A Study of Twelve Specifications of the Library Problem

Jeannette M. Wing

26 July 1987
CMU-CS-87-142

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520. Additional support was provided in part by the National Science Foundation under grant DMC-8519254.

A Study of Twelve Specifications of the Library Problem

Jeannette M. Wing

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-3068
wing@k.cs.cmu.edu

26 July 1987

Abstract

Twelve workshop papers [25] include an informal or formal specification of Kemmerer's library problem [28]. The specifications range from being knowledge-based to logic-based to Prolog-based. Though the statement of the informal requirements is short and "simple," twelve different approaches led to twelve different specifications. All twelve, however, address many of the same ambiguities and incompletenesses, which we describe in detail, present in the library problem. We conclude that for a given set of informal requirements, injecting domain knowledge helps to add reality and complexity to it, and formal techniques help to identify its deficiencies and clarify its imprecisions.

A Study of Twelve Specifications of the Library Problem

Jeannette M. Wing

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

26 July 1987

1 Introduction

The purpose of this paper is to summarize and compare twelve different papers that address the same set of informal requirements—Kemmerer's library problem—as assigned to participants of the Fourth International Software Specification and Design Workshop held in Monterey, California in April 1987. Some of the specifications contained in the papers are formal; some, informal. Some authors followed a particular specification method or used a particular specification language. This paper focuses on what each of the methods or languages elucidates of the problem statement.

1.1 History of problem

Richard Kemmerer first posed the library problem in 1981 as part of his formal specifications class at the University of California at Santa Barbara. He introduced the problem to Susan Gerhart in 1982 when they team taught an extension class at the University of California at Los Angeles. In 1984 Gerhart used the problem as a focal point of discussion for the "tools" group during the second meeting of this workshop (under a different title) that took place in Orlando, Florida [1]. In 1985 Kemmerer's *IEEE Transactions on Software Engineering* paper on testing formal specifications included an Ina Jo specification of the problem [12]. Finally, in 1986 in the Call for Papers [28], the organizers of the fourth workshop encouraged authors to address a set of four problems, one of which was the library problem, in their position papers. Of the final batch of papers published in the proceedings of the workshop [25], twelve addressed the library problem. This paper discusses only those twelve, although other library specifications have been written, in particular Kemmerer's in Ina Jo, Gerhart's in Affirm [20] (unpublished), and King and Sorensen's in Z [31] (unpublished).

1.2 Informal Requirements

What follows is the statement of the library problem as it appears in the Call for Papers (K-Call) [28].

> Consider a small library database with the following transactions:
>
> 1. Checkout a copy of a book / Return a copy of a book;
> 2. Add a copy of a book to the library / Remove a copy of a book from the library;
> 3. Get the list of books by a particular author or in a particular subject area;
> 4. Find out the list of books currently checked out by a particular borrower;
> 5. Find out what borrower last checked out a particular copy of a book.
>
> There are two types of users: staff users and ordinary borrowers. Transactions 1, 2, 4, and 5 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find out the list of books currently borrowed by themselves. The database must also satisfy the following constraints:
>
> 1. All copies in the library must be available for checkout or be checked out.
> 2. No copy of the book may be both available and checked out at the same time.

Authors	***Formality***	***Lifecycle Phase***	***Key Idea***	***Language/ Project***
Kerth	informal	requirements-informal spec.	Structured Analysis and human interface	
Fickas	AI	requirements-formal spec.	usage scenarios	KATE
Rich, Waters, Reubenstein	AI	requirements-formal spec.	Requirements Apprentice	Programmer's Apprentice
Lubars	AI	spec.-design	design schema	
Dubois, van Lamsweerde	logic	requirements-formal spec.	meta-specification	
Wing	logic	requirements-formal spec.	benefits of formalism	Larch
Rudnicki	logic	analysis	testing and proving	Ina Jo
Yue	logic	analysis	sufficient-completeness and pertinence	GIST
Levy, Piganiol, Souquieres	logic and executable	requirements-formal spec.	multiple methods	SASCO
Terwilliger, Campbell	logic and executable	design-code	Anna and Prolog	PLEASE
Lee, Sluizer	executable	requirements-formal spec.	models of behavior	SXL
Rueher	executable	design-code	rapid prototyping and graphical Prolog	Prolog

