CHAPTER VI

DESIGNING INTERFACE

REQUIREMENTS
6.1 INTRODUCTION

In section 2.2.4, we have discussed that it is better to incorporate interface requirements within the functional requirements model than to handle it separately. In addition, formal method for interface requirements also makes it possible to verify the design using model-based test cases. This chapter presents a systematic approach for analysis and design of interface requirements by introducing a new methodology to use formal methods in a uniform way for functional as well as interface requirements (Sengupta & Dasgupta, 2013 and Sengupta & Dasgupta, 2015).

Software requirements are conventionally classified as functional and non-functional requirements. Functional Requirements define the actions that must take place in the software in accepting and processing the inputs and generating the outputs with or without support from the library functions, standard database etc. (IEEE, 1998). Functional requirements are easier to specify using formal models because such requirements can be broken down into a sequence of sub requirements and each such sub requirements can be expressed in some mathematical and/or logical form, which is also easier to code and test. Our discussion in RE (Chapter III), Architectural design (Chapter IV), and detailed design (Chapter V) are so far has been confined within functional requirements. However, proper analysis of non-functional requirements is also equally important to any software development process. In contemporary web based LMS developments, user interface requirements are predominant among the other non-functional requirements. The end users’ influence in eliciting and controlling such requirements makes it challenging to the designers to map these interface requirements with functional requirements. As a result, we often use different techniques for different categories of requirements, like use case and activity diagram for functional requirements, component diagram for architectural requirements, ERD and Class diagram for data requirements and layout, sketch, and screen shot for interface requirements. However, interface requirements, if handled separately from functional requirements, bring many hazards to the later integration stages. It may lead to problem related to traceability, inconsistency, and missing interfaces. Introduction of a uniform formal method to all types of requirements can solve these issues but since interface-requirements for LMS are
usually exposed to the event driven environment of web application, it is harder to define them in a formal way.

We propose some add-ons to VDM-SL syntaxes to cover the interface-requirements of a web-based application. A formal specification not only ensures correct understanding of the requirements but also supports for model based testing on the design artifact against the requirements. A pertinent framework is proposed to support the transformation of the conventional SRS to a design specification, and construction of FSM model from the design artifact for verification of the design with the SRS.

6.2 PROPOSED FRAMEWORK

The proposed framework [Figure 6.1] incorporates different components of SE methods like the SRS, the Analysis model, the verification model and the design specification. The first step in our approach is to build a formal analysis model, which is composed of VDM-SL descriptions for the requirements at two different abstraction-levels, Level-1 and Level-2. The level-1 specification is suitable for requirement analysis whereas the level-2 specification is more detailed and suitable for the design model. We assume that the functional requirements are specified within the SRS in conventional manner using NL text and the interface-requirements are represented by graphics, drawing or screen-shots. Then we create the level-1 description by translating the SRS with functional requirements
CHAPTER VI- DESIGNING INTERFACE REQUIREMENTS

into VDM-SL. Next, the level-1 specification is converted into level-2 specification using a proposed technique (specified in section 4). The interface-requirements are incorporated within the level-1 specification in an abstract way and they are more detailed in the level-2 specification during the conversion. The objective of the conversion of level-1 specification to level-2 specification is to produce a formal design specification from a formally written requirement specification. The level-2 specification is less abstract and considered as the formal representation of the design of the system. The level-2 specification could be alternately constructed by translating a design artifact, built directly from the SRS using conventional methods, into a VDM-SL specification. However, this path is not explored in this work, since it is much harder to develop a conversion technique for conventional design element to a formal specification than converting the level-1 specification to level-2 specification. We propose a set of data types and operations, for level-1 and level-2, which helps in formal modeling of the interface-requirements along with the functional requirements within the specifications. Next, we propose a technique to build an FSM model from the level-2 specification and generate test cases from that. Finally, we compare the results with the SRS in order to verify the design specification.

6.3 PROPOSED ADD-ONS TO TWO LEVELS OF VDM-SL

In our approach, formal modeling is first applied at the requirements specification level and then it is refined further at the design level. The first level involves translating the system requirements into formal notations in terms of the high-level entities, their properties, and interactions, without attempting any technical aspects of implementation. In the design-level specification, formal methods are used to describe the information on the objects to be included in the system, their properties and operations, and some details on how they would work. Unlike the previous level, level-2 specification gives at least a loose description about the relationship between the implementation and the formal model (Geer, 2011). Subsequently, we propose two levels of data types and operations as add-on to VDM-SL library; one for abstract level and other for more-detailed level. Table
CHAPTER VI- DESIGNING INTERFACE REQUIREMENTS

6.1 and Table 6.2 represent add-ons for the level-1 specification whereas Table 6.3 and Table 6.4 depict the same for the level-2 specification.

