Japanese HWR

Steven B. Poggel
steven.poggel@gmail.com

February 10, 2010

Contents

0.1 Abstract . . . 6

1 Introduction **7**
1.1 Motivation . . . 7
1.1.1 Integrating NLP and e-learning . . . 7
1.1.2 Another subsection with a yet unknown title . . . 8
1.2 A CJK environment . . . 8
1.3 Running text . . . 8

2 Japanese Script **9**
2.1 A Short History of the Japanese Script . . . 9
2.1.1 Historical Development . . . 9
2.2 The Modern Japanese Writing System . . . 10
2.2.1 Kana かな . . . 11
2.2.1.1 Hiragana ひらがな . . . 12
2.2.1.2 Katakana カタカナ . . . 13
2.2.2 Composition of the Kanji 漢字 . . . 13
2.2.2.1 Typology . . . 13
2.2.2.2 Radicals . . . 15
2.2.2.3 Readings . . . 16
2.2.3 Structure of the Japanese Writing System . . . 17
2.2.4 Machine Writing of Japanese . . . 18
2.3 Difficulties of the Japanese Script for Learners . . . 18
2.3.1 Learner's Problems . . . 19

3 On-Line Handwriting Recognition **21**
3.1 Introduction . . . 21
3.2 Handwriting Features . . . 21
3.2.1 Handwriting Properties of Latin Script . . . 21
3.2.2 Handwriting Properties of East Asian Scripts . . . 22
3.3 Automated Recognition of Handwriting . . . 22
3.3.1 Short History of Handwriting Recognition . . . 22
3.3.2 Pattern Recognition Problems . . . 23
3.3.2.1 Related Problems . . . 23
3.3.2.2 Problem of Similar Characters . . . 23
3.3.3 Hardware Requirements . . . 24
3.3.4 Recognition vs Identification . . . 24
3.3.5 Interpretation of Handwriting . . . 24
3.3.6 On-Line vs. Off-Line Recognition . . . 25
3.3.6.1 Basic Features of On-Line Recognition . . . 25
3.3.6.2 Basic Features of Off-Line Recognition . . . 25
3.3.6.3 Similarities and Differences . . . 25
3.3.7 HWR of Hànzì (汉字) and Kanji (漢字) . . . 25
3.4 A Typical On-Line HWR Application . . . 26
3.4.1 Data Capturing . . . 26
3.4.1.1 Sampling . . . 26
3.4.2 Preprocessing . . . 27
3.4.2.1 Segmentation . . . 27
3.4.2.2 Noise Reduction . . . 27
3.4.2.3 Normalisation . . . 28

3.4.3 Character Recognition . . . 28
3.4.3.1 Feature Analysis . . . 29
3.4.3.2 Zone Sequences . . . 29
3.4.3.3 Stroke Direction . . . 29
3.4.3.4 Curve Matching . . . 29
3.4.3.5 HMM Analysis . . . 30
3.4.4 Postprocessing . . . 30
3.5 Overview of a Typical OLCCR system . . . 30
3.5.1 Character Segmentation . . . 31
3.5.2 Preprocessing . . . 31
3.5.3 Pattern Description . . . 31
3.5.3.1 Statistical Character Representation . . . 32
3.5.3.2 Structural Character Representation . . . 32
3.5.3.3 Statistical-Structural Character Representation . . . 32
3.5.4 Classification . . . 32
3.5.4.1 Coarse Classification . . . 33
3.5.4.2 Structural Classification . . . 33
3.5.4.3 Probabilistic Classification . . . 34
3.5.5 Postprocessing . . . 35
3.5.5.1 Contextual Processing During Recognition . . . 35
3.5.5.2 Contextual Processing After Recognition . . . 35

4 E-learning **37**
4.1 General E-Learning methods . . . 37
4.2 E-Learning of languages . . . 37
4.3 E-Learning of Japanese . . . 37
4.3.1 Conceptual issues . . . 37
4.3.2 Japanese e-learning software . . . 37

5 Conceptual Design of Kanji-Coach **39**
5.1 General Requirements . . . 39
5.1.1 Character Learning Aspects . . . 39
5.1.1.1 Character Repetition . . . 39
5.2 Tackling the Specific Difficulties of the Japanese Script . . . 39
5.3 Integration of HWR Into the Learning Process . . . 39
5.4 Use Cases . . . 39
5.5 HWR Applied to E-learning of Japanese Kanji . . . 40
5.5.1 Integration of HWR Into E-Learning Application . . . 40
5.6 Handling Errors . . . 40
5.6.1 Sources of Error . . . 40

6 Technical Design of the Application **41**
6.1 System Architecture . . . 41
6.1.1 Global Architecture . . . 41
6.1.2 System Data Flow . . . 43
6.1.2.1 Communication . . . 43
6.1.2.2 Recognition Data Flow . . . 44
6.1.2.3 Learning Data Flow . . . 44
6.1.3 Software Modules . . . 45
6.1.3.1 Handwriting Data Input View . . . 45
6.1.3.2 Main View . . . 45
6.1.3.3 Web Service . . . 45
6.1.3.4 Recognition Module . . . 45
6.1.3.5 Learning Module . . . 46
6.2 Framework and Devices . . . 46
6.2.1 Operating System . . . 46
6.2.2 Framework . . . 46
6.2.3 Desktop Computer . . . 47
6.2.4 Pen Input Device . . . 47

7 Handwriting Recognition Engine **49**
7.1 Data Capturing . . . 49
7.1.1 Writing Surface . . . 49
7.1.1.1 Writing Surface GUI . . . 49
7.1.1.2 Writing Surfaces Background Module . . . 49
7.2 Data Format . . . 50
7.2.1 Requirements of the Data Format . . . 50
7.2.2 Existing Handwriting Formats . . . 50
7.2.2.1 The Microsoft Ink Serialized Format (*ISF*) . . . 50
7.2.2.2 InkML . . . 51
7.2.2.3 The Standard UNIPEN Format . . . 52
7.2.2.4 Handwriting Dataset (*hwDataset*) . . . 53
7.2.2.5 The UNIPEN XML Format (*UPX*) . . . 53
7.2.3 Data Format Description . . . 55
7.2.3.1 Point Data Format . . . 55
7.2.3.2 Stroke Data Format . . . 55
7.2.3.3 Radical Data Format . . . 56
7.2.3.4 Character Data Format . . . 57
7.3 Database . . . 58
7.3.1 Database Organisation . . . 58
7.3.2 Alternative Database Structures . . . 60
7.3.2.1 A Unified File Format . . . 60
7.3.2.2 Relational Database . . . 60
7.4 Recognition Architecture . . . 61
7.5 Stroke Recognition Process . . . 61
7.5.1 Advanced Point Lists . . . 61
7.5.2 Normalisation . . . 61
7.5.2.1 Boxing . . . 61
7.5.2.2 Scaling . . . 62
7.5.3 Recognition of Straight Strokes . . . 62
7.5.3.1 Identification of a Straight Stroke . . . 62
7.5.3.2 Straight Stroke Matching . . . 62
7.5.4 Curve Handling . . . 63
7.5.4.1 Detecting Curves in Strokes . . . 63
7.5.4.2 Feature Extraction for Curved Strokes . . . 64
7.5.5 Dynamic Time Warping . . . 64
7.5.5.1 Standard DTW . . . 64
7.5.5.2 3-Dimensional DTW . . . 65
7.5.6 Stroke Matching Summary . . . 65
7.6 Radical Recognition Process . . . 65
7.6.1 Radical Recognition Features . . . 65
7.6.2 Exploitation of Stroke Recognition . . . 66
7.7 Character Recognition Process . . . 66
7.8 Error Handling . . . 67
7.8.1 Error Recognition . . . 67
7.8.2 Error Processing . . . 67

8 Implementation and Evaluation **69**
8.1 Implementation Details . . . 69
8.2 Evaluation of the HWR . . . 69
8.2.1 Evaluation Metrics . . . 69
8.2.2 DTW vs. 3-D DTW . . . 69
8.3 Evaluation of E-Learning Application with Integrated HWR . . . 70
8.4 Evaluation of the Error Hints . . . 70

9 Conclusions **71**

10 Outlook **73**

A Japanese Language **75**
A.1 Kanji timeline 75
A.2 Kana かな 75
A.2.1 Hiragana ひらがな 75
A.2.2 Katakana カタカナ 75

B Technicalities **77**

0.1 Abstract

In this work I present and application that uses state of the art Chinese/Japanese handwriting recognition methods in order to provide an Kanji teaching application with an error correction.

Conceptually, the application is an e-learning environment for Japanese characters, intended for the foreign learner of the Japanese language. In order to provide more than a multiple choice method, like most other systems, the application contains a handwriting recognition engine that can be used preferably with a hand-held device like a PDA, but generally any stylus input device.

Chapter 1

Introduction

1.1 Motivation

In the history of Computational Linguistics there have been a several attempts to integrate natural language processing techniques with existing technologies. This work is one just like that. Concretely, we will try to create a handwriting recognition for Japanese Kanji. That seems interesting, because Kanji is an iconographic writing system, thus handwriting recognition (HWR) can follow different patterns than in alphabetical writing systems like latin.

Studying Japanese language is a complex task, because a new learner has to get used to a new vocabulary that - coming from a European language - has very little in common with the vocabulary of his mother tongue, unlike in European languages where quite often there are several intersections. The learner also needs to learn a new grammar system. Broadly speaking, most of the central European languages follow a subject-verb-object (SVO) structure. Japanese follows a subject-object-verb (SOV) structure therefore creating additional difficulty, comparable with German subclause structures that are a source of error for learners of German. Yet, the most notable difference for a language learner with a central European mother tongue is of course the writing system. The Japanese writing system uses three different scripts. The Kana scripts Hiragana and Katakana are syllabic, each character represents a syllable. Each syllable consists of either a vowel, a consonant and a vowel, or a consonant cluster and a vowel. Hiragana and Katakana represent roughly the same set of syllables and both have around 40-50 characters that can be modified with diacritics and thus yield additional syllable representations. Therefore, these scripts are a hurdle, but relatively unproblematic, due to their limitation in number of characters. Besides, they look quite distinct, so there is the problem of confusing one character with another, but this is limited to a relatively short period of learning those characters.

Kanji, however, is an iconographic writing system that has around 2000 characters, which are built up of around 200 subunits called 'radicals'. So one part of the complexity lies in the number of characters. The other part of the complexity lies in the general concept of representing an idea or concept with a character instead of representing the phonemes of the spoken language with graphemes in connection with some language specific pronunciation rules. Another difficulty lies in connecting the characters with their pronunciations. Most characters have multiple pronunciations and for a language learner, studying Japanese vocabulary is a double or triple task compared to languages using a Latin or at least an alphabetic writing system. Therefore, the two tasks of learning the Kanji and studying the vocabulary together can epitomise a very high learning curve. A subordinated issue connected to that is that quite often subjectively 'simple' vocabulary comes with complex Kanji. Some e-learning applications have taken on that issue by creating a learning environment in which a learner can connect learning vocabulary with studying the Kanji.

xxx: see santosh2009: the statement of need

1.1.1 Integrating NLP and e-learning

In this project, we would like to approach the issue of studying Kanji in an e-learning application. The novelty about it is a handwriting recognition that gives the learner the ability to actually practise writing the Kanji, instead of the rather limited multiple choice recognition that most other applications use.

1.1.2 Another subsection with a yet unknown title

1.2 A CJK environment

Rather than selecting a CJK font as the main document typeface, you might want to define a CJK environment for text fragments used in the midst of a document using a normal Roman font. This allows me to say `\begin{CJK}`東光`\end{CJK}` to generate 東光, without putting the whole paragraph into the Far Eastern font. Or I could define a command that takes the CJK text as an argument, so that `\cjk{`北京`}` produces 北京. It's that easy!

1.3 Running text

コンピューターは、本質的には数字しか扱うことができません。コンピューターは、文字や記号などのそれぞれに番号を割り振ることによって扱えるようにします。ユニコードが出来るまでは、これらの番号を割り振る仕組みが何百種類も存在しました。どの一つをとっても、十分な文字を含んではいませんでした。例えば、欧州連合一つを見ても、そのすべての言語をカバーするためには、いくつかの異なる符号化の仕組みが必要でした。英語のような一つの言語に限っても、一つだけの符号化の仕組みでは、一般的に使われるすべての文字、句読点、技術的な記号などを扱うには不十分でした。

これらの符号化の仕組みは、相互に矛盾するものでもありました。二つの異なる符号化の仕組みが、二つの異なる文字に同一の番号を付けることもできるし、同じ文字に異なる番号を付けることもできるのです。どのようなコンピューターも（特にサーバーは）多くの異なった符号化の仕組みをサポートする必要があります。たとえデータが異なる符号化の仕組みやプラットフォームを通過しても、いつどこでデータが乱れるか分からない危険を冒すことのなるのです。

Chapter 2

Japanese Script

The Japanese writing system has a long history. It goes back to around 800 A.D. The Japanese script is in fact a writing system, as Japanese is denoted in a combination of three different scripts: *Hiragana*, *Katakana* and *Kanji*. Kanji is a conceptual script, where each character bears the meaning of one or more semantic concepts and represents morphemes. Hiragana and Katakana are both syllabic scripts, and the individual characters do not bear reference to concepts or even words, but merely to phonological units, usually two phonemes.

In this chapter, the development of the script will be reviewed in section 2.1. In section 2.2 the current Japanese writing system will be exemplified, with a focus on the Kanji in section 2.2.2. Hiragana and Katakana will be reviewed in section 2.2.1, which centres around the Kana scripts. Machine processing of the different Japanese scripts and the difficulties that go along will be demonstrated in section 2.2.4. The difficulties of learning to use the Japanese script will be illustrated in section 2.3.

2.1 A Short History of the Japanese Script

The historical development of the Japanese script is tightly connected to the history of the Kanji characters. Kanji, in Japanese 漢字 (Jap. pron. カンジ / Kanji; Eng. lit. *Han characters*) refers to the 'characters of the Han', meaning the Han Dynasty (206 B.C.-220 A.D.; simplified Chinese: 汉朝; traditional Chinese: 漢朝) (Foljanty 1984). In Mandarin the same characters are referred to as *Hànzì* (simplified Chinese: 汉字; trad. Chinese: 漢字). Note, that the first character 漢 (Chin. 'han', Jap. 'kan', Eng. 'Han') of both the words *Han dynasty* and *Kanji* is identical in Japanese and traditional Chinese, even though it has a different reading in the Chinese and Japanese language. In traditional Chinese the character with the same meaning (汉) has a different shape. This apparent oddity will be explained in greater detail in section 2.1.1.

2.1.1 Historical Development

The Kanji script as developed and coined by the Han is in principle still valid today. It is used alone or in combination with phonetic spelling in China, Japan, Taiwan, Hong Kong. In Vietnam it was used before it was replaced with the Vietnamese alphabet (Viet.: 'quốc ngữ', Eng. lit. 'national language', Eng. 'national script'), a script based on the Latin alphabet. In South Korea the Han characters were in use until they were replaced with Hangul (Kor. with Han characters 韓國語; Eng. 'Korean') (Foljanty 1984).

The Kanji characters were brought to Japan by Koreans living in Japan around 300-400 A.D. Since the Kanji were used by the Koreans to write Hangul they also used it to write Japanese. There was no other Japanese script before that time. Reports about an original Japanese script called *Jindai Moji* (Jap. 神代文字; Eng. 'scripts of the age of the gods') could not be proved. They are now assumed to be a political and speculative invention by Japanese Nationalists in the early 19th. century (Foljanty 1984). According to (Lange 1922) the *Kogo Shūi* (Jap. 古語拾遺; a historical record of the Inbe clan), which was written around 800 B.C. denies the presence of a Japanese native script before the introduction of the Han characters. However, the questions seems irrelevant in the sense, that no longer text or document has been found, written in that script.

In the Christian year 712 an ancestral act of writing was performed at Japanese emperor Temmu's court. Hieda no Are, a member of the guild of the *Kataribe* or reciters, basically a Japanese Griot, dictates the *Kojiki* (Jap. 古事記; Eng. 'Record of Ancient Matters') to Ō no Yasumaro. Ō no Yasumaro wrote the Kojiki, which is not the first written document found in Japan, however it is Japan's oldest attempt to write down spoken Japanese (Grassmuck 1997; Chamberlain 1982).

At the time the Han characters were first used to write Japanese, they were already a developed script, more than 1,000 years old, as they stabilised to their modern form within the Han period[1]. The first Chinese characters were found on oracle bones from the Shang Dynasty (Chinese 商朝), which ruled over China some 500 to 600 years within the time period between 1600 B.C. and 1046 B.C. (Grassmuck 1997; Guo et al. 2000).

According to the Kojiki, a scholar called Wani (Jap. 王仁) from Korea brought two foundational Chinese books to Japan, the *Lunyu* (Simplified Chin. 论语; trad. Chinese: 論語; Eng. 'Analects'), also known as *The Analects of Confucius* and the *Qianziwen* (Chin./Jap. 千字文; Jap. pron. センジブン/senjibun; Eng. 'The Thousand Character Classic'), which is a Chinese poem used as a primer for teaching Chinese characters to children. It contains exactly one thousand unique characters (Grassmuck 1997). However, (**?**)Lange1922) demurs strongly against this view and assumes an evolving adoption of the Kanji in Japan. This view seems more plausible, since it has been proved that Japan had contact to Korea even in the time before common era and had contact to China at least from the first century A.D. The Chinese language comprehends more than 40,000 Hànzì characters lexicographically. Only around 25% of those including about 250 *Kokuji* (Jap. 国字; Eng. 'national characters') are in Japanese dictionaries. Only around 2,000-3,000 of those are part of the common characters (Foljanty 1984).

The Japanese Ministry of Education issued a list of 1,850 standard Kanji in 1946 under the name of *Tōyō kanjihyō* (Jap. 当用漢字表; Eng. 'list of Kanji for general use'). The list of Tōyō Kanji was slightly revised and extended in 1981 and comprised 1,945 Kanji as the Jōyō Kanji (Jap. 常用漢字; Eng. 'often used Kanji') (Foljanty 1984). As of 2010 a revised list of 2,131 characters is in official use (Noguchi 2009).

In China there had been a spelling reform in the 1950s, affecting many of the general use characters, resulting in simplified Chinese. In Japan, the Ministry of Education issued it's own reform when the Tōyō Kanji list was introduced. However, the Japanese reform affected a smaller set of characters of only a few hundred and resulted in Shinjitai (Jap. shinjitai: 新字体; Jap. kyūjitai: 新字體; Jap. pron. シンジタイ/shinjitai; Eng. 'new character form'), which replaced the Kyūjitai (Jap. shinjitai: 旧字体; Jap. kyūjitai: 舊字體; Jap. pron. キュウジタイ/kyūjitai; Eng. lit. 'old character forms'). This explains how some characters are still identical in traditional Chinese and Japanese, because they were not affected by any spelling reform, like the aforementioned 漢 (Jap. pron. カン/kan; Chin. pron. 'hàn'), while other characters are different, like the simplified Chinese 'hàn': 汉. Henceforth, and throughout this document, all Japanese characters are in the new character form shinjitai.

2.2 The Modern Japanese Writing System

The Japanese writing system has a complex structure. The three scripts *Hiragana* (section 2.2.1.1), *Katakana* (section 2.2.1.2) and *Kanji* (section 2.2.2), are combined to one writing system. Each script has its task within the system:

- Kanji are used to write lexical morphemes, i.e. content-bearing morphemes.
- Katakana are used to transcribe foreign words, borrowings and nonstandard areas.
- Hiragana are used to write grammatical morphemes and anything else that is not written in one of the other two scripts, e.g. the spoken syllables of a word that should be written with a Kanji character unknown to the writer of that word.

The actual writing system is mainly based on Kanji and Hiragana, catenated to Kanji-Kana blended writing.

[1] Also see time line in section A.1.

(1) a. マリア：山田さん、「火の鳥」というアニメをもう見ましたか。
Maria: Yamada-San, [Firebird] say anime OBJ-PARTICLE already seen QUESTION-PARTICLE
'Maria: Ms Yamada, say, have you seen the Firebird cartoon yet?'
Taken from (Katsuki-Pestemer 2006)

b. マリア：山田さん、「火の鳥」というアニメをもう見ましたか。

In example (1a), the blending of the different scripts can be seen:
Both the foreign name マリア (Eng. 'Maria') and the borrowing アニメ (*anime*, Jap. short for Eng. 'animation') are written in Katakana. Kanji are used for:

- The Japanese name 山田 (*Yamada*).
- The nouns 火 (Eng. 'fire') and 鳥 (Eng. 'bird').
- The verb stem 見 (Eng. 'see').

The rest is written in Hiragana:

- The politeness ending さん (*san*; Eng. equiv. 'Mr/Ms/Mrs') for addressing a person with their name.
- The genitive particle の (*no*) between 火 (Eng. 'fire') and 鳥 (Eng. 'bird'), to yield 火の鳥 (Eng. 'Firebird').
- The interjection とうい (Eng. 'say').
- The object particle を (*wo*).
- The adverb もう (Eng. 'already').
- The past tense conjugation and politeness ending of the verb ました (*mashita*).
- The question particle か (*ka*).

Three different scripts are used next to each other in one sentence, indistinguishable for the untrained eye. Example (1b) shows the sentence as it is printed in (Katsuki-Pestemer 2006). Without prior knowledge of the different Japanese scripts it is hard to even distinguish the individual word tokens, as blanks are usually absent in Japanese writing. Other than actually knowing the words, which is not the usual case for a beginner of learning Japanese, often the change of script is the only way to recognise a new token. However, Kanji and Hiragana are often used within the same word, too.

Despite those complexities, other features of the Japanese writing system are simpler than in latin-based alphabetic scrips. For example, there is no capitals or lowercase letters. Each character has a reserved space of roughly the same size. In the following sections, the different scripts will be presented in greater detail. Their composition and use will be discussed.

2.2.1 Kana かな

If the Japanese had abolished the Chinese characters after formation of the syllabic scripts and used only those, studying the Japanese script would be a less complex task. (Lange 1922) reports about attempts to remove the Kanji from Japanese and use the Kana or even Latin script, so called ロマジ (*Romaji*, a Latin or 'Roman' transcription of Japanese; Eng. lit. 'Roman characters'). However, non of those attempts succeeded and both Kana scripts serve as auxiliary scripts to the predominant Kanji characters.

The Chinese characters have been used in two ways in Japanese. Firstly, in order to express the morphological content of a character, but also in order to use the sound of the character as a syllable. The characters that have been used as syllables were transformed to two separate short-hand notations. One way used cursive writing of the sound Kanji, reducing the character graphically, such that its original shape became virtually in-cognisable. This development resulted in the Hiragana script. An example of that kind of reduction is shown in figure 2.1. The other method of reduction uses only one

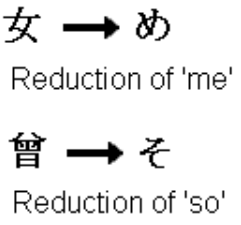

Figure 2.1: Reduction from Kanji to Hiragana

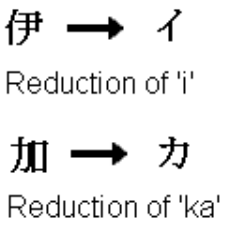

Figure 2.2: Reduction from Kanji to Katakana

of the character's Graphemes in order to represent the whole character. That reduction process (see figure 2.2) lead to the development of the Katakana script.

Using cursive or reduced characters became popular in the 9th century already. Both Hiragana ('smoothened Kana') and Katakana ('fragmented Kana') can easily be distinguished from the more complex Kanji. They represent a different linguistic content than the Kanji, namely rather a syllable than a morpheme. The fact that two parallel scripts came into existence can be explained by their use of different social groups. Hiragana are a product of the literately active court ladies, while Katakana were developed in the Buddhist seminaries. The current system knows 46 Hiragana and Katakana for identical syllables (Foljanty 1984). See appendix A.2 for a complete list of characters.

2.2.1.1 Hiragana ひらがな

Hiragana is one of the three syllabic scripts. The third one, *Hentaigana* can be neglected, as it is not in active use any more. In principle each character represents a syllable, which can contain either a vowel (like 'u', う), a consonant and a vowel (like 'ta', た) or the nasal consonant ん ('n'). Hiragana are used for any words for which no Kanji exist, like から (*kara*, Eng. 'from') or grammatical particles like the object particle を (*wo*, works like a case marker). Additionally, Hiragana are used, when the Kanji is not known to the writer or reader (Foljanty 1984).

