TR-067

NATURAL LANGUAGE BASED SOFTWARE DEVELOPMENT SYSTEM TELL

by
Hajime Enomoto, Naoki Yonezaki, Motoshi Saeki
(Tokyo Institute of Technology)
Kazuhiko Chiba, Takashi Takizuka
(KDD Research and Development Laboratories)
Toshio Yokoi
(ICOT)

June, 1984

NATURAL LANGUAGE BASED SOFTWARE DEVELOPMENT SYSTEM TELL

Hajime Enomoto, Naoki Yonezaki, Motoshi Saeki
Department of Computer Science
Tokyo Institute of Technology
2-12-1 Ookayama Meguro-ku Tokyo 152 Japan

Kazuhiko Chiba, Takashi Takizuka
KDD Research and Development Laboratories
2-1-23 Nakameguro Meguro-ku Tokyo 153 Japan

Toshio Yokoi
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F
1-4-28 Mita Minato-ku Tokyo 208 Japan

A software development method based on natural language description and its support system named TELL are presented. Very high level specification language NSL is designed to describe software systems in a natural way and precisely. The language is a fragment of natural language. By limiting paradigms of natural language, we can express software systems structurally and comprehensively. Structure of a software system reflects structure of natural language semantics i.e. the way how we understand facts or events. In this abstraction specification method, we introduce a new module decomposition principle which is based on lexical decomposition. Descriptions in NSL can be translated into formulas of modal logic so that rigorous semantics is determined and semantical treatment by machine becomes available. TELL system provides natural language interface and semantical processing functions. Project of TELL system is outlined along its structure.

INTRODUCTION

A software life cycle is roughly divided into the following steps i.e. requirement specification, module design, precise design, coding, test, debug and maintenance. It is almost all the cases that these steps are not straightforwards, since changes and backtracking at various steps are needed. Those turnarounds are so costly that software crisis is worried about. A difficult and critical step in the life cycle of software systems is concerned with requirement specification and maintenance of a system. In a specification step, we have to describe requirements unambiguously, consistently and precisely. It is also preferable that the requirement description can be understood easily. On the other hand, one of the reason for such costly backtracking is that it is difficult to write specifications completely. This fact supports validity of rapid prototyping. However, specifying systems as precisely as possible and verifying specifications are needed to reduce the software development and maitenance cost.

These requirements may concern with human mechanism of conception and logical treatment. Conception mechanism has close relation to structure of natural language. Logical treatment employs many important concepts such as independency, orthogonality, abstraction, funccionality, referential transparency and so on.

Tell system has been designed to include above concepts and overcome many difficulties appeared in many previously developed software development systems. Specific features of Tell systems (23) are as follows.

1) Use of natural language like language with restricted syntax and semantics as a specification language.
   Semiotics describes there are a large number of parallelisms between mental behaviour of human beings and expressive structure of natural language. Therefore style of natural language should be adopted but paradigms are kept limited for simplicity of computer processing.
2) Use of symbolic expressions and iconic symbols.
   In some case, it is cumbersome and incomplete in logical sence to express everything in natural language. Then some symbols having artificial concept may be used. These iconic symbols are useful for symbolic expression of some operations.
3) Use of itemized sentence representation of object.
   Itemized sentence representation is widely used for arrangement of facts and is easy to understand total properties of an object because of decomposition of properties which are mutually independent. This is helpful for better debugging and maintenance.
4) Use of generic type of objects.
   If an object must be represented individually until atomic level for everything, variation of representation becomes enormously large according to the number of types. In order to avoid the matter, generic type of objects is allowed to use orthogonal or abstract feature efficiently.
5) Use of knowledge base.
   It is possible to represent facts effectively and association mechanism combined with use of generic type can be efficiently implemented in knowledge base.
6) Use of interface mechanism.
   It is necessary to have some interface mechanism between specification and implementation description, since specification is mainly represented in a static way but implementation has almost dynamic portion. Some interface mechanisms are necessary for description of a layered architecture. Interface section will be helpful to avoid duplicate description and increase maintainancibility.
7) Use of transformation mechanism to modal logic.
   Montague's approach is suitable for transformation of a fragment of natural language to modal logic expressions.(7) Therefore Tell system has a translator to logical expressions which are useful for further development.

