Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

AMK: an interface for object-oriented Newtonian particle mechanics

P. Mitic, P.G. Thomas
Faculty of Mathematics and Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK

1. Abstract

This article describes an object-oriented environment with an associated user interface, *AMK*, for modelling simple Newtonian particle mechanics. It is intended for educational use, and provides a framework for modelling which generalises methodology. Physical objects are treated as logical objects, and mathematical models are formulated by linking them. The implementation is within the Windows environment using Mathematica and Visual Basic. Modelling is done by constructing objects and linking them to produce new objects. The aim is to produce an equation of motion object. The interface forces the user into a modelling cycle of constructing and linking objects, and accessing their methods. It constructs a Mathematica input automatically from information supplied by the user, and communicates with Mathematica. The combination of a generalised environment plus interface produces correct answers when modelling many specific physical systems.

2. Introduction

Undergraduate courses on Newtonian mechanics often rely on providing examples of problem solving rather than giving a framework of theorems and proofs, upon which problems may be posed and solved. Elements such as particles, forces, strings and springs are introduced almost implicitly as examples are given. General approaches to problem solving are not always stressed. There are certain problems with this approach. Students can have difficulty in isolating the necessary steps which are needed to arrive at a formulation of the problem, and in applying the correct techniques and processes at the appropriate time. Non-dedicated computer algebra systems (CASs) have been applied to particular applied mathematics contexts through the use of imported utility files or links to

external programs. Specific examples in Newtonian mechanics have been considered, and Dubisch [1] provides an example of a more general approach in which a toolkit is used to solve the same type of mechanics problems that we consider. This interface is command driven, and is therefore very dependent on syntax. It is also uses specific templates to solve problems. We approach the problem in an object-oriented (O-O) way, and have developed a set of objects which we term the 'Applied Mathematics Kit' (*AMK*). The same name is used for the software suite developed. We discussed the way in which it operates from the command line in Mitic and Thomas [2]. A similar approach has been implemented in the context of finite element computations by Viklund and Fritzson [3]. They supplement Mathematica by ObjectMath, with which object classes can be defined and manipulated. Mathematica alone provides an insufficient support structure for the complex models considered. Modelling is done using formulae and equations. *AMK* models are much simpler, but still benefit from an O-O treatment. Commonality can be highlighted between *AMK* objects and we can consider the way in which these objects interact in order to construct a mathematical model of a physical situation. *AMK* supports the learning process for modelling: producing a mathematical model of a physical situation is reduced to defining objects, specifying how they interact and invoking objects' methods. Hence, formulae and equations are not needed except when certain transformations are required. The aim of the present discussion is to derive an equation of motion in the case of one and two dimensional Newtonian particle mechanics, using Newton's Second Law of Motion. Solving the resulting equation(s) of motion is intended to be the subject of a further paper.

3. Objects and the Object Hierarchy

A search through standard texts on mechanics, (Milne [4] and Dyke [5]), reveals consistency in the objects that appear and in the situations in which they appear. We can therefore identify the physical elements used in simple particle mechanics, and then classify them in terms of an object hierarchy. Each object is assigned relevant attributes and methods. All objects are embedded in the Mathematica O-O framework developed by Maeder, the principles of which are discussed by Maeder ([6], [7]). The implementation used is from Maeder [8]. Implementation details of AMK objects are given in Mitic and Thomas [2]. Mathematica proved to be a natural choice for this work because of the existence of this environment and the usefulness of symbolic manipulation within it.

4. Problem Solving with AMK

Formulating and solving problems is done by linking objects and generating new objects. When a particle is linked with a sum of forces, an equation of motion is generated. This is achieved by invoking the procedure MakeLink, which is defined in multiple forms, one for each meaningful object combination. For

Transactions on Information and Communications Technologies vol 10, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

example, linking a particle, *P*, with a spring, *S*, produces a force - the tension in the spring, *T*. The appropriate input is `T = MakeLink[P, S]`. There are no links for objects for which the resulting combination is not meaningful.
At any stage, an object's method can be invoked. The goal is to extract the equation, *EM*, of an *EquationOfMotion* object, *EoM*, with an input such as `EM=Equation[EoM]`. The following example illustrates the 'modelling by construction and linking' paradigm, and shows how the equation of motion for the system illustrated in Figure 1 is obtained.

