Bicycle Drive Chain Simulation

Abstract

This paper discusses the application of *Empirical Modelling* to a simple mechanical simulation; a bicycle drive chain simulation. We look at the methods used to construct the model and some of the definitions used. Finally, we conclude by detailing the advantages and disadvantages of EM and any possible future developments.

1 Introduction

1.1 What is Empirical Modelling?

Empirical Modelling (EM) is a software paradigm in computer science developed at the University of Warwick. The two key features of empirical modelling are that is a fundamentally agent-oriented approach to programming and secondly, that it is a definitions based language.

So, what does this mean? An agent is essentially an entity whose actions depend on the actions of other entities. This is where the dependency comes in. In a conventional programming language it is possible to assign a value to a variable based on the values of other variables. However, if these variables change their value, the *dependent* variable must be manually updated by the program for it to hold the current value. Empirical Modelling is different, the EDEN[1] interpreter (University of Warwick, 2005) automatically propagates variable changes through the program, updating the dependencies instantaneously.

1.2 How can EM help model the real world?

EM lends itself easily to experimenting with models based on the real world, as will be demonstrated with the *Bicycle Drive Chain Simulation*. The dependency based approach allows a user to add new functionality to a script, while it is executing. Since the dependencies automatically propagate, any changes they make to existing definitions take effect immediately, without the need to manually update them. This allows a user to build up a model iteratively. As the model evolves, existing definitions can be redefined with more complex definitions.

A definition is of the form:

definition `is` *expression*

Due to the finite nature of computers, recursively defined dependencies are not permitted.

[1] EDEN: Evaluator of DEfi nitive Notations

2 The Bicycle Drive Chain Model

2.1 Introduction to the Model

This paper is by no means a report on the *Bicycle Drive Chain Simulation* itself. Instead, its chief focus is on the suitability of empirical modelling for developing and investigating models of real-world situations.

It is, however, necessary for the reader to be aware of some of the features of the model before we can start our discussion in full. The aim of the model is to demonstrate the dependencies between the gears selected on a bicycle and the speed and acceleration (amongst other values) of the rider. The model takes the pedal force, rider mass and maximum cadence[2] as input, along with the selected gears.

The model uses torque to calculate the ratio between the pedal force and the force applied by the rear wheel. The equations were formulated with the help of the *Bicycle Physics Online* web-site (van der Schans, 1999).

The forces involved in a bicycle drive chain are shown in Figure 1. The torques involved can be calculated as follows:

$$t_1 = f_1 * l_1$$

$$t_2 = f_2 * l_2$$

$$t_3 = f_2 * l_3$$

$$t_4 = f_4 * l_4$$

[2] Cadence: cycling terminology — number of revolutions of crank per minute

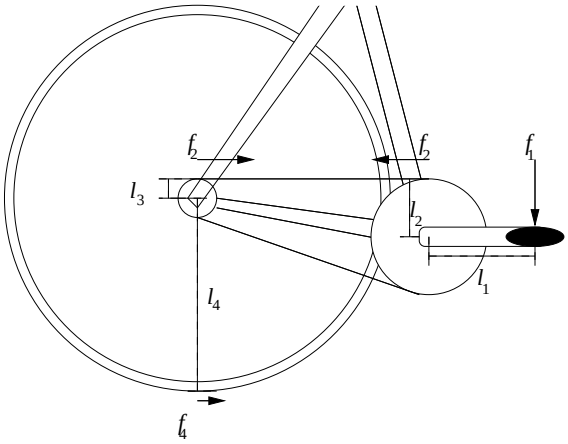

Figure 1: Forces in a drive-chain

Combining these forces and rearranging the equations, the force on the ground (f_4) can be calculated from the pedal force (f_1):

$$f_4 = f_1 * \frac{l_1 * l_3}{l_4 * l_2} \quad (1)$$

Having calculated the force, the acceleration can easily be calculated. Kinematics can be used to calculate the velocity and distance travelled.

2.2 Application of EM

The initial model started by calculating the speed of the bicycle from the cadence and the gear ratios, similar to the *Bicycle Gear Inch and Shifting Pattern Calculator* (Masterson, 2003). However, experimentation with the model revealed that it would be difficult to add additional features, such as acceleration calculation. For this reason, the model was re-implemented using torques and forces.

Equation 1 (above) is implemented in the model as a dependency, taking advantage of the flexibility this offers. Dependency is something that is used extensively in the model, since many of the values are based on the clock value. By making these values (such as speed, distance) depend on the clock value, they are updated automatically as time progresses.