Above features are described along the philosophy of Tell system. If there exist some integrated specifications, readers may guess the rough sketch of the specifications by Tell system. Therefore, TELL system is designed to be available throughout software life cycle.

TELL/NSL(Natural language like Specification Language) is its kernel language. Specification is written in the natural language (a fragment of English), which is used for decomposing a module into sub-modules based on lexical decomposition. This guideline is the one which is different from the decomposition method proposed before.(1)~(5) The semantics of the sentences in the natural language is defined formally. Our specification method belongs to an abstract module (8)~(10) family of specification methods.

Another important objective of this paper is to demonstrate that a restricted natural language can be used for formal specifications, and that lexical decomposition presents natural module decomposition so that it can be understood easily and gives not only a way to mechanization of formal semantics treatment such as verification, consistency checking and document generation but also good human interface and guidelines.

DESCRIPTION STRUCTURE OF TELL/NSL

Some of [illegible] for requirement specification languages are discussed in great detail in (6). Especially communicability is strongly requested since multiple persons are concerned in developing a software system.

Natural language is used to describe specifications in general. It is very natural and adequate for being read by men to describe specifications in a natural language.(12) However, the semantics of a sentence in an unrestricted natural language tends to be ambiguous, so that it does not fit to formal specification and semantical processing by machine, and it has been used just for human documents and comments. On the other hand requirement description method based on logic or algebra whose specification body is a set of axioms are studied. (8),(9),(13) However, it is difficult to read for untrained persons in mathematics.

Therefore, one effective method is to provide a concrete relation for limited natural language paradigms with logical expressions, so that the paradigms refrect naive structures of the world model in our mind, since structure of concepts follows the structure of natural language paradigms. One of the main objectives of this research is to develop a subset of natural language satisfying properties described above.

In general, a software system is a model of a real world. A word in a natural language is used for representing a unit of concepts. If we select an appropriate word which represents a concept of an object or a matter in a real world modeled by a software system, we can get image on the system at least vaguely by the word at a glance.

Meanings of the words used in the sentences except for reserved words are specified by a set of sentences hierarchically. This specification method corresponds to the notion of lexical decomposition. In the field of natural language semantic theory, the method that describes semantics of a word by the construction of more atomic semantic elements or atomic concepts is called lexical decomposition method. The syntax of the restricted natural language is so simple that selection of words which corresponds to module decomposition is guided.

In brief, the way to define the meanings of words is to give
1) a collection of itemized sentences which is semantically equivalent to them
   or
2) a collection of itemized sentences which represents relations between the other words related to each other.

We consider that each word appearing in a specification corresponds to a software module, so in this specification method, a software system is decomposed into smaller software modules according to occurrences of the words in sentences of a specification. Since each itemized sentence in a specification is simple sentence easy to be understood the number of words appearing in one sentence is small, then interface of software modules corresponding to the words becomes simple. When we construct specifications by using itemized sentences, we select words unconsciously, as the result we can get natural module decomposition.

Each itemized sentence in natural language is translated into logical formula according to the semantic rules associated with syntactic rules. This translation rules are so naive and simple that there is no inconsistency between machine processed semantics and human understanding, and that it provides readability of sentences and no misreading.

Considering the general properties of words and their lexical categories, there are four kinds of modules of words or phrase definitions, that is functional, action, class, and dynamic class definition as shown in Fig.1.

Because Tell system has parameterization function by using generic words, it is avoided to define similar definitions many times. This leads to single writing principle in all documents. Furthermore, for all categories, additional integrity conditions can be added in 'satisfy' section as well as data integrity conditions, and the names of modules are modified by an adjective.

	static properties	dynamic properties
class	**class definition** Construction properties of the object words are defined with necessary operations (abstract data type) : common noun and adjective with generic common noun	**dynamic class definition** Time coordinating properties of manipulation words for the object word are supplementally defined (abstract data type + timing): common noun and adjective with verb and adverb
func-tion	Relational properties among a subject word and object words are defined (input -output relation): noun and adjective with generic common noun **functional definition**	Time coordinating properties of object words are defined. (input-output elation + timing): verb, common noun(converted from verb) with adverb **action definition**

Fig.1. Definition categories of words in TEll/NSL.

