American Journal of Applied Sciences 5 (4): 378-384, 2008
ISSN 1546-9239

SNL2Z: Tool for Translating an Informal Structured Software Specification into Formal Specification

Mohamed A. Sullabi and Zarina Shukur
Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia
Block F, FTSM, UKM, 43600 Bangi, Selangor Darul Ehsan

Abstract: In the area of software engineering there have been very few efforts to automate the translation from specifications written in natural language to the formal specification languages. Writing of the specifications in natural language is always depending on context and it is commonly vagueness; this represents the major reasons of the challenge. This paper discusses the design of a tool for translating a software specification written in natural language into a formal specification. We apply controlled natural language that limits the syntax and semantics when the natural language statements been written by proposing structured natural language (SNL) to avoid the ambiguity problem. The tool uses basic information about the operation schemas and statements describing the specification of the system written by a group of user collaboratively as input. The output of the tool is a translation and interpreting of the specification statements into equivalent statements in LAT_EX form, which are compiled to produce an equivalent statements in Z.

Key words: Collaborative writing, formal specification, machine translation, Z

INTRODUCTION

The natural language has remained as practiced choice to specify the software specification because formal specification languages are not easy to master [1]. Informal software specification normally has a lot of ambiguity, especially when it is read and interpreted by different people [13]. Natural language is descriptive with representation power, but its semantics and syntax are not formal enough to be used directly as a specification language. Therefore the requirements written in natural language has to be reinterpreted into a formal specification language[1], so that one can analyze the informal specification to reduce its ambiguity, and derive an efficient program which satisfies the specification [8].

Formal specifications, such as Z are based on mathematics. Even though formal specifications are very precise and accurate and they have been considered to be more effective in representing software specifications, and the benefit of using formal specifications is generally accepted by most of software practitioners, they are not widely used in software development[13] due to the additional technical knowledge needed [9]. Software developers find that writing of the formal statements is too complicated due to they are not familiar with mathematical notation [7].

Over the last decades, a few tools have been developed, such as SPECIFIER [11], RA [12], NL2ACTL [5], and FORSEN [17]. The major obstacle of the conversion of natural language into formal specification is from the inborn characteristic of ambiguity of natural language and the different level of the formalism between the two domains of natural language and the formal specification. This is why there have been very few attempts to automate the conversion from requirements documentation to a formal specification language [1].To handle this ambiguity problem of natural language, some have argued that the requirements document has to be written in a particular way to reduce ambiguity in the document [18]. Others have proposed controlled natural languages [6] which limit the syntax and semantics of NL to avoid the ambiguity problem.

A possible solution to this problem is by providing software developers with a tool that can help in writing structured natural language statements (SNLS), then translating these statements into mathematical statements. This paper discusses the design and implementation of tool that can help software developers in writing a semi-formal natural language

Corresponding Author: Block F, FTSM, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia UKM, 43600 Bangi, Selangor Darul Ehsan. Malaysia

specification describing the operation schemas of the software, then these specifications will be translated automatically into Z formal specifications in LAT_EX form.

Background of SNL2Z: SNL2Z is a part of web-based system that has been developed in Faculty of Information Science and Technology, University Kebangsaan Malaysia (UKM) to help a team of software specifiers to collaborate in preparing a formal software specification. The team that is involved in preparing the specification consists of two types of member: a drafter who co-ordinates and supervises the process, and a group of rectifiers who write and edit the specifications document, in a setting where the drafter and rectifiers are not located in the same room. Through the Internet, the team members collaborate to write, edit and correct the shared specifications document. For more details see [16,15].

The Z notation is a stylized form of mathematics that is amenable to a standardized syntax and computer processing [3, 14]. In Z formal notation, specification constructs are used to modularize system state and behaviour. Among these constructs, schema is the most important tool to encapsulate specification chunks. Schema construct is used to model both system state (represented by state schemas) and system behaviour (represented by operation schemas) [10].

Z documents can be placed on-line in PostScript format or PDF format, but most Web browsers are not configured to display such documents directly. The easiest way to solve the problem of accessing and displaying an on-line Z document directly within a Web browser window is by using the LAT_EX.