For instance, a writer may even passively know a Kanji, but not be able to actively produce it. Say the Kanji in question was 新 (Jap. pron. atara/あたら and shin/シン; Eng. 'new'). Example (2) shows first the spelling with the Kanji character in (2a), where the Hiragana parts are underlined. Then the alternative spelling without the Kanji character in (2b) (with blanks in between, in order to visualise the individual characters and their readings). The underlined last part is identical, just the Kanji character has been replaced with an alternative Hiragana spelling.

(2) a. 新しい
atarashii
'new'

b. あ た ら し い
a ta ra shi i
'new'

Hiragana can be modified with

- **Dakuten** (濁点, Jap. pron. ダクテン/dakuten; Eng. 'turbid'; Jap. coll. *ten-ten*, Eng. 'dot dot') for syllables with a voiced consonant phoneme. The dakuten glyph (゛) resembles a quotation mark and is directly attached to a character (Foljanty 1984).

- **Handakuten** (半濁点, Jap. pron. ハンダクテン/handakuten; Eng. 'half-turbid'; Jap. coll. *maru*, Eng. 'circle') for syllables with a /p/ morpheme. The glyph for a 'maru' is a little circle (゜) that is directly attached to a character (Foljanty 1984).

(3) a. た → だ, き → ぎ, ふ → ぶ, へ → べ, そ → ぞ
ta → da, ki → gi, fu → bu, he → be, so → zo

b. は → ぱ, ひ → ぴ, ふ → ぷ, へ → ぺ, ほ → ぽ
ha → pa, hi → pi, fu → pu, he → pe, ho → po

Example (3) shows the dakuten in use. In (3a) the turbidity, i.e. the transformation from a fortis to a lenis consonant is exemplified. The dakuten can only be applied to characters that begin with the sounds /k/, /s/, /t/ and /h/. For the consonants 'k', 's' and 't' this is natural in the way that the place of articulation remains the same for the consonants 'g', 'z' and 'd' and their corresponding sounds /g/, /z/, /d/, compared to /k/, /s/ and /t/. Assuming the place of articulation, a changed intensity and use or no use of the vocal cords as the natural connection between the characters with and without a *ten-ten*, there is a difference for the sound /b/:
Since there is no Hiragana character to match a /p/ sound, the characters with an /h/ sound serve as a basis to form the modified characters with a /b/ sound. Quite logically, since the Handakuten only yields a /p/ sound, the same set of characters, namely, the 'h'-row in the Hiragana character set, is modified in order to obtain the 'p'-row. In (3b) the transformations from the /h/ and /f/ sounds to /p/ are shown in a complete list[2].

2.2.1.2 Katakana カタカナ

Just like the Hiragana, the Katakana form a syllabic script. They are mainly used to transcribe foreign words like names and borrowings, like マリア (*maria*) and アニメ (*anime*) from example (1). Another use of the Katakana is called *Furigana*, little notations next to Kanji in order to indicate their reading. They were developed around the period of the Heian, from *Manyogana*, Kanji characters that were used to denote pronunciation. For example the character 加, which was used to represent the pronunciation *ka*, was shortened to カ, by leaving out the second part 口 (Hadamitzky 1995). Compare also figure 2.2. For a full table of Katakana characters see appendix A.2.2. The dakuten can be applied to Katakana as well and have the same meaning.

2.2.2 Composition of the Kanji 漢字

2.2.2.1 Typology

In order to study the Kanji and their composition, it is useful to know how they were first indented and built. Integrated as the integral part into the Japanese writing system, despite the reform and the choice different subsets of what is considered the standard character set, the characters are still mainly composed the way as intended by the scholars of the Han period.

From the religious writings on the oracle bones mentioned in section 2.1.1 a secular script emerged. In parallel, the process of graphical abstraction advanced and finished around 100-200 A.D., leaving aside the modern reforms of the 20th century. The invention of the paint-brush around 100 B.C. improved and simplified writing, also the writing surfaces in their order of appearance, bone, stone, metal, wood and then paper, brought further simplification and spreading of writing. Paper and paint-brush offered the possibility to write without hindrance and technical coincidences, therefore it was possible to standardise the characters and improve them from artistic and aesthetic viewpoints.

The Kanji can be classified according to their building principle:

[2]The /f/ sound in Japanese is 'lighter' than in English. Concretely, the place of articulation for /f/ in English is labio-dental, whereas in Japanese it is virtually bi-labial. Therefore, the phonation stream creates less friction and the allophone of the /f/ phoneme that is typical for the Japanese language sounds similar to an /h/ sound.

Figure 2.3: Kanji pictograms

Radical 1	Radical 2	Result	Meaning
日 ('sun')	月 ('moon')	明 ('bright')	both the sun and the moon are 'bright'
人 ('man')	木 ('tree')	休 ('rest')	a man is 'resting' beside a tree
田 ('rice field')	力 ('power')	男 ('man, male')	'a man' is powerful on the rice field
女 ('woman')	子 ('child')	好 ('love')	a woman 'loves' a child
木 ('tree')	木 ('tree')	林 ('wood, grove')	two trees make a 'wood'
木 ('tree')	林 ('wood')	森 ('forest')	three trees form a 'forest'
門 ('gate')	日 ('sun')	間 ('between')	the sun can be seen 'between' the doors of the gate

Table 2.1: Kanji ideograms

Class radical	Sound radical /ki/	meaning
土 ('earth')	奇	埼 'spit, promontory, cape'
山 ('mountain')	奇	崎 'promontory, cape, spit'
石 ('stone')	奇	碕 'cape, promontory, spit'
王 ('jade')	奇	琦 'gem, precious stone'
糸 ('thread')	奇	綺 'figured cloth, beautiful'
馬 ('horse')	奇	騎 'riding on horses'
宀 ('roof')	奇	寄 'to gather'
金 ('metal')	奇	錡 'cauldron, chisel' (Chinese only)

Table 2.2: Kanji phonograms

1. **Pictograms** are graphically simplified images of real artefacts. The examples in figure 2.3 after Kano et al. (1990) show the graphical reduction process. Pictograms are only a small minority among the Kanji, their number ranges around 120. Another 100 pictograms appear as a part of more complex characters (Foljanty 1984).

2. **Ideograms** are combinations of two or more pictographical characters. They often bear a more abstract meaning than a simple pictogram. The abstract meaning of the complex character is meant to be associated with the content of the individual parts. The number of ideograms is fairly small, too. Abstract terms like 'top' (Jap. 上, pron. うえ/ue), 'bottom' (Jap. 下, pron. した/shita), 'left' (Jap. 左, pron. ひだり/hidari), 'right' (Jap. 右, pron. みぎ/migi) and numbers like 'one' (Jap. 一, pron. いち/ichi), 'two' (Jap. 二, pron. に/ni), 'three' (Jap. 三, pron. さん/san), 'four' (Jap. 四, pron. し/shi), 'five' (Jap. 五, pron. ご/go) and so forth can be regarded as parts of the ideograms (Foljanty 1984). See table 2.1 (after (Kano et al. 1990)) for some examples on Kanji pictograms.

3. **Phonograms** are combinations of two Kanji characters. One of those refers to a concept class (for *class characters* or *radicals*, see section 2.2.2.2), while the other character exclusively bears a phonetic value. The content of the second part of a phonogram is not relevant and can be ignored. In table 2.2 the character 奇 (Jap. pron. き/ki, Eng. 'strange') is used for the purpose of pronunciation only (/ki/), while the radical defines an object class. Object classes can be categories like 'human and human actions', 'metal', 'horse', 'roof / under a roof' etc. The semantic identity within the *Morphogram* is assembled with two reference figures. The pronunciation part is identical for all characters, it serves as a selection criterion within a semantic class selected by the class radical (Foljanty 1984).

As a character type, phonograms are predominant among the Hànzì. Therefore, the phonogram concept, including the radical concept, was transferred to all Chinese characters. Pictograms that are class radicals themselves, are interpreted as characters with an empty sound radical. As the phonograms are historically the last development step of the Han characters, they constitute a different quality in the Chinese script. Phonograms mark the transition between a non-linguistic pictographic script that does not represent linguistic units, but rather images of objects, to a linguistic script. In principle, there is no difference to an alphabetical or syllabic script. Except, morphemes are represented instead of phonemes or syllables. However, one character often denotes more than one morpheme (Foljanty 1984).

In Japanese, basically the same relation between the Kanji characters and morphemes can be observed. However, the correspondence between the morphemes and the syllables (and thus the characters and syllables) is often missing. Since Chinese has a monosyllabic morpheme structure, a one-to-one correspondence between morpheme, character and syllable can be observed. Congruence of character, morpheme and syllable in Japanese can only be found for Chinese borrowings, but not all of them. The original Japanese vocabulary has multisyllabic morphemes, therefore some impreciseness arises in the graphical reproduction of the morpheme structure (Foljanty 1984).

2.2.2.2 Radicals

In section 2.2.2.1 the function of the class characters or Radicals has been mentioned. The Radicals usually derive from pictograms. The Chinese systematics with 214 Radicals is used in Japan, as well. Among the 1945 Jōyō Kanji, only around 50% of the 214 Radicals is in autonomous use. 22 of them denominate empty classes, 46 denominate classes of only one Kanji. The Radicals are grouped according to their graphical position inside a Kanji. Figure 2.4 shows the groupings of the Radicals after (Foljanty 1984). The groupings of the Radicals account for a preference of certain Radicals to appear in a certain category. The selection of the class Radical is primarily based on semantic criteria. For example, the character 杉 (Jap. pron. スギ/sugi; Eng. 'cedar') both graphemes have the ability to serve as a class Radical. Therefore there is a choice of using either the Radical 木 (Jap. pron. キ/ki; Eng. 'tree') in the *hen* position, or the Radical 彡 (Jap. pron. カミカザリ/kamikazari; Eng. 'hair ornament') in the *tsukuri* position (see Figure 2.4). This is very pictographic, since the leaves of a cedar are long and thin, (see figure 2.5), therefore the character is most likely an ideogram. Now, the categorisation is purely a semantic choice. The cedar has certainly more qualities of a tree than qualities of hair, therefore, the class radical is 木 (tree) and not 彡 (hair). (Hadamitzky 1995) files it under the 木 (tree) class Radical.

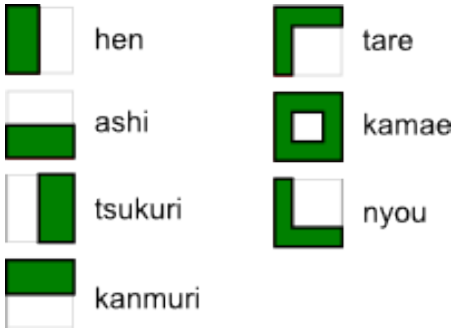

Figure 2.4: Radical positions in Kanji characters

Figure 2.5: Cedar leaves

2.2.2.3 Readings

Each character has at least one reading. The Kanji script, being a morphemic script does not have a correspondence from character to spoken sound in speech, like alphabetic scripts. Certainly, alphabetic scripts are often far from a one-to-one correspondence between sound and letter, however, the Kanji have their very own ways. There are two main readings of the Kanji:

- *On'yomi* (音読み, Jap. pron. オンヨミ/onyomi; Eng. 'sound reading'), which roughly based on its original Chinese reading.
- *Kun'yomi* (訓読み, Jap. pron. クンヨミ/kunyomi; Eng. 'concept reading, native reading'), which represents the original Japanese word that happens to be written with a Chinese character. The Kun'yomi are further classified into different types of readings.

Because of this, the Kanji that have been developed in Japan only have a Kun-reading. A bit more than half of the official Jōyō-Kanji (999 of 1945) have two readings. The Japanese Kun-reading was part of the adaption process to the Han-characters, where the character was taken for its meaning and underlaid with the Japanese word. The Sino-Japanese reading originated from Chinese borrowings. The complex Chinese pronunciation, however, was adapted to the Japanese phoneme inventory.

Take for example the Kanji 山 (Eng. 'mountain'). The original Japanese word for *mountain* is *yama* (やま), therefore the character 山 has this reading attached as a Kun-reading. The Chinese borrowing is *san* (サン), with the meaning *mountain*. The reading that is used depends on the context (Foljanty 1984). 山 is read Sino-Japanese *san* or in a phonetic assimilation *zan* in:

- 山水, Jap. pron. サンスイ/**san**sui, Eng. 'landscape'
- 火山, Jap. pron. カザン/ka**zan**, Eng. 'volcano'
- 下山, Jap. pron. ゲザン/ge**zan**, Eng. 'descent'

- 富士山, Jap. pron. フジサン/fuji**san**, Eng. 'Mount Fuji'[3]

山 is read in the Japanese Kun-reading in:

- 小山, Jap. pron. コヤマ/ko**yama**, Eng. 'hill'
- 山山, Jap. pron. ヤマヤマ/**yamayama**, Eng. 'mountains'

(Hadamitzky 1995)

2.2.3 Structure of the Japanese Writing System

Having demonstrated the Hiragana in 2.2.1.1, the Katakana in 2.2.1.2 and the Kanji in section 2.2.2, it is now possible to report about the structure of the writing system as such. It now becomes apparent, why a sentence like the one presented in example (1) in section 2.2 is written and spelt the way it is.

(4) 帰りたくない
kaeritakunai
go-home-want-not

'I do not want to go home'

An important part of the blended writing in Japanese are the *Okurigana* (送り仮名, Jap. pron. オクリガナ/okurigana; Eng. 'accompanying characters'). Okurigana are Kana that form a word together with a Kanji character (Lange 1922). For instance, the word in example (4) has only one Kanji character: 帰 (Jap. pron. カエ/kae; Eng. 'go-home'), while the other characters are all Kana characters, that are part of the same word and modify the verb.

- り (*ri*): Flexion of the verb used for *tai*
- たく (*taku*): derivation of *tai* = want
- ない (*nai*): Negation ending

Knowing the two syllabic scripts by heart helps distinguish the Kanji from them. Therefore it is possible to use the three scripts next to each other, even without blanks. The Okurigana further help distinguish different versions of the same Kanji, as they sometimes even partially describe the sounds of a verb stem.

(5) a. 変ーえる
ka-eru
change

'to change something' (transitive)

b. 変ーわる
ka-waru
change

'to change oneself' (reflexive)

Compare for example (5a) and (5b). The Japanese verb ending is る (Hiragana: *ru*). A part of the verb stem are expressed with Hiragana, too. In (5a) it is the え (Hiragana: *e*) while in (5b) it is the わ (Hiragana: *wa*). The Kanji are further used to write nouns, the Hiragana are used for particles and other grammatical functions, while the Katakana are used for foreign words.

[3] 'Mount Fuji' is often called 'Fujiyama' in several languages. Thus, the Kun-reading *yama* of 山 has emerged as a borrowing into other languages, even though in Japan the On-reading is used in the particular case of 'Fuji san'.

2.2.4 Machine Writing of Japanese

Machine processing of the Japanese scripts has been an issue, ever since humans started to automate their writing. Lange (1922) reports in the foreword to the first edition (1896) of his work about difficulties during the printing process. Firstly, there was only one printing office in Germany that was able to print these kind of things at all. Also, there were technical issues, like incompatible letter size and missing characters among the letters available. The manufacturing of the appropriate letters delayed the publication.

> Bei der Drucklegung des Werkes in der Reichsdruckerei, der einzigen Druckerei in Deutschland, welche dergleichen zu drucken im stande ist, stellten sich leider einige Übelstände heraus. Einerseits sind die Kana-Zeichen, welche dieselbe besitzt, au gross, um als erklärende Zeichen bequem neben die chinesischen Zeichen gesetzt werden zu können, andererseits fehlten unter den 10000 chinesischen Zeichen, die aus Shanghai gekommen, eine Anzahl in Japan ganz gebräuchlicher Zeichen. Diese mussten entweder aus den vorhandenen Teilzeichen, die eine andere Form als die übrigen Typen haben, zusammengesetzt werden oder sie mussten erst geschnitten werden; haupsächlich durch den letzteren Umstand hat sich die Fertigstellung des Werkes verzögert.

Difficulties in machine writing Japanese also occurred at later stages. The technical and practical problems remain to this day, because of the great number of characters. (Foljanty 1984) reports that in Japan the number of machines needed to print Japanese characters was small, because even business correspondence was mainly done by hand up to the 1970is. Computers worked with Katakana only. There were devices like mechanical-type Japanese typewriters, with around 2000 keys for the different Kanji. A Kanji synthesis system worked with the internal structure of the Kanji. The 'multi-radical lookup' in Breen's (2004) system *WWWJDIC* has a similar concept. A user chooses a number of Radicals from a list and the system computes all Kanji that contain these Radicals. In order to handle word processing the Industry standard are Input Method Editors that come with the operating system. Input Method Editors (IME) use Romaji, the Latin transcription of Japanese and a language model in order to generate a Kanji list, appropriate to the spelling of a word. That is necessary since one Kanji can have several readings and Japanese is rich in homophones (Yuan et al. 2005; Hadamitzky 1995). Certainly the most modern input method for Japanese characters is handwriting recognition. The difficulties of other input methods show that there is great need for a better and faster method of input. See section 3.3.7 for a description of research efforts in order to provide technology for using handwriting as an input method for Japanese.

2.3 Difficulties of the Japanese Script for Learners

There are at least four ways to learn the Japanese Script. With *learning the Japanese script* we mean *learning the Kanji*. For the Hiragana and Katakana are only small sets of syllabic characters, the effort to study those is only slightly larger than learning the Latin alphabet. The number of different learning methods and the diversity of advice that can be found on this topic on the Internet and in teaching books suggests that learning the Kanji is regarded a difficult issue. It is apparent that the difficulties lie in the complexity of the Kanji, but also on the structure of the writing system as such. Stahlmann (2004) reports the subjective impression of many learners that Chinese is among the most challenging languages and even seems daunting. Japanese is no different, as most of the difficulties of the Chinese language are characteristics of Japanese as well. The pronunciation of Japanese is comparatively simple for a European learner, as the phonetic inventory of the language is similar to that of English (Tsujimura 2007). On the other hand, the Japanese writing system with its three different scripts and several readings of a character has other interweavements as demonstrated in the previous sections. Despite the difficulties, the interest in the Japanese language and culture is unbowed:
There are many books available centred around cultural interchange between the western and Japanese culture. For example, (Haschke and Thomas 2008) present German words of Japanese heritage, i.e. a miniature etymological lexicon. The sheer existence of that type of book on the free market suggests that Japanese language and culture radiate fascination into the western world. Therefore, there is a great need for learning methods.

The four groups of learning methods are:

1. **Writing Repetition** - Each Kanji has to be written several times until its meaning and the readings are full known to the learner. Most Kanji books generally follow this concept.

2. **Flash Cards** - Instead of writing the Kanji, the flash card method teaches the passive knowledge. Flash cards for Kanji are readily available and can be home made easily.

3. **Mnemonic methods** - Methods that use mnemonics in order to help a learner memorise the Kanji.

   (a) **Textual mnemonics** The *Heisig method* assigns a unique mnemonic to each Kanji in order make it more memorable. These mnemonics are little stories oriented toward the shape of a Kanji and do not necessarily have a connection to the actual meaning of a Kanji. They are just mnemonics. The Heisig method is a two-part method. In the first part the learner studies all the Kanji and their translations, in the second part the Japanese readings are introduced (Heisig and Rauther 2007).

   (b) **Visual mnemonics** *Kanji Pict-O-Graphix* is another mnemonic method. In this method the Kanji are depicted with images that represent the meaning of the Kanji. The stories are therefore more visualised than in the Heisig method (Rowley 1992).

2.3.1 Learner's Problems

1. **Similar Kanji.** In order to be able to distinguish characters like 営 in 営業 and 管 in 管理 a high level of concentration is necessary. Especially, when Japanese is mostly written on the computer and not by hand, the muscular memory does not help any more. Therefore the more Kanji are known, the more difficult it becomes to distinguish similar Kanji.

2. **Compounds.** Often, new vocabulary is studied, rather than new Kanji. The vocabulary are often composed of Kanji characters that are already known. Studying time is needed in order to study the vocabulary and compounds. Therefore it is difficult to study new Kanji at the same time.

3. **Unusual readings.** Since Kanji have several readings, it is useful for a learner to study the most frequent ones. For example, the Kanji 雪 (Eng. 'snow') has two standard readings (see section 2.2.2.3 for more on readings):
   The Chinese reading (*on-reading*) is セツ/setsu, while the Japanese reading (*kun-reading*) is ゆき/yuki. These two readings occur in very usual Japanese words, having a relation to *snow*. *Setsu* occurs in 雪害 (Jap. reading セツガイ/*setsu*gai; Eng. 'damage through snow') and 新雪 (Jap. reading シンセツ/shin*setsu*; Eng. 'fresh snow'). *Yuki* occurs in 初雪 (Jap. reading ハツユキ/hatsu*yuki*; Eng. 'first snow of the season'), 大雪 (Jap. reading オウユキ/ou*yuki*; Eng. 'heavy snow') and 雪合戦 (Jap. reading ユキガッセン/*yuki*gassen; Eng. 'snowball fight'). Similarly, the Kanji 崩 (Eng. 'break down, destroy') has two standard readings:
   The Chinese reading (*on-reading*) is ホウ/hou, while the Japanese reading (*kun-reading*) is くず/kuzu. The Kanji 崩 in the reading *kuzu* does not appear as an individual word, in the following examples it is used as a verb stem only. These two readings occur in Japanese words related to *break down*. *Hou* occurs in 崩御 (Jap. reading ホウギョ/*hou*gyo; Eng. 'death of the emperor'). *Kuzu* occurs in 山崩れ (Jap. reading ヤマクズレ/yama*kuzu*re; Eng. 'landslide') and 切り崩す (Jap. reading キリクズス/kiri*kuzu*su; Eng. 'erode'). The compound of the two, forms a logical unit 雪崩 (Eng. 'avalanche', which could be described semantically as a *'snow landslide'). The reading however, is quite unexpected. Instead of a combination of 'yuki' and 'kuzu(re)' to yield **yukikuzure*, the correct reading is *nadare* (Jap. ナダレ). Unexpected readings like that can be a frustrating and exhausting experience for a learner. The reading *nadare* in this example is a *Jukujikun*. Jukujikun (Jap. 熟字訓; pron. ジュクジクン/jukujikun; Eng. 'compound kun readings') are specialised Kun-readings that only occur in fixed compounds, comparable to *irregular* words in English (Wydell 1998).

4. **Alternative Kanji.** Some Kanji have the same meaning and reading, yet look differently. For instance the number *one* can be written 壱 or simply 一.

5. **Homophones.** Japanese has several homophones. An extreme example is きょう (*kyou*): Around 80 Kanji have at least an alternative reading *kyou*. Also, for example, there are three Kanji that can be read アツイ/atsui,namely, 熱い (Eng. 'hot' - for objects or feelings), 暑い (Eng. 'hot' - for the weather) and 厚い (Eng. 'thick' - for clothes; 'kind, warm' - as a characteristic of a person). This can be confusing for a learner.

6. **Infrequent Jōyō Kanji.** There are Kanji that are in the official list of daily use Kanji, but in fact they are actually not frequently used. These can be forgotten easily by a learner, since those Kanji do not appear often in Japanese texts.

7. **Non-Jōyō Kanji.** The fact that a character is not in the list of frequently used characters does not mean that the character is negligible. It may well be necessary to know, at least passively.

Chapter 3

On-Line Handwriting Recognition

3.1 Introduction

Handwriting is a very personal skill to individuals. It consists of graphical marks on a surface and can be used to identify a person. Communication is the main purpose and this is achieved by drawing letters or other *graphemes*, which in turn represent parts of a language. The characters have a certain basic shape, which must be recognisable for a human in order for the communication process to function. There are rules for the combination of letters, which have the ability - if known to the reader - to help recognise a character or word.

Handwriting was developed as a means of communication and to expand one's own memory. With the advent of each new technologies the question arose, if handwriting was going to survive. However, the opposite seems to be the truth: For example, the printing press increased the number of documents available and therefore increased the number of people who learnt to read and write. Through the increased rate of alphabetisation, naturally there was an increased use of handwriting as a means of communication.