DESCRIPTION OF FUNCTIONAL SYSTEMS

In this section, we present language features of natural language like specification language NSL. NSL is designed for specifying both functional systems in which inputs and outputs relations of modules are essential, and dynamic systems in which dynamic concurrent behaviour of systems is concerned. To present fundamental functions of the language, we first concentrate on specification of functional systems. The syntax of this specification is as follows.

```
<Defining sentence>
   means that
     <Specification Body>
     <Functional sub-definitions>
     <Class definition>
     <Lexicon declaration>
   end
```

Fig. 2. The syntax of a specification.

Defining sentence declares a relation of words by a sentence of natural language. The relation corresponds to a module name and is presentend by a common noun or an adjective (phrase). Therefore only be-verb is used for defining functional definition. General verbs are used for defining dynamic definitions. These typical sentences may contain information about inputs and outputs of a module. Specification body is a set of itemized sentences which explains conditions that words in the defining sentence should satisfy. Those sentences can be written by symbolic expressions such as arithmetic expressions or logical formulas in the case that they seem more informative than sentences in natural language. Those are reserved words and there are other reserved words such as 'every', 'some', 'a', 'or' and 'and'.

Functional sub-definitions part consists of a set of specifications which have the same syntax as the functional definition. That is to say, this specification method supports hierarchical specification of modules. Defining sentence for each functional sub-definition contains a word which is used in the sentences in the specification body of its parent definition.

Class definition are written according to the following syntax similar to that of functional definition.

```
      < Class name> associated with  {related words}
          Construction
             { Sentences in natural language
                 which represent their functional type }*
           <Functional definition>
           <Class definition>
           <Lexicon declaration>
       end
```

Fig. 3 The syntax of class definition.

Class name is a word in natural language which are used as a common noun. When we specify a class, first we define words or phrases of natural language whose syntactic category is common noun or adjective, depending on range type of corresponding functions or predicates. Functional definition part in a class definition defines words representing attributes of the class. Words which follow 'associated with' should be defined in the class definition and can be used in other definitions.

When a common noun is modified by an adjective, such a noun phrase is considered as a sub-class of a class represented by the common noun, for example, a noun phrase 'finite set' corresponds to a sub-class of class 'set'. If a common noun is modified by prepositional phrase, this common noun corresponds to a meta-class and the noun phrase with propositional phrase is its instance, for example a noun phrase 'sequence' corresponds to a meta-class of class instance 'sequence of integer'. This concerns with a generic type.

Be-verb is also used for defining sub-class. Sentence 'A is B.' is interpreted as a class represented by A is a sub-class of a class represented by B, and A inherits all the properties associated with B. An example is shown as follows.

```
      Natural number is integer.
       satisfy
      1)Every natural number is greater than 0.
      end
```

Conditions that each operation should satisfy are specified in the part of specification body. Those conditions consist of sentences in natural language with which logical expression can be mixed. Class expressions are also allowed. In this case, a class is defined according to the following syntax.

```
       <Class name> is
           <Class expression>
        end
```

Fig. 4. The syntax of class definition with class expression.

Class expression includes enumeration type, record type, function type and so on whose syntax is just like in Pascal. Finally it should be noted that synonymuos words can be defined in a lexicon declaration part.

Example of specification for eight queens' problem is given in Fig.5. Eight queens' problem is to compute the arrangement of eight queens on a standard chess board such that each row, column, and diagonal contains no more than one queen.

Eight queens problem is a software module which consists of two sub-definitions, checking and placed, and also three classes, chess board, queen and arrangement. First two sentences defines conditions that the solution should satisfy.

