End-User Programming by Linguistic Expression employing Interaction and Paraphrasing

Nozomu Kaneko *1and Takehisa Onisawa *1

*1 Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1, Tennodai, Tsukuba, Ibaraki 305-8573, Japan
E-mail: nozom@fhuman.esys.tsukuba.ac.jp

An end-user programming system is presented in this paper, which applies the interaction and the paraphrase to computer programming with non-programming language. Users can build computer programs through inputting the procedures of problem-solving expressed with non-programming language into the presented system. The interaction between users and the system can make up for the understanding gaps between users and computer, which are caused by ambiguity of non-programming language. Paraphrase is used in order to change an undefined expression into a defined one, and makes a computer understand users' intention. Case-based reasoning is used for paraphrase, so that users can add own paraphrase knowledge into the system. Experiments verify the validity of the presented system.

1. Introduction

In recent years, personal computers have come to wide use, and computer users are not necessarily experts in a computer. On the other hand, a programming language is not so easy to understand as a natural language because the logical correctness is considered preferentially in a programming language and the conventional programming language is based on the computer-centered view. That is, although computer users should master a programming language itself, a computer does not need to understand human language. Therefore, the conventional software has been usually developed by experts in the special knowledge on computer programming. Users without programming knowledge cannot help using ready-made software to enjoy a computer.

In order to deal with such a problem, many studies on the end-user programming have been progressed. One of them is famous as the PBD (Programming By Demonstration) technique [1], which generates a computer program automatically by user's illustration. However, there are some open problems even in the PBD technique. For example, illustrations show neither end conditions nor conditional branches. Furthermore, a user cannot correct system's misunderstanding of user's intention by illustrations.

A natural language is the easiest language for human to understand and many intelligent systems using a natural language have been developed from the early days of artificial intelligence [2]. For example, one of the well-known early systems is SHRDLU [3], which interacts with human using a natural language about the world of blocks. Linguistic expressions are often used for computer programming based on the formal specification in the automatic programming field which aims at realizing the improvement of reliability and productivity of software. Recently, the concept of "everyday language computing" is proposed [5], in which information processing is performed with everyday language approach rather than with the conventional one based on numerical value. As for the end-user programming system with linguistic expressions, some systems are presented. For example, in [6], the syntax tree of Java language is generated from linguistic expressions. In [7], the problem-solving is performed on the conceptual level using the ontology. However, there are some open problems even in these systems. For instance, the former system deals with only limited world, and the latter one should prepare a large amount of knowledge beforehand.

This paper considers our everyday communication in which mutual understanding is possible even with ambiguous linguistic expressions through the interaction. This study tries to construct an end-user programming system by which end-users perform computer programing according to users' requests through the interaction using linguistic expressions. Text editing is considered as the example of computer programming because users easily understand the task and how to complete it, even if they have no any experience of programming.

2. Programming by Paraphrasing

The conversion of linguistic expressions into a machine language is regarded as a machine translation between two languages. The translation process is divided into two parts, the translation part (in a narrow sense) and the paraphrasing part as shown in Fig. 1. The set L of linguistic expressions has too many elements. On the other hand, the set M of a machine language has a fewer elements than that of L. Let L_0 be the set of which elements have approximately one-to-one correspondence to those of M.

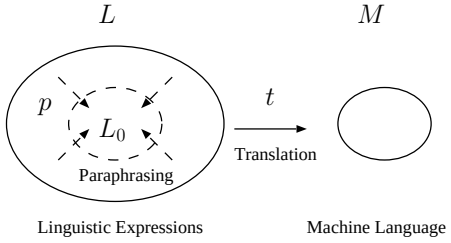

Figure 1: Translation and paraphrasing

The translation in a narrow sense is defined by a map t from specific linguistic expressions to a machine language. Paraphrasing is defined as a map p from linguistic expression in L into some expression in L_0 which is translated into M. Since the paraphrase is the conversion within the language L, it has the advantage that even users without the special knowledge on a computer language understand the conversion. As for the translation, however, it is assumed that knowledge on the translation is prepared beforehand and that users do not correct it nor add new knowledge.

