The Software Architecture of the Physolator–a Physical Simulation Framework

Dirk Eisenbiegler
Faculty for Digital Media
University of Furtwangen
Furtwangen, Germany

***Abstract*—Physolator is Java based physics simulation framework. Physolator supports an object oriented style for building physical models. This article describes Physolators core architecture and it explains how the framework architecture contributes to building physical simulations in a modular, object oriented style.**

Keywords-physical simulation; Java; object-oriented framework

I. INTRODUCTION

Computer based physical simulations are an important tool for physicists. Besides that, physical simulations are also used in engineering sciences, in computer games, in animations and inside e-learning tools.

Physolator is a Java based framework for building physical simulations. Physolator is designed for beginners as well as for professionals. It only takes basic Java programming skills and little knowlege about physics to get started. The Physolator provides a Java based infrastructure that makes it easy for beginners to get their first physical simulation up and running. At www.physolator.de there is a video tutorial explaining in ten minutes, how to write your first physical simulation [2].

Physolator is also well suited for professionals. It provides powerful object oriented techniques for building complex physical simulations in a modular style where physical components, numerical procedures and graphics components are reused in different physical systems. Many of the features can be found on the website www.physolator.de. More details can be found in [1].

II. APPROACHES TOWARDS BUILDING PHYSICAL SIMULATIONS

Physical systems are nothing but computer programs. Basically, it only takes a programming language to implement a physical simulation program and run it on the computer. However, you have to be aware, that some very specific skills are required. Sure, you must be familiar with physics. Furthermore, you must have an understanding of numerical mathematics in order to implement the numerical procedures. Finally, you the results of the physical simulations have to be visually represented on the screen. This is why you must also be familiar with 3D graphics programming.

There are lots of different ways to build a physical simulation. You can build your physical simulation from scratch or you may use one of the libraries and frameworks available in the internet. Developing physical simulations should be done in componentwise and the components should be shared inside a community. However, many of the existing architectures are not designed for reuse. Furthermore, the big variety of architectures makes it very difficult to share components.

Numerical frameworks like MATLAB/Simulink or GNU Octave can help you to overcome the burden of implementing numerical procedures. They also provide you with a set of graphical tools. However, these frameworks use very specific programming languages. In order to use these frameworks you first have to learn these programming languages.

III. PHYSICAL SYSTEMS

Physolator is an object oriented framework that runs physical systems. PhysicalSystem is a class provided by the Physolator framework. Users of the Physolator implement subclasses of PhysicalSystem. After loading these subclasses to the Physolator, the Physolator can run the simulation.

```
public class TestSystem extends PhysicalSystem {

   @V(unit = "m", derivative = "v")
   public double x = 3;

   @V(unit = "m/s", derivative ="a")
   public double v = 5;

   @V(unit = "m/s^2")
   public double a;

   @Override
   public void f(double t, double h) {
      a = -9.81 - 0.1 *Math.signum(v)*Math.pow(v,2);
   }
}
```

FIGURE I. EXAMPLE FOR A PHYSICAL SYSTEM

When working with Physolator, physical variables are nothing but object attributes. Java annotations are used to describe relations between the variables. The example above represents a vertical trajectory of a point mass with an air resistance force against the direction of movement. The system consists of the physical variables x, v and a, representing the height, the speed and the acceleration of the point mass. Java annotations are used to define the derivation relationships between the variables: v is the first derivative of x and a is the first derivative of v. The annotations also define the physical units of the variables.

Inside a physical system the method f contains the formulas. In this simple example, there is only one formula. The formula describes how to compute the actual value of the acceleration a.

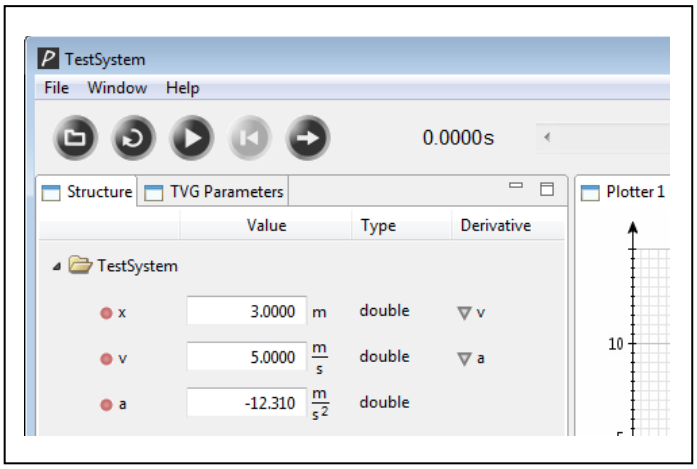

FIGURE II. PHYSICAL SYSTEM AFTER LOADING

The variables with their initial values and the formulas define the behavior of the physical system. The physical system is nothing but an ordinary differential equation. After loading the physical system to the Physolator, the Physolator will "numerically solve" the differential equation. In other words: The Physolator will run the physical system.

