SOFTWARE VERIFICATION RESEARCH CENTER

DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072

Australia

TECHNICAL REPORT

No. 99-47

Visualization of Formal Specifications

Soon-Kyeong Kim and David Carrington

December 1999

Phone: +61 7 3365 1204

Fax: +61 7 3365 1999

Visualization of Formal Specifications

Soon-Kyeong Kim
Department of Computer Science and Electrical Engineering
The University of Queensland, Brisbane, Australia
+61 7 3365 1204
email: **soon@csee.uq.edu.au**

David Carrington
Department of Computer Science and Electrical Engineering
The University of Queensland, Brisbane, Australia
+61 7 3365 3310
email: **davec@csee.uq.edu.au**

ABSTRACT
Formal specification techniques provide precise and analyzable software specifications. However, formal notations provided by most formal specification techniques are not easy to use and understand for most people. Our approach counters this difficulty by visualizing formal specifications. In this paper, we use various diagrams to visualize a Z specification. In our work both static and dynamic aspects of formal specifications including complex constraints are included in the visualization scope. This is in contrast to other work that develops visual representations of formal specifications, without visualizing the complex constraints in the mathematical notation. Our work also supports a mechanical translation process from Z specifications to diagrams by providing transformation rules between the two representations. Representing a Z specification using various diagrams should enhance the readability and the understandability of the Z specification, and should make Z specifications more understandable for non-specialists.

Keywords

Formal Specifications, UML, Z notation, Visualization

1 INTRODUCTION

For developing software systems, the main objective of a requirement specification is to express user requirements precisely and understandably. During the software development process, the main role of a requirements specification is for communication between participants. Therefore, a requirements specification should be unambiguous and understandable for all potential users who may need to refer to it, such as clients, designers, programmers, or testers. The precision and the understandability of a requirements specification depend on the notation used in expressing the specification.

In practice, informal specification techniques based on graphical notations such as various Object-Oriented (OO) modeling techniques (OMT, Booch, and the UML [1]) are often used to specify user requirements. Their visually appealing and simple notations, using boxes, circles, and lines, produce specifications that are easy to use and understand. However, the lack of a precise semantic basis for the notations used in most informal specification techniques often produces ambiguous specifications, which can cause misinterpretation of user requirements. Also the simplicity of the notations limits the precise expression of user requirements.

In contrast, formal specification techniques based on mathematical notations provide a precise and analyzable specification, and allow proof of properties of the specification before implementation. Over the last twenty years, formal methods have shown their ability to improve the quality of software systems by reducing possible faults in various ways [3, 4, 15]. Despite their potential, formal methods are rarely adopted in industry. There are numerous reasons proposed for this. The most often stated limitations of formal specification techniques are the difficulties in using and understanding formal notations [5, 7, 8, 18].

In this paper, we address this difficulty by combining graphical and formal specification techniques. In our work graphical notations are used as an alternative representation of Z [17] specifications. This is in contrast to some other work [6, 9] that combines graphical and formal specification techniques in an informal method’s framework, either to give a formal semantics for graphical notations or to formalize informal models. The aim of our work is to increase the readability and the understandability of Z specifications by visualizing them, and to provide a systematic translation process between the two representations. There is similar work [5, 7, 11, 16, 18] that develops or uses graphical notations to represent formal specifications. Most of these studies however limit the scope of their visualization to the structural aspects of a formal specification and pay little attention to visualizing the complex constraints in the mathematical notation. Our work intends to visualize not only the system structure specified in a formal specification but also the complex constraints. The reason for including complex constraints in our visualization scope is that we believe one of the greatest difficulties in understanding a formal specification for non-mathematicians is to understand the complex constraints. Without understanding these constraints precisely, a complete understanding of a formal specification cannot be achieved.

In this paper, we use three different diagrams for representing different aspects of a Z specification. First, we use the UML class diagram to visualize the system structure specified in the state schema. Complex constraints that cannot be expressed by the UML notation alone are described with a separate notation, the Constraint diagram [13]. To visualize dynamic aspects of a Z specification, none of the well-known modeling diagrams are ideal for representing Z operation specifications. For this reason, we use a recent notation, Contract box [10, 14] (which is related to the Constraint diagram notation) to express the pre and post-state of each Z operation diagrammatically. By describing different aspects of a Z specification with the most appropriate diagrams, we achieve a complete visualization of the Z specification. This should improve the readability and the understandability of a Z specification, and should make Z specifications more accessible for non-mathematicians.

The structure of the rest of this paper is as follows. In section 2, we give an informal description of a library case study used in this paper and a corresponding Z state schema. We also show how to visualize the state schema using the UML class diagram. In section 3, we introduce visualization of complex invariant constraints in Z using Constraint diagrams. In section 4, we describe visualization of operation specifications in Z using Contract boxes. Finally, in section 5, we conclude and discuss future work.

2 A LIBRARY SYSTEM IN Z

In this section, we give an informal description of a library system and a corresponding Z state schema. Then we describe how to represent the state schema with a UML class diagram. The library example has been widely used to demonstrate formal specification techniques [19] and we use it because it needs little explanation.

2.1 Library system: Informal Requirements

The objective of our computer-based library system is to support the management of loans. The following is an informal description of the library system.

The library has registered readers who may borrow. The library maintains a catalogue that lists all publications that are available for loan to registered readers. There may be several copies of the same publication. Each copy of a publication is assigned a unique copy identifier. At any time some copies are on loan and the remaining copies are available for loan. Some constraints given for the library system are:

- a reader can borrow at most five publications;
- a reader may not place more than one reservation for the same publication.

The required operations for the library system are:

- readers, publications and copies can be added to or removed from the library system;
- registered readers can borrow copies that are available for loan;
- readers can reserve publications, when no copies are available for loan;
- when a copy is returned and there is a reservation for that publication, the copy should be held for the reader who made the reservation.