Figure 1: Summary of the Papers

2.1.1 Formality

A specification is *formal* if it has a precise and unambiguous semantics. It is *informal* otherwise. A precise and unambiguous semantics is given by mathematics, usually in the form of a set of definitions, a set of logical formulae, or an abstract model. These three approaches to giving formal semantics roughly correspond to the denotational, axiomatic, and operational approaches of giving semantics to programs. If an abstract model of the specification is a machine-executable interpreter, e.g., a Prolog interpreter, we consider the (formal) specification to be *executable.*[1]Usually a formal specification is written in a concrete language, which is like adding "syntactic sugar" on top of mathematics. If this language has a precisely defined syntax and semantics, then a specification written in it is formal.

Informality is introduced by the reliance on English and/or uninterpreted diagrams in writing and giving the semantics of the specification. Of the four informal specifications presented in the papers, three use domain knowledge (indicated as "AI" in the table). All four are presented with tool support in mind so that some amount of machine processing, e.g., pattern-matching on keywords, could be performed on the

[1]"Less obvious is that a specification in logic may also be executable, although its 'execution' may involve the search of a solution space, and hence may be infeasible for examples of nontrivial dimensions [1]." In this paper, we do not classify such specifications as executable.

functions are selected; and (3) a textual description of the operational behavior of a user interface, including what normal or undesirable events occur when keystrokes are entered or mouse buttons are clicked.

Fickas [7] describes the components of a system, KATE, used to automate the process of transforming informal requirements into a formal specification. The paper shows how to criticize refinements of informally stated requirements through the use of domain knowledge, *usage scenarios*, and *intermediate summaries*. Fickas also relies on human experts, e.g., professional librarians, to aid in critiquing a specification and to make KATE smarter by enlarging its knowledge database.

In the context of the ongoing Programmer's Apprentice research project, **Rich, Waters, and Reubenstein** [26] discuss using a *Requirements Apprentice* (RA) to assist a user in converting an initial informal requirement into a formal specification. The RA relies on simple deductive methods applied to extensive domain knowledge represented as *cliches*. For example, the library example uses cliches about repositories (where objects like books are stored), information systems (programs for storing and reporting data), and tracking systems (programs for keeping track of the current state of objects like the physical library repository).

Lubars [19] addresses design reusability by defining abstract graph representations of designs and then instantiating them to yield specific designs. In particular, he instantiates an inventory-control schema with domain knowledge (library databases) to model the library.

Dubois and van Lamsweerde [5] discuss the meta-issue of how to specify the process of specifying. Based on a dual object-model and operation-model of specification, they suggest two meta-models in which the process of specification is made explicit: the *process-model* captures the steps used by a specifier while constructing a specification; and the *method-model* captures the control information and rationale used for the steps taken, yielding an overall specification strategy.

Wing [33] demonstrates the benefits of formal specifications by identifying numerous problems with informal statements of requirements such as in the library example. She presents a specification of the library using Larch, which combines algebraic and predicative specification techniques into one framework.

Rudnicki [27] argues for detecting errors in specifications by both testing and proving properties about them *by hand* (his emphasis). Each test case is related to some property that might best be simulated through symbolic execution of the specification or might more easily be provable from the specification itself. His specification is based on Kemmerer's Ina Jo specification.

Yue [34] formally defines two properties, *sufficient completeness* and *pertinence*, both with respect to a set of *goals*. They capture the notion of whether a specification contains enough, but no more than necessary, information to achieve the goals. He discusses how to analyze a specification in terms of these properties, using a GIST specification of the library problem as an example.

