MSDL Talk: Intro to Hybrid System Modelling

Simon Lacoste-Julien
School of Computer Science, McGill University
Montréal, Québec, Canada

July 10, 2002

Abstract

This paper outlines what I've said in a talk for the Modelling, Simulation and Design Lab at McGill University on June 12, 2002. I gave an introduction to Hybrid System Modelling and on what I'm working on this summer.

Contents

1 Intro to Modelling of Physical Systems

I give here the essence of physical modelling, from a physicist point of view, and without getting into the modelling formalism details like validation of models, the relation between the experiment and the model, etc. which I guess are explained in the course 308-522A given by Hans Vangheluwe...

1.1 What is Modelling?

Modelling a physical system is to give a description of its **structure**, that is, to describe the state of the system (physical properties) and the state transition mechanism (physical laws). On the other hand, **simulating** a physical system is to describe its **behaviour** when specific inputs are given to the system (like initial conditions, driving forces, etc.) (see [1]).

When doing experimentation on a system, we study its behaviour. We can then try to infer a partial model for the system. Then, using simulation, we can compare with actual experimental data to evaluate the validity of the model.

1.2 Three Main Steps when Modelling:

1. Determine the **goal** of the model. This gives the framework in which you will work and influence the assumptions you need to take.

   For example, say we want to build a pool table model. Depending if we're a player or a pool table builder, the objective for the model will be different. The player will care mostly about the trajectory of the balls, so that their composition or their cost won't need to be included. On the other hand, the pool table builder will want to care about those parameters.

2. Determine what is needed to *describe the* ***state of the system:***

   - **State variables** (those are usually called generalized coordinates in physics; they can be the temperature of the system, the position and velocity of its constituents, etc.) They give the *quantitative* description of the state of the system.
   - **Categories** (this is my own terminology I have just created for this talk!) These are used to describe the *qualitative* states of the system (like in a Finite State Automaton (FSA), for example: hot vs. cold). They could take the form of nodes in a graph, for example.

3. Determine the **laws** and **interactions** in the system which describe its *evolution in time* and the **constraints** relating its state variables. For a FSA, you specify the state transition function, for example. For a continuous system, you give Ordinary Differential Equations (ODE) or Differential Algebraic Equations (DAE) for more generality. In this step, you also determine the **parameters** of the system, which are usually defined as quantities which stay constant during a simulation run, but which can

vary from one system to another (example of parameters are the gravitational acceleration constant g; the length, width, mass of the system; etc. By comparison, state variables are e.g. temperature, position, velocity, number of people in a train, amount of fuel in a tank, etc.)

1.3 Pool Table Example

Say we want to model a pool table from a player point of view. So the steps become as follow:

1. The goal for a player is mainly to know the trajectory of the ball.
2. The state variable of the system would be the x-y position of the ball (x, y) and its x-y velocity (v_x, v_y).
3. The physical laws would be Newton laws and the law of reflection (when have collision with wall), say. The parameters would be the dimension of the ball and the table.

2 What is Hybrid System?

A **hybrid system** is a physical system that we model using continuous components as well as discrete components. For example, a bouncing ball is a hybrid system since the evolution of its state variables (position and speed) vary continuously according to Newton laws when falling, but have a discrete change (speed is reversed) when entering in collision with the ground. Note that the discrete change is due by the way we *model* the collision (for simplicity); in reality, everything is continuous (more on this later).

To model the continuous components, we use ODE's ($\dot{x} = f(x,t)$), DAE's ($f(\dot{x}, x, t) = 0$) or even Partial Differential Equations (PDE's) (I won't talk about these though). To model the discrete components, we use FSA and DEVS (Discrete Events Specification).

3 Why using Hybrid System?

Or the question could be stated: why discreteness? Indeed, from a classical mechanical point of view, everything is *continuous* in nature: for example, the collision of the ball with the ground is not an instantaneous event, but in reality is the continuous deformation of the ball (like a spring) which will have the effect at the end of having inverted its speed. This continuous world is no more true in Quantum Mechanics (even though we still use continuous wave functions to model everything...), but this point is not relevant at our level of modelling since QM is rarely present in engineering situations. So then why do we want to use discrete models?

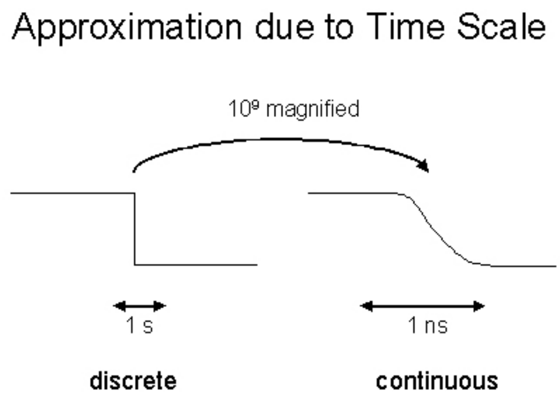

Figure 1: **Falling edge of a computer clock signal.** Depending on the time scale under study, a phenomenon will appear continuous or discrete.

3.1 Abstraction

We want to focus on the important things, so we abstract the details. For example, if we model a queue or train, we don't care about the exact trajectory of the people when entering the train. All we care is if the person is in the train, or not. So we'll consider only the number of people in the train, not their exact position. This will yield a discrete behaviour in our model since the number of people will be an integer.

3.2 Approximation

The real world is very complicated. To deal with this complexity, we have to make approximations. The bouncing ball is a good example. We usually don't know how the ball will deform itself when colliding with the ground (and we usually don't care anyway). A good first approximation (which is discrete) is to assume that the ball will have its speed inverted after the collision. This is an approximation since in reality, the collision is not elastic, i.e. some kinetic energy is transformed in thermal energy due to deformation of the ball (which could have been modelled more precisely as a damped spring). Also, some momentum is transferred to the ground (but the ground is so heavy that this is negligible...).

This approximation business is closely related to the **time scale** we are interested in. For example, if we want to study effects on the ball which occur in a microsecond period, then we'll have to study in details its collision with the ground and not consider it as instantaneous. The time scale gives us approximation thresholds for the problem. Figure 1 shows the effect of the time scale on the continuity of the falling edge of a computer clock signal.

3.3 Control systems

Discrete behaviour happens often in control systems, where switches are usually used (electrical switch, for example). A canonical example of control system used in the modelling world is *bang-bang control*. This can be used, for example, to keep the speed of a car inside a speed range. When the car is going too slow, it starts accelerating; when it is going to fast, it starts breaking (the acceleration undergoes a discontinuous change in bang-bang model). We'll come back to this example in section 4.

3.4 Discontinuity is inherent to some systems

This is the case in statistical mechanics during phase transition. Because of the so large number of components in a statistical mechanic system (of the order of 10^{23}), the variations in state variables can be very steep. We can thus consider those variations as discontinuous, which is by the way one of the sign of a phase transition.

3.5 Efficiency

Finally, discrete systems are simulated much more efficiently than continuous ones.

3.6 Hybrid systems

So for all the preceding reasons, we will often use discrete systems to model physical systems. But there can still be some continuous behaviour that we have to take into account (like for the bouncing balls), so we'll want to model both the discrete and continuous behaviour at the same time in our model. For this we need hybrid systems.