We add two more constraints to the library system to assess the expressiveness of the Z notation and graphical notations used. The two constraints are:

- a reader cannot borrow more than one copy of the same publication;
- a reader cannot reserve publications that he/she has currently borrowed.

2.2 Z specification for the Library System

When we develop a mathematical model for a real system, a decision should be made about an appropriate level of abstraction. For example, in a real library system, there would be a requirement to keep information about registered readers such as name, address and so on. However, we do not need such details to formally specify the operations listed above. Equivalent assertions apply to copies and publications. Thus, we define three given sets: *Reader*, *Copy*, and *Publication* from which all possible readers, copies and publications can be drawn respectively. A complete Z state schema for the library system is shown in Figure 1.

The declaration part may be explained informally as follows:

- *registered* is the set of registered readers;
- *collection* is the set of copies;
- *catalogue* is the set of publications;
- *available* is the set of copies that are available for loan;
- *stock* records the publication associated with each copy;
- *loan* records the borrower of each copy on loan;
- *reservation* identifies publications reserved by readers;
- *held* identifies copies that are held for reservations.

$[\, Reader, Copy, Publication \,]$

$$\begin{array}{|l}
Maxloan: \mathbb{N} \\
\hline
Maxloan = 5
\end{array}$$

$$\begin{array}{|l}
\text{—}Library\text{————————————} \\
registered : \mathbb{P}\ Reader \\
collection : \mathbb{P}\ Copy \\
catalogue : \mathbb{P}\ Publication \\
available : \mathbb{P}\ Copy \\
stock : Copy \pfun Publication \\
loan : Copy \pfun Reader \\
reservation : Reader \leftrightarrow Publication \\
held : (Reader \times Publication) \pinj Copy \\
\hline
dom(loan) \cap ran(held) = \varnothing \\
available = collection \setminus (dom(loan) \cup ran(held)) \\
dom(stock) = collection \\
ran(stock) = catalogue \\
dom(loan) \subseteq collection \\
ran(loan) \subseteq registered \\
dom(reservation) \subseteq registered \\
ran(reservation) \subseteq collection \\
dom(held) \subseteq reservation \\
ran(held) \subseteq collection \\
\forall r : registered \bullet \\
\quad \#\ (loan \rhd \{r\}) \leq Maxloan \\
\quad \forall c1, c2 : loan^{-1} \llparenthesis \{r\} \rrparenthesis \bullet \\
\qquad c1 \neq c2 \Rightarrow stock(c1) \neq stock(c2) \\
\quad reservation \llparenthesis \{r\} \rrparenthesis \cap (loan^{-1} \fatsemi stock) \llparenthesis \{r\} \rrparenthesis = \varnothing \\
\hline
\end{array}$$

Figure 1. Z State schema for the Library system

2.3 Visualizing Static Aspects of a Z Specification

In Z, a state schema represents the abstract states of a system being modeled. In detail, a state schema describes the entities that exist in the system, the relationships that exist between the entities and the constraints that are placed on the entities and their relationships. The first two aspects are described in the declaration and the last one is described in the predicate.

In the library example, the sets *registered*, *collection*, and *catalogue* represent the three entities in the system: readers, copies and publications. The relations *stock*, *loan*, *reservation*, and *held* represent relationships between these three entities. In our work, the system being modeled is translated to a root class of a UML class diagram, with which all other classes in the diagram are associated directly or indirectly. The state schema name is used as the class name and a stereo type <<*system*>> is added to clearly describe that the class represents the system. Each variable stated in the state schema is then translated to the most appropriate UML class construct depending on its semantics. A brief description of translation rules is given and then explained in more detail:

- Variables representing entities are translated to individual classes and connected to the root class with associations. The variable names are used as the association names.
- Variables representing relationships between entities are translated to associations or association classes in UML.

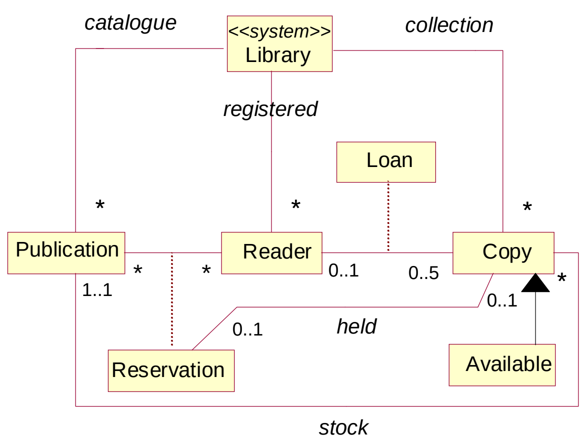


Figure 2. UML class diagram: the visual representation of Library state schema

Variables representing individual entities:
In Z, each entity in a system is associated with a given type or several types that are used to model that entity. For example, type *Reader* is a given set from which all possible readers can be drawn. Readers in the library system at any point in time is then modeled as the set *registered* of type *Reader*. Each reader is recognized by his/her identifier. In this sense, the semantics of type *Reader* and set *registered* is very similar to a class in UML.

Semantically, a class in UML has two aspects. First a class in isolation is interpreted as a type from which objects with the same properties such as attributes and operations can be drawn. Second, a class represents a set of existing instances of that class when it is interpreted as a component of a UML class diagram. The first interpretation of a UML class is very similar to that of type *Reader*. The second interpretation is very similar to that of set *registered*. Based on this semantic interpretation, we translate each variable declared in the state schema representing an entity in the system and its type to a UML class.

In the library example, type *Reader* and set r*egistered* are translated to a class *Reader*. For the same reason, type *Copy* and set *collection*, and type *Publication* and set *catalogue* are also translated to the classes *Copy* and *Publication* respectively. These three classes are connected to the root class *Library* with associations.

In the case of set *available*, the set is restricted to a subset of set *collection* representing the group of copies that are in the available state so that it can be best expressed as a subtype of type *Copy*. However, the UML class diagram does not express state information of objects in the diagram. For this reason, we use a notation, a filled triangle, which is used to define the states of particular types in [13], to express sets like *available* in the UML class diagram (see Diagram 1).