In various situations handwriting seems much more practical than typing on a keyboard. For instance children at school are using notepads and pencils or ink pens, which are regarded as a better tool to teach writing by German teachers. Therefore it can be concluded that there is little danger of the extinction of handwriting as a communication tool. In fact, as the length of handwritten messages decreases, the number of people using handwriting increases (Plamondon and Srihari 2000).

3.2 Handwriting Features

Any script of any language has a number of features. The fundamental characteristic of a script is that the differences between the features of different characters are more decisive than the different features of drawing variants of the same letter in individual handwriting styles. There might be exceptions, because *0* (number between '-1' and '1') and *O* (letter between 'n' and 'p') or *1* (number between '0' and '2') and *I* (letter between 'h' and 'j') respectively, can be written alike. However, in those cases, context makes clear which one was intended by the writer. Despite the exception, written communication can only work with that fundamental quality (Tappert et al. 1990).

3.2.1 Handwriting Properties of Latin Script

In the Latin script we have 26 letters, each of which has two variants, a capital and a lowercase variant. When writing a character in the Latin script, there are four main areas, in which the character can reside. All characters have their main part between a top line and a ground line. There is also a middle line. Capital characters stretch out to use the full space between the ground line and the top line, whereas lowercase characters usually use the space between the ground line and the middle line. Some lowercase characters (like lowercase *b, d, f, h, k, l, t*) have an ascender and use the area above the middle line as well, some lowercase characters have a descender and use the area below the ground line (like lowercase *g, j, p, q, y*). In handwritten cursive script, there are writing variants where also some lowercase letters (*f, z*) and certain uppercase characters (*G, J*) expand below the ground line. For all latin-based alphabets, usually one character is finished before the next one starts, however, there are

exceptions: In cursive handwriting, the dots on i and j and the crosses of t might be delayed until the underlying portions of all the characters of the word are completed. Figure 3.1 shows examples of letters that expand below the ground line with their descender or stretch up to the top line with an ascender.

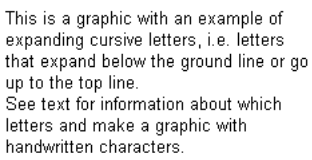

Figure 3.1: Expanding cursive letters

3.2.2 Handwriting Properties of East Asian Scripts

Generally, a handwriting is a formed of a number of strokes, that are drawn in a time sequence. Opposed to the latin-based alphabets, consider Chinese and Japanese script. Chinese has a larger alphabet, up to 50.000 characters, 3.000-5.000 of which are in active use. There are also two writing styles, block style - which corresponds to printed characters in Latin alphabets, even if handwritten. The other style is cursive style. In block style the individual parts of the character are usually written in proper stroke order, and abide by the proper stroke number. In cursive style the characters are written faster, with less care and don't necessarily abide to stroke number or order. In fact, they are usually written with fewer strokes, connecting some block-style strokes by using simpler radical shapes (Tappert et al. 1990).

In Japanese, three different scripts are in active use at the same time, mixed and next to each other. They are called *Hiragana* (ひらがな), *Katakana* (カタカナ) and *Kanji* (漢字). Hiragana and Katakana are syllabic alphabets, each containing 46 characters (see section 2.2.1), whereas Kanji are essentially the Chinese *Hànzì* (汉字) characters as they were imported into the Japanese language (see section 2.1).

The different scripts can even be blended with each other within one word. Take for instance the verb '食べる' (pron. */taberu/*, Eng. *to eat*). The first character '食' is a Kanji character, pronounced */ta/*, which also bears the meaning of the word. The second and third characters are the Hiragana characters 'べ' *be* and 'る' *ru*, which are there for conjugation only as well as for phonetic reasons. Without them, the character '食' still bears the meaning of the concept *eat*, but the character alone does not result in the verb *taberu*.

3.3 Automated Recognition of Handwriting

3.3.1 Short History of Handwriting Recognition

Handwriting recognition (HWR) as a technological discipline performed by machines has been around for many years. The quality of the systems recognising handwriting has improved over the decades. It is the key technology to pen-based computer systems. The first research papers concerned with *pattern recognition* on computers were published in the late 1950s, *Handwriting recognition* as an individual subject in the early 1960s. (Goldberg 1915) describes in a US Patent a machine that can recognise alphanumeric characters as early as 1915. However, despite the surprise of how early such a device was invented, it should be taken into consideration that that was before the times of modern computers, therefore the methods he employs are quite different from the algorithms used after the advent of computers, more concretely, computers with screens.

(Tappert et al. 1990) describe in their review the development of handwriting recognition, which was a popular research topic in the early 1970s and then again in the 1980s, due to the increased availability of pen-input devices. Generally speaking, handwriting recognition (HWR) involves automatic conversion of handwritten text into a machine readable character encoding like ASCII or UTF-8. Typical HWR-environments include a pen or stylus that is used for the handwriting, a touch-sensitive surface, which the user writes on and a an application that interprets the strokes of the stylus on the surface and converts them into digital text. Usually, the writing surface captures the x-y coordinates of the stylus movement.

3.3.2 Pattern Recognition Problems

The general problem of *pattern recognition* is to take a non-symbolic representation of some pattern, like mouse or pen coordinates and transform it into a symbolic representation like a *rectangle* with its coordinates, or in the case of handwriting recognition, a character. Pattern recognition is a symbol manipulation procedure, that tries to generate discrete structures or sub-sets of discrete structures. Some see it as a game theory problem, where machine 'players' try to match the interpretation of an input produced by machine 'experts' (Zanibbi et al. 2005).

3.3.2.1 Related Problems

There are several related problems, the recognition of equations, line drawings and gestures symbols. The recognition of language symbols includes the different large alphabets of Chinese, the different scripts of Japanese, alphabetic scripts like Greek, or Arabic and other non-alphabetic scripts like the Korean Hangul, but also various writing styles of the latin-based alphabets, and diacritics that are used to denote pronunciation variants in different languages using Latin script, like Turkish or Vietnamese.

Other Problems that are related to handwriting recognition include for example mathematical formula recognition, where mathematical formulae are analysed and put into a computable format (Chan and Yeung 2001). In diagram recognition both the characters and the diagram layout are recognised (Blostein and Haken 1999).

3.3.2.2 Problem of Similar Characters

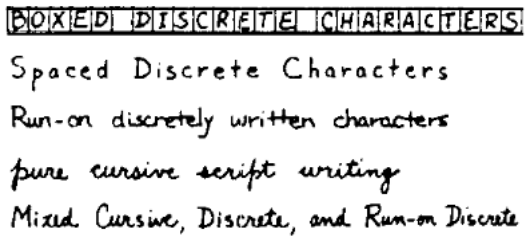

Figure 3.2: Different handwriting styles

There are several subproblems to the task of pattern recognition of a character. Different styles of handwriting after (Tappert et al. 1990) can be seen in figure 3.2. The scripts toward the bottom of figure 3.2 are harder to recognise. In the case of boxed discrete characters, segmentation is done for the machine by the user. Run-on discrete characters are easier to recognise than pure cursive handwriting, because there is a pen-up and pen-down between each character. In cursive handwriting, segmentation between the characters becomes a more difficult task. Some parts of the writing may be delayed, like the crosses of *t* or the dots on *i* or *j*. Besides the segmentation, the discrimination of shapes is often not trivial. Humans may or may not be able to decipher somebody else's handwriting and clearly distinguish between, say *U-V*, but most of the times, context helps with that task. Other characters have similar shapes, too, like *C-L*, *a-d* and *n-h*. Confusion can arise between characters and numbers like *O-0*, *I-1*, *l-1*, *Z-2*, *S-5*, *G-6*, *I-1*, *Z-2*. Lowercase and uppercase are hard to distinguish in the cases of *C-c*, *K-k*, *O-o*, others are mainly distinguished by their position relative to the base line of the rest of the text: *P-p*, *Y-y*. Therefore, context helps the human reader to identify the correct character. This could be used as an advantage in automated pattern recognition, as well. However, the other characters nearby would have to be recognised first, which creates a (solvable) hen and egg problem. In the Japanese script the problem is taken to another dimension. Consider 本 (root) and 木 (tree) those two characters are only a very basic sample of characters that might lead to confusion of the different symbols. However, due to the hierarchical organisation of the characters with their radicals (see section 2.2) there are many more shapes that look very much alike. From the shape recognition perspective, minor changes to the shape of a character can change its meaning drastically. Compare 嘸 (how, indeed), 撫 (stroke, pat), 蕪 (turnip) and 憮 (disappointment). They all contain the radical 無 (nothingness, none), which doesn't seem to have any semantic connection with the characters, however, it can be seen as the main radical in those characters.

3.3.3 Hardware Requirements

In order to perform on-line handwriting recognition, the handwriting needs to be captured in some way. Special hardware is necessary to perform the task of capturing both the x-y coordinates and the time information of the handwriting input device.

Several different hardware commercial products are available in order to capture the x-y coordinates of a stylus or pen. Graphics tablet like the products of the Wacom Co., Ltd.[1] are popular input devices for hand motions and hand gestures. The use of pen-like input devices has also been recommended, since 42% of mouse users report feelings of weakness, stiffness and general discomfort in the wrist and hand when using the mouse for long periods (Woods et al. 2002). The sampling rates of digital pens are usually around 50-200 Hz. The higher the sampling rate the finer the resolution of the resulting co-ordinates, which leads to more accurate measurement of fast strokes.

Moreover there are PDAs and Tablet PCs, where the writing surface serves as an output device, i.e. an display at the same time. If the pen capturing area is transparent and has an instantaneous display behind that shows immediately whatever input the user drew with the stylus, a high level of interactivity can be reached (Santosh and Nattee 2009). These displays include touch screens, which are a newer development. New generation mobile phones like the notorious iPhone from Apple Inc. also contain touch-displays, but for those it is more common to be operated without a stylus. For the task of handwriting recognition, a stylus can be regarded as the more natural device, since people usually write with pens on paper, therefore a stylus on a display seems more natural than using a finger on a display for writing. In order to interpret user gestures, an input given directly with the fingers is a more natural option. Gestures for zooming into digital pictures, or turning to the next page of a document are interpreted on these devices.

Another rather new development are real-ink digital pens. With those, a user can write on paper with real ink, and the pen stores the movements of the pen-tip on the paper. The movements are transferred to a computer later. It can be expected that with technologies like Bluetooth it may be possible to transfer those data in real-time, not delayed. In a fairly new development accelerometer technology has been used for handwriting recognition, using a mobile phone as a device to write in the air (Agrawal et al. 2009). That approach can be regarded as an area that is only loosely related to classical handwriting recognition, as the phone stores an image of the strokes in the air that were measured by the accelerator device, but does not transform the strokes into characters.

3.3.4 Recognition vs Identification

Handwriting recognition is the task of transforming a spatial language representation into a symbolic representation. In the English language (and many others) the symbolic representation is typically 8-bit ASCII. However, with *Unicode* being around for more than a decade now, storage space on hard disks not being as much of an issue any more and *RAM* being readily available to the Gigabytes, it has become more common to use a *UTF-8* encoding, which is a variable-length character encoding for Unicode (Unicode Consortium 2000). Akin disciplines to handwriting recognition are *handwriting identification*, which is the task of identifying the author of a handwritten text sample from a set of writers, assuming that each handwriting style can be seen as individual to the person who wrote it. The task of *signature verification* is to determine if a given signature stems from the person who's name is given in the signature. Thus, handwriting identification and verification can be used for analysis in the field of jurisdiction. They determine the individual features of a handwritten sample of a specific writer and compare those to samples given by a different or the same writer. By analysing those features one can find out if a piece of handwritten text is authentic or not.

3.3.5 Interpretation of Handwriting

Handwriting recognition and interpretation are trying to filter out the writer-specific variations and extract the text message only. This conversion process can be a hard task, even for a human. Humans use context knowledge in order to determine the likeliness of a certain message in a certain context. For instance, a handwritten message on a shopping list that could be read as *bread* or *broad* due to the similarities of the characters for 'e' and 'o' in some cursive handwriting styles, will be interpreted as *bread*, since it is a much more likely interpretation in the shopping list domain. However, if the next

[1] www.wacom.com

word on the shopping list is *beans*, the likelihood for the interpretation of the first word as *broad* rises, because the collocation *broad beans* is a sequence that is likely on a shopping list, at least more likely than having the interpretation *bread* and then *beans* without a clear separation between the two. Even with non-handwritten, but printed characters, the human mind can be tricked because of the brain's ability to perform these interpretations within milliseconds without conscious thinking. An example of that are modern T-Shirt inscriptions that state things like *Pozilei* in a white font on a green ground (the German police colours in most federal states are green and white), which German native speakers usually read as *Polizei* (police), because that is the most likely interpretation.

3.3.6 On-Line vs. Off-Line Recognition

3.3.6.1 Basic Features of On-Line Recognition

On-line HWR means that the input is converted in *real-time*, *dynamically*, while the user is writing. This recognition can lag behind the user's writing speed. (Tappert et al. 1990) report average writing rates of 1.5-2.5 characters/s for English alphanumerics or 0.2-2.5 characters/s for Chinese characters. In on-line systems, the data usually comes in as a sequence of coordinate points. Essentially, an on-line system accepts as input a stream of x-y coordinates from an input device that captures those data combined with the appropriate measuring times of those points.

3.3.6.2 Basic Features of Off-Line Recognition

Off-line HWR is the application of a HWR algorithm after the writing. It can be performed at any time after the writing has been completed. That includes recognition of data transferred from the real-ink pens (see section 3.3.3) to a computing device after the writing has been completed. The standard case of off-line HWR, however, is a subset of optical character recognition (OCR). An scanner transfers the physical image on paper into a bitmap, the character recognition is performed on the bitmap. An OCR system can recognise several hundred characters per second. Images are usually binarised by a threshold of its colour pattern, such that the image pixels are either 1 or 0 (Santosh and Nattee 2009).

3.3.6.3 Similarities and Differences of On-Line and Off-Line Recognition

There are two main differences between on-line and off-line handwriting recognition. Firstly, off-line recognition happens, hence the name, after the time of writing. Therefore, a complete piece of writing can be expected as an input by the machine. Secondly, on-line devices also get the dynamic information of the writing as input, since each point coordinate is captured at a specific point of time, which can be provided to the handwriting recogniser along with the point coordinates by the operating system. In addition, the recogniser has information about the input stroke sequence, the stroke direction and the speed of the writing. In the off-line case these pieces of information are not readily available, but can be partially reconstructed from the off-line data (Santosh and Nattee 2009).

All these information can be an advantage for an on-line system, however, off-line systems have used algorithms of line-thinning, such that the data consists of point coordinates, similar to the input of on-line systems (Tappert et al. 1990). When line thinning has been applied, an off-line system could estimate the trajectory of the writing and then use the same algorithm as an on-line system (Plamondon and Srihari 2000). Vice versa, an on-line system can employ algorithms of off-line systems, since it is possible to construct a binary image from mouse coordinates of points. However, only few systems of that kind have been developed. A promising approach was developed in the middle of the 1990s by (Nishida 1995), where an on-line and and off-line system are fully integrated with each other as a *blend system*. (Velek et al. 2002) determine the likelihood of different classifiers in a multi-classifier system, before they are combined and yield a result.

On-line systems can refer interactively to the user in case of an unsuccessful or uncertain recognition. Along these lines, an on-line system can adapt to the way a specific user draws certain characters and a user can adapt to the way a system expects characters to be written distinctively.

3.3.7 HWR of Hànzì (汉字) and Kanji (漢字)

The HWR of the Chinese Hànzì (汉字) and the Japanese Kanji (漢字) in practise are merged as *On-line Chinese Character Recognition* (OLCCR). The techniques are essentially the same, only the language

models differ. Hereafter, I'm going to use the term OLCCR for on-line character recognition of both Chinese and Japanese characters.

From the 1990s, On-Line Japanese and Chinese Character Recognition systems have been aiming at loosening the restrictions imposed on the writer when using an OLCCR system. Their focus shifted from recognition of block style script ('regular' script) to fluent style script, which is also called 'cursive' style. Accuracies of up to about 95% are achieved in the different systems.

(Nakagawa et al. 2008) report their recent results of on-line Japanese handwriting recognition and its applications. Their article gives important insights into character modelling, which are employed in this application. Multiple subpatterns of characters are detected and used for character modelling.

3.4 A Typical On-Line HWR Application

A typical HWR application has several parts that follow up on each other in a procedural fashion. The main parts of such an application are the following:

- **Data capturing**: The data is captured through an input device like a writing surface and a stylus. Data capturing is described in section 3.4.1.
- **Preprocessing**: The data is segmented, noise reduction like smoothing and filtering are applied. Preprocessing is described in section 3.4.2.
- **Character Recognition**: Feature analysis, stroke matching, time, direction and curve matching. A description of character recognition can be found in section 3.4.3.

In a typical HWR application there are some intermediate steps that can be regarded as partial processes of the ones mentioned above. In preprocessing, there is often a segmentation step (see 3.4.2.1), most systems perform one or several methods of noise reduction (see 3.4.2.2) and it is common to employ a data normalisation step (see 3.4.2.3). Additionally, some systems that employ a postprocessing step (see 3.4.4).

3.4.1 Data Capturing

Special hardware is needed in order to capture the data necessary for on-line HWR. It is possible to input data with a regular mouse on a regular computer, however the data will be less *noisy*, if a stylus is used for input of the handwriting. See section 3.3.3 for more information on the hardware requirements of HWR. The data that is served by a device driver or the operating system is structured as mouse coordinates. In case of pressure-sensitive input devices the device driver also returns the pressure intensity. However, a typical on-line HWR application uses only the mouse coordinates, not the pressure intensity. Generally, HWR applications work with mouse coordinates, thus the trajectory of the stylus, in accordance with the appropriate time information for each of the sampling points.

3.4.1.1 Sampling

The pen-up and pen-down information is a central part in data capturing. Depending on the sampling rate, there will be between 50 and 200 points per second. Those can be viewed as a function of time. With theses pieces of information, it is possible to track the number and order of strokes drawn by the user. According to the writing speed, the number of coordinates per distance can vary. A fast stroke will have fewer point samples, even if the same distance was covered on the writing surface. Some see this as a problem and *re-sample* the points to be of equal distance to each other and to have a constant number of sample points in every stroke (Santosh and Nattee 2009). (Joshi et al. 2005) re-sample the strokes in a way that they yield a constant number of point samples in space, rather than in time. They propose an approach for Devanāgarī [2] characters, where each of the characters is represented by 60 sample points.

[2] Devanāgarī is a script used to write the Sanskrit, Prākrit, Hindi, Marathi, and Nepali languages, developed from the North Indian monumental script known as Gupta and ultimately from the Brāhmī alphabet, from which all modern Indian writing systems are derived (Encyclopædia Britannica Inc. 2009).

3.4.2 Preprocessing

Most On-Line HWR systems apply some kind of preprocessing. Generally, preprocessing serves to smoothen the data, eliminate noise and normalise the data, in order to retrieve a more accurate recognition in the next step. (Tappert et al. 1990) distinguish between the phases *external* and *internal segmentation*, a *noise reduction* step that has several sub steps:
Smoothing, *filtering*, *wild point correction*, *dehooking*, *dot reduction* and *stroke connection*. Some systems also employ a *normalisation* step, that can include *deskewing*, *basline drift correction* and *size normalisation*. (Santosh and Nattee 2009) employ a similar perception in their review about several on-line HWR systems. Their *noise elimination* corresponds to *noise reduction* and the aforementioned *normalisation* describes roughly the same techniques that (Tappert et al.) depict. (Santosh and Nattee) also present the additional processing step of *repetition removal*, is comparable to the *dot reduction* mentioned above. In the remainder of this section the typical preprocessing steps are presented in more detail.

3.4.2.1 Segmentation

The term *Segmentation* describes the processing step of segmenting the individual characters or strokes from each other.

External Segmentation *External segmentation* is the step of isolating writing units. That can be single strokes, characters or words respectively. In alphabetic scripts it is sometimes difficult to segment each character externally. In CJK (Chinese, Japanese and Korean) cursive script it is difficult to distinguish individual strokes, however, since it is common to write in boxes, usually each character can be isolated without much processing. The earliest method of external segmentation is an explicit signal from the user, other work used the projection of the trajectories on the X-axis for spatial segmentation. Another method uses temporal information and employs a time-out between the end of one and the beginning of another character. A fairly easy possibility is of course the use of boxed characters, where the user is required to write each character in a separate box. Taking the box idea to the next level of abstraction, some systems provide different boxes on a single screen for different types of characters (Tappert et al. 1990).

Internal Segmentation In cursive script, several strokes are connected together, even several characters can be written without a pen-up movement. Therefore some partial recognition is needed in order to perform the segmentation. In *overlaid handwriting recognition* a sequence of characters is written on the same area, e.g. on a very small display like a wrist watch with a touch screen. In that case, segmentation becomes a problem even for OLCCR systems. (Shimodaira et al. 2003) employ substroke HMMs in a wearable computing environment. They achieved performances of 69.2% with free stroke order and 88% with fixed stroke order characters.

3.4.2.2 Noise Reduction

Any stroke drawn on a device with a pen input contains noise. Digitising errors like limited accuracy of the tablet or erratic pen-down indication or hand fluctuation are sources of noise in the input data. There are different types of noise and different types of filters for elimination of the noise from the data (Santosh and Nattee 2009).

Smoothing *Smoothing* is usually a technique that averages a point with its neighbour point. A common variant of that is only averaging a point with its previous point. That ensures that the recognition can proceed as the input is received (Tappert et al. 1990). (Joshi et al. 2005) employ a 5-tap low pass Gaussian filter in order to smoothen the data. Today, Gaussian filters are often used as a means of smoothing data, despite the fact that *filtering* usually refers to reducing the number of samples in a sequence of points.

Filtering *Filtering* describes a technique that thins out the number of samples in a point sequence. It is sometimes referred to as *thinning* but should not be confused with *line thinning* in off-line HWR. Here, thinning means reducing the number of data samples. Also, duplicate points can be detected

and eliminated through filters. The ideal filtering method depends on the recognition method. There are filtering techniques that enforce a minimum distance between points. When a filtering method of that type is used, points tend to be equally spaced. This filtering technique assumes many samples per distance. When a stroke is drawn quickly on a tablet, it can happen, that the distance of the actual sample points is larger than the minimum distance the filter permits. In that case, interpolation can be used in order to introduce new data points in between and achieve equidistant points. Smoothing and filtering can also be performed in one operation for time optimisation (Tappert et al. 1990).

Wild Point Correction *Wild Point Correction* is used for elimination of spurious points. Those points occur usually as a result of errors in data capturing. Acceleration and generally any change of velocity of the hand movement can cause the digitiser tablet to detect wild points (Tappert et al. 1990).

Dehooking Hooks occur at the beginning and more frequently at the end of a stroke. They are recognised by the data capturing device because of inaccurate pen-down detection or random motion when lifting the stylus off the tablet. *Dehooking* is a technique to remove the hooks at the end and at the beginning of a stroke (Tappert et al. 1990).

Repetition Removal *Repetition removal* or *dot reduction* reduces dots to single points (Tappert et al. 1990). Points that are accumulated on a very close area and form a dot, do not improve recognition quality, but rather distort the trajectory. There can even be co-occurring points. Slow handwriting will generate repeating points, and those are generally at the dominant points, such as corners (Santosh and Nattee 2009).

Stroke Connection A *stroke connection* preprocessing method can eliminate pen-up movements that are suspected to be accidental or inaccurately measured. There are different methods to do this, one of which connects strokes if the distance between pen-up and the next pen-down is short in comparison to the size of the character (Tappert et al. 1990).

3.4.2.3 Normalisation

Data *normalisation* is a preprocessing step that prepares data to better match the gold standard and the lexicon entries. It can correct slants or size.

Deskewing is a technique that modifies the slant of an input. Deskewing algorithms can be applied to individual characters or whole words (Tappert et al. 1990). In Chinese and Japanese character recognition, slant correction can be applied in a similar fashion like with latin-based alphabets.

Baseline Drift Correction detects a baseline or horizontal line relative to a word or a set of characters. After detection, that line is corrected to be horizontal (Tappert et al. 1990).

Size Normalisation Adjustment to a standard size can be done with *size normalisation*. This process also normalises for location by relocating the centre of a character (Tappert et al. 1990).

Stroke Length Normalisation sometime called *resampling* normalises the number of points in a stroke or character. Dealing with strokes that were normalised in such a way provides for easy alignment and classification (Tappert et al. 1990).