### 6.3.1 Add-on for level-1 specification

Table 6.1: Level-1 data types

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Symbol</th>
<th>Declaration</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-Page</td>
<td>WP</td>
<td>p:WP</td>
<td>Represents any web page visible in single window frame in the browser</td>
</tr>
<tr>
<td>link</td>
<td>LK</td>
<td>l: LK</td>
<td>Represent the link through which we can navigate from one page to another</td>
</tr>
<tr>
<td>input</td>
<td>IP</td>
<td>b: IP</td>
<td>Represent how user enter value to the system</td>
</tr>
<tr>
<td>output</td>
<td>OP</td>
<td>d:OP</td>
<td>Represent how system shows information to user</td>
</tr>
</tbody>
</table>

Table 6.2: Level-1 operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Use</th>
<th>Return type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type binding</td>
<td>::</td>
<td>b:IP::Z</td>
<td>-</td>
<td>b is a input type variable that takes integer value</td>
</tr>
<tr>
<td>Include</td>
<td>∑</td>
<td>P ∑ b</td>
<td>WP</td>
<td>The web page P contains the input variable b</td>
</tr>
<tr>
<td>Set-link</td>
<td>∏</td>
<td>l ∏ P2</td>
<td>LK</td>
<td>The link l is connected with the page P2</td>
</tr>
<tr>
<td>Navigate</td>
<td>▶</td>
<td>P1 ▶ P2</td>
<td>WP</td>
<td>The web page P1 navigates to the web page P2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[possible if at least one link of P2 exists in P1]</td>
</tr>
</tbody>
</table>
While modeling interface requirements, we must consider the web based nature of contemporary LMS software. Any web-based application can be seen as a collection of web pages where web pages are connected with links. Links can be established in two different ways: i) static links are html elements $\text{href}$ or $\text{form}$; ii) a dynamic link can be established by server-side $\text{redirection}$ script or by client-side $\text{form submission}$ script. In the level-1 specification, both the cases are represented by the data type $\text{link}$ ($LK$). The navigation operator $\text{Navigate}$ works only if there is an already established link between the web pages. The system may impose restrictions on the allowed data values of the input parameter. This can be realized at the time of binding of the basic data types of VDM-SL with the input data types. For example, $b: IP::Z$ represents $b$ as an input type variable that takes only integer ($Z$) values. The system passes information back to user by writing on the visible section of the web page. In our approach, we incorporate this by simply assigning the value to the variable of output data type.

6.3.2 Add-on for Level-2 Specification

Table 6.3: Level-2 data types

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Symbol</th>
<th>Declaration</th>
<th>Remarks and Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web-Page</td>
<td>WP</td>
<td>p:WP</td>
<td>Creates an web page</td>
</tr>
<tr>
<td>ServerForm</td>
<td>SF</td>
<td>f:SF</td>
<td>Carries data to server-end</td>
</tr>
<tr>
<td>ClientForm</td>
<td>CF</td>
<td>c:CF</td>
<td>Carries data to client-end</td>
</tr>
<tr>
<td>Hyperlink</td>
<td>HL</td>
<td>l: HL</td>
<td>$l = \text{“kolkata”}$ means the link is visible as “kolkata”</td>
</tr>
<tr>
<td>TextBox</td>
<td>TB</td>
<td>t: TB</td>
<td>$b= \text{“Hello Kolkata”}$ means the textbox contains “Hello Kolkata”</td>
</tr>
<tr>
<td>CheckBox</td>
<td>CB</td>
<td>c: CB</td>
<td>$c= \text{“checked”}$ means the Check box is ticked, the other possible value is “unchecked”</td>
</tr>
</tbody>
</table>
### Chapter VI - Designing Interface Requirements

| **RadioButton** | RB | r: RB | r= “on” means the Radio button is selected, the other possible value is “off” |
| **Dropdown**   | DD | d: DD | d=<”kolkata”| “Delhi”> means the dropdown has two values |
| **PushButton** | PB | b:PB | b = “OK” means the label on the button is “OK” |
| **SubmitButton** | SB | s:SB | s= “Submit Data ” means the label on the submit button is “Submit Data” |
| **MessageBox** | MB | x: MB | x= “Hello Kolkata” means the text will be displayed in a message box |
| **PageScan**   | PS | s:PS | Represents how input data are fetched and only one PS is allowed in a page |
| **PageWrite**  | PW | p:PW | Represents how to write in a web page and only one PW is allowed in a page |
| **ScriptedLink** | SL | s:SL | Represents s as a client-post or server-redirection link |

