Pelican Crossing-Car Model

0960372

Abstract

This paper will try to bring out similarities and dissimilarities between two very similar yet very dissimilar types of modelling paradigms. Empirical modelling and Windows Presentation Foundation (WPF) have many basic principles in common such as dependencies. As we go in depth in the following sections and subsections we will conclude with a simple fact that both Empirical modelling and WPF have been made for different purposes altogether.WPF being more procedure oriented language, where everything has been described beforehand in form of procedures. Empirical modelling on the other hand tends to be more dynamic, introducing dependencies on run time which is not the case with WPF. There is another aspect of programming in EDEN or DOSTE which this paper will be discussing which will be integrating another module which is the person module with this module i.e. how easy the integration process was.(In this paper Silverlight and WPF might have been used interchangeably)

1 Introduction

We start this paper by describing what the model is supposed to and what are learning experiences which one goes through when developing a real life model. Let's begin by describing the three main principles of m which are 1) Observable-Which can be said to be the variable whose values keep on changing in the modelling, 2) Dependency- it is a kind of relationship between one or more observable, so that if value of one observable changes the value of the other observable is also affected.3) Agent- the agent can be the user which gives the values to the observables on the fly. This model is a procedure driven model. In this model we try to model the behaviour of a car at a pelican crossing. We try to acquire knowledge about what processes the car can control and which processes are completely out of control of the car but still taking place. This model tries to model the way in which the person sitting in the car react to few of the situations presented in front of him/her. The observables (variables) in this case are the position of the car, speed of the car, position of man on the road, the traffic signal status. As we proceed further we will describe what the dependencies are and in the end talk about the agency. Throughout this process this paper will try to bring to the readers notice difference between the WPF technologies such as Microsoft Silverlight and our Empirical modelling languages such as DOSTE and EDEN.

2 Building the Model

As stated earlier this model has been developed using Microsoft Silverlight and EM languages such as EDEN and DOSTE. In this section we describe how the model was made in these two languages.

2.1 DONALD

In Donald notation the overall layout of the model has been defined. Donald makes it easy for drawing objects. Many types of objects such as that of openshape have been used extensively. The basic layout of this model consists of the road, car, pathway for man and the traffic signal for the car. There is an observable 'x' having an initial value of 0 being constantly being incremented and being subtracted from the co-ordinates of the car, which in turn makes the car appear to move from right hand side to the left hand side of the screen.

2.2 EDEN

The eden part of the model describes many procedures and observables which help the external agent control the model. The DONALD observable 'x' is declared as an eden observable using the 'is' statement present in eden. There are many procedures like 'checkForMan' defined in eden which keeps on updating the value of the observable 'manOnRoad' to either 'true' or 'false' depending upon the position of the man on the path. We have used the 'setedenclock' function to poll the current position of the man.

2.3 DOSTE

Using DOSTE in this model we have defined a relationship or dependency between the eden variable ‘carPos’ and ‘step’. This dependency will be the deciding factor for whether the car should or should not be moving. Along with these dependencies we have defined a DOSTE observable called ‘car_speed’ which controls the speed of the car.

2.4 Microsoft Silverlight

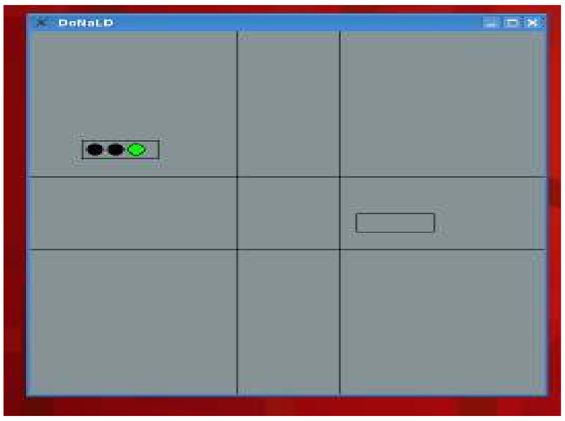
In a Silverlight application we use a XML based language called XAML to design the layout of the model. XAML gives very strong features for animation and making a better graphical interface. Silverlight uses the .Net libraries and hence is able to provide variety of options for the user. The code for the model is written using the .cs file. In the XAML file we have made use of elements such as story boards to run the car animation. Here the name of the element becomes the observable which we access in the code behind .cs file. In the code behind file we can either change this observable or make it dependent on another observable. In the Silverlight model we call the processes as events. After the completion of one event we can trigger another event very easily, and the whole process can be controlled very easily. Silverlight also provides us with debugging facilities so debugging the code becomes very easy and saves a lot of time. One thing which should be kept in mind though is that all the Silverlight code is compiled first and then ran, hence the debugging and error checking becomes easy.