Like Dubois and van Lamsweerde, **Levy, Piganiol, and Souquieres** [16] are interested in the specification process itself. They briefly describe SASCO, a system for supporting the evolution of an informal description into a formal specification. Through operations like refinement, enrichment, reuse, and abstraction, SASCO supports a multi-method approach to specification, where *method* means a

Authors	*Orientation*	*Modularity*	*Readability*
Kerth	operation		graphics
Fickas	operation		
Rich, Waters, Reubenstein	operation		
Lubars	operation		schemata
Dubois, van Lamsweerde	both	yes	
Wing	data	yes	statecharts
Rudnicki	operation		
Yue	operation		
Levy, Piganiol, Souquieres	operation	yes	
Terwilliger, Campbell	operation	yes	
Lee, Sluizer	operation		
Rueher	data		graphics

Figure 2: Summary of the Specifications

and Rueher focus on data. Dubois and van Lamsweerde, and Wing specify the semantics of data in terms of algebraic abstract data types. Since Rueher's ultimate concern is to transform a high-level design into code, he generates from his Prolog specification a high-level design consisting of Ada package definitions for types; data semantics are still in terms of Prolog predicates. The other specifications focus more on specifying the effects of the operations, and not the properties of the data. Note that this orientation could be attributed to the operational (transactional) presentation of K-Call, and not on the particular orientation of the authors' specification method.

In fact, the orientation of a specification *method* may be different from the orientation of a resulting specification. For example, contrary to what the table may imply, two other specification methods could be used in a way to generate specifications that are just data-oriented or both operation- and data-oriented. As previously mentioned, Levy et al. support a multi-method approach; the user is free to choose one or design his or her own. Also, although Terwilliger and Campbell's actual specification focuses only on the operations of interest, PLEASE's use of Anna, and hence Ada, could be used in a data-oriented manner.

For large, complex, and realistic systems, favoring one orientation over the other is likely to be too simplistic. A dual specification method (Dubois and van Lamsweerde), or more generally, a multi-methods approach (Levy et al.), would be more appropriate. Giving a formal meaning to a specification that results from a mix of methods remains a challenge and is of current interest in the research community.

Authors	***library***	***user***	***book***	***available***	***last checked out***
Kerth	yes	= 2	book ≠ copy		
Fickas		> 2		redundant	last = current
Rich, Waters, Reubenstein	yes		book ≠ copy		
Lubars	yes				last = current
Dubois, van Lamsweerde	yes	> 2	book ≠ copy	⇒ in library	
Wing		> 2	book ≠ copy	⇒ in library	last = current
Rudnicki		= 2	book ≠ copy	redundant	
Yue	yes		book ≠ copy		
Levy, Piganiol, Souquieres	yes	= 2	book ≠ copy	⇒ in library	
Terwilliger, Campbell		= 2	book = copy	⇒ in library	last = current
Lee, Sluizer	yes	= 2	book ≠ copy	⇒ in library	last ≠ current
Rueher	yes	= 2	book = copy	⇒ in library	last ≠ current

Figure 3: Ambiguities

3.1.1 What is a library?

The table indicates with a "yes" which authors explicitly distinguish between a library *database* and the entire library *system*. A library database includes records of books (e.g., author and title, and perhaps copy number) and records of users (e.g., name and status). Transactions are performed on the database explicitly by some implicit set of users. An entire library system includes not only a library database (also called "inventory," "repository," or "card-catalog"), but also the people using the library, the books on the shelves, and the transactions involving all these objects.

The distinction between a library database and a library system arises from deciding what of the concept of the library is part of the specificand (the library) and what is part of the specificand's *environment* [10]. If the library is just the "database," then the environment must include the people who have access to the database, i.e., the people who perform the transactions on it. If the library is the entire "system," including the people (and books), then the environment of the database becomes a part of the library system itself; the library's environment would then be the rest of the university (if a university library), or the other public services (if a public town library). Though some of the authors discuss this ambiguity, none of the specifications makes clear the distinction between the specificand and its environment. In fact, the GIST specification language [6], used by Yue, models "closed-systems" which by definition both a system and its immediate environment.