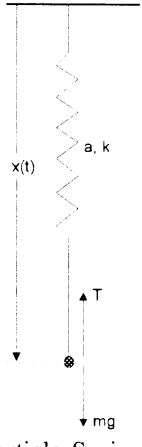

Figure 1: Particle-Spring system

Three objects are constructed: the particle, *P*, the spring, *S*, and a gravitational field, *GF*. *P* is linked to *S* to produce the weight, $W (= mg)$. *P* is linked to *S* to produce the tension in the spring, *T*. The sum of forces, $W + T$, is then linked with *P* to produce the equation of motion, *EoM*. Its *Equation* method is invoked to produce the standard simple harmonic motion equation, `{mx''[t] == k L + g m - k x[t]}`. The following Mathematica expressions can be entered directly.

```
P = new[Particle,Cartesian[{{x[t]},0,1}],{},t,m]
S = new[Spring,Cartesian[{{0},0,1}],{},
        Cartesian[{{x[t]},0,1}],L,k]
GF = new[GravitationalField, Cartesian[{{g},0,1}],{}]
W = MakeLink[P,GF]
T = MakeLink[P,S]
EoM = MakeLink[P, T+W]
Equation[EoM]
```

The final result is the output:

```
{m x''[t] == k L + g m - k x[t]}  .
```

5. The AMK User Interface Rationale

The AMK user interface is dedicated, and is specifically designed to fulfill a small number of very specific purposes. In this respect it differs from some general purpose interfaces, which are designed to be versatile, system-independent and extendible. We specifically mention Kajler's CAS/PI [9] in this respect. CAS/PI was designed as a front end for a number of algebra engines, including Maple, and also peripheral packages such as graph plotters. Consequently it has to deal with a number of mutually incompatible communications protocols. Issues of interface design are summarised in Kajler [10]. *AMK* falls into the category cited as the most common type of interface, which is one based on a toolbox. However, it does not need to deal with editing and manipulation of expressions except in a very elementary way, and does not require a command language. The latter is replace by an event driven environment, the principal event being the mouse click. Our interface is a prototype and its construction reflects a wish to experiment with what can be done, rather than to produce marketable software. Hence, it does not use some features which were felt to overcomplicate matters at this stage. It constructs Mathematica inputs, such as in the example above, and communicates with Mathematica. Its implementation allows the user to be presented with relevant data input forms at appropriate stages in building a Mathematica script. The user needs only to stick to the modelling cycle referred to at the end of the section below.

6. The AMK User Interface

The *AMK* interface aims to address some issues described in Kajler and Soiffer [11]. Specifically, the purpose of the interface is for educational use, and concentrates on easing expression syntax problems by formatting a suitable input expression script. A further aim is to reduce errors by providing a general strategy and making disallowed input combinations impossible. Major problems of editing and notation are not addressed, although an elementary filer allows for some time saving. Since the *EquationOfMotion* object is central to modelling physical situations in Newtonian mechanics, we aim at this stage to derive an equation of motion (or system thereof), without attempting a solution. This is, in principle, easy to do because it merely involves adding a *Solve* method to *EquationOfMotion*. We have implemented the interface in Visual Basic (currently version 3). This provides an environment for placing dialog boxes, command buttons, picture boxes and similar items on a form, which is what the user actually sees, in a visually appealing way. It automatically deals with the detailed scheduling of events. To some extent, a click in a given place on a form is self contained, and there are no complicated interactions with other parts of the software. One only needs to consider the effect of a mouse or keyboard event. These can be written so that there is either a response on the same form, or the current form is replaced by a new one, with different options available. In

general, a response to any given user input will depend on a particular object. A response is determined by using Select Case constructs liberally. This allows for easy, but somewhat tedious programming. An .EXE file, which is not truly self contained but requires an additional run time library file, can be made, and this system provides a stand alone Windows application.

When in use, Mathematica and *AMK* are started, and *AMK* is made the active window. Control is automatically returned to *AMK* after Mathematica has ended its task. Communication with Mathematica is via the Windows clipboard when inputting to Mathematica. Visual Basic provides constructs which makes this particularly convenient. Unfortunately, relevant output cells cannot be placed on the clipboard without specifically selecting them from the Mathematica notebook. Mathematica is not a Windows Object Link and Embedding program, so it is not possible to direct specific information to the clipboard from a front end. The result is that the clipboard cannot be used to return a result to *AMK* neatly. We overcome this problem by directing Mathematica to store selected outputs as a text file, which can be read by the front end when the computation has finished. Using a text file in this way at least avoids relatively complex communications protocols.

