Symbol Support

- A Research Concerning the Development of a Web Authoring Tool in the WWAAC Project

Karolin Bengtsson
Mia Nyström

Examensarbete, ISRN CERTEC-ER-03/4-SE

Institutionen för Designvetenskaper
Certec, avdelningen för Rehabiliteringsteknik
Lunds Universitet

Abstract

This Master's thesis is a part of the present EC-project called WWAAC, World Wide Augmentative and Alternative Communication. WWAAC is a co-operation between a few companies and organisations in Europe that aim for making a larger part of Internet services more accessible for people with communication difficulties. The users are often not capable of reading, writing or speaking so they may use different symbol systems to communicate, a range of methods referred to under the label of Augmentative and Alternative Communication (AAC). The tools that are being developed within the WWAAC project will make it easier for AAC-users to browse the Internet, send e-mail and chat. The work performed in this Master's thesis is related to the WWAAC Supportive Web Browser development and the Web Authoring Tool.

The assignment was to research the possibilities of developing a Web Authoring Tool, a tool for developing web pages with the kind of symbol support that the AAC users need. The symbol support will be a part of the page's source code and it provides the possibility to present information with representations from different communication systems. Different approaches were examined, including both a stand-alone Java application and extensions to Macromedia Dreamweaver MX, which is a commercial web designers' tool.

The Java application was developed to determine the required functionality of the extension. Based on the functional requirements a research was performed to determine what parts needed to be implemented in order to create the extension. The development resulted in a partly implemented Macromedia Dreamweaver MX extension. Instructions for further development, that will complete the product, are also specified.

Sammanfattning

(Extended Swedish Abstract)

I dagens samhälle har tekniken blivit en del av vardagen och mycket information och tjänster finns tillgängliga på Internet. Alla människor i vårt samhälle kan dock inte ta del av denna information. Det finns personer som har kommunikationssvårigheter och varken kan läsa eller skriva, vilket är en nödvändighet för att använda sig av Internet som det ser ut idag.

Just nu pågår ett EU-projekt som kallas WWAAC, vilket är en förkortning för World Wide Augmentative and Alternative Communication. WWAAC är ett samarbete mellan några företag och organisationer i Europa som syftar till att göra en större del av Internetbaserade tjänster tillgängliga för personer med kommunikationssvårigheter. Dessa personer använder sig av olika symbolsystem och annan assistans för att kommunicera – en samling metoder som refereras till under namnet Alternativ och Kompletterande Kommunikation (AKK eng. AAC). De verktyg som utvecklas inom WWAAC-projektet ska komma att underlätta för AKK-användare att surfa, skicka e-post och chatta.

En möjlighet att underlätta för AKK-användarna att ta till sig information på en hemsida är att presentera text tillsammans med symboler. Symbolstödet blir en del av hemsidas källkod och detta symbolstöd kan anpassas till att visa just de symboler som AKK-användaren är van vid och förstår. Med en anpassad webbläsare kan användaren välja att visa allt ifrån symboler, foton och animeringar till helt vanlig text för att kunna ta åt sig informationen på det sätt som passar bäst.

Eftersom symbolstödet bygger på ny och komplex teknologi har vårt examensarbete delvis inneburit att de format och utformningar som är under utveckling i WWAAC-projektet har kunnat testas i praktiken. Uppgiften i vårt examensarbete var att undersöka möjligheterna för att utveckla ett verktyg för webbdesign där symbolstödet automatiskt genereras och infogas i hemsidans källkod. För att underlätta utvecklingsarbetet har verktyget gjorts som en utökning av det befintliga webbdesignprogrammet Macromedia Dreamweaver MX.

Visionen är att lärare, professionella webbdesigners eller vem som helst som vill göra information tillgänglig för AKK-användare ska använda verktyget för att skapa symbolstödda hemsidor. På så sätt uppkommer fler anpassade och tillgängliga hemsidor för AKK-användarna och de blir inte längre utestängda från den enorma kommunikationskanalen som Internet och e-post utgör idag.

Contents

Preface

Finally, the long and many years in school have come to an end. This Masters' thesis will be our last effort before we reach real life. The possibility to work at DART and to participate in the WWAAC project has been a good appetizer for entering working life and we have learned a lot. There are many persons who have made our work possible and that we want to express our thankfulness to.

First of all, we would like to thank Mats Lundälv for believing in us from the very beginning and for being persistent when convincing the WWAAC-team that we would be useful. We also want to thank the wonderful team at DART; Gunilla, Ingrid, Eva, Anna, Karola, Marika, Janne, Lotta, Lage, Birgit and then of course Britt at SIT. They have all taught us a lot within areas we did not know existed.

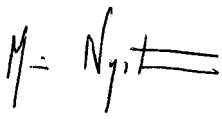
For wild, technical discussions and for wonderfully muddled drawings, we would like to thank Bengt. And thank you Lars for making the drawings understandable :)

Also, many thanks to our supervisor Kirre who, despite both heat and pregnancy, has given us her time and guidance. We wish you the best of luck with Johannes. The staff at Certec has also been very helpful, concerning all sorts of things.

Mia wants to thank Solvändan for room and a lot of good food. Karolin can certify that huge lunch boxes have been brought every day – sometimes enough for two…

Martin and Carl-Henrik, you have been a great support!

The winning concept,



Karolin Bengtsson Mia Nyström

Lund, 15th of August 2003

1. Introduction

1.1. Background

Today, a lot of information and services are available on the Internet. But the ability to use the Internet to accumulate information is not natural for everyone. Many people with communication disabilities have difficulties to use this technology. Their impairment may prevent them from browsing and they often use Augmentative and Alternative Communication (AAC) to communicate. AAC involves symbols to support written text, but symbols are unfortunately not used to a great extent on the Internet. If any support is present on a web page, the symbols are inserted as ordinary images in the source code when the page was constructed; static symbol support.

1.2. Problem

Due to the lack of symbol support on most web pages, it is hard for AAC-users to understand the information on the Internet. There are many different available symbol systems and an AAC user may want the support in the specific system she is used to. A person that is able to read the text may not want any symbols at all. Hence, there is a need for combining these requirements and provide some kind of support on a web page that is suitable for most people.

An adaptive symbol support is required to add to a web page during construction. Thus, Bliss-symbols would be shown only to Bliss-users, PCS-symbols only to PCS-users and people that do not need symbols to communicate would see only text when the web page is visited.

Two approaches are of interest and they concern both a stand-alone Java application that would operate on its own, but also extensions to the proprietary web design tool Macromedia Dreamweaver MX.

An underlying problem to this study was also to see the possibilities in obtaining a connection to a symbol server when the web authoring tool is operating. A symbol server holds symbols from one or more symbol systems and hundreds of representations from every system. Some servers are public and other servers require licenses or login and password. The one to use during this project is Symbol for Windows.

1.3. Goals

In this thesis we have five goals that we want to fulfil. The goals are:

- To understand and participate in the development in the WWAAC project.
- To learn and to use new technologies that are needed to solve the task.
- To perform a research to determine what needs to be implemented to develop a Web Authoring Tool.
- To implement the results from the research concerning the Web Authoring Tool, both as a plug-in for Macromedia Dreamweaver MX and as a plug-in for Netscape Composer.
- To document the performed work.

1.4. Purpose

The purpose of this Master's thesis is to perform research concerning the development of a web authoring tool that supports concept coding, a technology developed in the WWAAC project.

1.5. Delimitations

After obtaining knowledge about and experience with the assignment, we realized that our first goals were too extensive. Therefore, four of the goals were rewritten and delimited as follows:

- To understand and participate in the development concerning the concept coding in the WWAAC project.
- To investigate the possibilities to access the Symbol for Windows' representation database and to conclude which way is the best one.
- To partly implement the results from the research concerning the Web Authoring Tool for Macromedia Dreamweaver MX.
- To document the performed and the remaining work.

In time, this assignment has been delimited to encompass a study of both a stand-alone Java application and an extension to the proprietary software Macromedia Dreamweaver MX. Macromedia Dreamweaver MX was chosen, since it is a widely spread commercial tool for web design and it is used by both professionals and amateurs. It also allows anyone to extend its functionality by implementing extensions.

The Java application is not intended to be a working application, but is developed to determine the requested functionality of the Macromedia Dreamweaver MX extension.

Neither the extension nor the Java application, are fully implemented. The functionality is split into different areas that can be separately implemented and the limitations of which areas that are implemented were made during the development process.

The extension is developed with the purpose to support concept coding, but since the concept coding technology is not yet fully specified, it could not be completely inserted into the Web Authoring Tool at the time this thesis was written. The extension should support connectivity to several representation databases but is now limited to access only one existing symbol database, Symbol for Windows.

In the beginning of the process, this study was supposed to include a research of development of a plug-in to Netscape Composer, another web development software application. This component had a low priority from start and was excluded early on in the process due to lack of time. The implementation of the Netscape Composer plug-in would be unnecessary since implemented parts can be reused and applied to other existing applications.

1.6. DART - Description of the Organisation

DART is a centre for Augmentative and Alternative Communication and Computer Access for children, young people and adults with disabilities. DART is located in Gothenburg and is a part of the public health service in the west of Sweden. The team at DART consists of 13 specialists with knowledge of disabilities, communication and IT; occupational therapists, speech therapists, special education teachers, IT pedagogues, facilitators and engineers. The team recommends and helps clients to try out computer-based aids and to bring the products into line with consumer needs. They give knowledge of equipment, software and methods for the disabled persons but also offer education to persons involved, such as family, facilitators or teachers. [1]

1.7. Glossary and Definitions

Abbreviations and words that are used often in this thesis are explained here.

AAC	Augmentative and Alternative Communication
DART	A Centre for AAC and Computer Access in the west of Sweden for children, young people and adults with disabilities.
Extension	Adds functionality to an existing software product, in this thesis Macromedia Dreamweaver MX.

[1] http://www.dart-gbg.org/, 2003-01-27

Handicom	A company located in the Netherlands that focuses on AAC users.
SWB	The WWAAC Supportive Web Browser
Symbol support	When information is presented together with symbols
WWAAC	World Wide Augmentative and Alternative Communication
WAT	Web Authoring Tool

1.8. Outline

Chapter 1, Introduction	In this chapter we present the background and the problems that lead to the main task of this thesis. Our goals, our purpose and the delimitations that we have made are presented as well as a presentation of DART.
Chapter 2, Our right to communicate	In this chapter we present Augmentative and Alternative Communication, an area that concerns this thesis and the whole WWAAC project. The three different representation systems, PCS, Pictogram and Bliss, are discussed and different devices are presented. The reader is also introduced to an AAC-user, Marika.
Chapter 3, The WWAAC project	In this chapter we present the WWAAC project and the information is mainly based on internal sources. We introduce the purpose of the project and also give a short description of the different parts in it. We focus on the WWAAC Supportive Web Browser, of which we also present an end user evaluation.
Chapter 4, The Concept Coding Framework	In this chapter we present a new technology that is being developed within the WWAAC project; the Concept Coding Framework. Three terms that are needed to understand the purpose of the framework are explained; representations, concepts and ontologies, and an existing ontology called WordNet is introduced. The framework is used in the development of a web-authoring tool in this thesis.
Chapter 5, The Web Authoring Tool	In this chapter we present the research and development of the Web Authoring Tool. We start with a preparatory study where the need of such a tool is discussed, and where possible end users are identified. Then two different implementation approaches are explained; the development of a prototype written in Java and the development of an

	extension to Macromedia Dreamweaver MX. Our main focus will be on the development of the extension.
Chapter 6, Analysis and conclusions	In this chapter we present the results and conclusions of our thesis. We give concrete suggestions of what is left to be developed and also discuss issues that have emerged during the process.

2. Our Right to Communicate

In this chapter we present Augmentative and Alternative Communication, an area that concerns this thesis and the whole WWAAC project. The three different representation systems, PCS, Pictogram and Bliss, are discussed and different devices are presented. The reader is also introduced to an AAC-user, Marika.

2.1. Communication Disabilities

"Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers."

These lines are stated in article 19 in the United Nations' Universal Declaration of Human Rights.[2] Communication, written or verbal, is essential for interacting with people in our surroundings, but the ability to communicate is not obvious for everyone. Many people suffer from some kind of language impairment, which affects one or both of these communication processes.[3] The language impairment is often seen in combination with physical disabilities and may for instance be a result of Autism, Down's syndrome, Aphasia or Alzheimer.[4] When physical disabilities constrain the speech process, it is difficult to learn how to read and write. Therefore the communication disabilities are closely related to each other and some individuals might lack all the abilities. These individuals need an alternative way of talking and the area that provides aid for this is called Augmentative and Alternative Communication.

2.2. Augmentative and Alternative Communication

Augmentative and Alternative Communication provides help and improves the communication process for persons who lack the ability to communicate due to some kind of disability. AAC often involves symbols in the communication process but also different ways of managing computer aided communication tools. A person that uses AAC to communicate is called an AAC-user. We have chosen to split the AAC-area into two parts; symbol systems and communication devices.

[2] http://www.un.org/Overview/rights.html , 2003-07-03
[3] http://www.wata.org/resource/communication/, 2003-02-20
[4] http://www.wwaac.org , 2003-02-20

2.2.1. Symbol systems

There are several public symbol systems in use such as Bliss, PCS and Pictogram[5] and they will all be discussed later in this chapter.