4 How are they implemented?

4.1 Model: FSA + ODE

One way to model hybrid systems is using a finite state automaton (FSA) which describe the discrete allowable states, each of which contains a specific set of ODE's. Figure 2 shows an example of such a representation for the Bang-Bang control system of a car. The FSA contains two states: an accelerating one and a breaking one. In each state, the ODE's describing the behaviour of the system are given. The transition between the two states is determined by the guards $v > v_{max}$ and $v < v_{min}$. How these guards are implemented is given in next section.

I will briefly mention here that one can want to add the **event scheduling formalism** to the model. Since I haven't studied this in details yet, I won't say much about it. But one has to be aware that this can be included in Hybrid

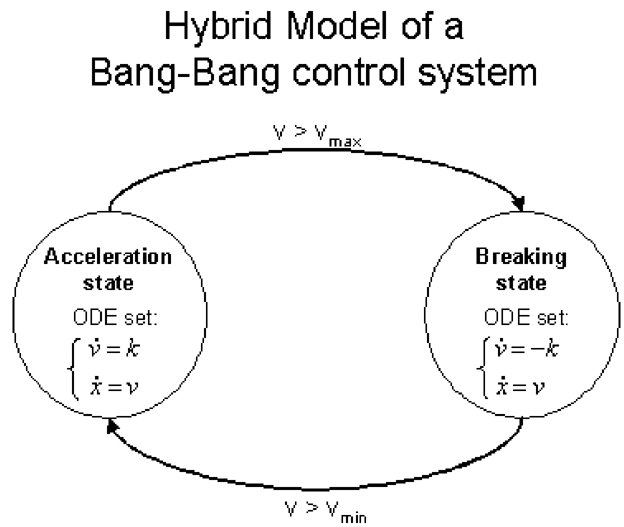

Figure 2: **Hybrid model of a Bang-Bang control system using FSA of ODE's.** x is the state variable for the position of the car. v_{max} and v_{min} are parameters which give the desired speed range of the car.

Systems (and it was included in the simulator described in section 5). Also, for more expressiveness, the FSA can be replaced by DEVS.

4.2 Simulation: using monitoring functions for transition

To simulate a normal continuous system, we simply use a numerical ODE solvers (like Euler or Runge-Kutta algorithm) which takes initial conditions as inputs and can give the value of the state variables at any time requested. The problem happens when we want to simulate also the transition of states, i.e. the transition from one ODE set in the FSA to another one. Recall that in our model, we used guards to trigger transition.

So how are the guards implemented? One way to do it in the code would be to use if-statements:

$$if \quad v > v_{max} \quad then \quad \ldots$$

This is not very good for our simulation purpose since we usually wants to know *when* the transition occurs, and without getting too much in the details of simulation, we can just say that this method doesn't yield this information because of the way the ODE numerical solvers are implemented. So in order to have this information, **monitoring functions** are used and the transitions are triggered by **zero-crossing detection**. For the example in figure 2, the monitoring function in the acceleration state could be $f(v) = v_{max} - v$. When this function passes from positive to negative, we know we have obtained $v >$

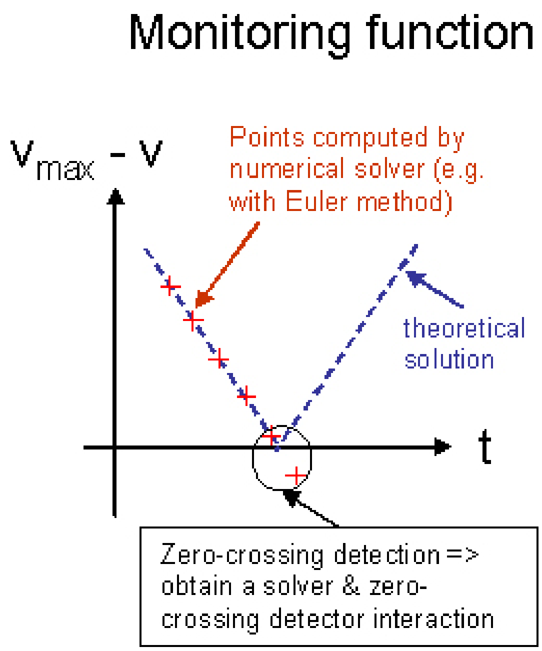

Figure 3: **Monitoring function for Bang-Bang control system.** Notice that the solver has jumped over the zero-crossing point. It will need to back-off in order to find the exact zero-crossing time.

v_{max} so that a transition should occur. The monitoring function of a typical simulation run of our Bang-Bang example is shown in figure 3. The process of finding the zero-crossing point is non-trivial since the numerical solver only gives the value of the state variables at discrete time. So as shown in the figure, the zero-crossing point is usually skipped, and the simulator has to make the solver back-off to find points in between until it finds the exact zero-crossing time. This usually means some interaction between the numerical solver and the zero-crossing detector...

5 Simulator developed in 308-522A

I now give a brief overview of a Hybrid Simulator which was developed by Olivier Dubois and Eric McSween for the course 308-522A in Fall 2001 (see http://www.cs.mcgill.ca/ emcswe/cs522/project/index.html for their web page [1]). For the remaining of the presentation, I'll call their simulator the 522 Simulator. This is the simulator I have been using for the past month to study the

[1] Note: their class diagram is not consistent with their final program. You better stick with the one I have provided in this presentation.

modelling of Hybrid Systems.

5.1 Simulator structure

First of all, when we talk about a complete simulator, we're talking about a simulating environment which would need two components: a *modelling environment* in which we create our models, and *simulator kernel* which is used to simulate our models. For the 522 Simulator, the simulator kernel was used with a GUI, whereas to create the models, we need to program them in python (using inheritance of some specific classes which provide the modelling environment in some sort).

Figure 4 shows a rough (incomplete) UML Class diagram for the 522 Simulator. Three divisions are clearly shown: the simulator kernel, the modelling environment and the model implementation. The details of the simulator kernel are not shown for simplicity. The modelling environment consists of two superclasses: Model (provides the Model interface to the simulator) and FirstOrderDE (which has to be the parent of any specific ODE set). To implement a specific model we need to build a subModel class which will contains the details of the model. If the model contains ODE's, we need to implement those ODE sets by creating a subclass of FirstOrderDE (one subclass of each ODE set). For example, in the Bang-Bang control system example, we could build two subclasses of FirstOrderDE: the BreakingODE and the AccODE which would represent the breaking state ODE and accelerating state ODE respectively.

In the subModel class, we need to provide the list of state variables in the model (modelvarnames), a list of the parameters (params), the current state ODE set (in diffeqs), and finally, we need to give a list of triplets for the zerocrossingfuncs variable, which implements the monitoring function mechanism. The triplets contain three things (!): a monitoring function, a condition of crossing which triggers an event (like from positive to negative, negative to positive, or both), and finally an event handler function. So indeed, we need to provide also the monitoring functions definitions and the event handler definitions, which will state what happens what a certain event happens (like changing the current ODE state by changing the diffeqs variable). Finally, we need to provide a start() function which is called by the simulator when the model is initialized.

