat”.

Inserting a phrase If the user selects the space between two words $w_1 \in [\![v_1]\!]_t$ and $w_2 \in [\![v_2]\!]_t$, this is interpreted as the user wants to insert a phrase between the two nodes. We search for similar trees t' satisfying the constraints $[\![v_1]\!] \prec [\![v']\!] \prec [\![v_2]\!]$, where v' is the inserted node. Note that v' can be part of the original tree t, but does not have to. For each matching tree t', we create a menu item $[\![v']\!]_{t'}$. When the user selects a menu item, the tree t is replaced by t'.

3.2 Use cases

The following are two example use cases where multimodal text editing can prove useful. We plan to implement prototypes for both of them in the project. Some other possible use cases are discussed in section 3.5.

Mobile phrasebook translation aid The grammar formalism that we are using is multilingual, meaning that the underlying abstract syntax term is language-independent and can be linearized into several different languages, depending on the grammar that is used. Multilinguality can be incorporated into our text editing system, e.g., by showing linearisations of the abstract term in two different languages – the user's native language, and the language of the country that the user is visiting. The user can edit the sentence in her own language and all the time the sentence in the other language keeps updated. When the user is satisfied with the sentence, she can get the device to speak it out loud it in the other language and thus have a mobile communication tool between two languages. This communication tool will be more advanced than a traditional phrasebook, but it will still produce more correct translations than statistical machine translation (such as Google Translate).

AAC communication tool People with communicative disabilities, such as Autism Spectrum Disorder (ASD) or Cerebral Palsy (CP), often communicate via Augmentative and Alternative Communication (AAC) tools. Standardly these tools consist of a grid of input symbols which the user can select by pointing with a finger, or by using a switch or via eye gaze. The input symbols can be letters, or words, or even graphical symbols such as Blissymbolics, PCS or ARASAAC.

Most existing AAC communication tools are based on inputting words or symbols incrementally left-to-right, and it is often very cumbersome to edit something in the middle of an utterance, a task that our proposed editing system could make much easier. In addition, since the suggestions made by the system can be ordered by syntactic or semantic similarity, they will make more sense to the user and thus be a good alternative for people with cognitive disabilities. The multilingual grammar is also very useful in this case – the user can edit the text using graphical symbols, and the system will automatically edit an equivalent text in Swedish or English or whatever target language.

3.3 Evaluation

As we mentioned in the introduction, one of our main reasons for introducing the new approach to text editing is to be able to reduce cognitive load when authoring and editing text messages. When the prototypes are implemented we will perform rigorous evaluations, where we will investigate the efficiency and cognitive load when performing editing tasks, and compare with other editing interfaces.

These evaluations will be performed in collaboration with researchers experienced in evaluating cognitive load for dialogue systems, and with researchers experienced in evaluation of AAC tools (see section 6 for information about these collaborations).

3.4 Work plan

The project is budgeted for 5 years, with Peter Ljunglöf (PI) contributing 30% of his time, and there will be one PhD student that will work 80% during the whole project. A more detailed time plan follows here.

Year 1 During the first year we will further develop the algorithm for suggesting similar semantic trees so that it can work efficiently on large-sized grammars. We will also start developing a first prototype for the AAC communication tool described in section 3.2, together with an associated grammar that can translate between AAC symbols and Swedish.

Year 2 In the second year, a first version the AAC prototype will be finished and we will perform an initial evaluation on users. The evaluation will guide us on how the editing interface and the AAC grammar should be developed further. At the same time we will extend the grammar into more languages, such as the Scandinavian languages, Finnish, English, German, Spanish and others.

Year 3 We will integrate the AAC tool with other input methods, such as different ways of authoring the initial text that can be edited afterwards. This could be speech recognition or non-linear selection of AAC symbols from a grid of images [28].

We will improve upon the semantic similarity measure so that it can be learned automatically from existing lexica and corpora. At the same time we will start developing our second proof-of-concept, a multilingual phrasebook grammar that can be used by anyone by anyone with a mobile phone travelling abroad.

Year 4–5 Apart from refining the methods, interfaces and algorithms, we have time in the final years for an extensive evaluation of the multilingual phrasebook, and in particular for determining the efficiency and cognitive load, compared to alternative text authoring methods. Furthermore we will strenghten our collaboration with other research groups working on AAC communication, mobile translation or language learning.

3.5 Possible future use cases

In this project we only have the time to develop two proof-of-concept prototypes, one AAC tool and a mobile phrasebook translation app. Our methods have more potential uses, and here are two more possible useful tools which we won't pursue in this project.

Quick text messaging Most mobile phones have a possibility for sending quick text messages which you can use when you don't have the time to answer, or if you're busy with something else. These normally consists of a number of pre-stored messages that you cannot edit at all. By using keyboard-less text editing, you can quickly personalize the message by just a few screen interactions, thus closing the gap between canned text and writing full text messages from scratch.