Another possibility exists (Wing): K-Call lacks Kemmerer's original fourth constraint which states that a user may have only one copy of a book checked out at once. This constraint is consistent with the informal statement of Transaction 4 in K-Call since now it would be clear that the transaction need not be concerned with returning copy numbers as well as book identities (author and title).

3.1.4 What does "available" mean?
Two papers (Rudnicki, Fickas) consider Constraints 1 and 2 as stating the same thing (indicated as "redundant" in the table). A book is either available or checked out; it cannot be both.

Other authors (see "$\Rightarrow$ in library" in the table), however, distinguish between not only whether a book is available or not, but whether it is even associated with the library at all (it could be in a bookstore or privately owned). Thus, if a book is available (or checked out), it must be associated with the library. There may be books not associated with the library that are neither available nor checked out. This interpretation is consistent with Constraints 1 and 2, yet do not cause one to be a restatement of the other.

As an aside, Fickas notes that besides being available or checked out, there are other states, such as being lost or stolen, that a library book may be in.

3.1.5 What does "last checked out" mean?
Fickas's professional librarian notes that of the books that are on the shelves it is not interesting to find out who last borrowed them so "last" must mean "currently" (last = current). Three others authors also equate the notion of "last checked out" with "currently checked out," although a distinction is implied by the difference in wording between Transactions 4 and 5. Equating the notions means that Transaction 5 returns a current borrower.

Lee and Sluizer, however, interpret "last checked out" to be different from "currently checked out" (last $\neq$ current) by making the set of books currently checked out a subset of books that are last checked out. If someone currently has a book checked out, that person must also be the last person to have checked out the book. Transaction 5 returns either the current borrower if the book is checked out, or the last borrower if the book is not checked out.

Rueher also interprets "last checked out" to be different from "currently checked out." His Transaction 5, however, faithfully reflects the informal specification and returns the last borrower of only available books (and no current borrowers).

3.2 Incompletenesses
There are many kinds of incompletenesses in the informal requirements. The table in Figure 4 summarizes the six different incompletenesses we discuss in detail below. We will not discuss undefined terms like "title" or "subject," which could also be classified as a kind of incompleteness.

3.2.1 Initialization
As the table indicates, three papers explicitly characterize what properties must hold in the initial state of the system. In the state-transition model used by Lee and Sluizer and by Rudnicki, properties that must hold in the initial state are explicitly written in the specification. Lee and Sluizer specify that initially there exists a normal user, a staff user, an available book, and the book's entry in a card catalog (the library

The following two operations are strictly not necessary. Without the first, however, there would be no need to distinguish between two types of users, if the operation of adding a staff user is included as above. Including the second makes the set of transactions more closely reflect reality, and more symmetric, if adding users are included.

- Add a regular user (Wing).
- Remove a user (Wing).

3.2.3 Error Handling

The informal requirements do not state what should happen if an error or undesired situation is encountered, e.g., trying to return a book that has not been checked out. A specification could either strengthen the precondition of a transaction in order to prevent the undesired situation from arising or it could strengthen the postcondition by explicitly specifying behaviors for the exceptional cases. In strengthening the postcondition, one could use a single "catchall" error or treat each exceptional case individually. The table indicates whether the specification handles errors either by strengthening only the precondition ("pre"), using a single catchall error ("error"), tuning error handling for different situations ("signals"), or some combination ("pre + X") of preconditions and error handling.