Sentences are written in an itemized style and each sentences should be satisfied conjunctively. There are undefined words, 'quuen', 'placed' and 'checking' in the sentences. The meaning of those words are defined in sub-definitions. For example in the specification of 'checking' definition, defined word 'checking' is introduced in a sentential form of 'Queen q1 is checking against queen q2 in arrangement X'. In the sentence 'queen' and 'arrangement' are used as common nouns which correspond to classes. Q1, q2 and X are used as proper nouns whose classes correspond to preceding words respectively.

```
 Arrangement X is a eight queens' solution
  means that
      1) Eight queens are placed in X .
      2) No queen is checking against
                       any other queen in X .

      Queen q1 is checking against queen q2 in arrangement X
       means that
         1) q1 and q2 are on the same row in X or
            q1 and q2 are on the same column in X or
            q1 and q2 are on the same diagonal in X .

        Queen q1 and queen q2 are on the same row in arrangement X
         means that
             1) The X-coordinate of the position of
                  q1 in X is the X-coordinate of the position of q2 in X .
        end  on the same row ;

        Queen q1 and queen q2 is on the same column in arrangement X
          means that
             1) The Y-coordinate of the position of
                  q1 in X is the Y-coordinate of
                           the position of q2 in X .
        end  on the same column ;

        Queen q1 and queen q2 is on the same diagonal
                   in arrangement X
          means that
               1) /X-coordinate[position[q1,X]] + Y-coordinate[position[q1,X]]
                    = X-coordinate[position[q2,X]] + Y-coordinate[position[q2,X]] v
                   X-coordinate[position[q1,X]] - Y-coordinate[position[q1,X]]
                    = X-coordinate[position[q2,X]] - Y-coordinate[position[q2,X]]/ .
        end  on the same diagonal ;

      end  checking ;

      Queen is
            [1..8] .
      end  queen ;

      Chessboard's square is
          (X-coordinate: [1..8], Y-coordinate: [1..8]) .
      end  chessboard's square

      Arrangement is
          sequence of  chessboard's square.
      end arrangement ;

    lexicon
        1) p is a position of q in X
                  := the index of p in X is q.
        2) c is a X-coordinate of y
                  := /c=X-coordinate[y]/ .
        3) c is a Y-coordinate of y
                  := /c=Y-coordinate[y]/ .
        4) q is placed in X
                  := /q≤lengh[X]/.

  end eight queens' solution
```

Fig. 5. Specification of eight queens' problem.

Prepositions 'against' and 'in' are used for determining positions of arguments. The reason is because 'checking' is translated three place predicate, and q1, q2 and X are used for its arguments. Prepositions, however, have no semantics but work just like markers. As mentioned before, when sentences in natural language become complex, that is, notion we want to specify is essentially a mathematical one, we can use symbolic expression instead of one in natural language. In that case, expressions are parenthesized by /../. Definition 'on the same diagonal' is such an example.

DESCRIPTION OF DYNAMIC SYSTEMS

Our specification language is also designed to specify by natural language sentences such a system in which timing of process action is very important requirement. One of the main diffrent aspects from a description of a functional system is that we take a state notion into account. In a specification of a dynamic system, we use a general verb which is not a be-verb and which describes timing of process called 'Action'.

As well known, a multi-processing system which accesses shared objects needs synchronization. Usually resources can be specified as a class. However, in this formalism specifications of synchronization are implicit. We introduce here an alternative definition called 'Dynamic class' that consists of a class and an explicit synchronization specification. We explain specification description method of dynamic concurrent systems using the example of the description of alternating bit protocol (AB-protocol), which is a part of layered architecture description(23).

(Alternating bit protocol)
Fig.6. shows the block diagram of alternating bit protocol, which has a simple recover facility for transmission errors, e.g. loss, duplication, or delayed transmission, by repeated transmissions.