Conventional programming languages have many keywords whose meanings are strictly pre-defined. In this study, the processes represented by linguistic expressions are obtained by interaction with users dynamically, rather than prepared beforehand. From this point of view, the case-based reasoning method is used. Paraphrasing is the way to compare cases with linguistic expressions and to add new cases into the database.

3. System Structure

The structure of the presented system is shown in Fig. 2. Computer programming is performed by the translation of linguistic expressions of users' requests to a machine language, and by the paraphrase through the interaction between the system and users. In this paper, the case-based reasoning method is used for the paraphrase and the translation, and users add the paraphrase case into the case database through the interaction using linguistic expressions. The system has two kinds of database, the paraphrase case database which describes the correspondence between two linguistic expressions, and the translation case database which describes the correspondence between a linguistic expression and a computer program expression. The system outputs the paraphrase or the translation according to the results of the case-based reasoning. The system is implemented by Emacs Lisp on Meadow Version 2.00 pre1, which is a Windows port of the GNU Emacs.

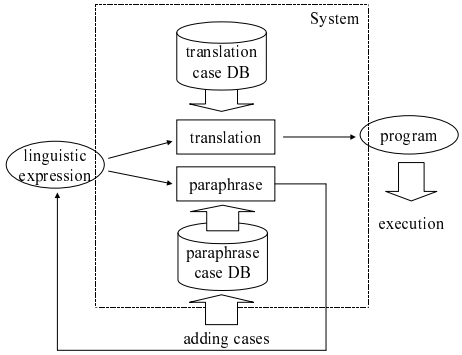

Figure 2: System structure

3.1 Morphological and Syntactic Analyses

The morphological and the syntactic analyses are applied to the generation of a syntax tree of inputted linguistic expressions. The inputted expressions are compared with the linguistic expressions in the case database based on the generated syntax tree. Morphological analysis system "ChaSen" [8] is used to obtain a morphological sequence from Japanese linguistic expressions. A morphological sequence includes a part-of-speech, a conjugated type, a conjugated form, a base form, and a pronunciation of each word. Context-free grammar rules shown in Table 1 are defined for the syntactic analysis, and the syntax tree is composed of words and phrases based on these grammar rules. As the result of the syntactic analysis, some features, which are obtained both from the grammar rule for the generation of the phrase and from attributes of all words in the phrase, are added to a phrase. Table 2 shows features obtained from attributes of words. As for features obtained from the grammar rules, for example, let us consider the situation that the grammar rule for the generation of a supplementary phrase from a noun and a case-marking particle is applied to some phrase. The grammatical features such as case-marking of "wo" and "ni" are added to the phrase. The features obtained in this way are employed in the case-based reasoning and the output generation.

Table 1: Grammar rules

Clause	→	Supplementary phrase + Predicate phrase
Clause	→	Supplementary phrase + Clause
Clause	→	Conjunction + Clause
Clause	→	Adverb + Clause
Clause	→	Noun (adverbal) + Clause
Clause	→	Exclamation
Supplementary phrase	→	Noun phrase + Case-marking particle
Noun phrase	→	Noun phrase (adnominal) + Noun phrase
Noun phrase	→	Noun phrase + Particle
Noun phrase	→	Adnoun + Noun phrase
Noun phrase	→	Clause + Noun
Noun phrase	→	Number + Numerative
Verb phrase	→	Verb phrase + Attached verb
Verb phrase	→	Verb phrase + Auxiliary verb
Verb phrase	→	Verb phrase + Particle
Verb phrase	→	Verbal noun ('Suru'-verb stem) + Word "Suru"
Verb phrase	→	Verbal noun ('Suru'-verb stem) + Period (clause end)
Predicate phrase	→	Verb phrase
Predicate phrase	→	Predicate phrase + Auxiliary verb
Predicate phrase	→	Noun phrase + Copula