For the subODE classes, we simply need to provide a derivative function for each of the variables in the ODE set, to represent the $\dot{x} = f(x, t)$ system (where x is a vector). The parameters are also specified (they can then be changed at runtime by the event handlers in the subModel class). A commented example of model implementation of a pool table can be found at http://moncs.cs.mcgill.ca/people/slacoste/research/files/PoolTable.py.

5.2 Screenshots and examples

To start the 522 Simulator kernel, we execute the GUI.py file. Figure 5 shows the GUI interface when first starting the simulator. The first thing we need to do is to load a model, by pressing the "click here!" button. I'll load for my first

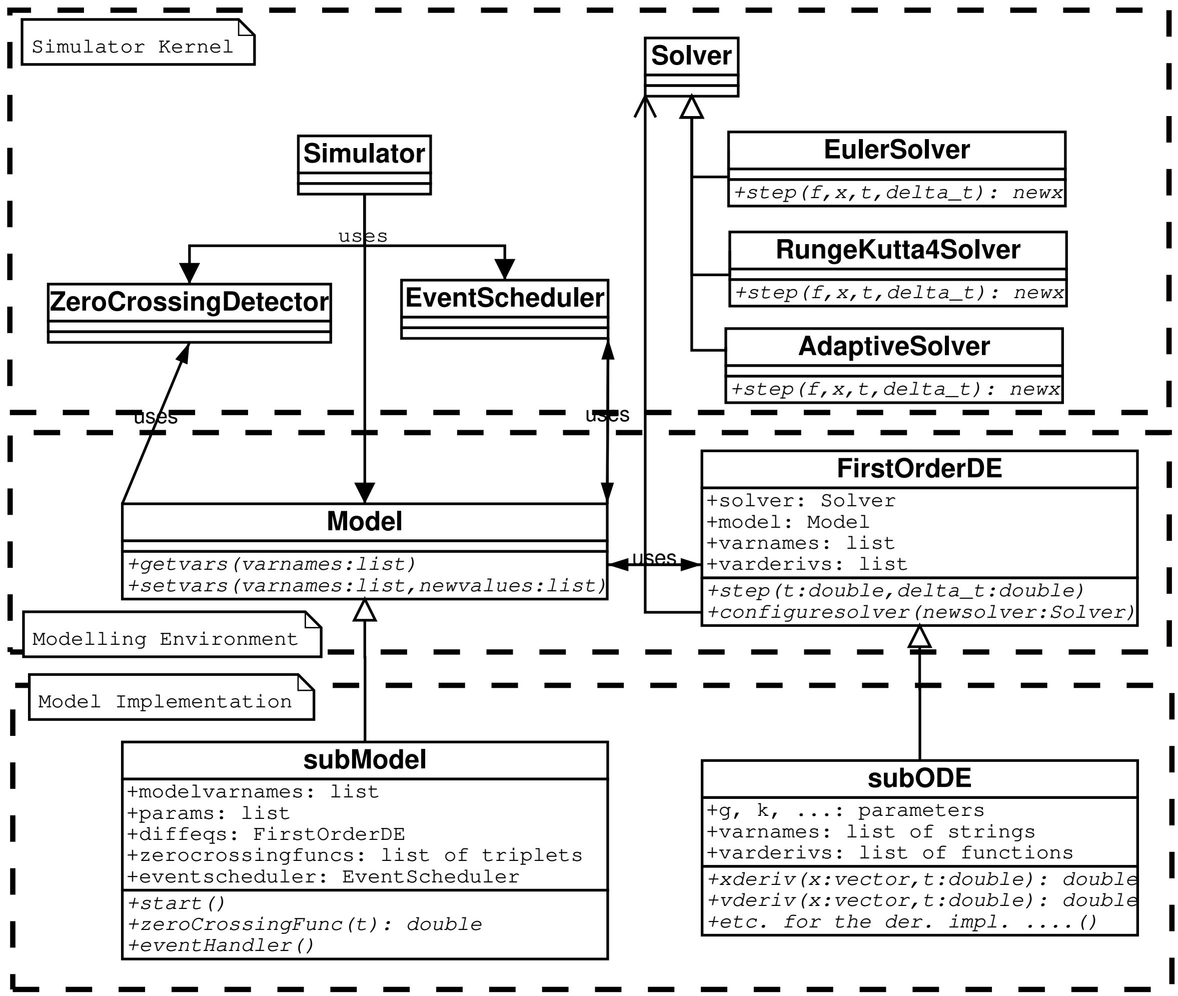

Figure 4: **Rough UML Class diagram for the 522 Simulator.** Notice the three divisions for the Simulator: the simulator kernel, the modelling environment, and the model implementation

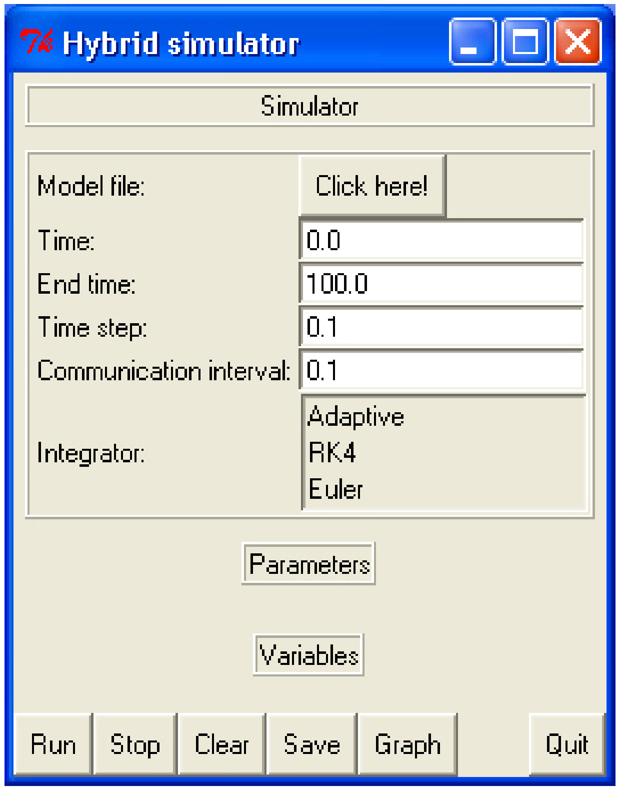

Figure 5: **GUI of 522 Simulator.** To load a model, simply press on the *click here!* button.

example the module PoolTable.py which contains the subModel and subODE classes for a pool table with a magnetic field model.

Figure 6 shows the interface after having loaded the model file (PoolTable.py). The first thing we can notice is that the parameters and state variable names are automatically loaded.

Let me now describe the different fields of the simulator. The "time" field represents the current time of the simulation run (it can be changed to change the starting time of the run). The simulation run is simply started by pressing the "run" button. As the simulation runs, the "time" field and the state variables field are updated to show the evolution of the system. The amount of time before each update of the fields is determined by the "communication interval" field. The "time step" field determines the length of the integration step for the numerical solvers (the smaller the time step, the more precise is the integration). The numerical solver (integrator) can be chosen by clicking the appropriate solver in the "integrator" field (Euler is chosen by default).