3.4.3 Character Recognition

Character recognition is a special case of shape recognition. It is the recognition of the shape of writing units. Character recognition falls into different categories, there are methods for the recognition of characters, cursive script, words, gestures, equations, line drawings. Also, there are several different methods for the recognition of the shape of a character. The most common methods include

- **Feature Analysis**, where the distinctive features of characters are taken into account (see 3.4.3.1).

- **Zone Sequences** that code zones on the writing surface (see 3.4.3.2).
- **Stroke Direction** - the individual strokes or substrokes of a character are identified and matched against a stroke sequence database (see 3.4.3.3).
- **Curve Matching** is a method that comes from signal processing and is essentially independent of the alphabet (see 3.4.3.4).
- **HMM Analysis** is independent of the other methods. Basically HMM analysis performs a statistical analysis of the features used, regardless of what they are, i.e. curves, time sequences or writing features of the characters themselves (see 3.4.3.5).

3.4.3.1 Feature Analysis

A character or shape can be represented by a set of features. These features are dependent on the alphabet. Therefore, systems can employ a preconceive analysis of the individual characters of the alphabet. For the Latin alphabet, the systems use features like descender or not descender, dot or no dot (Tappert et al. 1990). Some systems use a decision tree for **binary features**. If there is a descender, for lowercase characters, the recognition choice is reduced to *f, g, j, p, q, y, z*. If an ascender is present, for lowercase characters, the choice is reduced to *b, d, f, h, k, l, t*. In case there is an ascender and a descender, the intersection of the two sets leaves only one element, namely the character *f*. A HWR system can come to similar conclusions by checking if a dot is present *i, j*, if the character has an open line *c, u, v, w, y* or a closed line *a, b, d, e, g, o, p, q*. Systems that perform feature analysis have the potential to use psycholinguistic knowledge and drive the recognition process along the lines humans distinguish characters from each other. The feature analysis method has the disadvantage that it may not produce alternative recognition choices. (Jäger et al. 2001) use ascenders and descenders as features in neural network, where the number of points above or below the baseline is counted.

There are also systems that use **non-binary features**. It is a very common technique in pattern recognition to use a fixed number of non-binary features. A feature space of that kind can be further divided into decision regions (Tappert et al. 1990).

3.4.3.2 Zone Sequences

There are systems that use *zone sequences* in order to represent characters. The zones can be specified by sub-dividing the *bounding box* of a character. The trajectory of the pen that is drawn through the zones is then transformed into a time sequence of zones. That time sequence can be matched against a dictionary of sequences, in which each character is represented (Tappert et al. 1990).

3.4.3.3 Stroke Direction

Stroke direction is another popular feature among HWR systems. The pen-tip motion is captured and each stroke is categorised as a primitive direction (up, down, left, right, other systems also use the diagonal directions and a pen-down, pen-up). (Nakai et al. 2001) and (Tokuno et al. 2002) use the stroke direction feature combined with a pen-coordinate feature. (Okumura et al. 2005) and others follow a similar approach. The substroke approach is very popular in OLCCR, it is often combined with HMMs (see 3.4.3.5).

3.4.3.4 Curve Matching

Curve matching is a set of methods that has been popularised in signal processing. Prototype characters are stored in a database as descriptions of curves. During the recognition process the input curves are matched against the prototypes. It is common to use curves that are functions of time, the direction of the tangent to the stroke or both. Characters have been encoded as a time sequence of eight stroke directions, and as time regions (Tappert et al. 1990). Chinese characters have been encoded as a small number of fixed points, since they consist of mainly straight strokes (Nakagawa et al. 2008). The approach of Jäger and Nakagawa (2001) utilises the Unipen format (Guyon et al. 1994; Unipen Foundation, International 2010) in order to create Japanese character databases.

Elastic matching Curve matching and pattern matching become equivalent in their feature space when there is a constant number of points in a curve and in one-to-one correspondence. However, since many strokes are non-linear the best fit is often not a linear alignment of the points. In order to solve point sequence comparison problem, elastic matching has been used successfully (Tappert et al. 1990). (Joshi et al. 2004a) compare different elastic matching methods for Tamil handwriting recognition. *Dynamic Time Warping* (DTW) is an elastic matching technique that has been applied to handwriting recognition by a number of systems e.g. (Vuori et al. 2001) (for Latin script), (Niels 2004) (for Tamil), (Joshi et al. 2004b) (for Tamil) and (Joshi et al. 2005) (for Devanagari). While the aforementioned systems use it for a single script, (Bharath and Madhvanath 2009) attempt to create a HWR for several Indic scripts using the same technology. (Bahlmann and Burkhardt 2004) use elastic matching combined with HMM modelling for their HWR system. The DTW algorithm yields promising results, especially for the different scripts of the Indian subcontinent, however, due to its complexity, (Keogh and Pazzani 2002), (Chu et al. 2002) as well as (Bashir and Kempf 2008) proposed improvements to the algorithm, mainly through pruning.

3.4.3.5 HMM Analysis

HMM analysis is a technique that is not uncommon among modern systems. It generally follows the standard procedures described in the literature. A detailed description of how On-Line Chinese character recognition can be done using HMM analysis can be found in the works of Hu et al. (2000), Hu et al. (1996) and also Bahlmann and Burkhardt (2004). In this paper consider section 3.5.4.3 for some details of HMM analysis, embedded into the actual classification process.

3.4.4 Postprocessing

When the recognition process is finished, some systems start another process that takes as input the output of the character recognition. In this postprocessing step, language information can be used to improve the recognition accuracy. Postprocessing is mainly used for continuous character recognition, where more than a single characters is recognised. Since some systems yield a single string of characters others yield a number of alternative recognitions, often in addition with a certainty measure. A postprocessing unit can use the information to calculate estimates for words or even sentences. When the character recogniser yields single choices for characters, the post-processor can apply a correction algorithm, based on a language model (Tappert et al. 1990). (Shimodaira et al. 2003) proposed a method for overlaid handwriting recognition, where the segmentation problem (see 3.4.2.1) is resolved at the end of the recognition process: Alternative choices of characters and character boundaries are generated and probability estimations calculate the most likely sequence of characters, using both the alternative recognition results and a language model for Japanese.

3.5 Overview of a Typical OLCCR system

Typical handwriting recognition systems for Chinese and Japanese characters have broadly the same structure as systems for latin-based alphabets. The process begins with *Character segmentation*, goes on with *Preprocessing*, *Pattern description*, *Pattern recognition* and ends with *Contextual processing*, if applicable. However, there are differences to the standard process, due to the nature of the Chinese characters (see section 3.2.2).

A typical OLCCR system is depicted after Liu et al. (2004) in figure 3.3. The first two steps, character segmentation and preprocessing are virtually identical to systems dealing with Latin script. The next step, pattern representation is not only different from Latin script systems, but it has great diversity among the different OLCCR systems. The pattern description is naturally more alike in the systems focusing on Latin characters. This is due to the fact that the Latin alphabet is quite small, but has more variation concerning writing style, whereas the Chinese alphabet has a larger inventory of characters, but less variation in how to write a character. The reason for that is that it is widely agreed upon a standard stroke sequence for a character, even across country borders (Nakai et al. 2003). The next step after the pattern representation is the actual character recognition or classification. Different flavours of comparison methods are applied in different systems, but some systems employ coarse classification first, then a fine classification, whereas other systems try to find the corresponding character model in

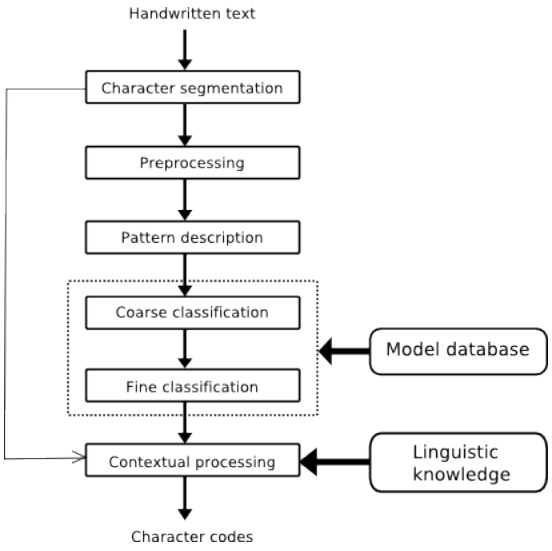

Figure 3.3: Overview of an OLCCR system

a model database in just one step. The coarse classification is done to reduce the candidate set, the fine classification is used to find the best match. There are systems that accomplish another step *contextual processing*, which can be regarded as an equivalent to postprocessing (Liu et al. 2004).

3.5.1 Character Segmentation

Character segmentation is a technique that does not apply to OLCCR systems in the same way it does to HWR systems for other scripts. Each character is written separately, even in cursive script. In Chinese characters, *cursive* refers to reduction of lines and lesser pen-up movements within a character. In e.g. Latin script it usually refers to not lifting up the pen before the end of a word (Tappert et al. 1990). Therefore character segmentation is often a trivial task in OLCCR systems, also due to the fact that systems usually only allow writing in boxes - or, in box-free application, assume horizontal character orientation (Nakagawa et al. 2008).

One system attempts to recognise overlaid handwriting characters on a small display, allowing continuous writing for the user. That is essentially boxed writing, but because of the small display, there is only one box in which all characters have to be drawn. The system can therefore be regarded as a continuous character recognition system. In that system, character segmentation is not done as a preprocessing step, but during recognition. The system uses HMM technique, based on substrokes and segments the characters alongside the recognition process (Shimodaira et al. 2003).

The system developed by Nakagawa et al. in 2008 performs continuous character recognition, regardless of the text orientation. The average character size is estimated from all strokes drawn and the estimate defines a threshold for the separation of characters.

3.5.2 Preprocessing

The preprocessing techniques used in OLCCR systems are mainly the same as the ones described in section 3.4.2, in (Tappert et al. 1990) and in (Santosh and Nattee 2009). However, since the input devices offer higher quality input, less noise reduction is necessary and smoothing is enough to remove undesirable points from the input trajectory. Therefore, many OLCCR systems limit preprocessing to the other techniques. Data points are reduced by one of two methods: approximation of lines (by feature points) or resampling (Liu et al. 2004). Size normalisation is realised in most HWR systems, OLCCR systems included.

3.5.3 Pattern Description

An important part of each HWR system is the description of the patterns that need to be recognised. Since the Japanese and Chinese alphabet has a manifold character set, pattern representation faces greater difficulty, because confusion of characters must be avoided. There are three main types of pattern

description in OLCCR systems. The classification of characters depends highly on that representation. In the beginning of research into OLCCR systems, *structural* character representation was the natural choice. As systems and computing power evolved, *statistical* character representation became more relevant. In order to optimise for speed and recognition accuracy, system designers started blending the two methods to *hybrid statistical-structural* character representation (Liu et al. 2004). Systems that use statistical character representation usually store the input patterns in feature vectors. The character model database holds the parameters for classification.

3.5.3.1 Statistical Character Representation

In statistical character representation, the central focus lies on transferring input patterns into feature vectors. The model database contains the same type of classification information for the characters. Since each stroke of a character is represented in a feature vector, systems are enabled to perform stroke order free recognition. The trajectory can also be mapped into a 2D image, in order to apply off-line recognition features. An on-line version of the direction feature, commonly used in off-line recognition systems, can be found in several systems (Suen et al. 2003). Generally, using features and a statistical representation, enables the use of several feature extraction techniques that have been developed for off-line systems (Liu et al. 2004).

3.5.3.2 Structural Character Representation

Several different approaches have been proposed to structural character representation. The most basic version of a structured representation is simple point sequences. Systems that use point sequences as a representation, calculate the distance between the strokes. Another approach deals with feature point or line segments. Feature points are calculated from the original point sequence, which is ideal for characters with mostly straight lines. Instead of matching complete point sequences, only feature point sequences need to be matched with the character database (Liu et al. 2004).

Using stroke codes for character representation has been adopted by a number of OLCCR systems. Each of the graphemic strokes of the Kanji stroke inventory is named. A character representation consists of a stroke sequence of the named strokes or a relation between the strokes.

Structured Character Representation is a approach where the characters are structured in their graphemic subcomponents. (Nakagawa et al. 2008) employ basic sub patterns and then structural information about how these are combined to form a full character. That way, the dictionary is smaller, because there is only limited number of sub patterns and their variations. The sub patterns are not chosen randomly, but taken from the inherent hierarchical structure of the Hànzì and Kanji. (Chen and Lee 1996) propose a hierarchical representation for the reference database of on-line handwriting recognition. The characters are organised hierarchically in the way that each one of them is built from the same set of radicals. Therefore the characters can be described as a trees or directed acyclic graphs, using the radical representations. In the dictionary, the radicals are shared between the characters, reducing the dictionary size immensely (Liu et al. 2004). There are around 200 radicals but around 3000 Kanji characters in active use in the Japanese script. Creating a character representation for each one of them is a considerable effort however, using a structured approach can lessen that labour-intensive task. (Breen 2004) proposed a multi-index dictionary that bears the potential to support the creation of structured character databases. Besides the alleviation of the database creation, the storage space used can limited, which will be advantageous when building a system for small computing devices.

3.5.3.3 Statistical-Structural Character Representation

A common example of statistical-structural character representation is the *substroke approach*. (Nakai et al. 2001), (Shimodaira et al. 2003), (Okumura et al. 2005) and others define 8 or more[3] stroke types, each of which has its own direction and orientation. These are used to describe input patterns sequentially but also non-sequentially.

In statistical-structural models, characters are described in graph or tree structures. The primitives, e.g. the substrokes and their relations to each other are represented in a probabilistic model. Most of the substroke systems use HMMs. The substrokes are represented by nodes and the transition

[3]Some systems distinguish between short and long strokes and thus use 16 stroke types.

between strokes is measured probabilistically. Some systems use points or line segments. Other systems use full strokes or even radicals. Attempting to avoid stroke-order dependence of the system, often multiple HMMs are generated with stroke-order variations. The substroke approach responds to that by hierarchical character models, such that the stroke order variations can be stored in a network (Liu et al. 2004).

3.5.4 Classification

Classifying an input pattern representation as an entry found in a pattern database is the kernel of each pattern recognition application. Representing data in the same format as the original patterns (see 3.5.3 is another key task, however the actual classification is the main piece of software. In this section, different kinds of classification are reviewed to fit with the different kinds of pattern description. The two main groups of classification methods are reviewed: *structural classification* and *probabilistic classification*. Additionally, a short section about *coarse classification* shows how to speed up the classification process by preprocessing in the sense of classification preprocessing, not to be confused with the data preprocessing described in section 3.4.2.

3.5.4.1 Coarse Classification

Coarse classification is a name for any method that pre-selects characters to match an input pattern, before the detailed or fine classification is done. There are different coarse classification methods, but nevertheless the overall goal of coarse classification remains the same. Increasing the speed of the classification process. That appears to be necessary due to large vocabulary size. Comparing an input pattern with each pattern in the pattern database is a time-consuming process. Therefore, many system designers subdivide their database into *character classes*. When a new input pattern needs to be analysed, the system first assigns a character class and therefore reduces the search space. This coarse classification method is called *class set partitioning*. Another method of coarse classification is *dynamic candidate selection* (Liu et al. 2004).

Class Set Partitioning methods divide the large set of characters into smaller groups - character classes. These groups can be completely disjoint or sometimes overlap. The system assigns the input pattern to one or more character classes. After that, in the next step of the recognition process, the input pattern is compared to the members of the character class it has been assigned to. The groups are devised in the database design phase (Liu et al. 2004).

Dynamic Candidate Selection methods calculate a matching score between the input pattern and each character class. A subset of character classes is selected for further enquiry. (Liu and Nakagawa 2000) have shown that the increase in recognition speed can be achieved without loss of precision by choosing a variable number of classes according to a confidence value. The dynamic grouping can be based on several different factors. For instance the overall character structure, the basic stroke substructure, the stroke sequence (Liu et al. 2004). Selection the partition classes dynamically is less labour-intensive, because it avoids the training process or manual division of the character classes. Stroke number of the input pattern can also be used as a preselection of characters. Systems that employ structured character representations (see section 3.5.3.2) can also use radical detection. All characters in the database that do not contain the confidently detected radical are ruled out from fine classification. However, the radical detection has a close relation to structural classification (see section 3.5.4.2) and cannot be done without it. Another possibility to perform coarse classification is feature vector matching, where only a subset of the feature vector is compared and thus rules out all character database entries that do not match (Liu et al. 2004).

3.5.4.2 Structural Classification

Structural classification refers to a set of methods for classification or character matching that use the internal structure of the character in order to perform the matching. Input patterns are compared to structural representations of the candidate character classes. The one with the minimum distance measure is considered the resulting character.

The different methods of structural classification can be grouped as follows:
Stroke matching or *stroke correspondence*, *elastic matching* or *DP matching (dynamic programming)*, *relational matching* and *knowledge-based matching*. Stroke matching compares the strokes that have been drawn with the strokes of the character model in the character database. Elastic matching is performed on ordered stroke sequences and contains stroke deformation techniques. Relational matching considers the strokes and the interstroke relationships, whereas stroke matching only considers the strokes, but not their relations to each other. *Hierarchical matching* and *deformation methods* are connected to these approaches in an orthogonal fashion, i.e. they can be combined with the approaches mentioned above.

Hierarchical Matching can improve recognition speed, because parts of the recognition and representation are done only once for several characters. That approach would of course not be possible without the hierarchical composition of the Kanji characters. The accuracy of that approach is limited in the way that the accuracy of the recognition depends on the accuracy of the radical recognition. However, when a radical has been recognised correctly, the classification can succeed by traversing a decision tree or network (Liu et al. 2004).

Deformation Methods deform the input pattern in a way to better match with a character representation from the database. In order to do that a deformation vector field is computed, based on the stroke correspondence. Then the prototype is deformed by local affine transformations in order to fit the input pattern.

Stroke Matching is a technique where a distance between the input strokes and the strokes in the database are calculated. The distance between an input pattern and a character in the database is the sum of the between stroke distances. When the input strokes are reordered according to domain-specific rules, alternative stroke orders can be matched. The domain-specific rules contain linguistic information such as the stroke order precedence. Alternatively, there can be several database entries for the same character with varying stroke orders. Defining these rules is a labour-intensive task (Liu et al. 2004).

Elastic Matching does not differ much from the general elastic matching techniques described in section 3.4.3.4. Elastic matching searches for the ordered correspondence between primitive symbols, such as coordinate points or line segments. During that process the algorithm seeks to minimise the edit distance (*Levenshtein distance*). A popular elastic matching technique is called *Dynamic time warping* (DTW), which is used by many different systems, especially the concerned with HWR for scripts that have rather curvy characters like Tamil (Niels and Vuurpijl 2005), (Joshi et al. 2004b) or Devanāgarī (Joshi et al. 2005), but has also been applied to HWR of latin-based alphabets (Negussie 2008), (Vuori et al. 2001). (Niels 2004) and (Joshi et al. 2004a) have done research into which elastic matching approach is suitable for HWR, as well as, if DTW is useful for that task at all. Both conclude that the method bears advantages, because certain problems concerning handwritten input, such as stroke length, or number of sample points can be dealt with.

Relational Matching systems search for correspondences between element sets. That is, for example the spatial relationships between strokes in a set of strokes in a character representation. Since the stroke order within radicals tends to be invariant, it is possible to perform elastic matching for the strokes within the radical and use relational matching for the strokes on a character level. Just like elastic matching and stroke matching, relational matching can be performed by systems focusing on character patterns or radicals. The advantage of relational matching over elastic matching is that it is stroke-order independent. Since the relationship constraint improves the accuracy of the matching, relational matching also has an advantage over stroke matching. However, it is computationally complex and therefore slower than the other two methods (Liu et al. 2004).

Knowledge-Based Matching utilises the knowledge of the internal structure of characters. The knowledge is formulated and represented as heuristic rules or constraints. The constraints reduce the search space of structural matching efficiently. Radical detection and stroke reordering have been performed by rule-based methods. The rules hold the information of permitted basic strokes for a character or fixed spatial feature of strokes. Building knowledge-based systems can be laborious, because of the tasks

of acquiring and organising the knowledge. However, even simple heuristics have proved to be valuable. Some heuristics can be acquired in an automatic fashion, or from corpus studies, for example statistical distribution of stroke order for characters. With rule-based heuristics, the overall recognition accuracy of systems based on other classification methods can be improved (Liu et al. 2004).

3.5.4.3 Probabilistic Classification

The use of probabilistic methods for classification has increased. Many modern OLCCR systems use probabilistic methods successfully. For example, the stroke-type probability can help computing the between-stroke distance. Alternatively, if the prototype strokes are modelled as Gaussian density functions, the matching score of an input pattern respective to a character model can be computed using the joint probability of constituent strokes (Liu et al. 2004).

HMM-based Classification has been a popular method in recent years of research in OLCCR systems. The recognition task is the decoding of the observed sequence (i.e. the input sequence) into the most probable state sequence. This is done with the Viterbi algorithm, that is very popular for these types of decoding tasks, such as automated translation (Koehn et al. 2007). Radicals can be represented in HMMs and therefore be shared between different characters in the character database. Hu et al. (1996) have incorporated HMMs into a stochastic language model for Latin characters. Tokuno et al. (2002) employed a substroke model for cursive Kanji and Hiragana handwriting recognition. Their model contains contend-dependent substrokes and their character model therefore needs 25 HMMs as opposed to one for each character. The substroke approach became popular, and has been used by (Nakai et al. 2001). In a follow-up article, (Nakai et al. 2003) propose the automated generation of a dictionary for stroke-order free OLCCR. That approach is as well based on substroke HMMs. (Shimodaira et al. 2003) propose a handwriting recognition interface for wearable computing. In their approach the HWR has do be done overlaid, i.e. the user draws sequential characters in the same box. The resulting segmentation problem is solved by using substroke HMMs, where recognition is done in parallel with segmentation. In order to achieve that, a stochastic bi-gram language model is combined with the HMMs. Another system (Okumura et al. 2005) uses not only the pen-direction feature of the other substroke-based HMM approaches, but additionally incorporates the pen-coordinates into the system. This is done at the inter-state transitions of the HMM, whereas the pen-direction feature is utilised at the intra-state transitions.

3.5.5 Postprocessing

3.5.5.1 Contextual Processing During Recognition

OLCCR systems, focus on finding the appropriate candidate for an input pattern. Once the correct character has been found the task is solved. Most systems deal with extraction of features and classification or matching, resulting in scores for individual characters. There are systems that have to deal with multiple character input. That can make the problem more complex, for example when dealing with overlaid handwriting and the accompanying segmentation problem (Shimodaira et al. 2003).

In Japanese, it is possible to write from left to write in lines, while the lines go from to to bottom. It is also possible, to write from top to bottom in columns, while the columns go from right to left. The task of recognising multiple character input becomes more difficult, when attempting to recognise Japanese handwriting without assumptions about the writing direction (Nakagawa et al. 2008). However, it is possible to make use of the linguistic context for contextual processing of already recognised character classes. The context can provide additional information about which character class is the most suitable among a selection of possibilities with their scores. Also, geometric features like size, location or aspect ratio can support the segmentation process.

For page-based recognition processes - especially in the case of off-line recognition, but also when handwriting input can be placed freely on some input device with a stylus-based or touch display - it is important to integrate the segmentation and the recognition modules. Frequently, segmentation can not be done unambiguously before the recognition (Liu et al. 2004).

Some segmentation ambiguities need to be solved during the recognition process. Usually, the systems generate some candidate character classes and verify them by their geometric features, the recognition results and also linguistic knowledge. The candidates can be represented in a network, where the

edges denote combinations or segments that build up the candidate pattern. Each path in the pattern represents a segmentation of the input and its recognition result. A dynamic programming search yields the path with the highest probability value. Linguistic knowledge can be used to verify the path or can be inserted into the path scores (Liu et al. 2004).

3.5.5.2 Contextual Processing After Recognition

Linguistic processing of system results after segmentation and recognition is called *postprocessing*. For instance, based on the recognition result, other candidate characters can be given, due to statistical information about confusion of characters. That procedure reduces the risk of excluding the true class from the candidate set.

