Agent-oriented Modelling for a Billiards Simulation

Monica Farkas Meurig Beynon Yun Pui Yung

Department of Computer Science
University of Warwick CV4 7AL
Tel (+44) 203-523089 FAX (+44) 203-525714

Abstract

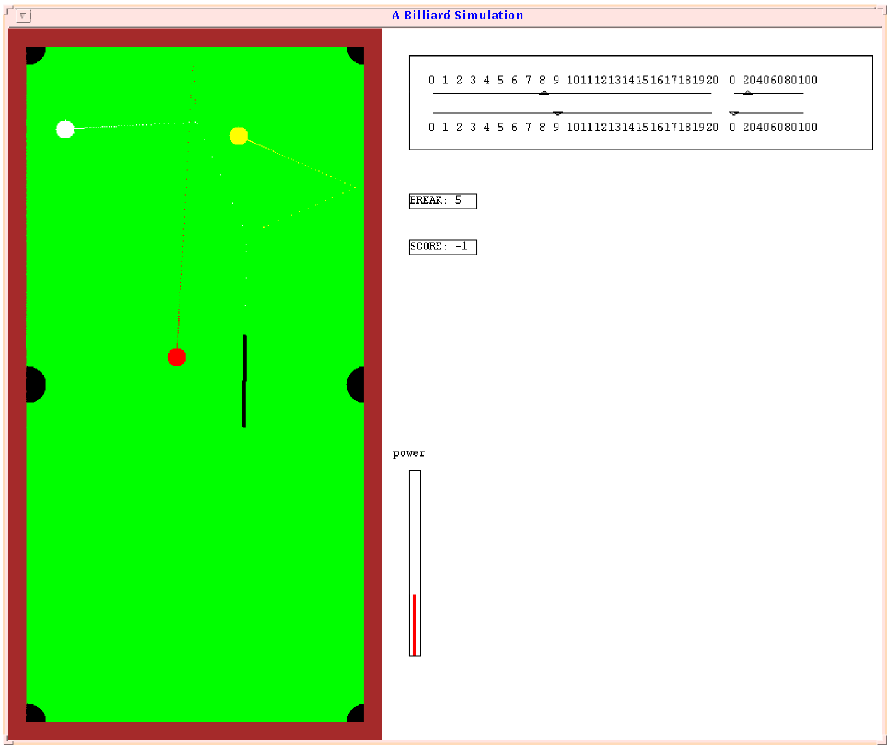
This paper describes the application of an agent-oriented modelling technique outlined in [BBY92] to the simulation of a game of billiards. This case-study is chosen to demonstrate the versatility of our agent-oriented approach, and its potential for integrating discrete event modelling with process control. The scope for modelling a variety of modes of control is explained and illustrated.

1. Introduction

As technology advances, the control of engineering systems becomes progressively more sophisticated. Even prior to cheap computer technology, there was a general trend from machines powered and controlled by the user, to powered machines under user control, to machines with feedback for limited autonomous control. Modern electronics has transformed the potential for control of engineering systems. The central challenge is to devise control systems with "intelligent" feedback to take account of the factors that would enable a human user to operate a machine in relatively uncircumscribed environments.

In this paper, we illustrate a new method of modelling that is well-suited to representing engineering systems subject to different control strategies. A feature of the method is that it involves incremental construction that mirrors the historical progression to more sophisticated levels of control. At first, the designer animates the model. Next, the designer introduces autonomous behaviour into the system. Next, the designer introduces feedback, so that events in the simulation serve to trigger new patterns of autonomous behaviour. Finally, the designer introduces interpretation of simulation events into the model, and makes this interpretation the trigger for autonomous behaviour.

