Chapter 3

Modelling

This chapter groups some modelling exercises, only some of which involve UML.

3.1 The vending machine revisited

Consider the vending machine described in Sect. 2.1.2, 2.2.3 and 2.3.6. The proposed use case diagram (Fig. 2.2), sequence diagram (Fig. 2.6) and class diagram (Fig. 2.12) make up for a very poor system modelling indeed. The vending machine is always thought of as a monolithic entity: this makes the external relationships clear but says nothing about how to plan and build one. In particular, the monolithic view is incompatible with the fact that a vending machine is composed of different parts. Given the following list of parts:

1. main controller
2. mechanical robot
3. coin acceptor
4. remote messaging system
5. door

and the fact that 2,3,4,5 can only be interfaced with 1, draw a use case diagram and a sequence diagram to provide an initial blueprint for the inner workings of a vending machine.

3.1.1 Solution

The use case diagram is found in Fig. 3.1. The sequence diagram is found in Fig. 3.2. Notice that they do not provide mechanisms for calling assistance operators on failure of providing change and/or food. How should these diagrams change to cater for these occurrences?

3.2 Mistakes in modelling a tree

Fig. 3.3 describes the class diagram of a tree node, which can be used recursively to build an expression tree.
Generate the header file and implementation code using Umbrello, then add the implementation of the only non-obvious functions (getNumberOfChildren and getChildType) as follows:

```cpp
int TreeNode::getNumberOfChildren () {
    // number of children
    int nc = 0;
    switch(m_operatorLabel) {
    case 0: // sum
        nc = 2;
        break;
    case 1: // difference
        nc = 2;
        break;
    case 2: // multiplication
        nc = 2;
        break;
    case 3: // division
        nc = 2;
        break;
    case 4: // square
        nc = 1;
        break;
    case 5: // cube
        nc = 1;
        break;
    }
    return nc;
}
```
Figure 3.2: The revised sequence diagram of a vending machine.

Figure 3.3: The UML class diagram for the TreeNode class.

case 6: // sqrt
    nc = 1;
    break;
case 10: // number
    nc = 0;
    break;
default:
    break;
int TreeNode::getChildType (int childIndex, int theChildLevel) {
    int ret = -1;
    // increase the level by one unit
    theChildLevel++;
    if (childIndex == 0) {
        // left child
        ret = m_leftChild->getOperatorLabel();
    } else if (childIndex == 1) {
        // right child
        ret = m_rightChild->getOperatorLabel();
    } 
    return ret;
}

Now consider the following main function in the file TreeNode_main.cxx:

// TreeNode_main.cxx

#include <iostream>
#include "TreeNode.h"

int main(int argc, char** argv) {
    int ret = 0;

    // expression tree t: number + number^2
    TreeNode t;
    t.setOperatorLabel(0);
    t.setLevel(0);
    t.setLeftChild(new TreeNode);
    t.setRightChild(new TreeNode);
    t.getLeftChild()->setOperatorLabel(10);
    t.getLeftChild()->setLevel(1);
    t.getRightChild()->setOperatorLabel(4);
    t.getRightChild()->setLevel(1);
    t.getRightChild()->setLeftChild(new TreeNode);
    t.getRightChild()->getLeftChild()->setOperatorLabel(10);
    t.getRightChild()->getLeftChild()->setLevel(2);

    // get right child type and level
    int theLevel = 0;
    int theOperatorLabel = -1;
    theOperatorLabel = t.getChildType(1, theLevel);

    // expect theOperatorLabel = 4, theLevel = 1;
    std::cout << theOperatorLabel << ", " << theLevel << std::endl;
    // actual output is 4,0

    return ret;
}
and verify whether the output is as expected (4, 1). If not, why? Is this a bug or a modelling error?

We would now like to code in `TreeNode_main.cxx` a new function that accepts a tree node and returns
the number of children of the root node of the expression tree. Convince yourself that you cannot do this
easily, and explain why. How can you fix this modelling error? Change the UML diagram and the code
accordingly.

### 3.2.1 Solution

The output is 4, 0. The problem is given by the fact that the second argument of `getChildType`, that
is, `theLevel`, was not declared as an `inout` (read/write) parameter but as an `in` (read only) parameter
instead, so it cannot be changed by the function itself (notice the compiler issues no warning about this
occurrence: it is perfectly legal syntactically if not semantically).

The new function cannot be coded in because the model provide no mechanism for going from a given
node to its parent node, much less the root node of the tree. The correct UML class diagram is given in
Fig. 3.4.

![UML Diagram](image_url)

*Figure 3.4: The corrected UML class diagram for the `TreeNode` class.*