Improving pelican crossing efficiency: A comparison of Silverlight with EM tools

0961191

Abstract

The model performs a study of a pelican crossing and illustrates a common real-time scenario, demonstrating how wastage occurs at the pelican crossing. Using the principles of empirical modelling (EM) the model proposes a solution to improve the efficiency of the pelican crossing.

Two models have been built using two different technologies. The first model has been built using a conventional technology called Silverlight (SL), (Silverlight is a web based technology and it is a subset of Windows Presentation Foundation). Another model has been built with Empirical Modelling Tools (DOSTE, Eden, Donald and Scout). A comparison of both the models and the two technologies is made, highlighting the advantages of both environments.

The development of the two models has been shared between two persons, and each person has built half part in each model. A brief account of the collaboration and division of model into components is explained.

1 Introduction

This model presents a common scenario that occurs at the pelican crossing (PC) signal. Using this scenario, an explanation is made that shows how there is a substantial amount of wastage at the crossing. The model proposes a possible solution to this problem, showing how the efficiency of a pelican signal can be improved.

1.1 The Pelican crossing problem

A pedestrian (referred to as simply "man") wanting to cross a Pelican Crossing can use a switch to request to stop the vehicles (referred to as "car") on the road, and thus allowing him "Green" signal to safely cross the road (Figure 1).

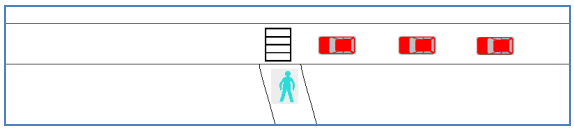


Figure 1: A man approaching a PC.

On pressing the button at the pelican crossing signal, [Figure-2 (a)], the Man's signal turns green after a short duration, and the cars' signal is turned to red (the vehicles have to wait). The man's signal stays green for a brief duration – this is when the pedestrians who were waiting can cross the road, after which the man-signal turns back to red.

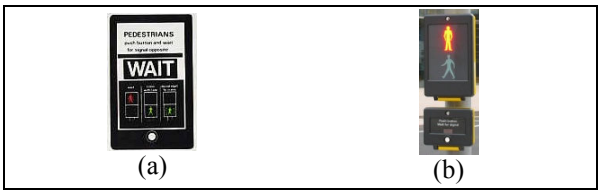


Figure 2: (a) PC Button (b) "Man's" signal.

A very common observation is that, even before the man's signal turns green, the cars would pass ahead, and the road is empty (in front of the man). Very rarely does the man wait for the signal anymore (in such a scenario) and usually - the man walks through the road and crosses it away.

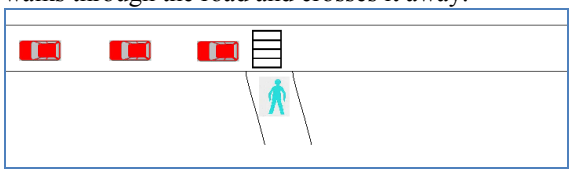


Figure 3: Cars pass even before the signals switch.

Even though the man has crossed the road and walks away, the signal still turns man=green and cars=red as usual. This often creates a situation when a car approaching a road is required to wait even when there is no man crossing (or waiting to cross) the road.

1.2 A solution to the pelican crossing problem using the model

Through this model, this paper proposes a solution to the pelican crossing problem. It is briefly necessary to explain the model here, so that the solution can be explained. (Please refer to Figure 4).

The model contains a section at the top left containing a scene depicting the real world objects. The black horizontal road in the road for the cars and the vertical brown path in the middle is the man's walking path. The buttons at the bottom (left) are the man's controls for user interaction. The agent (interacting with the system) can use the buttons labelled "Walk upwards" or "Walk downwards", which makes the man move up or down the path.

(a) Silverlight version. (b) DOSTE/Eden version.

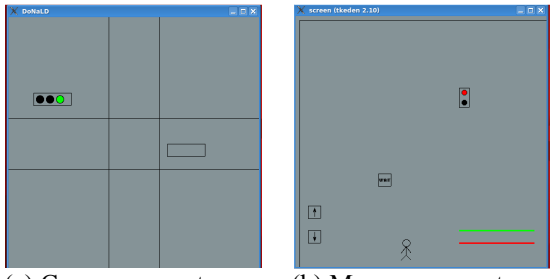
Figure 5: Pelican crossing models.