Additionally to that, there are systems that ruthlessly exploit linguistic knowledge in dictionaries or character-based or word-based n-grams models. Word-based n-gram models usually provide syntactic or semantic classes, for example parts of speech. Using n-gram models involves dividing text into words and some degree of morphological analysis. It is beneficial to use writer-specific dictionaries. Generally, linguistic postprocessing improves the accuracy of the recognition (Liu et al. 2004).

Chapter 4

E-learning

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

4.1 General E-Learning methods

4.2 E-Learning of languages

in section
xxx: s. 12 beachten: WICHTIG!

4.3 E-Learning of Japanese

4.3.1 Conceptual issues

4.3.2 Japanese e-learning software

put all your bashing and criticism here
shortcite nagata2002 not in bibtex yet
(Bailey and Meurers 2008) (Zimmer 2009) (Stahlmann 2004)

Chapter 5

Conceptual Design of Kanji-Coach

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

5.1 General Requirements

Wichtige Fragestellung: Wie sieht eine HWR in Lernumgebung aus? Mappe S. 19 genaue anforderungen s. 19! waehle bestimmte architekture unter moeglichen ansaetzen.

s. 12 beachten: WICHTIG: lernkomponenten, muss kein ganzes system sein.

5.1.1 Character Learning Aspects

- greife typische probleme der lerner auf (siehe japanischkapite 2.3) s. 11 hinten kurze auflistung. geschichten?

5.1.1.1 Character Repetition

5.2 Tackling the Specific Difficulties of the Japanese Script

5.3 Integration of HWR Into the Learning Process

welche art von character recognition muss geleistet werden?

was sind die moeglichkeiten (im vergleich zu anderen produkten), die sich durch eine HWR ergeben? wie kann man die ausschoepfen? s. 16 unten und s. 15

Error Recognition what type of errors? semantical errors? cow vs sheep vs pig phonological errors (readings) kanji that sound the same. theoretically: compounds - for the kanji readings. heft: s. 52

- compare with normal paper-based learning of kanji - compare with other kanji-learning systems klare abgrenzung von skritter. s. 51 unten im heft.

5.4 Use Cases

siehe 'screenshot' - grafiken von s. 2 - 9 auch: was kann man aus e-learning machen? welche (technischen) moeglichkeiten sind eroeffnet, insbesondere auch durch handschriftenerkennung?

idee: schoenschreibekurs, bei dem einzelne striche gesondert geuebt werden.

5.5 HWR Applied to E-learning of Japanese Kanji

5.5.1 Integration of HWR Into E-Learning Application

educational aspects / the e-learning view

5.6 Handling Errors

5.6.1 Possible Sources of Error When Writing Japanese Characters

error handling, see page 58.

See notes on paper, seite 58 - for example stroke number and stroke sequence - length of strokes - stroke velocity

Chapter 6

Technical Design of the Application

The focus of this chapter is on the general architectural choices made during the development of the system. In this chapter, the technical design aspects of the application are described. The general system architecture is layed out in section 6.1. It contains the global view on the software architecture in section 6.1.1, the data flow in within the system in section 6.1.2 and describes the design of the individual modules in section 6.1.3. Section 6.2 describes the technical set-up and framework choices. However, the handwriting recognition engine is described in detail in a separate section (see chapter 7).

6.1 System Architecture

The system architecture of the Kanji Coach follows the requirements of an e-learning environment dealing with the specific difficulties for learners of the Japanese script (see chapter 2) and those of an on-line handwriting recognition. Techniques of handwriting recognition are reviewed in chapter 3. The general requirements of an e-learning application are presented in chapter 4. The resulting specific conceptual design choices have been layed out in chapter 5. This section deals with the technical aspects of the system design.

6.1.1 Global Architecture

The global architecture of the application follows the Model-View-Controller (MVC) design pattern. This paradigm is used as a general model, however, it is not designed the strict way proposed by (Krasner and Pope 1988). Figure (6.1) shows the general set-up of the MVC design pattern after (Krasner and Pope 1988). xxx: Also figure (6.2) - decide how it should be done and use the appropriate gfx. xxx! In the MVC paradigm the *model* is a domain-specific software, an implementation of the central structure of the system. It can be a simple integer, representing a counter or it could be a higly complex object structure, even a whole software module. The *view* represents anything graphical. It requests data from the model and displays the result. The *controller* is the interface between the model and the view. It controls and scheduls the interaction between the input devices, the model and the view (Krasner and Pope 1988).

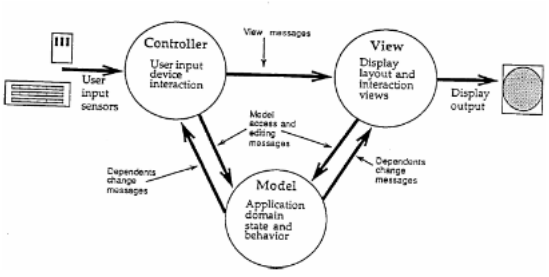

Figure 6.1: The Model-View-Controller paradigm

A global overview of the system architecture can be seen in figure (6.3).

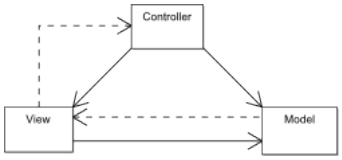

Figure 6.2: The Model-View-Controller paradigm AGAIN!

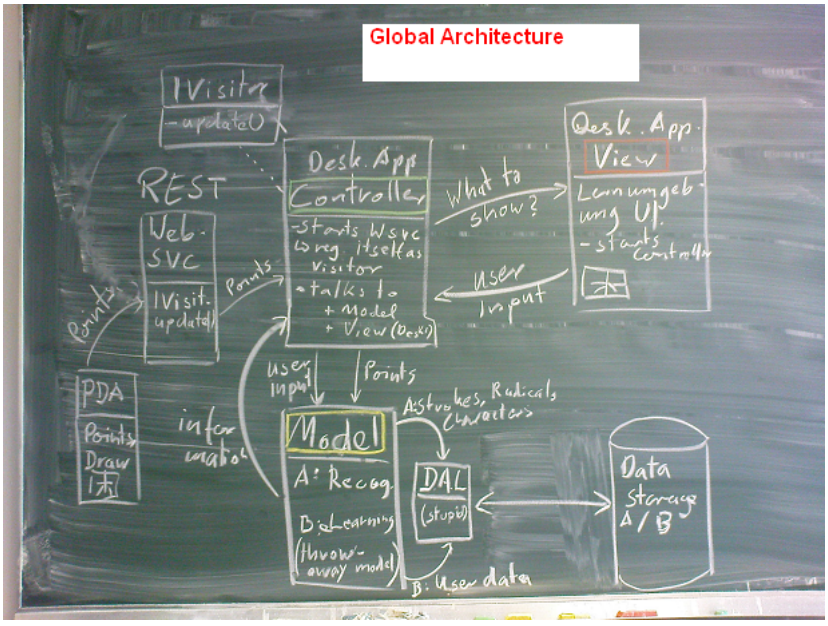

Figure 6.3: The global architecture of the software system

6.1.2 System Data Flow

The system data flow is shown in figure (6.4). The controller lies in the centre of the application, it runs

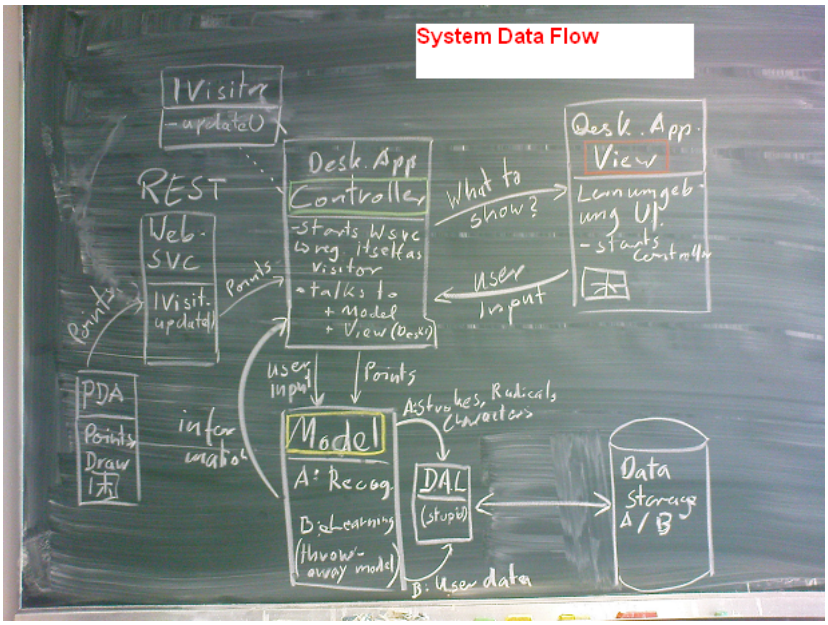

Figure 6.4: The data flow within the software system

on the stationary device. It contains a web service that is used as an interface to receive data from the handwriting input data view. The desktop view is the main interaction point for the user. The model contains the logic, while the data access layer provides a reusable interface for storing data. The details of figure (6.4) are described in subsequent sections 6.1.2.2 and 6.1.2.3.

6.1.2.1 Communication

The communication between the different parts of the application is realised in two independet and distinct ways. The communication between the modules running under the same process is realised via a messaging system. The communication between the modules that run on different devices or at least as a separate process is realised via a web service.

Loose coupling is used here as a term to emphasize that different modules in a larger system are only loosely connected and do not depend largely on each other. *Coupling* can be understood as the degree of knowledge a module or class have of each another. The lesser the knowledge of each other the software modules can manage with, the more loose the coupling.

In the course of the design and development cycles of the Kanji Coach, it became apparent that it has to be possible to attach different views, input devices and data storage systems to the controller. This is due to the distributed nature of the application. In order to ensure that the handwriting data input view, which currently runs on a mobile device, can run on a different device, it had to be loosly coupled to the main controller. Therefore, the communication between the handwriting input data view and the main controller is realised via a web service. Because of this communication structure, it is possible to exchange the handwriting data input view with a different one, for instance when running the application on a device like a tablet PC. The design of the communication structure within the web service is described in greater detail in section 6.1.3.3.

The messaging system that forms the communication structure between the software modules running as the same process is realised with a message class that can be manipulated by the modules using these messages. Technically, the web service and the controller both run as a subprocess of the main desktop view. When the web service receives a request and accompanying data from the handwriting input data view, both request and data are bundled into an encapsulated message and passed to the controller. A similar type of message is used for requests from the controller to the model, for instance a recognition task that needs be performed.

6.1.2.2 Recognition Data Flow

The recognition data flow is simply the flow of data occuring in a recognition scenario. It can be described in 6 steps as depicted in figure (6.4).

1. Clearing the input GUI screen [Controller ⇒ Web service ⇒ Handwriting data input view]
2. Transmitting user input data to the web service [Handwriting data input view ⇒ Web service]
3. Encapsulating data into a message and passing it to the controller [Web service ⇒ Controller]
4. Requesting recognition [Controller ⇒ Model]
5. Returning recognition result [Model ⇒ Controller]
6. Displaying recognition result [Controller ⇒ Desktop view]

When the desktop application requests the use to input a Kanji character, it sends a message to the web service advising the handwriting data input view to clear the screen (step 1). The handwriting data input view receives the information by polling the web service. When the user starts drawing on the surface of the handwriting data input view, the data is caputered and transmitted to the web service (step 2). Each stroke is captured and sent individually in order to ensure a faster recognition. That way, the recognition process is initiated before the user finishes writing a character. The web service receives the data and creates an encapsulated message. That message is passed to the controller (step 3). The controller initiates a request to the model, containing all strokes subsequent to the last clear screen event (step 4). The model performs the recognition of the set of strokes and returns the result or partial result to the controller (step 5). The controller advises the desktop view to display the resulting character (step 6).

6.1.2.3 Learning Data Flow

The learning data flow is the data flow between the learning module of the application, the recognition process and the user interaction. The learning data flow design can be summed up in 5 steps.

1. In learning mode or test mode: Asking user to draw a character, based on the current lesson data. The controller sends a display request to the desktop view. [Controller ⇒ Desktop view]
2. After recognition process: Request storing recognised character in learning profile. [Controller ⇒ Model]
3. Calculation of error points for a character (creation of new data) and storage. [Model ⇒ Data access layer]
4. Returning learning state of character [Model ⇒ Controller]
5. Displaying learning state [Controller ⇒ Desktop view

The learning data flow is described in the middle of a user interaction with the learning application as it illustrates a typical data flow most appropriately as shown in figure (6.4).

In step 1 of the learning data flow, the controller sends a message to the desktop view with a request to display an invitation to the user to draw a specific character. That step is a part of the general messaging system design that manages the communication between the controller and the other modules. When the recognition process is finished, the character needs to be stored. The controller requests the model to store the character (step 2). The logic layer then creates new data, namely the error points of a character that naturally become a part of the data flow. Both the character recognition result and the error points are stored using the data access layer (step 3). The resulting learning state defines which character will be displayed next. This calculation result is transmitted from the model to the controller after storage in the penultimate step (4). The last step in the learning data flow is the display of the resulting learning state to the user in the desktop view (step 5).

6.1.3 Software Modules

6.1.3.1 Handwriting Data Input View

The handwriting data input view is a graphical user interface. It is designed for simplicity and usability. It's main task is data capturing. It contains only an input area, with a cross in order for the user to better locate the strokes of the character. This follows a common practice in Kanji teaching - a cross departs the writing area in four areas.

There is no *commit* or *finish* button on the handwriting data input GUI, since the end of a stroke sequence is handled with a time out in the learning module. When the user needs too long to input a character there is a problem and help should be offered. Therefore, it is sufficient to only display a reset button on the writing area. Whenever the reset button is pressed, the controller is notified of that event. The current drawing will be removed from the screen. The next user input will be treated as a new character

In the background of the handwriting data input view the user input is sent to the controller module. Besides that, a polling mechanism ensures that any messages from the controller to the handwriting input data view are received. Whenever the user finishes a stroke, the module sends a message via a web service (see section 6.1.3.3) to the controller.

6.1.3.2 Main View

The main view of the system is a graphical user interface. It contains a simple learning environment, a display area for the characters that have been drawn in the handwriting input area an information area that displays addidional information about a character. In a configuration area the user can choose between the different lessons the system offers and between a training mode and a test mode.

6.1.3.3 Web Service

The main communication module between the handwriting data input view and the controller is a web service that runs on the main part of the application as a subprocess of the controller. When the desktop application requests the user to input a Kanji character, it sends a message to the web service advising the handwriting data input view to clear the screen. The handwriting data input view receives the information by polling the web service. However, the main task of the web service is to receive data from the handwriting input data view and forward that data to the controller of the system.

A web service is a standard means of interoperation between different software systems, possibly running on different platforms and frameworks. The web service as such is an abstract definition of an interface, it must be implemented by a concrete agent. The agent itself then must be a software that sends and receives messages (Booth et al. 2004). The web service in the Kanji Coach system is not a web service in the strict sense as intended by the Booth et al. (2004). It does not provide a service as such. The calling client does not receive a reply to a request, just an acknoledge message stating that the data has been received. A reply with more information is not necessary, since the handwriting data input view does not display any information to the user. In summary, the web service design for the Kanji Coach system is a web service in the technical sense, but not in the conceptual sense, as there is no real crosswise interaction between server and client. The concrete implementation of the web service is realised with the *Windows Communication Foundation* (WCF) (Smith 2007). The WCF uses a *SOAP protocol* (originally: *Simple Object Access Protocol*) internally, but the functionality can be accessed from within the .NET framework, the concrete SOAP XML messages will be created and unpacked into instances of *Common Language Runtime* (CLR) objects automatically by the WCF framework (Kennedy and Syme 2001; Gudgin et al. 2007; Smith 2007).

6.1.3.4 Recognition Module

As a part of the model within the MVC paradigm the recognition module plays a crucial role. From an abstract viewpoint it functions as a black box. A set of pen trajectories is passed to to the recogniser and a set of characters is returned as a result, sorted by their matching value. Additionally, an intended character can be passed to the recognition engine, such that the recognising module can generate an error analysis based on the intended character. If no intended character is given, the recogniser is designed to use the best result as a basis for an error analysis, with lower confidence values attached.

The detailed design and implementation choices of the handwriting recognition engine are described in chapter 7.

6.1.3.5 Learning Module

The learning module is designed as a basic module, mainly for the purpose of demonstration. It performs user and character administration tasks. Each user has a personalised character set, containing the lessons that have been studied and the error points of each character. The application is therefore not a vocabulary teaching application, but rather a Kanji teaching application. The error point system is rather simple, the more error points a character has, the more often it will be repeated. Similarly, the character repetition depends on a vocabulary training method, that is employed in a systematic fashion. For details see section 5.1.1.1.

6.2 Framework and Devices

In this section the choices for framework and devices are reviewed briefly. The decision of which framework to use depends on the operating system.

6.2.1 Operating System

Despite the quasi-religious views of which operating system (OS) to use, the choice for the Kanji Coach system was simply user oriented. The user group 'students of Japanese' consists mainly of non-technical people. They usually use a version of the MS Windows operating system. Therefore, even when attempting to provide for cross-plattform usability, the windows platform is chosen as the basis for the implementation. Neither the author nor the application seek to convince anyone of using one or the other operating system, therefore it is a logical choice to provide software for a system that most people use.

6.2.2 Framework

The framework to run the system had to fulfill a few criteria.

- It should provide an easy-to-use graphical user interface.
- It should mirror the natural look-and-feel of the operating system it runs on.
- It should be cross-plattform if possible and not be bound to native OS-specific code.
- It should provide cross-device communication.
- It should provide a modern high-level programming language

Even though native code that runs directly on the operating system may provide for speed and natural look-and-feel, any efforts to port the system to a different platform become laborious tasks. That rules out the choice of *C++*. Generally, the design ideas and framework criteria seem to apply well to both the Java platform and the .NET framework. Both provide functionality for an easy-to-use GUI. .NET does not mirror the natural look-and-feel of the windows operating system, it *is* the natural look-and-feel on Windows. Java has the ability to adopt the look-and-feel of different systems it runs on, MS Windows as well. The Java virtual machine has long been known to run on different platforms, most importantly Linux and Windows, while the .NET framework has been implemented for X-Systems by Novell and is called *Mono*. The two differences that gave the .NET framework a slight advantage over Java was the comfort and ease the Windows Communication Foundation offers. The web service functions can be called exactly the way as if they were methods of a local software module. The integration of the different parts of the system, namely the handwriting data input view and the controller does not require much additional effort. Another advantage of the .NET framework is the programming language *C#* that combines many features of the Java with some of C++. Additionally, C# offers a smoother handling of generic lists, which is useful when dealing with lists of mouse coordinates, lists of strokes that are lists of mouse coordinates and ultimately lists of characters. With the CLR, which is an ECMA standard like C#, it is also possible to easily integrate with other languages available for the .NET environment.

Lastly, despite the ease of integration with a .NET client, the WCF also allows for other systems to connect to a webservice. Since it uses SOAP internally, it is possible to communicate with a WCF web service even without .NET, using so called *POX* (Plain Old XML) messages that can be generated in any programming language on any framework and operating system.

6.2.3 Desktop Computer

The choice for a stationary computing device is driven by the kind of user the learning application is aimed at. A notebook computer the way it can be bought in any computer store is the closest match to what a user of the system most probably will be running. The learning system is a standard desktop application with a standard MS Windows forms GUI. Therefore, this kind of system is most suitable for the designed system. Nevertheless, it is possible to run both the handwriting data input view and the desktop application on the same PC - for example a tablet PC. The integration could stay exactly the same.

6.2.4 Pen Input Device

The pen input device has been chosen to be a *Personal Digital Assistant* computer (PDA). The reason for that design choice is simple. At the time of drawing the user should be able to see what he is drawing on the writing surface. At the same time the data should be available to the controller in real-time, i.e. after a pen-up movement. The main decision had to be made between a graphics tablet and PDA. The option of mouse input of charaters was ruled out from the beginning, because the mouse is not a useful device to enter handrwitten characters. Graphics tablets like a Wacom tablet are cheaper than a PDA, however, their distribution on the market and therefore among potential users is small. The solution with a PDA leaves room to exchange the device that runs the input area with any mobile device that has the ability to connect to a wireless LAN, e.g. an iPhone. Another possibility would be pens that write on paper and also transmit data to a PC. Those, however, either do not perform the transmission in real-time, or do not come as plain mouse coordinates but bitmaps or can not connect to custom made software at all, but only to their OEM software.

Chapter 7

Handwriting Recognition Engine

7.1 Data Capturing

Each handwriting recognition process begins with the data capturing. The user's handwriting must be captured and fed into the system. The data capturing is therefore a crucial part of the whole process. In this system a GUI is used for the capturing of the pen movements on a writing surface.

7.1.1 Writing Surface

The writing surface module, the view is split into two parts. The writing surface GUI and the writing surface background module. The technical design of the data input GUI is described in section 6.1.3.1.

7.1.1.1 Writing Surface GUI

The GUI works with pen-down and pen-up events. It has a cross in the middle in order to partition the writing surface the same way, character practicing paper sheets are usually partitioned. The GUI class is listening to *pen-down*, *pen-up* and *pen-move* events. These are the equivalent in the mobile world for regular mouse-down, mouse-up and mouse-move events. However, there is one difference - the pen-move event can only be captured between a pen-down and a pen-up event. An earlier conceptual idea for the HWR engine included using the mouse-move events during the input of a character between the strokes. This could not be realised, however, because the series of pen-move events can only be captured when there has been a previous pen-down event and no pen-up event yet. The GUI captures the events and passes the point coordinates on to the background class.

7.1.1.2 Writing Surfaces Background Module

When a pen-down event is detected, the background class of the GUI starts listening for the pen movement. All point coordinates and the time of their capturing are stored in two separate lists with the same indeces. An alternative solution would have been to store a number of instances of a custom-made encapsulated Point class including the time stamp in one list only, however, using two separate lists resulted in increased speed. Therfore, the point coordinates are stored in the frameworks Point class that does not account for time stamps. Therefore a separate list is used for the timestamps only.

The background module of the writing surface mainly administrates the capturing of the pen trajectory and sends it to the recognition module as soon as a stroke is finished. Therefore the system does not receive a separate signal indicating that the drawing of the character is finished. That design creates as segmentation problem that is solved with the *clear* button and the *clear* message. The segmentation of characters is left undetermined - only the beginning of it is determined through the *clear* message that is sent to the moblie view from the controller - or the clear message that is sent to the controller because the user clicked the *reset* button. After a stroke is finished the according point and timestamp sequences are passed to the main part of the HWR engine.

7.2 Data Format

7.2.1 Requirements of the Data Format

The data format for the recognition process underlies a number of requirements. Firstly, it has to be stated that two main data formats are necessary. One for the actual recognition process as a data structure in the RAM. Secondly, there needs be a storage format for the data base that represents the handwriting of a character, radical, stroke, point list or simple point. The requirements for both data structures are:

- **Expressiveness**: The data structure should be able to fully represent a complete character and all sub-elements that belong to it. That requirement is due to the structural approach to handwriting recognition that is the basis for the error handling.
- **Well-definedness**: The information structure should be unambigous, two different characters must have a different structure.

The storage structure has some additional requirements:

- **Accessibility**: The data should be formatted in a way that a human editor can access the data, but it should also be prepared for programmatical access.
- **Well-formedness**: There should be a way to check if the data is well-formed
- **Parsabilty**: It should be possible to create the run-time data structure from the storage data structure.

The run-time data structure should in addition provide

- **Serialisability**: It must be possible to create the storage data structure from the run-time data structure.

Any combination of two formats used should meet the above requirements, at best in a uniform way. That is, any combination of storage format and run-time format should ideally be compatible without the need of a complex data format converter.

7.2.2 Existing Handwriting Formats

There are some existing formats for the description of handwriting. Their intended use is for handwriting recognition systems. The formats that will be reviewed in this section are the *Microsoft ISF* format in section (7.2.2.1), which is a binary format. The other formats are text based and human-readable. In this short review, the formats will be checked against the requirements defined in section (7.2.1). We will consider *InkML* in section (7.2.2.2), *UNIPEN* in section (7.2.2.3), *hwDataset* in section (7.2.2.4) and *UPX* in section (7.2.2.5). The central question is, if one of these formats meets the above requirements in a sufficient way. This is unclear, because the most of the formats have been originally developed mainly for alphabetic scripts or for usage scenarios different from the one proposed in this chapter.