The editing system does not have to be controlled by pointing, instead we could use a voice-controlled menu system such as the VoiceCursor [8]. Then the user would not have to look at a screen at all, which makes it useful in cognitively disabled situations such as when driving a car.

Foreign language learning Multilingual text editing can also be used for implementing a tool for learning a foreign language. The idea is that it will work as an interactive textbook, where the user can read different texts and also experiment with and modify the texts. The system will be divided into modules dealing with different linguistic features, e.g., inflection, simple phrases and more advanced constructions. The system would also be able to generate quizzes, e.g., by generating two slightly different sentences in the source and target languages, and let the user modify the target sentence so that it becomes a correct translation of the source sentence.

4 Significance

The ideas that we want to pursue in this project suggest a completely new way of thinking about text editing, as a two-dimensional process where any word can be edited any time, instead of a one-dimensional process of adding new words at the end of a text. This is ground-breaking and has the potential to make a difference in the everyday life of anyone who wants to communicate efficiently using multimodal input, be it a mobile phone or an AAC communication tool.

Compared to a touch-screen with a virtual keyboard of ≈30 characters taking up half of the screen, our proposed method will allow more room for displaying the text, and the touchable areas will be larger. This will make it easier for users to select the intended word, which in turn will reduce the congnitive load. Furthermore, the conceptual simplicity of the interaction will make it possible to use different input modalities, such as eye tracking or switch access scanning.

5 Preliminary results

GRASP ("Grammatikbaserad språkinlärning") was a small project financed by Sunnerdahls Handikappfond in 2010 where we developed the basic theory of syntax editing that was described in section 1.2 [10, 11]. There is a working online prototype which uses a simple non-heuristic search strategy to find new replacement suggestion.[3] We are currently fine-tuning the prototype and will demonstrate it on *Vetenskapsfestivalen* here in Gothenburg 8 May 2014.

6 Collaboration

CLT (Centre for Language Technology) is an organization for collaboration between language technology researchers at the at the University of Gothenburg and Chalmers University of Technology. We will work in close collaboration with the developers of Grammatical Framework (Aarne Ranta, Thomas Hallgren and other people in the Grammar Technology Lab at CLT) when we develop the grammars and algorithms for the project, by building upon existing grammars and algorithms.

The VR project REMU[4] is a project for reliable multilingual communication, which also builds upon GF technology. Our project is independent from and has a different purpose than REMU, where we focus on the interaction and cognitive load of text editing, while REMU focuses on making grammars more reliable by using formal methods and logical reasoning. There are possible synergy effects between the projects, which we of course will make use.

The Dialogue Technology Lab at CLT has long experience on developing and evaluating dialogue systems (Staffan Larsson, Simon Dobnik and others), and in particular they have experience on how to reduce and evaluate cognitive load. The user evaluations will be conducted in collaboration with them.

DART (Centre for Augmentative and Alternative Communication and Assistive Technology, Sahlgrenska University Hospital, Gothenburg) is the leading centre for alternative and augmentative communication in Western Sweden, and has long experience of working with people with communication difficulties, as well as conducting research on AAC and accessive technologies. Peter Ljunglöf has collaborated with DART in several projects about using language technology in AAC tools (Katarina Heimann Mühlenbock, Mats Lundälv and others). We will collaborate with DART in the design and evaluation of the AAC prototype tool.

7 Ethical considerations

There are always ethical considerations when conducting user evaluations. Ethics is even more important when it comes to people with disabilities, since they are relatively few and in a vulnerable position. Therefore it is customary in Sweden that all research on disabled people should be approved by an independent ethics committee. Our collaborators DART has long experience in designing user evaluations and writing ethics applications, which we of course will take advantage of.

8 References

[1] Shiri Azenkot, Jacob O. Wobbrock, Sanjana Prasain, and Richard E. Ladner. Input finger detection for nonvisual touch screen text entry in Perkinput. In *Proceedings of GI'12: 38th Graphics Interface Conference*, Toronto, Canada, 2012.

[2] Marc Dymetman, Veronica Lux, and Aarne Ranta. XML and multilingual document authoring: Convergent trends. In *COLING'00, 18th International Conference on Computational Linguistics*, pages 243–249, Saarbrücken, Germany, 2000.

[3] Michael D. Fleetwood, Michael D. Byrne, Peter Centgraf, Karin Q. Dudziak, Brian Lin, and Dmitryi Mogilev. An evaluation of text-entry in Palm OS – Graffiti and the virtual keyboard. In *Proceedings of the HFES 46th Annual Meeting*, Santa Monica, California, 2002.

[3]Online prototype: http://www.grammaticalframework.org/∼peter/grasp/

[4]REMU (Reliable Multilingual Digital Communication: Methods and Applications), Project within VR "Rambidrag: Det digitaliserade samhället - igår, idag, imorgon", 2013–2017.

