Formal Specification Concepts in Critical Analysis

Prakash S. Teltumde, PawanKumar I. Patil and Vishal R. Ambhire

M. Tech. Research Scholar, Department of Computer Science, VJTI, Mumbai, India

pst205@gmail.com, pavanipatil1@gmail.com, vishalambhire@gmail.com

Abstract

With the growing significance of computer systems within industry and wider society, techniques that assist in the production of reliable software are becoming increasingly important. The complexity of many computer systems requires the application of a battery of such techniques. One of the most promising approaches are formal methods. The paper gives an idea of formal specification and use of Z. It also contains use of formal specification in critical system may be in air traffic control, prevent mobile attacks. Also case study have been explained in detail. This paper describes the conceptual and formal models.

Keywords: ATC (Air Traffic control), mobile based agent, Z notation, Z/EVES tools

I. INTRODUCTION

Formal specification is a technique for the unambiguous specification of software. Formal specifications use mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved. The use of a formal specification or model eliminates ambiguity and thus reduces the chance of errors being introduced during software development. Where a formal specification exists, both the source code and the specification may be seen as formal objects that can be analyzed and manipulated.

The objective of formal specification techniques are:

To describe the use of algebraic techniques for interface specification.
To describe the use of model-based techniques for behavioural specification.
To explain why formal specification techniques help discover problems in system requirement.

Problem with Natural language:
Natural language is the language of choice for the requirements specification. Natural language specifications are informal and usually contain ambiguities. Natural language does not serve as a common means of communication between the client and software engineering.

In Natural language when person read a document the person interpret document in his or her words and it varies from person to person.

It does not serve good means of modularization of software, because of this drawbacks look at method that can give more structure or formal means of writing down specification of software.

Formal specification languages:

Algebraic: Sequential - Larch, OBJ
Concurrent - Lotos, Brinkema

Model based: Sequential - Z notation, VDM, B
Concurrent - CSP, Petri Nets

Here we present an informal statement - A Library Example:

- A book can either be on stack, on reserve, or loaned out.
- If a book is in the stack or on reserve, then it cannot be requested.

We want to: Formalize the concepts and statements, Prove some theorem to gain confidence that the specification is correct.

Library example - Formalization

Let’s formalize some concepts
- S: the book is in the stack requested
- R: the book is on reserve
- L: the book is on loan
- Q: the book is requested

Now through formal notation shown as

A book can either be on a stack, on reserve, or loaned out

- S ^ ¬(RvL)
- R ^ ¬(SvL)
- L ^ ¬(SvR)

If the book is in the stack or on reserve then it can be requested
- Q => (SvR)

Now to prove L => ¬Q

We assume first by negation
- (L => ¬Q)
- (¬L v ¬Q) rewriting

L ^ ¬Q - Demorgans law
L => Simplifying
¬ (SvR) ----L => ¬ (SvR)
¬ (SvR)
=> Q
=> SvR
\[ \neg (L \Rightarrow \neg Q) \]
\[ L \Rightarrow \neg Q \]
Hence proved

II. Z NOTATION

Z Notation is a formal specification technique based on fairly standard mathematical notation. It is based on Typed Set Theory which avoids certain technical difficulties; It is designed to be added as notation in specification documents as way to complement the prose. The most popular implementation is based on Latex. Z decomposes specifications into manageable sized modules, called schemas:

Schemas are divided into 3 parts:

State
A collection of state variables and their values

There are also some operations that can change its state.

Here are the main steps for the specification:

Create a model of the system using discrete mathematical structures (primarily sets), together with predicate logic that specifies the relationships between the structures.

Using logic show how the model will be changed for each operation that the system can perform expressed in terms of ‘what’ changes should occur rather than ‘how’ they should come about.

The process is repeated to gradually refine the specification.

There are three basic elements in the vocabulary of Z:

Sets
Set Operations
Logic

A. Sets
Collection of elements in which no sense of ordering of elements is involved.

\[ \text{AIRCRAFT} \] the set of all aircraft in a given airspace

\[ \text{RUNWAY} = \text{main} | \text{north} | \text{west} \]

Variables of set types may be declared, in lowercase italics

\[ \text{Available Runways} \]

\[ \text{AIRCRAFT} \]

\[ \text{gate Numbers} = \{1\ldots31\} \]

B. Set operation

Set membership operator, which returns Boolean depending on whether the element is or is not a member of a set.