To illustrate our modelling principles, we have developed a partial simulation of a game of billiards. Our simulation is limited as an engineering model, as it presumes idealised physical characteristics, but clearly demonstrates the progression from a simple user-driven preliminary model to a model that incorporates a degree of intelligent control. In other respects, the simulation is very sophisticated, as the principles behind its construction are such that it is just one of any number of behaviours that could be extracted from the underlying model. In principle (setting aside mundane imperfections of our software tools), it would be easy to modify it interactively in quite fundamental ways that were not preconceived in its construction. For instance, we could change the rules of billiards, resize the pockets, or change the simulation to resemble a pinball machine, even in the course of game, or a shot. All this, alas, cannot be illustrated in the following sample display:



2. An Overview of the Modelling Process

2.1. Observations

The first step in our modelling process is the identification of the relevant observations of the system. Notice that by *observations* we refer not to visual observations alone, but to the class of scientific observations that can in principle be made. The concept of observation involves *identity* and *value*. Different values may be associated with the same identity according to context, as happens when we make an observation in a scientific experiment.

The characteristic activity associated with a game of billiards involves actions of the cue, motion of the balls, ball-ball and ball-cushion impacts, and the interpretation of these actions with respect to the rules and scoring conventions. In analysing this activity, we consider those observations we need to make to appreciate what is happening. Examples of relevant observations include: the colour of the balls, their position and velocity, and the cue velocity on impact.

The appropriate selection and classification of observations is an essential part of the modelling process. Certain observations, such as the shape of the table legs, the details of the illumination of the balls, and the absolute location and weight of the table are irrelevant. There are also several *modes of observation* to be considered:

- observations of the balls and table in a static configuration

- observations associated with the balls in motion
- observations concerned with the interpretation of the game.

There is a hierarchical relationship between these modes of observation, in that velocity is defined with reference to position, and the progress of the game with reference to interactions involving the cue, balls and table.

2.2. Agents

Analysis of observations leads us to recognise dependencies that in turn reveal the state-changing agents of the system. Observation of a static configuration of balls on the table gives no clue as to what properties of the observations remain invariant during a conventional game. For instance, someone with no experience of billiards might suppose that the game was played by shaking the table, or by relocating the balls as if they were chess pieces. The dynamic characteristics of the balls and cushions, and the protocols for intervention by the players, can likewise only be inferred from particular modes of observation.

The activity associated with a billiard game can be factored into concurrent threads of state-change. For instance, during the course of a shot, several balls may be in motion, and cushions and pockets may influence their behaviour. The entities that play a role in changing the state of the system are specified as *agents*. This specification has two aspects:

- identifying what each agent can do in principle;
- specifying how and when agents are present and active.