A dependency approach is useful for implementing GUIs as well. Since a GUI is supposed to represent the current internal state of a program to the user, making this representation dependent on the program variables is required. While most programming languages bolt this on under the label "event driven", it is an integral part of EDEN and Empirical Modelling. Similarly, values in the software can be set to depend on a GUI widget value. Figure 2 shows the GUI for the *Bicycle Drive Chain Simulation*. Note the slider bars down the left hand side; variables used in the calculations depend on the values selected here. Also note the three graphical output windows; an overall view of the bicycle, the speedometer and the cyclist's relative distance and velocity.

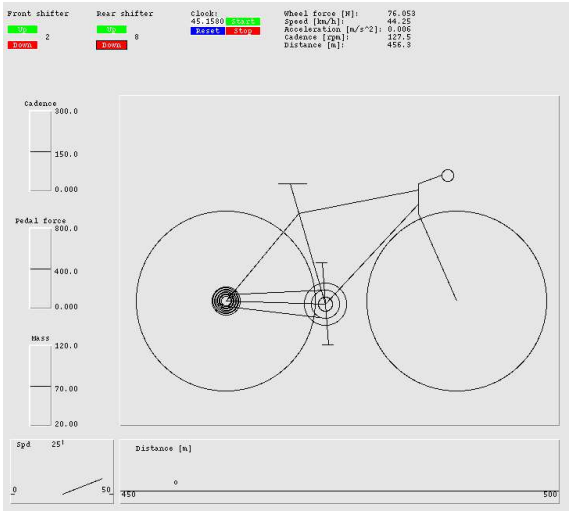

Figure 2: The model in action

One of the advantages of EM is that building a model can often give you a greater understanding of the relationships present in the item being modelled. In this example, a lot was learnt about the jumps between gears, and how significantly this can affect the performance of the cyclist. If the gears are too close, the cyclist will never get up enough speed. However, if they are two far apart they will not be able to accelerate. With a large enough step up, the cyclist will even start to decelerate.

2.3 Future Expansion

While this is currently a medium sized model (some 900 lines of definitions), there is much scope for further expansion. Firstly, as in the *Vehicle Cruise Control System*, functionality could be added to support gradients. The bicycle does not have brakes fitted and it's only way of slowing down is through wind resistance alone, this is another area of possible expansion.

Other expansions relate more to the mechanics involved in the system. For example, no friction is currently modelled in any of the pivots or the chain. Rolling resistance[3] is currently not compensated for.

[3]Rolling resistance: the force used to compress the tire as it

More advanced models could be created to allow investigation into specific areas. For example, many mountain bikes now have rear suspension. The force f_2 in Figure 1 can cause the suspension to compress under pedalling load. The degree to which this occurs depends on the pivot location (amongst other things). An expansion to the model could allow the user to select different pivot configurations and monitor the effect that pedalling has on the suspension action.

Further expansions could improve the interface. For example, Sasami can be used to create a 3D graphical representation of the system. Sasami is a definitive language extension EDEN that allows models to use OpenGL 3D graphics library. This should allow better representation of the different parts of the drive chain, including the cassette[4], chain-rings[5], derailleurs and shifters.

Currently the gear ratios are calculated using the radius of the cogs. It would be more convenient to calculate the ratio from the number of teeth on each cog since this is how gear sizes are measured in the cycling world.

3 Conclusions

3.1 Advantages of EM

There are many advantages of using Empirical Modelling. While certain aspects of the EM tools are not particularly mature, the dependency feature is without a doubt a very strong feature. Dependency functionality would be useful in all programming languages. As an example, imagine updating your operating system with the latest patch and not having to restart it. Instead the dependencies would propagate through the system automatically.

As mentioned above, the dependency based approach has clear advantages when it comes to GUI programming. The consistent approach to dependencies makes GUI development simple. This is useful these days since most software has a GUI.

A further advantage of EM and EDEN is that it naturally lends itself to prototyping. The dependency approach allows a developer to build up a piece of software piece by piece. Definitions are redefined and dependencies are updated automatically. It is not just software and computer science projects that can be simulated or prototyped in EM. Using the AOP[6], new notations can be written that would allow mechanical simulations (for example) to be specified. The interpreter can be interacted with in real-time, which is another useful feature for prototyping. Through the tkEDEN interface, variables can be queried, dependencies can be changed and new parts of the model added.

Since EDEN is a definitions based language it does not have a 'thread of execution' in the conventional declarative sense. Instead, functions and procedures can be specified to trigger when a variable or definition changes value. This is similar to the event driven paradigm (see Section 2.2 that many programming systems use to cope with GUIs, except it can be used in any part of the program. In the EM paradigm it is particularly useful when running procedural code when a dependency changes.