The different systems can be compared using the term transparency.[6] A high transparency means that the symbol is a direct representation of the object or action. Accordingly, objects and photographs are the most transparent symbols that are being used. Drawn pictures of objects are next in range with less transparency. More conventional and arbitrary symbols have low transparency.[7] The more transparent a symbol is, the easier it tends to be to guess its meaning and thereby learn and understand it.[8] The disadvantage is that a transparent system has a limited vocabulary since mostly concrete words can be expressed. In *figure 2.1* the three different symbol systems discussed in this thesis are ranked according to their grade of transparency.

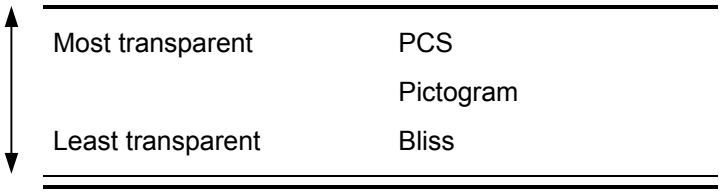


Figure 2.1: Transparency of PCS, Pictogram and Bliss, authors' interpretation, Glennen et al.[9]

To determine which symbol system that suits the AAC-user the best, tests and practice with the user together with specialists are carried out. Persons in the user's surroundings are interviewed to get a good understanding of the situation. If a user is young, is willing to learn and has the ability to use Bliss, this is the system that is strongly recommended by the specialists since it is the closest one to spoken language.

To start with, the user may have a set of simple symbols from the Bliss system in combination with photos and after some experience and follow-ups more Bliss symbols are introduced. It is also common to mix the different symbol systems if there is a need

[5] http://www.wwaac.org , 2003-02-20
[6] Glennen, S. and DeCoste, D., (1997), The Handbook of Augmentative and Alternative Communication, Singular Publishing Group, Inc. p. 109
[7] Glennen, S. et al, (1997), p. 108-110
[8] Reichle, J., York, J. and Sigafoos, J., (1991), Implementing Augmentative and Alternative Communication, Paul H. Brookes Publishing Co., p. 6-8
[9] Glennen, S. et al, (1997), p. 110

for it. For example, if a seven-year old child thinks that it is more difficult to learn a Bliss symbol than a PCS symbol it is necessary to use the PCS symbol for this specific word. When the child advances it is easy to replace the PCS symbols with Bliss symbols or/and to extend the vocabulary by adding more symbols. In this way it is still possible for the user to use her complete vocabulary although some Bliss symbols are too difficult at the time.

Another way of using the symbol systems may be to start off with a more transparent system that is easier for the user to learn. After practice and progress a change to a less transparent system can be made. Hence, it is possible to have all sorts of combinations of the systems as long as it suits the user and makes it possible for her to progress.

Whenever symbols are used to support written information we, from now on, refer to this as *symbol supported information* or *symbol support*.

PCS

Picture Communication Symbols (PCS) is a communication system that contains more than 3000 symbols and it is one of the most widely used systems. Roxie Johnson, a speech therapist in the USA, developed it in the 80s.[10] In *figure 2.2* a sentence, expressed with PCS symbols together with text, is presented.

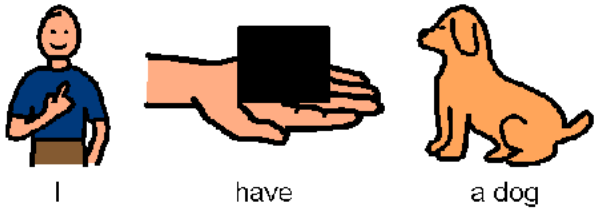


Figure 2.2: The sentence "I have a dog" expressed with coloured PCS symbols.[11]

Each symbol is a graphical image represented by a simple sketch picturing the specific object or action. The sketch is most commonly black lines on a white background but coloured PCS symbols are also available; either on a coloured background or filled in.[12] Many of the symbols have more than one representation and some of them are specially designed so that they can be altered to suit the end user. The symbols often look very similar to their corresponding three-dimensional object and it is therefore easy to guess their meaning. This characteristic is also known as a high level of transparency.

[10] http://www.mayer-johnson.com/main/index.html , 2003-07-01
[11] The PCS symbol selection is created with the Handicom software Symbol for Windows.
[12] Reichle, J. et al, (1991), p. 4

Pictogram

Pictogram Ideogram Communication symbols, also called Pictogram, is developed by Subhas C. Maharaj in the 80s and nowadays contains around 1000 different symbols.[13] *Figure 2.3* presents the sentence "I have a dog" with Pictogram symbols together with text.

The Pictogram symbol system was introduced as a communication aid for children with physical disabilities and is made with a child's vocabulary as a basis. The symbols are white-on-black drawings and they are designed to be simple; contrast rather than colour. Because of the strict and simple graphics they are liked among persons with autism.

Figure 2.3: The sentence "I have a dog" expressed in Pictogram symbols together with text. There is no available symbol for "have" in Pictogram.[14]

The symbols are good to use as a complement to sign language, since motor impairment may prevent a complete use of it.[15] They can also be used as a transition between PCS and Bliss.

Bliss

Bliss is a communication system developed by Charles Bliss in the 40s. The original purpose of the system was to promote international communication so that people from different geographic areas could communicate despite the fact that they had different native languages. Being a chemist, Charles Bliss wanted to construct a language that, similarly to the periodic system, could be used internationally and that was logically built. When structuring and designing the Bliss language, he also got inspired by Chinese symbols; a written language he had been studying during a six-year stay in China.[16]
Charles Bliss never predicted that people with communication difficulties would use his symbols, but it is within this area that the symbols have been most useful. The symbols were first used as communication aid for children with physical disabilities at a centre in

[13] http://hogan.ist.utl.pt/~tfc2000-02/pictograms/ , 2003-02-20
[14] The Pictogram symbol selection is created with the Handicom software Symbol for Windows.
[15] Heister Trygg, B., Andersson, I., Hardenstedt, L. and Sigurd Pilesjö, M., (1998), Alternativ och Kompletterande Kommunikation (AKK) i teori och praktik, Tryckfolket, Malmö, p. 54
[16] Heister Trygg, B. et al, (1998), p. 54

Canada in the early 70s and they soon spread to several countries around the world. They were first used in Sweden in 1976.[17] Today, the symbols are used in more than 35 countries all around the world.[18]

The standardised Bliss system contains more than 3500 graphical symbols[19] and the symbols can be combined in an endless number of ways. It is possible to express both simple sentences as well as very complex and grammatically correct ones; a strength that makes it possible to individualise the usage according to the users personal needs and skills. The sentence "I have a dog", expressed in Bliss symbols together with text, is presented in *figure 2.4*.

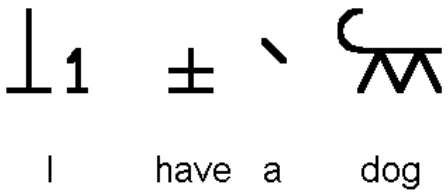


Figure 2.4: The sentence "I have a dog" expressed in Bliss symbols together with text.[20]

The base in the standardised system consists of approximately one hundred main symbols, of which some are presented in *figure 2.5*. The words are simple and represented by only one graphical symbol, for example *house, car, animal, sun, feeling* etc.[21]

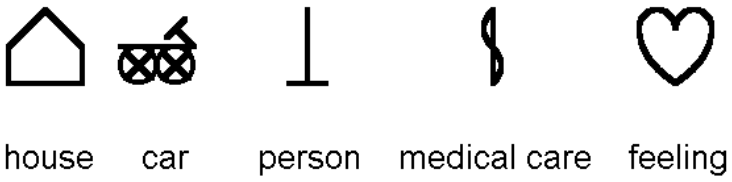


Figure 2.5: Five of the approximately one hundred main Bliss symbols.

Two or more of the main symbols can be combined into new symbols and the majority of the symbols in the standardised library are built in this way. For example, the symbols for *hospital, ambulance* and *doctor* are all strongly connected to the word *medical care*.

[17] Heister Trygg, B. et al, (1998), p. 54
[18] http://home.istar.ca/~bci/intro.htm , 2003-03-10
[19] http://home.istar.ca/~bci/intro.htm , 2003-07-02
[20] The Bliss symbol selection is created with the Handicom software Symbol for Windows.
[21] Heister Trygg, B. et al, (1998), p. 55

Therefore, they all contain the basic symbol for *medical care* in combination with, respectively, *house, car* and *person*. The combinations can be viewed in *figure 2.6*.

Figure 2.6: Symbols that are constructed as a combination of main symbols.

It is also possible to decline words and to create new forms of the word. The operators that indicate pluralism, infinitive, imperfect and adjective are presented in *figure 2.7*.[22]

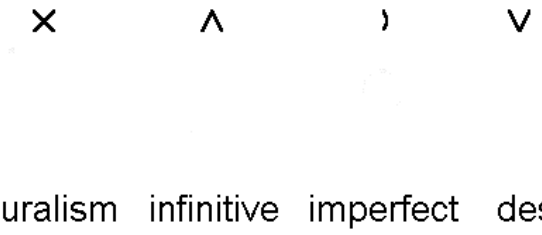


Figure 2.7: Bliss operators that indicate pluralism, infinitive of a verb, imperfect of a verb and a describing adjective.

With the opportunity to express different tenses, to use both concrete and abstract words and to create endless combinations of words, the Bliss system can be used to express very complex sentences that are grammatically correct.

The symbols are separated into different categories and the background colour shifts according to what group the word belongs to. Yellow background indicates nouns, red indicates verbs, green indicates adjectives, blue indicates pronouns and white indicates prepositions.[23] In this way, it is easier for the user to find the word and speak fluently and fast.

The development of the Bliss system is an active on-going process. Each new symbol that is added to the standard Bliss library has been examined and approved by Blissymbolics Communication International (BCI). BCI is a charitable organisation that has the world

[22] Heister Trygg, B. et al, (1998), p. 55
[23] Heister Trygg, B. et al, (1998), p. 55

wide and perpetual license for the use and publication of the symbols and they strive to maintain a standardised Bliss system.[24]

2.2.2. Communication Devices

Apart from the fact that all AAC-users communicate via different symbol systems they still use the same devices. A device may be either low-tech or high-tech.

Low-tech Devices

A low-tech device is a device that does not need a power supply.[25] It may be a flat communication board, which can be mounted to the wheel chair. This board has a map, which consists of a matrix filled with symbol representations; each cell contains one symbol. When talking, the user points at the symbols. The receiver reads the words out loud and helps the user to construct the sentence.

The size of the map and cells depends on the user's impairment and precision. The better the precision is and the more words the user learns, the more symbols are added to it. The map is individually created and contains the words that the user selects most frequently. A simple map may consist of only a few symbols but an advanced user may have around 500 symbols.

Other kinds of low-tech devices are communication books. They vary in size depending on the user's motor impairment and ability to turn pages. Communication books vary in endless ways; size, shape and contents are all accustomed to the user. It may, for example, be a small binder with a couple of pages filled with symbols. The books are easy for the user to handle and bring along. It may also be a larger file to fold in thirds or quarters with symbols on both sides. Different alternatives of books are shown in *figure 2.8*.

The symbols in communication books are categorised in a way that suits the user, for example, school symbols on one page and leisure-time activities on another page. The symbols may also be categorised according to word classes, as the Bliss symbols described in *chapter 2.2.1*. Hence, any kind of structure is possible as long as it is simple for the user to find the correct symbol.

[24] http://home.istar.ca/~bci/intro.htm , 2003-03-10
[25] http://www.wata.org/resource/communication/ , 2003-07-02

Figure 2.8: Different kinds of low-tech communication devices. A and D shows communication books in the shape of large files. B shows smaller communication books and C shows a kind of communication map.

High-tech Devices

A high-tech device needs a power supply[26] and is often a computer aided communication tool. It can be either computer software or a hardware device.

The computer software vary from stand-alone programs to additional supporting tools that are connected to standard programs. A free-standing program might be a text editor with symbol support. A supportive program is used to connect two standard programs. For example, if an AAC-user wants to send an e-mail it can be done by using two or three different software tools in a combination. It may be one standard text based e-mail program, one symbol supportive program that helps the AAC-user to write the text in the letter and a third program to obtain the connection between the two programs. This is a rather complicated way of communicating via the Internet that will be improved due to the result of the WWAAC project, *chapter 3*.

[26] http://www.wata.org/resource/communication/ , 2003-07-02

A hardware device can be an individually adapted input device, such as a head-mouse or a joystick. A few different input devices are presented in *figure 2.9*. It can also be different types of speech synthesis machines; either used separately or used as a software tool on the computer. The keyboard of the speech synthesis machine can hold either letters or symbols and when a key is pressed, the corresponding word or phrase is read out loud.

A **B** **C**

Figure 2.9: Three kinds of high-tech input devices are presented. A and B show different kinds of head mouses and C shows adapted hand mouses.

All the existing tools, low-tech or high-tech, can be combined in numerous ways as long as there are no technical restrictions. Every adaptation is personal, since all the users have various kinds of impairments.

2.3. Marika – an Advanced AAC-user

This chapter is based on an interview with Marika Landgren at DART, May 13, 2003. She is an adult Bliss symbol user who suffers from the congenital brain injury cerebral palsy, CP.