2 Working of the model

A brief working of the model found in the model documentation is contained within the submitted model. The screenshots of the completed models are given in Figure 5.

3 Collaboration in development

As mentioned in the abstract, the model development has been shared and built in collaboration by two persons. The two parts can be seen in Figure 6: one component deals with car and the other with man. This paper deals with the *man's component*.



(a) Car component (b) Man component

Figure 6: Collaboration in development.

3.1 Collaboration in development

The two components cooperate, by communicating with each other, with the help of the dependency of observables from one component upon the observables of the other.

For example: if there is a car on the road, and the agent (interacting with the system), presses a button to move the man across the road, the system will respond back that "the man cannot cross, as there are cars on the road" (to avoid a collision). The observables describing the state (position, in this example) of the car will be used by the system to validate the action of the man.

3.2 Learning through the model

Identifying the set of dependencies that the components require from each the other, in each of the versions (SL and DOSTE) is interesting to be discussing here. The Silverlight version was built first. As a result, it gave a clear idea as to the exact observables involved in inter-component dependency while building the second model, making the second task a bit easier.

A second example of learning through the model is: while developing features for user interaction of the man (in SL model). Initially the model was built with 3 fixed actions for the man: 1. walk up the first path, 2. Cross the road and 3. Walk away (upwards). But after the car component was built and integrated, it was realised that, it was difficult to demo the model (with only one direction movement), hence the man's actions were changed to move up and down the paths (as it is now).

3.3 Modelling processes

The model has a process based approach for modelling. Certain some tasks (for example, a man waits for car) implicitly depend on another to be accepted as sensible. This makes it possible for an agent to see if it "looks" like a previous task has been com-

pleted, to validate that information before he can start his next task. In this way processes (a dependant chain of tasks) have been modelled [2].

3.4 Advantages of collaboration

Code sharing was an important learning through the collaborative development. In the first model (SL), development happened in mutually exclusively. One component was built and the other could be started only after the code shared with the second person (as the development was made in the same files). Although it should be noted here that a clearer division of components development could have been decided earlier, but for simplicity (of code) it was not done. But in the second model (DOSTE), code was developed completely separately from the beginning (as a result the DOSTE/Eden version is in separate files: PC_man.d and PC_car.d) again partly because of the knowledge leveraged by building one model, and hence essentially integration was an easier effort in the second model. Thus the shared development through collaboration was an interesting learning.

3.5 Construal

Throughout the development of both the models, there has been a continuous construal of experience. Just like a classical program development phenomenon, designs would be made, for example at the start of implementation of any particular feature. As it is expected, the programming continuously involved updating the states on the fly, adapting the programs to achieve the end result that is in the current state.

4 Semantics

After having discussed the collaborative development of the two components, it is relevant to discuss comparisons of a few implementations here.

4.1 Drawing objects

Drawing an object, like "a man" in Silverlight can be done in various ways, using line drawing graphics, but an easier way of doing it is by creating an <Image> using the XAML (eXtensible Application Markup Language).

In the DOSTE model, drawing objects is done using the Line Drawing abilities DoNaLD.

4.2 Moving the man

One of the primary operations in the model is moving the man. In Silverlight, it is done using an animation. "An animation is iterator that can be used to make properties of a visual object (like size, colour, style etc.) dependant on the iterator." [3]

In DOSTE, the movement of man is done using an 'is' dependency definition. A snippet of code is shown here:

```
.eden base man_top is
{@root eden base man_top +
(@root eden base man_step *
(@root man_speed))
};
```

Here it can be observed, that the "man_top" observable has a continuous dependency on itself a few other observables, hence continuously changing (in the form of iteration). The change in this value will draw the "man" object at different positions, thus making the man to appear like a "move". The code constructs in this implementation are referenced through Lab session 2 [4].

5 Comparison of Silverlight and EM tools

Though there are different levels of ease-and-difficulties in both the technologies while developing the models, it is still important to understand that the EM tools have been to cater to the key principles of empirical behind. In both the models EM principles can be demonstrated, but at different levels.

5.1 ODA in Silverlight

An attempt has been made to demonstrate the principles of Empirical Modelling: Observables Dependency and Agency (ODA) in the SL model. Observables have been used to define the states of "objects" in the model (for example, the position of the man).