SNL2Z is used during the second stage of the system, that is after the team has specified the state of the software system. Fig. 1 illustrate the first stage process of the system. In brief, the drafter passes the draft document (written in LAT_EX form) of the specification of system state which consists of: basic types and state schemas, as well as propose a list of operation names to the rectifiers. Rectifiers then examine the document for necessary corrections. Rectifiers also study on the proposed operation names for comment. All the views, responses, and comments from rectifiers will be sent to the drafter. The drafter will make changes to the document by considering rectifiers comments. After the team member satisfied with the document, then the system will analysis the document to extract and store all the keys, relations and the structures in the document to be used later on. Now, the system are ready for the second stage of the process that is preparing the specification of the operations. However, before starting the second process, each member will be assigned with at least one operation. Their responsibility are: to prepare the specification of the given operation and to correct the specification according to the comments given by other members.

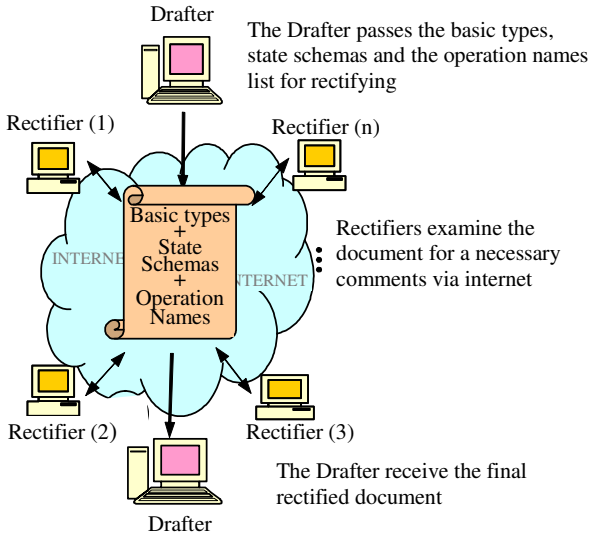

Fig 1: Rectifying process of State Schemas

SNL2Z Pre-processing: Preparing the Specification of Operation: The structure of the operation specification in SNL2Z has been adopted and modified from the table proposed by Bottaci and Jones [2]. They suggest that it is helpful for the specifiers to summarize the decision about the operations in a table form. This will free the specifiers from the details of mathematical formulas at the first round of preparing the formal specification. Normally, one operation of a system consists of several basic operations. So as Z language, one operation might be represented by several schemas. Each schema handle different type of precondition of the operation. The table proposed by Bottaci and Jones is designed in such a way that: a row represents a schema whilst five columns are used to record the schema name, the inputs of the operation, the pre-conditions, the changes to the system state, and the outputs of the operation. In SNL2Z, we add a new column which is used to record the other schemas' name that will be included in the currently specified schema.

SNL2Z offers the facilities for the team member to add a new schema to the list and edit or delete his/her existing one. Besides responsible to the assigned schemas, every member are encouraged to evaluate other operation specifications written by other team