Marika was introduced to the Bliss symbols at the age of three and then started using a communication board. Her first board consisted of four symbols; *mum*, *dad*, *sister* and *ice cream*. She made a fast progress and after a few years her board held around 200 symbols. The successful advance was a consequence of her stubborn characteristics in combination with a very supportive family.

At the age of seven, Marika started school together with five other children. Three of her classmates were able to communicate via speech and the other three, including Marika, used Bliss boards to talk. The communication boards, with speech synthesis, were electric and connected to a screen. When the boards were used, the text was shown on a screen that everyone in the room could see. In this way the teacher and the classmates could communicate with each other and Marika could participate actively.

Marika also started to use another kind of speech synthesis machine during her Swedish lessons and at home. On its keyboard, there were only letters and no symbols. The machine's speech synthesizer read the word exactly as it was spelled, so if a word was spelled incorrect, Marika immediately could hear that it was wrong and correct the spelling. When working with the machine this way, Marika obtained good knowledge of spelling and learned to connect a specific sound to its corresponding written word. She was eager to learn how to read and write and she used the machine every night at home. Over a night, she realised the connection between symbol, word and sound. So after having used the machine for several years, Marika was suddenly capable of both reading and writing. This breakthrough happened at the age of fourteen.

Today, Marika lives in her own apartment and manages most of her daily duties without any assistance. She has a part time work where she, among other things, composes Bliss communication maps that suit other AAC-users. Her own communication board now contains 500 symbols. Most of the symbols are Bliss symbols but 28 squares contain the alphabetic letters, so that Marika can spell words that are not on the board. 10 squares on her board contain numbers and 28 squares contain written words like *to, will, would, hope, never, always* and *probably*; short, everyday words that might not have a corresponding symbol or that are equally easy to understand without symbols. Marika's Bliss map can be viewed in *figure 2.10* but also in a larger scale in *Appendix A*.

It is unusual that communication maps contain written words without the corresponding symbols, but in Marika’s case it works perfectly well. She would be fully capable of communicating with a board containing only written words, but she speaks faster using the combination of symbols and words.[27]

[27] Two months after the interview, Marika has changed her board to contain only words and no symbols.

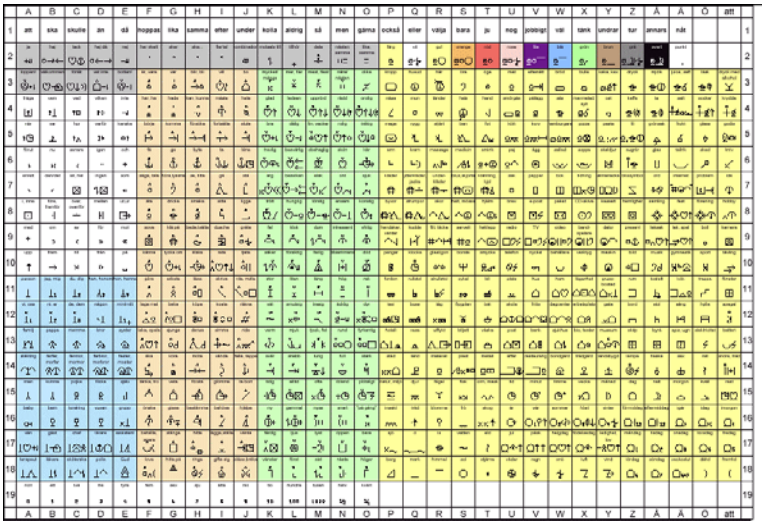


Figure 2.10: Marika's Bliss map. The map can be viewed in full scale in Appendix A.

A very important part of Marika's social life is the contact with people that use the Internet. By sending e-mails and chatting she keeps in touch with most of her friends. She doesn't use phones very much for talking, but finds text messages (SMS) on her cell phone to be an extremely efficient communication complement. Since Marika can write and read, she has no problem accumulating information on the Internet and she is fully capable of using the Internet tools that are available today. Therefore, she is not a considered end user of the browsing and e-mail tools that are being developed within the WWAAC project, *chapter 3*. Although, she may be interested in using the Web Authoring Tool, which is developed with this thesis as a basis.

3. The WWAAC Project

In this chapter we present the WWAAC project and the information is mainly based on internal sources. We introduce the purpose of the project and also give a short description of the different parts in it. We focus on the WWAAC Supportive Web Browser, of which we also present an end user evaluation.

3.1. An Overview of the WWAAC Project

The WWAAC project, the World Wide Augmentative and Alternative Communication project[28], started in January 2001 and will be completed in May 2004. It is a co-operation between a few companies and organisations in Europe. They aim for making the Internet available for people with communication disabilities; AAC-users.

In the beginning of the project a feasibility study was performed. No adequate tools for web browsing with symbol support were found, but only some insufficient tools for emailing and chatting. A need for AAC user friendly tools, such as a symbol supported web browser and a symbol supported e- mail client, was identified.

The project also identified the end users and the identification resulted in:

"People who use graphic symbol-based augmentative and alternative communication (AAC) systems to support/replace speech and/or reading and/or writing, and older people with communication disabilities who may, or may not, be relying on symbol based communication aids, but who would benefit from adapted computer software, to help access Internet services." [29]

To set up the user requirements, end users were interviewed. These persons vary in age and may have different kinds of impairments, have learning difficulties or may use some kind of communication equipment. Accordingly, they were divided into three groups:

1. The first, and largest, group included school-aged persons, 10-18 years old, who suffers from congenital communication disabilities. This group contained about 50% of the participants, since younger people have a great interest in technology and they are more likely to be interested in the usage of the Internet. They also have access to professional support, through school personnel and therapists, when using AAC systems.

[28] http://www.wwaac.org , 2003-02-17
[29] WWAAC, (2001), User Requirements Document, Issue 1, p. 1

2. The second group of end users consisted of adults with congenital communication disabilities. This group represented 20% of the participants. The majority of the group is likely to be interested in and motivated to use the Internet technology; especially the young adults. There is also a good economic foundation to provide this category of persons with the resources they need.

3. The third, and last, group of end users encompassed elderly people with acquired communication disabilities. This group was smaller than the other two, because they do not have as much professional support as the others. They may also show little interest in using the Internet technology, because they are unfamiliar with it and have little experience of it. Although, the group was small, it was of great interest for research and development, since many elderly people suffer a stroke and may need communication aids at a late stage in life.

Other categories of people were also interviewed. These categories included software and hardware manufacturers, facilitators, teachers, speech and language therapists and additional family members; persons who have experience of the AAC area. The interviews were performed to get a wide range of aspects concerning the end users. All the aspects were taken into count and the resulting requirements were set up. An extract from the user requirements' document is presented below[30].

"User requirements capture prioritised the work of the WWAAC consortium:

- *Priority 1: Access to and use of information and services of the World Wide Web*
- *Priority 2: Use of email between people using the same language (e.g. English to English, Swedish to Swedish and so on.)*
- *Priority 3: Use of discussion fora*
- *Priority 4: Use of email between people using the different languages (e.g. between English and Swedish)*
- *Priority 5: Echat"*

According to the user requirements, the development of five symbol-supported services became the main goal of the project. The idea was, and still is, to make the services less text-oriented than the ones on the market today. Information is to be presented together with symbols and text is to be read with speech synthesis. The five services were soon reduced to three; a web browser, an e-mail client and an e-chat client. Today, the web browser has the highest priority and the e-chat is more or less replaced with the development of a web-authoring tool. At the time this thesis was written, an alpha version of the web browser was available. The web browser and a user evaluation of it are

[30] WWAAC, (2001), User Requirements Document, Issue 1, p. 2

presented in *chapter 3.2*. The development of the web-authoring tool is presented in *chapter 5*.

3.2. The WWAAC Supportive Web Browser, Alpha Version

The WWAAC Supportive Web Browser (SWB) is an adapted web browser that acts as a wrapper to Internet Explorer. It differs from available browsers in several ways. The user interface and the navigation methods are adapted to fit the needs for AAC-users. The browser also provides alternative ways for text input and text output. These are the main differences from available browsers and this is what makes it possible for AAC-users to accumulate information on the Internet.

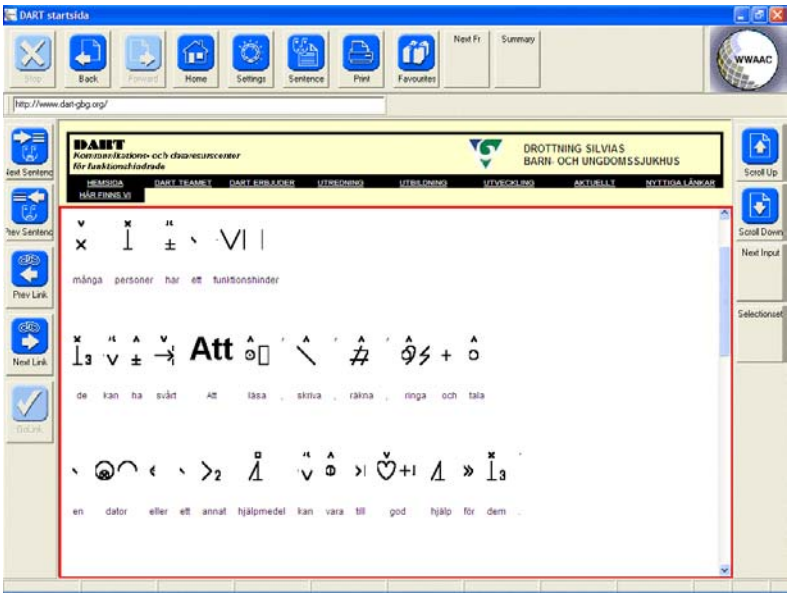


Figure 3.1: A symbol supported web page shown with the WWAAC supportive web browser, Alpha version.

Graphical User Interface

All the functions in the web browser, that are needed to browse, are available from the button panel and may be reached by different key combinations. The buttons provide possibilities to follow links, activate the speech synthesis, go back to the previous page and navigate to different sections of the page. These are just alternative ways of reaching the functionality and most of the functions can also be reached by mouse-clicks.

The interface can be configured in many different ways to suit every user. Buttons can be removed, replaced or added and the icons on the buttons can be changed. In *figure 3.1*, a screen shot of the web browser is presented.

Navigation

To navigate on a web page may be difficult for people who are unable to read. Since this, most commonly, is the case for AAC-users, the SWB provides support for it. One feature is that the currently read text is marked, so that it is easier to follow the reading process when the speech synthesis is activated. Not only is the text marked, but also the frame where the speech synthesis is currently active. The possibility to move between the different frames on the web page is available from the button panel.

Another difficulty for AAC-users is to discern which parts of the text that are links and where the links lead. The problem can be solved by using the speech synthesis for reading the links. There are also buttons to move between the available links on the page and to follow them.

Text Input and Output

One severe problem for an AAC-user is to produce and understand written text on a web page. When entering input on a web page, there is a possibility to connect external symbol supportive programs to the browser. A supportive program may, for example, let the user write text by using her adapted symbol set, hence, the user can produce the required text input without assistance.

The SWB provides two ways of making written information available to the AAC-user; symbol support and built-in speech synthesis. The browser is able to interpret symbol support on a web page if it is present. If symbol support is implemented in the source code of the page and if there is a connection to a symbol database, the user will get the information presented together with symbols. The principles of symbol support will be further explained in *chapter 4*.

The speech synthesis is a built-in functionality that can be configured, concerning voice, speed and sound, to suit the user. The speech synthesis can be activated by using the buttons on the button panel. The settings can specify if a single word, a sentence or a whole section is to be read when the button is pressed. The speech synthesis is not dependent on whether there is symbol support on the web page or not; it reads any available text.

3.2.1. Evaluation of The WWAAC Supportive Web Browser

As participants in the WWAAC project we got the opportunity to take part in the Swedish evaluation[31] of the WWAAC Supportive Web Browser, Alpha version. The evaluation tested the software and its functionality and resulted in valuable feedback from the test persons. This feedback was evaluated to see if the product satisfied the users and if their attitude to such a program was mainly positive. The results will be considered during the development of the Beta and the Final version.

Two AAC-users, both using Bliss, were involved in the test. Both the test persons had some experience with Internet Explorer, but were unable to use it without assistance. Therefore, they are both representative candidates for the SWB.

The test persons had access to the browser for one day. The test started off with an introduction and continued in separate rooms together with therapists and facilitators, so that the test persons could get the support that they needed. The tasks to perform during the evaluation consisted of different scenarios. Functionality, such as navigating and using the built-in speech synthesis, were tested, but also the design of the user interface.

According to the test persons, the SWB would enable them to use the Internet without any assistance. The evaluation showed that one of the better parts was the speech synthesis, especially since it made it easier to follow links on the page. The alternative method for text input was also appreciated.

Aspects concerning the user interface involved the design of the browser buttons. Due to the too homogenous design, it was hard for the test persons to distinguish the buttons from each other. Two suggestions of how to make them clearer were made. One of them was to give the buttons different colours depending on their functionality. The second suggestion was to add speech synthesis to the buttons; when right-clicking on the button, the functionality of the button should be read out loud.

[31] The same evaluation has taken place in the Netherlands and in the United Kingdom.

