Development of an Augmentative and Alternative Communication App for the Maltese Language

Sylvan Abela, Owen Casha, May Agius
owen.casha@um.edu.mt

Abstract

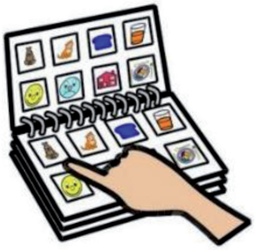
Augmentative and Alternative Communication (AAC) embodies all methods of communication, serving to augment or function as an alternative to speech. Maltese children having complex communication needs use various AAC devices on a daily basis. Their conversation skills are mainly limited by two key factors. The fact that AAC users communicate up to 20 times slower than those who use regular speech is the first of these two limiting factors. The second one is the unavailability of an AAC app for the Maltese language. This paper presents the development of an AAC app targeted for the Maltese language, which provides an intelligent word suggestion mechanism to improve AAC rates. The app is based on a trigram language model which is able to predict the subsequent word by considering the previous two. The model was trained by means of a specifically created corpus and uses the Interpolated Kneser-Ney Smoothing technique to correctly resolve contexts which were not observed during training. The app enables users to retrain and update the language model, such that it may provide additional personalised word suggestions. The app was evaluated by a number of clinicians and educators who regularly work with AAC users. They remarked that it will be potentially helpful in aiding Maltese children during intervention sessions in view of its effective features. The underlying language model features an average perplexity of 90.47 when tested with non-similar training and test data and an average perplexity of 3.61 when evaluated for highly similar training and test data. The low perplexity values suggest that the language model employed in this app is remarkably accurate and is effectively performing as other trigram language models reported in the literature.

Keywords: AAC, Natural Language Processing, *n*-grams, Statistical Language Modelling, App Development

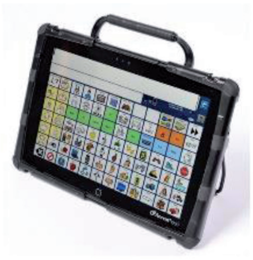
Introduction

For most people, the process of converting thoughts into words and making the necessary muscle movements to speak them out loud occurs naturally. However, there are some individuals who are either unable to speak or whose speech in unintelligible. This creates the need for an AAC system. AAC is the term used to refer to any method of communication serving to augment or as an alternative to speech (Glennen and DeCoste, 1997). Common examples of AAC systems include unaided systems which do not require any equipment, such as gestures and facial expressions, and aided systems including communication boards and speech generating devices (SGD) containing pictograms that correspond to different words (Agius and Vance, 2015; Borg et al., 2015). Aided AAC systems are in turn categorised into low and high tech systems. Low tech AAC systems enable the user to communicate by pointing to pictures on a picture board or in a communication book, as shown in **Figure 1a**, while high technology systems refer to SGDs which produce spoken words in response to the user touching the computer screen (**Figure 1b**).

The main limitation of AAC systems is the communication rate. While most people communicate their thoughts effortlessly using speech, AAC users must type letters in a text box or choose pictures from a display to construct sentences. In practice, AAC users have communication rates which are below 10 words per minute, compared to speech rates which may range between 130 and 200 words per minute (Trnka et al., 2009). Communication rates in high tech AAC systems can, however, be improved significantly by implementing next word prediction (Trnka et al., 2009; Garay-Vitoria and Abascal, 2006; Langer and Hickey, 1999).



(a) Communication Book (Garay-Vitoria and Abascal, 2006)



(b) Speech Generating Device (SGD) (Langer and Hickey, 1999)

Figure 1. Low technology and high technology AAC systems.

The Access to Communication and Technology Unit (ACTU) is an agency responsible for assessing children for AAC solutions and providing AAC interventions in Malta (Agius and Vance, 2015). Emerging studies show that children require more effort to use a low technology AAC system as opposed to a smart SGD such as a tablet (Borg et al., 2015). Therefore, in order to facilitate AAC interventions, the agency also recommends high technology AAC systems. Unfortunately, AAC technologies are still very limited in Malta, as no AAC app is currently available for the Maltese language. As an alternative, the ACTU customises existing AAC apps which were designed for the English language. This is inadequate, since the structure of the English language is very dissimilar to that of Maltese. In addition, this approach is limited, as only very basic Maltese structures can be created. Furthermore, customisation to the Maltese language is extremely time consuming. Due to these issues, Maltese children have, currently, no other option but to use the English language for communication purposes. The aim of this project is to develop a mobile app (MaltAAC), with the intention of breaking the barrier between the Maltese language and high tech AAC systems. The app was developed to:

i) allow Maltese children with speech, motor, or cognitive impairments to communicate in their native language;
ii) reduce the amount of effort they require to communicate their message by improving their communication rates when making use of AAC.