The interface gives rise to the modelling cycle shown in Figure 2. The sequence **Construct→Link→Methods** usually occurs in this order, but not necessarily. For example, coordinate transformations are sometimes needed in order to combine forces, and methods can be accessed at any time to retrieve information.

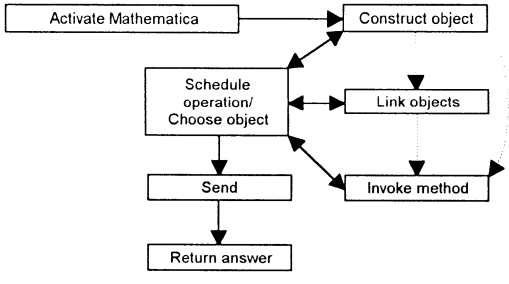

Figure 2: *AMK* Modelling cycle

7. The Interface in use

The initial *AMK* form enables the user to activate Mathematica, coordinate other activities and choose an object to be constructed. Control is returned to this 'Schedule' form after each activity. The most time consuming of these is construction of objects, since attributes have to be specified in full. The other principal forms deal with linking objects and invoking their methods. Other peripheral forms support these activities. In particular, inputs scripts can be saved and reloaded. They can also be edited to some extent. Relatively minor editing

facilities is a design feature, and serves to force the user to consider each construction, link and method. Editing is favoured only if there is a clear similarity between a script before and after editing. For example, it is possible to copy and correct a minor error, and delete the original. A more useful purpose might be to create an object identical to an existing object except in one respect. Similarity and symmetry are supported in this way. The *AMK* editor thus functions like the editor described for GI/S, a front end for Macsyma (Young and Wang [12]). This controls the history of individual expressions by allowing for recall and editing of previous expressions. However, they differ in that the *AMK* editor cannot manipulate subexpressions.

The session below shows how the input for the spring-particle system referred to above is made. The first stage after activating Mathematica is to choose an object to construct using the 'Schedule' form. This activates the 'Construct' form. The spring, *S*, and the gravitational field, *GF*, are constructed in the same way. The 'Link' command button is used three times to make the links `W = {P, GF}`, `T = {P, S}` and `EoM = {P, T+W}`. Using the 'Methods' form then writes the expression to invoke the method *Equation* of *EoM*. The order in which the parts of a given form are filled in is immaterial, and amendments can be made at any stage. A 'progress' text box on each form gives the current status of the expression being constructed, and there is a similar 'progress box' on the 'Schedule' form. The contents of the latter are sent to Mathematica by clicking on the appropriate command button. Part of the Mathematica window must be visible so that the user can see that an answer has been returned. The result may be viewed in a fully scrollable text box, in the same output format which appears in the Mathematica notebook. The result of the above operations is shown, with the Mathematica window in the background, in Figure 3. This ends the session, and the Filer can then be used to save it, clear and start again.

