IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 11, Issue 5 (May. - Jun. 2013), PP 37-39
www.iosrjournals.org

Comparison of the Formal Specification Languages Based Upon Various Parameters

Dr. A.K.Sharma, Monika Singh
Mody Institute of Technology and Science (MITS), Lakshmangarh, Rajasthan

Abstract:*This paper focus on various Formal method and Formal specification languages and choosing the appropriate one for a particular problem. It describes how a particular specification language is used for a specific problem. A comparison of different formal specification languages is done taking various parameters.*
Keywords:*Formal method, Formal specification language, schema.*

I. Introduction

One of the major problems with software intensive system is the inadequacy of system and software specification. The requirements should usually define the major functions of the system and/or software adequately, but many of the details which should be drawn out, cleaned up, and solidified in a more detailed specification are not addressed. This causes inconsistencies and misinterpretations which consequently cause problems in later stages of design and implementation. These problems are often recognized in the integration stages of system. There are software development methods based on graphical techniques, such as data-flow diagrams, finite state machines, and entity relationship diagrams which are proved helpful in developing better specifications, but these lack precisions in details of the specification, and a smooth way in developing a design and implementation.

The formal specification methods overcome these problems. They specify the system precisely, and provide a smooth way from specification through design to implementation. There are successful applications of formal methods to real-life, complex software specifications for example, CICS work at IBM [1], and the oscilloscope work at Tectronix. There are a number of formal specification languages such as Z notation, OCL, VDM, LARCH etc. Basically all these formal specification languages include – formal specification, refinement, and verification which in term consist of set theory, logics, and algebra etc. The primary goal of our current endeavors is to compare these formal specification languages on the basis of their syntax.

II. Z notation

Z notation is based on typed set theory and first-order logic. Z provides a construct, called a schema, to describe a specification's state and operations. A schema groups variable declarations with a list of predicates that constrain the possible value of variable.

Some Z notation concept
Data Invariant-A data invariant is a condition that is true throughout the execution of the system.
State-In Z specification, the state is represented by the system's stored data.
Operation-Operation is an action that takes place within a system and read or writes data. Three types of conditions are associated with operation:
Invariant-an invariant define what is guaranteed not to change.
Precondition- a precondition defines the circumstances in which a particular operation is valid.
Postcondition- postcondition of an operation define what is guaranteed to be true upon completion of an operation. This is defined by its effect on data.
For example, Block handler – maintains a reservoir of unused blocks and will also keep track of blocks that are currently in use. When blocks are released from a deleted file they are normally added to queue of blocks waiting to be added to the reservoir of unused blocks. Now in Z notation [2], this has been depicted schematically as follow:
BLOCKS- a set BLOCKS will consist of every block number.
ALLBLOCKS- is a set of blocks that lie between 1 and MAXBLOCKS.
The state will be modeled by two sets- used and free. The state can be described as
Used, free: BLOCKS
BlockQueue: seq P BLOCKS
The above description tells that used and free will be sets of blocks and that BlockQueue will be a sequence, each element of which will be a set of blocks.
The data invariant can be written as

Used ^ free = null
Used ∪ **free = AllBlocks**
The precondition and postcondition is also verified like this, mathematically. In Z specification language we use the set theory, logics, and algebra.

III. OCL (Object constraint language)

The Object Constraint Language (OCL) is an expression language which describes constraint on object-oriented languages and on other modeling artifacts. OCL is part of Unified Modeling Language (UML) and also play an important role in the analysis phase of software lifecycle. Traditional graphical model, like class models are not enough for a precise and unambiguous specification. For that we need to implement some additional constraint on the objects. But the problem with the traditional formal method is that one have a good grip on mathematics, hence are difficult for average business or system modeler to use. To fill this gap, OCL has been developed. It has been developed as a business modeling language within the IBM Insurance division. Since OCL is an expression language, the state of the system will never change by imposing constrain on the object. It just limit their range. The state of the objects in system cannot change during evaluation.
Some of the OCL notations are given below:
Some OCL notation
C→f() – Apply the built-in operation f to collection C.
X.Y – Obtain the property Y of object X.
C1→ includesAll(C2) —True if every element of C2 is found in C1.
C→isEmpty() – True if C has no element.
C→size() – The number of elements in collection C.
S1→intersection(S2) – The set of those elements found in S1 and also in S2.
Block Handler, for example the very first step for using OCL [3] is to develop a UML model (class diagram), which specifies many relationship among the objects involved; however we should also add OCL expression to ensure that implementers of system know more precisely what they must ensure remain true as the system runs.
For Example:-No block will be marked as both used and unused.

Context BlockHandler inv: self.used→intersection(self.free))→ isEmpty()

Here, **context** indicates the element of the UML diagram that the expression constrains. And the keyword **self** refer to the instance of object (in this case instance of BlockHandler). **Self.a** defines the 'a' attribute or property of object.One of the plus point of using OCL is that it is more friendly to people who are less mathematically inclined. Also it is more easily processed by computer. But it makes OCL a little wordy in places.