Variables representing relationships:
In Z, relationships between entities are modeled using various mathematical relations and the types of the relations are decided by the characteristics of the relationships.

In the library example, variable *stock* is a partial function because each copy identifier corresponds to at most one publication. However, there can be many copy identifiers that are associated with one publication. Variable *loan* is also a partial function because each copy can be on loan to at most one reader, while not all of copies are on loan at a certain time. Each reader can borrow up to *Maxloan* copies. Variable *reservation* is a relation because readers can make several reservations. Variable *held* is defined as a partial injection from a pair of type *Reader* and *Publication* to type *Copy* because each reservation can have at most one corresponding copy that is held for that reservation. Also, each copy can correspond to at most one reservation.

In UML, a relationship between objects can be represented as an association or an association class. Therefore, variables representing relationships between entities in the state schema can be translated to one of these class constructs. In UML, the only difference between an association class and an association is that the association class has class-like properties such as its own attributes and operations that do not belong to any of the associated classes. Otherwise, the semantics of an association class is the same as that of an association. In fact, defining a relationship between classes as an association or an association class is a modeling decision. It also depends on the level of abstraction taken for developing the model. Thus, our translation rules described here also involve modeling decisions. For example, variable *loan* records the borrower of each copy on loan. In OO modeling, this kind of information is usually modeled as separate objects. Moreover, in a more comprehensive system, there would be additional information about loans such as loan date and due date. However, this information does not belong to either readers or copies. For these reasons, variable *loan* is translated to an association class *Loan*. For the same reason, variable *reservation* is translated to an association class *Reservation*.

In the case of variable *stock*, the variable associates each copy with its publication details. However, at the level of formal description, there is no additional information associated with this relationship. For this reason, variable *stock* is translated to an association between class *Copy* and *Publication*. For the same reason, variable *held* is also translated to an association between class *Reservation* and *Copy*. However, if there is additional information that is associated with these relationships, they could be expressed as association classes.

Multiplicity constraints for associations are determined by the types of functions or relations used to define the variables and constraints given for them (Table 1 shows a mapping between Z relations and multiplicity constraints in UML).

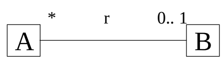
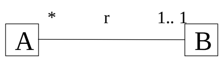
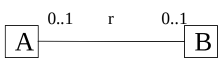
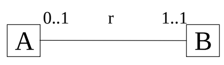
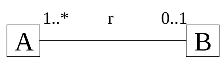
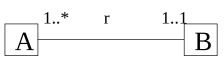
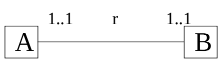
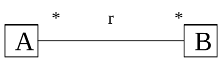
Z functions	Predicates	UML associations
$r: A ⇸ B$ *partial*	$dom(r) \subseteq \mathbb{P}A$, $ran(r) \subseteq \mathbb{P}B$	
$r: A \rightarrow B$ *total*	$dom(r) = \mathbb{P}A$, $ran(r) \subseteq \mathbb{P}B$	
$r: A ⤔ B$ *partial injection*	$rom(r) \subseteq \mathbb{P}A$, $ran(r) \subseteq \mathbb{P}B$	
$r: A \rightarrowtail B$ *total injection*	$dom(r) = \mathbb{P}A$, $ran(r) \subseteq \mathbb{P}B$	
$r: A ⤀ B$ *partial surjection*	$dom(r) \subseteq \mathbb{P}A$, $ran(r) = \mathbb{P}B$	
$r: A \twoheadrightarrow B$ *total surjection*	$dom(r) = \mathbb{P}A$, $ran(r) = \mathbb{P}B$	
$r: A ⤖ B$ *bijection*	$dom(r) = \mathbb{P}A$, $ran(r) = \mathbb{P}B$	
$r: A \leftrightarrow B$ *relation*	$dom(r) \subseteq \mathbb{P}A$, $ran(r) \subseteq \mathbb{P}B$	

Table 1. Z relations and their corresponding UML expressions

2.4 Analysis of the UML Class Diagram
The class diagram in Figure 2 is a visual representation of the *Library* state schema. The diagram represents the entities and their relationships stated in the schema with the most appropriate UML class constructs such as classes, association classes or associations. The semantics of the relationships described in Z is accurately depicted by the multiplicity constraints on the associations. For example, the variable *loan* is a partial function from *Copy* to *Reader*, which means a reader can borrow many copies (further restricted by *Maxloan* (five) in the predicate), but a copy can be borrowed by a reader or none at any time. This constraint is exactly expressed by the multiplicity constraints on the association class *Loan 0..1* and *0..5*. These multiplicity constraints also express the domain and range constraints given for the *loan* such as $dom(loan) \subseteq collection$ and $ran(loan) \subseteq registered$.

On the other hand, more complex constraints in Z notation such as the first two lines and the last three lines in the predicate of the *Library* state schema are not expressed in the diagram. We use object diagrams in UML to explain the complex constraints cannot be expressed with the UML notation alone. Figure 3 shows three valid object diagrams of the UML class diagram for the library system. Diagram (a) shows a case where a reader can loan two copies for the same publication. Diagram (b) shows a case where a reader can place a reservation for the same publication that he/she has currently borrowed. Diagram (c) shows a case where a copy is in *loan* state and in *held* state at the same time. However, these three states conflict to the requirements of the library system described in section 2.1. In UML, Object Constraint Language (OCL) [2] can be used to supplement lack of expressiveness of the graphical notation to specify complex constraints precisely. In our work we express these constraints diagrammatically as explained in the following section.