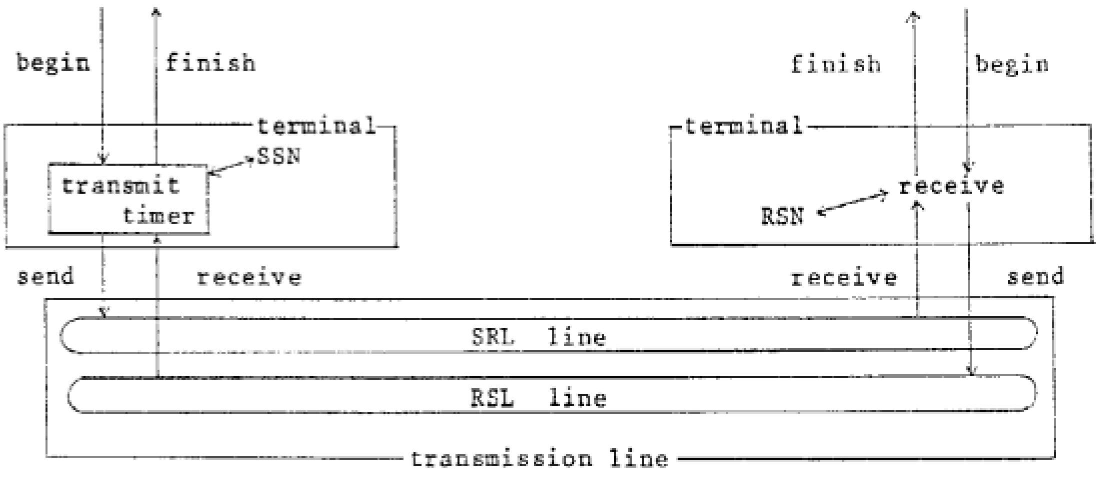


Fig. 6. Block diagram of AB protocol.

The scenario of transmitting one data is as follows. A transmission data consists of a message (bit sequence) and one bit sequence number called alternating bit. Alternating bit is complemented and added to a message whenever a new message is transmitted. After the transmitter sends a transmission data through SRL line to the receiver, it waits until it receives the acknowledgement through RSL line from the receiver. The receiver knows the value of an alternating bit which should be sent in the next transmission.

When the receiver receives a data, it checks its own alternating bit with that of the received data. If the check of the alternating bit is successful, the receiver has correctly received the data. The receiver sends an acknowledgement which is the same as the received data to transmitter through RSL line to tell that the check was successful. If transmitter receives no acknowledgement in a time-out period or the received acknowledgement is different from the sent data,

```
AB-protocol machine is the system such that
 Configuration
     1) There are line RSL and line SRL.
     2) There are counter SSN and counter RSN.
 Timing
 1) Initially RSL and SRL is empty.
 2) Initially RSN and SSN is 0.

 It transmits sequence of bit t means that
   1) Initially it begins to send the message made
       from sequence of bit t and SSN to line SRL.
   2) If it finishes sending to line SRL,
        then it begins to read from line RSL
          and timer is started.
   3) If it finishes reading acknowledgement A
          from line RSL and
    3-1) the alternating bit of A is SSN,
          then it begins to complement SSN.
    3-2) the alternating bit of A is not SSN,
          then it begins to send the message
            made from sequence of bit t and SSN
                 to line SRL
           and timer is reset.
   4) If it finishes complementing SSN,
        then it finishes transmitting.
    5) If it doesn't finish reading from line RSL
           until timer is time-out,
        then it begins to send the message
            made from sequence of bit t and SSN
                 to line SRL
           after timer is time-out.
 end  transmit ;

 It receives a sequence of bit means that
   1) Initially it begins to read from line SRL.
   2) If it finishes reading
        message m from line SRL and
    2-1) the alternating bit of m is RSN,
          then it begins to reply acknowledgement m
                   to line RSL
            and it is successful in receiving
                 the data unit of m
                   until it finishes receiving.
    2-2) the alternating bit of m is not RSN,
          then it begins to reply acknowledgement m
                   to line RSL and
            it is not successful in receiving
            until it finishes replying to line RSL.
   3) If it finishes replying to line RSL and
      it is successful in receiving,
        then it begins to complement RSN.
   4) If it is successful in
            receiving sequence of bit t
        and it finishes complementing RSN,
       then it finishes
             receiving sequence of bit t.
   5) If it finishes replying to line RSL and
      it is not successful in receiving.
         then it begins to read from line SRL.

  Lexicon
     1) It replies acknowledgement A to line RSL
          := it sends message A to line RSL.
 end  receive ;
```

```
Line associated with  send, read, and empty line
  construction
   1) It sends message m to line l
        := the result of sending m to l
               is a line.
   2) It reads message m from line l
        := the result of reading from l
               is a message and a line.
   3) Empty line is a line.
 Satisfy
   1) Message m is correctly conveyed
                through line l
      or message m is lost to line l.

 Message m is correctly conveyed through line l
  means that
    1) The result of reading from
        the result of sending m to l is m and l.
 end  correctly conveyed ;

 Message m is lost to line l
  means that
    1) The result of sending m to l
          is empty line.
 end  lost ;

 Timing
   1) If it begins to read,
        then it is waiting to read
          until line is not empty.
   2) If it is waiting to read
       and line is not empty.
         then it will be reading.
   3) If it begins to send infinitely often,
        then it will finish sending
            and then line is not empty.
 end line;

 Timer  is
      started,reset,  or stopped .

  Timing
     1) If timer is started,
         then timer will be time-out
              or reset.
     2) If timer is reset,
         then timer isn't time-out
              until timer is started.
 end  timer ;

 Counter is  boolean .
 end  counter;

 Lexicon
   1) message := sequence of bit.
   2) acknowledgement := sequence of bit.
   3) Message m is made
       from sequence of bit t and bit b
        := m is the concatenation of b and t.
   4) The alternating bit of message m is bit b
        := b is the head of m.
   5) The data unit of message m
         is sequence of bit t
        := t is the tail of m.
end  AB-protocol machine
```