EDEN and the extensions (Scout and Donald) are relatively small languages and so very easy to learn and use. Building the *Bicycle Drive Chain Simulation* took a short space of time, compared to the time it would take to build an equivalent program in a procedural system. This is also partly down to the definitive nature of EM programming, allowing the user to simply add a new definition without having to manually update all definitions reliant on this.

3.2 Disadvantages of EM

Empirical Modelling and the modelling tools are not without their disadvantages. This section will look at some of these disadvantages in more detail. One of the biggest problems with EM and EDEN at the moment is their relative immaturity. There are bugs in the interpreter that can cause unexplained crashes and events. The error messages that are produced when an invalid definition is entered are often obscure.

Inconsistency between the EDEN language and it's extensions is also a problem. Not only are these languages very different in their construction, but accessing variables between extensions is tricky. As an example, although Donald can access EDEN dependencies and variables, accessing EDEN list items is not possible.

Although EDEN features three different extensions for handling graphics, there are very few GUI *widgets* such as slider bars and radio buttons. Instead the user must create them. This is something that goes against EM's usefulness at prototyping — usually the user will not be concerned with specifying these parts of the model as they are using EM as a tool.

An empirical model is not suitable for all applications. The automatic dependency propagation can have a significant performance hit in large applica-

rolls over the ground

[4]Cassette: cluster of cogs on the rear wheel

[5]Chain-rings: cogs (usually three) attached to the crank

[6]AOP: Agent Oriented Parser

tions with many dependencies. This is particularly noticeable in dynamic models, where dependencies are calculated as a function of time. Indeed, in the documentation for the *Vehicle Cruise Control System*, it is noted that EDEN was not designed for dynamic applications. Implementing this in a model is currently difficult and not particularly elegant.

Finally, putting the immaturity problems aside, it is very easy to write 'bad code in EDEN. Since in a procedural language statements that relate to each other can easily be separated out into functions (and objects in OOP[7]). However, in a definitions based language this unconscious grouping of statements does not occur naturally. Instead the programmer must make a conscious effort to group statements together otherwise the model can be very difficult for others to read and understand. Edward Yung noted this in his MSc thesis (Yung, 1989); "a definitive program, especially the EDEN program, is not easily modularized". He said this was for two reasons; a definitive program is usually modified by the user at run-time, and secondly because the interdependency in the program makes it hard to separate variables.

3.3 Future of EM

Given the above criticisms, it is important to point out that Empirical Modelling is an area of on-going research. There is still some way to go, but the ideas and philosophy behind EM are sound.

Standardisation of the different language subsets is a possibility in the future. Thus removing the need to switch between programming styles when working on a program. The tools will obviously mature with time. A useful addition to tkEDEN would be a graphical dependency display, similar to that found in the *Dependency Modelling Tool* (DMT). Additional sub-languages could be included into tkEDEN. Perhaps, for example, a language that can be used for the modelling of GUI widgets such as slide bars and radio buttons.

The use of dependencies is very useful when programming and it is hoped that future programming languages will adopt some of the ideas that have been conceived by the Empirical Modelling project here at Warwick.

Much current research is involved with the use of the EM tools as an educational tool. The interactive nature of EM helps in this respect as pupils can see the result of their interaction immediately. For more information on this area, see Chris Roe's PhD thesis (Roe, 2003).

[7] OOP: Object-Oriented Programming

3.4 Conclusion

To sum up, Empirical Modelling is a very useful tool when developing a model similar to the *Bicycle Drive Chain Simulator*. The use of dependencies is very natural and applies well to this sort of problem. Similarly, the iterative and interactive approach to development is appropriate and genuinely useful in this situation.

Although there have been some criticisms of EM and the tools, it is important to remember that it is not designed to be a heavy weight programming language and is instead described as a "scripting language".

Finally, the assessment of any exercise or case study such as this one hinges on how much has been learnt. Over the course of this document we have introduced some key Empirical Modelling concepts, how these apply to the case study in question and some of the advantages and drawbacks that have been discovered.

References

Barry Masterson. Gear inch and shifting pattern calculator. World Wide Web page, 2003. `http://www.jbarrm.com/cycal/cycal.html` Last Accessed January 21 2005.

Chris Roe. *Computers for Learning: An Empirical Modelling Perspective*. PhD thesis, Department of Computer Science, University of Warwick, November 2003.

DCS University of Warwick. EM tools home page. World Wide Web page, 2005. `http://www.dcs.warwick.ac.uk/research/modelling/tools/` Last Accessed January 21 2005.

Bar van der Schans. Bicycle physics online. World Wide Web page, 1999. `http://www.science.uva.nl/research/amstel/bicycle/partic/Bart/Project/` Last Accessed January 21 2005.

Edward Yung. *EDEN: An Engine for Definitive Notations*. MSc thesis, Department of Computer Science, University of Warwick, September 1989.