4. The Concept Coding Framework

In this chapter we present a new technology that is being developed within the WWAAC project; the Concept Coding Framework. Three terms that are needed to understand the purpose of the framework are explained; representations, concepts and ontologies, and an existing ontology called WordNet is introduced. The framework is used in the development of a web-authoring tool in this thesis.

4.1. Representations

The development of the Concept Coding Framework (CCF) will involve not only symbols, but *representations*. The generalisation is made to avoid the limitation of saying that symbol support only involves symbols.

A representation may be any kind of format that can be used to describe a word; a text string, a symbol or a picture. A text string in English is one kind of representation and a text string in Swedish is another. Other representations may be photos, animations and sounds. For example, a person using sign language may be interested in getting the support as animated sign language instead of symbols. Hence, the common expression of all these formats is representations and the purpose of the CCF is that all kinds of representations will be supported. *Figure 4.1* gives a few examples of different kinds of representations.

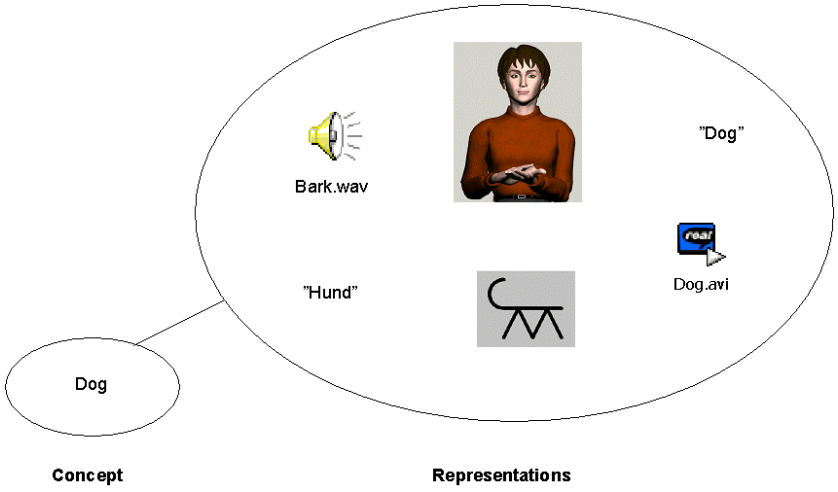


Figure 4.1: The relationship between the concept dog and its representations; sound, sign language animation, text strings, a symbol and a film sequence.

4.2. Concepts

A *concept* is a superior expression for representations and it is this term that the representations describe. In *figure 4.1* the relationship between the concept and its representations is shown.

4.3. Ontologies

An ontology is a description of concepts and relationships that exist in our language and it is defined as a specification of a conceptualization[32]. The relationships form a structure that can be seen as a hierarchical tree structure. This structure can vary from simple one-layer word lists to very complex structures with fifteen or more layers.

One example of a very simple existing ontology is the one in Handicom's software *Symbol for Windows*[33]. The ontology describes the vocabulary for the symbol databases that they provide. In *figure 4.2* an extract of their ontology is shown. The example shows the hierarchical structure for the word *dog*.

Figure 4.2. The hierarchical tree structure, for the chosen word dog, in the ontology that is defined by Handicom. The example is from the software Symbol for Windows.

Today, there is no standardised ontology, but the providers of representations often construct their own ontologies. Since the WWAAC project concerns such a wide area as the Internet, the support for representations has to be standardised. This standardisation would facilitate for distributors and web authors to use symbol support. In order to perform the standardisation, a new WWAAC ontology has to be defined; an ontology inspired by WordNet.

[32] http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2003-06-02
[33] http://www.handicom.nl/, 2003-03-19

4.4. WordNet

A larger and more complex existing ontology is *WordNet*. WordNet will be used as reference ontology in the WWAAC project, to help identifying useful concepts. This will be necessary when retrieving representations to support text. Some words may not have a representation in a specific representation system, but can be described by other related words that have corresponding representations. To retrieve those related words, WordNet is used.

WordNet is a free online lexical database that is developed at the Princeton University. The main purpose of the database was to build structures based on human lexical memory and to show how the words are linked to each other, rather than just listing the words and their definitions as in an ordinary dictionary. WordNet contains more than 138 000 English words and concepts that are organised in different tree structures. Nouns, adjectives, adverbs and verbs are represented in separate structures, where the words are organised into synonym sets. [34]

A specific word can belong to one or more synonym sets. It means that the word may have several different meanings. A word may also reside in more than one tree if the word, for example, is both a noun and a verb. The word *dog* resides in both the noun tree and the verb tree and has several synonyms. Its senses, as defined in WordNet, can be viewed in *Appendix B*.

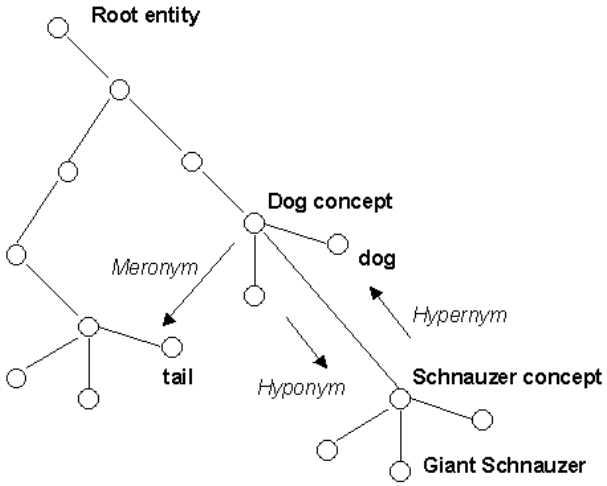


Figure 4.3. An example of what a branch, in the noun structure, in WordNet may look like.

[34] http://www.cogsci.princeton.edu/~wn/, 2003-03-28

The tree structure in WordNet consists of numerous branches and layers. An example, of what a branch in the noun structure may look like, is presented in *figure 4.3*. Five layers are shown, and the top layer is called the *root entity*, which means that every concept in the noun tree can be derived to this concept root entity. One disadvantage with separate trees is that it is impossible to get the connection between words from different parts of speech even if they are closely related. For example, the relationship between *dog* and *bark* cannot be found in WordNet, since they reside within different trees. The problem could be solved by connecting the trees on an even more general layer.

The relations in the tree structures have different names. On the basis of *figure 4.3* we describe three relations that are interesting:

- Hypernyms
  A hypernym is the parent node to the current word. If the word is *schnauzer*, the hypernym is *dog*, since it completes the statement: *Schnauzer is a kind of...*

- Hyponyms
  A hyponym is a child node to the current word. If the word is *dog*, one hyponym is *schnauzer*, since it completes the statement: *...is a kind of dog.*

- Meronyms
  A meronym is a part of the current word. If the word is *dog*, one meronym is *tail*, since it completes the statement: *... is a part of a dog.*

For example, say that we need a representation for the word dog. If there is no representation available for that word, the hypernyms, hyponyms and meronyms can be used to find semantically reasonable alternatives for the word. Hence, instead of ending up with no representation at all, there are several possibilities of finding representations to describe the word anyway.

4.5. Concept Coding

In an attempt to co-ordinate the global software development concerning symbol support, the Concept Coding Framework is developed within the WWAAC project. The CCF is based on *concept coding*. The idea of concept coding is to produce unique codes for concepts. A selection of commonly used concepts is to be identified and given identification numbers called *concept codes* (CC).

Hence, concept coding is an attempt to standardise the use of concepts and representations. It can be compared to the internationally standardised ISBN-number for books. When different users are using a specific ISBN-number they are all referring to

the same book. Likewise, when a WWAAC concept code is being used, it always refers to the same concept.

An example of the relations between the concept code, the concept and the representations for *dog* is presented in *figure 4.4.*

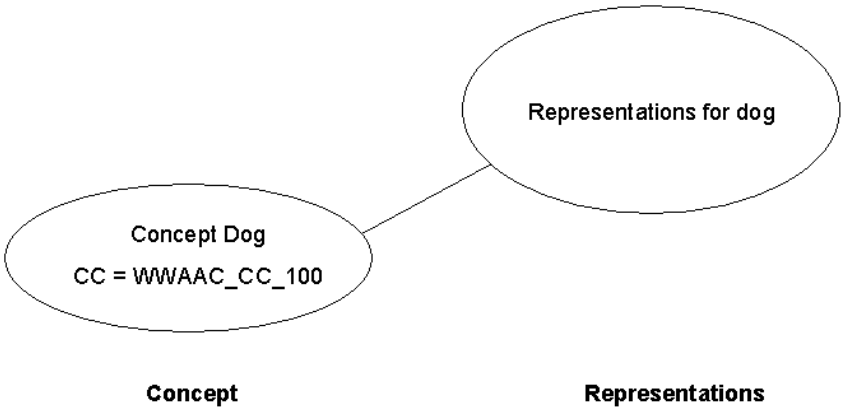


Figure 4.4: The concept dog is identified with the concept code WWAAC_CC_100.

The selection of concepts that are to be identified and standardised is mainly chosen from WordNet. The first version of the WWAAC standardised selection will contain 1500 concepts from WordNet. Since WordNet does not contain any pronouns, prepositions or conjunctions, the selection needs to be complemented with about 200 such words. The selection has to be big enough for practical user evaluation but small enough for practical implementation exercises. [35] It will be extended continuously.

Concept Bridges

The development of the Concept Coding Framework is an attempt to make the representation distributors "speak the same language". Today, different distributors have their own ontologies, hence, different concept identification numbers. If an extern application, such as a web-authoring tool, needs to communicate with two representation databases from different distributors, two separate solutions have to be implemented. Each implementation is unique and interprets the distributors' individual concept numbers. Hence, the development of compatible software with connections to different distributors requires a unique implementation each time it is performed. This leads to problems when it comes to the Internet, since the different representation systems are hard for developers to co-ordinate.

[35] WWAAC, (2003), Concept Coding Framework – Technical Specification, Issue 0.5, p. 2-4

Concept coding makes it easier for extern developers to use symbol support, since it will be up to the distributors to implement a common API[36], the CCF. Only one mapping per distributor needs to be implemented; the mapping between the distributor's identification system and the WWAAC concept codes. Once the mapping is implemented, the distributor's representations can be reached through the CCF. The mapping can be thought of as a *concept bridge* and is visualised in *figure 4.5*.

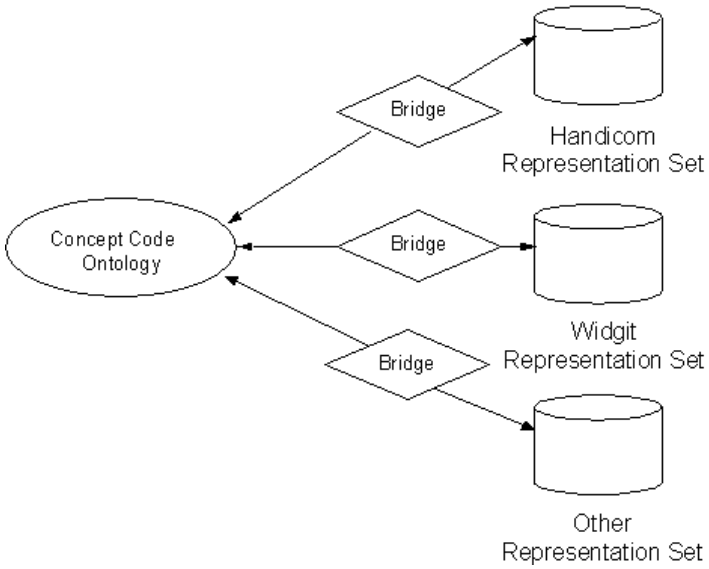


Figure 4.5: The concept coding framework acts as a bridge between different distributors, author's interpretation. [37]

The implementation of the concept bridge will be essential for the distributors to perform, since they thereby, contribute to the global dissemination of accessibility for AAC-users. There are several distributors on the market, so it is in their interest to keep their representation databases easily accessible.

Database Accessibility

The range of representation databases varies. Most databases are licensed since they contain licensed representation systems, for instance, PCS and Pictogram. Such a database may be available on a licensed server that, accordingly, requires user id and a password. It could also be locally installed on the client.

[36] API is short for Application Programming Interface
[37] WWAAC, (2003), Concept Coding Framework – Technical Specification, Issue 0.5, p. 1.

Some representation systems, such as Bliss, are public and do not require a license. It would be desirable to make those systems available on public servers, so that they are accessible for everyone. This solution does not exist today.

When representations are globally used i.e. on the Internet, as many representation databases as possible should be easily accessible. In this way, written text on web pages would get satisfactory symbol support, since the databases can complement each other.

Implementing the Support

The representation support on a web page will be represented by a code-fragment in the source code of the page. The fragment will be described in RDF/XML, where RDF is a Resource Description Framework and XML is an Extensible Markup Language. [38] This format is common when representing information on the Internet.

```
<concept rdf:id="23:1" cc="WWAAC_CC_100">

dog

<Representation name="Distributor" type="image" value="any
attributes in a string to access the database"/>

</concept>

<rdf:Description  rdf:about="23:1"  rdf:reference="any
attributes in a string to access Symbol for Windows'
databases"/>
```

Figure 4.6: A fictitious example of an RDF/XML-fragment that implements symbol support.