Fig. 7. Specification of AB protocol.

transmitter sends the data repeatedly until it gets the expected acknowledgement. At this time, alternating bit added to the message is not complemented. The protocol guarantees correctly sequenced delivery of messages even if messages and/or ackowledgements are lost to the line.

In Fig.6, SSN and RSN are registers of current alternating bits and 'timer' lets a transmitter know that time-out event occurs. Fig.7. shows a part of the specification of alternating bit protocol in Tell/NSL. 'AB-protocol machine', 'transmit', and 'receive' are defined by action definitions, and 'line', 'timer', and 'counter' by dynamic class definition. Proper nouns 'SRL' and 'RSL' are declared as instances of dynamic class 'line'. Instances of classes are declared in 'configuration' part, which presents the number of the actions and shared resources. Counter SSN and RSN, in which the value of alternating bit is stored, are declared in the same way. Two sentences following the 'configuration' part present the initial values of share resources.

Itemized sentences in action definitions 'transmit' and 'receive' are specification bodies respectively and present the scenario of AB protocol in detail. The verbs or adjectives modified by infinitive or gerund of verbs, e.g. 'begin', 'finish', and 'successful' etc. represent execution states of the action corresponding to the modifying verb. If the declaration part of words representing states is omitted, words modified by infinitive or gerund of verb corresponding to the action definition distinguish its execution states.

Dynamic class definition 'line' in Fig. 7. defines not only common noun 'line' but also proper noun 'empty line', verbs 'send' and 'read'. In 'satisfy' part, static relations among the words are specified. 'Timing' part specifies control and timing conditions in the same way as in a action definition. For example, first two sentences describe that the controls of 'read' action from line L depend whether line L is empty or not. Sentence 3) in Fig. 7. says that a data will not be lost if it is re-sent infinitely many times.

'Lexicon' part defines a new word available only in the definition as a synonym of a pre-defined word. First sentence 'message:=sequence of bit' declares 'message' as a new word, whose meanings is the same as 'sequence of bit' previously defined.

