FORMAL SPECIFICATION ISSUES IN DESIGNING OF WEB-BASED MODEL OF COMPUTER SUPPORTED COOPERATIVE WORK (CSCW)

Mohamed A. Sullabi
The Libyan Academy, Misurata Branch, Misurata,Libya
sullabi@yahoo.com.

ABSTRACT

Having a correct and quality software specification is not easy task. WBCS has been implemented based mainly on the proposed supported cooperative work model and a survey conducted on the existing Web-based collaborative writing tools. The theoretical framework behind the proposed web-based model consists of some issues that bring together strands of the study. These issues being combined here are categorized into two; collaborative issues and formal specification issues. This paper focuses on the formal specification issues carryout in the study.

Keywords: *Formal methods, formal specifications, collaborative writing, computer supported cooperative work, CSCW.*

---

1. Introduction

Progress in the development of formal methods with respect to techniques and tool support is stable, but the acceptance of formal methods is still relatively low. One of the reasons for this problem is the lack of adequate tool [1]. The lack of suitable support tools is a part of the relatively scant use of formal techniques in software development. Why does formal specification need tool support? Z specification particularly, the mathematics on which it is built is typed set theory. A well-defined grammar exists for the language, so it is possible to check every construct. Z also needs special symbols (such as $\subseteq$ and$\in$) and schemas boxes that are not supported by the keyboard. Therefore, we need to decide on several issues in order to have the CSCW system work effectively.

WBCS has been implemented based on a supported cooperative work model [2][3]. This model provides software developers with a web environment that supports them to collaborate and to help them to produce correct software formal specifications.

2. Access and Display

Z uses many mathematical symbols and this represents the main problem for displaying it within HTML [4]. Most of these symbols in general, are used in standard mathematics, but some are specific to Z itself. To make formal specifications more accessible and used in industry, the current research direction is using a graphical and appealing language to encapsulate formal notations [5].

The fact remains that none of the WWW browsers can display Z symbols, schemas, etc. There are two ways to produce a Z specification document, either as WYSIWYG (What You See Is What You Get) based, or by using ASCII-based formatters [4].
The WYSWYG editor allows are to create content with word processor-like formatting.
Annex E of the draft Z standard specifies a proposed ASCII Z Interchange Format for the exchange of Z specifications between tools[6]. However, researchers like to publish formal

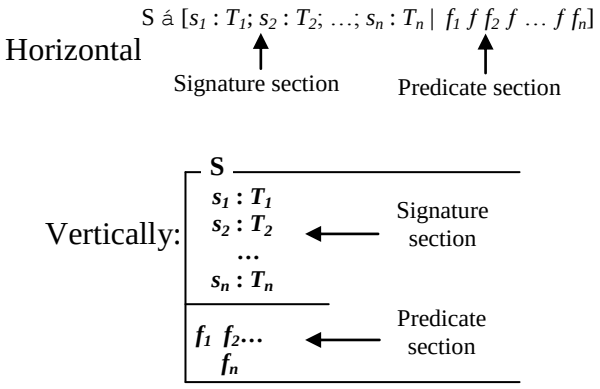
specifications using Z notations on the Web in a convenient form for as wide a readership as possible [7]. Z documents can be placed on-line in PostScript or PDF format. PostScript files can be large and are thus often compressed to reduce loading time and to save file space [4]. It is not easy to configure Web browsers to uncompress such files automatically, particularly on a Windows-based system.

Z specifications can be written using the LAT_EX document preparation system. LAT_EX has become the standard for the interchange of Z documents, and many Z tools can process LAT_EX documents using zed.sty [8], fuzz.sty [8], oz.sty [9]. Another approach is to use the Z Browser plug-in [8]. The Z Browser plug-in, a part of Z specification is displayed, and thus could potentially be used to display Z specifications written using LAT_EX.

3. Structure and Format

Formal specifications describe the properties in a precise way that an information system must have in precise way by using mathematical notation, without unduly constraining, the way in which these properties are achieved [10]. According to [8], the main component in Z is a way of decomposing a specification into small chunks called schemas. By breaking up the specification into schemas, we can present it piece by piece. Each schema can be linked with comments that explain in natural language the significance of the formal mathematics.

A schema is a named unit consisting of a signature that contains the declarations of variables together with a predicate that contains the relationships between the entities in the signature. The schema *S* that shown below can be represented:



The set of s_i(i = 1 … n) are called variables and the T_i(i = 1 ... n) are called types.

3.1 States and operations in Z Specification

In Z, schema is the most important tool to encapsulate specification chunks [11]. Schema construct is used to describe both static and dynamic aspects of a system.

