FORMAL LANGUAGES: A COMPARISON OF PROCESS ALGEBRA AND MODEL ORIENTED APPROACH

Arun Kumar Singh¹ and Vinod Kumar Singh²

¹Electronics Engineering Department, IET, UPTU, Lucknow, India
²Professor, Electronics Engineering Department, IET, UPTU, Lucknow, India

ABSTRACT

A Formal language is an abstraction of the general characteristic of programming languages. It is used for system analysis, requirement analysis and system design. It describes the system at much higher level of abstraction than any programming language. Formal language can be categorized into model oriented, algebraic, process oriented, logical, imperative, and hybrid. Although there are several formal methods and languages, this paper emphasized basically on model oriented languages B, VDM, Z and process oriented languages CSP, CCS. Issues like abstraction, ambiguity, consistency, completeness, concurrency, looseness, readability and reusability have proven to be key for articulating this paper. This paper also addresses the scope of B and Event-B modeling along with other related technologies.

Keywords: Formal Language, Model Oriented, Process Oriented, Z, VDM, CCS, CSP, B and Event-B.

1. INTRODUCTION

To make sure that some solution solves a problem correctly, first of all it must be stated correctly [1]. Formal methods provide frameworks within which one can specify, develop and verify systems in a systematic, rather than an adhoc, manner [2]. Formal methods are system design techniques that use rigorously specified mathematical models to build software and hardware systems. Formal methods are used to reveal ambiguities, incompleteness and inconsistencies of the system. Design flaws can be detected in the early stage of system development process using formal methods hence increasing the possibility of reducing cost of testing and debugging phase. One tangible product of the application of a formal method is a formal specification. Formal method only determines what the problem is and how it will be said, is laid by the formal language.
A Formal language is an abstraction of the general characteristic of programming languages. It is the set of all strings permitted by the rule of formation. It can be used in the various phases of software development model. Requirements analysis, system analysis and system design, are the main phases of software development model in which it is used. Formal language can be categorized into model oriented, algebraic, process oriented, logical, imperative, and hybrid. Formal methods are mainly concerned with the specification and verification of the system. In this paper, we have tried to belittle precise on verification rather exploring the specification. Verification is nothing but the proof that specifies whether the program satisfies the specification or not. Verification can be done by either writing the program in actual close to the specification or automatically i.e. by the notion of model checking. Coming on to the specification, we have explored it in section 2. In section 3, the emphasis is on different formal specification styles. In section 4, some criteria have been elaborated and evaluated for some model oriented and algebraic approached formal languages, section 5 present general discussions on Event-B, a formal method supported by RODIN platform, an Eclipse-based IDE for Event-B, B methods and some related techniques, lastly section 6 concludes the paper.

2. SPECIFICATION

Specification is formulating the requirement of the system. In other words it is the description of desired system properties. It generally focuses on what rather than how. When specification is given in some natural language along with some figures, tables, examples or scenarios it is known as informal specification. Unified modeling language when used for specification with precise syntax and loose semantics it is known as semi formal specification. A formal model when used for specification with precise syntax, semantics and proof theory then it is known as formal specification. Generally speaking, a formal specification is the expression in some formal language and at some level of abstraction it is a collection of properties that the system should satisfy [1].

A formal specification lays emphasis more on completeness and consistency. It covers all situations and has no contradictions. Formal methods allow progressive refinement of an abstract specification into a more concrete specification using well-defined rules. The development of formal specification provides insights and understanding of the software requirement and design. A formal specification defines a system in an implementation independent way by describing its internal state in terms of abstract data type which is characterized only by the operation allowed over them.

A formal specification language provides the notation, set of objects and precise rule defining which objects satisfy each specification. The notation acts as a syntactic domain and the set of objects as semantic domain. A syntactic domain is the set of rules for determining whether the statement is syntactically correct or not. The semantic domain refers to the rules for interpreting sentences in a precise and meaningful way in the given domain. Hence we can say that formal method is based on some well defined formal specification language. Several specification languages with different semantic abstraction functions for a single semantic domain encourage complementary specification of different aspects of a system. Different kinds of formal specification language have different properties but they share some common properties and they are as follows:

- Reliability of the software increases
- Chances of error deduce
- Helps in software maintenance
- Helps in proper documentation
- Creating reusable modules
3. FORMAL SPECIFICATION STYLES

Formal specification techniques differ mainly by the particular specification paradigm they rely on. There are different categories of formal specification languages depending upon the approaches they adopt.