**Table 6.4: Level-2 operations**

<table>
<thead>
<tr>
<th><strong>Operator</strong></th>
<th><strong>Symbol</strong></th>
<th><strong>Use</strong></th>
<th><strong>Return type</strong></th>
<th><strong>Remarks</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Ω</td>
<td>pw Ω “error”</td>
<td>WP</td>
<td>The text “error” will be displayed in the web page that contains pw</td>
</tr>
<tr>
<td>Retrieve-Value</td>
<td>⊥</td>
<td>P ⊥ s where, s: PS s ⊥ f,t where, f: SF</td>
<td>Token</td>
<td>The value inside the textbox t is now available to page P either by client form c or by server form f. Other than text box it can</td>
</tr>
</tbody>
</table>
In level-2, we extend the basic input and output data elements as the control-objects of web based applications. The users interact through the users’ interface controls like textbox, radio button, check box, dropdown box, hyperlink, push button and submit button. Parameters passed by the user are collected at the client side by a container called ClientForm and at the server-side by ServerForm; therefore, all the control-objects should be placed inside either a ClientForm or a ServerForm. In the level-2 specification, the data type Hyperlink (HL) represents the static link. A dynamic link is established either by the ScriptedLink data type or by the Submit operator. Data values can be passed from one web page to another in two different ways. The implicit way is to put the input control inside a ServerForm and add the SubmitButton in the web page; all the variables available in the form will be intrinsically available to the target page when the Submit operation is executed. The other way is to pass the parameters explicitly using the pass_param operation with the Hyperlink. In order to handle the event based nature of the web based systems we propose intervention function. It is invoked by the system’s environment whenever there is any user’s intervention in the
system. The reason could be client-side events like clicking button, submitting, selecting from the dropdown list, etc. or server-side events like redirecting, session expired, etc. The keyword *intv* is used to define an intervention function. The syntax is: *intv<system> (ActiveControl, ActiveEvent) {}*

It allows two parameters, the first one is of *ActiveControl* type and the other is of *ActiveEvent* type. The parameter values are dynamically set by the system’s environment based on the intervention type. For instance, if a submit button is clicked then the values are set to *SB* and *Click* respectively. A hierarchical structure of the proposed abstract syntaxes and their relationships, used in the two-leveled specification, is shown in Figure 6.2.

![Figure 6.2: Hierarchical structure for the add-on components](image)

#### 6.4 CONVERSION OF LEVEL-1 TO LEVEL-2 SPECIFICATION

**Definition1. Interface Variable**: The web control objects are included in the VDM-SL specification as interface variables. Interface variables are considered as state variables and any changes made to them results in a state change.
Next, we propose a conversion technique for level-1 to level-2 within the VDM-SL specification.

**Step I:** All the *inclusive* type operations are converted into *exclusive* type operations.

**Step II:** All the *input* and *output* controls of level-1 are refined into corresponding web control objects in level-2 in accordance with the interface specification.

**Step III:** *Control-objects* are added as interface variables under the state variable section.

**Step III:** *PageScan* and *PageWrite* objects are incorporated to access and to response to user’s data respectively.

**Step IV:** For all the navigations that follow a change in *ext* type variable, implicit or explicit parameter is added to the navigation.

**Step V:** Pertinent *events* are introduced in the level-2 specification by overriding the *intv* function at level-1 specification.

### 6.5 CONVERSION OF LEVEL-2 SPECIFICATION TO FSM MODEL

**Definition 2.** *Logical Page (LP):* A web page can be split into different logical pages; where each of the logical pages reflects either any change of value in its interface variable or change of state due to any action performed by/on the system.

**Step I:** Each of the Logical Pages represents a Node.

**Step II:** Each of the events represents an edge between the source LP and the destination LP; there can also be self-loop.