IV. Vdm (Vienna Development Method)

VDM refers to a collection of techniques for formal specification and development of computing systems. Basically it includes specification language called VDM-SL, rules for data and operation refinement, and a proof theory. VDM-SL is a model-oriented specification language. i.e a specification in VDM-SL consists of a mathematical model built from simple data types like sets, lists, and mappings, along with operations which change the state of the model. For example, a specification of a hotel reservation system would contain a mapping from room number to names and addresses of occupants (modeled as character strings), along with the operation to add and remove guests and rooms occupation dates etc. A formal Model in VDM is composed of Basic types,Defined types (with many useful constructors),Invariant of those types, Explicit function definition (include precondition), and Implicit definitions (postcondition).There is also tool available for VDM specifications. SpecBox is such an industrialized tool for inputting, checking and printing VDM specifications [4, 5]. It is used in industries of avionics, civil nuclear, railway, security applications and in academic applications.

V. SDL (Specification and Description Language)

SDL is an object-oriented formal language which is defined by The International Telecommunication Union-Telecommunications Standardization sector. SDL basically intended for specification of complex, event-driven, real-time, and interactive applications involving many concurrent activities that communicate using discrete signals. Instead of schema in Z notation or Class model in OCL, SDL [6] system consists of a set of extended Finite State Machines (FSMs) that run in parallel. A SDL system consists of following components:
Structure- system, block, process, and procedure hierarchy,
Communication- signal with optional signal parameters,
Behavior- processes,
Data- abstract data types (ADT),

Inheritance- describing relations and specialization.
The above mentioned components are described briefly below:
System- The overall design of the software product is called the system. The universe outside the system is called the environment.
Blocks- Blocks are the primary divisions of the system. Dividing the system into blocks thus enables the pictorial representation of all the essential components of the given system.
Process- When the system is divided down to the simplest block, the way the block fulfils its functionality is described by processes. Basically a process is the code that will be executed. The complete syntax in process symbol is:
<Process name> [(<number of instances at startup>, <maximum number of instances>)]
Communication- SDL has two basic communication mechanisms: asynchronous signal and synchronous remote procedure calls.
Inheritance- Since SDL is an object-oriented, which provides the user the powerful tool for structuring and reuse.

VI. Larch

Larch is an ongoing project at MIT's laboratory for Computer Science and DEC's system Research Centre. Basically it support a two tired specification, each of which has two component written in two language named as LIL(Larch Interface Language) and the other one is LSL (Larch Shared Language). LIL [7, 8] is designed for a specific programming language. It is used to describe the observable behavior of the component and is language dependent.

VII. Comparison

Here we are trying to describe the differences in specification languages by the means of the following parameters:
Process-oriented – These are designed to describe concurrent networks of communicating component's behaviors, and since most of the language describe system in term of process, so it is called as process-oriented.
Sequential-oriented – It provide the appropriate way to describe the input-output behavior of sequential system. **Model-oriented** – The language which fall in this category provide an abstract model of system and the operation are specified by either state change or by event affect the model.
Property-oriented – These focus on the property desired by the system being specified and on data type instead of services/ functions that the system provides.
Mathematics used—Algebra, logic, set theory. All the specification languages discussed here differ in the following:
Z–sequential-oriented, property-oriented, uses set theory and logic.
VDM–process-oriented, model-oriented, uses the set theory and logics.
Larch – sequential-oriented, property-oriented, and uses algebra and logics
SDL-process-oriented, property-oriented
OCL—model-oriented, sequential-oriented, and uses set theory and logic.

VIII. Conclusion

The use of Formal specification language in software development process do not force you to invest a significant cost or time overhead across the entire development, rather it is helpful for understanding the system being developed and preventing error from being propagated through the early stage of lifecycle to the later one.

References

[1]. Nix, C.J. and Collins, B.P. "The use of Software Engineering, including the Z notation, in development of CICS."1988
[2]. Spivey, J.M.. The Z Notation: A reference manual. Prentice- Hall, 1989.
[3]. Warmer, J., and A. Kleppe. Object Constraint Language, Addis ion- Wesley 1998.
[4]. C. B. Jones, Systematic Software Development using VDM, 2nd ed. Englewood Cliffs, PH, NJ:, 19190.
[5]. M. Jackson, "Description is our business," in VDM '91: Formal Soft.Dev. Methods. New York: Springer-Verlag, LNCS-551, pp. 1-9.
[6]. Software Development using Specification and Description Language- Real time, Sharath Gopalappa.
[7]. J. Guttag and J. Horning, Larch: Languages and Tools for Formal Specification, Springer-Verlag; 1993.
[8]. Cooke, D., and Gates, A., "Language for the Specification of Software," Journal of System Software, 1996, pp 269-307.