Cardinality which returns the number of elements in a set.

C. Logic:

Not, And, Or, Implies, Equivalence (if and only if), For all (universal quantifier), There exists (existential quantifier).

There exists exactly one, It is true that, Such that.

Z specification consists of four sections:

1. Given sets, data types, and constants
2. State definition (shown as Z schema)
3. Initial state
4. Operations (shown as Z schema)

Now consider the elevator problem

Given Sets

A Z Specification begins with a list of given sets, sets that need not be defined in detail

✓ Names appear in brackets

✓ Here we need the set of all buttons

✓ Specification begins

[Button]

State Definition

Z specification consists of a number of schemata. The schema consists of

Schema name

Group of variable declarations, also called signature or declarations

List of predicates that constrain values of variables

\[ S \]

declarations

predicates

Four subsets of Button

✓ The floor buttons

✓ The elevator buttons

✓ buttons (the set of all buttons in the elevator problem)

✓ pushed (the set of buttons that have been pushed)

Initial State

State when the system is first turned on

\[ \text{Button_Init} \equiv [\text{Button State} \ | \ \text{pushed} = \emptyset] \]

Operations

If elevator arrives at a floor, the corresponding button(s) must be turned off

The solution does not distinguish between up and down floor buttons.

Case Study :1

Air Traffic Control System Using Formal Methods

Formal methods are used in safety critical system of air traffic control known as (ATC). Here Automation of air traffic control system is one of the most challenging because it is one of the highest critical system. Even if minor/small error may cause a major risk to the human life. Formal methods are rapidly growing technology in sense of mathematical notation and it reduces ambiguities present in the specification. Here the aim is to explain, (i) to model highly critical system of ATC using formal methods, (ii) to apply graph theory in representation of formal ATC Model. Here ATC system been modelled as a directed graph with zones (airspace segment) as a set of
nodes and airway segments connecting them as set of edges. Here ATC is modelled using Z notation.

1.1 Air traffic Control in Real world
The International Civil Aviation Organization (ICAO) is one of the global forums which has the goal of safe and efficient air transportation. In this ICAO airspace is classified into controlled and uncontrolled. A controlled airspace is being monitored by a team of controllers. Controlled airspace is a three dimensional area where active ATC services are provided for aircrafts flying in it. There is a large network of aircrafts which allows unidirectional or bidirectional flow of aircrafts.

1.2 ATC Model represented in Graph Theory
The ATC model is transformed as a directed graph. The controlled airspace is further divided into smaller airspace segments or Zones. If there is an airway segment connecting two zones, it means the two zones are connected and aircrafts can fly directly between them. Therefore, set of all zones of the airspace represents nodes of the graph and the set of airway segments connecting them as arcs. The direction of arc indicates the direction of flow of traffic. Figure 1 shows ATC Model in graph theory representing six Zones $z_1, z_2, z_3, z_4, z_5$ and $z_6$ and ten airway segments represents interconnections between them.

![Figure 1: ATC Model in graph theory](image)

As shown in the figure that arc between $z_1$ and $z_2$ is unidirectional so aircraft can fly from $z_1$ to $z_2$ and not vice versa while $z_2, z_6$ is bidirectional, hence can move in both the direction.

1.3 ATC System using formal methods
The ATC system modeled as directed graph can be formalized using Z notation, as known Z notation is based upon set and mathematical logic. The major component of formal ATC are shown as

![Figure 2: ATC system using formal components](image)

**Static Topology:** The Static topology aim is to perform intended task. They are the physical layout of the component. Zone, Airport Connections represents the static Topology of an ATC System Model.

Zone: Each Zone is defined as a three dimensional entity of Airspace. A zone is a collection of abstract type points.

Zone $== P \text{ Point}$

Connection: The connectivity between two zones is called as connection. It not only depicts the connection but represent direction of flow too. Here zone is not connected with itself.