Some of the undesired situations that the authors address include:[3]

- *Checkout:* Make sure the book being checked out is not already checked out (Kerth, Lubars, Wing, Terwilliger and Campbell, Lee and Sluizer, Rueher). Make sure the book is part of the set of library books (Rich et al., Wing, Terwilliger and Campbell, Lee and Sluizer).
- *Return:* Make sure the book is checked out by the user returning the book. (Wing, Rudnicki, Rueher). Here, one could argue that this is not necessarily an undesired situation since it may not matter who returns a book, just as long as it is returned.
- *Add book:* Make sure the book does not exist (Kerth, Yue, Rueher). If a distinction is made between a book and a copy then adding a copy should check to see if the book exists (Levy et al., Lee and Sluizer) or explicitly state that a new entity is added (Wing, Lee and Sluizer)
- *Remove book:* Make sure the book exists (Kerth) or is available, implying that it exists (Wing, Terwilliger and Campbell, Lee and Sluizer, Rueher).

Finally, for completeness, specifications should treat type errors. If an argument or result of an operation is of the wrong type, then the specification contains an inconsistency. All of the methods do implicit type-checking through the declarations of the types of an operation's arguments and results. This type information is implicitly conjoined to the pre- and postconditions of individual operations or defined in the underlying semantics by using predicate logic with typed variables.

3.2.4 Missing Constraints

In an informal sense, all authors added more "constraints" to K-Call, simply by (informally) elaborating the requirements, or making them unambiguous and more precise. The AI papers added domain-specific constraints, for instance by introducing knowledge about information retrieval systems for which a library is a special instance.

In a more formal sense, however, a "constraint " can be defined to be a state invariant to be maintained

[3]Recall that not all authors gave specifications of all transactions and for informal specifications, the authors may have only discussed the problem in text.

process of testing his specification. Lee and Sluizer actually provide a limit (= 5) in their specification for the purposes of making their specification concrete and hence "fully" executable. Note that Constraint 3 says "books," not "copies," which reraises the question of "what is the difference between a book and a copy?"

Fickas discusses at length the implication of removing the borrowing limit constraint. For example, he notes that placing a borrowing limit may prevent a user who needs more books than allowed from achieving his goal. He also questions what "small" means. "Small library database" (K-Call) could mean a small-library database, a small library-database, a small-time system, or a simple problem involving a library database.

Three papers discuss progress as a desired liveness property of the library system. Lubars assumes that the class of inventory system he instantiates to get a library system is one for which goods (books) are returned as opposed to one for which they are not (like food in restaurants).

Yue explores the constraint on borrowers even further. He argues that progress could be impeded if either of the following problematic situations arises:

1. A user wants to check out a book and has a maximum number already. He is forced to return a book first.
2. A user wants to check out a book, but it is not available because someone else has checked it out.

To solve the second, the library could simply keep adding books—an unrealistic solution. Thus, Yue solves both problems at once by adding the constraint that a borrower may not keep a book forever, later refining "forever" to be "a pre-defined period of time."

Dubois and van Lamsweerde do not discuss liveness explicitly but introduce enough formalism, in particular a sequence of times, so that they could characterize liveness properties. For instance, they use these times to determine whether a book has been returned by checking to see that each returned date associated with the book is less than the last checkout date.[4]

4 Discussion

We chose to compare the specifications according to how they address problems of the library example in order to illustrate the imprecision of natural language specifications and how twelve different approaches to the same set of informal requirements reveal many of the same problems. We admit that the revelation of a problem may be due to the authors' cleverness, and not to the particular approach they use; however, each of the approaches used undoubtedly helps to prod each of the authors into considering certain aspects of the informal requirements, and perhaps not others.

Our comparison highlights for the reader what issues should be addressed in refining an informal set of requirements and how these issues are resolved in different specification approaches. Thus, for each of the twelve cases, the interesting result of the specification exercise is not the specification itself, but the insight gained about the specificand. This insight is evidence that benefits can be gained by a systematic application of formal, or even informal, specification methods.