Table 2: Features obtained from attributes of words

part-of-speech	attribute	value	features
Auxiliary verb	Conjugated form	Subjunctive	subjunctive, subordinative, conjunctive
	Conjugated type	unusual "non"	negative
Particle	Part-of-speech	adverbial/coexistence/ sentence-final particle	interrogative
		conjunctive particle	conjunctive
Adverb	Base form	"all"	all

3.2 Case-Based Reasoning

Once a syntax tree is obtained from inputted linguistic expressions, the expressions are compared with cases in the case database based on the syntax tree. The comparison process is performed in the following way:

- The comparison of clauses is performed in the way that the clause is divided into the predicate and supplementary phrases, and the comparison is performed between predicates and between supplementary phrases. The

supplementary phrase is a part of a sentence, which is expressed in the form of "Noun + Case-marking particle", reinforcing the meaning of a predicate.

- The comparison of phrases is performed in the way that words included in a phrase are compared with each other sequentially. In the noun phrase comparison, if the last noun fits some case in the database, it is assumed that the presented noun phrase fits the case. In the verb phrase comparison, if the first verb fits some case in the database, it is assumed that the verb phrase fits the case.

- As for the words comparison, the part-of-speech and the reading of the base form are compared with each other.

Words in the inputted linguistic expression not corresponding to any words in the case are disregarded in order to perform the comparison successfully even if there are some extra words in the inputted linguistic expressions. On the other hand, if any words in the case do not fit words in the inputted linguistic expressions at all, the comparison is assumed to be failure. If some cases are obtained as the result of the comparison, the number of incomplete matches is considered in order to choose. The case with the smallest number of incomplete matches is chosen in the first place.

3.3 Output Generation

The translation case database is prepared beforehand in the form as shown in Table 3. The part in the case, which fits some part in the inputted linguistic expression, is called variable. The variable has the type, integer or string, and the variable fits any word or phrase if their types are same as the variable. In the translation case, the variable is expressed, e.g., as "`?x<type>`", in the linguistic expression and is expressed as the parameter of a `lambda` expression in a computer language. When a case is found in the translation case database, the corresponding part of the inputted linguistic expressions is substituted into the variable in the case.

The subordinate clause in a complex sentence is translated into the conditional clause. The subordinate clause is a clause that has the feature *subjunctive* and *subordinative*. If the subordinate clause has the feature *all*, the clause is translated into the *while*-clause. Otherwise, it is translated into the *if*-clause. The clause without these features is translated into the parallel clause. The sequence of the translation results of all these clauses is the translation result.

The paraphrase is performed in the same way as the translation, where the output is not a computer program but a linguistic expression.

3.4 Addition of Paraphrase Case

If no cases are found in the database corresponding to the inputted linguistic expression, the system asks users to paraphrase it into another expression in order to continue the process. This is the addition of a paraphrase case, and the added linguistic expression is available in the next interaction. An example of the addition of the paraphrase case through the interaction is shown in Fig. 3. In this interaction, a user defines the meaning of the linguistic expression "quote this line".

Table 3: Translation database (an extract)