**Step III:** Value for all the *ext* variables must be set before any state transition.
6.6 CASE STUDY

Next, we will explore the proposed methodology with help of a case study from LMS development. We consider a learning unit under ‘Basic Science’ course that has the following requirement:

*The web page should be able to convert the measure of length and weight from FPS unit to SI unit and the reverse.*

6.6.1 Functional requirements

FR1 The user should specify the physical quantity to be changed

FR2 The user should specify the source unit

FR3 The user should specify the target unit

FR4 The user should enter the value

FR5 The user should click on the “calculate” button

FR6 The result will be displayed on the user’s screen

Due to the inherent ambiguity involved in such NL-written specifications, one SRS can lead to multiple interface specifications. In the above example, different designers can interpret the word ‘specify’ differently [Figure 6.3].
Therefore, an additional interface specification must supplement the SRS in order to avoid any ambiguity and misconception in the designing phase. Let us now consider that the client and end users agree to have dropdown menu to input value for the first three parameters and a textbox to enter the value for the last parameter.

### 6.6.2 Interface requirements

IR1. The interface should provide drop-down list, along with a label for the physical quantity to be changed.

IR2. The interface should provide drop-down list, along with a label for the source unit

IR3. The interface should provide drop-down list, along with a label for the target unit

IR4. The interface should provide textbox, along with a label to enter the value
IR5. The interface should have a “calculate” button

IR6. The result should be displayed as plain text

6.6.3 VDM-SL level-1 specification

types
QuanType = <WEIGHT>|<LENGTH>
UnitType= <FPS>|<SI>

state Convert_Calc of
Quan : IP::QuanType
Unit1 : IP ::UnitType
Unit2 :IP:: UnitType
InpVal :IP::Z
Result : OP::Z

P:WP
init mk- Convert_Calc (Quan : IP::QuanType, Unit1 : IP ::UnitType, Unit2 :IP:: UnitType, InpVal :IP::Z, Result : OP::Z, P:WP) A
P = $\sum$ (quan ,unit1,unit2,inpval,op)
end

operations
convertUnit()

ext wr Result : OP
rd Quan : IP::QuanType
Unit1 : IP ::UnitType
Unit2 :IP:: UnitType
InpVal :IP::Z
Result : OP::Z
Pre unit1#unit2
Post
Quan=(LENGTH ^ unit1=FPS ^ unit2= SI ^ result=inpval * 0.3 )
\lor Quan=(LENGTH ^ unit1=SI ^ unit2= FPS ^ result=inpval * 3.2)
\lor Quan=(WEIGHT ^ unit1=FPS ^ unit2= SI ^ result=inpval * 0.4)
\lor Quan=(WEIGHT ^ unit1=SI ^ unit2= FPS ^ result=inpval * 2.2)
CHAPTER VI- DESIGNING INTERFACE REQUIREMENTS

The level-1 specification specifies the web-application in an abstract way by introducing only three major data types: input, output and web-page. All input and output variables are included in the web-page variable that work as a container variable. In this particular example, one web page is sufficient to represent the system so we have not used any link data type and its operations. The proposed VDM-SL level-1 uses a generic approach to specify the conventional functional requirements at an abstract-level along with the interface requirements without much detail. Next, we show how this level-1 specification is converted to the level-2 specification.

6.6.4 VDM-SL level -2 specification

types
QuanType = <WEIGHT>|<LENGTH>
UnitType= <FPS>|<SI>

state SystemName of
quan : DD::QuanType
unit1 : DD::UnitType
unit2 : DD::UnitType
InpVal : TB::Z
Result : token
q: token
u1: token
u2: token
calc : PB
    P : WP
    f:CF
    S:PS
    pw:PW

init mk- Convert_Calc (quan : DD:: QuanType, f:CF, unit1 : DD:: UnitType,unit2 : DD:: UnitType, InpVal : TB:: Z, Result : Z,P : WP, ps:PS, pw:PW, calc : PB )

$P = \sum pw$
$P = \sum f$
$pw \Omega \text{“quantity:”}$
\[ P = \bar{P} \sum_{\text{quan}} \]

\[ pw \ \bar{\Omega} \ \text{“given unit:”} \]

\[ P = \bar{P} \sum_{\text{unit1}} \]

\[ pw \ \bar{\Omega} \ \text{“target unit:”} \]

\[ P = \bar{P} \sum_{\text{unit2}} \]

\[ pw \ \bar{\Omega} \ \text{“enter the value:”} \]

\[ P = \bar{P} \sum_{\text{(InpVal,calc)}} \]