The app features a user interface (UI) which is similar to that of most high technology AAC systems, and an initial dictionary of around 400 Maltese words which are commonly used by children to express their needs. The central UI component is the image grid, which may display categories, words, or predictions at any point in the life-cycle of the app (**Figure 2**). The app also includes a smoothed trigram language model which generates a list of the most probable next word each time the user selects a new word, thus improving a child's communication rate and reducing the amount of effort required to exchange information. MaltAAC includes a mechanism which enables the user to add new words, delete unused words, change images and sounds, change the category to which a particular word pertains or change the word itself.

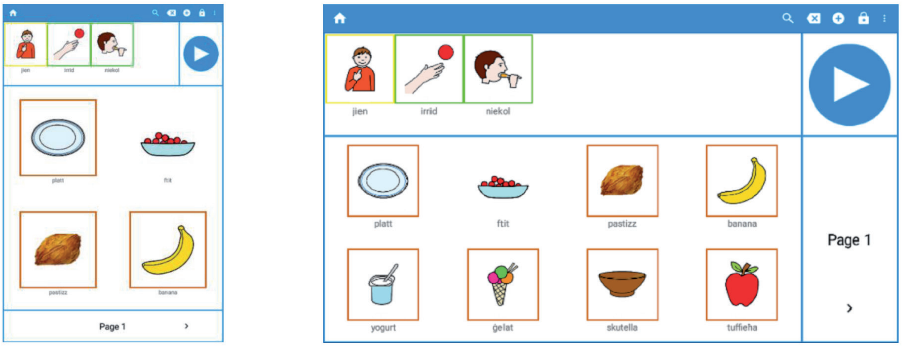


Figure 2. Central user interface of the developed AAC.

A search icon is available on the action bar at the top of the screen. When this icon is tapped, the search bar is expanded and the on-screen keyboard is displayed to the user. One can search for words using the keyboard, and word suggestions are shown based on the characters which have already been typed. MaltAAC provides the option to set a security pin. If this pin is set, then an incorrect pin entry would deny access to settings and certain features of the app, such as profile backup/restore, screen un/pinning, and updating predictions (Abela, 2018).

Background

Natural Language Processing (NLP) is a branch of Artificial Intelligence in which natural languages are studied from a computing perspective to enable human beings to have flowing language interactions with computers (Kumar, 2011). Next word prediction in NLP is concerned with predicting the next word in a sentence given a sequence of words. This can be achieved by making use of a statistical language model which estimates the probabilities of all the possible words that can follow a given context. In next word prediction applications, training a language model involves using an algorithm to calculate next word probabilities based on what is observed in the training data (Jurafsky and Martin, 2009). The collection of sentences on which a language model is trained is called the training corpus, which is a collection of written words that are gathered based on specific criteria (Manning and Schütze, 1999). In order for the language model to provide satisfactory results, this collection of sentences should be taken from the same genre of words which is likely to be encountered in the NLP application (Jurafsky and Martin, 2009). In this work, a new corpus was created to train the language model on Maltese sentences which are commonly used by children to express their needs, thus maximising the accuracy of the word suggestions provided by the app.

Rosenfeld (2000) surveyed the various language models that can be used for word prediction applications, namely the *n*-gram language model, the context-free grammar model, decision tree models and maximum entropy models, among others. Subsequently, Goodman (2001) compared the accuracy of these models and found that the *n*-gram language model has the best performance. While some of the other models result in more accurate predictions than the *n*-gram model, this comes at the expense of much greater time and space complexity, making them impractical to use in mobile applications. An *n*-gram model is a type of probabilistic language model for predicting the next item in a sequence in the form of an $n-1$ order Markov model. Two benefits of *n*-gram models are simplicity and scalability – with a larger order *n*, a model can store more context with a well-understood space–time trade-off, enabling an efficient scale up. In practice, it is not necessary to consider the entire sequence of words to determine the probability of the next word. Thus, in an *n*-gram language model, the probability of the next word is approximated by limiting the context for conditional probability to the preceding $n-1$ words (Jurafsky and Martin, 2009). The probability of each *n*-gram (sequence of *n* words) is calculated during the language model training. The simplest method to calculate *n*-gram probabilities is the Maximum Likelihood Estimation (MLE) (Abela, 2018).