3 Working Of the Model

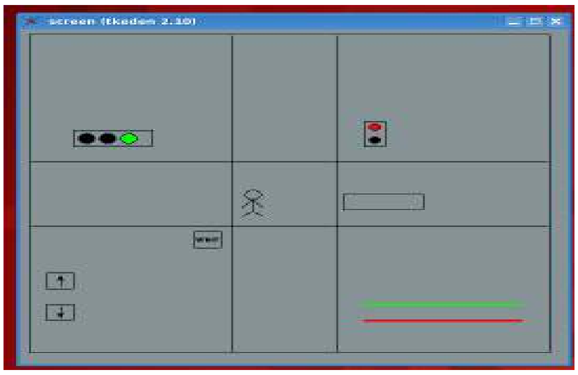
In both the model built using Silverlight and DOSTE, EDEN, the basic functionality of the car remains the same i.e. by default we have made the traffic signal as green and the car moves along the road with a specified speed.

3.1 Working of the Model using EM languages

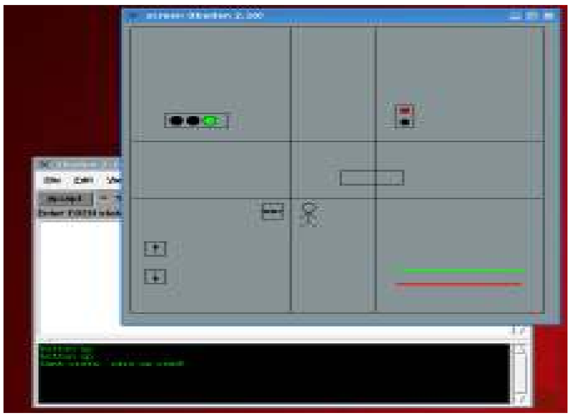
The layout defined using DONALD gives us all the observables which are required for the motion of the car. The observable which is responsible for the motion of the car has been declared as an int type variable called ‘x’. We establish a dependency in the eden part with this variable linking it to an eden variable. We change the value of this eden observable in DOSTE. The motion of the car is dependent on basically three observables in this models which are 1) The position of the man on the road 2) The traffic signal 3) Speed of the car. The observable which defines the speed of the car is declared in the DOSTE part of the code. When the car approaches the pelican crossing and if there is a person at that instant on the road then whatever the traffic signal may i.e. either red or green, the car will stop and wait for the man to cross the street. The car has been designed such that under normal circumstances a car can never run over a man. The second condition which will make the car to stop will be the signal turning to red. The model has been designed in such a way that the car under normal circumstances will not run over the man. The modelling is a procedure based modelling i.e. procedures have been written to move cars, move man, for the traffic signal. To change the behaviour of the car for example making it move backwards will involve writing a procedure. But on the other hand if we want to change the position on the fly we can do it by setting the variable ‘carPos’ which is an eden observable. In DOSTE we establish a dependency between the eden observable ‘carPos’ and the eden observable ‘step’ and another DOSTE observable called ‘car_speed’. ‘car_speed’ is responsible for the speed of the car and can be initialised to 0 to make the car stop at any position on the road. Linking DONALD, EDEN and DOSTE we establish a dependency which makes the car to move. There are many advantages of introducing dependency which can be seen here, one of them being that we are able to use the design features available in DONALD and modify them in eden and doste. Hence we are able to use the features of both eden and doste.



The above figure shows the only car model. If the agent wants to introduce some type of dependency in the above standalone car model it will need to set the observable ‘manOnRoad’ to either true or false.



The above figure shows that the car will not be able to cross the road if the man s on the path, even if the traffic signal is green.



The above figure shows that if the car crosses a particular point which in this case is the 'rightSideOfPath' observable then it becomes the responsibility of the man to not cross the road.

3.2 Working of the Model in WPF

The model's behaviour remains the same in Silverlight as in EDEN and DOSTE. The traffic signal is green by default and the car moves along with a constant speed. The motion of the car is again dependent on two observables mainly one the position of the man and the traffic signal light. The car polls at regular interval to check for the position of the man. '`AreAnyMenInFront`' observale is used in the code behind file to check for the postion of the man. From the right handside towards the first intersection of the path and the road, this is the point where the person in the car checks for man on the road and the traffic signal. After this point it has been asumed that it be the responsibility of the man to chaeck for the car infront of him if he wants to cross the road. The 'CarGoAhead' observable tells the car about the traffic signal status. Only one language i.e c# was used to write the whole model so there were no problems like remebering syntaxes of diiferernt noations like in DOSTE and EDEN.