A special-purpose notation LSD is used to specify the agents in this manner. LSD is not itself executable, but is used as a basis for describing simulations. Its function is to classify the observations of a system according to how they are viewed by the various agents. The specification of the referee agent, for instance, implicitly incorporates a specification of the rules of the game. More details of the notation are given in [BBY92] and [NBY93]. Sample LSD specifications for the ball and referee agents are given below:

```
agent referee()
{
state
        status[i]          i = 0 .. 2
        turn = White
derivate
        opponent = (turn == White) ? Yellow : White
        cannon = (turn == White && WhitY && WhitR) || (turn == Yellow && WhitY && YhitR)
        in_off = status[turn] == Dropped
        firstHitR = when (striked != None) ((turn == White && WhitR) happens before firstHitY ||
                            (turn == Yellow && YhitR) happens before firstHitW)
        firstHitY = when (striked != None) (turn == White && WhitY) happens before firstHitR
        firstHitW = when (striked != None) (turn == Yellow && WhitY) happens before firstHitR
        firstHitOpponent = (turn == White && firstHitY) || (turn == Yellow && firstHitW)
        potRed = status[Red] == Dropped
        potOpponent = status[opponent] == Dropped
        miss = (turn == White && !WhitY && !WhitR) || (turn == Yellow && !WhitY && !YhitR)
        posScore = (cannon ? 2 : 0) + (potRed ? 3 : 0) + potOpponent ? 2 : 0) +
                ((in_off && firstHitR) ? 3 : (in_off && firstHitOpponent) ? 2 : 0)
```

```
            negScore = (miss && in_off) ? -3 : (miss && !in_off) ? -1 : 0
            score = posScore + negScore
            end_of_shot = striked = turn && all_ball_stopped
            end_of_player = end_of_shot && score   0
protocol
            end_of_shot ->
                        break[turn] = | break[turn] + score |
            end_of_player ->
                        totalscore[turn] = | turnscore[turn] + break[turn] |;
                        turn = | opponent |;
                        clearshot();
                        clearbreak()
            end_of_shot && !end_of_player && in_off ->
                        ball(turn)(Brown_Spot); respotable[turn] = true
            end_of_player && in_off ->
                        ball(turn)(White_Spot)
            end_of_shot && potRed ->
                        ball(Red)(Red_Spot)
            end_of_player && potOpponent ->
                        ball(opponent)(Brown_Spot); respotable[opponent] = true
}

agent ball(i)(Position)
{
state
            t[i] = | now |
            ball_radius[i]
            ball_pos[i]
            ball_velocity[i]
            ball_init_pos[i] = Position
            ball_init_velocity[i] = (0, 0)
handle
            status[i] = On_Table
derivate
            ball_velocity[i] = ball_init_velocity[i] $\times$ friction$^{now - t[i]}$
            ball_pos[i] = ball_init_pos[i] +

                                          $\int_{t[i]}^{now}$ ball_ velocity[i] dt
            LIVE = status[i] == On_Table
protocol
            ball[i] collides with ball[j] ->
                        ball_init_velocity[i] = ...; t[i] = | now |; ball_init_pos[i] = | ball_pos[i] |;
                        ball_init_velocity[j] = ...; t[j] = | now |; ball_init_pos[j] = | ball_pos[j] |
                                                            j = 0 .. i-1
}
```

Our agent-oriented modelling method corresponds closely to intuitions about the behaviour of real-world systems. Changes to system state are attributed to agents, and are constrained by the capabilities ("privileges to change") of the agents that are currently active. The agents we identify, like the observations we make, reflect the nature of the model and the perspective of the observer. Within the scope of the billiard game, we are concerned with:

- actions governing ball location and motion (cueing, setting up);
- discrete interactions that redirect the ball motion (impacts);
- decisions and responses of the referee (scoring and respotting).

2.3. System Behaviour

There is a significant distinction between an agent-oriented view of system behaviour and a traditional mathematical model. In an agent-oriented view, the system behaviour is only loosely and implicitly defined by the privileges ascribed to agents. General statements about the system behaviour can only be inferred from knowledge of agents. For instance:

- the dimensions of the table and the colour of the balls are not subject to change – there is no agent capable of changing them;
- the effect of simultaneous impact upon of a ball upon another ball and the cushion is unspecified – the actions of the two agents acting on the ball potentially interfere;
- the introduction of new privileges or agents may transform the system behaviour – as would happen if players could cue balls in motion, or a table mover were introduced.

In contrast, a mathematical model of behaviour involves circumscribing the observations and agents in a real-world system once-and-for-all.

3. Computer Modelling and Engineering Prototyping

3.1. Engineering Prototypes

Our framework for modelling systems resembles that used in conventional engineering design, where an "engineering prototype" is developed. The engineering prototype is a physical system with characteristics similar to those of the actual system to be constructed. Interpretation is needed to appreciate the relationship between the prototype and the system in general. In a prototype car, the layout of the engine compartment may be indicated without using fully functional components. In a scale model of a manufacturing plant, illuminated paths might be used to indicate how items are being transmitted through the production line.

By exploiting the power of the computer to represent the states of exceedingly complex systems, we can in principle develop models that represent the system to be constructed in greater detail than is possible using traditional engineering prototyping methods. Simulating experience of the system itself, as in a virtual reality, is only one aspect of effective modelling of this nature. To be useful to the designer, the models we construct must in general "perceptualise" the observations of a system that are required to understand its operation fully, even though these cannot be perceived in interaction with the system. For instance, animation of a circuit diagram can give greater insight into the design of an electrical system than direct experience of the system in operation.