A language model trained using MLE incorrectly assigns a probability of zero to sequences of words which are never observed during training. In order to improve the probabilities of *n*-grams having zero or low-frequency counts in the training corpus, smoothing must be implemented when training the model. In statistical language modelling, smoothing refers to adjusting the MLE probabilities to produce more accurate probabilities for word sequences which rarely or never occur in the training corpus (Chen and Goodman, 1999). This is achieved by reducing some of the probability mass from *n*-grams with large counts in the training corpus and redistributing it to *n*-grams with zero or low-frequency counts, resulting in a smoother probability distribution. This is known as discounting (Jurafsky and Martin, 2009; Manning and Schütze, 1999). Various smoothing methods have been proposed to improve the accuracy of *n*-gram probabilities, namely Laplace smoothing, Good-Turing discounting, Jelinek-Mercer smoothing, Katz smoothing, Witten-Bell smoothing, absolute discounting, and Kneser-Ney smoothing. These smoothing techniques have been compared and evaluated by Chen and Goodman (1999) and Kneser-Ney smoothing was found to be the most accurate method. This is why Kneser-Ney smoothing was adopted in this work (Abela, 2018).

Design Considerations

The top two mobile operating systems for smart devices are Android and iOS, with market shares greater than 70% and 25% respectively (StatCounter, 2018). In order to create a widely accessible AAC tool, the proposed app was targeted for Android, since it has the largest mobile operating system market share and since Android devices are generally cheaper than their Apple counterparts. This AAC app was developed using a combination of Java and Extensible Markup Language (XML). XML was used to create the outline of the app's graphical user interface (GUI) and Java was used to create the app's functional components and make programmatic changes to the GUI. Java was chosen as the programming language since it is an official language of the Android platform and therefore all Android operating system features can be accessed through application programming interfaces (APIs) written in Java (AndroidDevelopers, 2018a).

Storage

Database Selection

The need for a storage method that can archive categories and words together with their relationship was identified. An intuitive way to model and store related pieces of information is to use a relational database (Halpin and Morgan, 2008). Since the app is targeted for mobile devices, a set of criteria had to be met when deciding on which database to use. The database had to provide a lightweight solution, since storage space is limited on most mobile devices. Moreover, it had to provide server-less functionality, since internet connectivity is not always available. Lastly, the selected database had to provide a very fast access, with very low memory and power consumption. Several database options exist for the Android operating system, namely BerkeleyDB, CouchbaseLite, LevelDB, ORMLite, Realm, and SQLite. Most of these databases fulfil the specified criteria, but SQLite was the most ideal solution for this work since it is the only database which is fully supported by Android (AndroidDevelopers, 2018b).

Storage of Media Files

Initially, storing media files as binary large objects (BLOBs) in the database looked promising, since it was the best way to keep the words, images and sounds consolidated in one structure (Shapiro, 1999). However, during the testing phase of the app, it became evident that this approach had to process excessively large amounts of data, resulting in slower response times. Indeed, Shapiro (1999) found that the query response time is inversely proportional to the size of the database. To improve the app's performance and usability, the image and audio files were stored in the app folder and accessed directly from the internal storage. As a result, query response times became considerably shorter and independent of the image and audio file sizes.

Storage of User Preferences

While storing user preferences as records in a relational database was possible, it was more feasible to use the *SharedPreferences* interface from the Android software development kit (SDK). Primarily, storing or retrieving a user preference can be achieved in a simpler way by calling the required method provided by the *SharedPreferences* interface, as opposed to constructing structured query language (SQL) queries. Secondly, the *SharedPreferences* interface works in tandem with the *PreferenceActivity* class from the Android SDK. By inheriting this class, it was possible to create an activity which displays a hierarchy of preferences that can be set by the user. These preferences are stored as key-value pairs of primitive data types (Wei, 2012) and are used to determine the app's behaviour in different scenarios, according to the user's preference.

Performance Considerations

Multithreading

Accessing a file from internal storage is a rather slow operation compared to, say, fetching a message from a *String* variable and displaying it on screen, since secondary storage has a much slower access speed than main memory (Hamacher et al., 2012**)**. If such a lengthy operation were to be carried out on the same thread which is responsible for generating the UI (known as the main thread or UI thread), then any other operations would be blocked until it completes, hence causing the UI to hang. Thus, lengthy operations are carried out on separate threads.