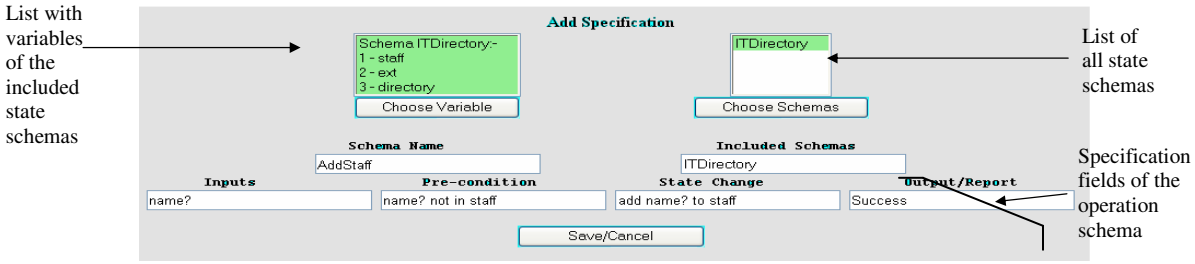

Fig. 2: Preparing a schema in SNL2Z

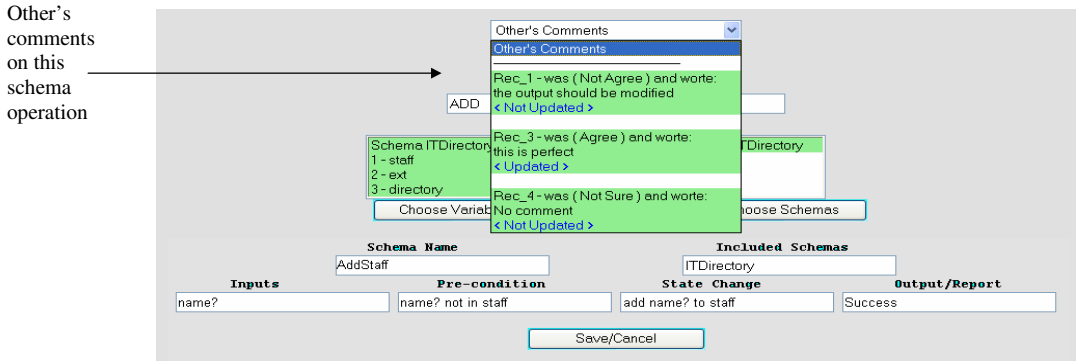

Fig 3: Modifying or Deleting a schema in SNL2Z

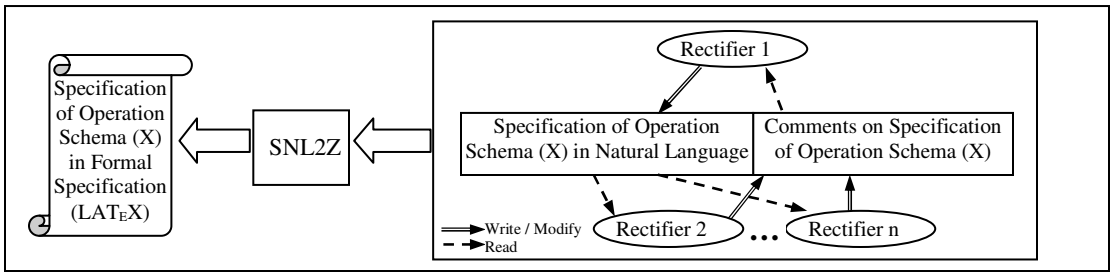

Fig. 4: Overall Processes of the SNL2Z

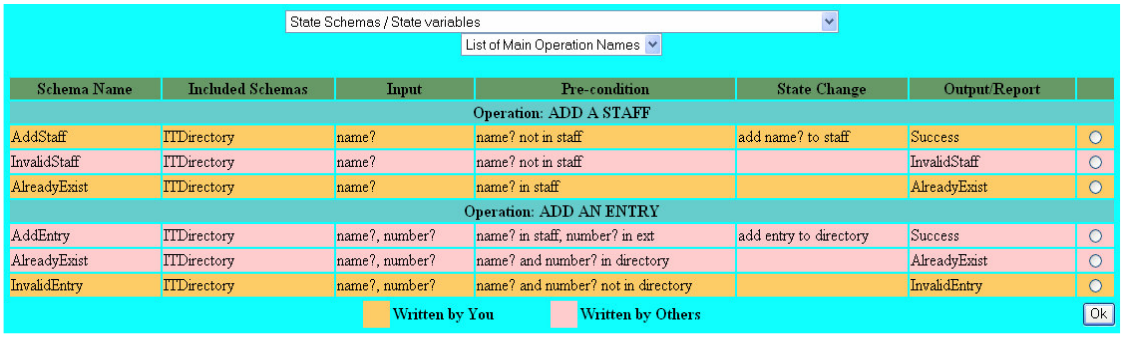

Fig. 5: SNL2Z: Formulating of the Specifications

members. As shown in Fig. 2, in the case of adding a new schema, the system provides the user with a form to write the details of the schema. The form consists of six fields which represents the six column of the table discussed previously. Usually, an operation schema composes of local variables and state variables. Therefore, the system helps the user by providing two pull down menu that provide an information for the state schema names and it's component. Both information are automatically generated by the system.