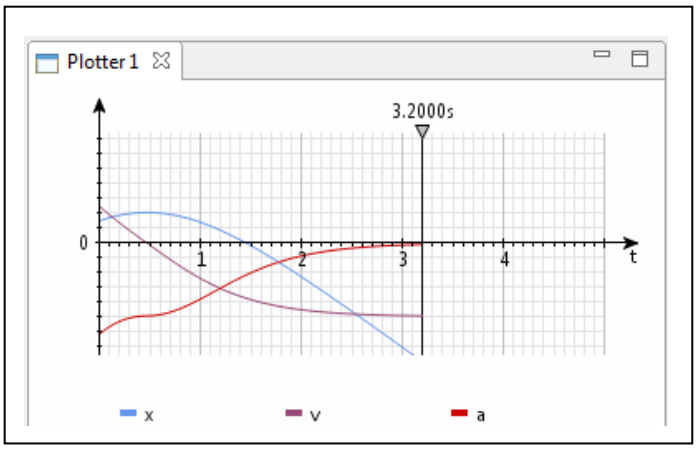

FIGURE III. PHYSICAL SYSTEM DURING RUNTIME

IV. PHYSICAL COMPONENTS

The Physolator framework supports a modular, object oriented development style. The core elements are scalar physical variables. In Java, scalar physical variables correspond to object attributes of type *double*. Step by step more and more complex components are build by combining scalar physical variables to objects and by combining physical variables and objects to more complex objects: vectors, point masses, springs, rotation objects, grids of point masses, springs, scalar fields and so on.

The following picture shows a more complex physical system with a satellite revolving around moon and earth. The physical system is build in a modular style. It consists of vectors, rotation objects and point masses. Derivation relationships are defined on the level of objects rather than on the level of scalar physical variables. Each component is not only a container for the scalar physical variables, but it also contains its mathematical and physical formulas. Vectors contain methods for vector arithmetic and point masses contain formulas for gravitation, Corriolis forces and centripetal forces.

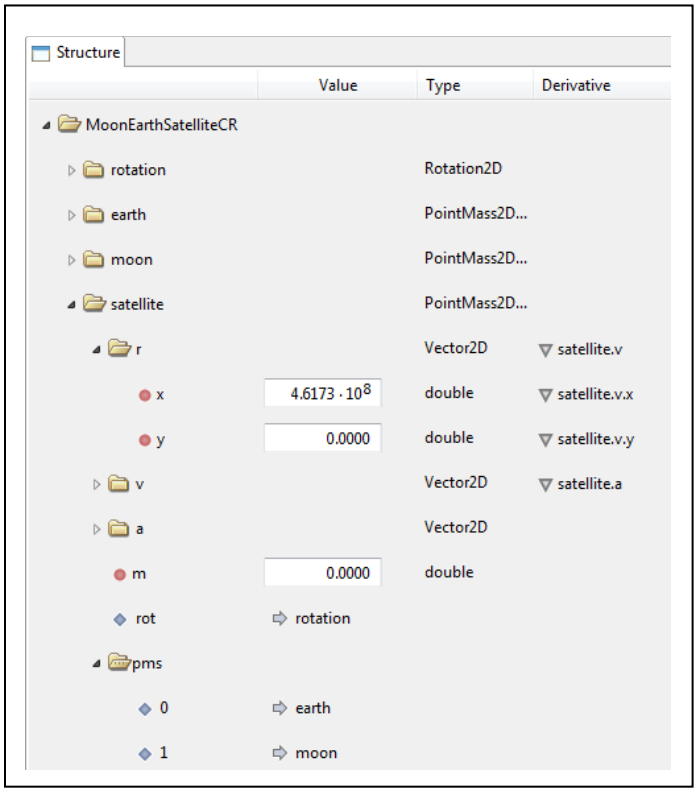

FIGURE IV. OBJECT ORIENTED MODELLING

Physical components are objects, that contain both: the current state of the object (variables) and its behavior (formulas, i.e. methods). The objects are linked with one another. Links are described via Java annotations. In the above example, a point mass contains a link to a rotation object describing the rotation of the coordinate system. Due to this link the point mass "knows" how to compute its Coriolis and centripetal force.

Physical components are designed for reusability. Once you have declared a physical component, you can build several instances in different physical systems. You can also use inheritance to build variations of the physical components and add extra behavior to the components. So far, the point mass class contains the physical formulas for gravitation, Coriolis force and centripetal force, but it does not yes describe the behavior of springs. To add this physical rule to point masses, you have to build a subclass and add appropriate formulas by overwriting method *f*.

Physical components are nothing but Java classes. Physolator supports Java 8. In order to improve reusability and performance, you can all the features that Java provides: generics, inner classes, anonymous inner classes, lambdas/closures, enumerations, multithreading [4,5].