In order to have information about the simulation run, we can create graphs

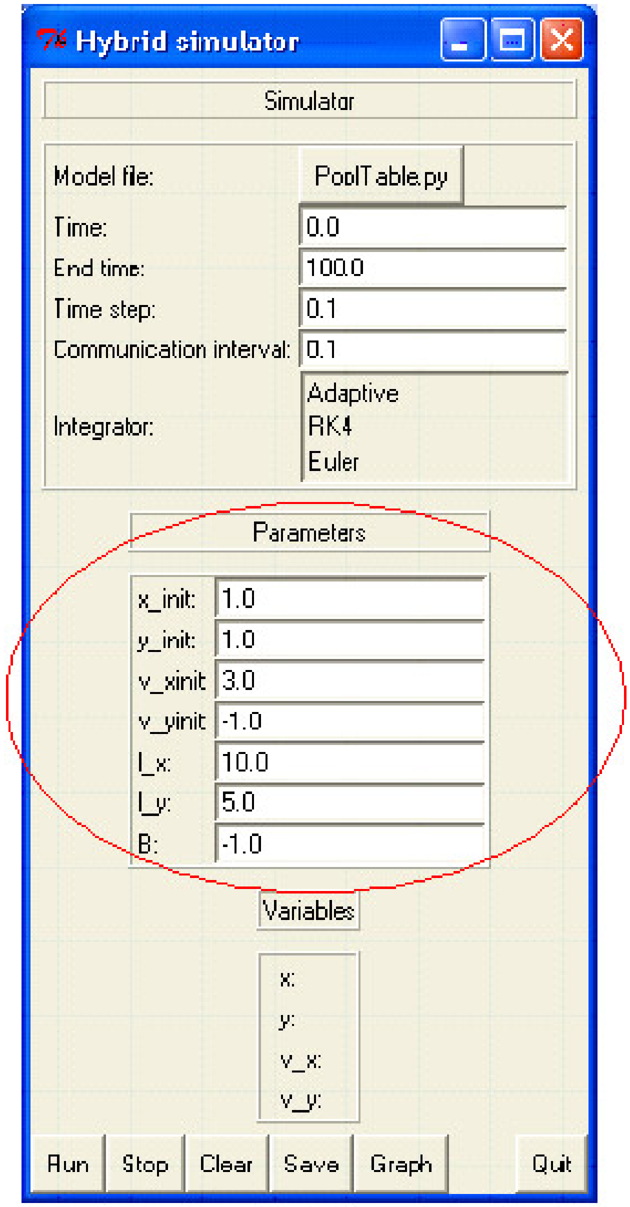

Figure 6: **After having loaded PoolTable.py** Notice that all the parameters that can be changed in my model are automagically loaded! The x_init and stuff parameters represent the initial conditions for my ball. The l_x and l_y are the width and height of the 2D pool table. B is the magnetic field strength. To know the units, you have to look at the comments in the model file... Also, the state variable names are also loaded.

by clicking on the "graph" button (each click creates a new graph). Several trajectories can be added in the *same* graph by clicking on the "new" button in the appropriate graph. Figure 7 shows the dialog box to edit a trajectory. We decided to graph the y-position of the ball with respect to the x-position, in order to follow the trajectory of the ball. The style for the graph was chosen to be black dots.

When we press the button "run", the simulation starts and the graph is automatically updated (and resized if needed) at the same rate as specified by the communication interval. A sample run for the pool table example is shown in figure 8. Note that we have turned off the magnetic field B (set it to 0) in order to show the periodicity of the problem (and that the collisions with the walls are well detected by the zero-crossing detector). Figure 9 shows another run with a magnetic field of -1.0. A very nice picture! In this case, though, the size of the time step and the numerical solver did make a difference. See the note under the figure for more info.

Figure 10 shows a simulation run for a model of two discs with electromagnetic attraction. I have slightly modified the GUI python code in other to make the whole model fits in the screen. The collisions between the discs are assumed to be elastic (these collisions are what make this model hybrid). In this model, r1, m1 and q1 are the radius, the mass and the charge respectively for the first disc. k is the Coulomb's constant. The graph on left of figure 10 shows the horizontal speed of disc 1 (blue) and disc 2 (red) with respect to time. The graph on right shows the trajectory of both discs (using y-position with respect to x-position) (disc 1 in blue, disc 2 in red). The numerical solver was RK4. The symmetry of the trajectory is due to the symmetry of the initial conditions. For comparison between numerical solvers, the exact same problem (same initial condition) but simulated with Euler (with the same time step) is shown in figure 11. We can see a very different trajectory, thus indicating that the solver we use does really matter in our simulation (RK4 is more accurate than Euler).

Finally, we can show some other examples. Figure 12 shows the same model but with the mass of disc 2 ten times heavier than the mass of disc 1. The simulation clearly shows the difference (which is due by the conservation of momentum and energy laws). Figure 13 shows a simulation run with a slightly different initial condition for disc 2, thus breaking the symmetry of our problem. The resulting trajectory is a lot more complex, maybe chaotic.