SNL2Z Pre-processing: Handling Comments: When the user intends to edit the assigned schemas, the system, as shown in Fig. 3, provides the user with currently received comments by clicking a pull up menu at the top of the form. Also, in order not to get redundant comments, the user can also refer to the list of other comments before writing his/hers comments on the other team member's schemas.

Because of rectifying is an ongoing process, the system needs to stamp each comment to record it's validity. For the user perspective, the message <Updated> or <Not Updated> is shown together with the comments. This will help the user maintains and modifies his/her specifications in the light of the updated comments.

SNL2Z: The Mapping: After all the operation are completely specified and satisfied by the members of the team, as Fig. 4 illustrated, the drafter will do the final review, and convert the whole specifications into Z (in LAT_EX form) by using SNL2Z. Through the simple case study, Internal Telephone Directory taken from a book [4], we will show how SNL2Z translates the specification prepared in previous way (controlled natural language) into formal specification statements in LAT_EX form.

A university wants to computerize its internal telephone directory. The database must keep a record of all the people who are currently member of the university (as only they can have telephone extensions). The database must cope with the possibility that one person may be reached at several extensions and also with the possibility that several people might have to share an extension. The system has several main operation names such as adding a number, adding a name, adding an entry, removing a number and several other operations. Each main operation name may has a several operation schemas. As stated previously, one operation might compose of several small operations, therefore for an operation such as adding an entry, it can compose of several small operations; an ideal situation of adding an entry, an operation that handle a situation where the entry is invalid, and lastly a situation where the entry is already exist in the database.

In this case study, a basic type *PERSON* represents a set of person and *PHONE* represents a set of telephone numbers.

```
\begin{zed}
        [ PERSON, PHONE ]
\end{zed}
```

A *REPORT* is a type with specific values and is declared by using free type definitions:

```
\begin{zed}
    REPORT ::= success | invalidStaff | invalidExt |
invalidEntry | alreadyExist | alreadyRemoved
\end{zed}
```

The state space for the system is represented by the following:

staff: to store information about staff members of the university.

ext: to store information about internal telephone numbers.

directory : denotes the relation that exists between people and their internal telephone numbers.

Fig. 5 shows an example of specifications of two main operations that are; adding a staff to the list and adding an entry to the database. Entry means the telephone number of the respective staff. We decide that adding a staff composes of three small operations that are an ideal situation, namely *AddStaff*, *AlreadyExist*, and *InvalidStaff*. For the second operation, as has been discussed above, we decide that it composes of three small operation: *AddEntry*, *AlreadyExist*, *InvalidEntry*. Assume that both operations have been completely specified and satisfied by the team member and the result is shown as in Fig. 5. The following section shows how the translation is made based on the case study.

Construction of the Knowledge: The knowledge is build from the syntactic, semantic, structure and contextual of the operation schema specifications. The knowledge representation has to capture the corresponding structure for the later translation. Because of the space limitations of this article, we only show the translation of one of the operation name's operation schemas that is adding an entry to the database. This operation schema is called *AddEntry*.

Schema Inclusion Translation: Based on the AddEntry specification in Fig. 5, the fifth column (labelled State Change) indicate that the state of the system will change. And in the 2nd column (labelled Included Schema) indicate that only one state schema is included in the operation that is ITDirectory. Based on this two information, SNL2Z will include ΔITDirectory in AddEntry schema. In Z, the inclusion of ΔITDirectory (written in LAT_EX form as \Delta ITDirection) in schema operations introduces all the respective variables and the before-state and after-state invariants. The following table summarizes the action taken by SNL2Z.