Foreground Services

Users can continue using their device while backing up or restoring a profile, since the cloud backup and restore operations were implemented as foreground services (MacLean, Komatineni and Allen, 2015). MaltAAC gives a foreground notification while uploading the backup file to the cloud, which displays the progress as a percentage, and provides the option to cancel the upload. Similarly, a foreground notification displaying progress is shown while restoring a user profile.

Data Structure Selection

An appropriate data structure was required to store *n*-grams in main memory together with the occurrence frequency of each *n*-gram in the training corpus. The main criteria for selecting the data structure were insertion time, searching time and memory consumption. Robenek, Platos and Snasel (2013) compared the performance of seven data structures for indexing *n*-grams, and the *HashMap* data structure was found to be the fastest solution in terms of insertion time and searching time. The *HashMap* data structure ranked fourth in terms of memory consumption, but since training speed was a priority for MaltAAC, the *HashMap* structure was still the most practical option.

Optimal Order of n-gram

Initially, MaltAAC was based on a 5-gram language model that needed an excessively long training time. Goodman (2001) tested the performance of *n*-gram models up to an order of $n = 20$ and found that the performance of Kneser-Ney smoothing saturates at the order of $n = 3$. Since training speed was a priority for MaltAAC, and little performance improvements could be obtained by training a model of order $n = 5$, it was decided to opt for a trigram language model instead.

Results and Evaluation

Perplexity

A language model can either be evaluated extrinsically, i.e. embedding the model into the application and measuring end-to-end performance, or intrinsically, where the performance of the model is measured independently of the application. The most common metric for intrinsic evaluation of *n*-gram models is perplexity (Jurafsky and Martin, 2009). Perplexity is used to measure how effectively a language model can predict the next word, where a low value indicates a high model accuracy. The calculation of the perplexity of a language model for a given test data set was implemented in Java. The program uses the *HashMaps* containing the *n*-gram probabilities to calculate the conditional probability of each word in the test set. The developed language model was trained and tested on 10 different pairs of training and on test data sets which were randomly generated from a corpus specifically created for this project. An average perplexity of 3.61 was obtained (refer to **Table I**).

A perplexity value of *k* suggests that on average, the language model is as uncertain as it would be if it were to predict the next word out of *k* equally probable words (Manning and Schütze, 1999). The obtained results were remarkably low due to the high similarity between the training and test data sets. Children with complex communication needs often tend to use the same simple sentence structure (Binger and Light, 2008) and so most sentences formed using MaltAAC will have the same syntax as the ones used to train the language model, which is essentially equivalent to having identical training and test data sets. Hence, in spite of its artificially low value, the resulting average perplexity implies that the language model is able to predict the next word in a given sequence by choosing one out of four equally probable words. Since the initial dictionary contains 400 distinct words, the language model is capable of narrowing the list of possible next words to 1% of the initial dictionary size. This performance is quite satisfactory and results in reasonably accurate word predictions.

Test Number	Perplexity
1	3.615
2	3.636
3	3.582
4	3.631
5	3.635
6	3.556
7	3.562
8	3.517
9	3.612
10	3.776
Average	**3.6122**

Table I. The language model's average perplexity for training and test data sets taken from the created corpus.

Table II presents the perplexity results obtained by the language model when trained and tested on data sets taken from the same genre, but which were not as similar as the training and test data sets generated from the created corpus. The training and test data were taken from the same children's book, so as to maintain the same style of vocabulary. An average perplexity value of 90.47 was obtained.

Book	Number of Training Words	Number of Test Words	Perplexity
Roses and Forget-Me-Nots	1,925	240	76.048
Marjorie's Three Gifts	3,169	376	90.88
Jack and Jill	47,286	504	77.171
The Lost Prince	7,649	849	89.197
The Secret Garden	11,879	1,409	119.047
		Average	**90.4686**

Table II. The language model's average perplexity for training and test data sets of the same genre.