[4] Brian Frey, Caleb Southern, and Mario Romero. BrailleTouch: mobile texting for the visually impaired. In *Proceedings of UAHCI'11: 6th International Conference on Universal Access in Human-Computer Interaction: Context Diversity*, volume III, pages 19–25, Orlando, Florida, 2011.

[5] Vittorio Fuccella, Poika Isokoski, and Benoît Martin. Gestures and widgets: Performance in text editing on multi-touch capable mobile devices. In *Proceedings of CHI 2013: The ACM SIGCHI Conference on Human Factors in Computing Systems*, Paris, France, 2013.

[6] Janna Khegai, Bengt Nordström, and Aarne Ranta. Multilingual syntax editing in GF. In A. Gelbukh, editor, *CICLing–2003: Intelligent Text Processing and Computational Linguistics*, LNCS 2588, pages 453–464. Springer-Verlag, 2003.

[7] Per Ola Kristensson and Shumin Zhai. Command strokes with and without preview: Using pen gestures on keyboard for command selection. In *Proceedings of CHI 2007: The ACM SIGCHI Conference on Human Factors in Computing Systems*, San Jose, California, 2007.

[8] Staffan Larsson, Alexander Berman, and Jessica Villing. Multimodal menu-based dialogue with speech cursor in DICO II+. In *Proceedings of ACL-HLT'11: 49th Annual Meeting of the Association for Computational Linguistics, Systems Demonstrations*, Portland, Oregon, 2011.

[9] Peter Ljunglöf. Dialogue management as interactive tree building. In *DiaHolmia'09, 13th Workshop on the Semantics and Pragmatics of Dialogue*, Stockholm, Sweden, 2009.

[10] Peter Ljunglöf. GRASP: Grammar-based language learning. In *SLTC'10, 3rd Swedish Language Technology Conference*, Linköping, Sweden, 2010.

[11] Peter Ljunglöf. Editing syntax trees on the surface. In *Nodalida'11: 18th Nordic Conference of Computational Linguistics*, Rīga, Latvia, 2011.

[12] I. Scott MacKenzie and R. William Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17(2–3):147–198, 2002.

[13] Päivi Majaranta and Mick Donegan. Introduction to gaze interaction. In Päivi Majaranta, Hirotaka Aoki, Mick Donegan, Dan Witzner Hansen, John Paulin Hansen, Aulikki Hyrskykari, and Kari-Jouko Räihä, editors, *Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies*. IGI Global, 2012.

[14] Kathleen F. McCoy. Simple NLP techniques for expanding telegraphic sentences. In *Proceedings of the EACL'97 Workshop on Natural Language Processing for Communication Aids*, Madrid, Spain, 1997.

[15] Kathleen F. McCoy, Christopher A. Pennington, and Arlene Luberoff Badman. Compansion: From research prototype to practical integration. *Natural Language Engineering*, 4(1):73–95, 1998.

[16] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[17] R. Power and D. Scott. Multilingual authoring using feedback texts. In *Proceedings of COLING-ACL'98: 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, 1998.

[18] Aarne Ranta. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2(2), 2009.

[19] Aarne Ranta. Grammatical Framework: A multilingual grammar formalism. *Language and Linguistics Compass*, 3(5):1242–1265, 2009.

[20] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

[21] Aarne Ranta, Ramona Enache, and Grégoire Détrez. Controlled language for everyday use: the MOLTO phrasebook. In *Proceedings of CNL'10: 2nd Workshop on Controlled Natural Language*, Marettimo Island, Italy, 2010.

[22] Brian Roark, Jacques de Villiers, Christopher Gibbons, and Melanie Fried-Oken. Scanning methods and language modeling for binary switch typing. In *Proceedings of SLPAT'10: 1st Workshop on Speech and Language Processing for Assistive Technologies*, Los Angeles, California, 2010.

[23] Martin Sulzmann and Peter J. Stuckey. HM(X) type inference is CLP(X) solving. *Journal of Functional Programming*, 18(2):251–283, 2008.

[24] Kuo-Chung Tai. The tree-to-tree correction problem. *JACM, Journal of the Association for Computing Machinery*, 26:422–433, 1979.

[25] Hussain Tinwala and I. Scott MacKenzie. Eyes-free text entry on a touchscreen phone. In *Proceedings of TIC-STH: Science and Technology for Humanity*, Toronto, Canada, 2009.

[26] David J. Ward, Alan F. Blackwell, and David J. C. MacKay. Dasher: A gesture-driven data entry interface for mobile computing. *Human-Computer Interaction*, 17(2–3):199–228, 2002.

[27] Karl Whitfield. Mobile messaging markets: January 2014. Technical report, Portio Research Ltd, 2014.

[28] Karl Wiegand and Rupal Patel. SymbolPath: a continuous motion overlay module for icon-based assistive communication. In *Proceedings of the ASSETS'12: 14th international ACM SIGACCESS Conference on Computers and Accessibility*, New York, USA, 2012.