3.2. Spreadsheet Principles

The computer models we develop have essential qualities in common with spreadsheets:

- the values recorded in a spreadsheet at any time correspond to observations of a real-world system *in a particular state*. In this respect, a spreadsheet resembles an engineering prototype, whose state (e.g. illumination of paths representing pipelines) reflects a particular state of the system it represents;
- the redefinition of one cell in a spreadsheet leads automatically, as in one atomic operation, to the re-evaluation of all cells with dependent values.

The effectiveness of the spreadsheet as a modelling tool stems from the fact that automatic updating of cells can faithfully reflect the indivisible relationships between observations that arise in real-world systems.

The modelling tools we have developed exploit generalisations of spreadsheets to represent relationships between observations of a real-world system in a computer model. As a simple illustration of this, consider two observations we may make of a billiard ball:

- ball_pos: a measurement of its location made, for example, by sensing the point of contact between the ball and the table;
- ball_image: the appearance of the ball, as defined by light reflected from its surface.

When we move a billiard ball to a new position, both ball_pos and ball_image may be deemed to take up their new values simultaneously. In contrast, if we were to scent a billiard ball in such a way that the intensity of its scent at a given location depended upon how long and how recently it had resided there, then the change of ball_pos and ball_scent would not necessarily be so closely correlated.

In our computer model, we represent the state of the real-world system by a family of variables, corresponding to relevant observations, together with definitions (similar to the defining formulae in a spreadsheet) that reflect how changes to observations are indivisibly linked. For instance, we can formulate a set of definitions (a *definitive script*) to express the fact that a figure on the display (representing ball_image) has coordinates that are indivisibly linked to a pair of real numbers (representing ball_pos):

ball_image = f (ball_pos).

The notations and techniques used to represent such definitive scripts are described in detail elsewhere (see, for instance [BY90]).

The indivisible relationships between observations in a system are a function of the mode of observation, and are subject to change according to the purpose of the modelling exercise. In astronomy, it is inappropriate to bind the actual location of an object to its apparent location, but the distinction between the actual and perceived location of a billiard ball is irrelevant for many practical purposes. The same mode of observation of a system may apply to many different kinds of modelling task, and is relatively insensitive to whether the context is oriented towards design, experiment or simulation. For instance, our billiard game model could play a part in:

- constructing a billiard table;
- developing a variant of the game of billiards;
- designing a pin-ball machine;

- developing a theory about the motion of billiard balls;
- simulating the behaviour of an actual billiard table in precise detail;
- devising an "electronic billiards" arcade game;
- teaching the rules and principles of billiards to a novice.

Of course, as it stands, it is much better suited to some of these roles than others, but its limitations concern lack of detail in the model (such as no consideration of spin, an inadequate account of friction, poor geometry of the billiard table etc) rather than discrepancies in observation due to the finite speed of light.

In the billiards simulation, many indivisible relationships are represented by derivates in the LSD specification. In the ball agent, for instance, a derivate defines the current position of the ball according to simple laws of motion (cf [BBY92]). In the referee agent, a derivate expresses the way in which the referee detects "at the speed of thought" that a cannon has been scored.3.2. Virtual Scientific Reality

The computer model supplies an environment, similar to an engineering prototype and a spreadsheet, that mirrors the observed state of the real-world system it represents. The correspondence is established by a correlation between observation of the actual or conceived real-world system and observation of the computer model. Selected observations of the real-world system have a counterpart in the computer model, so that, for example, ball_pos is represented by a pair of floating point numbers, and ball_image by a collection of pixels on the computer screen.