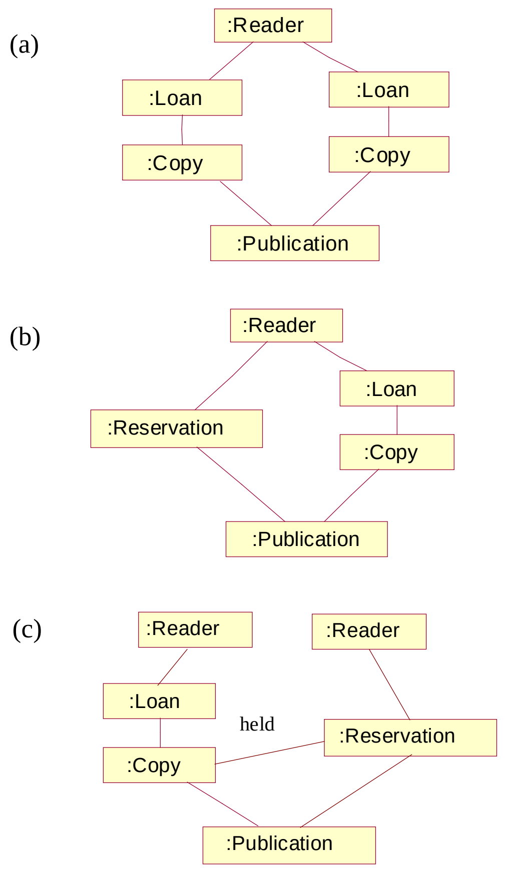


Figure 3. Object diagrams for the three classes

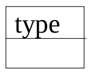
3. VISUALIZING COMPLEX CONSTRAINTS

In this section, we give a brief description of the Constraint diagram and explain how to visualize complex constraints in mathematical notations using Constraint diagrams.

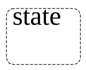
3.1 Constraint Diagrams

Constraint diagrams, developed by Kent [13], are designed to express invariant constraints on OO models precisely (i.e., especially for UML models). The notation is based on Venn diagrams and other diagrams used by mathematicians to illustrate properties of functions and relations. The distinctive feature of Constraint diagrams is the ability to express universal and existential set membership, in addition to general set relations such as equivalence, intersection and containment. In this sense, Constraint diagrams are well suited to express constraints on sets and set members visually. Since Constraint diagrams are designed to supplement the UML notation, the notation is similar to the UML notation. A Constraint diagram can be considered as a variation of the UML class diagram, which can express states of objects and sets in the diagram as well as all the other information that the UML class diagram can represent such as types, associations, and so on. The detailed notation is defined in [13]. We summarize the basic notation used in this paper.

A type (class) is depicted as the set of objects in that type.



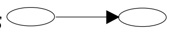
A state is depicted as the set of objects in that state.



Associations are depicted as relations between sets of objects and the notation for the associations are links. Links are directed to indicate the direction to read the association. Venn diagrams are used to express relationships between associations. Set relationships such as equivalence, intersection and the containment can be depicted. Members of a set are described by introducing singletons. A filled small circle indicates a singleton.

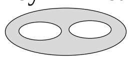
A set is depicted in four ways depending on the number of elements in the set.

⬭ means Null set	● means 1 element
○ means 0..1 elements	⬭n means n elements
s ⬭ means 0 or more elements (s is a label)	

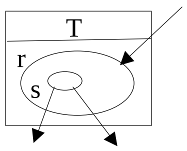
Navigation always begins at an object or set with no incoming arrows. Navigation describes the starting point (set s) for reading the diagram. s  t

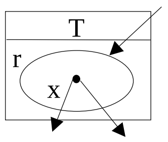
Areas

Grey fill means that there are no elements in that area.



Universally quantified sets

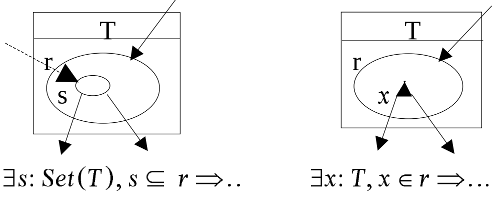




$\forall s\colon Set(T), s \subseteq r \Rightarrow. ..$ $\quad$ $\forall x : T, x \in r \Rightarrow \ldots$

Existentially quantified sets

Existential quantification is depicted by introducing a temporary, unlabeled association, a dotted line with an arrow. The icon,▲ , is used to distinguish existential from universal quantification set members.



$\exists s\colon Set(T),\ s \subseteq r \Rightarrow ..$ $\qquad \exists x\colon T,\ x \in r \Rightarrow \ldots$

Constraint diagrams can be drawn in two different ways. One way encloses the classes within an object which is the system object. For example, all classes are enclosed within an object of class *Library* and the associations between the *Library* and other classes are drawn as links with no incoming arrow (e.g. see Figure 4). The other way is to draw all classes including *Library* class and the navigation starts from the least defined set (a singleton set) on the diagram. In this paper, we draw Constraint diagrams the first way.

3.2 Representing Constraints with Constraint Diagrams

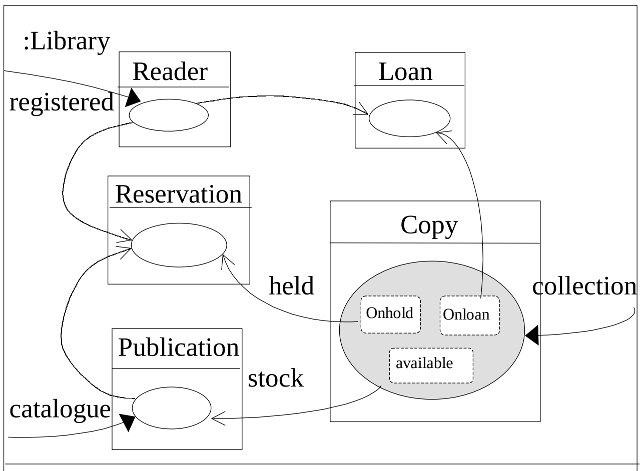
Four invariant constraints that are not represented in the UML class diagram (Figure 2) are as follows:

$[1]\quad dom(loan) \cap ran(held) = \varnothing$

$[2]\quad available = collection \setminus (dom(loan) \cup ran(held))$

$[3]\quad \forall\ r : registered \bullet$
$\quad \forall c1, c2 : loan^{-1}(\!|\{r\}|\!) \bullet$
$\quad\quad c1 \neq cl2 \Rightarrow stock\ (c1) \neq stock\ (c2)$

$[4]\quad \forall\ r : registered \bullet$
$\quad reservation\ (\!|\{r\}|\!) \cap (loan^{-1} \mathbin{;} stock)\ (\!|\{r\}|\!) = \varnothing$

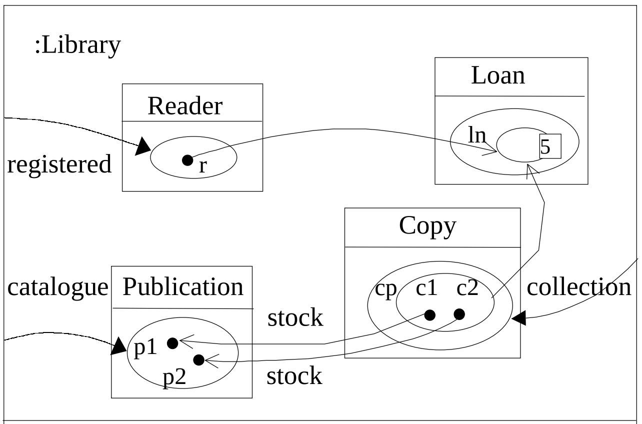


$$dom(loan) \cap ran(held) = \varnothing$$
$$available = collection \setminus (dom(loan) \cup ran(held))$$

Figure 4. Constraint diagram for constraint 1 and 2

Constraint 1 and 2:

Figure 4 shows a Constraint diagram that depicts all the variables stated in the *Library* schema and their relationships. Constraint 1 and 2 above are represented by introducing the three states (subsets) within set *collection* such as *Onhold, Onloan* and *available*. The links show that the copies in the *Onloan* state are related to set *loan* and the copies in the *Onhold* state are related to set *reservation*. Also these three sets are disjoint, which means that a copy in the *available* state can be neither in the *Onloan* state nor in the *Onhold* state at the same time. The gray color filled represents that there are no elements in that area, which means that any copy in set *collection* must belong to one of these three states. Thus, the diagram depicts the property captured in constraint 1 and 2 exactly.



$$\forall\ r : registered \bullet$$
$$\forall c1, c2 : loan^{-1}(\!|\{r\}|\!) \bullet$$
$$c1 \neq cl2 \Rightarrow stock\ (c1) \neq\ stock\ (c2)$$

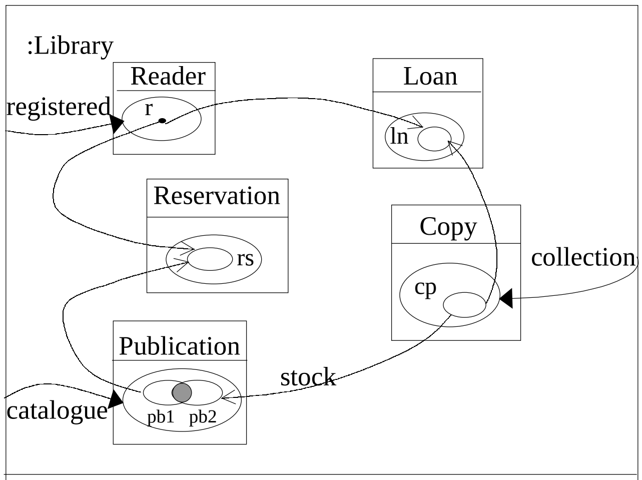
Figure 5. Constraint diagram describing constraint 2

Constraint 3:

Figure 5 shows a Constraint diagram describing constraint 3 above. The semantics of the diagram is that for all *r* in the set *registered*, there is a subset of set *loan* named *ln* that maps to *r* and also maps to a subset of set *collection* named *cp*, such that the size of the set *ln* is restricted to five, which means that a reader can borrow at most five copies. Also all distinct members in the set *cp* named *c1* and *c2,* map to different publications in set *catalogue* named *p1* and *p2* respectively, meaning that a reader can not borrow more than one copy of the same publication. In this diagram, we add labels to the diagram to clearly clarify the mapping from the mathematical expression of the constraint to the diagram.

Constraint 4:

Figure 6 shows a Constraint diagram describing constraint 4 above. The semantics of the diagram is that for all *r* in set *registered*, there is a subset of set *reservation* that maps to *r* named *rs*, such that there is no intersection between two sets named *pb1* and *pb2* that map to *rs* and *cp* respectively. This means that readers cannot reserve any publication that they have currently borrowed or borrow a copy of a publication that they have reserved without removing the reservation.



$$\forall\ r : registered \bullet$$
$$reservation\ (\!|\{r\}|\!) \cap (loan^{-1} \mathbin{;} stock)\ (\!|\{r\}|\!) = \varnothing$$

Figure 6. Constraint diagram describing constraint 3

4. VISUALIZING Z OPERATION SPECIFICATIONS

In this section, we give a brief description of the Contract box and explain how to visualize Z operation specifications using this diagram. In the Z operation specifications illustrated in this paper, we do not deal with error conditions.

4.1. Contract Boxes

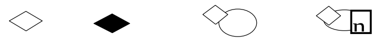
A contract box [10, 14] is a 3D notation designed to express constraints on dynamic behaviors, which are usually expressed textually or mathematically. The diagram is adopted in our approach to visualize Z operation schemas, which express dynamic behaviors of a system and their constraints.

The notation of a contract box is basically the same as those of Constraint diagrams with some extensions. However, as the term box implies, the diagram is drawn in 3D, with a Constraint diagram pasted to the lid and another pasted to the bottom. The top diagram constrains the pre-state and the bottom diagram constrains the post-state. Drawing contract boxes is quite simple. For each Z operation schema, a Constraint diagram describing the pre-condition of the operation and the pre-state of sets involved in the operation is drawn and pasted to the lid of a contract box. Another Constraint diagram describing the post-state of the sets involved in the operation is drawn and pasted in the bottom of the contract box. Detailed description of the diagram is provided in [10, 14]. The additional notations are as follows:

Lifelines connect sets in the top diagram and the bottom diagram to show how the set changes.