Connection $== \{ z_1, z_2: \text{ Zone } z_1 \neq z_2, (z_1, z_2) \}$

Static Model: This is referred to as connection of all the zones

[Runway]

<table>
<thead>
<tr>
<th>StaticTopology</th>
</tr>
</thead>
<tbody>
<tr>
<td>connections $\subseteq P \times P$ Connection</td>
</tr>
<tr>
<td>Airport: Runway $\rightarrow$ Zone</td>
</tr>
</tbody>
</table>

| Zone $z_1, z_2 \in \text{ Zone}$ |
| $(z_1, z_2) \in \text{ connections}$ |

Here it shows that only one zone is assigned one runway and not two zones have same runway. Hence Airport is modeled as a total injective function of Runway and Zone.

**Invariants:**

The Connections relation is asymmetric. It means, an aircraft can move directly from one zone to another it may or may not be possible for it to move in the opposite direction.

**Network State:**

The dynamicity of aircrafts flying within the zones is known as Dynamic Topology, which is known as Network State of ATC system Model. It states that only one aircraft at one zone at a time and every aircraft is assigned an unique identification mark represented by abstract data type AircraftId. If an aircraft resides in a zone, state is marked as OCCUPIED else the state is CLEAR.

[Aircraft]

State $== \text{ CLEAR} \setminus \text{ OCCUPIED}$

Invariants: It shows that only one zone will have one aircraft and not two zones have the same aircraft at the same time.

**Aircraft:**

Each Aircraft is assigned a unique identification mark called AircraftId. Here each AircraftId is assigned to only one aircraft and no two id can be assigned to aircraft. The variable Aircrafts is the total function of Aircraft and AircraftId.
Now source depicts the zone from which the aircraft has flown and destination depicts the aircraft intended to land. The speed, altitude, heading, speed limit, and altitude limit of an aircraft are represented as natural numbers in variables current and it represents the zone occupied by aircraft at particular time.

**Invariants:**
1. An Aircraft cannot have the same zone as its source and destination
2. The current speed of an aircraft should not exceed Aircraft’s speed limitation.
3. The current altitude should not exceed maximum altitude limitation of an Aircraft
4. The heading of an aircraft should not be greater than 360 degrees.

**Controller:**
A controller monitors and directs an aircraft within the airspace zones assigned to it. It also has unique identification mark known as Control Id, assigned to each controller. The variable Controls is the total function of Controller and ControlId.

\[
\text{Controls} = \text{ControlId} + \text{Controller}
\]

The variable declared here are

<table>
<thead>
<tr>
<th>Static Topology</th>
<th>Network State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Zones</td>
</tr>
<tr>
<td>Zones</td>
<td>Aircraft</td>
</tr>
<tr>
<td>State</td>
<td>Sector states</td>
</tr>
<tr>
<td>Zone States</td>
<td>Aircrafts</td>
</tr>
<tr>
<td>Capacity</td>
<td>Controls</td>
</tr>
<tr>
<td>Control</td>
<td>Controller</td>
</tr>
</tbody>
</table>

The variable declared here are

<table>
<thead>
<tr>
<th>Static Topology</th>
<th>Dynamic Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircrafts</td>
<td>Aircrafts</td>
</tr>
<tr>
<td>Controls</td>
<td>Controls</td>
</tr>
<tr>
<td>ATC System</td>
<td>Static Topology</td>
</tr>
</tbody>
</table>

Here as shown in the declaration Static Topology and Dynamic Topology is included with all its declarations and
constraints but with no state change privileges granted on this schema.
The set of aircrafts is represented by variable aircrafts. The set of controllers is represented by variable controls.

**Invariants:**
1. There must be state value between the connections of zones defined in Static Topology.
2. The destination of all the aircrafts flying within airspace must belong to the connection of zones defined in Static Topology.
3. All the aircraft in the ATC system must be under the control of the controller.

### 1.5 Analyzing Formal ATC Model Using Z/EVES

Z/EVES is one of the powerful tool for analyzing Z specifications. This tool can be used for parsing, type checking, domain checking, schema expansion, precondition calculation, refinement proofs, and proving theorem.