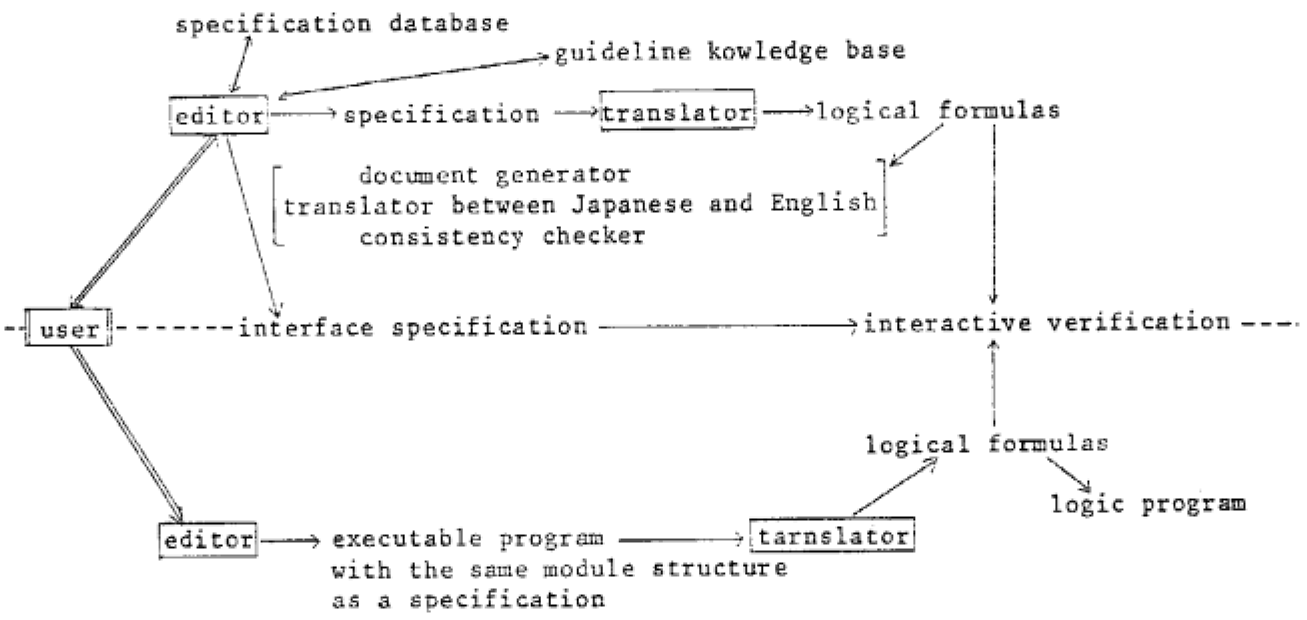
TELL SYSTEM

TELL system consists of various software tools to provide sophisticated software development environment. Such tools are structured editor with guideline for descriptions, logical formula generator, translator between distinguished natural languages, and so on.

Fig.8 shows the overview of Tell environment. Main part of the system is a structured editor. This editor includes interactive human interface for integrating a facility of specification of an object problem. Furthermore this editor will serve as an integrator for program generation of layered arcthitecture and verification purpose.

Specification database is used for specification designers to store the definitions of words which are often used. Therefore, it is possible to construct specifications hiding common knowledges which are stored in the database. The database can be considered to be a knowledge base for users. We have implemented prototypes of translator into logical formulas and of structured editor in English version on VAX/UNIX. Language design of NSL in Japanese version has been completed.

SPECIFICATION LANGUAGE PART



EXECUTION LANGUAGE PART

Fig. 8 Overview of Tell system.

CONCLUSION

We have introduced a software development system based on the structure of natural language which provides natural way of module design, readability and capability of semantical processing by machine such as verification, document generation.

Another advantage of our approach we should mention here is that there is a possibility to translate the specification into another natural language by giving correspondence of words of both languages and syntactic translation rules, since very simple semantics is exploited.

From the experiences of writing various specifications such as X.25 communication protocols, on-line character recognition system and text editor, we feel this specification method is very natural and comprehensive, and found out that more sophisticated environment or intelligent tools which support writing specification are desired.

Original idea of TELL (Total elaboration language) system has been proposed by Tokyo Institute of Technology. KDD Research Laboratory took an interest in potential power for dynamic concurrent system such as communication protocol systems, cooperated with Tokyo Tech. since 1983, and contributed to structured design, guidelines for description and X.25 protocol description. Simultaneously, Institute for new generation computer technology (ICOT) began the project of intelligent programming system. After several discussions, these three parties agreed to work together for Tell system. In order to achieve total rapid prototyping of Tell system within short period, Fujitsu and NEC joined this project and began close cooperation with them.

To integrate and minimize the specifications, it may be needed to exploit more expressive logic which can treat context information (21) or to exploit non-monotonic logic (22). Synthesis of a program from a specification in this style must be built and tested in a real environment. This is one major direction of future research.

Acknowledgement

We wish to thank Dr. Akira Kurematsu and Dr. Yoshiyori Urano of KDD (Overseas Telephon and Telegram corporation Japan) for insightful comments and numerous discussions. We would also like to thank many colleagues at TELL project for implementation in Tokyo Institute of Technology, KDD, Institute for New Generation Computer Technology, Fujitsu and NEC.