Action (operation) invocation shows action invoked on the targeted object.

Action arguments (operations arguments) depict sets of different cardinality, which are arguments to actions.



0..1 1 0..1 or more n elements

New objects depict sets of different cardinality containing new objects.

0..1 1 0..1 or more n elements

4.2 Representing Z Operations with Contract Boxes

In Z, each operation is specified as a separate operation schema. The declaration part of an operation schema includes the state schema of the system and input/output variables necessary for that operation. The predicate part expresses the pre-condition for that operation and the state change involved in that operation. Since in our approach one contract box represents one operation schema, there is a one to one mapping between diagrams and Z operation schemas. Thus, when an operation schema is modified, it is easy to trace and modify the corresponding diagram. In this paper, we introduce two operation schemas for the library system: *AddCopy* and *BorrowCopy*, and their visual representations. In the two operation schemas, we have omitted conjuncts equating all the variables that are to be unchanged.

1. Operation AddCopy

A Z operation schema for operation *AddCopy* is given in Figure 7. The first two lines in the predicate part describe the pre-condition of the operation and the last two lines represent the post-state of variables *collection* and *stock* respectively.

AddCopy

$\Delta\ Library$
$cid? : Copy$
$pid? : Publication$

$pid? \in catalogue$
$cid? \notin collection$
$collection' = collection \cup \{cid?\}$
$stock' = stock \cup \{cid? \mapsto pid?\}$

Figure 7. Operation schema for AddCopy

The visual representation of operation schema *AddCopy* is given in Figure 8. The lid of the contract box represents the pre-condition and the bottom diagram represents the post-state of variables *collection* and *stock*. In detail, the action invocation symbol indicates a new operation is invoked on a library object and the null set symbol represents that *cid?* should not be in set *collection*. The filled diamond represents that *pid?* should be in set *catalogue*. The star symbol in the bottom diagram indicates that a new object has been created in set *collection* for *cid?* as the result of the operation and *cid?* is mapped to *pid?* with association *stock*. The lifelines show the set changes. Sets not represented in the diagram are unchanged.

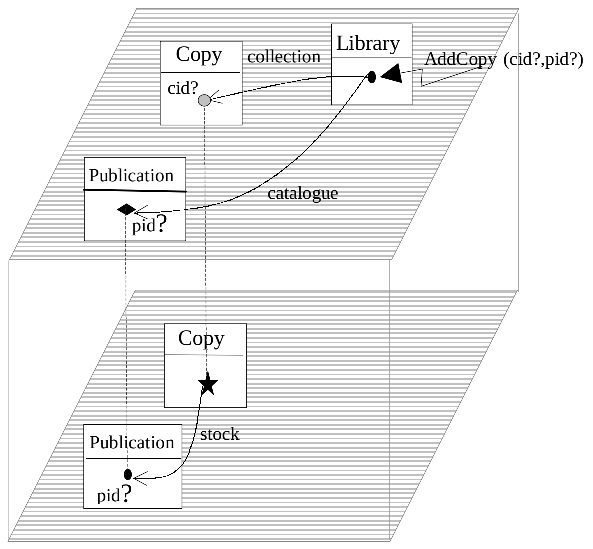


Figure 8. Contract box for operation AddCopy

2. Operation BorrowCopy

$$
\begin{array}{l}
\textit{BorrowCopy} \\
\hline
\Delta\, Library \\
rid? : Reader \\
cid? : Copy \\
\hline
rid? \in registered \\
\#\,(loan \rhd \{rid?\}) < Maxloan \\
cid? \in\ ran(held) \Rightarrow \\
\quad first\ (held^{-1}\ (cid?)) = rid? \\
\quad held' = held \ntriangleright \{cid?\} \\
\quad loan' = loan \cup \{cid? \mapsto rid?\} \\
cid? \in available \Rightarrow \\
\quad stock\ (cid?) \notin\ (loan^{-1} \fatsemi stock)\ (\!|\{rid?\}|\!)\ \cup \\
\qquad\qquad\qquad\qquad \{rid?\} \lhd reservation \\
\quad available' = available \setminus \{cid?\} \\
\quad loan' = loan \cup \{cid? \mapsto rid?\} \\
\hline
\end{array}
$$

Figure 9. Operation schema for BorrowCopy

A Z operation schema for operation *BorrowCopy* is given in Figure 9. The predicate part describes all constraints placed on the operation. The first two lines maintain consistency with the invariant in the *Library* state schema. The third to the sixth lines describe the case when *cid?* is in the *Onhold* state. In this case, if *cid?* is held for *rid?*, then *rid?* can borrow *cid?* and the state of *cid?* is changed from the *Onhold* state to the *Onloan* state. Thus, variables *held* and *loan* need to be changed. The seventh to the last lines describe the case when *cid?* is in the a*vailable* state. In this case, an additional constraint is given, which is that the publication of *cid?* should not be currently borrowed or reserved by *rid?*. If this constraint is satisfied, then *rid?* can borrow *cid?* and the state of *cid?* is changed from the *available* state to the *Onloan* state. Thus, variables *available* and *loan* need to be changed.

A constraint diagram describing pre-condition:
Since operation *BorrowCopy* describes two different valid pre-state involving *cid?*, we draw the contract boxes in a structured way. On the top-level diagram, the common pre-condition of the operation (the first two lines in the predicate) is represented. More detailed pre and post-state of sets involved in this operation are described in two separate diagrams *PostBox1* and *PostBox2* depending on the pre-state of *cid?*. In this way, we avoid a complexity problem that may arise by drawing all complex constraints in one diagram.