Agents (the "objects") refer to the state of the observables of themselves as well as of other agents while making decision at the start of some actions, thus demonstrating the concept of dependency and agency. Agency in this component (man's part) of the model has been provided through the user interaction with the model. The agent can control the actions of the man: walking "up" and "down" the paths, and the "wait" button – to trigger a request to allow man with a green signal to walk.

5.2 Introducing dependency at runtime

It is necessary to revisit the principles of EM here to study the affect of applying dependency dynamically.

In the DOSTE version model, the EM tools provide a great capability to introduce dependencies at runtime. For example, currently the model demonstrates a process based approach: if there is a car on the road, the man does not cross and gives the agent back a message – "cannot cross". But the same action of pressing the button to walk across the road can be associated with another definition, (say) to wait till the cars have passed and then automatically cross the road when there are no more cars.

This may or may not always lead to sensible consequences, but at times this could be viewed as a powerful capability of the EM tools.

In Silverlight, introducing any dependency requires normally to recompile the code, and run again. However it can be done dynamically by capturing the objects from the memory, introduce the dependencies at run time, although it would be an indirect approach and difficult. It must be admitted that effectively we would try to re-implement the features of EM tools a Silverlight model, which is not what Silverlight it is meant for.

The goals Silverlight are different: the purpose of this technology is "built to last" and "built for change." The meaning of these goals is explained as follows:

- Projects are “built for change” because redeploying a Silverlight project is easy as it is a web based solution, the web – page has to be re-opened in the browser, that’s all (the latest version will be automatically downloaded and available).
- Since “change” is so easy, SL projects are not necessarily a complete solution in the initial versions. Requirements are added or changed with progress in time and learning from the experiences of previous versions, thus newer versions are released regularly – enriching the product at each cycle of release. Hence SL projects are “built to last”.

Another important comparison is SL is rich in UI. Since SL technology is built with a goal to reach users on the internet, rich user experience is a priority for SL, with obvious reasons.

In Silverlight layouts in the UI can be defined using Grid/ StackPanel/ or Canvas. Canvas is analogous to a viewport in DoNaLD. Having more layouts in DoNaLD would be a rich addition to the tools.

6 Future development of this model

There is some scope for future development of this model in two ways. First, the model(s) has been built with only a single person and a single car. To study the effect in real time, and evaluate the worth of the model, it should be developed with a multiple person and a multiple-vehicles scenario.

Secondly a comparison has been made between the two models built using SL and EM tools. An even more exhaustive study can be done in this area of research.

Conclusion

This model aimed to demonstrate a new dimension of thinking: an effort has been made successfully to employ Empirical Modelling and its key principles, in solving a real-world problem which affects the common man in daily life.

Two separate models were built: using a conventional technology (SL) and EM tools, and the comparison has made between them has been fruitful.

Building two models helped the author in realising that having a completed former model eased the development of the latter, even though they were developed using different tools and implementations.

The key principles of EM: observables, dependency and agency have played an important throughout the development of these models.

Acknowledgements

We would like to thank Dr. Meurig Beynon and Dr. Steve Russ, for their continuous guidance and encouragement at every stage of this project.

References

[1] Russell Boyatt, Antony Harfield, Meurig Beynon. Learning about and through Empirical Modelling. In Proc 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006) Kerkrade, The Netherlands, July 2006, 662-666. [090]

[2] Y-C. Chen, S. Russ and W.M. Beynon. Participative Process Modelling. Proceedings of the IEEE SMC 2000 "Cybernetics Evolving to Systems, Humans, Organizations and their complex interactions", Tennessee, USA, October 8-11, 2000. [060]

[3] Anthony Harfield, Lecture 5 - Dependency in Action (22nd October 2009), Department of Computer Science, [Online]: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/dependencyinaction.pdf, University of Warwick.

[4] Meurig Beynon, Nicolas Pope, Lab 2 - *Objects, Cloning and Animation with DASM* (13th October 2009), Department of Computer Science, [Online]: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/lab2, University of Warwick.

[5] Meurig Beynon, Steve Russ, Lecture 2 - On construal (12th October 2009), Department of Computer Science, [Online]: http://www2.warwick.ac.uk/fac/sci/dcs/research/em/teaching/cs405/lecture2sbr.pdf, University of Warwick