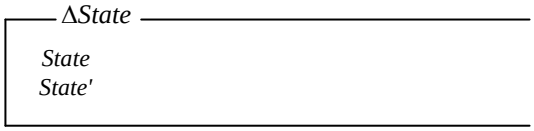
A. Static aspects of a system: (The system state), which represented by state schemas and that include the state aspects of both the states it can occupy and the invariant relationships that are maintained as the system changes from state to state [12]. The state specification of the system describes the different units and entities of the system, all the types of the entities, the restrictions on entity properties, and all the relations between the entities.

To specify a system, Z uses set theory for the definition of entities and their types, and mathematical logic for the description of constraints on the entity properties [13]. State space and initial state schemas describe the static aspects. In a state space schema, the state data of the system is described in the signature part and the predicate give the invariant conditions that maintained by the operations of the system. For the initial state schema, it has the same signature with the state space schema and the initial values of the state data is given in the predicate section.

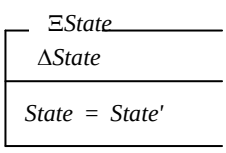
B. Dynamic aspects of a system: (The system behaviour), which represented by operation schemas that include operations that are possible. An operation represents a change of the system state. It may also supply some outputs. The relation between the system's states before and after the operation execution is defined by the operation specification and depend on entries and the initial state, it also defines how the output values. Each operation in the system model is described with the next pattern; entries start state (pre-condition), end state (post-condition), and outputs [13].
The declaration section of any operation schema, four types of variables can be introduced and the relationship among these four variables is given by the predicate. The four types of variable that can be introduced are:

1. Input variables: This variable represents the input of the operation. The input variable should be followed by a '?'.
2. Before-state variable: This variable represents the state data variables before the operation.
3. Output variables: This variable represents the operation output. The output variable is terminated by a '!'.
4. After-state variable: This variable represents the state data variables after the operation and is terminated by a ''.

In the operation schema, the notation Δ*State* represents the schema obtained by including *State* and *State'* in an otherwise empty schema. Inclusion of Δ*State* in the operation schema means that the state has changed with the operation, because it shows the 'before' and 'after' states of the variables and predicates.



In the operations that only access state data variables without changing them, Ξ*State* is included to make visible the 'before' and 'after' states of the variables are not changed by the operation.



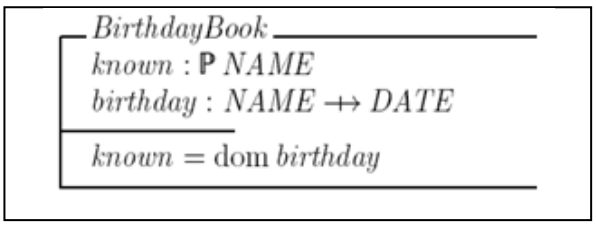
3.2 Birthday Book

In this section, we look at 'Birthday Book' as a small example about the Z specification to show how the issues and ideas mentioned in previous sections work out. The 'Birthday Book' is a well-known example from Chapter I of Spivey's definitive book on Z'. It is a system for recording birthdays [9].

'Birthday Book' deals with people's names and with dates. Therefore, we introduce the set of all names and the set of all dates as basic types of the specification:

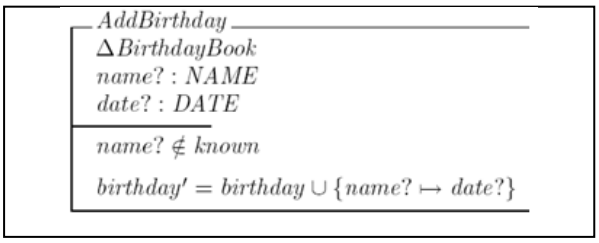
[NAME, DATE]:

The first aspect of the system to describe its state space, and we do this with a schema:

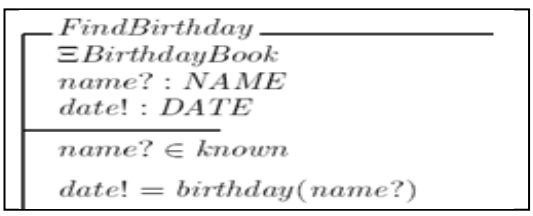


The above schema is describing the state space of a system. This schema consists of two parts: first, the upper part (above the central dividing line), two variables are declared. *known* is the set of names with birthdays recorded. These variables are also known as state variables. Second, the lower part (below the central dividing line), a relationship between the values of the variables is given. *birthday* is a function which, when applied to certain names, gives the birthdays associated with them. The relationship is true in every state of the system. Therefore, the set *known* is the same as the domain of the function *birthday*. This relationship is an *invariant* of the system.