Figure 10 is a Constraint diagram showing the pre-condition of operation *BorrowCopy*. The diamond symbol in set *registered* indicates that *rid?* should exist in set *registered*. The set size *4* in the *Loan* constrains the allowable maximum set size for the subset of set *loan* that maps to *rid?*.

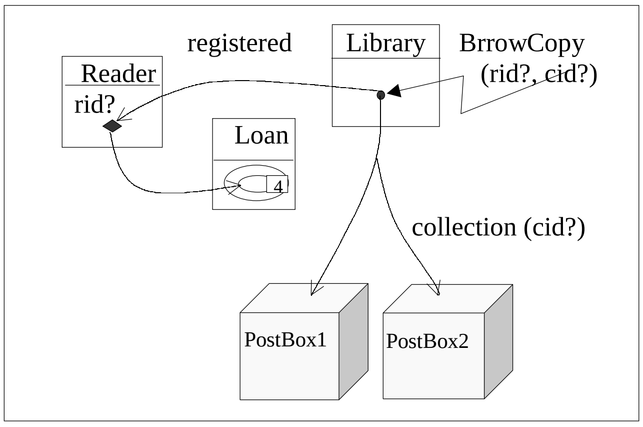


Figure 10. Constraint diagram showing the pre-condition of operation BorrowCopy

Two post boxes describing pre and post-state:
PostBox1 in Figure 11 describes the case when *cid?* is in the *Onhold* state. PostBox2 describes the case when *cid?* is in the *available* state. The top diagram of the PostBox1 depicts that only if *cid?* is held for *rid?*, then *rid?* can borrow *cid?*. The bottom diagram shows the state change of *cid?*, which is moved from the *Onhold* state to the *Onloan*

state. Also the null symbol in set *reservation* (a gray filled circle) describes that there should be no elements in the set that map to *cid?* via association *held*. The top diagram of the PostBox2 shows the pre-state when *cid?* is in the *available* state. The mappings between type *Copy* and *Publication* depict that *cid?* should not map to any publication that *rid?* has currently borrowed or reserved. The bottom diagram shows that the state of *cid?* has been moved from the *available* state to the *Onloan* state. For both cases, when all the constraints are satisfied, a new object indicated by a star symbol that maps to both *rid?* and *cid?* (see the navigation arrows) is created in set *Loan*.

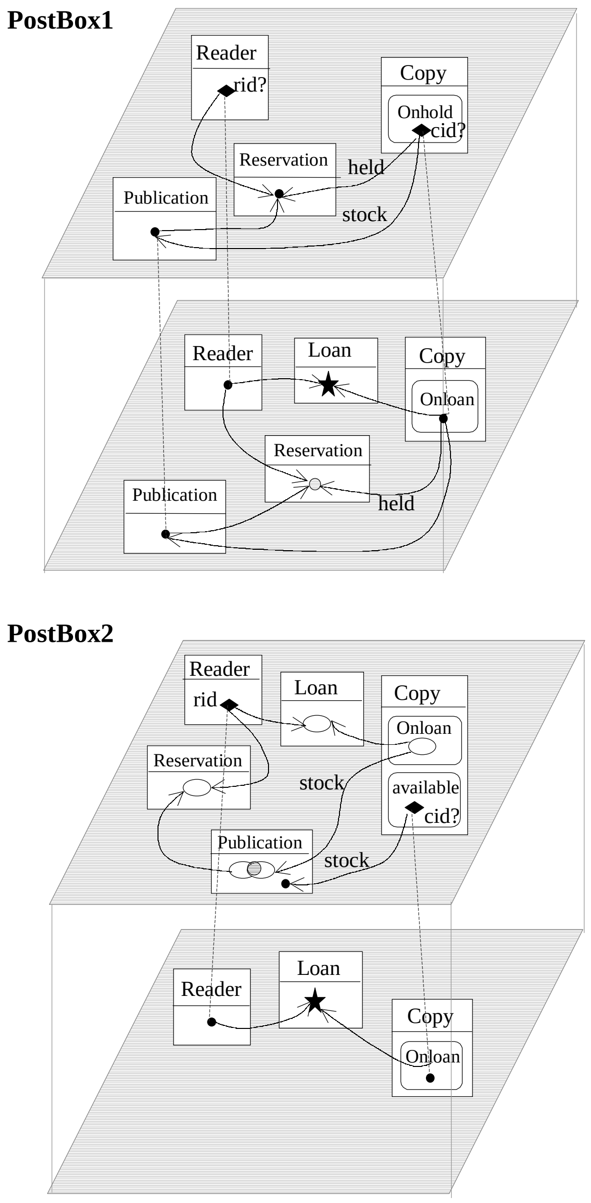


Figure 11. Post boxes

Diagrams	Z Expressions
Constraint diagram in Figure 10	$rid? \in registered$ $\#\,(loan \rhd \{rid?\}) < Maxloan$
The lid of PostBox1	$cid? \in ran(held) \Rightarrow$ $first\ (held^{-1}\ (cid?)) = rid?$
The bottom of PostBox1	$held' = held ⩥ \{cid?\}$ $loan' = loan \cup \{cid? \mapsto rid?\}$
The lid of PostBox2	$cid? \in available \Rightarrow$ $stock\ (cid?) \notin\ (loan^{-1} ⨾ stock)\ (\!\lvert\{rid?\}\rvert\!)$ $\cup \{rid?\} \lhd reservation$
The bottom of PostBox2	$available' = available \setminus \{cid?\}$ $loan' = loan \cup \{cid? \mapsto rid?\}$

Table 2. Diagrams and their corresponding Z expressions

Table 2 shows a mapping between the diagrams developed for operation *BorrowCopy* and their Z expressions. When constraints expressed in Z notation are complex, their corresponding visual expressions are also complex. In this sense, the complexity of the diagrams represents the inherent complexity of the mathematical expressions. However, we believe that understanding of the visual expressions is still easier than that of their corresponding mathematical expressions. The graphical notation used in these diagrams is simple and intuitive. Moreover, the level of mathematical knowledge required to understand the diagrams is less than that required to understand their corresponding mathematical expressions.