The logical extension of the principles we apply might be described as *virtual scientific reality*. The computer model generates perceptible outputs that the modeller interprets – or perhaps even experiences – as if they stemmed from the actual system being modelled. Our primary objective differs from "virtual reality" in that perceptualisation of scientific observations has greater priority than faithful imitation of direct experience. Of course, what appears on the screen is not a billiard table, but the screen image is a useful approximation to our view of a billiard table. In the correspondence between the two physical systems: the actual billiard table, and the representation of a billiard table that the computer generates, there is a trade-off between faithfulness and flexibility.

The observations of the computer model represent real observations by analogy, but convey only an approximation to part of what can be learned through observation of the real system. The advantage of the computer model derives from our scope to interact with it, whether (as in design) to simulate perturbations to the real system or (as in modelling) to refine the mode of observation: observing particular values more accurately, or taking account of additional observations. This advantage is exploited in the representation of analog variables, whose values can only be correctly defined with reference to the observer's ability to observe changing values arbitrarily frequently.

4. The Billiard Game Simulation

4.1. Command-driven Simulation

The development of the billiard game simulation reflects the systematic elaboration of observations and agents described in §2. The first parameters to be introduced are the dimensions and attributes of the balls and table; these correspond to observation of a static scenario. At this stage, each agent specification has a very simple form, comprising state variables alone (such as ball_pos, and ball_radius). Agents of this nature are object-like: they possess attributes, but have no stimulus-response capability. The grouping of observations by "object-like" agent in the early stages of our development resembles the initial phase of an object-oriented design method (cf Shlaer-Mellor [SM92]).

Even at this preliminary stage of the modelling process, definitions to bind the visual image of a billiard ball to its location are introduced. In the present simulation, each ball is depicted in a rudimentary fashion by a circle. The profile of the table and the pockets is similarly crudely represented, using points, lines and circles in the definitive notation for line-drawing DoNaLD [BY90].

An important feature of our modelling technique is the potential for interactive enhancement of the model at any stage. For instance, in principle it would be possible to substitute definitions of 3-d images of the table and balls if desired, without modifying other aspects of the model that were independently developed. (In practice, our experiments with definitive notations for 3-d graphics are at an early stage, and the speed of the visualisation process is also a serious concern.)

Though the behaviour of agents is yet to be introduced to the model, the potential for experimenting with state is present – the model developer can act as an omnipotent agent, to redefine parameters at will. For instance, it is possible to re-position the balls, to change their colour, or alter the dimensions of the table. In this respect, the model resembles a spreadsheet, where the user can perform "what if?" experiments on parameters and observe their effect. In this mode, the developer is operating within the simplest of the control paradigms described in §1; changes to the system state are "command-driven" under the sole control of the user.

4.2. Simulation of Autonomous Behaviour

The next level in the control hierarchy is the introduction of autonomous behaviour to the model. In the billiards game, this involves both discrete events – such as impacts, or actions taken by the referee, and processes – such as ball motion.

Autonomous behaviour is represented in terms of stimulus-response patterns. These are expressed in the protocols of the agents. A stimulus-response pattern may reflect a reaction we regard as inevitable, such as that of the ball on impact, or a potential action performed voluntarily, such as the referee takes when a ball has been potted. Whether we deem a response to be voluntary or obligatory is specific to the modelling context. The referee's actions are dictated by the rules and etiquette of the game of billiards. The manner in which the ball responds to impact is determined by the physical assumptions we choose to make (e.g. is the impact smooth? do we invoke Newton's Laws or Poisson's hypothesis [KR66]).

The introduction of agent protocols resembles the automation of a function that was formerly performed by the user of a mechanical device. Prior to the specification of the referee agent, the model developer can locate the balls at will by redefining ball_pos; in the final simulation, the red ball is automatically respotted after potting. After the referee's role has been specified, the computer model exhibits the second level of control, whereby the user can instruct other agents to operate in appropriate circumstances ("delegation of control").