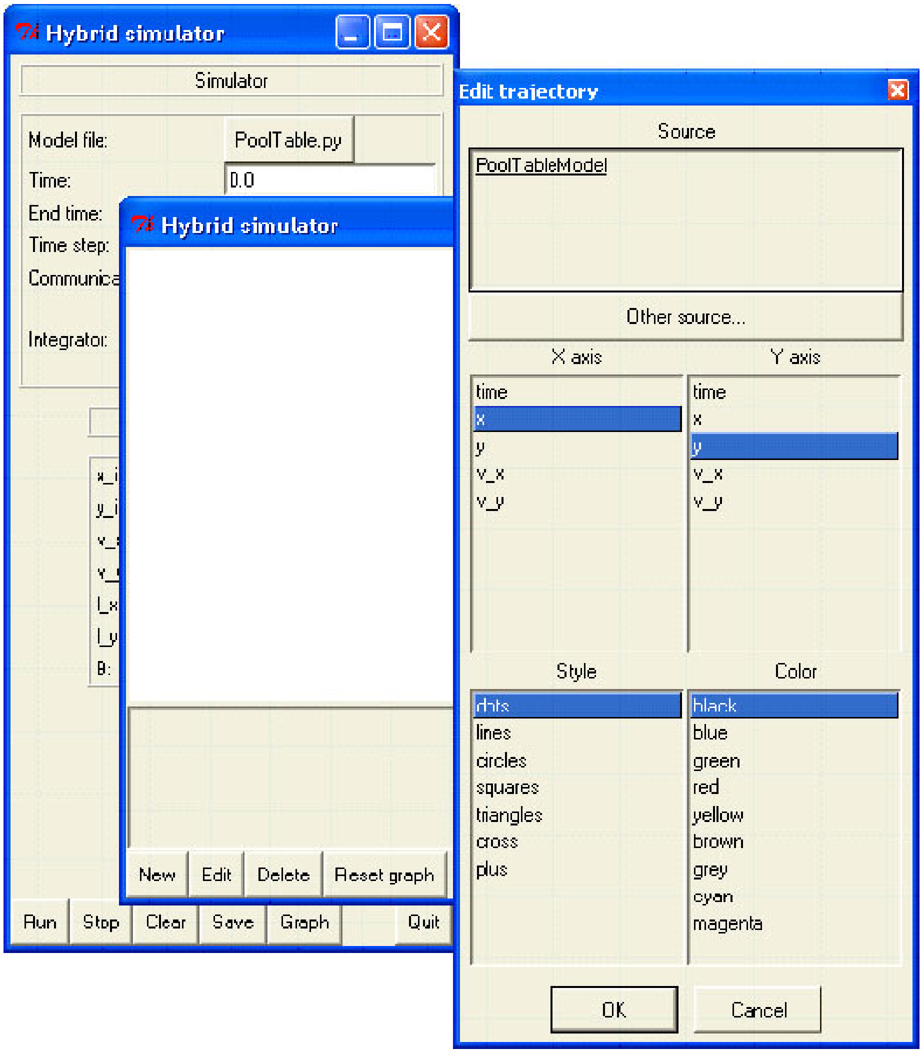

Figure 7: **Creating a new trajectory in a graph.** For each axis, all the state variables (plus the time) are all available to be graphed. In this example, we are graphing the y-position of the ball with respect to its x-position.

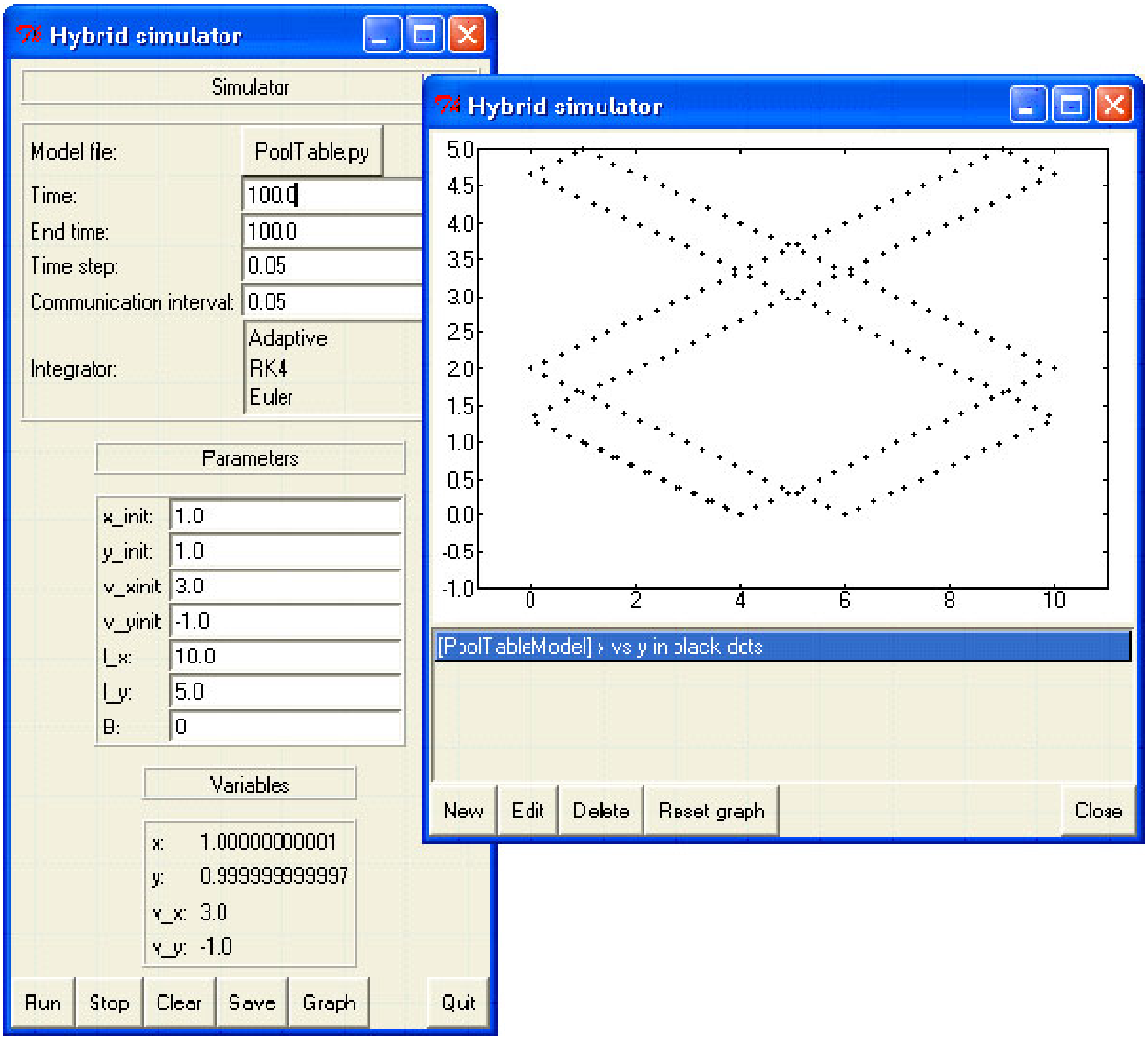

Figure 8: **A sample run for the pool table with no magnetic field.** Notice the regularity of the trajectory of the ball (due to the symmetry of the table), which shows that the simulator is well-behaved (no apparent imprecision yet for this problem).