This is marginally lower than typical perplexity values reported in literature. Indeed, with a vocabulary size of 19,979 words, Jurafsky and Martin (2009) obtained a perplexity of 109 for a trigram model which was trained on 38 million words taken from the Wall Street Journal and tested on 1.5 million words, whereas Kneser and Ney (1995) obtained a perplexity of 105.4 for a trigram model which was trained and tested on data sets generated from the 30,000 word German Verbmobil corpus, with a vocabulary of 2,000 words. The improved perplexity value does not necessarily imply that the developed language model is any better than other trigram models, but merely suggests that sentences in children's books are typically more uniform and easier to predict than newspaper contents. Nonetheless, the obtained perplexity results show that the developed language model can perform at least as well as other trigram language models, when trained and tested on data sets taken from the same genre.

Book	Perplexity
Roses and Forget-Me-Nots	117.716
Marjorie's Three Gifts	128.060
Jack and Jill	120.720
The Lost Prince	120.792
The Secret Garden	162.704
Average	**129.998**

Table III. The language model's average perplexity for training and test data sets of different genres.

Table III shows the perplexity results of the developed language model when trained on data taken from the technology genre and tested on sentences from children's books. The average perplexity was 43.7% higher than the average perplexity acquired for training and test data sets taken from the same genre. This increase in perplexity is quite significant and it confirms that the language model's accuracy is highly dependent on the genre of the training and test data sets. Especially when compared to the average perplexity value presented in **Table I**, this result proves that creating a training corpus which was tailor-made for MaltAAC was indeed more practical than using generic training data.

Time Complexity

Since MaltAAC provides the option to update the word predictions by retraining the language model, training time minimisation was key. However, as the number of words in the training corpus increases, so does the number of possible *n*-grams and the number of iterations in each *HashMap*. In order to assess how the training time varies as the number of *n*-grams increases, the language model was trained on different sized data sets obtained from the Gutenberg corpus. The plots presented in **Figure 3** show that the training time follows a cubic trend when plotted against the number of *n*-grams. A time complexity of $O(n^3)$ for the training process falls short of being satisfactory. While the number of *n*-grams in this AAC app may increase if the user chooses to add new words and retrain the language model, it is not likely to grow to a point where the training time becomes excessively long, since the number of words used by children is rather limited. Thus, in contrast to other NLP applications, the language model training time is tolerable for the purposes of MaltAAC.

App Functionality

White-box testing was useful to identify, debug and solve certain issues such as memory leaks, out of memory exceptions caused by the loading of large bitmaps and audio files into main memory, security exceptions due to missing permission requests, unpredictable UI behaviour, and flawed input control mechanisms. The app was given to seven professionals who work with AAC systems on a daily basis for black-box testing. Among these professionals were three occupational therapists, two speech and language therapists and two learning support educators who work at the ACTU. This enabled them to test MaltAAC's functionality from a user's perspective, identify any bugs which had been overlooked, and determine whether the app is suitable for AAC purposes. MaltAAC was tested for a month, after which a focus group was then formed to obtain feedback from the testers. The black-box testing stage verified that the app satisfies the initial requirements, although there is room for improvement. The suggested potential enhancements were taken into consideration for integration in future releases of MaltAAC.