7.2.2.1 The Microsoft Ink Serialized Format (*ISF*)

The *Ink Serialized Format* (ISF) format is owned by the Microsoft Corporation. However, it is not a proprietary format, the specification is freely available. Since the format is purely binary, it can not meet one of the requirements for the storage data structure: It is not feasible for a human to read and write files in that format, therefore ISF does not provide the necessary *accessibility* (Microsoft Corporation 2007). The format could still be used as a run-time data structure, if it meets the requested criteria. The requirement of *serialisability* is generally met. A binary format allows for fast and reliable programmatical access to the data, it can be stored in binary files on a data storage medium. The problem that accrues from this type of serialisation is that the storage format, the format used in a file or database, would again not be human-editable and therefore fail to meet the *accessibility* requirement. In order to provide a serialisation that results in a human-readable format, a format converter would be needed. Another problem originates from the ISF format specification:
It accounts only for the description of mouse coordinates or pen trajectories, but does not offer any structures for linguistic information. It fails to meet the ideal, a unified format for linguistic and graphic information, as it is lacking some degree of *expressiveness*.

7.2.2.2 InkML

InkML is a markup language for describing electronic pen trajectories. Hence the name *ink*. *ML* stands for markup language, as is customary for XML-based and SGML-based languages. The specification is currently in a draft status, maintained by the World Wide Web Consortium (W3C) (Chee et al. 2006). Judged from the viewpoint of the defined requirements, InkML is a candidate for the storage format. As an XML format it automatically fulfils the criterion of *well-formedness*: Any piece of InkML code can be evaluated with common techniques, since InkML has a well-defined scheme description.

The main elements in the simplest form of InkML are the **<ink>** and the **<trace>** element. These elements allow for a very simple form of a pen trajectory, basically a flat list of point coordinates. The <ink> element is the root element of any InkML code. The <trace> element holds point coordinates.

Listing 7.1: Demonstration of the *trace* tag

```
<ink>
  <trace>
    282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
    291 143, 294 157, 294 171, 294 185, 296 199, 300 213
  </trace>
</ink>
```

Listing (7.1) covers the general gist of the InkML format. Despite being an XML format, it is a flat format, each value pair, separated by commas, represents one point in a coordinate notation.

It is possible to add *time* information to the trace. In order to do so, a time channel needs to be defined. Listing (7.2) shows the definition and use of a time channel. The example shows a time channel whose values for a given point are the relative to the timestamp refered to to by *#ts001* (Chee et al. 2006). Below there are two *timestamp* tags. The first one has the ID *ts001* and is referred to by both the time channel and the second time stamp that defines a time offset to time stamp *ts001* with the *timeOffset* attribute. The possibility to define time channels enables the InkML format to hold information about the time at which a sample point has been taken. This is useful for the recognition mechanism used in this application, because it compares point coordinates as well as time stamps.

Listing 7.2: Demonstration of the *time channel*

```
<channel name="T"
         type="integer"
         units="ms"
         respectTo="#ts001"/>

<timestamp xml:id="ts001"
           time="1265122414000"/>
<timestamp xml:id="ts002"
           timeOffset="600000"
           timestampRef="#ts001"/>
```

The InkML data structure fulfills several of the requirements necessary for a storage data structure. InkML is

- *well-defined*: Any point sequence that has different coordinate values than another point sequence can be distinguished from the other.
- *accessible*: As an XML format it can be handled programmatically, but is also human-readable and can be edited by a human with a simple interface like a text editor.
- *well-formed*: As an XML format it can be validated according to it's specification.
- *parsabale*: As an XML format, InkML does not need to be parsed with custom-made methods, since most modern high-level languages offer an access method to XML tree structures.

However, InkML is lacking some *expressiveness*. Only pen trajectories and their time stamps can be expressed in the InkML format. InkML is not designed to hold any of the other information compulsory for the structural handwriting recognition. In the case of character recognition, it needs to be able to

account for more than coordinate points and time stamps. There needs to be a way to encode structural information about the characters and sub-elements of the characters. InkML does not to be a sufficient format for the given task.

7.2.2.3 The Standard UNIPEN Format

The standard UNIPEN format specification is a file format definition for a flat text file. It contains tags in dot-notation. For example the tag *.COMMENT* means that the following free text should be ignored, the tag *.KEYWORD* is used to define a new keyword, while the tag *RESERVED* states that the text that comes after that tag is a reserved word within the UNIPEN format. The format is self-defined with the three keywords above. Any new keyword is defined with *.KEYWORD* (Guyon et al. 1994; Unipen Foundation, International 2010).

As a data structure for the purpose of a handwriting recognition, UNIPEN can serve as a storage format, which is the primary purpose for the existence of the format. The UNIPEN format accounts for both pen trajectories as well as information about the characters that have been written.

Listing 7.3: Demonstration of the *UNIPEN* format

```
.COMMENT ##################################################################
.COORD   X Y

.SEGMENT TEXT 235:0-297:9 OK "Kurosu␣Masaaki"
.SEGMENT CHARACTER 235:0-255:9 OK  "JISx3975␣'Kuro'"

.PEN_DOWN
  486 -1456
  488 -1454
  490 -1452
  488 -1450
  488 -1450
  486 -1452
  480 -1456
  474 -1466
  464 -1480
  452 -1492
  440 -1506
  428 -1524
.PEN_UP
  406 -1556
  394 -1574
  384 -1590
  374 -1602
.PEN_DOWN
```

Listing (7.3) is an excerpt from a data file created by (Unipen Foundation, International 2010). It shows an example of a UNIPEN data structure. The *.COORD* tag defines the structure of the pen coordinates. The *.SEGMENT* tag with the *TEXT* modifier informs about the full text of the next number of segments. The *.SEGMENT* tag with the *CHARACTER* modifier informs about the character that is represented with the sequence of pen coordinates. The other information given is the start and end position within the file and the character code in JIS encoding. The *.PEN_DOWN* tag can be understood as a flag. All pen coordinates following this tag have been captured as *pen down* coordinates. The tag *.PEN_UP* sets the opposite flag: The coordinates stated after this tag were captured while the pen was not touching the writing surface. The last *.PEN_DOWN* tag is the beginning of a new coordinate sequence for the next stroke. From a requirements point of view the UNIPEN provides a high standard, as it meets a great number of requirements for the storage structure.
The UNIPEN format is:

- partially **expressive**: UNIPEN can represent a complete character and the pen trajectory along with it. It does however, not account for capturing the time information. At a constant sample

rate the time stamps of the individual points could theoratically be calculated, if time stamp of the first point is encoded in the meta information. However, that solution is not optimal, because it needs constant sampling, even between a *pen-up* and a *pen-down* event. There are input devices that do not offer those coordinates and the recognition system presented here does not rely on them for that reason. The examples given are encoded in JIS, but it seems possible and feasible to use the format with unicode encoding.

- **well-defined**: The UNIPEN is unambigous, two different characters do have a different data structure, because the JIS code can serve as an ID.
- **accessibile**: The data resides in flat text files, designed for human editing. It is accessble programmatically, too, but has a weakness on that requirement compared to an XML-based format.
- **parsable**: The flat file format can be parsed easily. Due to its procedural nature it is also possible to create a run-time data structure that is conceptually based on the storage structure and therefore serialisable.

The main weakness of the flat format file is the lack of *well-formedness*. There is no automatic way to check a document against a predefined format specification. It can not be assessed if a file is well-formed. Additionally, the expressive power of the UNIPEN format would be seriously challenged if it should provide for sub-structures of characters. In order to do so, it would be necessary to create new tag definitions for the Radicals of the Kanji. Agrawal et al. (2005) describe some more shortcomings of the UNIPEN format. It can be concluded that the standard UNIPEN format is not suitable for the task given.

7.2.2.4 Handwriting Dataset (*hwDataset*)

The *Handwriting Dataset* (hwDataset) is an XML format that is a complement to InkML. It is inspired by the UNIPEN format. The *hwDataset* format attempts to close the gap between pure ink data and the annotations that are needed for handwriting recognition (Bhaskarabhatla and Madhvanath 2004). The format contains three main parts. The *datasetInfo*, the *datasetDefs* and the *hwData*. The datasetInfo element holds any metadata, like *name*, *category* and the like. The dataSetDefs encompasses information about *data sources*, different *writers* and their features like *handedness*, *gender*, *age*. The *hwData* element in the XML code organises the handwriting data hiearchically. Each hierarchy level contains one or more *hwTrace* elements. The hwTrace element refers to an InkML file, containing the actual handwriting data. A detailed description of the hwDataset format can be found in (Bhaskarabhatla and Madhvanath 2004).

The format provides an XML scheme description and thus meets the *well-formedness* requirement. It is *well-defined*, because it accounts for unique structures. The *parsabilty* is not an issue for any XML format, since methods for accessing XML trees are provided by modern programming libraries. The *accessibility* is provided, the format has a clearly defined structure that can easily be understood by humans. The hwDataset format can structure many desirable information for the handwriting learning system proposed in this chapter. From an *expressiveness* viewpoint it can create substructures for different parts of a sequence of pen trajectories. The *hierarchy level* (H) of a trace accounts for the expressive power necessary. The H(n) elements of the hwData elements are designed to hold meaningful names like *PARAGRAPH* or *WORD*. It may be possible to use the format even to denote different parts of the same character. There is no reason why one should not introduce an H(n) definition for *RADICAL* or *GRAPHEM*.

7.2.2.5 The UNIPEN XML Format (*UPX*)

The format *Unipen for XML* (UPX) can be seen as an XML version of the UNIPEN format. The UNIPEN standard described in section (7.2.2.3) does not bear any resemblance with the InkML format shown in section (7.2.2.2). Both formats fulfil a number of the requirements presented in section (7.2.1), but both formats fail to fulfil them all. The hwDataset format described in section (7.2.2.4) solves many of the problems UNIPEN faces, as it brings together a superset of the expressiveness of UNIPEN with the well-defined InkML format in a uniform way. According to Agrawal et al. (2005) hwDataset was designed to support new data collection. The motivation to create another format came about from the fact that existing UNIPEN resources should be made available in a modern XML-based format that

unifies InkML and UNIPEN. hwDataset comes very close to this aim, but does not necessarily account for a conversion of UNIPEN resources. Agrawal et al. (2005) attempt to create a format that allows for a well-defined conversion from UNIPEN. The UNIPEN foundation considers UPX as the new de facto standard format for storing annotated databases of online pen input (Unipen Consortium 2006).

UPX is a multi file format. It resembles hwDataset in the way that it comprises both meta information and data annotations as well as pure InkML data in separate files. Within the UPX *hwData* element there can be any number of *hLevel* elements in order to define the hierarchy of the trajectory. Listing (7.4) shows an example after (Unipen Consortium 2006). In the listing, the *hLevel* tag is demonstrated. The word *sexy* is described hierarchically. Firstly, in the outermost *hLevel* tag, the *level* attribute specifies a *WORD*. Then the inner *hLevel* elements specify a lower level of hierarchy, the characters. Note that not all characters are defined individualy in this example. The *hwTraces* elements surround a number of *traceView* elements that in turn point to the corresponding positions in an InkML file that holds the pen trajectory. The meta data information that can be stored in UPX has a similar expressive power as the hwDataset format. Madhvanath et al. (2006) give a full format description of the UPX format in their technical report.

Listing 7.4: Demonstration of the *hLevel* tag in UPX

```
<hLevel id="WORD_0" level="WORD">
  <label id="WORD_0">
    <alternate rank="1" score="1"/>sexy</alternate>
  </label>
  <hwTraces>
    <inkml:traceView traceRef="w0629.inkml#dataSet_0_traces"
                     from="22"
                     to="24"/>
  </hwTraces>

  <hLevel id="CHAR_0" level="CHAR">
    <label id="CHAR_0">
      <alternate rank="1" score="1"/>e</alternate>
    </label>
    <hwTraces>
      <inkml:traceView traceRef="w0629.inkml#dataSet_0_traces"
                       from="22:91"
                       to="22:161"/>
    </hwTraces>
  </hLevel>

  <hLevel id="CHAR_1" level="CHAR">
    <label id="CHAR_1">
      <alternate rank="1" score="1"/>y</alternate>
    </label>
    <hwTraces>
      <inkml:traceView traceRef="w0629.inkml#dataSet_0_traces"
                       from="24:61"
                       to="24:162"/>
    </hwTraces>
  </hLevel>
</hLevel>
```

It seems as if the UPX format was able to fulfil all requirements. The UPX format fulfils the requirements of

- *Well-definedness*: As an XML structure providing IDs and names for each element and substructure, the format is well-defined.

- *Accessibility*: As an XML format it is both human and machine readable and editable.

- *Well-formedness*: As an XML format with a defined scheme it can be validated for its syntactic features.
- *Parsabilty*: As an XML format it can be easily parsed into a tree structure.
- *Expressiveness*: The format provides the missing link between the standard UNIPEN format and the InkML format. Additionally, it can describe hierarchical structures.

Since any XML format translates directly into a tree structure, the data structure for the run-time does not need to be defined separately. Conversing pen trajectories in InkML format into a data stream has been studied (Keshari 2008). These techniques however, are not necessary for the proposed system. The data exchange is provided as technique available within the framework without any conversion (see section 6.1.3.3).

7.2.3 Data Format Description

The data format description could trivially be described with the UPX description. In this case however, UPX needs to be adapted to the special needs. This section describes the adaption of UPX to the requirements of an on-line handwriting recognition for Kanji characters in a learning environment. The section describes both the file format and the run-time data structure.

7.2.3.1 Point Data Format

In UPX, points are not represented directly, but in an outsourced InkML file, where they are part of the InkML standard pen trajectory definition. In InkML files, points are stored in traces as specified in the InkML format description.

Listing 7.5: Definition of the trace format

```
<timestamp xml:id="ts001"
           time="1265122414000"/>
<traceFormat xml:id="kanjiStrokeTrace">
  <channel name="X" type="decimal" />
  <channel name="Y" type="decimal" />
  <channel name="T" type="integer"
                    units="ms"
                    respectTo="#ts001" />
</traceFormat>
```

Listing (7.5) shows the trace format description as used in the system. The format consists of three channels that are given in ordered sequence. There are no intermittent channels. The first two channels are X and Y for the pen coordinates. The third channel is T for a time stamp. The pen coordinates can be given as *decimal* numbers, while the time stamp (in milliseconds) must be an *integer*. The *respectTo* attribute in the time channel indicates that all values must be interpreted as offsets to the time stamp with the id *ts001*.

An actual trace would then contain three values, each value triple separated by a comma. The trace format definition ensures the correct interpreation of the values.

Listing 7.6: A sample trace

```
<trace id="id123abc">
  45 76 0, 3 5 100, 4 -8 200, 4 -5 300,
  9 -2 400, ...
</trace>
```

Listing (7.6) shows a trace. Given the file contains the trace format definition shown in listing (7.5), the trace in listing (7.6) is interpreted in table (7.1).

The run-time data structure for a point holds the coordinate values and an integer value for the time stamp. The conversion between the run-time instance and the trace format is trivial and needs no description.

Trace	Channel X	Channel Y	Channel T	Comment
45 76 0	45	76	1265122414000	The first trace initialises the values. The time stamp value is read from timestamp *ts001*.
3 5 100	48	81	1265122414100	The delta values are added to the existing data.
4 -8 200	52	73	1265122414200	The time stamp values are relative to *ts001*, not to the last time stamp, unlike the pen co-ordinate deltas that are added to the current value trace by trace.
4 -5 300	56	68	1265122414300	Negative values are treated as regular delta values.
9 -2 400	65	66	1265122414400	The values change as expected.

Table 7.1: Sample trace interpreation

7.2.3.2 Stroke Data Format

There is no explicit format for a stroke in the storage structure. A stroke is simply the trace between a pen-down event and the following pen-up event. All the points captured between form one stroke. Thus, in the InkML file format a stroke is simply a trace. If a greater number of strokes comes together, each one of them will be a trace and they can be groupe together as a tracegroup element in InkML. It would be possible to create a separate level for strokes, as demonstrated in listing (). However, since every *hLevel* element for a stroke would contain exactly one *hwTrace* element, it seem redundant to do so. Listing (7.7) shows a potential data structure for a stroke. As stated befere, this data structure is redundant in the way, that an *hLevel* definition would contain only exactly one trace. If a trace is interpreted as a stroke, the additional *hLevel* for strokes becomes unnecessary.

Listing 7.7: An example of how a stroke could be represented in UPX

```
<hLevel level="stroke" id="STROKE1">
<!-- The first stroke in this Radical of handwriting. -->
  <label id="STROKE1label" labelSrcRef="labelref_DW" labelType="truth">
    <alternate>[unicode value]</alternate>
  </label>
  <hwTraces>
    <inkml:traceView traceRef="/example.inkml" from="12" to="12"/>
  </hwTraces>
</hLevel>
```

The run-time data structure for a stroke is generated from a trace. Despite not being represented in the storage structure, the run-time data structure for a stroke plays a crucial part in the recognition process. The actual data wihtin the structure is purely a list of point instances. Any other numerical values are computed from the point list, with algorithms that are described in detail in section (7.5).

7.2.3.3 Radical Data Format

The Radical data format is a data structure that works with the UPX *hLevel* element for hierarchical description. Conceptually, a Radical is a hierarchical element in the composition of a Kanji character. With the UPX format specification in hand it is fairly straightforward to define an appropriate data structure. Listing (7.8) shows a UPX data structure for a Radical. The Radical 木 consists of four strokes. Since strokes are not represented explicitely in the data format, there are four *traceView* elements under the *hwTraces* tag.

Listing 7.8: A Radical representation in UPX

```
<hLevel level="radical" id="RADICAL75">
<!-- The first Radical in this Kanji character of handwriting. -->
  <label id="RADICAL75label" labelSrcRef="labelref_DW" labelType="truth">
      <alternate>U+6728</alternate><-- Unicode value -->
```

```
    </label>
    <hwTraces>
      <inkml:traceView traceRef="/example.inkml" from="12" to="12"/>
      <inkml:traceView traceRef="/example.inkml" from="13" to="15"/>
      <inkml:traceView traceRef="/example.inkml" from="16" to="19"/>
      <inkml:traceView traceRef="/example.inkml" from="20" to="23"/>
    </hwTraces>
  </hLevel>
```

The run-time data structure of a Radical is a collection stroke instances. All values for recognition of a Radical are computed from the strokes instances by using different permutations of the strokes to match a database entry. A more detailed description of the Radical recognition process will be given in section (7.6).

7.2.3.4 Character Data Format

The character data format is an ordered list of radicals, according to their standard sequence of writing within the character. In listing (7.9) the character 東 is shown. It consists of the radicals 日 and 木. The storage data structure for characters in general is totally straightforward and exploits the advantages of the UPX format. In the concrete example, the outermost *hLevel* element is used for the character and additional descripitons, the two inner *hLevel* elements are used for the two Radicals of that character. Each *traceView* tag witihn the Radicals represents a stroke. Thus, the data structure implicitely encodes the stroke number of a Radical. Here, both the Radicals 日 and 木 have four strokes each, the Kanji character 東 is comprised of eight strokes in total.

The stroke order of 東 is shown in figure 7.1. Notice, that the Radicals of this Kanji are not drawn one after another. The very first stroke is a part of the 木-Radical, the four following strokes constitute the complete 日-Radical and the last three strokes, again, are part of the 木-Radical. The UPX structure shown in listing (7.9) accounts for that by not specifying a total order of Radicals, but rather a preferred order. The traces are spatialised to their Radical, the order of drawing does effect the InkML traces and ultimately the recognition process, but not the UPX data structure.

Listing 7.9: A character representation in UPX

```
<hLevel level="character" id="CHARACTER71">
<!-- The character 71 consists of the Radicals 72 and 75 -->
  <label id="CHARACTER71label" labelSrcRef="labelref_DW" labelType="truth">
    <alternate>U+6771</alternate><-- Unicode value -->
  </label>

  <hLevel level="radical" id="RADICAL72">
  <!-- The first Radical in this Kanji character of handwriting. -->
    <label id="RADICAL72label"
           labelSrcRef="labelref_DW"
           labelType="truth">
      <alternate>U+65E5</alternate><-- Unicode value -->
    </label>
    <hwTraces>
      <inkml:traceView traceRef="/example.inkml" from="0" to="3"/>
      <inkml:traceView traceRef="/example.inkml" from="4" to="7"/>
      <inkml:traceView traceRef="/example.inkml" from="8" to="9"/>
      <inkml:traceView traceRef="/example.inkml" from="10" to="11"/>
    </hwTraces>
  </hLevel>

  <hLevel level="radical" id="RADICAL75">
  <!-- The second Radical in this Kanji character of handwriting. -->
    <label id="RADICAL75label"
```

```
            labelSrcRef="labelref_DW"
            labelType="truth">
        <alternate>U+6728</alternate><-- Unicode value -->
      </label>
      <hwTraces>
        <inkml:traceView traceRef="/example.inkml" from="12" to="12"/>
        <inkml:traceView traceRef="/example.inkml" from="13" to="15"/>
        <inkml:traceView traceRef="/example.inkml" from="16" to="19"/>
        <inkml:traceView traceRef="/example.inkml" from="20" to="23"/>
      </hwTraces>
    </hLevel>
  </hLevel>
```

Figure 7.1: Stroke order of the Kanji 東

7.3 Database

7.3.1 Database Organisation

The database used in the system consits of two parts. The first part is centered around handwriting information about characters and their substructures, he second part is concerned with lexical information only. The **handwriting part** is a UPX file structure that holds character descriptions, Kanji character unicode values, as well as Radical descriptions and pointers to InkML files holding pen trajectories representing the data structures given in the UPX file. The **lexical part** is an XML file in a format described by Breen (2004). The XML file contains lexical information about characters, such as their key Radicals, their Readings, and their translation into a number of languages. The handwriting part of the database has been discussed in section (7.2.3). The storage data structure layed out there is the structure used. The lexical part is provided[1] in an XML text file.

Listing (7.10) shows a sample database entry of the multi-index database created by Breen (2004). The *character* element is the root of an entry. The *literal* tag holds the digital form of the character. Under the *codepoint* tag, different code points can be stored, in the current version it is Unicode and JIS0208 (Unicode Consortium 2000; Japanese Industrial Standards Committee 1997). The *radical* element holds the number of the key radical of the character. In the current example for the character 東 the key radical is 木, which is number 75 in the classical Radical index. Miscellaneous information that may help a human finding the character in a paper-based dictionary are stored in the *misc* element. The file format provides room for the character index in different paper-based dictionaries and study books in the *dic_number* element. For example the *dic_ref* with the attribute *dr_type="sh_kk"* holds the character index in (Hadamitzky 1995). This information can be very useful for a learner when referring to study books. The *reading_meaning* element holds information about the pronunciation of the character and translations into a number of different languages. The *nanori* elements describe additional Japanese readings that occur only in person- and place names.

With this organisational approach it is possible for different parties to work on different information structures simultaneously. These can be accessed, modified and improved separately. This is due to the encapsulation of the different data structures.