After describing the state schema of the system, we can start on some operations on the system. The first operation is to add a new birthday, and we describe it with the following schema:

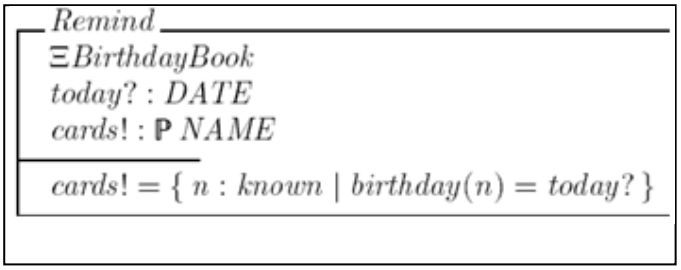


In the AddBirthday operation schema, the declaration Δ BirthdayBook (included schema) shows that the schema is describing a state change, while *name?* and *date?* are representing the input of this operation. The variables *known* and birthday are observations of the state before the change, whereas *known'* and *birthday'* are observations of the state after the change. The first line in the lower part of the schema states that the name to be added must not already be one of those known to the system (pre-condition). The second line says that the birthday function is extended to map the new name to the given date (post-condition).
The second operation might be to find the birthday of a person known to the system. First, we describe the operation with a schema:

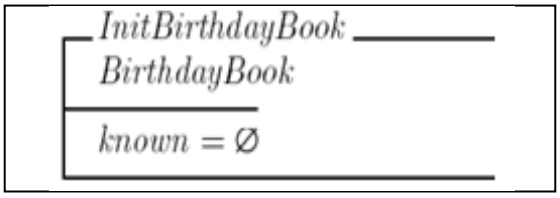


The declaration Ξ BirthdayBook (included schema) indicates the state of the operation does not change. *name?* is representing the input of this operation while *date!* is the output. The values *known'* and *birthday'* of the observations after the operation are equal to their values *known* and *birthday* beforehand. The pre-condition is that the *name?* is one of the names known to the system, and the output *date!* is the value of the birthday function at argument *name?*.

Another operation is to find which people have birthdays on a given date. *today?* is the only input for this operation and *cards!* is the only output.



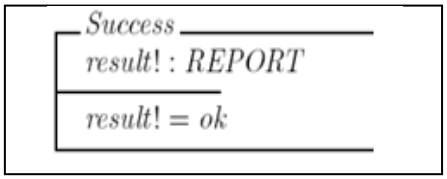
The Ξ BirthdayBook (included schema) indicates that the state does not change. There is no pre-condition. The output *cards!* is specified to be equal to the set of all values *n* drawn from the set known such that the value of the birthday function at *n* is *today?*.
The next operation is the initial state of the system, and a schema specifies it:



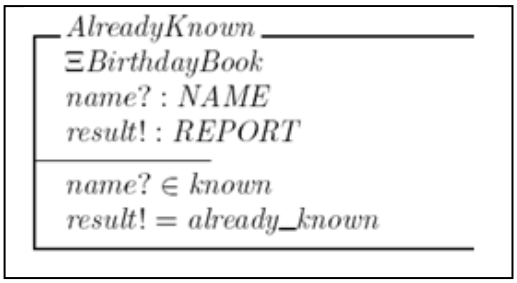
The birthday book is described in this schema, in which both the set *known* and the function *birthday* are empty.Extra output *result!* can be added to each operation on the system. When an operation is successful, this output will take the value ok,
The following free type definition defines *REPORT* to be a set containing exactly these three values:

REPORT ::= ok | already_known | not_known

The schema is defined Success when it specifies that the result should be ok.



In the *AlreadyKnown* operation schema, the report *already_known* should be produced when the input *name?* is already a member of *known*:



4. Document slicing

Slicing is a process used to break up a document into smaller units. By slicing the document, we can optimize various slices of a document differently. Slicing is a very important process for adding interactivity. When documents are broken up into parts, all the logical relationship between those parts is made explicit [14]. Each connected part of a document that can be reused under well-defined conditions is a potential slice and because the slices are expected for reuse, the content of the document should be put and organised in a well-structured way, thus the isolation of reusable units is possible.

LaTeX documents are suitable and perfect candidates because they are well-structured according to a Document Type Definition and they have the layout separated in their corresponding style sheets. Depending on the generalized mark-up language standard, the computer mark-up language comprises a document type definition and the document type definition permits the separating of documents into blocks; and a secure authenticator, for ensuring authenticity of a document and permitting the discarding of a document part without having access to the entire document [15].