The fragment, *figure 4.6*, will mainly consist of two tags that will be registered at the World Wide Web Consortium (W3C)[39]. The first tag, <*concept*>, specifies the unique WWAAC concept code. The following tag, <*representation*>, resides one or more times within the concept-tag. The collection of representation-tags specifies all the available representations that describe the concept, but each specific representation needs its own tag. The representation-tag holds essential information such as distributor, representation format and any other individual data that is needed to access a specific database.

The relations between all the components included in the Concept Coding Framework are visualised in *figure 4.7*.

[38] http://www.w3.org/TR/2003/WD-rdf-syntax-grammar-20030123/, 2003-02-10
[39] http://www.w3.org/, 2003-01-27

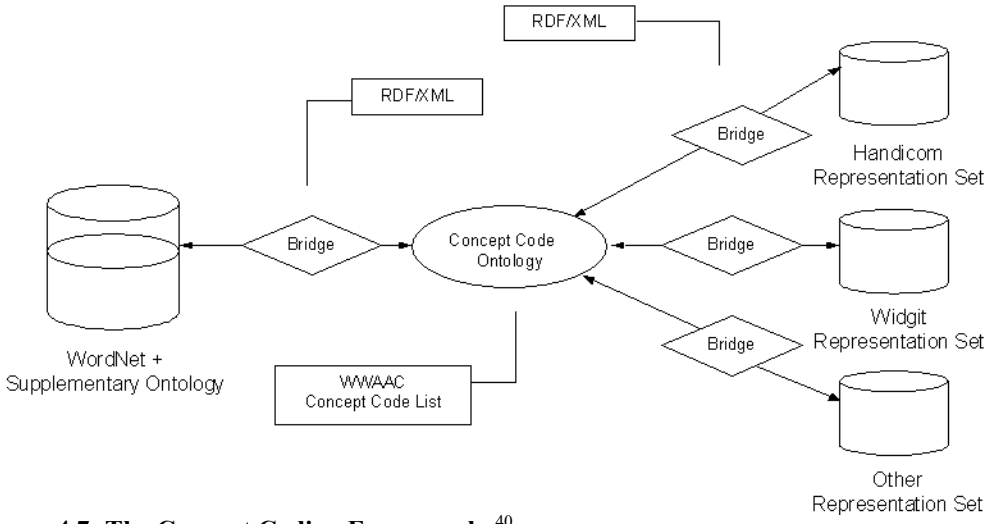


Figure 4.7: The Concept Coding Framework. [40]

Scenarios

The use of the Concept Coding Framework will now be presented in two scenarios. The following special cases will be given attention:

- No concept code is available for the current word. (1)
- No representation is available for the specified concept code. (2)

1. Development of a web page with symbol support

A web-author is creating a web page and wants to support a piece of text on the page with symbols. To keep the scenario simple we assume that this text contains only one word: *schnauzer*. To create the RDF/XML fragment for the source code, the concept codes for the concepts in the text string are needed. Thus, in our example, the concept code for *schnauzer* is needed.

The concept code can be retrieved through the Concept Coding Framework API. If the concept *schnauzer* is specified in the WWAAC Concept Code list, hence, has a concept code, this code is returned. If not, an alternative concept code needs to be returned. The CCF uses WordNet to find synonyms that can be used, and in our case, the hypernym *dog*, is an alternative good enough. The concept *dog* is defined in the WWAAC Concept Code list and its concept code can be used by the developer. So, instead of excluding the symbol support, an alternative description is added to the page.

[40] WWAAC, (2003), Concept Coding Framework – Technical Specification, Issue 0.5, p. 1.

2. Viewing a web page with symbol support

An AAC-user is browsing and reaches a web page with symbol support. She is a Bliss-user and her adapted WWAAC web browser is configured, with Bliss as the primary symbol system. She is also connected to a local symbol server that is installed on her own computer.

The supported text on the web page she reaches contains the word *Coke*, which is defined in the WWAAC Concept Code list and therefore has a concept code. The problem is that there is no available Bliss-symbol for *Coke* so the database she is using contains no corresponding symbol. The symbol database has a defined ontology where *Coke* is a specification of the superior term *fizzy drink*. In the database, there is a corresponding Bliss-symbol to *fizzy drink* and this symbol will be shown as the alternative symbol. In this way, the user gets sufficient support. For a PCS-user this problem would not appear since the PCS system contains a symbol for *Coke*.

5. The Web Authoring Tool

In this chapter we present the research and development of the Web Authoring Tool. We start with a preparatory study where the need of such a tool is discussed, and where possible end users are identified. Then two different implementation approaches are explained; the development of a prototype written in Java and the development of an extension to Macromedia Dreamweaver MX. Our main focus will be on the development of the extension.

5.1. Preparatory Study – The Need of a Web Authoring Tool

Today, there are no web authoring tools on the commercial market that support the insertion of adaptive symbols to a web page. The only alternative, when a web author wants to make information available for AAC-users, is to insert the symbols as static images.

To clarify the difference between static and adaptive symbol support we explain the terms separately:

- *Static symbol support* means that images are statically inserted to the web page. When such a web page is opened in a browser, it will contain exactly the symbols that were inserted into it. If the web author added only Bliss-symbols, everyone visiting the page will see the same Bliss-symbols, and there will be no support for other symbol systems on the page. Static symbol support is quite simply inserted as <IMG> tags in the HTML source code.

- *Adaptive symbol support* means that the symbol selection that is added to the web page is interpreted according to the browser settings. When a web page with adaptive symbol support is opened in a browser, different users will see different symbol sets. A Bliss-user will see Bliss-symbols, a PCS-user will see PCS-symbols and a person who is able to read will see plain text. Adaptive symbol support is inserted as an RDF/XML-fragment into the HTML source code.

There are some available software products on the market that provides the possibility to create static symbol support. *Symbol for Windows* is one of them. The program lets the user write documents with text and corresponding symbols. Depending on which licenses the user possesses, different symbol systems are available. The document can be saved as an HTML-document and viewed in a browser and everyone viewing the page will see

exactly the same symbols. Widgit Software's *Writing With Symbols* [41] is another product that can be used in a similar way.

There are no available products today that provide the possibility to insert adaptive symbol support to a web page. The procedure is more complicated than inserting static symbols, since it involves both concept coding and RDF/XML. If the insertion is to be done without using a tool, the web author needs to be familiar with both those technologies and also understand what specific parameters that are needed to retrieve the symbols from the databases. It is, therefore, more convenient to use a web authoring tool that automatically generates and inserts the code fragment with all the required information.

Among the different representation databases on the market, the development of the web authoring tool in this chapter is based on Handicom's database in Symbol for Windows, since Handicom constitute one partner in the WWAAC project. Hence, the concept bridge between the WWAAC Ontology and Handicom's representation set, which can be viewed in *figure 4.7*, is to be partly implemented.

5.1.1. Identifying Possible Users

A discussion with experienced AAC-specialists was performed in the beginning of the research. The purpose of the discussion was mainly to evaluate the idea behind the tool and its functionality, but also to identify possible users of the web authoring tool. Possible users that were identified are:

- AAC-users (with or without help).
- Family and relatives to an AAC-user.
- Facilitators.
- Teachers.
- Managers of collectives for AAC-users.
- Professional web authors.
- Private web authors that constructs web pages as a hobby.

There may be other categories of users identified later in the project, who are not known at the time.

The participants in the discussion were positive to a web authoring tool, but considered advanced AAC-users to be more interested in it than AAC-specialists. They agreed on, that there is a need for presenting information with symbols on the Internet and this

[41] http://www.widgit.com/, 2003-07-01

information may vary from personal home pages to pages holding relevant information for everyone in a particular group of AAC-users.
None of the participants had any knowledge of HTML except that they recognized the term and they knew that it was related to web pages. Therefore, they are not actually representative users of the web authoring tool, since the use of it requires some experience with HTML. Instead, they both claimed that, if they would like to present any information on the web, they would turn to the web master.

The minimum knowledge of HTML that is required to manage the insertion of the support is to know where on the web page the code is inserted, so that the layout of the web page can be controlled. Hence, the development is focused on minimize this amount of knowledge, since many of the identified users may be unfamiliar with HTML. The goal is that all the identified users should be able to use the tool after a short introduction to basic HTML.

All the aspects above were taken into count during the development of the web authoring tool.

5.2. Prototype development

Two approaches of implementing the web authoring tool were considered and tested. The first approach was to develop a prototype in Java. The main purpose of developing the prototype was to determine the functionality and in broad outlines design the graphical user interface.

5.2.1. Functionality

The web authoring tool should offer a simple way of inserting both static and adaptive symbol support into a web page. As little experience as possible with HTML should be required of the user. Its functionality is described in the following items:

1. **Act as a text editor for HTML-files**

   The tool should offer functionality to construct home pages with more content than just symbol supported text. It should also have basic functionality for working with documents such as open, save, close etc.

2. **Automatically connect to available symbol servers**

   The tool should be able to automatically connect to one or more pre-defined symbol servers, from which the representations could be retrieved. The definition should be done in the settings of the web authoring tool.

3. Insert static symbol support

The tool should offer functionality to insert static symbol support for one or more words or sentences. No concept coding is involved; the symbols are inserted as ordinary images in HTML with <IMG>-tags.

4. Insert adaptive symbol support

The tool should offer functionality to insert adaptive symbol support for one or more words or sentences. The support involves concept coding and the support is inserted as an RDF/XML-fragment. *Figure 4.6* in *chapter 4.5* presents the appearance of the fragment.

5. Preview with support

A specific parser should make it possible to preview the constructed home page with symbol support. The settings in the preview mode should be possible to alternate, so that the adaptive support could be viewed with the different available representations.

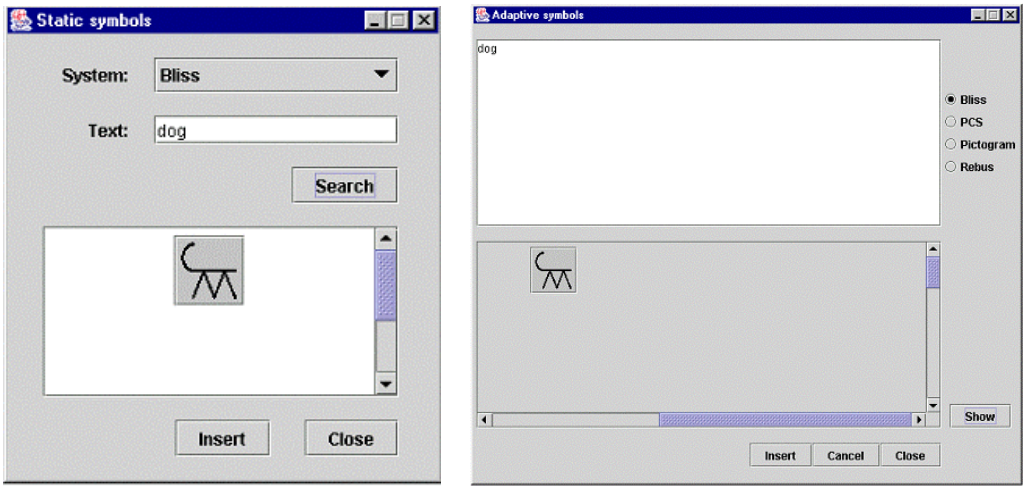
6. Cheap and easily available

The web authoring tool should offer a cheap solution and be reasonably accessible, since many of the identified users may be private, and not able to buy another expensive tool.

5.2.2. Graphical User Interface

The graphical user interface of the prototype was carefully designed and the most important components were the ones that offer insertion of static and adaptive symbol support. The design was clear and simple; a text field where to enter the text and a symbol panel where the available symbol set was shown. The interface for adaptive symbols offers a possibility to preview all the available symbol sets; one at a time. Both interfaces offer a possibility to edit the symbol selection if it is not satisfactory.

When the web author is satisfied with the symbol selection, the insert button is pressed. For the static support, the <IMG>-tags are created. For the adaptive support, the RDF/XML-fragment is constructed from the text input and concatenated with essential information about the database that is to be used. The code is then inserted into the source code of the web page. The two interfaces are visualised in *figure 5.1*.



A B

Figure 5.1: Parts of the graphical user interface in the Java prototype. A is used for inserting static support to a web page and B is used to insert adaptive symbol support. The text to support is entered into the text field and the selected symbol set is presented below.

5.2.3. Results of Prototype Development

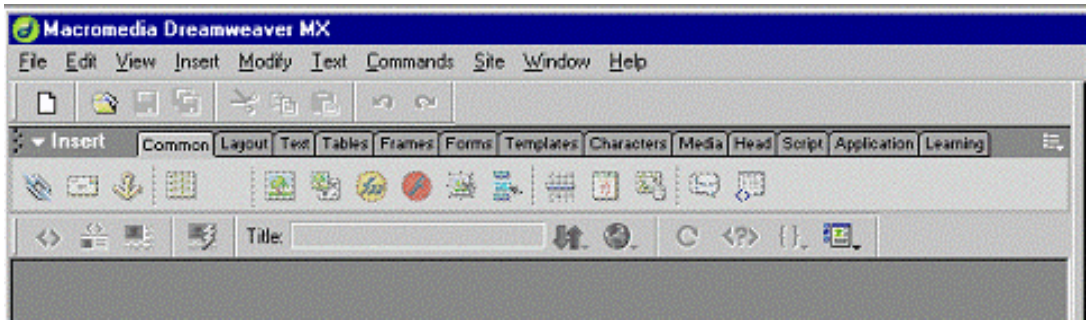
During the development of the Java prototype, an idea of creating a plug-in to an existing software product was formed. The disadvantages of developing completely new software were too many. A lot of time had to be spent on implementing functionality that had nothing to do with the purpose of the product, such as saving and opening files. It would take too much time to complete the product.