Listing 7.10: Sample lexicon entry for lexical data

```
<!-- Entry for Kanji: 東 -->
<character>
```

[1] Kanjidict2 is freely available from `http://www.csse.monash.edu.au/~jwb/kanjidic2/`

```
<literal>東</literal>
<codepoint>
  <cp_value cp_type="ucs">6771</cp_value>
  <cp_value cp_type="jis208">37-76</cp_value>
</codepoint>
<radical>
  <rad_value rad_type="classical">75</rad_value>
  <rad_value rad_type="nelson_c">4</rad_value>
</radical>
<misc>
  <grade>2</grade>
  <stroke_count>8</stroke_count>
  <freq>37</freq>
  <jlpt>4</jlpt>
</misc>
<dic_number>
  <dic_ref dr_type="nelson_c">213</dic_ref>
  <dic_ref dr_type="nelson_n">2596</dic_ref>
  <dic_ref dr_type="halpern_njecd">3568</dic_ref>
  <dic_ref dr_type="halpern_kkld">2221</dic_ref>
  <dic_ref dr_type="heisig">504</dic_ref>
  <dic_ref dr_type="gakken">11</dic_ref>
  <dic_ref dr_type="oneill_names">771</dic_ref>
  <dic_ref dr_type="oneill_kk">27</dic_ref>
  <dic_ref dr_type="moro" m_vol="6" m_page="0172">14499</dic_ref>
  <dic_ref dr_type="henshall">184</dic_ref>
  <dic_ref dr_type="sh_kk">71</dic_ref>
  <dic_ref dr_type="sakade">121</dic_ref>
  <dic_ref dr_type="jf_cards">63</dic_ref>
  <dic_ref dr_type="henshall3">201</dic_ref>
  <dic_ref dr_type="tutt_cards">164</dic_ref>
  <dic_ref dr_type="crowley">108</dic_ref>
  <dic_ref dr_type="kanji_in_context">39</dic_ref>
  <dic_ref dr_type="busy_people">2.10</dic_ref>
  <dic_ref dr_type="kodansha_compact">1053</dic_ref>
  <dic_ref dr_type="maniette">516</dic_ref>
</dic_number>
<query_code>
  <q_code qc_type="skip">4-8-3</q_code>
  <q_code qc_type="sh_desc">0a8.9</q_code>
  <q_code qc_type="four_corner">5090.6</q_code>
  <q_code qc_type="deroo">1564</q_code>
</query_code>
<reading_meaning>
  <rmgroup>
    <reading r_type="pinyin">dong1</reading>
    <reading r_type="korean_r">dong</reading>
    <reading r_type="korean_h">동</reading>
    <reading r_type="ja_on">トウ</reading>
    <reading r_type="ja_kun">ひがし</reading>
    <meaning>east</meaning>
    <meaning m_lang="fr">Est</meaning>
    <meaning m_lang="es">Este</meaning>
    <meaning m_lang="pt">Oriente</meaning>
  </rmgroup>
  <nanori>あい</nanori>
```

```
      <nanori>あがり</nanori>
      <nanori>あずま</nanori>
      <nanori>あづま</nanori>
      <nanori>こ</nanori>
      <nanori>さき</nanori>
      <nanori>しの</nanori>
      <nanori>とお</nanori>
      <nanori>はる</nanori>
      <nanori>ひが</nanori>
      <nanori>もと</nanori>
    </reading_meaning>
  </character>
```

7.3.2 Alternative Database Structures

The structure and organisation of the databases as it is now, seems optimal among several alternatives. The reason for that is that the data is stored in separate and self-containd units. The indexing refeencing between the different structures works seamless via the unicode codepoints. Another adavantage of using the available formats. KanjiDict2 is work in progress and updates may be available for other languages or for additional characters. Using the KanjiDict2 the way it is allows for reliablity. The same is true for for UPX and InkML: UNIPEN has been the de facto standard for handwriting data exchange for a long time and UPX is considered the official successor of UNIPEN and InkML is a W3C recommendation[2].

7.3.2.1 A Unified File Format

An alternative solution to the split information approach would be to establish a unified file format for the pen trajectories, the UPX metadata and the lexical information. The advantage of that approach is that all data would be in one single place. However, a number of disadvantages would emerge from that approach, too. The file formats, InkML, UPX and KanjiDict2 already exist. Creating a new file format based on those would on one hand not create much additional value, but on the other hand lead to not using available de facto standards. Every time, a new version of the lexical data is issued or every time one would want to update the gold standard pen trajectory of a character, the formats would have to be linked together.

7.3.2.2 Relational Database

Establishing a relational database to hold the different data structures sounds like a feasible plan. With consideration of the three formats InkML, UPX and KanjiDict2 it should be possible to design a relational database that holds the different information in a structured manner. However, again the same problem arises. Updates and extensions of the formtats of the data would create the need for a conversion tool. If the XML specification of one of the formats changes, the table structure in the relational database would have to be adapted, too. Alternatively, the actual XML code could be stored in relational database tables. That approach however seem awkward. XML is a format for structuring data in human readable text files. All three formats, InkML, UPX and KanjiDict2 have been designed to be human-editable and therefore accessible without any special software or tools but a text editor. Storing these XML structures in a relational database would compromise the advantage of a relational database that lies in querying and would also make editing the data a more complex task. Additionally, the application is a desktop application for a single user environment. Depending on what relational database was used that could force shipment of the database software along with the Kanji learning system. The conclusion is that once InkML and UPX have been standardised and are therefore more reliable formats it could be benefitial to design a relational database structure that mirrors these formats. Then, the relational database could be used in an online environment, where a server application performs the recognition for a web client. For the time being and the technical equipment given it does not seem useful to employ a relational database.

[2]However, until the end of 2009 InkML has not been standardised by the W3C and UPX is still in a planning stage concerning standardisation.

7.4 Recognition Architecture

7.5 Stroke Recognition Process

7.5.1 Advanced Point Lists

what happens to the points? nothing, really - the magic happens when normalisation and the other stuff starts. why this section? what's the purpose? oh, right - angles and vectors instead of simple points. from one point to the next, or rather from on point to ten points down the line, to get a rougher direction. vectors make it interesting. impacts on curve handling! gradient and stuff can be measured in the vector representation (even without any boxes) making the point list a cool mathematical object! show code samples in pseudocode if necessary. report about the cool stuff.

what's the similarity measure for points and strokes? show requirements. what alternatives were there to consider?

7.5.2 Normalisation

Many handwriting recognition systems perform some kind of normalisation. See section 3.4.2.3 for a review of common techniques. In this system, only size normalisation is performed. Other normalisation techniques reported in literature were often necessary, because the input devices returned unsufficient data. With modern input devices many of the normalisation techniques that were used to clean the data from different kinds of noise are not necessary.

In order not to lose any data, the complete list of points is maintained, there is no point reduction. Since the prototype is a learning application and not a pure handwriting recognition, the main issue seems the size of the character. The size normalisation is done in two steps. Each complex that is more complex than a point has a bounding box around that can be used for scaling.

7.5.2.1 Boxing

The term *bounding box* needs not much introduction. It is the smallest box surrounding a graphic object. In this prototype, bounding boxes are rectangles with the same height and width. Rectangles are defined as a 3-tuple of a handle point, a height and a width. $R :=<< x, y >, w, y >$ where the 2-tuple $< x, y >$ is a point in the coordinate system with the decimal values x and y as coordinates, h is the y-height of the rectangle and w is the x-width. In order to find an extended square bounding box of any sequence of points, a method finds the four outermost points C_i with the minimal and maximal values for x and y.

$$C_{x_{min}} :=< x_{min}, y_1 >$$
$$C_{x_{max}} :=< x_{max}, y_2 >$$
$$C_{y_{min}} :=< x_1, y_{min} >$$
$$C_{y_{max}} :=< x_2, y_{max} >$$

The intermediate handle point $H_{intermediate}$, the width w and height h of the minimal bounding box B' can be calculated as follows:

$$H_{intermediate} :=< x_{min}, y_{min} >$$
$$w := x_{max} - x_{min}$$
$$h := y_{max} - y_{min}$$
$$s := \arg\max(w, h)$$

Then, the handle point H can be calculated by:

$$H := \begin{cases} < x_{min} - \frac{s-w}{2}, y_{min} > & \text{if } s \text{ equals } h \\ < x_{min}, y_{min} - \frac{s-h}{2} > & \text{if } s \text{ equals } w \end{cases}$$

The minimal bounding box B' is a rectangle with the following values:

$$B' :=< H_{intermediate}, w, h >.$$

The extended, square bounding box B is a rectangle with the following values:

$B :=< H, s, s >$

The use of boxing is used multifunctional in the prototype. One of the uses is for distortion-free scaling (see section 7.5.2.2). Another use is for the comparison of positions of substructures of a character or Radical (see section 7.6).

7.5.2.2 Scaling

The bounding box of most Kanji is square, in cases where it is not, the space given to a smaller Kanji is still a square. In order to account for that, scaling is done with respect to the ideal of a square shape of a character. That concept is used for full characters, as well as Radicals and strokes. Before scaling the size of a stroke, a square bounding box is constructed. The handle point of the square bounding box is used as a vanishing point for vectorial projection. In a stroke data structure that consists of a point sequence $p_1, \ldots, p_n$ a number of vectors $\vec{v}_1, \ldots, \vec{v}_n$ is created. Each vector is defined by the handle point $H =< x_h, y_h >$ of a square bounding box and the p_i.

$\vec{v}_k := p_k - H = \binom{x_{p_k} - x_h}{y_{p_k} - y_h}$

In order to generate a scaled version of a sequence of points, all the $\vec{v}_k$ are multiplied with a scaling factor.

7.5.3 Recognition of Straight Strokes

7.5.3.1 Identification of a Straight Stroke

The notion of *stroke* essentially means *point sequence* in a technical sense. That is, strokes are not mathematical objects like functions. In order to perform stroke recognition, it is necessary to determine what shape a stroke has. For the identification of a straight stroke, a linear regression line is constructed from the point sequence with the Gaussian method of least squares. The construction is done with the standard method for calculating least squares, as described in literature. The method seeks for the minimal sum of squares of residuals (Marquardt 1963). If the sum of residual squares is below a certain threshhold that has been determined empirically, the system assumes that the point sequence generally follows a straight line. In that case, straight stroke matching will be performed. If the line appears to be curved, it will be matched with curve analysis methods as described in section 7.5.4.

7.5.3.2 Straight Stroke Matching

For the matching of a straight stroke there are a number of features that could be considered for a feature vector. In order to describe a straight stroke and then match it with another one, a feature vector is constructed that has the following features:

- **Length**: The total lenght of the line. Sum of the distances of succeeding points.
- **Initial point**: The coordinates of the initial point
- **Endpoint**: The coordinates of the endpoint
- Total number of measured points: The number of sample points that were measured by the input device
- **Velocity**: The velocity with which the stroke was drawn
- **Gradient/Direction**: The slope of the linear regression line that can be constructed for the point sequence, or the general direction of the line, represented as a numerical vector.

However, some of the information seem redundant. In order to match two strokes by their feature vectors F_{db} and F_{exp}, the following values are conidered:

$$F := \begin{pmatrix} \text{Length} & l \\ \text{Initial point} & p_I \\ \text{Endpoint} & p_E \\ \text{Velocity} & v \\ \text{Direction vector} & \vec{d} \\ \text{Sum of residual squares} & \sigma \end{pmatrix}$$

The features are used for matching with other straight strokes by comparison of their values. The length l is calculated as the sum of the euclidian distances d_i between the individual points p_i and p_{i-i} within a sequence of N points.

$$d_i := \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$
$$l := \sum_{i=1}^{N-1} d_i$$

In the case of the initial point and endpoint comparison of the two strokes, the euclidian distance between the original point $P_{I,db}, P_{E,db}$ and the candidate point $P_{I,exp}, P_{E,exp}$ is used as a comparison value. The velocity v is calculated from the time difference δ_T between sampling time of the initial point t_{p_I} and the endpoint t_{p_E}:

$$\delta_T := t_{p_E} - t_{p_I}$$

$$v := \frac{n_p}{\delta_T}$$

The velocities v_{db} and v_{exp} are both real numbers and can be compared directly. The direction vectors of the regression line $\vec{d}_{db}$ and $\vec{d}_{exp}$ are compared by the deviation of their directions. The deviation is expressed as the angle α between the original and the candidate vector.

$$\alpha := \vec{d}_{db} \quad \angle \quad \vec{d}_{exp}$$

The sum of residual squares σ is calculated from the measured point sequence with respect to the regression line of the database point sequence. Given the regression line f_{db}, the sum of the squares of the residuals for the input sequence is:

$$\sigma := \sum_{i=1}^{N} (f_{db}(x_{i,exp}) - x_{i,exp})^2$$

The smaller σ the closer the measured points to the regression line from the database. This is only done for strokes that have been calculated to be straight, since any set of points can have a regression line, even curved ones. In the case of curved strokes, a linear regression line as it is used here does not seem to bear the potential helping identify a point seqence to be a certain shape.

7.5.4 Curve Handling

7.5.4.1 Detecting Curves in Strokes

Different curve matching techniques are described in section 3.4.3.4. The technique used here refers to is a vector-based technique, that aims at the extraction of features to describe the curved stroke. In this prototype the direction feature plays a crucial role. It is generally follwing several approaches in on-line handwriting recognition literature. The direction feature as it is used in other applications has been discussed in section 3.4.3.3). The approach for curve recognition used here defines direction vectors.

The general concept is to define a direction vector between a point within the point sequence and a successor point. However, since the measured points are usually very close together, the direction vector is defined between a point and another point that is further away, defined by a dynamic threshold, considering distance, total number of points and the total length of the stroke. If non of these vectors deviates from the general direction of the stroke, a dynamic threshold, considering the total stroke length, the stroke is considered straight. Nevertheless, for matching straight strokes, a feature vector for straight strokes is used. If the slope of the curve changes, i.e. the direction of the vectors changes, a curve has been detected. There is a threshold T_{min} for the minimal size of a *recognisable angle* between two vectors. Any angle smaller than that will be ignored completely. A second threshold T_{max} describes the minimal size for an angle γ that is taken into account. Any angle β_i between T_{min} and T_{max} will be stored. When the sum of the β_i exceeds T_{max} the position will be treated as a curve.

$\alpha < T_{min} \Rightarrow \alpha$ will be ignored

$T_{min} < \beta_i < T_{max} \Rightarrow \beta_i$ will be stored

$T_{max} < \gamma \Rightarrow \gamma$ will be interpreted as an angle in the stroke

$T_{max} < \sum_{i=k}^{l} \beta_i \Rightarrow \sum_{i=k}^{l} \beta_i$ will be interpreted as an angle in the stroke

A successful curve detection extracts the key feature of the curve description. The remaining features can be extracted or calculated, by using the detected curves.

7.5.4.2 Feature Extraction for Curved Strokes

The feature vector for curved strokes contains the following elements:

$$F := \begin{pmatrix} \text{Length} & & l \\ \text{Initial point} & & p_I \\ \text{Endpoint} & & p_E \\ \text{Velocity} & & v \\ \text{Corner angles} & \gamma_i & i := 1, \ldots, n \\ \text{Corner points} & c_i & i := 1, \ldots, n \\ \text{Direction sequence} & \vec{d_j} & j := 1, \ldots, m \end{pmatrix}$$

The partial feature vector l, p_I, p_E, v is identical to the partial feature vector of the straight strokes with the same elements and needs no further description (see section 7.5.3.2). The corner angles have been calculated. In the case of a summation of angles the point with the greatest angle will be determined as the corner point. In the case of angles that have been observed directly without summation, the corner point is the point where the angular vectors meet. Therefore, the number of corner points m and corner angles is identical. The feature *Corner points* is defined as the sum of euclidian distances between the corner points from the stored datastructure with the candidate data structure.

$$c_p := \sum_{i=1}^{n} \sqrt{(x_{i,db} - x_{i,exp})^2 + (y_{i,db} - y_{i,exp})^2}$$

The corner angles feature is defined as the sum of the squares of the differences between the angles at the corner points.

$$c_a := \sum_{i=1}^{n} (\beta_{i,db} - \beta_{i,exp})^2$$

For both c_a and c_p the smaller the value, the closer the more similar the strokes. In fact, if c_a or c_p are equal to 0, the point analised point sequence was identical with the stored point sequence. The *direction sequence* feature takes into account all the direction changes, as opposed to the corner angles that use only the most significant ones. The direction sequence feature uses the angle of each line segment regarding the X-axis of the coordinate system. The angles are then matched against the time they were produced in.

7.5.5 Dynamic Time Warping

The technique of *dynamic time warping* (DTW) has been used for stroke matching by several studies and systems (see sections **??** and 3.5.4.2). It is an elastic matching technique that seeks an optimal path through a matrix of point distances. Details of the algorithm have been described in literature.

7.5.5.1 Standard DTW

Following Vuori et al. (2001) and Niels and Vuurpijl (2005), the DTW technique is implemented for two pen trajectories, namely the trajectory stored in the database $P_{db} = (p_{1,db} \ldots, p_{n,db})$ and the input trajectory generated by the user's drawing $P_{exp} = (q_{1,exp}, \ldots, q_{m,exp})$. Any point pair $p_{i,db}$ and $p_{j,exp}$ matches if one of two constraints is met:

1. **Boundary condition**: The points $p_{i,db}$ and $p_{j,exp}$ match, if one of the following boundary conditions is fulfilled:
   - $p_{i,db}$ equals the initial point $p_{I,db}$ and $p_{j,exp}$ equals the initial point $p_{I,exp}$ of their respective point sequences P_{db} or P_{exp}

- $p_{i,db}$ equals the endpoint $p_{E,db}$ and $p_{j,exp}$ equals the endpoint $p_{E,exp}$ of their respective point sequences P_{db} or P_{exp}

2. **Continuity condition**: The points $p_{i,db}$ and $p_{j,exp}$ match, if the follwoing equation is fulfilled

$$\frac{M}{N}i - cM \leq j \leq \frac{M}{N}i + cM$$

c is a constant between 0 and 1, indicating the strictness of the condition.

7.5.5.2 3-Dimensional DTW

In addition to the standard DTW algorithm, a second DTW classifier performs a 3-dimensional distance calculation. That is, a point sequence $P = (p_1 \ldots, p_n)$ is considered 3-dimensional in the way that it contains the relative time (in ms) from the beginning of the stroke as a third dimension. The point distance calculations are euclidian, ignoring the fact that the third dimension is a different type of coordinate.

$$d_i := \sqrt{(x_{i,db} - x_{i,exp})^2 + (y_{i,db} - y_{i,exp})^2 + (z_{i,db} - z_{i,exp})^2}$$

$< x_i, y_i >$ are the regular 2-dimensional point cooredinates, while the z_i are the relative times from the beginning of the stroke. The actual DTW algorithm works exactly the same, the difference lies in the distance calculation. This version of the algorithm has been conducted as an experiment. The direct comparision with the regular DTW is reported in section 8.2.2.

7.5.6 Stroke Matching Summary

Four different classifiers have been implemented. The straight stroke matching via regression lines, the curve matching classifier with direction sequences and two variants of the dynamic time warping algorithm, a regular DTW algorithm and the three-dimensional variant DTW. The stroke matching occurs in NUMBER parts. Firstly, the straight stroke recogniser attempts to establish if a stroke is straight or not. Then the straight stroke matcher analyses if the input stroke and the database stroke are equal. Secondly, if the stroke is regarded as being curved, the curve detector performs a search for the significant points and angles in both sequences. After that, the curved stroke matcher compares the features of the curved strokes. If both recognisers fail to recognise the expected stroke, dynamic time warping is used as a fallback option.

7.6 Radical Recognition Process

7.6.1 Radical Recognition Features

The recognition of a Radical is a search process. The data format of Radicals is described in section 7.2.3.3. Essentially, a Radical is an ordered collection of strokes. In order to recognise a radical, the sequence of strokes from the database is compared to an input sequence. The measure of similarity is a feature vector that consists of the number of strokes and the similarities between them. Furthermore, permutations of the stroke sequence will be penalised. In order to match two radicals by their feature vectors F_{db} and F_{exp}, the following values are conidered:

$$F := \begin{pmatrix} \text{Total number of strokes} & n_s \\ \text{Position of bounding box} & P \\ \text{Size of bounding box} & S \\ \text{Stroke matching values} & m_{i,j} \\ \text{Stroke sequence} & < s_1, \ldots, s_n > \end{pmatrix}$$

Some of the features need little explanation. The total number of strokes n_s is an integer. In the matching process - if the $n_{s,db}$ and $n_{s,exp}$ differ, the confidence value of the radical recognition value will be lowered. The position of the bounding box $P$is determined by the handle point of the box. The euclidian distance between the P_{db} and P_{exp} serves as a distance measure. The size of the bounding box S can support or refute assumptions about the general shape and size of a radical.

7.6.2 Exploitation of the Stroke Recognition Process

Stroke matching forms the core of the Radical recognition process. Each input stroke is matched against all the strokes of the Radical. A matrix of matching values will be generated by comparing each input stroke to each database stroke for a Radical. The confidence values in the stroke matching matrix M are used to analyse if the input stroke sequence signifies the currently analysed Radical.

$$M_{m,n} = \begin{pmatrix} \mu_{1,1} & \mu_{1,2} & \cdots & \mu_{1,n} \\ \mu_{2,1} & \mu_{2,2} & \cdots & \mu_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{m,1} & \mu_{m,2} & \cdots & \mu_{m,n} \end{pmatrix}$$

The matrix M forms the basis of the Radical recognition. The maximal confidence value for a Radical will be achieved if

- $m = n$
- the maximal matching value $\mu_{i,j}$ for each stroke is above a threshold in order to conclude that it matched with its corresponding stored stroke from the database.
- the maximal matching values $\mu_{i,j}$ for all strokes form a diagonal path through the matrix, with $i = j$ for each maximal matching value, starting at $\mu_{1,1}$, ending at $\mu_{n,n}$ (and $m = n$).

If two identical stroke sequences are analised, the ideal is fulfilled. Any deviance from these constraints will be penalised and leads to a lower confidence value for the Radical match. The matrix is the source for both the features of the stroke matching values $m_{i,j}$) and the stroke sequence $< s_1, \ldots, s_n >$. A deviance from the diagonal path means deviance from the ideal stroke sequence.

7.7 Character Recognition Process

The character recognition process relies on the Radical recognition process. Each character is comprised of a set of Radicals the position of their bounding boxes. In order to determine if a stroke sequence represents a certain character, firstly the individual Radicals are recognised. For character matching, the recognised Radicals are compared with the Radicals of the character. If they match with a confidence values above the threshold and their position within the character is correct and the character is recognised correctly. Generally, the confidence value of a character depends on the confidence values of the individual radicals and the distance and position of the bounding boxes to the database entry. A feature vector F here is defined as:

$$F := \begin{pmatrix} \text{Number of Radicals} & & n_r \\ \text{Position of Radical bounding boxes} & P_i & i = 1, \ldots, n_r \\ \text{Radical confidence values} & c_i & i = 1, \ldots, n_r \end{pmatrix}$$

The $n_{r,db}$ and $n_{r,exp}$ are compared numerically, as they are real numbers. The positions of the Radical bounding boxes $P_{i,db}$ and $P_{i,exp}$ are compared by their euclidian distance. The Radical confidence values c_i are averaged. The confidence value C for a character is calculated in several steps. The confidence value C_n for the number of Radicals is computed as follows:

$$C_n := \begin{cases} \frac{n_{r,db}}{n_{r,exp}} & \text{if } n_{r,db} \leq n_{r,exp} \\ \frac{n_{r,exp}}{n_{r,db}} & \text{if } n_{r,db} > n_{r,exp} \end{cases}$$

The confidence value C_p for the Radical positions is extracted from a matrix of euclidian distances between the Radical handles $P_{i,db} =< x_{i,db}, y_{i,db} >$ and $P_{i,exp} =< x_{i,exp}, y_{i,exp} >$. All the distance values $\mu_{i,j}$ are computed first to generate a Matrix of Radical position distances $M_{m,n}$, where $n = n_{r,db}$ and $m = n_{r,exp}$.

$$\mu_{i,j} := \sqrt{(x_{i,db} - x_{j,exp})^2 + (y_{i,db} - y_{j,exp})^2}$$

$$M_{m,n} = \begin{pmatrix} \mu_{1,1} & \mu_{1,2} & \cdots & \mu_{1,n} \\ \mu_{2,1} & \mu_{2,2} & \cdots & \mu_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{m,1} & \mu_{m,2} & \cdots & \mu_{m,n} \end{pmatrix}$$

The minimal value ξ_m of each line or column is chosen. f_p is a constant factor for scaling the confidence value.

$$C_p := \begin{cases} f_p * \sum_{i=1}^{n_{r,db}} \xi_i & \text{if } n \geq m \\ f_p * \sum_{i=1}^{n_{r,exp}} \xi_i & \text{if } n < m \end{cases}$$

The confidence value C is the averaged sum of the other confidence values. The sum consits of the confidence values for the number of characters C_n, for the position of the Radical bounding boxes C_p and the averaged sum of the Radical confidence values. All partial confidence values are weighted with the w_i constants.

$$C := \tfrac{1}{3}(w_n * C_n + w_p * C_p + \tfrac{w_c}{n_{r,exp}} \sum_{i=1}^{n_{r,exp}} c_i)$$

7.8 Error Handling

see section 5.6.1 in chapter 5 for possible sources of error

7.8.1 Error Recognition

why this section? to demonstrate own achievements of error recognition. the reader should know how it is done technically.

what goes into this section? the aspects of finding errors. finding errors is not a straightforward trivial task - whenever something does not match it is an error - doesn't work like that. instead, firstly, it needs to be made sure that it actually is an error. meaning - not a recognition error, but a user error. secondly, the type of error needs be identified. see section 5.6.1 (or handwritten page 58) for sources of error.

how will this section be written? technical - first describe how the error recognition integrates into the recognition process, then how errors are identified.