3.1 MODEL ORIENTED

In model oriented approach the specification of the system is done by constructing the mathematical model of the system. It specifies the admissible system state at some arbitrary time using mathematical entities like sets, relation and first order predicate logic. A system is seen as a set of state and operations that changes the state. An invariant is defined as a predicate that must be satisfied in all states. For each operation a precondition is defined and the operation can take place only if the precondition for that state is satisfied by the current state. For each operation a post condition is also defined and this predicate must be true at the modified state that results after the operation has taken place. The Vienna Development Method [3, 4], Z specification language [5, 6] and B [7] are the main model oriented languages.

3.2 ALGEBRAIC APPROACHES

In algebraic approach the admissible system behavior is specified in terms of properties or conditional equations that document the effect of composing function. In this approach one has to state explicitly which properties it wants the system to have, after which any model that satisfies the theory seems to be acceptable. The best known algebraic language is ACT ONE [8], ACT TWO [9] and the other being CASL [10], LARCH [11], OBJ [12].

3.3 TRANSITION BASED APPROACHES

The admissible system behavior is characterized by the required transition in state machines. There is always the corresponding output state for each and every input state, triggering event and Boolean expression. Promela [13], which is a language for the specification of interactive concurrent systems and Statechart [14] are the best known transition based language.

3.4 PROCESS ALGEBRA APPROACHES

The process corresponds to the behavior of the system. It is used to describe distributed and parallel system in algebraic manner where concurrency is the main issue. Calculus of communicating system (CCS) [15] and Concurrent sequential process (CSP) [16], designed to model asynchronous processes are the examples of the approach used by the process algebra. They are the best known process modeling languages.

3.5 LOGIC BASED APPROACHES

The system behavior in logic based approach is expressed by the logical expression. These logical expressions are based on traces of allowed operation call. Commonly used languages in logic based approach are: Temporal logic of actions (TLA) and Real time logic (RTL). TLA uses a logic that combines temporal logic and logic of actions for specifying and reasoning about concurrent as well as reactive discrete systems [17], TLA+ [18] is a specification language based on TLA. Real time logic (RTL) [19] is a first order logic with a predicate which relates the events of a system to their time of occurrence for the specification of real-time systems.
3.6 REACTIVE APPROACHES
The system behavior is specified in such a complete manner that its specification can run on any abstract machine as a structured collection of process. Petri nets [20] and SDL [21] are example of reactive approaches.

3.7 HYBRID APPROACHES
In hybrid approach the specification language describes the system in all stages of the development. The two best example of hybrid approaches are RSL [22] the specification language associated with the rigorous approach to industrial software engineering (RAISE) development methods [23] and LOTOS [24], a specification notation intended for the formal specification of open distributed systems, and in particular for those related to the Open Systems Interconnection (OSI) computer network architecture[25,26].

4. COMPARISON OF MODEL ORIENTED AND PROCESS ALGEBRAIC APPROACH

This section elaborated as well as evaluated the criteria or issues for which the model oriented and process algebraic approach can be compared and the best can be analyzed depending upon the issue. In process algebraic approach the CCS and CSP and in model oriented approach the Z, VDM and B have been considered (fig1).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Model Oriented Approach</th>
<th>Process Algebraic Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z</td>
<td>VDM</td>
</tr>
<tr>
<td>Abstraction</td>
<td>α</td>
<td>β</td>
</tr>
<tr>
<td>Ambiguity</td>
<td>γ</td>
<td>α</td>
</tr>
<tr>
<td>Consistency</td>
<td>γ</td>
<td>α</td>
</tr>
<tr>
<td>Completeness</td>
<td>γ</td>
<td>β</td>
</tr>
<tr>
<td>Concurrency</td>
<td>θ</td>
<td>θ</td>
</tr>
<tr>
<td>Looseness</td>
<td>β</td>
<td>β</td>
</tr>
<tr>
<td>Readability</td>
<td>β</td>
<td>α</td>
</tr>
<tr>
<td>Reusability</td>
<td>γ</td>
<td>α</td>
</tr>
</tbody>
</table>

Fig.1

The evaluation is done by considering the various applications of these two approaches. The symbol α is given to the strongest, β is given for strong and γ is given for adequate. The symbol θ is used for representing poorer in that criterion.

By evaluating the issues of these model oriented and process oriented languages it can be seen that some language is better in some way than the other language. However it can be concluded that depending upon the problem the approach is identified and further the language is chosen according to the suitability of the properties specified by the system.

5. THE B, EVENT-B AND RELATED TECHNIQUES

In this section, we briefly investigate how the B is related with other similar techniques, the tool support for development in B and Event-B.

5.1 THE B METHOD
The B method is a state-based method that involves the integration of set theory, predicate calculus and generalized substitution language. B and Z [5] both have evolved from the same root and are based on ZF set theory but then there are number of differences. The state changes in Z are
expressed by the use of before and after predicates, whereas in B the predicate transformer semantics of B allow a notation closer to programming.