Although the prototype in Java would offer cheap software and would be a great tool for private use, it would be too simple for professional use. The conclusion was made, that it would be easier to get advanced web authors to use the product if they could use the same tool that they currently use, instead of a completely separate tool. Web authoring tools currently on the market were discussed and Macromedia Dreamweaver MX turned out to be the most interesting one.

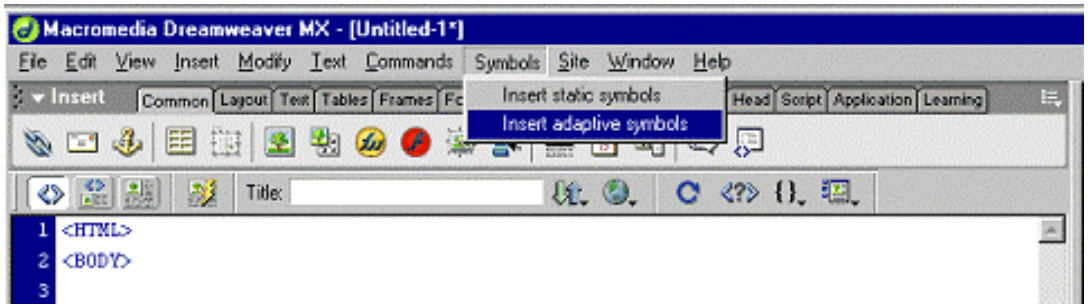
The major parts of the work performed when developing the prototype, could be used for the development of the extension as well. Information and knowledge were gathered on how to implement most of the functionality. The main parts of the most important functions, 3, 4 and 5, were implemented or have proposed solutions of how to be performed.

5.3. Macromedia Dreamweaver MX Extension

The second approach of implementing the web authoring tool was to create an *extension* to the existing software Macromedia Dreamweaver MX[42]. An extension extends the functionality of the installed program and consists of menus or buttons, as visualised in *figure 5.2*, that implement a specific behaviour. The behaviour may edit or affect the source code of the current web page in one way or another and it is also possible to specify XML-tags and what is to be performed when the tag is parsed. With these requirements fulfilled, it is possible to use such an extension to implement the web authoring tool.



A



B

Figure 5.2: A visualisation of Macromedia Dreamweaver MX, with (B) and without (A) an extension with symbol support.

When starting to work with Macromedia Dreamweaver MX, several problematic areas were identified. The development of the extension was therefore divided into six subparts to make it more comprehensive. The different functionality areas that need to be

[42] http://www.macromedia.com/, 2003-03-27

implemented in order for the extension to be complete are visualised in *figure 5.3*. Questions to answer during development are presented in the list below.

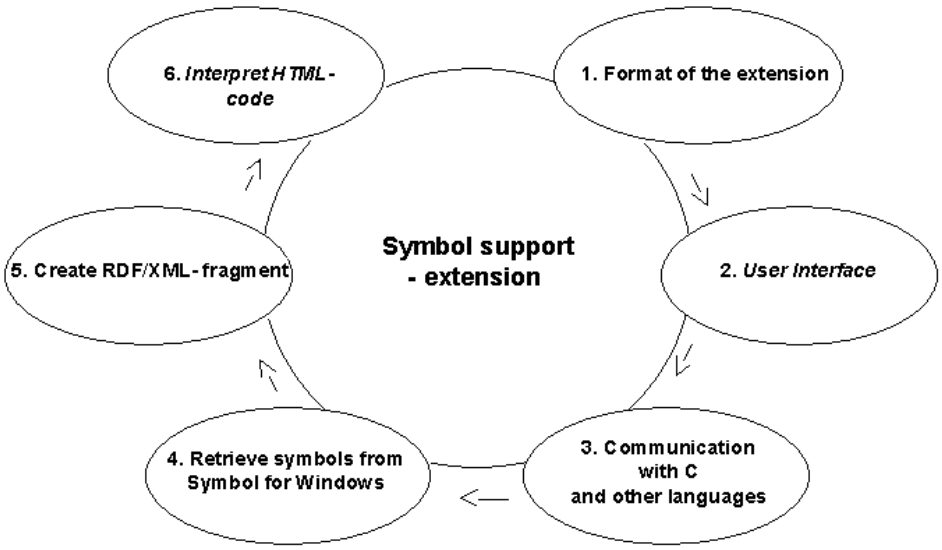


Figure 5.3: The construction of a Macromedia Dreamweaver MX extension was divided into smaller parts. The italic marked text constitutes parts that are theoretically solved but not fully implemented. The other four parts are mostly implemented. The user interface has a close connection to the part where C-code is communicating with other kinds of languages.

1. **Format of the extension**

   What files and formats are used when creating an extension? How is the extension package created? How is the package installed and uninstalled? The solutions are further described in *chapter 5.3.1*.

2. **User interface**

   Is it possible to use the graphical user interface that was created in the Java prototype? If not, how is a new graphical user interface designed and implemented? The solutions are further described in *chapter 5.3.2* and in *chapter 5.3.3*.

3. **Communicate with C-code from other languages**

   How can C-code and Java-code be called from Java Script? Does the API in Macromedia Dreamweaver MX support functions that call any external code?

This part is closely related to the development of the user interface and is further described in *chapter 5.3.2*.

4. **Retrieve symbols from Symbol for Windows**

   How are the symbols, which are to be shown in the graphical user interface, retrieved from Symbol for Windows? The solution is further described in *chapter 5.3.4*.

5. **Create RDF/XML-fragment**

   What information is needed to construct the code-fragment that constitutes the adaptive symbol support and how is the fragment constructed? The solution is further described in *chapter 5.3.5*.

6. **Interpret XML-fragment**

   What is needed to parse and interpret the RDF/XML-fragment that is inserted into the source code, in order to retrieve the right symbols to show on the web page? The task is further described in *chapter 5.3.6*.

5.3.1. The Extension format

An extension is built from several different types of files. They specify the behaviour of the extension and how it is supposed to appear in the user interface of Macromedia Dreamweaver MX. They also specify what kind of behaviour that the extension will have when the web author is using it. In *figure 5.4* the different components of an extension are shown and they are described in the list below:

- **Ext.xml / Ext.mxi**

  The XML file holds software specific information such as what version of a program the extension is compatible with, what kind of extension it is and how it will affect the basic user interface in Macromedia Dreamweaver MX. In our case the file specifies that the extension is of the type *command* [43] and that a new menu with submenus is to be added to the menu bar in the user interface.

  This file also specifies what other files that constitute the actual extension and that are needed to run the extension. In our case they are some of the mentioned files in the figure; Java Script (JS), HTML.

[43] One of Macromedia's specified extension types.

This XML file has to be re-saved with the file extension *.mxi*, so that the Macromedia Extension Manager [44] can understand it when bundling all files together. This is the specification file and parts of it are shown in *Appendix C*.

- **Ext.html / Ext.js**

  The HTML file and the Java Script file together specifies what action to perform when the web author uses the extension. In our case it means to provide the web author a possibility to insert symbol support to the current web page. These files are behaviour specific files.

- **Images**

  If the extension needs any images in its graphical user interface, they have to be bundled together with the other files. No such images will be used to create the actual extension in our case, since the images that are to be shown in the graphical user interface are retrieved dynamically.

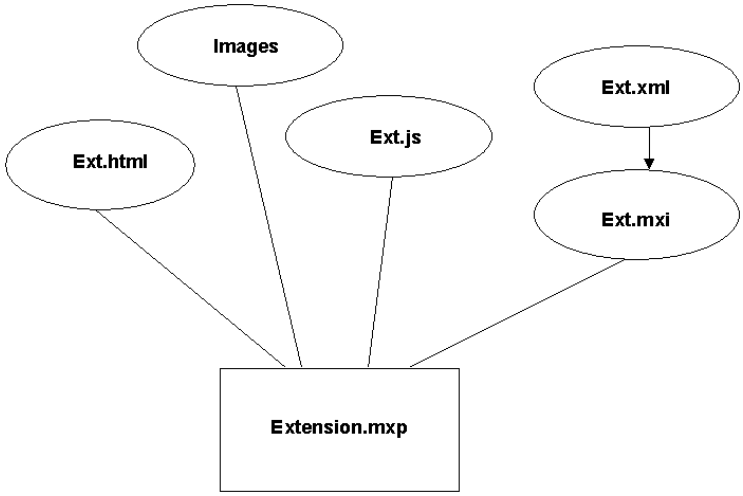


Figure 5.4: The parts of an extension in Macromedia Dreamweaver MX. The parts in the oval shapes are bundled together to a package with the file extension .mxp.

All the files are bundled together into an extension package, with the file extension *.mxp*, by using the Macromedia Extension Manager. This package is the one to install, hence the one that adds the specific behaviour to the basic software. To distribute a specific implemented behaviour, it is common to download a complete extension from the Internet.

[44] http://www.macromedia.com/, 2003-03-27

To install the extension, the Macromedia Extension Manager is needed. This software searches for extension packages in the current directory. This package now contains all the different files that are needed to install and run the extension; Ext.mxi, Ext.js, Ext.html and possibly images. When the extension package is executed and installed, the specification file directs the behaviour specific files into the Macromedia directory, where the extension files are intended to be stored. It also stores itself, the specification file, in a directory that is processed at startup. In this way the information in the specification file is added to the initialisation file of Macromedia Dreamweaver MX and the extended behaviour is added to the program.

5.3.2. User Interface – Reuse of the Java GUI

Since the major parts of the graphical user interface were implemented during the development of the prototype, an attempt was made to reuse the Java components within the extension. An overview of the communication process for this course of events is shown in *figure 5.5*. The HTML/Java Script-part corresponds to the extended user interface in Macromedia Dreamweaver MX, where the web author, via a menu, reaches the functionality of supporting text with symbols.

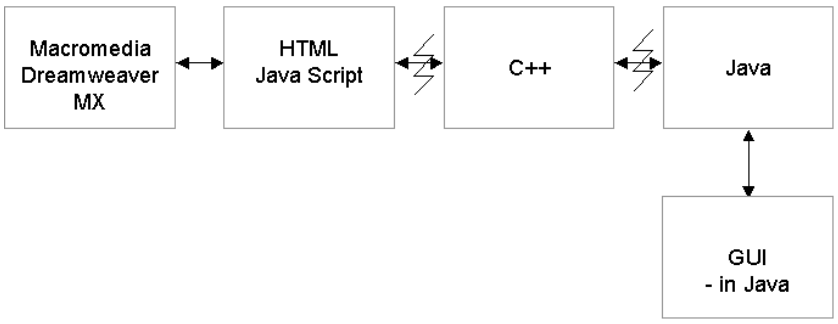


Figure 5.5: An overview of the communication process when re-using the Java GUI as an extension to Macromedia Dreamweaver MX. Special interfaces are needed between Java Script and C++ and between C++ and Java to obtain the communication link. The interfaces are marked in the figure.

To launch the Java GUI from Macromedia Dreamweaver MX, it was necessary to implement wrapper functions in both Java Script and C++; in JavaScript to reach the C functions[45] and in C to get access to the Java functions. Two different interfaces were used to obtain this communication link, but only one C file was used to implement both

[45] C++ will, from now on, be referred to as only C, because it simplifies for the reader when reading the text.

the interfaces. Example code representing both interfaces is presented in *Appendix C*, *figure C.2 – C.8*.

- **JS Native**

  JS Native is the built in interface in Macromedia Dreamweaver MX, which enables access to C functions from Java Script. The native C functions in a Dynamic Linked Library (DLL) can be reached from JS.

  The C functions that are to be called from Java Script has to implement the specific native signatures, which are known to Macromedia Dreamweaver MX. Though, it is possible to add functionality specific parameters in addition to the mandatory ones.

  The C functions also need to be defined in a specific interface function, which Macromedia Dreamweaver MX automatically registers when the C library is used in JS. The name and number of parameters of the native function in C is thereby known by the installed software.

- **Java Native Interface**

  The Java Native Interface (JNI) enables communication between Java and C. This feature makes it possible to implement parts of an application in a native language, for example, if a specific part is time consuming. The C functions have specific signatures which are automatically generated when the Java file is compiled with specific options. The actual native implementation resides in the C functions.

  Not only is the JNI involved, but also the Java Virtual Machine (JVM) library. It has to be loaded into the C code to get access to the JVM functions, since this library contains functions, such as, how to create a JVM and how to determine if there already is a running JVM. In our case we had to create a new instance of the JVM.

  When the JVM and the Java Native Environment were up and running, it was possible to let the environment search the directory and find the required Java class to launch the Java GUI. A native object of the Java class was created in the C code and the methods in the class were called from the environment. This resulted in the Java GUI being launched.

The C library, containing the complete connection from JS to Java, has to be located in Macromedia Dreamweaver MX's directory. This is the directory, in which Dreamweaver searches for specific libraries that are used. It is also necessary to set the class path in the

C code to point out the location where the Java Archive file resides, since they contain the code for the Java GUI.

