Mathematical Modelling and Workflow

Peter Mitic

Positive Corporation Ltd., 22 Lymington Bottom Rd., Medstead, Hampshire GU34 5EW, England
peter.mitic@virginnet.co.uk

Abstract

This work derives from a recent advance in mathematical modelling based on Object-Oriented principles. Its principal idea is to automate model-generation by defining a rule-based system. Objects within a problem domain can be created, and then iteratively linked in a well-defined manner, until precise goals are achieved. In general, the goal is to generate an equation of state for the system. The iterative process uses queue management principles usually associated with Workflow systems, but eliminates human intervention. The result is a flexible and extendible system that can, given object definitions and rules for progressing the workflow, generate equations of motion automatically within the context of Newtonian particle mechanics. I refer to this methodology as Object-Workflow in this paper.

Modelling with Object-Workflow

The following example shows a simplified version of how Object-Workflow works. Consider a particle P1 suspended vertically from a light spring S1. The particle moves in SHM in a gravitational field G1, with a coordinate system C1. These objects are placed in an *Active* List, which is processed iteratively by taking suitable pairs and linking them to produce new objects (Forces F1,F2,F3, an equation of motion E1, and a solution Sol1). Objects which are no longer needed in the model transfer to a *Complete* List. The following table shows the steps in this iteration. The Workflow rules determine which objects to link at each stage, and the Link rules determine the result of that link.

Stage	Active List	Complete List	Next link...
1	{C1, P1, G1, S1}	{}	P1+G1 => F1
2	{C1, P1, S1, F1}	{G1}	P1+S1 => F2
3	{C1, P1, F1, F2}	{G1, S1}	F1+F2 => F3
4	{C1, P1, F3}	{G1, S1, F1, F2}	P1+F3 => E1
5	{C1, E1}	{G1, S1, F1, F2, P1, F3}	E1 => Sol1
6	{C1, Sol1}	{G1, S1, F1, F2, P1, F3, E1}	

Recent Themes in Mathematical Modelling

In this work I rely heavily on domain knowledge to model particle mechanics using lists and rules. This approach counters current research into mathematical modelling, which centres on problem formulation, communication skills and teamwork [8]. Some authors have attempted to formalise rules for modelling given scenarios, but none provide a significant insight into either general principles or specific heuristics. Recent attempts include [2,5,15]. Ramsden [16] provides evidence that domain knowledge is not only necessary when applying mathematical techniques, but must be integrated into a context: a correspondance with the lists and rules of Object-Workflow is apparent.

In [14] I used evidence from the biennial ICTMA (International Conference on Teaching Modelling and Applications) conferences to argue that the modelling community had only recently realised the potential of computer algebra in modelling. The 1997 conference [9] contained a significant number of papers on computer algebra, in contrast with previous conferences. ICTMA8 papers indicate that using computer tools has shifted attention away from mathematical modelling methodologies and towards routine use of computer methods. Some ICTMA8 papers hint at the ideas that I implement in this paper: [7] - through attempts to isolate and connect key elements of a problem domain, and [10] - in providing evidence that a diagram is an aid to problem solving.

Process-Driven Mathematics and Prior work on Model Characteristics

The idea of using a sequence of processes to define an overall objective is not new in mathematics or computing. In object modelling, defining class methods goes part way to finding individual processes, but does not necessarily combine them. See (e.g.) [17 or 11]. Similarly, school mathematics in the 1960s depended on flow diagrams to teach ideas in algebra (e.g., [18]). Abidin [1] provides a later manifestation of this idea by computerising flow diagrams, although he fails to abstract properties and methods in the problem domain. Furse's Mathematics Understander [6] remains a nearest equivalent to this research in terms of mathematical automation.

The essential relations between elements in the particle mechanics problem domain were explored in [12]. I presented a formal class hierarchy for particle mechanics, with a Mathematica implementation to manipulate those classes and generate solutions to problems. The rationale behind that paper was the realisation that there is a uniformity in objects, interactions, overall structure and processes within a given problem domain. In [13] I described an interface for generating a model in particle mechanics which was user-driven: the user had to make all decisions about which objects to link, and when. The user received no overall strategy for progressing the algorithm. In this paper I provide the overall strategy which were lacking in previous papers.