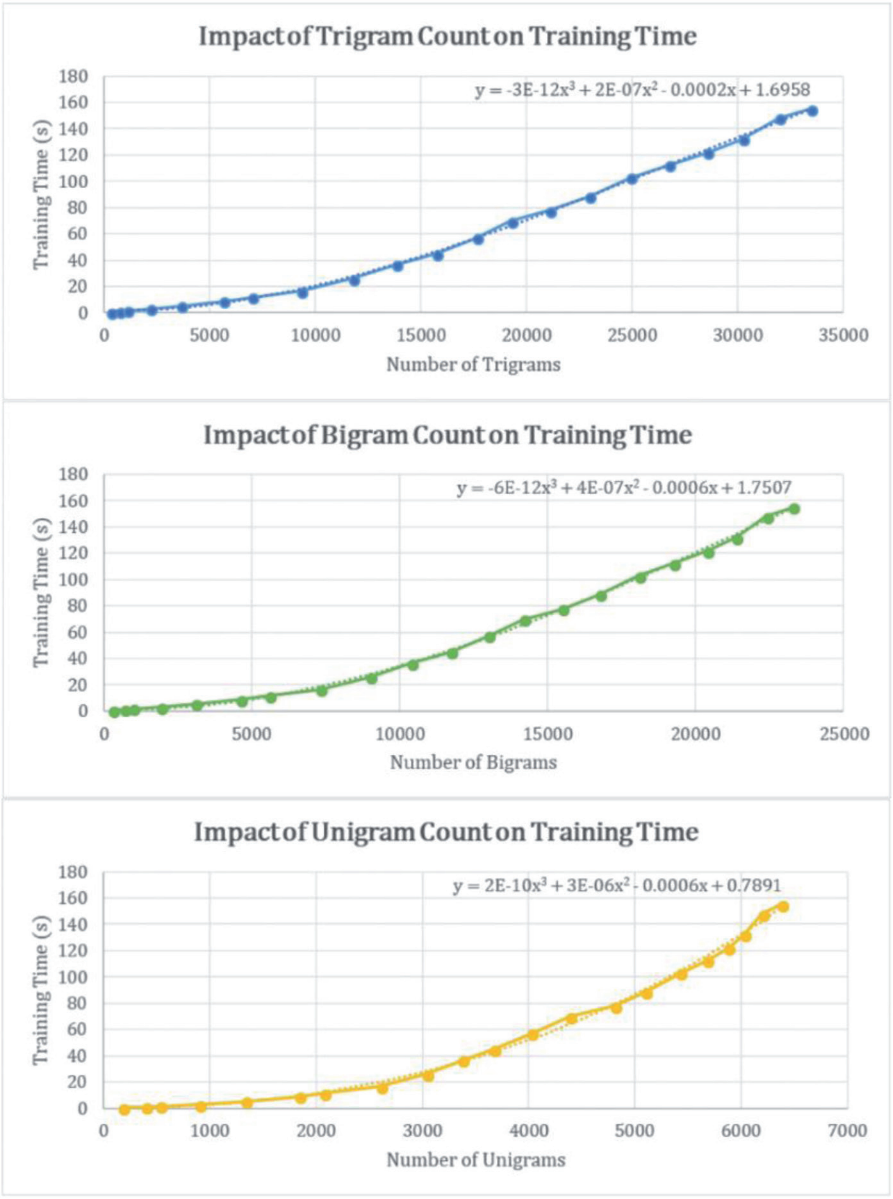


Figure 3. The impact of increasing the number of n-grams on the training time.

Conclusion

This paper presented the development of an app to provide an AAC solution for the Maltese language. MaltAAC features a smoothed trigram language model which predicts the next word given the sequence of previously selected words, with the aim of improving children's communication rates while reducing the effort required to convey their message. Interpolated Kneser-Ney smoothing was used to train the language model and calculate the *n*-gram probabilities. The accuracy of the developed trigram model was evaluated using the perplexity metric, and the app functionality was assessed using white-box and black-box testing. The generated word predictions were found to be remarkably accurate, especially when the language model was trained on a corpus specifically created for this AAC app.

MaltAAC was tested by specialists who are well acquainted with AAC technologies. These specialists commended MaltAAC, saying that it is a very helpful AAC tool. One minor limitation of the app is the substandard time complexity of the language model's training process, although its impacts are somewhat insignificant, especially for a small vocabulary size. On the whole, this app was completed successfully and may greatly improve the quality of AAC interventions for Maltese children. Nevertheless, there are still numerous improvements which can be made in future releases of MaltAAC, in order to design an even more convenient and complete AAC solution for the Maltese language, including an Android text-to-speech engine for Maltese.

MaltAAC is a solid step forward towards promoting the use of the Maltese language while providing an accessible solution to Maltese children having complex communication needs. In addition to assisting native speakers, MaltAAC can potentially be taken a step further and serve as a linguistic first aid to immigrants coming over to Malta, in order to facilitate their communication and integration with Maltese citizens. MaltAAC is currently available on the Google Play Store and there are plans to initiate the work on an iOS implementation of the app.

References

Abela, S. (2018). *Development of an Augmentative and Alternative Communication App for the Maltese Language*. Msida: Unpublished B.Sc. Dissertation, University of Malta.

AndroidDevelopers (2018a). Platform architecture. Available at https://developer.android.com/guide/platform/ [Accessed 1 February 2018].

AndroidDevelopers (2018b). Data and file storage overview. Available at https://developer.android.com/guide/topics/data/data-storage.html [Accessed 1 February 2018].

Agius, M.M. and Vance M. (2015). A Comparison of PECS and iPAD to teach requesting to Pre-schoolers with Autistic Spectrum Disorders. *Augmentative and Alternative Communication*, 32(11), 58–68.