5.3.3. User Interface – Macromedia Dreamweaver GUI

When testing the implementation, where the graphical Java components were re-used, it turned out to be very slow and volatile. Macromedia Dreamweaver MX had no support for using external Java code and the communication link was too long, which made the process slow. Therefore, other solutions of making a user interface were investigated, and Macromedia Dreamweaver MX offered a contingency to create the extended user interface by using HTML and Java Script. The communication link is visualised in *figure 5.6*.

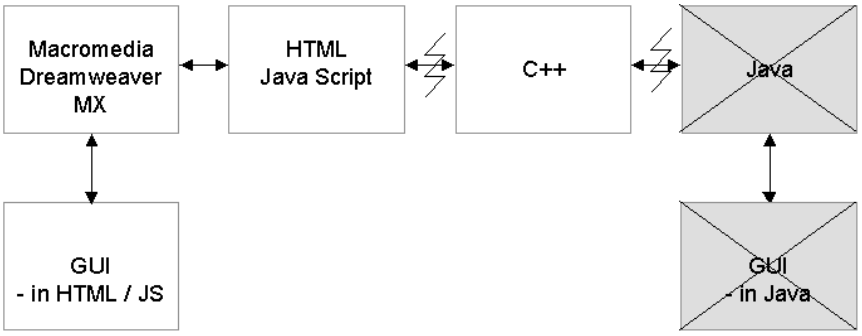


Figure 5.6: The approach of making a GUI in Macromedia Dreamweaver MX's design.

This interface was possible to design by the existing functionality in the software, and predefined forms could be used. The interface was specified in HTML/JS-files and action listeners could easily be added to the buttons and text fields. The design had the same look as the main program, and it was not as distinguished as the Java GUI.

Since the communication link to C already was obtained, it would be a very simple way of designing the interface. Unfortunately, this part was not fully implemented during the development process, but the proposed solution on how to solve the problem will be a good basis for further development.

5.3.4. Retrieve Symbols from Symbol for Windows Database

To dynamically retrieve the symbols, that were to be shown in the user interface of the extension, the existing database Symbol for Windows was used. The database

functionality was specified in an API, where the available language was either Pascal or C.[46] In our case we were only interested in the C-functions.

The most interesting parts of the API were the signatures, the return values and the parameters of the available functions that were to be used. The current library with functions had to be loaded from the C-code using a load library method. Function pointers to the functions were declared and then the corresponding procedure addresses were retrieved. In this way we had access to the API methods from our C-code.

Some of the C-functions in the API, that was needed to retrieve a symbol for a specific word, required a Symbol for Window's ontology identification number as an in-parameter, instead of the word itself. To retrieve that number for a specific word, a mapping between the word and the identification number was needed. The mapping was implemented by using the existing initialisation files in Symbol for Windows, but the procedure was time-consuming. Another approach was made to simplify the mapping. This approach involved using a local Java database that read the initialisation files and performed the mapping by a simple request to the database.

The procedure of retrieving the actual image data through the API required the use of the Windows Graphics Device Interface (GDI)[47]. The symbol was not returned as an object, but drawn as raw image data into a memory area in the computer memory. Before the data could be drawn, the memory area had to be pre-allocated and prepared with different sets of parameters. The area also had to be compatible with the colours and settings of the screen, since these conditions are checked before the data can be drawn.

The image data that was drawn to the memory area was represented by a Bitmap (BMP). To decide if the image was correctly retrieved from the database, a temporary implementation in Java was made, since we are much more familiar with Java than C when it comes to graphical representations. The JNI enabled us to return the BMP to Java and display it on the screen.

An overview of what files are involved, when retrieving a symbol from the Symbol for Windows' database to Java, is shown in *figure 5.7*. What each file represent and how the different files are created, is specified in the following list.

[46] Handicom, (2001), Polysym technology access library, Internal source, Issue 1.5, p. 1-16.
[47] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/devcons_7e2b.asp, 2003-03-24

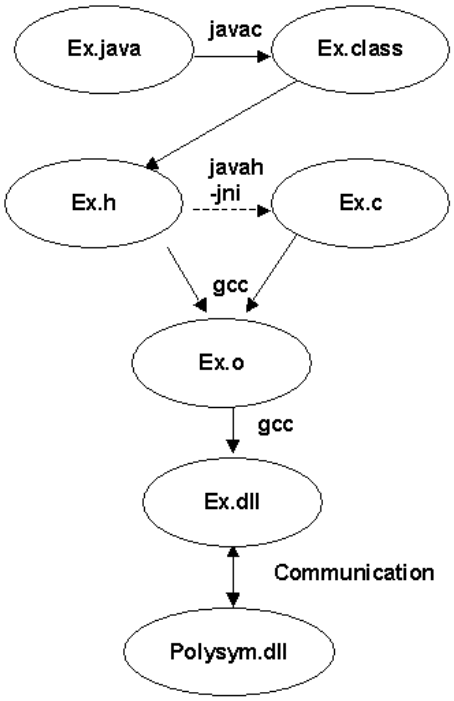


Figure 5.7: How the files collaborate when communicating with the database in Symbol for Windows.

1. **Example.java**

   This java file contains code and functions for the Java GUI, in which the BMP is to be viewed. It also contains signatures and parameters for the native functions that are to be implemented in C.

2. **Example.class**

   This class file is created when the java file is compiled.

3. **Example.h**

   This header file is automatically generated when the compiler command *javah* is used on the class file. The signatures and parameters of the native functions, which were declared in the Java-file, are extended and exported to this file.

4. **Example.c**

   This file contains the actual implementation of the functions that are specified in the header file. The database DLL is loaded into this file, so that the API functions are reachable.

5. **Example.o**

   This object file is created from the header file and the C file.

6. **Example.dll**

   This DLL will communicate with the database DLL and is created and linked by using the object file.

7. **Polysym32.dll**

   This file is the database DLL, which provides the database API, through which the symbols are retrieved.

5.3.5. Create RDF/XML Fragment

The RDF/XML fragment that constitutes the adaptive symbol support is created with input data from the user interface. It is mainly based on the entered text that is to be supported. This text string is parsed and concept codes to the corresponding words are retrieved via the Concept Coding Framework. The logic behind the user interface will concatenate this information with database specific information, such as, representation system and internal identification numbers in the database. The fragment will then be inserted into the source code of the web page.

Since the area is still under development, it is not sure if the fragment will look exactly as in *figure 4.6*. The tags and the structure of the fragment are currently under discussion with W3C and other proposed solutions may appear.

One vague proposition is that the text, which the writer wants to support, could remain unaffected; no tag with concept code will be inserted into the text. Instead, the support would be appended to the bottom of the web page by individual RDF/XML-fragments that refer to the corresponding text in the document by using x-y-coordinates. This fragment would then consist of concept codes and other necessary information.

5.3.6. Interpret HTML- code by Using a Translator

When a web page with symbol support is constructed in Macromedia Dreamweaver MX, it should be possible to preview the design in the application without launching a supportive browser. The addition of associated translators in Macromedia Dreamweaver MX enables this possibility and can be used to translate the concept code tags to the corresponding image.

The translator consists of a set of different files, *figure 5.8*. The HTML and Java Script files contain code to translate a tag. The Java Script code has to implement two functions

that provide translator information to the program and in addition to these, any number of functions could be added. If a translation is time consuming and complex there is support for extending the functionality by using native C files and thereby make the extension more efficient. The specification of the concept tag and its attributes is described in the EDML files; files that consist of XML-structured data.

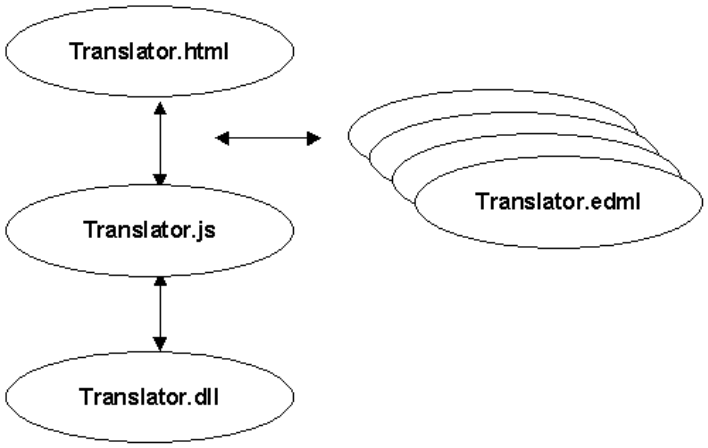


Figure 5.8: Files that describe the data translator and behaviours in Macromedia Dreamweaver MX.

The translator will operate on the current document when the user switches from code view to preview. If any occurrences of the concept tag are found in the document the translator for the concept tag is invoked. The search pattern, which is specified in the EDML files, is interpreted and temporarily replaced in the current document by the HTML specific image tag. In the design mode, this enables the possibility to view the representations that correspond to the concept codes in the source code.

This part was not fully implemented during the development process, but the proposed solution is considered to be a good basis for further development.

6. Analysis and Conclusions

In this chapter we present the results and conclusions of our thesis. We give concrete suggestions of what is left to be developed and also discuss issues that have emerged during the process.

6.1. Results

The thesis has resulted in a partly implemented Macromedia Dreamweaver MX extension, with proposed solutions for the tasks that remain unimplemented. When referring to the tasks as they are presented in *figure 5.3*, it is easier to explain what parts that have been implemented.

1. Format of the extension. This part is fully solved. We have managed to create, install and uninstall extensions. We have made a detailed description of the files that are needed to create an extension.

2. User interface. We have managed to launch our Java interface from within Macromedia Dreamweaver MX, but since it was too slow, the solution was not durable. Therefore, we have explained a proposed solution for how to create the graphical user interface using the Macromedia Dreamweaver MX API, but this solution is not yet implemented.

3. Communication with C and other languages. We have managed to establish the communication links that we needed between C, Java and Java Script. The communication processes have made use of native communication between dynamic linked libraries, DLLs. One very small part that remains unimplemented is to send strings between the C code and the Java Script code.

4. Retrieve symbols from Symbol for Windows. Symbols are successfully retrieved from Symbol for Windows through the database API. The process makes use of graphical C-functions in Windows' Graphic Device Interface and the symbols are retrieved as BMP-images. Though, they need to be saved as PNG-images before this task can be considered fully implemented.

5. Create RDF/XML-fragment. We know what is needed to create the fragment and we know how it can be done, but this part has not been implemented, since the construction of the fragment is not yet specified. This part simply consists of concatenating input data.

6. Interpret XML-fragment. The method of translating the XML-fragment when previewing in Macromedia Dreamweaver MX has a proposed solution and we have made short examples of how it can be done. Nevertheless, this task reaches over the development of the WWAAC Supportive Web Browser as well, and needs to be implemented once the design of the XML-fragment is decided. When a parser is implemented for the SWB, either this browser can be used for previewing, or parts of the code can be used for implementing the translation directly into Macromedia Dreamweaver MX. If using SWB, the difference will be that the browser has to be launched, which is a little bit more ungainly than simply switching to preview mode. On the other hand, it is an easier solution to implement.

6.2. Further Development

The Macromedia Dreamweaver MX extension will be further developed and, hopefully, completed by developers within in the WWAAC project. According to the results in *chapter 6.1*, there are four specific subparts left to implement before the extension is complete. The subparts are:

- The images that are fetched from the Symbol for Windows symbol server are currently saved as bitmap data. To be able to show the images in the user interface of Macromedia Dreamweaver MX, the images need to be transformed into the PNG format. The task should be easy to implement and not very time consuming.

- Strings have to be sent between the C code and the JavaScript code. Since the format of the string is differently specified, this doesn't work correctly in the current version. The task is easy and rated to be the least time consuming task that is left.

- The user interface must be implemented and input data must be taken care of. The interface design can be performed with help from Macromedia Dreamweaver MX and is not very difficult, but the task is time consuming.

- The translator of the XML-fragment needs to be implemented. This part must also be implemented in the browser, in order for the browser to understand the RDF/XML fragment. This is the most extensive task that is left to implement and might be difficult.

6.3. Reflections

To determine if our initial goals have been fulfilled, we present the following list of what we believe we have achieved:

- Participated actively in the WWAAC project.
- Obtained knowledge and understanding of technologies used in the WWAAC project, such as concept coding.
- Obtained knowledge and understanding of Alternative and Augmentative Communication and AAC-users.
- Performed a research to determine what functionality that needs to be implemented to develop a Web Authoring Tool.
- Partly implemented the results from the research.
- Documented and presented a detailed description of the remaining work.

We consider our goals to be fulfilled, despite the fact that we did not implement all functionality. The remaining parts are thoroughly researched and well documented for further development.

6.4. Problems and Issues

During the work with this thesis some problems and issues have been encountered. They have affected the performed work in different ways and with various extents.

6.4.1. Computers and Development Tools

Before the actual work with the thesis could start off the computers had to be prepared. The PCs were relatively old and worn out and not faultless, which were noticed during the development. Their operating system, Windows98, complicated the process, since it is incompatible with many of the software programs we used.

No software development tools that we could use for this task were pre-installed. Instead it was a procedure of downloading and installing trial versions of available software. It was often time-consuming and one of the disadvantages was that the trial versions only last for 30 days and then the license had to be renewed, if possible. There was no intention to buy any licenses because of the expense of them; it would cost too much for the organisation, which already had a limited amount of resources to use. But, in some aspects, it would have facilitated the development because we could have been concentrating more on our work and not on the development tools.