Object Modelling

In [14] I suggested a simple list-based implementation of the object domain. In this paper, I remove the idea of a class hierarchy, retaining only properties of objects, constructors, and functions describing how they interact with other objects. Strictly, my "objects" are not objects in a true O-O sense: there is no inheritance or polymorphism. Modelling proceeds by maintaining lists of objects, which represent queues, and using list operations and replacement rules. This queueing aspect provides the parallel with Workflow. The software implementation comprises each of the subsections below.

■ "Objects"

Shlaer [17] and Coad [3] give a useful definition and guide for finding classes and objects. *A class is an abstraction of the real world, members of which have the same characteristics and conform to the same rules. An object is an element of a class.* In the domain of particle mechanics, abstractions like particles, strings, planes etc. are always treated in the same way when solving problems. They are classes. We don't need to formalise them in this analysis, but we do need to define the properties and behaviour of the objects within classes. The characteristics of elements in the problem domain are abstracted by simply listing them. Producing such a list requires domain knowledge and experience, but forces you to think about significant properties of elements in the problem domain.

■ Objects and Persistence

I classify objects in the particle mechanics domain as either *Persistent* (they interact throughout the lifetime of the model) or *Transient* (they take no active part in the model after their first interaction). The persistent objects in this paper are: Particle, Coordinate System, Solution. Particles are particularly important because they determine distinct phases of the workflow. The transitory objects used are: Force, String, EquationOf-Motion, Gravity. For example, two forces may combine to produce a resultant force, which effectively replaces the original two forces. In addition a *NULL* object is defined, with *Zero* persistence. A *NULL* object results from two objects that should not interact, and leaves the original objects unchanged. Persistence is a fixed property of the object, so each class is either *Persistent* or *Transitory*.

The Gravity object, although classified as Transitory, falls uncomfortable near to Persistent. Its main purpose is to define a weight when linked to a Particle. It needs to link with all Particles before it's no longer needed, so it is persistent for a limited number of interactions. This contradiction is resolved by applying a set of dedicated queue management rules that create all necessary weights before any other links are done. The function **CreateAMKObject** creates an object using the object model in [20] to "remember" object properties.

■ Queues and Queue Management

The principal queue is QActive, the Active queue. It contains objects that currently interact with other objects in the model. When such interactions have ended for particular objects, they are transferred to QComplete. This queue is a parking area for objects that are no longer required to progress the model. A Gravity object needs special treatment. All Particle and Gravity objects are transferred initially to the QInitial queue. QInitial is then processed using **ProcessInitialQueueRules**, which creates weights for all masses concerned, and then transfers the Gravity object to the QComplete queue. **PopulateQueues** handles these queue allocations. Two other queues have a minor role. All solutions end up in the QEnd queue and QHold is the repository for the CoordinateSystem object. The CoordinateSystem object keeps track of which forces are "connected" to which masses (see the *Principle of Adjacency* in [14]).

Queues are managed using replacement rules. At each stage of the iteration **PersistenceOrderRules** brings *Persistent* objects to the front of QActive. QActive is then processed by considering its first two elements, linking them, and transfering them to QComplete if they are transient. If they are persistent they stay in QActive. The result of the link joins the end of QActive. This process is applied repeatedly until only two objects remain in the active queue: a Particle and a Force. This pair is then linked to form an EquationOfMotion object and can then derive a solution from that.

■ Workflow

The concept of workflow is well-established in large business software applications, and there is an abundance of software to support it. The underlying principle of workflow is to progress an entity (e.g. a logical case, document, person etc.) through a sequence until an appropriate end point is reached. Entities usually have *states*, which change as the entity progresses. Human intervention often determines what the state changes to at any given stage in the process, and software is usually designed to support entity manipulation as a result of a user input. It is often convenient to represent the *state* of an entity by placing the entity in a queue which mirrors the state. Two examples of widely-used general purpose workflow packages are Staffware [19] and the Automated Work Distributor (AWD) from Computer Sciences Corporation [4]. Both can be configured to suit particular situations. Links to workflow products and research may be found on www.workflowsoftware.com.