A different mode of observation is required to model the effect of autonomous processes, such as the continuous motion of the balls. Such behaviour is to be interpreted within a frame of reference that includes *observation of a clock*. Once a ball is set in motion, its subsequent behaviour can be predicted –subject to simplifying assumptions –by a theory. In specifying this behaviour in our model, we introduce analog variables to represent the current position and velocity of the ball. Conceptually, the values of analog variables are defined with reference to the clock in accordance with the equations of motion (cf [BBY92]).

The representation of analog variables exploits the flexibility to modify the computer model discussed in §3. The values of analog variables can only be represented approximately, but are subject to be represented and evaluated to greater accuracy under the control of the modeller. By automating this refinement process, we obtain a convenient method of introducing adaptive solution of the equations that describe processes. For instance, when balls approach the point of collision in our simulation, the step-size of the integration used to predict their position and velocity is influenced by their speed (see below).

4.3. Control with Feedback

Our agent-oriented modelling method attributes changes in system state to the action of agents. The action of the cue causes the ball to move. The potting of a ball causes the score to be incremented. Causality is a common abstraction behind the use of definitive scripts to represent indivisibly linked changes to observations and the stimulus-response patterns in agent protocols.

Feedback is associated with cyclic causation patterns [R83]. For instance, ball movement causes ball impact, and ball impact in turn affects ball movement. In televising a game of billiards, feedback is involved in the production process when a shot is replayed in slow motion to resolve a contentious issue, such as whether or not two balls made contact.

Our modelling methods are well-suited to implementing feedback. Because of the explicit representation of the observed state of the system during transition, it is possible to reference all kinds of information about the current system status and behaviour. Because it is possible to introduce new stimulus-response patterns into the system interactively, it is then easy to influence the system behaviour in a context-dependent way. Note that, in describing feedback for process control, it is essential to deal effectively with stimuli that take the form of "instantaneous" events: in our approach, this is achieved through the mechanism for adapting the simulation time-step described above.

A particularly powerful mechanism for feedback involves augmenting the definitive script that describes system state so as to include interpretations of low-level behaviour, then introducing agents that exploit these interpretations to influence low-level behaviour. For instance, we can readily implement a mechanism whereby a break is automatically terminated when the number of consecutive scoring shots without a ball being potted exceeds the permissible maximum. This type of control, in which intelligent interpretation of a complex history of ball interactions is used to guide the playing protocol automatically, is a form of "intelligent feedback".

The potential for intelligent feedback can also be appreciated from the subtle ways in which the modeller can make use of knowledge drawn from a whole variety of sources in controlling the simulation. The adaptation of the sampling rate associated with imminent critical events illustrates this.

Without any adaptation, balls that travel fast can apparently intersect each other to an arbitrary extent, or even pass through each other without seeming to make contact. Whatever fixed sampling rate we choose, there are always speeds at which these anomalies arise. In such cases, the point of impact of two balls cannot be accurately computed. If the speed of the balls is arbitrary we can in principle adapt the sampling rate dynamically to take account of this, but only at great cost in the speed of simulation. In practice, we can assume some limit on the speed at which billiard balls travel, and incorporate this in the model of the cue stroke. We can also assume that the physical characteristics of billiard balls preclude more than a certain amount of compression on impact. The computational strategy for adapting the sampling rate in our simulation is a compromise that reflects all these factors: estimating the point of contact to a reasonable accuracy, without slowing the simulation more than is necessary or beyond a reasonable limit, taking account of the physical parameters defined by the modelling context.

5. Concluding remarks

5.1. Background to the Paper

The chosen case-study and the theme of this paper are based upon an assignment carried out by Farkas in connection with a short postgraduate course on concurrent systems modelling given by Beynon. The first prototype was developed by Farkas after a short period of familiarisation and practical experience with the relevant ideas and software tools. All subsequent development of the simulation is due to Yung, and the text of this paper is due to Beynon. There is also a second paper developed in a similar fashion [NBY93].