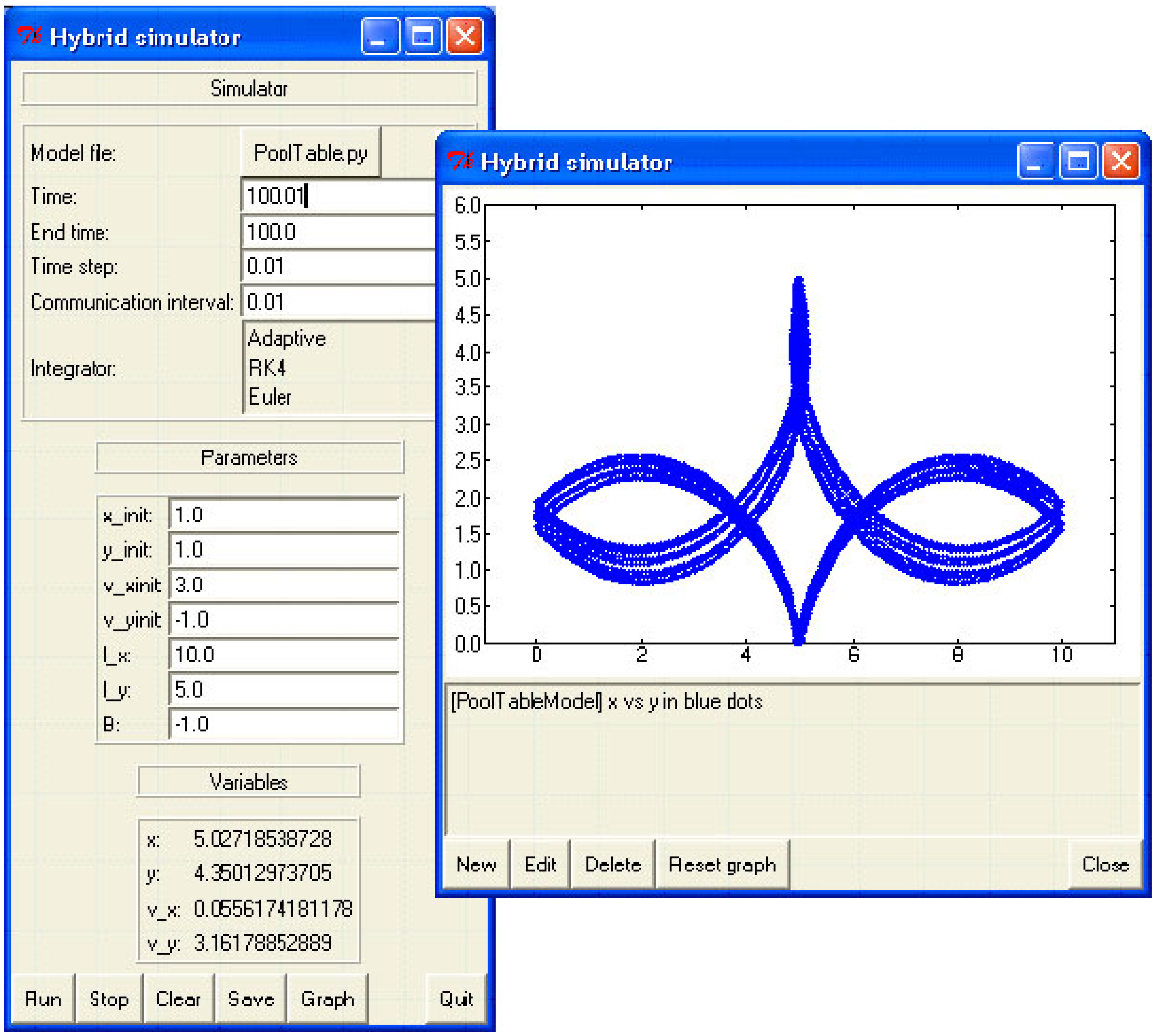

Figure 9: **Pool table with a magnetic field.** This simulation run was done with RK4 with a time step of 0.01. Note that with Euler and a time step of 0.1 or greater, the simulation sometimes crashes (infinite loop, ball traverses a wall, or other weird things)...