■ Overall Mathematica implementation

The Mathamatica implementation can be seen in the package **Miscellaneous`MMWorkflow`**: It contains functions and rewrite rules to create, manipulate and link objects, and to control the modelling process by maintaining queues. In order to generate a model, the following steps are required:

1. Define the objects in the problem domain using **CreateAMKObject**. These definitions must account for the geometry of the system by first defining a CoordinateSystem object, whose coordinates are used by other objects.
2. Initialise and populate the queues using **PopulateQueues**
3. Process QInitial, QActive and QComplete using **ProcessInitialQueueRules**
4. Process QActive using PersistenceOrderRules
5. Process QActive and QComplete using **ProcessActiveQueueRules**. This is the main processing loop, and progresses the model to the stage where an equation of motion can be formed.
6. Process QActive and QComplete using **ProcessActiveQueueRules2.** This produces the equation of motion.
7. Process QActive, QComplete and QEnd using **ProcessActiveQueueRules1.** This solves the equation of motion.

These processes are encapsulated in two functions: **GenerateEquation** (stages 1-6) and **GenerateSolution** (stage 7). Encapsulating the model-generating process in this way serves to emphasise how the same processes can be applied to any well-defined input list. Mathamatica is a useful tool for this analysis mainly because it provides extensive support for list processing and rewrite rules. Two methods are used to display information about the queues. **ViewQueue** provides the names of the objects in a given queue. **ShowAllQueues** shows all the queues side by side, in a semi-graphical format based on [21]. It is then possible to compare the position of elements in the queues directly, and the passage of any given element through the queues is easy to trace.

■ Multi-Particle systems

Multi-Particle systems require an element of geometry to be present in either the initial formulation of the objects or in the rewrite rules. The reason is to prevent a link between objects that violate the Principle of

Adjacency. For simplicity, each geometric part of the problem is processed separately. For example, given a 2-particle {P1, P2} system in which the particles are connected by a spring S, and a forcing term F is applied to P2 only, the {P1,S} sub-system is processed first, and then the {P2,S,F} sub-system. An alternative method is to incorporate a *SubSystem* property for each object, which links that object with a particle, and prevents links with other particles.

Examples in the Appendix

The three associated notebooks contain examples of how to formulate and solve simple mathematical models in Newtonian particle mechanics using Object-Workflow.

1-ParticleExamples.nb develops a simple example of a vertical projectile in stages: construction of the objects (Particle, Gravity, CoordinateSystem), processing the queues in turn, and then extracting the solution to the equation of motion. The vertical projectile is a very simple problem, but is not easy to implement using the Object-Workflow method. The Spring-Particle-Gravity system is a particularly comprehensive illustration of the Object-Workflow method as it uses all types of available links. The example of the particle subject to three forces illustrates a catch! It is not sensible to attempt to solve the equation of motion unless the forces are explicit functions (in this case of time). Furthermore, if any of the forces are explicit functions of something other than time (such as velocity), the method of solution will fail, as Dsolve[] would be required.

2-ParticleExamples.nb illustrates a general treatment for multi-particle problem domains. Dividing the problem into two 1-Particle problems is not necessarily the most efficient way: using the centre of gravity is probably better in this case. The 2-Particle-1-Spring problem shows how the geometry of the system determines the direction of the tension in the spring with respect to the two particles. The tension "pushes" one particle and "pulls" the other. This awkward property is built into the Particle-Spring link method.

1-CompleteSolutionExamples.nb illustrates the use of the functions **GenerateEquation**, **GenerateSolution** and **GenerateEquationAndSolution** as wrappers for the entire modelling process. All that is needed is to construct the necessary objects and then call **GenerateEquation** to produce the equation of motion.

Discussion

This paper demonstrates that there is a common overall strategy to problem solving within a problem domain, which can be implemented in terms of lists and rules. There is a conceptual and a technical disadvan tage to this treatment. It requires much analysis to cast a problem domain in O-O terms (even loosely!), and even more to implement definitions and rules that work. This overhead is likely to be an advantage in large systems where models are formulated repetitively. I would recommend further work as follows:

1. Implement the ideas already suggested for dealing with multi-particle problems, which is a generally weak area.
2. Build a mechanism to simplify processes such as defining links and rules (possibly through a GUI)
3. Finding an alternative to the concept of persistence, which can be hard to define and implement. One alternative is to always send new objects to the back of QActive, which could defer a Particle-Force link until a later stage

4. Incorporate solution 'goals' to initiate appropriate solution methods.

A Proof of Convergence

Processing the Active queue relies on an iterative process of linking existing objects to create new objects. If this process does not terminate, Mathematica will end up in an endless loop. This proof demonstrates that this situation cannot arise, and is based on the ProcessActiveQueue rewrite rules and the available links.