5.2. Context for the Research

The present case-study is one of several simulations currently being developed in connection with a research programme aimed at creating new tools and methods for software design. The results of our work have highlighted a number of issues that need to be addressed (cf [NBY93]), but have also given encouraging feedback.

It has proved moderately easy for inexperienced users to develop simulations using agent-oriented analysis in conjunction with definitive state-transition models. To some extent, this is surprising, in view of the evident need for a better understanding where the analysis and classification of observations and agents is concerned, and the limitations of our current software tools, which are research prototypes rather than commercial products. A possible explanation is that it is easier to keep conceptual control when modelling with definitive scripts, even though the development environment and built-in features available are not those best suited to the task. Re-use of fragments drawn from other models (such as the Vehicle Cruise Controller [BBY92]) has also played a significant part.

Yung has been able to adapt and enhance the prototype simulations with very little additional input from the original authors. In this respect, the documentation provided by an LSD specification has proved helpful in directing attention to the most significant issues for the simulation. The power to experiment within the simulation environment has also been significant in distinguishing misinterpretations of behaviour from mistakes in the model.

The principles we adopt in our modelling approach restrict the range of variation within models of a phenomena. In our models, there is a much more intimate connection between form and content than in a typical computer program. This is of particular interest in connection with Smith's concern for "less promiscuous modelling" in the programming process [S87]. It also helps to explain why, though program testing is acknowledged to be a quite inadequate way to guarantee correctness, there is good evidence to indicate that exploration of our models through simulation of critical scenarios is an effective way to eliminate conceptual errors and hidden assumptions.

As an illustration of this, the original protocol governing ball impact took the form:

balls a ball diameter apart $\rightarrow$ *transfer energy via laws of smooth impact.*

For certain combinations of angle of impact, ball speeds and sampling rate, this protocol can lead to a situation in which balls intersect on first impact as if subject to compression, and so become less than one diameter apart. If the balls do not then separate fast enough they will appear to be in contact at the next moment of sampling. The effect is to produce a combined motion of the balls such as might be observed if the colliding balls were to become instantaneously attached via a spring.

Acknowledgements

We are grateful to Valery Adzhiev, Alan Cartwright, Mike Joy and Steve Russ for useful discussions relating to this paper. We are pleased to acknowledge the financial support of the SERC under grant GR/J13458.

References

[B86] Booch, G., *Object-Oriented Development*, IEEE Trans Software Engineering, SE-12(2), 1986, 211-221

[Br87] Brooks, F.P., *No Silver Bullet: Essence and Accidents of Software Engineering*, Computer April 1987, 10-19

[BY90] Beynon, W.M., Yung. Y.P.,*Definitive Interfaces as a Visualisation Mechanism*, Proc. Graphics Interface '90, Canadian Inf. Proc. Soc. 1990, 285-292

[BBY92] Beynon, W.M., Bridge, I., Yung. Y.P., *Agent-oriented Modelling for a Vehicle Cruise Control System*, Proc ASME Conf ESDA '92, Istanbul, Turkey 1992, 159-165

[KR66] Kilmister, C.W., and Reeve, J.E., *Rational Mechanics*, Longmans 1966

[NBY93] Ness, P., Beynon, W.M., Yung, Y.P., *Applying Agent-oriented Design to a Sail Boat Simulation* , Proc ASME Conf ESDA '94, 1994

[R83] Roberts, Nancy et al, *Introduction to Computer Simulation: A Systems Dynamics Modelling Approach*, Addison-Wesley 1983

[S87] Smith, B.C., *Two Lessons in Logic*, Comput. Intell. 3. 214-218 (1987)

[SM92] Schlaer, S., Mellor, S.J., *Object Lifecycles: Modeling the World in States*, Yourdon Press, Prentice Hall 1992

[T88] Tomiyama, T., *Object Oriented Programming Paradigm for Intelligent CAD Systems*, Proc Intelligent CAD Systems II, 1988, 3-16