Quan = <LENGTH>

Unit1 = <FPS>

Unit2 = <SI>

Result = \text{nil}

end

operations

convertUnit() \Delta

ext wr Result : Z

rd quan : QuanType

unit1 : UnitType

unit2 : UnitType

InpVal : Z

q = P \bot c,quan

u1 = P \bot c,unit1

u2 = P \bot c,unit2

if \ q = \ ‘LENGTH’ \ and \ u1 = \ ‘FPS’ \ and \ u2 = \ ‘SI’ \ then

\ result = \text{inpval} \times 0.3

else

if \ q = \ ‘LENGTH’ \ and \ u1 = \ ‘SI’ \ and \ u2 = \ ‘FPS’ \ then

\ result = \text{inpval} \times 3.2

else

if \ q = \ ‘LENGTH’ \ and \ u1 = \ ‘FPS’ \ and \ u2 = \ ‘SI’ \ then

\ result = \text{inpval} \times 0.4

else

if \ q = \ ‘LENGTH’ \ and \ u1 = \ ‘SI’ \ and \ u2 = \ ‘FPS’ \ then

\ result = \text{inpval} \times 2.2
CHAPTER VI- DESIGNING INTERFACE REQUIREMENTS

end if

intv UserAuthentication (X: ActiveControl, Y: ActiveEvent)

if X= calc and Y= CLICK then
convertUnit()

pw $\Omega$ “Result=” $\Omega$ result
end if

if X= quan and Y= change then
if quan $\neq$ quan
then
quan=quan
end if

if X= unit1 and Y= change then
if unit1 $\neq$ unit1 then
unit1=unit1
end if

if X= unit2 and Y= change then
if unit1 $\neq$ unit2 and unit2 $\neq$ unit2 then
unit2=unit2
end if

if X= InpVal and Y= change then
if InpVal $\neq$ InpVal then
InpVal =InpVal
end if

The level-2 specification is used to describe the interface-requirements more precisely while keeping the functional requirements intact. We have included the ClientForm as the container for the input and output controls. We first converted the first three input types as dropdown object at the level-2. Since they are introduced after the inclusion of the ClientForm within the VDM-SL code, they would be automatically assigned under it. The main operation convertUnit() is invoked under the click event of the button calc.
6.6.5 The FSM model

From the above level-2 specification, we identify six logical web pages, each of them reflects different behavior of the system when the defined actions are performed on them [Table 6.5]. Each of the states represents a node and the corresponding actions represent edges on the FSM model.

![Figure 6.4: FSM model for testing](image)

Table 6.5: Action performed on logical pages

<table>
<thead>
<tr>
<th>Action Performed</th>
<th>State/Logical pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page loaded</td>
<td>S1</td>
</tr>
<tr>
<td>Quantity changed</td>
<td>S2</td>
</tr>
<tr>
<td>Unit1 changed</td>
<td>S3</td>
</tr>
<tr>
<td>Unit2 changed</td>
<td>S4</td>
</tr>
<tr>
<td>Value entered</td>
<td>S5</td>
</tr>
<tr>
<td>Button clicked</td>
<td>S6</td>
</tr>
</tbody>
</table>
The testing model [Figure 6.4] is based on the concept of logical pages and their participation on the events; the logical pages represent the states and the events represent the transitions. Table 6.5 presents a list of states and the actions that cause a state change. These actions are generic and can ultimately be realized by the events of the system. For example, the action ‘Button clicked’ can be realized by an event `calc_click()`. Every event can be further broken into a set of sub events; like `button_press` can be broken into `key_press` and `key_up` events or setting the value in a textbox can be decomposed as a series of character entry events. However, such sub events do not reflect any behavioral change in the state of the system and if considered within the FSM, they may lead to the state-space-explosion problem. Hence, we have concentrated only on events, not sub-events. Certain events like change in `DropDown`, check/uncheck `CheckBox`, on/off `RadioButton` and setting values in `Textbox` explicitly assign values to the interface variables. On the other hand, events like `button_click` do not directly assign values to any interface variables; instead, they can implicitly set values to the other state variables (except interface variable) as passing parameters. Table 6.6 shows the list of events that perform the transition of states and the corresponding state variables that they change (if any).