#### Case Study: 2

**A Formal Approach to prevent Attacks on Mobile Agent Systems**

Mobile agent (MAg) can be defined as an active entity which is able to migrate from one machine to another in order to satisfy requests made by their owner. Mobile agent technology offers a new computing paradigm in which a program, in the form of a software agent, can suspend its execution on a host computer, transfer itself to another agent-enabled host on the network, and resume execution on the new host. A host machine can contain one or more agent systems. An Agent System (AgS) provides the runtime environment which offers the basic functionalities for MAg management such as: agent transport, communication, security, etc. Within the same AgS can reside one or several societies of Service Agents (SAg) that provide several application services.

#### 2.1 Security Threats and Model

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. There are a variety of ways to examine these classes of threats in greater detail as they apply to agent systems. Here, we use the components of an agent system to categorize the threats as a way to identify the possible source and target of an attack. Mobile agents simply offer a greater opportunity for abuse and misuse, broadening the scale of threats significantly. A number of models exist for describing agent systems; however, for discussing security issues it is sufficient to use a very simple one, consisting of only two main components: the agent and the agent platform. Here, an agent is comprised of the code and state information needed to carry out some computation. The agent platform provides the computational environment in which an agent operates.

**Figure 4: Agent system model**

Four threat categories are identified: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform, and other entities attacking the agent system. The last category covers the cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, since these attacks are primarily focused on the communications capability of the platform to exploit potential vulnerabilities. The last category also includes more conventional attacks against the underlying operating system of the agent platform. In this security model we also concern about the security policies which control the behaviors in a Mobile Agent System(MbAS). One of the most important issues in MbAS is the security aspect which must appear in both MAg and AgS, thus there must be various types of security measure. Both AgS and MAg should have defined the security policies. Thus, a secure entity SEntity can be either a MAg or an AgS.

**SEntity ::= MAg | MobileA | AgS | ASystem**

In order to express various kinds of security rules we define constructs for Confidentiality, Integrity, Accountability, Availability.

**SConstruct ::= Conf | Integ | Acco | Avai**

Formally, we define a security rule with the following schema:

<table>
<thead>
<tr>
<th>SRule</th>
<th>Name : Attribute</th>
<th>Type : SConstruct</th>
<th>Subject : SEntity</th>
<th>Target : STarget</th>
<th>Constraints : Condition</th>
<th>Actions : PAction</th>
</tr>
</thead>
</table>

So, in the declaration part, we declare: the type of the security rule (Type), the secure entity (Subject) on which we apply the so-called rule, the target object (Target), the set of constraint (Constraint) under which we apply the
rule, the non empty set of actions (Actions) that are imposed by the rule to reach the desired behaviour, the period of validity (period valid) which specifies the time instants at which the rule is applied. Thus Z notation allows us to go to the depth of the security policies specification.

2.2 Possible Attacks in MbAS
There are list of security requirements they are four main security requirements: confidentiality, integrity, availability, and accountability. The users of agent and mobile agent frameworks also have these same security requirements.

Confidentiality
Any private data stored on a platform or carried by an agent must remain confidential. Agent frameworks must be able to ensure that their communications remain confidential. Eavesdroppers can gather information about an agent's activities not only from the content of the messages exchanged, but also from the message flow from one agent to another agent or agents. Monitoring the message flow may allow other agents to infer useful information without having access to the actual message content. Mobile agents may also want to keep their location confidential, and they communicate through a proxy whose location is publicly known if the agents want to conceal their presence on a particular platform. Agents must be allowed to decide if their presence will be publicly available through platform directories, and platforms may enforce different security policies on agents that chose to be anonymous. Since audit logs maintain a detailed record of an agent's activities on the platform, the contents of the audit log must also be carefully protected and remain confidential. Access to the audit logs must be restricted to authorized administrators. A mobile agents' audit log may be distributed across several security domains, each having a different audit policy, so some agents may want to carry certain parts of their audit log with them for future reference.

Integrity
The agent platform must protect agents from unauthorized modification of their code, state, and data and ensure that only authorized agents or processes carry out any modification of shared data. The agent itself cannot prevent a malicious agent platform from tampering with its code, state, or data, but the agent can take measures to detect this tampering.

Accountability
Each process, human user, or agent on a given platform must be held accountable for their actions. In order to be held accountable each process, human user, or agent must be uniquely identified, authenticated, and audited.

Availability
The agent platform must be able to ensure the availability of both data and services to local and remote agents. The agent platform must be able to provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required.