	linguistic expression	translation result
1	move to the left	`(lambda nil (backward-char))`
2	move to the right	`(lambda nil (forward-char))`
3	move ?x<integer> characters left	`(lambda (x) (backward-char x))`
4	move ?x<integer> characters right	`(lambda (x) (forward-char x))`
5	move to the previous line	`(lambda nil (previous-line 1))`
6	move to the next line	`(lambda nil (next-line 1))`
7	move up ?x<integer> lines	`(lambda (x) (previous-line x))`
8	move down ?x<integer> lines	`(lambda (x) (next-line x))`
9	move to the previous word	`(lambda nil (backward-word 1))`
10	move to the next word	`(lambda nil (forward-word 1))`
11	move to the beginning of line	`(lambda nil (beginning-of-line))`
12	move to the end of line	`(lambda nil (end-of-line))`
13	move to the beginning of document	`(lambda nil (goto-char (point-min)))`
14	move to the end of document	`(lambda nil (goto-char (point-max)))`
15	search ?x<string>	`(lambda (x) (re-search-forward x nil t))`
16	replace ?x<string> to ?y<string>	`(lambda (x) (replace-regexp x y))`
17	insert ?x<string>	`(lambda (x) (insert x))`
18	delete ?x<integer> characters	`(lambda (x) (delete-char x))`
19	begin selecting a region	`(lambda nil (set-mark-command nil))`
20	cancel selecting a region	`(lambda nil (deactivate-mark))`
21	delete the selecting region	`(lambda nil (and mark-active (delete-active-region)))`
22	is the beginning of line	`(lambda nil (bolp))`
23	is the end of line	`(lambda nil (eolp))`
24	is the beginning of document	`(lambda nil (bobp))`
25	is the end of document	`(lambda nil (eobp))`
26	copy	`(lambda nil (kill-ring-save (point) (mark)))`
27	paste	`(lambda nil (yank))`
28	cut	`(lambda nil (kill-ring (point) (mark)))`
29	is ?x<string> at the beginning of line	`(lambda (x) (save-excursion (beginning-of-line) (looking-at x)))`
30	is ?x<string> at the end of line	`(lambda (x) (save-excursion (beginning-of-line) (looking-at (concat ".*" x "$"))))`

```
> quote this line
No candidates: Please paraphrase ‘‘quote this line’’.

> insert "> " at the beginning of line
Performed normally.

>
```

Figure 3: Addition of paraphrase case

4. Experiment

In order to verify the usefulness of the presented system, some subject experiments are performed. In the experiments, the text shown in Fig. 4 is edited into the text shown in Fig. 5. The subjects should (1) change JPEG files and AVI files into IMG tags and A tags, respectively, and (2) erase the other files, i.e. the files with "~", ".BAK" or "Thumbs.db", because they are unnecessary. The experimental results are shown in Table 4 and Table 5. It is found that different paraphrase cases are obtained by each subject. Subject 1 obtains the computer program as shown

in Fig. 6. It is confirmed that the text shown in Fig. 4 is transformed into the text in Fig. 5 by the execution of the computer program shown in Fig. 6.

```
photo(hanami2004)/hanami2004-1.JPG
photo(hanami2004)/hanami2004-1.JPG~
photo(hanami2004)/hanami2004-2.JPG
photo(hanami2004)/hanami2004-3.JPG
photo(hanami2004)/hanami2004-4.JPG
photo(hanami2004)/hanami2004-5.JPG
photo(hanami2004)/hanami2004-6.AVI
photo(hanami2004)/hanami2004-7.JPG
photo(hanami2004)/Thumbs.db
photo(graduation2004)/DSCF1.JPG
photo(graduation2004)/DSCF2.JPG
photo(graduation2004)/DSCF3.JPG
photo(graduation2004)/DSCF4.JPG
photo(graduation2004)/DSCF5.JPG
photo(graduation2004)/DSCF6.JPG
photo(graduation2004)/DSCF6.JPG.BAK
photo(graduation2004)/DSCF7.JPG
photo(graduation2004)/DSCF8.JPG
photo(graduation2004)/DSCF9.JPG
photo(graduation2004)/DSCF10.JPG
photo(graduation2004)/Thumbs.db
```

Figure 4: Text before editing

```
<IMG SRC="photo(hanami2004)/hanami2004-1.JPG"><BR>
<IMG SRC="photo(hanami2004)/hanami2004-2.JPG"><BR>
<IMG SRC="photo(hanami2004)/hanami2004-3.JPG"><BR>
<IMG SRC="photo(hanami2004)/hanami2004-4.JPG"><BR>
<IMG SRC="photo(hanami2004)/hanami2004-5.JPG"><BR>
<A HREF="photo(hanami2004)/hanami2004-6.AVI">hanami2004-6.AVI</A><BR>
<IMG SRC="photo(hanami2004)/hanami2004-7.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF1.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF2.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF3.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF4.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF5.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF6.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF7.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF8.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF9.JPG"><BR>
<IMG SRC="photo(graduation2004)/DSCF10.JPG"><BR>
```