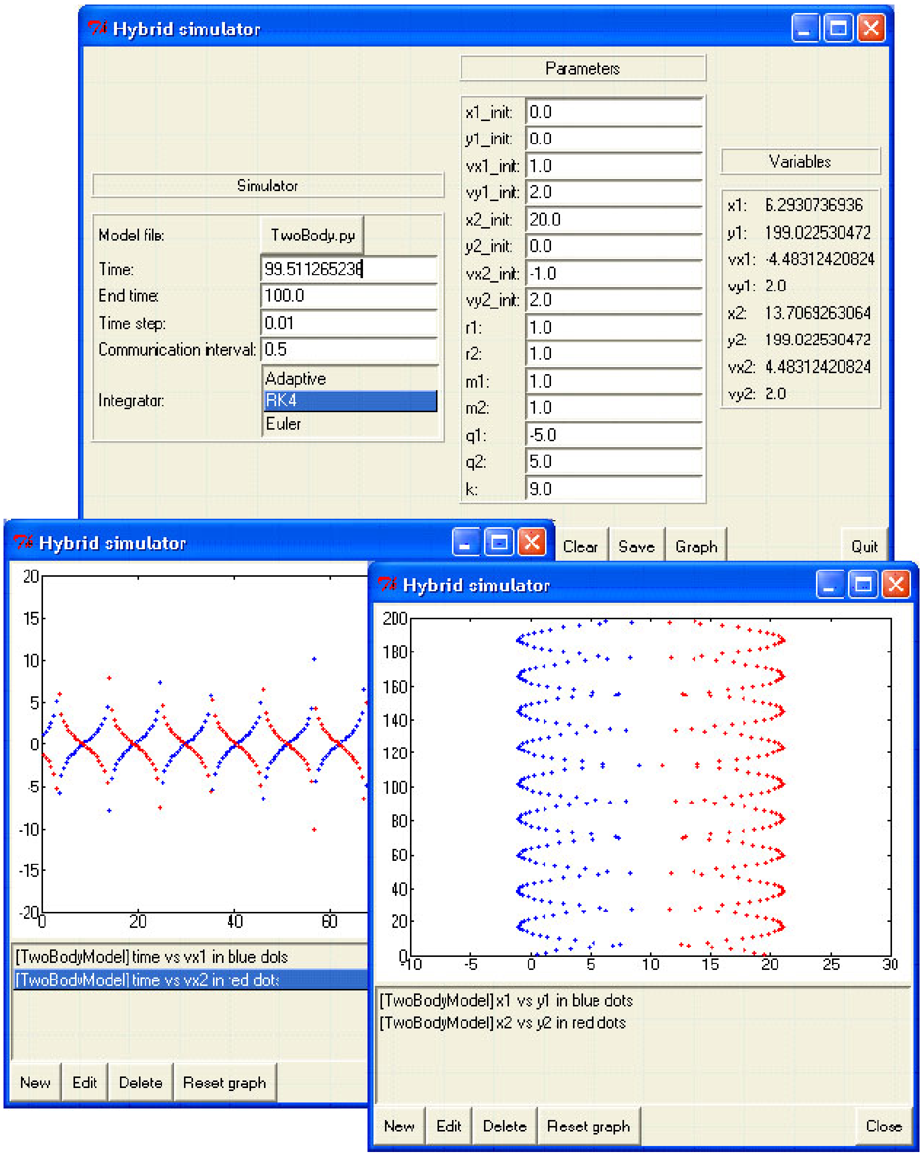

Figure 10: **Two Discs with attraction: RK4.**

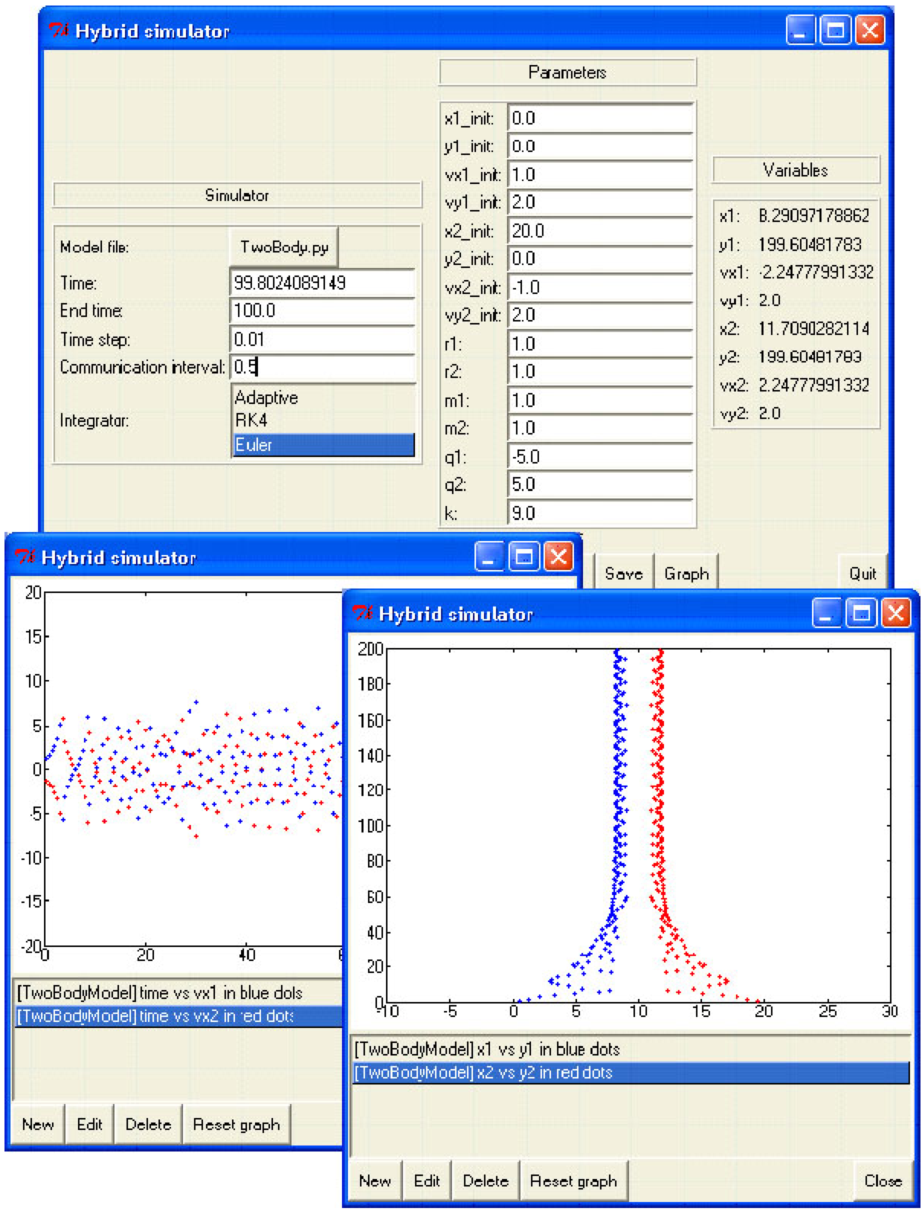

Figure 11: **Two Discs with attraction: Euler.** The difference with figure 10 for the resulting trajectory of the discs is due to the less accurate integration method of Euler.

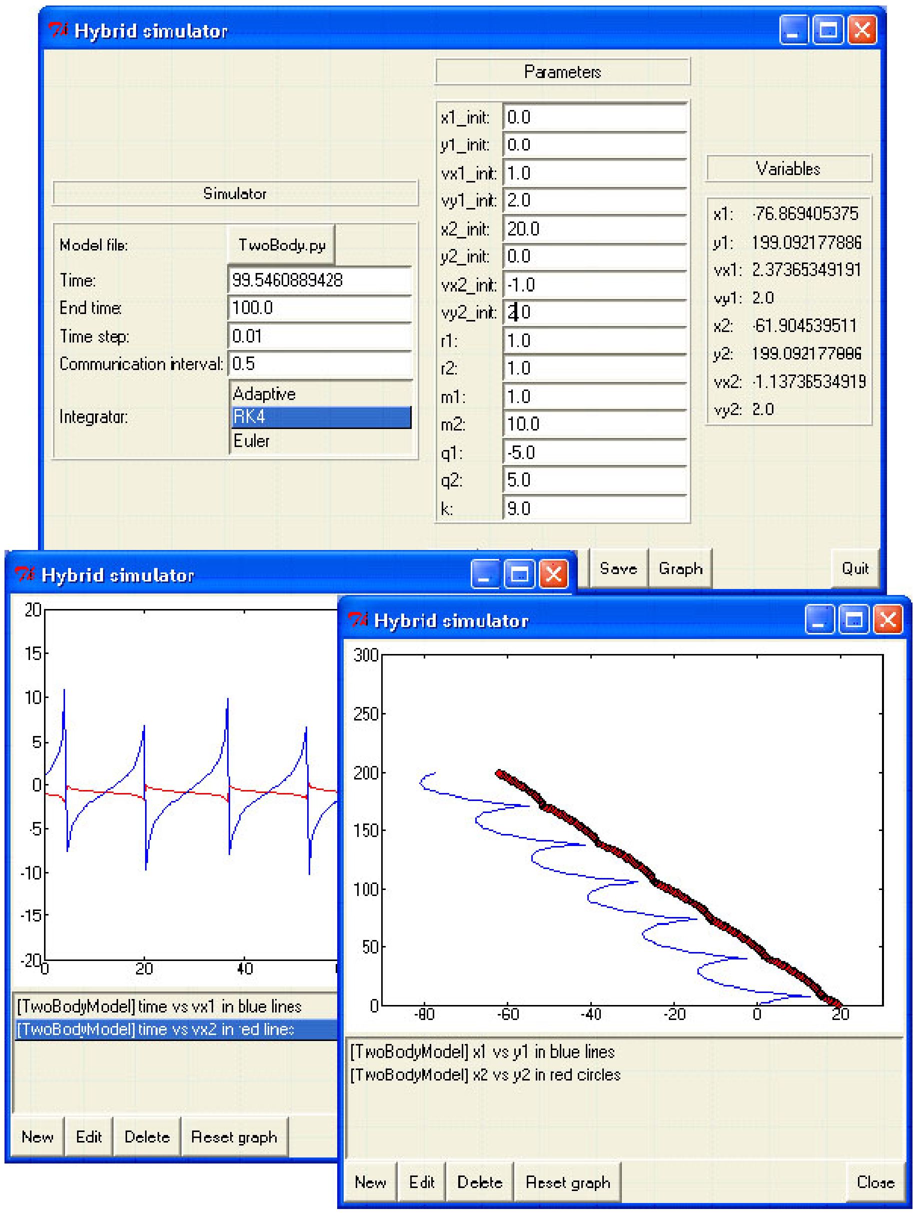

Figure 12: **Two Discs with attraction - heavy disc.** Disc 2 (red) is now 10 times heavier than disc 1 (blue).