Binger, C. and Light, J. (2008). The Morphology and Syntax of Individuals who use AAC: Research Review and Implications for Effective Practice. *Augmentative and Alternative Communication*, 24(2), 123–138.

Borg S., Agius M. and Agius L. (2015). A User and their Family's Oerspective of the use of a Low-tech vs a High-tech AAC System," *Studies in Health Technology and Informatics*, 217(11), 811–818.

Chen, S.F. and Goodman, J. (1999). An Empirical Study of Smoothing Techniques for Language Modelling. *Computer Speech and Language*. 13(10), 359–394.

Garay-Vitoria, N. and Abascal, J. (2006). Text Prediction Systems: A Survey. *Universal Access in the Information Society*, 4(3), 188–203.

Glennen, S.L. and DeCoste, D.C. (1997). *The Handbook of Augmentative and Alternative Communication*. San Diego: Singular Pub. Group.

Goodman, J. (2001). A Bit of Progress in Language Modeling. Computer Speech and Language, 15(10), 403–434.

Halpin, T. and Morgan, T. (2008). *Information Modeling and Relational Databases*. Morgan Kaufmann.

Hamacher, C., Vranesic, Z., Zaky, S. and Manjikian, N. (2012). *Computer Organization and Embedded Systems*. McGraw-Hill.

Jurafsky, D. and Martin, J.H. (2009). *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ, USA: Prentice Hall.

Kneser, R. and Ney, H. (1995). Improved Backing-off for m-gram Language Modeling," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 181–184.

Kumar, E. (2011). *Natural Language Processing*. I.K. International Publishing House PVT.

Langer, S. and Hickey, M. (1999). Augmentative and Alternative Communication and Natural Language Processing: Current research activities and prospects. *Augmentative and Alternative Communication*. 15(12), 260–268.

MacLean, D., Komatineni, S. and Allen, G. (2015). *Pro Android 5*. Apress.

Manning, C.D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Robenek, D., Platos, J. and Snasel, V. (2013). Efficient in-Memory Data Structures for n-grams Indexing. *Proceedings of the Annual International Workshop on Databases, Texts, Specifications and Objects*, 48–58.

Rosenfeld, R. (2000). Two Decades of Statistical Language Modeling: Where do we go from here? *Proceedings of the IEEE*. 88(8), 1270–1278.

Shapiro, M. (1999). Managing Databases with Binary Large Objects. *16th IEEE Symposium on Mass Storage Systems in cooperation with the 7th NASA Goddard Conference on Mass Storage Systems and Technologies* (Cat. No.99CB37098), 185–193.

StatCounter (2018). Statcounter Globalstats. Available at http://gs.statcounter.com/os-market-share/mobile/malta [Accessed 26 August 2018].

Trnka, K., McCaw, J., Yarrington, D., McCoy, K.F. and Pennington, C. (2009). User Interaction with Word Prediction: The effects of prediction quality. *ACM Transactions on Accessible Computing (TACCESS)*. 1(3), 17:1–17:34.

Wei, J. (2012). *Android Database Programming*. Packt Publishing.

Bio-notes

Mr Sylvan Abela has recently completed a bachelor’s degree in Computer Engineering at the Faculty of ICT within the University of Malta. As part of his final year project, he developed the first AAC app targeted for the Maltese language in collaboration with ACTU, with the intention to break the barrier between the Maltese language and high technology AAC systems. His interests include software development, microcontroller-based system design and artificial intelligence, particularly natural language processing and robot learning.

Dr Ing. Owen Casha received his bachelor’s degree in electrical engineering from the University of Malta in 2004. From September 2007 till June 2008, he was on a research collaboration with CEA-LETI (Grenoble, France) and ST-Microelectronics. He received a PhD in RFIC design from the University of Malta in 2010. He is currently a Senior Lecturer with the Department of Microelectronics and Nanoelectronics. His research interests include the design of high speed integrated circuits, embedded systems, RF MEMS and assistive devices.

Ms May Agius is currently a PhD candidate at the Manchester Metropolitan University. She also works as a Speech and Language Therapist with ACTU. Her interests include the use of AAC systems to support children with special needs to communicate. Her current research focuses on factors which influence the clinical decision making processes for children requiring AAC and have a diagnosis of Autism Spectrum Disorder. She presented this research on a number of occasions at international conferences such as ISAAC. She is particularly interested in supporting the development of AAC systems for children who require communication systems in Maltese.