Included State Schema Name	State Change	Action in LAT_EX
ITDirection	yes	\Delta ITDirection

Pre-condition Translation: As shown in *Pre-condition* field in the specification, the predicate " *name? in staff"* has three parts, as well as the predicate " *number? in ext* " as follows:

"name? in staff"		
Component		Type
Part_a1	<name?>	Input
Part_a2	<in>	Relation
Part_a3	<staff>	State Variable

"number? in ext"		
Component		Type
Part_b1	<number?>	Input
Part_b2	<in>	Relation
Part_b3	<ext>	State Variable

From the knowledge of state schema, we know that:

State Variable	Basic/Function	Type Name
Part_a3	Power Set	PERSON

State Variable	Basic/Function	Type Name
Part_b3	Power Set	PHONE

From the analysis, (*in* – relation in our library), we obtained the following action. Also we have new knowledge that is the introduction of two input variables (indicated by symbol ?).

Action	In LAT_EX
< Part_a1 > < Part_a2> < Part_a3 >	name? \in staff
< Part_b1 > < Part_b2> < Part_b3 >	number? \in ext

Input Translation: The 3rd column of the specification contains input variables. However, the type of the variables are not specified in the table. Therefore SNL2Z will automatically associate the type of the variables based on the predicate in the above pre-condition.

Input	Basic Type Name	Action in LAT_EX
< Part_a1 >	PERSON	name? : PERSON
< Part_b1 >	PHONE	number? : PHONE

Post-condition Translation: Post-condition translation is based on the column labelled *State Change*. Depending on the preposition, the sentence will be divided into two parts as follows:

"add entry to directory"	
Part_1	Part_2
<add entry>	<directory>

Part 1 of the sentence will be analysed and divided into small component:

<add entry>	
Part_11	Part_12
<add>	<entry>

Part_2 "directory " is identifier of the relation between people and their internal phone number. In the light of this function, the added entry mentioned in Part_12 is analysed as:

<entry>	
Part_12-a	Part_12-b
< name?>	< number?>

Action	< Part_2>' = < Part_2> < ¿ ><Part_12-a> < ↔ > <Part_12-b>
In LAT_EX	directory' = directory \cup \{ name? \mapsto number? \}

Because of the is only one predicate shows the state changes, therefore the other state variables (in *ITDirectory* schema) remain unchanged. SNL2Z explicitly specified that as follows:

Variable	Action in LAT_EX
staff	staff' = staff
ext	ext' = ext

Output Translation: In this case study, the output of the operation is very simple that is *success*. Based on a free type knowledge, SNL2Z knows that *success* is one of a value in *REPORT*. Therefore SNL2Z translates it

by assigned the message to a newly created output variables as follows:

msg! = success

and declare *msg* as a type of *REPORT*.

In the light of the above method, the output of the system "translation" of the operation schemas shown in Fig. 5 into LAT_EX sentences is:

```
\begin{schema}{AddStaff}
 \Delta ITDirectory
  name? : PERSON
  Msg! : REPORT
\where
  name? \notin staff
  staff' = staff \cup \{ name? \}
  ext' = ext
  directory' = directory
  Msg! = Success
\end{schema}

\begin{schema}{InvalidStaff}
 \Xi ITDirectory
  name? : PERSON
  Msg! : REPORT
\where
  name? \notin staff
  Msg! = InvalidStaff
\end{schema}

\begin{schema}{AlreadyExist}
 \Xi ITDirectory
  name? : PERSON
  Msg! : REPORT
\where
  name? \in staff
  Msg! = AlreadyExist
\end{schema}

\begin{schema}{AddEntry}
 \Delta ITDirectory
  name? : PERSON
  number? : PHONE
  Msg! : REPORT
\where
  name? \in staff
  number? \in ext
  name? \mapsto number? \notin directory
  directory' = directory \cup \{ name? \mapsto number? \}
  staff' = staff
  ext' = ext
  Msg! = Success
\end{schema}

\begin{schema}{AlreadyExist}
 \Xi ITDirectory
  name? :  PERSON
  number? :  PHONE
  Msg! : REPORT
\where
  name? \mapsto number? \in directory
  Msg! = AlreadyExist
\end{schema}

\begin{schema}{InvalidEntry}
 \Xi ITDirectory
  name? :  PERSON
  number? :  PHONE
  Msg! : REPORT
\where
  name? \mapsto number? \notin directory
  Msg! = InvalidEntry
\end{schema}
```