7.8.2 Error Processing

why this section? actually the 'handling' or 'processing' aspect could be described in the recognition section 7.8.1 as well. so this section is only for a better overview, for document structure, thematically they are the same section. thus they are put together under Error Handling 7.8.

what goes into this section?

Chapter 8

Implementation and Evaluation

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

8.1 Implementation Details

Pointer auf CD und auf Appendix mit Beispielinteraktionen (diese mit Foto). Screenshots. Zahlen zur Erkennung - z.B. wie lange dauert es, ein zeichen zu erkennen?

wie wurden einzelne dinge realisiert, z.b. vectorielle funktionen? was war neu? klassen wie box / bounding box, technisch, alles was in HWREngine nicht behandelt wurde.

abschnitt ueber optimierung. optimierungszyklus inklusive ausprobieren beschreiben. s. 51 rueckseite

s 49 rueckseite: interface-optimierung entscheidungen herausstellen.

s 27,28 vectorschnitt

s.11 iPhone - port of input app. checked out objective C and stuff!

see section 7.1.1.1. describe what was difficult concerning the lists and bloody point objects. performance issues! optimisation with try and error!

ISF - see section 7.2.2.1

dead end of data format description: how I first developed my own format and then found that UPX was better.

in 7.3 there is an undiscussed point: 3. Description of the production of the lexicon. it was not just taken from j.b. but it was intervowen?! (verflochten) with the trajectories. where did I get these from? how many chars are in the two dictionaries

8.2 Evaluation of the HWR

8.2.1 Evaluation Metrics

evaluation method: counting precision and recall section about precision and recall - the odd numbers. how can that be done honest and useful? how can I get meaningful numbers at all?

s. 12 beachten: WICHTIG: eval kurz und qualitativ.

8.2.2 DTW vs. 3-D DTW

This section al

8.3 Evaluation of E-Learning Application with Integrated HWR

qualitative auswertung, keine zahlen, sondern fragebogen. fuehlt der lerner sich unterstuetzt? glaubt er, dass es schneller geht als ohne HWR? besser als auf papier?

8.4 Evaluation of the Error Hints

use cases, inwieweit helfen die fehlerhinweise? geht das lernen dann wirklich schneller? wie laesst sich der mehrwert bewerten? system kann sagen: wo liegt die verwechslung? warum war das falsch?

Chapter 9

Conclusions

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

siehe s. 47 oben. ich habe erschaffen. arbeitshypothese - lernen - kann kaum evaluiert werden.

siehe introduction: was ist neu, was ist geil? vergleiche mit motivation? inwieweit ist motivation erfuellt? warum haben wir dass gemacht? was ist das ergebnis?

Chapter 10

Outlook

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

What can we do for other languages, using similar character sets? This HWR as a service for other applications?

What can be done with future devices? With a multi-touch device? With a pressure-sensitive device?

See source of section 3.4.1 for this paragraph here: xxx: The reason for not using pressure intensity as a recognition feature might be that it is plausible to assume that pressure intensity is even more individual to the writer than the shape of the drawn characters. Therefore, featuring pressure intensity in a HWR application could add more noise to the data. However, the differences in pressure intensity between different characters drawn by the same writer should be analysed. [xxx put the speculation about pressure intensity into outlook chapter? something along the lines of progress in device technology, subsection pressure intensity of tablets? Or maybe refer from here to outlook chapter?] :xxx

ISF - see section 7.2.2.1 and section in implementation chapter.

Appendix A

Japanese Language

A.1 Kanji timeline

14th-11th centuries B.C. or ca. 1200-1050 B.C. First Chinese characters on Oracle bones. (Guo et al. 2000) 206 B.C. - 220 A.D. Han dynasty, development of the *Hànzì* 300-400 A.D. The Hànzì characters were brought to Japan by Koreans. 712 A.D. Kojiki was written

A.2 Kana かな

A.2.1 Hiragana ひらがな

Hiragana is a syllabic script. For a description of use and structure of Hiragana see section 2.2.1.1. A full table of the Hiragana script can be found in table (A.1). Table (A.1) is adapted from (Hadamitzky 1995).

	a	i	u	e	o	n
∅	あ	い	う	え	お	ん
k	か	き	く	け	こ	
s	さ	し	す	せ	そ	
t	た	ち	つ	て	と	
n	な	に	ぬ	ね	の	
h	は	ひ	ふ	へ	ほ	
m	ま	み	む	ね	も	
y	や		ゆ		よ	
r	ら	り	る	れ	ろ	
w	わ				を	

Table A.1: The full set of Hiragana characters

A.2.2 Katakana カタカナ

Katakana is the second syllabic script in use in the Japanese language. For a more detailed description of what it is used for, see section 2.2.1.2. A full table of the Katakana script is depicted in table (A.2). Table (A.2) is adapted from (Hadamitzky 1995).

Ø	k	s	t	n	h	y	r	w	
a	アカサタナハヤラワ								
i	イキシチニヒ　リ								
u	ウクスツヌフユル								
e	エケセテネヘ　レ								
o	オコソトノホヨロヲ								
n	ン								

Table A.2: The full set of Katakana characters

Appendix B

Technicalities

- Why this section? The purpose of this section is It would be off purpose, if - What goes into this section? The main content of this section is * if describing a problem: why is the problem relevant. * if describing a solution to a problem: what alternatives were there to solve it, why was this solution chosen? what made it the best choice? was it the optimal solution? - How will this section be structured and organised? The organisational structure of the section - In what style will it be written? The style of writing will be - Next action - what to write first? The next part to write is

xxx: screenshots and stuff go here. s. 15 rueckseite

List of Figures

2.1 Reduction from Kanji to Hiragana . . . 12
2.2 Reduction from Kanji to Katakana . . . 12
2.3 Kanji pictograms . . . 14
2.4 Radical positions in Kanji characters . . . 16
2.5 Cedar leaves . . . 16

3.1 Expanding cursive letters . . . 22
3.2 Different handwriting styles . . . 23
3.3 Overview of an OLCCR system . . . 30

6.1 The Model-View-Controller paradigm . . . 41
6.2 The Model-View-Controller paradigm AGAIN! . . . 42
6.3 The global architecture of the software system . . . 42
6.4 The data flow within the software system . . . 43

7.1 Stroke order of the Kanji 東 . . . 58

Listings

7.1 Demonstration of the *trace* tag . . . 51
7.2 Demonstration of the *time channel* . . . 51
7.3 Demonstration of the *UNIPEN* format . . . 52
7.4 Demonstration of the *hLevel* tag in UPX . . . 54
7.5 Definition of the trace format . . . 55
7.6 A sample trace . . . 55
7.7 An example of how a stroke could be represented in UPX . . . 56
7.8 A Radical representation in UPX . . . 56
7.9 A character representation in UPX . . . 57
7.10 Sample lexicon entry for lexical data . . . 58

List of Tables

2.1 Kanji ideograms 13
2.2 Kanji phonograms 14

7.1 Sample trace interpreation 56

A.1 The full set of Hiragana characters 75
A.2 The full set of Katakana characters 76

References

Agrawal, M., K. Bali, S. Madhvanath, and L. Vuurpijl (2005). UPX: A new XML Representation for Annotated Datasets of Online Handwriting Data. In *ICDAR*, pp. 1161--1165.

Agrawal, S., I. Constandache, S. Gaonkar, and R. R. Choudhury (2009). Phonepoint Pen: Using Mobile Phones to Write In Air. In *MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, New York, NY, USA, pp. 1--6. ACM.

Bahlmann, C. and H. Burkhardt (2004, March). The Writer Independent Online Handwriting Recognition System *frog on hand* and Cluster Generative Statistical Dynamic Time Warping. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*(3), 299--310.

Bailey, S. and D. Meurers (2008). Diagnosing meaning errors in short answers to reading comprehension questions. In *EANL '08: Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, Morristown, NJ, USA, pp. 107--115. Association for Computational Linguistics.

Bashir, M. and J. Kempf (2008, Fall). Reduced Dynamic Time Warping for Handwriting Recognition Based on Multidimensional Time Series of a Novel Pen Device. *International Journal of Intelligent Systems and Technologies, WASET 3*(4), 194.

Bharath, A. and S. Madhvanath (2009, September). Online Handwriting Recognition for Indic Scripts. In V. Govindaraju and S. R. Setlur (Eds.), *Guide to OCR for Indic Scripts*, Advances in Pattern Recognition, pp. 209--234. London: Springer.

Bhaskarabhatla, A. S. and S. Madhvanath (2004). An XML Representation for Annotated Handwriting Datasets for Online Handwriting Recognition. In *Proc. 4th Int'l Conf. on Language Resources and Evaluation (LREC '04*, Lisbon, Portugal.

Blostein, D. and L. Haken (1999). Using Diagram Generation Software to Improve Diagram Recognition: A Case Study of Music Notation. *IEEE Trans. Pattern Anal. Mach. Intell. 21*(11), 1121--1136.

Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard (2004, Feb). *Web Services Architecture*. W3C Consortium. Online. Retrieved from `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/]` on 2010-01-25.

Breen, J. (2004, August). Multiple Indexing in an Electronic Kanji Dictionary. In M. Zock and P. S. Dizier (Eds.), *post COLING Workshop on Enhancing and Using Electronic Dictionaries*, Geneva, Switzerland. COLING: International Committee on Computational Linguistics.

Chamberlain, B. H. (1982). *The Kojiki: Records of Ancient Matters* (2nd ed.). Boston, USA: Tuttle Publishing.

Chan, K.-F. and D.-Y. Yeung (2001). Error Detection, Error Correction and Performance Evaluation in On-Line Mathematical Expression Recognition. In *Pattern Recognition*, Volume 34, pp. 1671--1684. Elsevier Science Inc.

Chee, Y.-M., K. Franke, M. Froumentin, S. Madhvanath, J.-A. Magaña, G. Russell, G. Seni, C. Tremblay, S. M. Watt, and L. Yaeger (2006, Oct). *Ink Markup Language (InkML)*. W3C Consortium. Online. Retrieved from `http://www.w3.org/TR/2006/WD-InkML-20061023/` on 2010-02-01.

Chen, J.-W. and S.-Y. Lee (1996). A Hierarchical Representation for the Reference Database of On-Line Chinese Character Recognition. In *Advances in Structural and Syntactical Pattern Recognition*, Volume 1121 of *Lecture Notes in Computer Science*, pp. 351--360. Berlin/Heidelberg, Germany: Springer.

Chu, S., E. Keogh, D. Hart, and M. J. Pazzani (2002). Iterative Deepening Dynamic Time Warping for Time Series. In *Proceedings of SDM '02*. Second SIAM International Conference on Data Mining (SDM-02).

Encyclopædia Britannica Inc. (2009). Devanāgarī. In *Encyclopædia Britannica*. Retrieved November 30, 2009 from `http://www.britannica.com/EBchecked/topic/159937/Devanagari`.

Foljanty, D. (1984). Die japanische Schrift (in german). In Institut für deutsche Sprache Mannheim, T. Kaneko, and G. Stickel (Eds.), *Japanische Schrift, Lautstrukturen, Wortbildung*, Volume 1 of *Deutsch und Japanisch im Kontrast*, Chapter 2, pp. 29--63. Heidelberg: Julius Groos Verlag.

Goldberg, H. E. (1915, December). Controller. *United States Patent 1,116,663 0*(0), 0.

Grassmuck, V. (1997). Die japanische Schrift und ihre Digitalisierung (in german). In W. Nöth and K. Wenz (Eds.), *Reden über Medien*, Volume II of *Reihe Intervalle. Schriften des WZ*. Kassel: Hochschulverlag.

Gudgin, M., M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon (2007, Apr). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C Consortium. Online. Retrieved from `http://www.w3.org/TR/2007/REC-soap12-part1-20070427/` on 2010-01-25.

Guo, Z., K. Liua, X. Lua, H. Maa, K. Lia, S. Yuanb, and X. Wub (2000, October). The Use of AMS Radiocarbon Dating for Xia–Shang–Zhou Chronology. In *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, Volume 172, pp. 724--731. 8th International Conference on Accelerator Mass Spectrometry: Elsevier B.V.

Guyon, I., L. Schomaker, R. Plamondon, M. Liberman, and S. Janet (1994). UNIPEN project of on-line data exchange and recognition benchmarks. In *Proc. of the 12th International Conference on Pattern Recognition*, Jerusalem, pp. 29--33.

Hadamitzky, W. (1995). *Kanji und Kana*, Volume 1 of *Kanji und Kana*. Berlin: Langenscheidt KG.

Haschke, B. and G. Thomas (2008). *Kleines Lexikon Deutscher Wörter Japanischer Herkunft - von Aikido bis Zen (in German)* (1st ed.). Munich: Beck.

Heisig, J. W. and R. Rauther (2007). *Die Kanji lernen und behalten (in German)* (2 ed.), Volume 1 of *Die Kanji lernen und behalten*. Göttingen: Klostermann.

Hu, J., M. K. Brown, and W. Turin (1996). HMM Based On-Line Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*(10), 1039--1045.

Hu, J., G. Lim, and M. K. Brown (2000). Writer Independent On-Line Handwriting Recognition Using an HMM Approach. *Pattern Recognition 33*(1), 133--146.

Japanese Industrial Standards Committee (1997). *JIS X 0208-1997 7-bit and 8-bit Coded Kanji Sets for Information Interchange*. Japanese Standards Association.

Jäger, S., S. Manke, J. Reichert, and A. Waibel (2001). On-Line Handwriting Recognition: The Npen ++ Recognizer. *International Journal on Document Analysis and Recognition 3*(3), 169--181.

Jäger, S. and M. Nakagawa (2001). Two On-Line Japanese Character Databases in Unipen Format. *International Conference on Document Analysis and Recognition 0*, 0566.

Joshi, N., G. Sita, A. G. Ramakrishnan, V. Deepu, and S. Madhvanath (2005). Machine Recognition of Online Handwritten Devanagari Characters. *International Conference on Document Analysis and Recognition 0*, 1156--1160.

Joshi, N., G. Sita, A. G. Ramakrishnan, and S. Madhvanath (2004a, October). Comparison of Elastic Matching Algorithms for Online Tamil Handwritten Character Recognition. *International Workshop on Frontiers in Handwriting Recognition*.

Joshi, N., G. Sita, A. G. Ramakrishnan, and S. Madhvanath (2004b, October). Tamil Handwriting Recognition Using Subspace and DTW Based Classifiers. In *Neural Information Processing*, Volume 3316 of *Lecture Notes in Computer Science*, pp. 806--813. Berlin/Heidelberg, Germany: Springer.

Kano, C., Y. Shimizu, H. Takenaka, and E. Ishii (1990, January). *Basic Kanji Book*, Volume 1. Bonjinsha.

Katsuki-Pestemer, N. (2006). *Grundstudium Japanisch 2 (in German)* (2nd ed.), Volume 2 of *Grundstudium Japanisch*. Troisdorf: Bildungsverlag EINS.

Kennedy, A. and D. Syme (2001). Design and implementation of generics for the .net common language runtime. In *PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, New York, NY, USA, pp. 1--12. ACM.

Keogh, E. and M. J. Pazzani (2002). Derivative Dynamic Time Warping. In *Proceedings of SDM '01*, Chicago, USA. First SIAM International Conference on Data Mining (SDM'2001).

Keshari, B. (2008). Techniques for Transformation and Exchange of Standartized Digital Ink. Master's thesis, Western Ontario University, London, Ontario, Canada. Manuscript committee: Dr. Stephen M. Watt (chief advisor), Dr. David J. Jeffrey, Dr. Mahmoud El-Sakka, Dr. Eric Schost. Online. Retrieved from `http://www.csd.uwo.ca/~watt/home/students/theses/BKeshari2008-msc.pdf` on 2010-02-02.

Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst (2007). Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL '07: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, Morristown, NJ, USA, pp. 177--180. Association for Computational Linguistics.

Krasner, G. E. and S. T. Pope (1988). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program. 1*(3), 26--49.

Lange, R. (1922). *Einführung in die Japanische Schrift (in German)* (2nd ed.). Number 15 in Lehrbücher des Seminars für orientalische Sprachen. Berlin: Walter de Gruyter.

Liu, C.-L., S. Jaeger, and M. Nakagawa (2004). Online Recognition of Chinese Characters: The State-of-the-Art. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 198--213.

Liu, C.-L. and M. Nakagawa (2000). Precise Candidate Selection for Large Character Set Recognition by Confidence Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 636--642.

Madhvanath, S., L. Vuurpijl, K. Bali, M. Agrawal, V. Jagannadan, D. Vijayasenan, D. Willems, and L. Schomaker (2006, Oct). <UPX> Schema Version 0.9.5. Technical report, HP Labs India, Bangalore, India. Online. Retrieved from `http://unipen.nici.ru.nl/upx/docs/UPXSchemaVersion0.9.5.doc` on 2010-02-02.

Marquardt, D. W. (1963, Jun). An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics 11*(2), 431--441.

Microsoft Corporation (2007). *Ink Serialized Format Specification*. Microsoft Corporation. Online. Retrieved from `http://download.microsoft.com/download/0/B/E/0BE8BDD7-E5E8-422A-ABFD-4342ED7AD886/InkSerializedFormat(ISF)Specification.pdf/` on 2010-02-02.

Nakagawa, M., J. Tokuno, B. Zhu, M. Onuma, H. Oda, and A. Kitadai (2008). Recent Results of Online Japanese Handwriting Recognition and Its Applications. In D. Doermann and S. Jaeger (Eds.), *Arabic and Chinese Handwriting Recognition*, Volume 4768 of *Lecture Notes in Computer Science*, pp. 170--195. Berlin/Heidelberg, Germany: Springer.

Nakai, M., N. Akira, H. Shimodaira, and S. Sagayama (2001). Substroke Approach to HMM-Based On-Line Kanji Handwriting Recognition. *International Conference on Document Analysis and Recognition 0*, 0491.

Nakai, M., H. Shimodaira, and S. Sagayama (2003). Generation of Hierarchical Dictionary for Stroke-Order Free Kanji Handwriting Recognition Based on Substroke HMM. In *Proceecings of Seventh Int'l Conf. Document Analysis and Recognition*, pp. 514--518.

Negussie, D. (2008, August). Writer Independent Online Handwriting Recognition for Ethiopic Characters. Master's thesis, Addis Ababa University, Addis Ababa, Ethiopia. Advisor: Dr. Solomon Atnafu.

Niels, R. (2004). Dynamic Time Warping: An Intuitive Way of Handwriting Recognition? Master's thesis, Radboud University, Nijmegen, The Netherlands. Manuscript committee: Dr. L.G. Vuurpijl (supervisor), Prof. Dr. C.M. Jonker and Dr. P.A. Kamsteeg.

Niels, R. and L. Vuurpijl (2005, August 29-September 1). Dynamic Time Warping Applied to Tamil Character Recognition. In *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR2005)*, Seoul, Korea, pp. 730--734.

Nishida, H. (1995). An Approach to Integration of Off-Line and On-Line Recognition of Handwriting. *Pattern Recogn. Lett. 16*(11), 1213--1219.

Noguchi, M. S. (2009, October 21). Get set for next year's overhaul of official kanji. *The Japan Times Online*. Online. Retrieved 2010-01-14 from `http://search.japantimes.co.jp/cgi-bin/ek20091021mn.html`.

Okumura, D., S. Uchida, and H. Sakoe (2005, August). An HMM Implementation for On-Line Handwriting Recognition Based on Pen-Coordinate Feature and Pen-Direction Feature. In *Proceedings of International Conference on Document Analysis and Recognition*, Volume 1, Korea, pp. 26--30.

Plamondon, R. and S. N. Srihari (2000). On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*(1), 63--84.

Rowley, M. (1992, August). *Kanji Pict-O-Graphix: Over 1,000 Japanese Kanji and Kana Mnemonics*. Berkeley CA, USA: Stone Bridge Press.

Santosh, K. and C. Nattee (2009). A Comprehensive Survey on On-Line Handwriting Recognition Technology and its Real Application to the Nepalese Natural Handwriting. *Kathmandu University Journal of Sience, Engineering and Technology 6*(I), 30--54.

Shimodaira, H., T. Sudo, M. Nakai, and S. Sagayama (2003). On-line Overlaid-Handwriting Recognition Based on Substroke HMMs. *Document Analysis and Recognition, International Conference on 2*, 1043.

Smith, J. (2007). *Inside Microsoft Windows Communication Foundation*. Redmond, USA: Microsoft Press.

Stahlmann, R. (2004). Didaktische, inhaltliche und funktionelle Optimierung einer selbst entwickelten Chinesischlernsoftware (in german). Master's thesis, Offenburg University of Applied Sciences, Offenburg, Germany. Manuscript committee: Prof. Dr. Roland Riempp (supervisor), Prof. Dr. Thomas Breyer-MaylÃ¤nder.

Suen, C. Y., S. Mori, S. H. Kim, and C. H. Leung (2003). Analysis and recognition of asian scripts - the state of the art. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, Washington, DC, USA, pp. 866. IEEE Computer Society.

Tappert, C. C., C. Y. Suen, and T. Wakahara (1990). The State of the Art in Online Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*(8), 787--808.

Tokuno, J., N. Inami, S. Matsuda, M. Nakai, H. Shimodaira, and S. Sagayama (2002). Context-Dependent Substroke Model for HMM-Based On-Line Handwriting Recognition. *International Workshop on Frontiers in Handwriting Recognition 0*, 78.

Tsujimura, N. (2007). *An Introduction to Japanese Linguistics* (2nd ed.). Blackwell Textbooks in Linguistics. Oxford: Blackwell Publishing.

Unicode Consortium (2000). *The Unicode Standard. Version 3.0*. Addison-Wesley.

Unipen Consortium (2006, Oct). *UPX -- The best from UNIPEN and inkML*. Nijmegen, Netherlands: International Unipen Foundation. Online. Retrieved from `http://unipen.nici.ru.nl/upx/index.html` on 2009-11-08.

Unipen Foundation, International (1994-2010). Unipen Specification. Online. Retrieved from `http://www.unipen.org/]` on 2009-10-25.

Velek, O., S. Jaeger, and M. Nakagawa (2002). A New Warping Technique for Normalizing Likelihood of Multiple Classifiers and Its Effectiveness in Combined On-line/Off-line Japanese Character Recognition. *International Workshop on Frontiers in Handwriting Recognition 0*, 177.

Vuori, V., J. Laaksonen, E. Oja, and J. Kangas (2001, September). Speeding up On-line Recognition of Handwritten Characters by Pruning the Prototype Set. In *Proceedings of the 6th International Conference on Document Analysis and Recognition (ICDAR'01)*, Seattle, USA, pp. 501--505. ICDAR.

Woods, V., S. Hastings, P. Buckle, and R. Haslam (2002). *Ergonomics of using a mouse or other non-keyboard input device*, Chapter 3, pp. 23. Number 045 in HSE research report. London: Health and Safety Executive.

Wydell, T. N. (1998). What Matters in Kanji Word Naming: Consistency, Regularity, or On/Kun-Reading Difference? In K. Tamaoka and C. K. Leong (Eds.), *Cognitive Processing of the Chinese and the Japanese Languages*, Volume 14 of *Neuropsychology and Cognition*, pp. 359--373. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Yuan, W., J. Gao, and H. Suzuki (2005, September). An Empirical Study on Language Model Adaptation Using a Metric of Domain Similarity. In R. Dale, K.-F. Wong, J. Su, and O. Y. Kwong (Eds.), *Natural Language Processing – IJCNLP 2005*, Lecture Notes in Artificial Intelligence, pp. 957--968. Heidelberg: Springer.

Zanibbi, R., D. Blostein, and J. R. Cordy (2005). Recognition Tasks are Imitation Games. In S. Singh, M. Singh, C. Apte, and P. Perner (Eds.), *Pattern Recognition and Data Mining*, Volume 3686 of *Lecture Notes in Computer Science*, pp. 209--218. Berlin/Heidelberg, Germany: Springer.

Zimmer, G. (2009). Bildung mit E-Learning. In B. Mikuszeit and U. Szudra (Eds.), *Multimedia und ethische Bildung*, Chapter 1.3, pp. 61--92. Frankfurt am Main, Germany: Peter Lang.