[REFERENCES]

(1) Parnas,D.L. : A Technique for Software Module Specification with Example, Comm. ACM, 15, 5 (May 1972) pp.330-336.
(2) Parnas,D.L. : On the Criteria to Be Used in Decomposing Systems into Modules, Comm. ACM, 15, 12 (Dec.1972) pp.1053-1058.
(3) Mealy,G.H. : Another Look at Data, Proc. of AFIPS, FJCC Vol31, AFIPS Press, Montrale, N.J., (1967), pp525-534.
(4) Liskov,B.H. : A Design Methodology for Reliable Software Systems, FJCC, (1972), pp191-199.
(5) Myers,G.J. : Reliable Software through Composite Design, Mason/Charter Pub. (1975).
(6) Liskov,B.H. and Zilles,S.N. : Specification Techeniques for Data Abstractions, IEEE SE-1, 1, (Mar. 1975) pp.7-19.
(7) Montague,R. : The Proper Treatment of Qualification in Ordinary English, Approaches to Natural Language, Reidel Dordrecht (1973). (8) Guttag,J.V., Horowitz,E., and Musser,D.R. : Abstract Data Types and Software Validation, Commun.ACM, Vol.21, No.12 (1978) pp.1048-1064.
(9) Goguen,J.A. : An Initial Algebra Approach to the Specification of Reliable Software, Boston M.A. (1979) pp.162-169.
(10) Hoare,C.A.R. : Proof of Correctness of Data Representations, Acta Informatica, Vol.1, No.4 (1972) pp.271-281.
(11) Gallin,D. : Intensional and Higher Order Modal Logic; With Applications to Montague Semantics, North-Holland Publishing, Amsterdam (1975).
(12) Teichroew,D. and Herskey III,E.A. : PSL/PSA:A Computer-aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Trans. Software Engineering, Vol. SE-3, No.1 (1977) pp.41-48.
(13) Nakajima,R., Honda,H., and Nakahara,H. : Hierarchical Program Specification and Verification -- a Many-sorted Logical Approach, Acta Informatica, Vol.14, Fasc.2 (1980) pp.135-155.
(14) Owicki,S. : Axiomatic Proof Techniques for Parallel Programs, Ph. D. Th., Cornell University, (Aug. 1975).
(15) Pnueli,A. : The Temporal Logic of Programs. 18th Annual Symposium on Foundations of Computer Science. (Nov. 1977) pp.46~57.
(16) Manna,Z. and Pnueli,A. : The Modal Logic of Programs, 6th International Colloquim on Automata, Language and Programming, Graz, Austria, Lecture Notes in Computer Science, Vol. 71, Springer Verlag (July 1979) pp.386~408.
(17) Hailpern,B. : Verifying Concurrent Processes Using Temporal Logic. Technical Report 195. Computer Systems Laboratory, Stanford University. (Aug. 1980).
(18) Keller,R.M. : Formal Verification of Parallel Programs, CACM 19, 7 (1976).
(19) Lamport,.L : Proving the Correctness of Multiprocess Programs, IEEE Transaction on Software Engineering 3, 2 (1977) pp.125~143.
(20) Yonezaki,N. and Katayama,T. : Functional Specification of Synchronized Processes Based on Modal Logic, Proc. of 6th ICSE (Sept. 1982) pp.208-217.
(21) Hausser,R. and Zaefferer,D. : Question and Answering in a Context-dependent Montague Grammar, Formal Semantics and Pragmatics for Natural Languages (Guenthner,F. and Schmidt,S.J. Ed.) , Reidel Dordrecht (1978) pp.339-358.
(22) McDermott,D. and Doyle,J. : Non-Monotonic Logic 1, Artificial Intelligence 13 (1980) pp.41-72.
(23)H. Enomoto, N. Yonezaki, M. Saeki, S.Kunifuji : Paradigms of Knowliged Based Software System and Its Service Image, Economics nad Technology of Software Engineering -3rd Seminar for Technology of Software (1983)
(24)H. Enomoto et al. : Formal Specification and Verification for Concurrent Systems by TELL, Proc. of ECAI-84 (1984)