Another encountered problem concerned the compiler, which was used to compile and link the C-code; GNU's compiler[48]. According to the documentation, there was a possibility to us it together with Windows98, but no documentation explained how to do it. It was a lasting process to get it running and, eventually, the right options for compiling and linking were found with help from our technical support.

6.4.2. Documentation of Macromedia Dreamweaver MX

Several problems with Macromedia Dreamweaver MX have been encountered during the work. There is a lack of information in the help files of the program; when searching the documentation for one topic, only a limited amount of information is received. The documentation is not extensive enough, but refers in circles. There is also incorrect information in the help files and no reference to any online help is given although there is one. Neither the home page of Macromedia nor the help files were very informative. The information seems hidden and it is not easy to find. A better way to get help was to take part in the discussion groups on the Internet.

6.4.3. New Technologies

The need to learn different technologies and get them working together has sometimes affected the work in a negative way. The number of new technologies we needed increased with the tasks when new subparts were to be investigated. It was a very time consuming part where a lot of information had to be processed during a short time, and hence the schedule has partly been dislocated because of the reading. One of the technologies were Windows' GDI, where the documentation was clumsy and rather hard to understand. This was the part where most of the reading was done.

6.4.4. Technical Support

Technical support has not been easy to get when it was needed. Our supervisors have not been in to the kind of programming we have been doing, but we have had contact with some of the developers in the WWAAC project. The contact was sporadic, because the other developers were not situated in the same building, as we were; some of them, not even in the same country. Communication problems via e-mail occurred.

6.4.5. WWAAC – project still going on

The overall issue during the thesis was that the WWAAC project is present and will not finish until 2004. The technology concerning the concept coding was under development

[48] http://www.gnu.org/, 2003-07-01

and since it was not completed, there was no possibility to implement some of the parts of the Web Authoring Tool.

6.5. Future Aspects

With the performed research as a basis there is a good chance for the WWAAC project to develop a complete Web Authoring Tool for the commercial market, either as a stand-alone application or as an extension to Macromedia Dreamweaver MX. There are also great possibilities to develop a plug-in to Netscape Composer, which will have the same functionality as the extension. This plug-in may also be able to reuse parts of the Java code developed in this thesis.[49]

The functionality to support text with various representations could be used in different ways. One way is to insert support in the summary part of the web page. It is already possible to create a summary with only text, but with representations it would be more accessible to people.

Another area where text together with representations is desirable to use is in the new technology concerning annotations. This technology is under development of W3C and is encouraged to be used. Annotations are personal or public comments that are added to a web page and they are either stored locally on the user's computer or public on an annotation server. An annotation could make the information of a web page more clear and accessible to an AAC-user. This technology is good to use on web pages that already exists and that won't be remade with an overall support.

Further development regarding different databases and representations would be interesting to see. Once the concept-coding framework is fully developed it is up to the software developers to implement the framework, and then there is a possibility to use other databases than Symbol for Windows' database when inserting symbol support. One possible candidate may be the database in Widget's software products.

[49] http://wp.netscape.com/eng/mozilla/3.0/handbook/plugins/index.html, 2003-07-01

7. References

Literature

Glennen, S. and DeCoste, D., (1997), The Handbook of Augmentative and Alternative Communication, Singular Publishing Group, Inc.

Reichle, J., York, J. and Sigafoos, J., (1991), Implementing Augmentative and Alternative Communication, Paul H. Brookes Publishing Co.

Heister Trygg, B., Andersson, I., Hardenstedt, L. and Sigurd Pilesjö, M., (1998), Alternativ och Kompletterande Kommunikation (AKK) i teori och praktik, Tryckfolket, Malmö

Internet Sources

http://www.dart-gbg.org/, 2003-01-27

http://www.un.org/Overview/rights.html , 2003-07-03

http://www.wata.org/resource/communication/, 2003-02-20

http://www.wwaac.org , 2003-02-20

http://www.mayer-johnson.com/main/index.html , 2003-07-01

http://hogan.ist.utl.pt/~tfc2000-02/pictograms/ , 2003-02-20

http://home.istar.ca/~bci/intro.htm , 2003-03-10

http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2003-06-02

http://www.handicom.nl/, 2003-03-19

http://www.cogsci.princeton.edu/~wn/, 2003-03-28

http://www.w3.org/TR/2003/WD-rdf-syntax-grammar-20030123/, 2003-02-10

http://www.w3.org/, 2003-01-27

http://www.widgit.com/, 2003-07-01

http://www.macromedia.com/, 2003-03-27

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/devcons_7e2b.asp, 2003-03-24

http://www.gnu.org/, 2003-07-01

http://wp.netscape.com/eng/mozilla/3.0/handbook/plugins/index.html, 2003-07-01

http://mozquito.markuplanguage.net/, 2003-07-01

http://www.x-smiles.org/, 2003-07-01

Internal sources

WWAAC, (2001), User Requirements Document, Issue 1

WWAAC, (2003), Concept Coding Framework – Technical Specification, Issue 0.5

Handicom, (2001), Polysym technology access library, Internal source, Issue 1.5

Appendix A

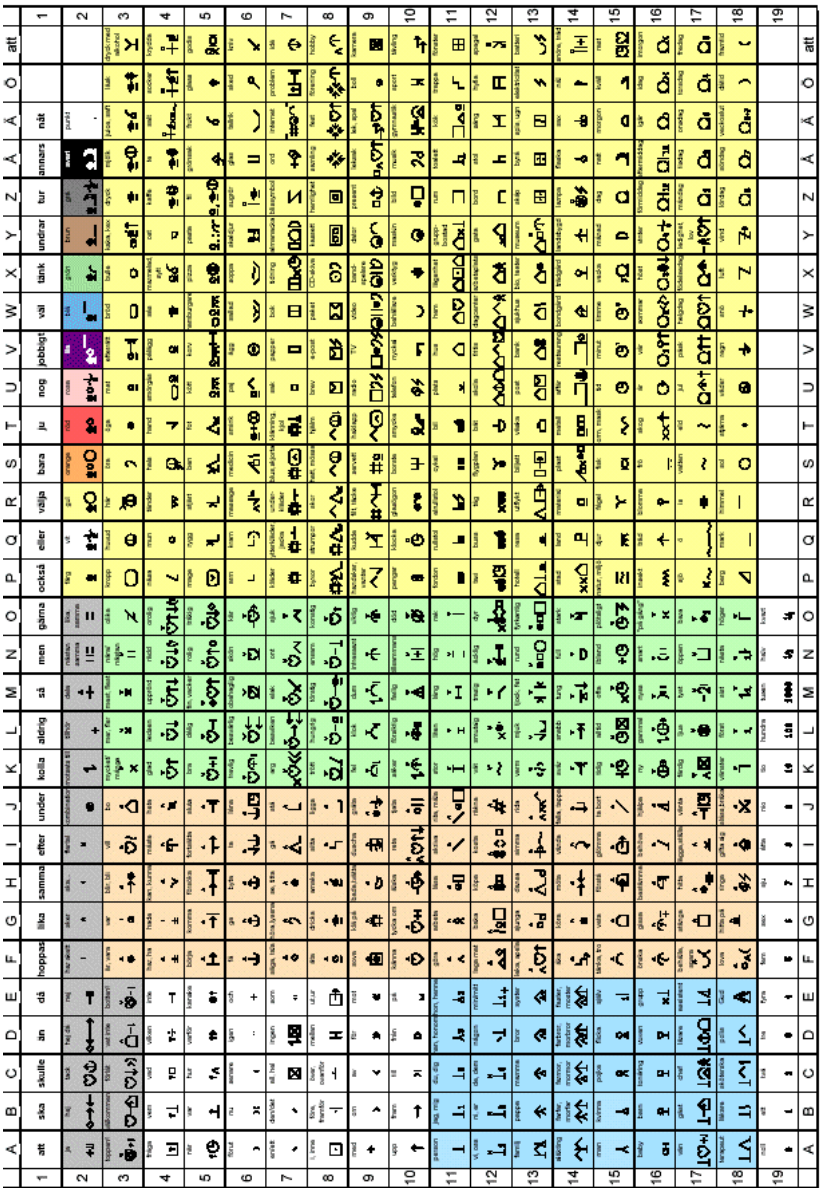


Figure A.1: Marika's Bliss map.

Appendix B

The **noun** "dog" has 6 senses in WordNet.

1. **dog**, domestic dog, Canis familiaris -- (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds; "the dog barked all night")

2. frump, **dog** -- (a dull unattractive unpleasant girl or woman; "she got a reputation as a frump"; "she's a real dog")

3. **dog** -- (informal term for a man; "you lucky dog")

4. cad, bounder, blackguard, **dog**, hound, heel -- (someone who is morally reprehensible; "you dirty dog")

5. pawl, detent, click, **dog** -- (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)

6. andiron, firedog, **dog**, dogiron -- (metal supports for logs in a fireplace; "the andirons were too hot to touch")

The **verb** "dog" has 1 sense in WordNet.

1. chase, chase after, trail, tail, tag, **dog**, go after, track -- (go after with the intent to catch)

Figure B.1: The different senses for the word dog as specified in WordNet.

Appendix C

```
<macromedia-extension name="SymbolExtension" VERSION="1.0.12"
TYPE="command">
<!-- List the required/compatible products -->
<products>
<product name="Dreamweaver" VERSION="6" required="true" />
</products>
<author name="Karolin Bengtsson & Mia Nyström" />
<description>
<![CDATA[This command creates a new menu in the menu bar.]]>
</description>
<ui-access>
<![CDATA[This command is found in the menu bar.]]>
</ui-access>
```

```
<files>
<file SOURCE="SymbolExtension.htm"
DESTINATION="$dreamweaver/Configuration/Menus/Symbols" />
<file SOURCE="GettingSymbols.htm"
DESTINATION="$dreamweaver/Configuration/Menus/Commands" />
<file source="CC.htm"
destination="$dreamweaver/Configuration/Translators" />
<file source="dog.gif"
destination="$dreamweaver/Configuration/Objects/Symbols" />
</files>
```

```
<configuration-changes>
<menu-insert insertAfter="DWMenu_Commands">
<MENU name="Symbols" id="DARTMenu_Symbols"/>
</menu-insert>
<menu-insert appendTo="DARTMenu_Symbols">
<MENUITEM name="Insert static symbols"
id="DARTMenu_InsertStaticSymbols"/>
</menu-insert>
<menu-insert insertAfter="DARTMenu_InsertStaticSymbols">
<MENUITEM name="Insert adaptive symbols"
id="DARTMenu_InsertAdaptiveSymbols"
file="Menus/Symbols/SymbolExtension.htm"/>
</menu-insert>
</configuration-changes>
</macromedia-extension>
```

Figure C.1: Different parts of the extension specification file, .mxi.

```
/* Every implementation of a Javascript function must have this
signature */

  JSBool launchGUI(JSContext *cx, JSObject *obj){
    // C-code
  }
```

Figure C.2: The signature of a JS Native function. The JSContext and JSObject have to be in the signature and on top of these it is possible to add more parameters.

```
/* Dreamweaver calls MM_Init when the library is loaded. */
void MM_Init(){

  /* Declare the Javascript function. */

JS_DefineFunction((char*)"launchGUI",(JSNative)launchGUI,(unsignedin
t)0);
}
```

Figure C.3: Define a C-function to Macromedia Dreamweaver MX; the definition signature tells the name of the native function and how many parameters the function has.

```
// Find the class.
cls = (*env)->FindClass(env, "symbols/DynamicTextEditor");

// Get the constructor method and create a new object.
mid = (*env)->GetMethodID(env, cls, "<init>", "()V");
javaobj = (*env)->NewObject(env, cls, mid);

// Get the method to prepare the GUI and call it.
mid = (*env)->GetMethodID(env, cls, "prepareGUI", "()V");
(*env)->CallVoidMethod(env, javaobj, mid);

// Get the method to initialise and launch the GUI and call it.
mid = (*env)->GetMethodID(env, cls, "initialiseGUI", "()V");
(*env)->CallVoidMethod(env, javaobj, mid);
```

Figure C.4: Example code of how to create an instance of a Java Object from C code, and how to call its methods. The parameter <init> means that the constructor is requested, and the parameter ()V means that the function has no parameters and that it is a void function i.e. returns a void.

```
class ExampleClass {
   static {
      System.loadLibrary("library");
   }
   public native void function(parameters);
}
```

Figure C.5: Example code of the Java Native Class

```
ExampleClass a = new ExampleClass();
a.function();
```

Figure C.6: Example code of creating an instance of ExampleClass, and how to call native methods.

```
JNIEXPORT void JNICALL Java_ExampleClass_function(JNIEnv *,
jobject);
```

Figure C.7: Signature of native method in the automatically generated header *file*.

```
OpenDatabase (int, HANDLE *);
GetData (HANDLE, int, int, HANDLE *);
DrawH (HANDLE, HDC, RECT, HANDLE, DWORD);
ReleaseData (HANDLE, HANDLE);
CloseDatabase (HANDLE);
```

Figure C.8: Methods in the Symbol for Windows API used for accessing symbols.