Table 6.6: State transitions

<table>
<thead>
<tr>
<th>Transition</th>
<th>Variable changed</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>quan</td>
<td>quan_change</td>
</tr>
<tr>
<td>a2</td>
<td>inpval</td>
<td>InpVal_text_set</td>
</tr>
<tr>
<td>a3</td>
<td>unit1</td>
<td>unit1_change</td>
</tr>
<tr>
<td>a4</td>
<td>unit2</td>
<td>unit2_change</td>
</tr>
<tr>
<td>a5</td>
<td>-</td>
<td>calc_click</td>
</tr>
<tr>
<td>a6</td>
<td>-</td>
<td>calc_click</td>
</tr>
<tr>
<td>a7</td>
<td>-</td>
<td>calc_click</td>
</tr>
<tr>
<td>a8</td>
<td>-</td>
<td>calc_click</td>
</tr>
</tbody>
</table>
## Table 6.7: Test sequences

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Start, Terminate</th>
<th>Sequence</th>
<th>Expected behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>S1,S6</td>
<td>a1 → a9 → a10 → a11 → a7</td>
<td>Correct result</td>
</tr>
<tr>
<td>T2</td>
<td>S1,S6</td>
<td>a1 → a9 → a5</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T3</td>
<td>S1,S6</td>
<td>a1 → a9 → a10 → a6</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T4</td>
<td>S1,S6</td>
<td>a1 → a8</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T55</td>
<td>S1,S6</td>
<td>a3 → a5</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T6</td>
<td>S1,S6</td>
<td>a3 → a10 → a6</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T7</td>
<td>S1,S6</td>
<td>a3 → a10 → a11 → a7</td>
<td>Correct result</td>
</tr>
<tr>
<td>T8</td>
<td>S1,S6</td>
<td>a4 → a6</td>
<td>Error/No result</td>
</tr>
<tr>
<td>T9</td>
<td>S1,S6</td>
<td>a4 → a11 → a7</td>
<td>Correct result</td>
</tr>
<tr>
<td>T10</td>
<td>S1,S6</td>
<td>a2 → a7</td>
<td>Correct result</td>
</tr>
</tbody>
</table>
6.6.6 Test case generation

Test cases can be generated easily from the above FSM model [Figure 6.4]. The complete path of a test case should have S1 as the start node and S6 as the end node. It is obvious that only few sequences of user’s action will turn out to correct result, others will produce error or no result. Table 6.7 shows all such possible sequences by the users and their expected behaviors. As stated in the technique at section 6.5, every state change must set the $ext$ variable before any transition.

For example, Test T1 requires the following actions on variables: [Table 6.7]

Set $quan$, Set $unit1$, Set $unit2$, Set $inpval$, click $button$

As all the test cases start from S1 (page load), we must assume default value for all the ext variables at the beginning and when they are by-passed in the test sequence.

For example, Test T9 requires the following actions on variables to form a valid test path:

$[quan=<\text{default}>, \ [unit1=<\text{default}>], \ \text{Set} \ unit2, \ \text{Set} \ inpval, \ \text{click} \ button$

Thus, the above discussed procedure is not only useful for testing the functional and interface requirements, the traceability of the design elements to the requirement specification is also easily verifiable. Since, every state change corresponds to firing of an event in the system and each such event must correspond to at least one operation (by overriding $intv$ function) in the VDM-SL specification, we can relate a test sequence with an operation. Now, as all operations in the design artifact are bound to some requirements in the SRS, the test cases can also be used for ensuring traceability of the design components.

6.7 CONCLUSION

In this chapter, we introduce a methodology to perform formal representation and verification of the interface requirements of web-based LMS applications. We propose two levels of add-ons to VDM-SL for modeling functional and interface requirements in an integrated way. The level-1 specification is predominantly
suitable for functional requirements at the analysis model. The interface requirements are stated in an abstract way at this level. In contrast, the level-2 specification is suitable for design of the system with help of compound requirements, i.e., interface-requirements being stated in an integrated way with functional requirements. At this level, interface requirements are described in a more detailed way, usually keeping the functional requirements intact. In our proposed approach, we first create the level-1 specification and then convert it into level-2 specification, using the proposed conversion technique. Then we introduce another technique to develop a FSM model from the level-2 specification. This model is then used for verification of the design against the requirements specification by generating the test cases from the model. The proposed methodology is advantageous to including interface requirements within the model-based analysis process. In addition, FSM model could be produced from the design artifact in an automated way, which in turn could be used to generate the test cases. We explore a case study of LMS development to illustrate the proposed methodology. Although the case study does not necessitate use of multiple web pages and server side data processing, the proposed model is capable of handling such requirements.