[4]This is an informal paraphrase of what they actually specify.

both the similarity among the state-transition models and that among the logic-based ones.[5] The popular, and generally accepted, technique for specifying an operation's effects is to use pre- and postconditions. There is less of an agreement on how to specify data. Algebraic and set-theoretic approaches are common, but the dominating approach of the twelve presented is model-oriented where one might model a set of books by a list of books, and a book by a record of three components (title, author, copy number). We can confidently conclude that existing formal specification techniques can be used:

- to identify many, but not all, deficiencies in a set of informally stated requirements;
- to handle "simple" and "small" problems; and
- to specify the functional behavior of sequential systems.

Except for perhaps the third, these are not new conclusions, but they are reassuring. Two broad challenges remain and currently are of interest to those active in formal specifications: (1) demonstrating that existing techniques scale up or scaling up the techniques themselves; and (2) specifying non-functional behavior such as concurrency, reliability, performance, and human factors.

Finally, we conclude with a reminder to the authors: It is the responsibility of each of the advocates of a particular specification method to tell potential users not only what the method is good for, but also what it is not. Each method is often intended for use on a specific class of applications, e.g., databases, and for specifying a specific class of properties, e.g., functionality. If so, the method should not be expected to be suitable for classes of applications and properties outside of their intended ones. However, students of a particular specification method should not be expected to guess what those suitable classes are; teachers must state the limitations of their methods.

Acknowledgments

I thank Dick Kemmerer and Susan Gerhart for introducing the library problem, keeping it alive, and relaying its history to me. I also thank Mehdi Harandi for encouraging me to write this paper based on the oral summary I gave at the fourth workshop. Finally, I thank all the authors of the twelve papers for their timely response to my appeal for comments on and corrections to an early draft of this paper.

[5]Perhaps we should not have been surprised since all formal techniques are based on some common set of mathematical concepts.

[14] S. Lee and S. Sluizer.
private communication.
1987.

[15] S. Lee and S. Sluizer.
SXL: An Executable Specification Language.
In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 231-235. Computer Society Press of the IEEE, April, 1987.

[16] N. Levy, A. Piganiol and J. Souquieres.
Specifying With SACSO.
In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 236-241. Computer Society Press of the IEEE, April, 1987.

[17] B.H. Liskov, et al.
Lecture Notes in Computer Science. Volume 114: *CLU Reference Manual.*
Springer-Verlag, 1981.

[18] B.H. Liskov and J.V. Guttag.
Abstraction and Specification in Program Development.
The MIT Press, 1986.

[19] M.D. Lubars.
Schematic Techniques for High Level Support of Software Specification and Design.
In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 68-75. Computer Society Press of the IEEE, April, 1987.

[20] D.R. Musser.
Abstract Data Type Specification in the Affirm System.
IEEE Transactions on Software Engineering 6(1):24-32, January, 1980.

[21] C.H. Papadimitriou.
The serializability of concurrent database updates.
Journal of the ACM 26(4):631-653, October, 1979.

[22] D.L. Parnas.
On the Criteria to be Used in Decomposing Systems into Modules.
Communications of the ACM 15(12):1053-1058, December, 1972.

[23] D.L. Parnas.
The Use of Precise Specifications in the Development of Software.
In *Information Processing 77*, pages 861-867. IFIP, North-Holland, 1977.

[24] D.L. Parnas.
Software Aspects of Strategic Defense Systems.
American Scientist :432-440, September-October, 1985.

[25] IEEE Computer Society.
Proceedings of the 4th International Workshop on Software Specification and Design, IEEE Computer Society Press, 1987.

[26] C. Rich, R.C. Waters, and H.B. Reubenstein.
Toward a Requirements Apprentice.
In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 79-86. Computer Society Press of the IEEE, April, 1987.

[27] P. Rudnicki.
What Should Be Proved And Tested Symbolically In Formal Specifications?
In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 190-195. Computer Society Press of the IEEE, April, 1987.