2.3 Formal Representation of DoS Attack
Mobile agents can launch denial of service attacks by consuming an excessive amount of the agent platform's computing resources. These denial of service attacks can be launched intentionally by running attack scripts to exploit system vulnerabilities, or unintentionally through programming errors. The DoS attack can be divided into 3 types
- **DoS Agent-to-AgentSystem**: It is when the agent tries to degrade the performance of the hosting system and which this makes the agent service unavailable to the incoming agents, it also tries to consume excessive amount of agent system’s computing resources.
- **DoS Agent-to-Agent** (DoS AA): This aim at making agent to unable to work properly, the malicious mobile agent launches the DoS attack by sending sending continuously the same messages or by sending spam messages to other agent.
- **DoS Agent System-to-Agent** (DoS SA): This attack is launched by ignoring the request of authorized agents and putting them, constantly, into waiting list.

2.4 Formal Specification of DoS AS Attack
The formal specification of DoS AS attack is presented by the DoS AS schema and the specification of DoAS attack should be consistent with the concepts presented in mobile agent model and security model. So here there is the attacker entity (attacker) which is a mobile agent (MobileA), the target entity (target) which is an agent system (ASystem), the set of all possible pairs (condition, action) which cause DoS AS attack.

The agent platform must be able to provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required.
of so, greater than the maximum value required by the target system \((\text{maccept value(target, so, c)})\).

Now the Z/EVES tool can be used to check the consistency of constraints specified in the predicate part, but they aren’t that helpful. So we prove this by initialization theorem. So, suppose that State describes the state of the system, and that StateInit characterises the initial state. In order to check the consistency of the DoS AS schema Let’s assume that the initial state consists of an attacker Ag, a target system As and launched when Ag execute the action ac under cc11 condition

\[
\begin{aligned}
\text{Ag} & : \text{MobileA} \\
\text{As} & : \text{System} \\
\text{cc11} & : \text{condition} \\
\text{ac} & : \text{Action} \\
\{((\text{cc11, ac})) \} \in \text{conditions} \rightarrow \text{Action} \\
\forall \text{so} : \text{Object}; t_2 : \text{Task}; v : \text{Value}; c : \text{Constraint} \\
& \text{ac} = \text{actual} v(\text{value}(\text{Ag, so, t2}) \\
& \text{t} \in \text{ac} \land \text{so} \in \text{ac} \land \text{cc11} = \text{val}(\text{val}, \text{val}, \text{val}(\text{maccept value(As, so, c)})) \\
\forall \text{so} : \text{Object}; t_1, t_2 : \text{Task} \\
& \text{t} \in \text{agree} \text{action(As, so, ag)} \\
& \text{so} \{t_1\} \in \text{authority(As, Ag, ac) } \\
& t_1 = t_2
\end{aligned}
\]

Then, we describe the initial state of DoS AS by the following schema DoS AS int:

\[
\begin{aligned}
\text{DoS AS int} \\
\text{DoS AS} \\
\text{attacker} = \text{Ag} \\
\text{target} = \text{As} \\
\text{attack} = \{((\text{cc11, ac})) \}
\end{aligned}
\]

Thus, we prove by the initialization theorem Consistency DoS AS the consistency of DoS AS schema:

\[
\text{theorem Consistency DoS AS} \\
\text{DoS AS \& DoS AS int}
\]

 Proposed: We have taken the small case study of library where book to be requested or reserved been mentioned there and here informal or natural language been given and we have proved this using formal specification concept, which will be also helpful for understanding the software and viceversa.

 Conclusion: Here this paper give us idea of formal specification concept in critical analysis, it clearly shows the advantage of using formal specification instead of natural language. As seen in air traffic control a graph theory has been shown because it clearly mention all the connectivity problem regarding different zones. And second case we have shown attacks on mobile agents, a security policy has been shown to prevent those attacks. Here a formal conceptual models has also been shown along with some theorem proving and we have used Z/EVES tool for type checking and the theorem proving.

**REFERENCES**

1. Maryam Jamal, Department of Computer Science, Nazir Ahmad Zafar Department of Computer and Information Sciences.

2. David Leadbetter, Peter Lindsay, Andrew Neal, and Mike Humphreys, "Integrating the Operator into Formal Models in the Air-Traffic Control Domain".