Figure 5: Text after editing

Table 4: Paraphrases obtained by subject 1

	before paraphrasing	after paraphrasing
1	do the image tag 1.	move to the beginning of line. insert “<IMG SRC=”.
2	do the image tag 2.	move to the end of line. insert “”> ”.
3	add an image tag.	do the image tag 1. do the image tag 2.
4	add the image tag.	add an image tag.
5	do the link tag 1.	move to the beginning of line. insert “<A HREF=””. search “/”.
6	do the link tag 2.	begin the selecting. move to the end of line. copy. insert “”>”. paste.
7	do the link tag 3.	move to end of line. insert “</A>”.
8	add the link tag.	do the link tag 1. do the link tag 2. do the link tag 3.
9	delete one line.	move to the beginning of line. begin the selection. move to the end of line. delete the selection. delete 1 character.
10	do the tagging decision 2.	if there is “AVI” at the end of line, add the link tag.
11	do the tagging decision 3.	if there is “JPG” at the end of line, add the image tag.
12	insert the tag.	do the tagging decision 2. do the tagging decision 3.
13	do the deletion 1.	if there is “.BAK” at the end of line, delete one line.
14	do the deletion 2.	if there is “~” at the end of line, delete one line.
15	do the deletion 3.	if there is “Thumbs.db” at the end of line, delete one line.
16	delete the excessive line.	do the deletion 1. do the deletion 2. do the deletion 3.
17	do the process.	insert the tag. delete the excessive line.

Table 5: Paraphrases obtained by subject 2

	before paraphrasing	after paraphrasing
1	insert “<IMG SRC=”” at the beginning of line.	move to the beginning of line. insert “<IMG SRC=””.
2	insert “”> ” at the end of line.	move to the end of line. insert “”> ”.
3	add the IMG tag.	insert “<IMG SRC=”” at the beginning of line. insert “”> ” at the end of line.
4	is “JPG”.	is “JPG” at the end of line.
5	do the image process.	if there is “JPG”, add the IMG tag.
6	insert “”>” at the end of line.	move to the end of line. insert “”>|’’.
7	insert “<A HREF=”” at the beginning of line.	move to the beginning of line. insert “<A HREF=””.
8	do the end-of-line process of A tag.	search “/”. begin selecting. move to the end of line. copy. insert “”>” at the end of line. paste.
9	insert “</A> ” at the end of line.	move to the end of line. insert “</A> ”.
10	add the A tag.	insert “<A HREF=”” at the beginning of line. do the end-of-line process of A tag. cancel selecting. insert “</A> ” at the end of line.
11	is “AVI”.	is “AVI” at the end of line.
12	do the anchor process.	if there is “AVI”, add the A tag.
13	do the tagging process.	do the anchor process. do the image process.
14	delete this line.	move to the beginning of line. begin selecting. move to the end of line. delete the selecting region. cancel selecting.
15	is “BAK”.	is “BAK” at the end of line.
16	delete the line of “BAK”.	if there is “BAK”, delete this line.
17	is “db”.	is “db” at the end of line.
18	delete the line of “db”.	if there is “db”, delete this line.
19	is “~”.	is “~” at the end of line.
20	delete the line of “~”.	if there is “~”, delete this line.
21	do the deleting process.	delete the line of “~”. delete the line of “db”. delete the line of “BAK”.
22	achieve the goal.	do the tagging process. do the deleting process.

An example of interactions in the experiment is shown in Fig. 7. The system confirms that the first input can be paraphrased by the combination of the translation and paraphrase cases already defined. The user answers “yes”, and the translation is performed. On the other hand, the input “*do the tagging decision 2*” is quite a new linguistic expression. While the system retrieves some cases and asks the user whether they have similar meaning of the inputted expression or not, the user continues answering “no”. Therefore, the system asks the user to paraphrase inputted linguistic expression, and the user paraphrases it into another expression.

Linguistic expressions not only prepared beforehand but also saved in paraphrasing case database can be used for paraphrasing. Any linguistic expression is used for making a program, whether it is defined or not. If it is already defined, the translation process is performed and the program corresponding to the inputted linguistic expression is generated. If it is not defined yet, the system asks a user to paraphrase it. After paraphrasing, the translation process is performed in the same way. At the same time, the linguistic expression for paraphrasing is added to the case database. Therefore, a user can make a program with their own words by paraphrasing undefined linguistic expressions into defined ones.