LaTeX is a document preparation system that formats text for printing. It makes producing simple and professional documents as easy as creating a text file. LaTeX is more suitable for creating and distributing most kind of documents than any other modern document preparation systems [16]. It is produces superbly typesetted high quality printout documents, especially for scientific text [17]. LaTeX is designed to make it easy for the user to typeset mathematical expressions, and most mathematical texts are edited in LaTeX [14].

According to [18], LaTeX derives the typographical form of the text based on the rules given in the document class. It gives users the ability to structure their documents with a different hierarchal constructs, e.g. chapters, sections, and paragraphs [18]. Thus, it is considered different from other typesetting systems. The most important unit in LATEX text is the paragraph. Paragraph is the typographical form that should reflect one coherent idea. In LaTeX a blank line(s) between two units of text lines means the end of a paragraph and the beginning of a new one; Therefore, a new thought will begin.

As mentioned above, formal specification can be broken up into schemas, and each schema can be presented its own all the comments related to it. This feature can be exploited if the formal specification was created in LaTeX and were integrated with the paragraph feature in LaTeX.

4.1 Birthday Book (LaTeX source)

In this example, the various schemes described above, are created here in the form of latex. Each schema is shown in a separate paragraph.

```
\begin{document}
\begin{zed}
  [NAME, DATE]
\end{zed}

The $BirthdayBook$ schema defines the \emph{state space} of the birthday book system.

\begin{schema}{BirthdayBook}
   known: \power NAME \\
   birthday: NAME \pfun DATE
\where
   known=\dom birthday
\end{schema}

This $InitBirthdayBook$ specifies the initial state of the birthday book system.  It does not say explicitly that $birthday'$ is empty, but that is implicit, because its domain is empty.

\begin{schema}{InitBirthdayBook}
   BirthdayBook'
\where
   known' = \{ \}
\end{schema}

Next, we have several operation schemas to define the normal (non-error) behaviour of the system.

\begin{schema}{AddBirthday}
   \Delta BirthdayBook \\
   name?: NAME \\
   date?: DATE
\where
   name? \notin known
\\
   birthday' = birthday \cup \{name? \mapsto date?\}
\end{schema}
\begin{schema}{FindBirthday}
   \Xi BirthdayBook \\
   name?: NAME \\
   date!: DATE
\where
   name? \in known
\\
   date! = birthday(name?)
\end{schema}
```

```
\begin{schema}{Remind}
   \Xi BirthdayBook \\
   today?: DATE \\
   cards!: \power NAME
\where
   cards! = \{ n: known | birthday(n) = today? \}
\end{schema}
```

Now we strengthen the specification by adding error handling.

```
\begin{zed}
   REPORT ::= ok | already\_known | not\_known
\end{zed}
```

First, we define auxiliary schemas that capture various success and error cases.

```
\begin{schema}{Success}
   result!: REPORT
\where
   result! = ok
\end{schema}
\begin{schema}{AlreadyKnown}
   \Xi BirthdayBook \\
   name?: NAME \\
   result!: REPORT
\where
   name? \in known \\
   result! = already\_known
\end{schema}
\begin{schema}{NotKnown}
   \Xi BirthdayBook \\
   name?: NAME \\
   result!: REPORT
\where
   name? \notin known \\
   result! = not\_known
\end{schema}
\end{document}
```

5. Syntax Checking

Most of the formal specifications have their own syntax checkers. In order to use it, the format of the specification should compatible with the checker. The formal specification language, Z notations is a non-executable but strongly-typed specification language and there is no compiler for Z. However, there are tools to animate, or execute, subsets of Z. ZTC is one of the tools that uses as a type-checker for Z specification [7]. It determines if there are any syntactical or typing errors in the specifications.

LATEX mark-up is used to specify components in Z notation and it maps to/from unicode representation [19]. It is based on a 7-bit ASCII is suitable for processing by tools. ZTC

accepts most of the LATEX source files as input. A Z specification consists of informal text and formal specifications in input files. ZTC will type check the formal specifications and ignore the informal text. However, some rules must be observed in order to type-check the input files.

6. Translation

Normally, operations of any software system are discussed in terms of three parts; the input, the process and the output. In addition, for most formal specifications, the process is observed in term of two states which are the state before the operation which is also known as the pre-condition, and the state after the operation, which is known as the post-condition.

Formal specifications that are based on mathematics have been considered to be more effective in representing software specifications [20]. Most software developers find that writing mathematical statements is not easy, because they are not familiar with mathematical notation. By having this formation, providing software developers with a tool that can aid in translating natural language statements into mathematical statements could be the a possible solution to this problem [20]. A few such systems have been developed (such as the M2Z system [20], SNL2Z system [3]). These systems were designed to translate natural language statements into statements in formal notation.