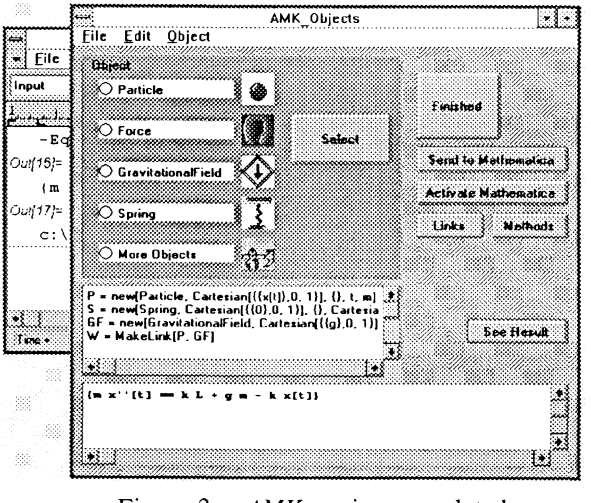

Figure 3: *AMK* session completed

8. Evaluation

The *AMK* object system is able to cope with a wide variety of modelling situations. Simple ones include particle plus force systems. More complicated ones include a particle moving along the line of greatest slope on the surface of a circular surface or a system of linked particles and springs. The coordinate system is successful in dealing with the many conceivable variations in modelling a given problem. The onus is still on the user to provide meaningful inputs, but this problem is not unique to this system. Interpretation can be slow at times, particularly when combining two forces which contain structurally complicated expressions. There seems to be no ready explanation for this since all that is involved is to extract two explicitly stated coordinates from each and add them. It helps to free memory by exiting Mathematica and restarting. The interface forces the user into the **Construct→Link→Methods** modelling cycle, which is the essential component of this O-O methodology. It also forces the user to specify the geometry of a situation accurately, since without geometrical considerations coordinates cannot be specified reliably. To some extent this detracts from the activity of modelling. The interface works sufficiently fast and the slowest step is nearly always interpretation of the Mathematica code. Details can be tedious to fill in, but the editing facility eases this. Other stages are quick to complete. A certain degree of error protection is inherent. For example, only methods relevant to the object in question can be chosen. There is additional protection in specific cases. For example, an attempt to link two objects for which there is no meaningful outcome, such as a *Particle* and an *EquationOfMotion*, is detected and aborted.

The communications protocols are not sophisticated, and this is a clear area for improvement. Returning the result via a text file means that the user needs to know when Mathematica has finished, so the Mathematica window must be partly visible.

The interface software has the advantage that it requires little memory. The AMK .exe file and the Visual Basic run time dynamic link library together require less than 0.5 MB (storage and RAM). We contrast this with CAS/PI, which needs about 17MB storage space. *AMK* does not have a problem with expression size or formatting. The *EquationOfMotion* object is quantifiable in terms of its text file representation, which is typically less than 1K in size. Furthermore, this order of magnitude is known in advance and does not vary much from problem to problem. The combination of scrollable Visual Basic text boxes and Mathematica output formatting ensure that the result is easily accessible and readable.

9. Future Work

The present interface could benefit from a number of improvements. The first is to establish a better communications protocol, for which MathLink is the natural vehicle. An alternative project, which we have already started, is to build an interface which constructs Mathematica code automatically from the way the user positions graphics on the screen using drag and drop operations. This enables a diagram of a model to be drawn, in the same form as it would appear on paper. The associated code generation means that the user does not have to worry about Mathematica syntax. An alternative to maintaining the object hierarchy within the Mathematica environment is to maintain it within a C++ environment. This may prove fruitful provided that it does not involve frequent communication with Mathematica, which could be slow. The possibility also exists to widen the scope of the mechanical concepts to energy, momentum, analysis of phase planes and extended objects.

10. References

1. Dubisch, R.J. The Toolkit: A Notebook Subclass. *Mathematica Journal* vol 1, 2. Miller Freeman. 1990
2. Mitic, P and Thomas, P.G. An Object-Orientated Environment for Newtonian Particle Mechanics. (Eds. Keranen,V. and Mitic, P.) *Proc. First International. Mathematica Symposium*, Southampton, England. Computational Mechanics Publications 1995
3. Viklund, L and Fritzson, P. An Object-Oriented Language for Symbolic Computation - Applied to Machine Element Analysis, (Ed. Wang, P.) *Proc. ISSAC 1992*, pages 397-404, Berkeley, USA. ACM Press 1992
4. Milne,E.A. Vectorial Mechanics. Methuen 1948
5. Dyke,P and Whitworth,R. Guide to Mechanics. MacMillan 1992
6. Maeder, R.E. Polymorphism and Message Passing. *Mathematica Journal* vol 2, 4. Miller Freeman 1992
7. Maeder, R.E. Object Oriented Programming. *Mathematica Journal* vol 3, 1. Miller Freeman 1993
8. Maeder, R.E. The Mathematica Programmer. Academic Press 1994
9. Kajler, N. CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems. (Ed.Wang, P.) *Proc. ISSAC 1992*, pages 376-386, Berkeley, USA. ACM Press 1992
10. Kajler, N. Building graphical user interfaces for Computer Algebra Systems. (Ed. A. Miola) *Proc. of DISCO 1990*, pages 235-244, Capri, Italy. Springer Verlag 1990
11. Kajler, N. and Soiffer, N. Some Human Interaction Issues in Computer Algebra. *SIGSAM Bulletin 107*, vol 28, 1. ACM Press March 1994
12. Young, D.A. and Wang, P.S.GI/S: A Graphical User Interfaces for Symbolic Computation Systems. *J. Symbolic Computation*, vol 4, pages 365-380, 1987