However the system has some problems. In this system, translation cases must be prepared beforehand since a user cannot input translation cases by paraphrasing. Therefore, it is required to estimate how many translation cases are necessary for the system.

```
(progn
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "AVI" "$")))
      (progn
        (beginning-of-line)
        (insert "<A HREF=\"")
        (re-search-forward "/" nil t)
        (set-mark-command nil)
        (end-of-line)
        (kill-ring-save (point) (mark))
        (insert "\">")
        (yank)
        (end-of-line)
        (insert "</A>")))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "JPG" "$")))
      (progn
        (beginning-of-line)
        (insert "<IMG SRC=\"")
        (end-of-line)
        (insert "\"><BR>")))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" ".BAK" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1)))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "~" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1)))
  (if
      (save-excursion
        (beginning-of-line)
        (looking-at (concat ".*" "Thumbs.db" "$")))
      (progn
        (beginning-of-line)
        (set-mark-command nil)
        (end-of-line)
        (and mark-active
             (delete-active-region))
        (delete-char 1))))
```

Figure 6: Program generated for subject 1

```
> if there is ‘‘AVI’’ at the end of line, add the link tag.
Translation: Are you sure to execute ‘there is ‘‘AVI’’ at the end of line
(subjunctive). do the link tag 1. do the link tag 2. do the link tag 3.’?

> yes
Performed normally.

> do the tagging decision 2.
Translation: Are you sure to execute ‘do the link tag 2.’?

> no
Translation: Are you sure to execute ‘do the image tag 2.’?

> no
No candidates: Please paraphrase ‘do the tagging decision 2’.

> if there is ‘AVI’ at the end of line, add the link tag.
Performed normally.
```

Figure 7: Example of interactions in the experiment

5. Conclusion

This paper proposes an end-user programming system using linguistic expressions. The system consists of the paraphrasing part and the translation part. In the former part the system has the paraphrase case database. Users can add paraphrase cases into the database by the interaction with the system by linguistic expressions. In the latter part the system has the translation case database that describes corresponding between linguistic expressions and computer program language expressions, which is fixed beforehand. The result of the subject experiments using the present system show the feasibility and the following usefulness of the system: (1) Users can make computer programs without any knowledge on a computer programming language. (2) The computer programming with linguistic expressions is easier than that with a special computer programming language.

The followings are future problems: (1) the improvement of the present system for the realization of more natural interaction, and (2) the estimation of the amount of the necessary translation cases.

References

[1] Henry Lieberman: “Your Wish is My Command: Programming by Example”, Morgan Kaufmann Publishers (2001).

[2] Amit Konar: “Artificial Intelligence and Soft Computing”, CRC Press (1999).

[3] Terry Winograd: “Understanding natural language”, Academic Press (1972).

[4] Minoru Harada: “A New Paradigm for Automatic Programming”, Journal of Japanese Society for Artificial Intelligence, Vol. 6, No. 2, pp. 19-23 (1991). (in Japanese)

[5] I. Kobayashi, T. Sugimoto, S. Iwashita, M. Iwazume, J. Ozawa and M. Sugeno: “A new communication method using natural language as a computer communication protocol”, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 7, No. 2, pp. 215-222 (2003).

[6] David Price et al.: “NaturalJava: A Natural Language Interface for Programming in Java”, Proc. of 5th International Conference on Intelligent User Interfaces, New Orleans, Louisiana, pp. 207-211 (2000).

[7] Kazuhisa Seta, Mitsuru Ikeda, Osamu Kakusho, and Riichiro Mizoguchi: Capturing a Conceptual Model for End-User Programming: Task Ontology As a Static User Model, Proc. of 6th International Conference on User Modeling, Chia Laguna, Sardinia, pp. 203-214 (1997)

[8] Yuji Matsumoto, et al.: Morphological Analysis System ChaSen Version 2.2.1 Manual, Nara Institute of Science and Technology (2000).