Invariants in Z are encapsulated into the operation description which alters its meaning. The invariants in B are checked against the state changes described by operation and events to ensure consistency. The logical properties of pre-condition and guards make a more careful distinction in B which is not clearly distinguished in Z.

The refinement calculus used in B for defining the refinement between models in the event-based B approach is very close to Back’s action systems.

In TLA the temporal operator’s semantics is expressed over traces of states whereas in B the semantics is expressed in the weakest precondition calculus.

TLA+ [18] can be compared to B, since it includes set theory with the € operator of Hilbert. With respect to safety property both semantics are equivalent but the trace semantics of TLA+ allows an expression of fairness and eventuality properties that is not directly available in B.

VDM [3] is a method with similar objectives to classical B. Like B it uses partial functions to model data, which can lead to meaningless terms and predicates e.g. when a function is applied outside its domain. VDM uses a special three valued logic to deal with undefinedness.

ASM [27, 28] and B share common objectives related to the design and the analysis of software and hardware systems. ASM and B both methods provide the link between human understanding and formulation of real-world problems and the deployment of their computer-based solutions [29].

B is based on set theory and ASM is based on the algebraic framework with an abstract state change mechanism. An Abstract state, the signature and some finite set of rules define the abstract state machine. The rules provide an operational style that is very useful for specifying the model and the mechanism for programming.

The B formal approach supports a step-wise development from basic abstract specifications to a detailed design of a system in the refinement steps. Through refinements the design of the detailed system is verified which conforms to the abstract specification.

Consistency checking and refinement checking are two main proof activities in B. Consistency checking is used to show that the operations of a machine preserve the invariant while refinement checking, is used to show that one machine is a valid refinement of another.

The B tools provide significant automated proof support for generating the proof obligations and discharging them. These proofs help us to discover new system invariants providing a clear insight into the system and augment our understanding of why a design decision should work. They also help us to understand the complexity of the problem and the correctness of the solutions.

These activities are supported by industrial tools, such as Click’n’Prove [30] B tool and the B-toolkit [31].

If each of proof obligations generated by B-tools is proved, then the machine is consistent and refinement is correct (in the case of refinement checking). The B-tools have an interactive prover and an automatic prover. Normally the more complex proof obligations are not proved automatically and need to be proved interactively. The tools also provide automatic translation of low level B specifications into executable code.

Another available tool ProB [32] is a model checker and an animator for the B-Method. ProB allows interactively/automatically animation of B specifications, and can also be used to systematically model check a B specification check a specification for errors.

5.2 EVENT-B

Event-B [33] has been developed from the B-Method using many ideas of Action Systems [34]. The system behavior in Event-B is modeled by a collection of state variables and a collection of
guarded actions that act on the state variables (action system approach), also the mathematical language for defining state structures and events is typed set theory (B method). The B Method is aimed at software development; Event-B is aimed at system development. It is a formal technique that consists of describing rigorously the problem in the abstract model, introducing solutions or design details in refinement steps to obtain more concrete specifications, and verifying that the proposed solutions are correct. When modeling a system aspect of the non-software parts of the system and its environment may also be included. This structure allows, using Event-B for varying modeling domains, e.g., reactive, distributed and concurrent systems [35], sequential programs [36], or electronic circuits [37], not being constrained to semantics tailored to a particular domain.

The Rodin platform [38], an Eclipse-based IDE for Event-B, provides effective supports for refinements and mathematical proof. It is a powerful and effective toolset for Event-B development. It is extensible and configurable and supports a seamless integration between modeling and proving.

4. CONCLUSION

In this paper the first focus was on formal method and then formal languages came into seen. This paper also explores the scope of Event-B modeling. Two application of formal language came into picture i.e. specification and verification. Specification was further explored with different formal specification techniques like model oriented, process, algebraic, transition, reactive, hybrid, Model oriented approach (B, VDM and Z) and process algebraic approach (CCS and CSP) is compared on basis of abstraction, ambiguity, consistency, completeness, concurrency, looseness, readability and reusability. Event-B is one of the formal methods which is successfully used to model different complex systems and can be used for development of distributed and safety critical system. It is supported by an Eclipse-based IDE, RODIN, which provides tools for working with Event-B models.

ACKNOWLEDGEMENTS

The author's acknowledge the valuable suggestions of Er. Brijesh Pandey, department of Computer Science and Engineering, GITM, Lucknow and Prof. Girish Chandra, Department of Computer Science and Engineering, IET, UPTU, Lucknow.

REFERENCES