CONCLUSION

In this paper, we have presented a technique of writing a structured natural language specification (SNL) used by our SNL2Z system and the method of translating these specifications in which the sentences have implicitly specified parameters into an algebraic specification. The early testing of the system by using various examples taken from [4] and [2] shows that it is capable to translate the input specifications into formal LAT_EX sentences. We are in the process of improving the design of the tool by extending the rules and range of the natural language statement that is accepted by the tool, in the way to lessen the limitations that the tool is suffering from .

REFERENCES

1. Beum-Seuk, Lee and Barrett, R., Bryant, 2002. Automated conversion from requirements documentation to an object-oriented formal specification language. In Proceedings of SAC 2002, Madrid, Spain, ACM.
2. Bottaci, L. and Jones, J., 1995. Formal Specification Using Z A Modelling Approach, International Thomson Publishing.
3. Bowen, J., 1996. Formal Specification and Documentation using Z: A Case Study Approach, International Thomson Computer Press.
4. Diller, A., 1994. Z: An Introduction to Formal Methods, 2nd edn, John Wiley & Sons, West Sussex, UK.
5. Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., and Moreschni, P., 1994. Assisting requirement formalization by means of natural language translation. in Formal Methods in System Design, Vol. 4, 243-263.

6. Fuchs, N., and Schwitter, R., 1996. Attempto Controlled English (ACE), Proc. CLAW 96, 1st Int. Workshop Controlled Language Applications.
7. Holloway, C., 1997. Why Engineers Should Consider Formal Methods, Proc. 16th Ann. Digital Avionics Systems Conf., IEEE Press, Vol.1, 16-22.
8. Ishihara, Y., Seki, H., and Kasami, T., 1992. A Translation Method from Natural Language Specifications into Formal Specifications Using Contextual Dependencies, in: Proceedings of IEEE International Symposium on Requirements Engineering, San Diego, IEEE Computer Society Press, 232-239.
9. Mehandjiska D., and Palac, J., 2002. Towards Bridging Component Specification Technologies, International Conference on Software Engineering, the 20th IASTED International Multi-conference Applied Informatics (AI2002) Austria.
10. Mirian, S. and Mousavi, M., 2002. Making Nondeterminism Explicit in Z, Proceedings of the Iranian Computer Society Annual Conference (CSICC'02), Tehran, Iran, February.
11. Miriyala, K. and Harandi, M., 1991. Automatic Derivation of Formal Software Specifications from Informal Descriptions, IEEE Transaction on Software Engineering, Vol. 17(10), 1126 -1142.
12. Reubenstein, H. and Waters, R., 1991. The Requirement Apprentice: Automated Assistance for Requirements Acquisition, IEEE Transaction on Software Engineering, Vol. 17(3), 226 -240.
13. Shukur, Z., Zin, A., and Ban, A., 2002. M2Z: A Tool for Translating a Natural Language Software Specification into Z. International Conference on Formal Methods and Software Engineering, ICFEM 2002: 406-410.
14. Spivey, J., 1992. The Z Notation: a Reference Manual, 2nd edn, Prentice Hall International Series in Computer Science, London.
15. Sullabi, M. and Sukur, Z., 2006. Model of CSCW for Z Specification Document, in: Proceedings of Business, Law, & Technology (IBT2006), Vol. (2), Edited by Sylvia Mercado Kierkegaard, Denmark, 465-473.
16. Sullabi, M. and Sukur, Z., 2006. Web-based Collaborative Model for Preparing Formal Software Specifications, in: Proceedings of the Eighth International Conference on Information Integration and Web-based Applications & Services (IIWAS2006), Indonesia, 433-441.
17. Vadera, S. and Meziane, F., 1994. From English to Formal Specifications. The Computer Journal Vol. 37(9), 753-761.
18. Wilson, W., 1999. Writing Effective Natural Language Requirements Specifications, Technical report, Naval Research Laboratory.