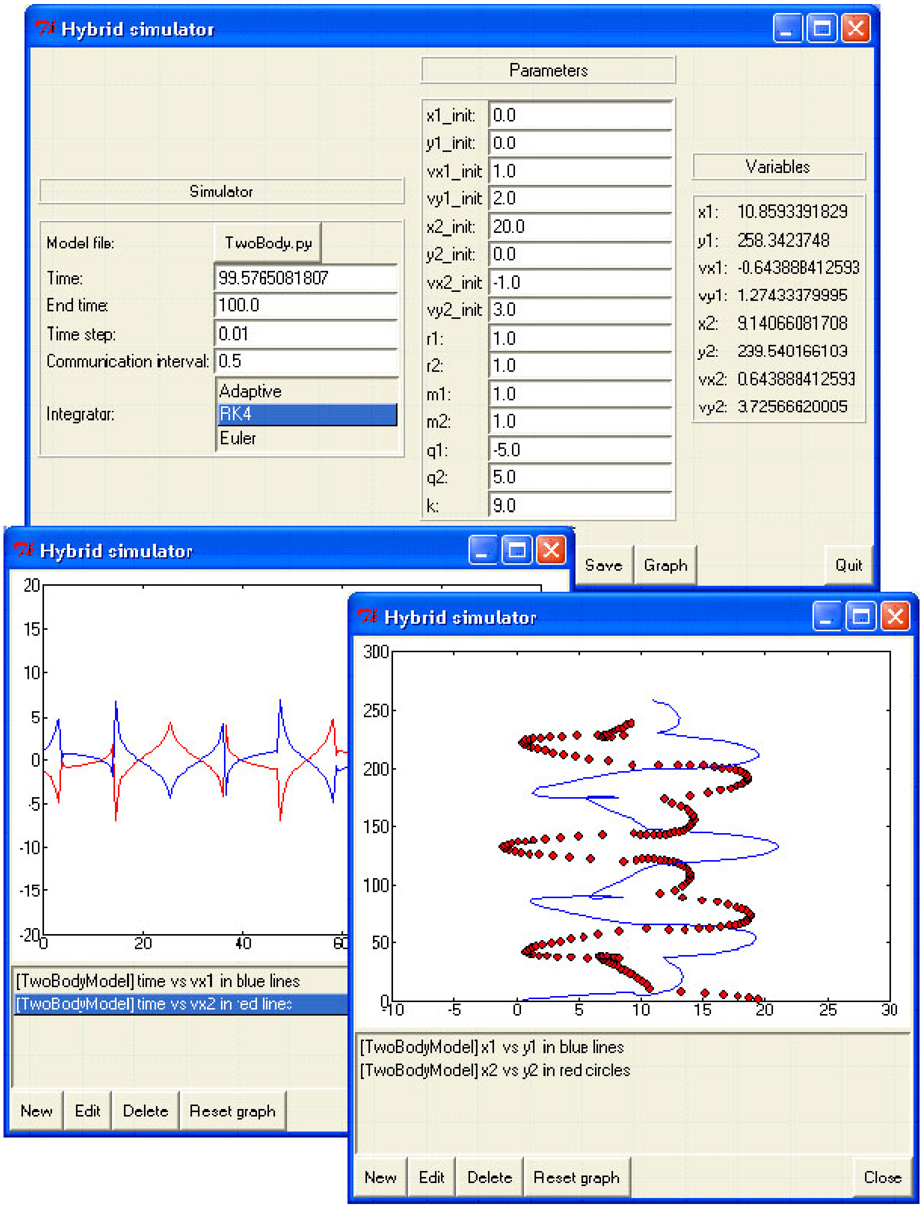

Figure 13: **Two Discs with attraction - chaotic disc.** The initial speed for disc 2 was made slightly different than the one for disc 1. The trajectory of both discs is now a lot more complicated.

6 My Summer Research

6.1 Big picture for the summer

I didn't have any specific goal for this summer; well, at least, not as specific as writing a OCL to python code translator or a UML Class diagram to ER translator...

The most general goal for me was to get a feel for multidisciplinary research and get it touch with a little of many fields about Hybrid Systems, which reunite mathematics, physics and computer science (my three majoring subjects). Since the domain of Hybrid Systems is quite huge, I couldn't do an exhaustive research on the subject (no time, too large). But maybe, by starting with an open mind, we can find something innovative. So in summary, I'll have to trust Hans' experience to orient me in this huge field!

6.2 Some issues about Hybrid Systems

More in relation to the MSDL group, one thing that should be investigated about hybrid systems is to find the best formalism to model them, and the best one to simulate them. The goals for modelling and simulation are different, so we could think to use different formalisms for each and find a way to translate between the two (multi-formalism, this is one of the main components of the MSDL philosophy, right?). The main need for modelling is *expressiveness* whereas simulation needs *efficiency*. From a "brief" meeting Hans and I had at the beginning of the summer, one quick hypothesis was to use *statecharts* for modelling and *DEVS* for simulation.

Another issue for the modelling vs. simulation part is using causal vs. non-causal modelling. For example, $V = RI$ is a causal form of Ohm's law since if we know R and I, V is known and we don't need to solve for it (R and I are assumed to be the "cause" for V). The non-causal form could be $V - RI = 0$. There, depending on which variables are known, we are free to solve for the required variable, so no causality has yet been assigned. Explicit ODE's are causal, whereas DAE's are non-causal. ODE's are a lot easier to solve and simulate, but it is a lot easier to model using DAE's directly (since we don't have to solve for the required variable: we can let the computer do it for us, like the Fortran package DASSL (Differential Algebraic System Solver Library) for example). So again, we have here the modelling vs. simulation issue.

Finally, another issue that I'll just mention here is how do we reinitialize state variables after a transition during the simulation. Because the transition of state sometimes changes the whole structure of the equations in our model, we sometimes need to change the value of some state variables in order to obtain physically allowable states which don't violate physical laws (like conservation of energy) for example. How to do this is very far from trivial, and is worth investigating.

6.3 Paths for me

What I've done for the past month is to get familiar with Hybrid Systems, understand the 522 Simulator and implement some simple physical models with it to get a feel for what I can do with it, what I can't and how it could be improved. As you've seen, this simulator is already impressive. But as a physicist user who wants to model, some improvements could be made in the modelling environment. We don't know yet (June 12) what will be the plan for the remaining of the summer (I have a meeting with Hans about this today!), but several paths are available to me.

The first one would be to study how to improve the 522 Simulator. One improvement could be to add the possibility to use DAE instead of ODE to model, and also to be able to use vectors directly instead of having to use several scalar variables to simulate those vectors. Another improvement would be to redesign the 522 simulator so that it has a modelling component and a simulation component which are more distinct (for now, they are quite intermingled).

Another path (which is quite linked to the first one) would be to add a GUI layer to the modelling environment using AToM^3. Using meta-modelling in AToM^3, I could specify the syntax of a Hybrid System model, which was roughly described as a FSA with ODE's specified for each state. The user can draw this FSA and type ODE's inside them, etc. in the modelling environment generated by AToM^3. Then, using graph grammars, I could generate python code for this model built in AToM^3 which could be reused by the 522 simulator.

A third path which is more theoretical and general would be to think about the modularizing and structuring tools for modelling and how to implement them for Hybrid Systems. Those tools come from the object oriented philosophy and some researches have already been done about them in the modelling group in which Hans was a member in Europe, and that I've forgotten the name... The objective is to be able to have maintainability, reusability and generality for our models. Examples of those tools are: using types and classes in our model, using the coupling formalism for the interactions, using inheritance, using multiple levels of abstractions and multiple formalisms. Don't ask me what those are now, I don't know yet! But this could be investigated...

References

[1] Zeigler, Bernard P. et al., *Theory of Modeling and Simulation*, Academic Press, San Diego, 2000, QA76.9 C65 Z44 2000. 1.1