5. CONCLUSIONS

In this paper, we have presented a case study that uses various diagrams to represent various aspects of a Z specification. The static aspects are represented by a UML class diagram. Complex invariant constraints that cannot be expressed in the UML notation alone are described using Constraint diagrams. The dynamic aspects of operation schemas are represented with contract boxes. As far as we are aware, no study yet reports using diagrams to express complex mathematical assertions found in formal specifications.

In translating a Z specification to a UML class diagram, we analyze the semantic relations between variables declared in the state schema and the UML class constructs. As a result, the variables are translated to the most appropriate UML class constructs depending on their semantics.

In the case of the Constraint diagram, the semantics of the diagram has not been defined precisely. Therefore, as the developers of Constraint diagrams state [14], the diagrams cannot yet be considered as a true Visual Formalism [12]. However, this semantic issue is not critical for our work because we use diagrams as visual representations of formal specifications. Thus, the formal specifications are the semantic models of the diagrams.

In future work, we will define a development environment to support generating diagrams from Z specifications. In our approach, we use well-known diagrams rather than developing new notations. Therefore, the creation and manipulation of the diagrams can be done using existing tools that are already available to support the diagrams. Our tool will interconnect existing tools for both notations and support automatic translations between the two representations.

In conclusion, a Z specification is basically a 2D textual description with mathematical symbols. In this paper, a Z specification is documented with 3D graphical notations, which provides a visual and structured way to read and understand the Z specification. We believe this should result in a significant improvement in the readability and understandability of Z specifications and should make formal specifications more accessible for non-mathematicians.

REFERENCES

1. G. Booch, J. Rumbaugh, and I. Jacobson, editors, *UML Notation Guide, Version 1.1*, Rational Software Corporation, Santa Clara, CA-95051, USA, January, 1997. http://www.rational.com.

2. G. Booch, J. Rumbaugh, and I. Jacobson, editors, *Object Constraint Language, Version 1.1*, Rational Software Corporation, Santa Clara, CA-95051, USA, January, 1997. http://www.rational.com.

3. J. P. Bowen and M. G. Hinchey, Ten Commandments of Formal Methods, *Computer*, vol. 28(4), pp. 56-63, 1995.

4. J. P. Bowen and V. Stavridou, Safety-critical systems, formal methods and standards, *Software Engineering Journal*, vol. 8(4), pp. 189-209, 1993.

5. J. Dick and J Loubersac, A Visual Approach to VDM, SOFSEM'96, *Theory and Practice of Informatics*, LNCS, vol. 1175, pp. 275-284, Springer-Verlag, 1996.

6. A. Evans, R. B. France, K. Lano, and B. Rumpe, The UML as a Formal Modeling Notation. In Pierre-Alain Muller and Jean Bezivin, editors, *Proc. UML'98 Workshop" Beyond the notation*", Mulhouse, France, pp. 336-348, ESSAIM, Mulhouse, France, 1998.

7. G. P. Faconti, A. Fornari, and N. Zani, Visual Representation of Formal Specification: An Application to Hierarchical Logical Input Devices, *Interactive Systems: Design, Specification and Verification-1st Eurographics workshop*, pp. 349-367, Springer-Verlag, 1994.

8. K. Finney, Mathematical Notation in Formal Specification: Too Difficult for the Masses?, *IEEE Transactions on Software Engineering*, vol. 22(2), pp. 158-159, 1996.

9. R. B. France, J.-M., Bruel, M. M. Larrondo-Petrie, and M. Shroff. Exploring the Semantics of UML type structures with Z, Proc. 2nd IFIP conference, *Formal Methods for Open Object-Based Distributed Systems(FMOODS'97)*, pp. 247-260, Chapman and Hall, London, 1997.

10. J. Gil and S. Kent, Three Dimensional Software Modeling, *In Proc. of ICSE98*, IEEE Press, 1998. http://www.cs.ukc.ac.uk/people/staff/sjhk/pubs.html

11. M. Gogolla and M. Richters, On Combining Semi-Formal and Formal Object Specification Techniques, In Francesco Parisi-Presicce, editors, *Proc. 12th Int. Workshop on Abstract Data Types(WADT'97)*, LNCS 1376, pp. 238-252, Springer-Verlag, 1998.

12. D. Harel, Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, vol. 8(3), pp. 237-274, 1987.

13. S. Kent, Constraint Diagrams: Visualising Invariants in Object-Oriented Models, *In Proc. of OOPSLA97, ACM SIGPLAN Notices*, vol. 32(10), pp. 327-341, ACM Press, 1997.

14. S. Kent and J. Gil, Visualising Action Contracts in Object-Oriented Modeling, *VISUAL98 a workshop at ETAPS98*, 1998. http://www.cs.ukc.ac.uk/people/staff/sjhk/pubs.html

15. S. Owre, J. Rushby, N. Shankar, and F. von Henke, Formal Verification for Fault Tolerant Architectures: Prolegomena to the Design of PVS, *IEEE Transactions on Software Engineering*, vol. 21(2), pp. 107-125, 1995.

16. G. Reggio and M. Larosa, A Graphical Notation for Formal Specifications of Dynamic Systems, In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods, 4th International Symposium of Formal Methods Europe*, LNCS, vol. 1313, pp. 40-61, Graz, Austria, 1997.

17. J. M. Spivey, *The Z Notation: A Reference Manual*, Prentice Hall International Series in Computer Science, 2nd edition, 1992.

18. E. N. Wafula and P. A. Swatman, FOOM: a diagrammatic illustration of Object-Z specifications, *Object Oriented Systems*, vol. 3, pp. 215-242, 1995.

19. J. M. Wing, A Study of 12 Specifications of the Library Problem, *IEEE Software*, vol. 5(4), pp. 66-76, 1988.