Consider a system with NP persistent objects and NT transitory objects. Denote iteration step r by I(r), r = 1, 2, … and the number of objects in QActive at step I(r) by N(r). Then N(1) = NP + NT. Consider the transition S(r) to S(r+1). There are 6 cases when combining pairs in QActive:

Case 1: Particle + Gravity = Force: N(r+1) = N(r) - 1

Case 2: Particle + Spring = Force: N(r+1) = N(r) - 1

Case 3: Particle + Force = Equation: N(r+1) = N(r) - 1

Case 4: Particle + Force = Null (delayed interaction): N(r+1) = N(r)
(the Particle is moved to the back of QActive)

Case 5: Force + Force = Force: N(r+1) = N(r) - 1

Case 6: Equation + Equation = Equation: N(r+1) = N(r) - 1

The transition S(r+1) to S(r+2) cannot be case 4 because of the reordering of QActive in the previous stage (in practice it will be case 5 if the length of QActive>2 or case 3 if the length of QActive=2).
Therefore N(r+2) = N(r+1) - 1.
Hence N(r+2) < N(r) for all r > 1.
Therefore the **ProcessActiveQueue** iteration will terminate in at most NP + NT steps.

References

[1] Abidin,B. and Hartley,J. Developing Mathematical Problem Solving Skills, in J. Computer Assisted Learning, 14, pp278-291, Blackwell Science, 1998
[2] Chinnappan,M., Lawson,M. and Gardner,D. The use of Microcomputers in the Analysis of Mathematical Knowledge Schemas. Int. J. of Math. Edu. In Science and Technology (ed. Harrison,M.) vol 29 #6 pp 805-811 Dec 1998
[3] Coad, P. and Yourdon,E. Object-Oriented Analysis. Prentice Hall 1990
[4] Automated Work Distributor (AWD) Computer Sciences Corp. Woking, UK. 2000
[5] Day,A. and Suri,A. A Knowledge Based System for Postgraduate Engineering Courses. in J.Comp. Assisted Learning (ed. Lewis,R) vol15 #1 pp 14-27 March 1999
[6] Furse,E. The Mathematics Understander, in Proc. Artificial Intelligence in Mathematics (eds. Johnson,J., McKee,S. and Vella,A.), Strathclyde,1991. IMA 1994
[7] Gethins, T. Drug Models and their use in Mathematics Courses. ICTMA8
[8] Houston,K. Embedding Core skills in Undergraduate Math Courses, in Proc. Int.Conf on the Teaching of Mathematics (Barker,B., ed.), Samos. Wiley. July 1998
[9] Mathematical Modelling: Teaching and Assessment in a Technology-rich world (Proc. ICTMA8, Brisbane, August 1997). Eds: Galbraith,P., Blum,W., Booker,G and Huntley,I. Horwood Publishing, December 1998
[10] Ikeda,T and Stephens,M. The influence of Problem Format on Students' approach to Mathematical Modelling. ICTMA8
[11] Jacobsen,I., Christerson,M., Jonsson,P. and Overgaard,G. Object-Oriented Software Engineering. Addison-Wesley 1992
[12] An Object-Oriented Environment for Newtonian Particle Mechanics, in Proc. 1st Int. Mathematica Symposium, Southampton (eds. P.Mitic and V. Keränen) CMP. July 1995
[13] An Event-Driven Interface for the AMK Object-Oriented Newtonian Particle Mechanics System, in Proc. 2nd. Conference on Software Engineering (SEHE), Alicante (eds. L. Sucharov, P.Mitic and J-L. Uzo) CMP. November 1995
[14] Mitic, P. Applications of Computer Algebra in Object-Orientated Mathematical Modelling. PhD Thesis, The Open University, England. May 1999.
[15] Ogborn,J. Cognitive Development and Qualitative Modelling, in J. Comp. Assisted Learning 14, pp292-307 (ed. R. Lewis) Blackwell Dec 1998
[16] Ramsden, P. Classroom Chemistry Simulations, Mathematica J., 7, #3, 1999
[17] Shlaer, S. and Mellor, S. Object-Oriented Systems Analysis: Modelling the World in Data. Yourdon Press, Englewood Cliffs,

NJ. 1988
[18] School Mathematics Project, books 1,2,3,4,5,T. CUP 1964
[19] Staffware v8.1   Staffware House, Maidenhead, England.
[20] Gray, J.   Mastering Mathematica. AP Professional. 1994
[21] Abbott, P. Formatted Cells. Mathematica J., 7, #4, 2000