4 Collaboration

Two models displaying the same characteristics were developed in Silverlight and then in DOSTE and EDEN. Making the model in silverlight gave us insights as in what are the dependencies are going to be. For example before building the EDEN and DOSTE model we already knew that the motion of the car should be dependent upon the position of the man on the path and the traffic signal light. So the time taken to start with the DOSTE and EDEN model was very less. Both the models are fairly procedure driven models i.e. most of the actions which we observe on the screen have been defined inside a procedure and are meant to happen that way. We can try and change few of the parameters but this will result in anomalous behaviour of the model . For example by changing the position of the man the whole procedure named 'checkForMan' will give erroneous result and as a result the man might be run over by the car. We can say in a way that the dependency helped us to learn the model better and make it more robust in a way. Before the the rocedures were written for both the models, there was just simple man and car motion and by initial testing of the model we actually learnt what kind of dependencies are necessary to make this model appear real or logical.
In Silverlight model by using subversion tools availbale on the web we were able to write our separate code in the same file without much hassle. In the DOSTE and EDEN model we did not follow the same procedure. With the help of the silverlight mdel we already ascertained which were the depencies for the moving car. Before the integration in the EDEN and DOSTE model the agent used to specify the value for the observables . The integration of the code in DOSTE and EDEN was the easiest part of model making, just by making few x and y axis changes we were able to overlap one model over the other.
Before the dependency between the various observables was established the car used to traverse along the path without stopping, but after the introduction of the man module in the model we learnt that the car has to check the position and the traffic signal before it approaches the signal and not after it has passed the signal. If the car had to stop it would do so before it crosses the signal and not after it has crossed the signal. So it was decided that the car should check certain observables before it reaches the traffic signal and make the decision of continuing to move or to stop at that position only.

5 Limitations

5.1 Limitations of WPF

Silverlight which is a subset of WPF does establish the dependency principle between the observables. The limitation which we realised after running the Silverlight model was bringing the dependency at the run time. Silverlight code is a compiled code i.e. it runs after compilation and we have to compile the code every time before we try and run the code. The agent in Silverlight cannot provide input on the fly unless or until there is a procedure written which deals with the change of that observable. Due to this reason the agent will not be able to change the speed of the car on the fly in the Silverlight model. Same case is with the position of man or the car which cannot be changed by the agent on the fly.

In DOSTE and EDEN the dependency can be generated on the fly between the observables by the agent. They do not need the compilation of the whole code again to show the changes.

5.2 Limitations of EDEN and DOSTE

EDEN and DOSTE do support few things which are not quite achievable in Silverlight. But the major hurdle which one faces in developing a model in EDEN or DOSTE notation is the use of different syntaxes. Different syntaxes often cause problems when dealing with building models with a mix of different notations.

It would be hard for a novice user to make a complete working model on DOSTE due to the lack of the documentation present. There is no debugging facility available so the finding where one has gone wrong is a cumbersome process.

We can access the EDEN observables from DOSTE but the same is not true from DOSTE to EDEN.

There is only one type of dependency present which is of the form a=b+c, where as in WPF we bring out more than one type of binding [1].

6 Future scope

In the future we can try and bring in more dynamicty to the model. For example change the postion of the man and the car still reacts in the same way as it does now. In the silverlight model we can introduce speed of the car observable so that the agent can change the speed on the fly.

In the fututre one might try to build this project more state based rather than procedure based which is how it has been implemented now.
The graphical interface can be improved using DOSTE.
This model can be extended to encapsulate multiple cars coming from both the directions, which make the model more realistic. This model can also be made to show the environment in which the car might be moving in i.e. day or night.

7 Conclusions

Both EM and Silverlight (WPF) are related in way through dependency. They bring in dependency among observables, and by changing the value one observable the other observables also get affected.
Having established the aforesaid facts, both are made yet for completely different purposes. WPF being more procedure based in which all the actions are predefined and the model will behave the in way it is supposed to.
EM on the hand tries to bring in dependency at run time which is not suitable in WPF. EM can be used to bring in more dynamicity. It can be used to study a model more in depth by actually learning from the failures of the model which might be some condition or dependency brought in by the agent at random. For example one might give the speed of the car as an inverse relation to the distance between the car and the traffic signal i.e. the car will move faster while it is nearer to the signal. In this way one might want to observer what happens to the man on the road.

Acknowledgements

To Meurig Beynon and Steve Russ for their support in the creation of this paper and in organising the ‘Introduction to Empirical Modelling’ module.

References

[1] Antony Harfield, Dependency in action (lecture 22nd October 2009).slide number-20