7. Conclusion

This paper presents and discuses the different formal specification issues. The formal specification issues represent a part of the theoretical framework behind the proposing of a web-based model in order to have the CSCW system work effectively to enable a group of people work together to prepare and write formal specification documents.

Reference

1- Knight, J., Hanks, K., & Travis, S. (2001). Tool Support for Production Use of Formal Techniques. Proc: International Symposium on Software Reliability Engineering, pp. 242-253.

2- Mohamed A. Sullabi and Zarina Shukur. (2006). Model of CSCW for Z Specifications Document. *International Conference on Business, Law and Technology*. Copenhagen Denmark, 5-7.

3- Mohamed A. Sullabi and Zarina Shukur.(2008). SNL2Z: Tool for Translating an Informal Structured Software Specification into Formal Specification, *American Journal of Applied Sciences,* 5(4): 378-384.

4- Bowen, J., & Chippington D. (1998). Z on the Web using Java. ZUM'98:11th International Conference of Z Users, *Lecture Notes in Computer Science, Springer-Verlag*, Vol. 1493, Berlin, pp. 66-80.

5- Borges, R. M. & Mota, A. C. (2007). Integrating UML and formal methods. *Electronic Notes in Theoretical Computer Science*. 184, pp. 97-112

6- Z Standard. (2005). *Annex E – Interchange Format*. Draft 1.2. http://web.comlab.ox.ac.uk/oucl/work/andrew.martin/zstandards/cd.html.

7- Mikusiak, L., Adamy, M., & Seidmann, T. (1997). Publishing Formal Specifications in Z notation on theWWW. volume 1214 of *Lecture Notes in Computer Science*, Lille, France, Springer-Verlag, Berlin, pp. 871–874.

8- Spivey, J. M. (1992). *The fuzz Manual*. 2nd ed. Computing Science Consultancy, 2 Willow Close, Garsington, Oxford, UK.

9- King, P. (1991). *Printing Z and Object-Z LaTeX documents*. Technical Report, Software Verication Research Centre, Department of Computer Science, The University of Queensland, Australia

10- Kaur, N., (2005), Retrieving Best Component From Reusable Repository. Master thesis, Thapar Institute of Engineering and Technology, Deemed University, Patiala (Punjab).

11- Mirian, S. H. & Mousavi, M. (2002). Making No determinism Explicit in Z, *Proceedings of the Iranian Computer Society Annual Conference (CSICC'02)*, Tehran, Iran, February.

12- Pierantonio, A. (2003). Reification of Algebraic Specifications by means of Z Specifications. Dipartimenti di Informatica, Università di L'Aquila, TR 010/2003. http://www.di.univaq.it/di/pub-dl.php?id=327.

13- Bogdan, C. (2002). Visual Representation of Formal Software Specification. Analele Universitatii Ovidius, *Constanta, seria Matematica, Vol. 10(1)*, pp. 35–48.

14- Dahn, B.I., Armbruster, M., Furbach, U., & Schwabe, G. (2001). Slicing Books – The Authors' Perspective, in R. Bromme, E. Stahl (eds.): *Writing Hypertext and Learning: Conceptual and Empirical Approaches*, Pergamon Press.

15- Anderson, M., Jaffe, F., Hibbert, C., Kravitz, J., Chang, S., Palmer, E., & Virkki, J. (2001). Method and system for processing electronic documents. US Patent Issued. http://www.patentstorm.us/patents/6209095-fulltext.html.

16- Akhmechet, S. (2006). What is LaTeX and Why You Should Care. http://www.defmacro.org/ramblings/latex.html.

17- Klum, B. (2006). LaTeX backend for Asciidoc. http://www.methods.co.nz/asciidoc/latex-backend.html.

18- Oetiker, T., Partl, H., Hyna, I., & Schlegl, E. (2008). The Not So Short Introduction to LATEX2. Version 4.24, http://tobi.oetiker.ch/lshort/lshort.pdf.

19- Bhatia, R.K., Dave, M., & Joshi, R.C. (2007). A Hybrid Technique for Searching a Reusable Component from Software Libraries. *DESIDOC Bulletin of Information Technology*, Vol. 27(5), pp. 27-34.

20- Shukur, Z., Zin, A., & Ban, A. (2002). M2Z: A Tool for Translating a Natural Language Software Specification into Z. *International Conference on Formal Methods and Software Engineering. (ICFEM)*. pp. 406-410.