V. PHYSOLATOR ARCHITECTURE

Most numerical frameworks work with scripting languages. The user writes his code in the scripting language and the frameworks links the user code by interpreting the scripts. The Physolator uses a different approach. Both the framework and the user code are written in Java. The user code is loading using Javas class loading techniques.

Physolator makes extensive use of annotations and reflection. Annotations and the reflection API are advanced elements of the Java programming language [4]. The user attaches configuration information to its code by adding annotations. During load time, the Physolator creates an instance of the class and analyzes the structure of the components and subcomponents. This is where the Java reflection API is used. Furthermore, the Physolator uses the reflection API to read the annotations: derivation relationships, physical units, links between objects.

Running a physical system means applying a numerical procedure. The Physolator comes with a set of built-in numerical procedures like Runge-Kutta, Fehlberg and Dormand-Prince. It supports simulations with a fixed step with as well as simulations with variable step widths. Physolator supports both single step methods as well as multi-step methods like Adams-Bashforth and it also supports predictor-corrector techniques. As already mentioned, there is a set of built-in numerical methods. However, you may also program your own numerical procedures, load them to the Physolator and use them during the physical simulation. Have a look at [3] for details.

Physolator has an integrated recorder. For every simulation time, the recorder stores the values of all variables. This data is used in different ways. There is a built in plotter for printing the function graphs of selected variables (see figure III). The recorded data may also be accessed from graphical components (see figure V).

The Physolator implementation is based on Eclipse RCP [6]. Eclipse RCP provides Physolator with the look and feel that programmers are used to when dealing with integrated development environments (IDEs) such as NetBeans, Eclipse or IntelliJ IDEA. Physolator, however, is an independent piece of software. Neither is it an IDE nor is it a plugin for an IDE. The user can choose any IDE to build physical system for the Physolator. Physolator comes with a library of classes. To build your own physical system, all you have to do is add this library to the class path.

Physolator provides an auto-start mechanism. To get a physical system up and running you would usually start the Physolator and then manually load the physical system into the Physolator framework. Instead you can equip your physical system with a main method, that invokes the start()-Method. Physical systems with such a main-method can be started from the IDE. Starting a physical system results in starting the Physolotor framework and automatically loading the physical system into the framework.

Once a physical system is loaded into the Physolator, you can continue developing your code inside the IDE. A reload-button inside the Physolator allows you to update your physical system to the current state from the IDE.

VI. GRAPHICAL COMPONENTS

Graphical components visually represent the current state of a physical system. Physolator supports both 2D and 3D graphic components. Graphical components are attached to physical systems. They are loaded whenever a physical system is loaded, that references one ore several graphical components.

In many situations 2D graphics are a good choice for a clear depiction of the physical system state. 2D graphics components can be used for both: visualization on the screen and high quality printed publications. The built-in graphical component API provides an adapter for painting the 2D drawing objects to the screen using OpenGL. But it has also an adapter for producing scalable vector graphics (SVG). Graphical components come with a built-in snapshot functionality. The snapshot function produces SVGs. SVGs are well suited for high quality printed publications.

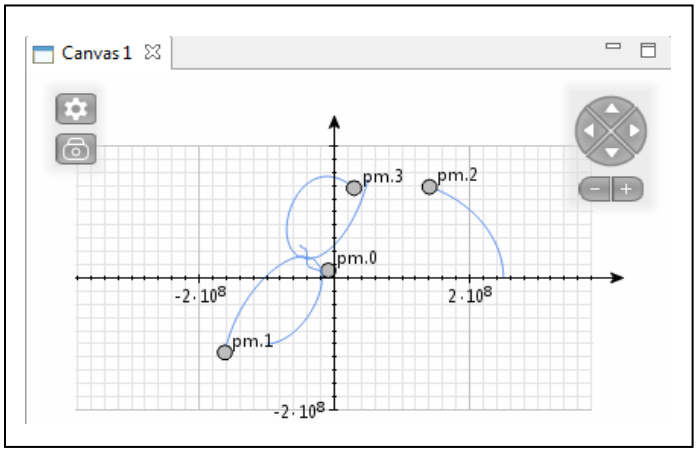

FIGURE V. GRAPHICAL COMPONENT

Physolator also supports 3D graphical components based on OpenGL. The picture of an OpenGL based 3D screen is computed by the graphics card. This is why in this case a snapshot in can only produce bitmaps and the resolution is limited by the screen resolution.

All graphical components – 2D and 3D – are interactive. As soon as your mouse is over the graphical components, interactive buttons show up (see figure V). These buttons are used for navigating inside the graphics: zooming in and out, moving to any direction. Furthermore, there is also a button for taking snapshots and another one for changing the settings. The 2D graphics have a built-in mechanism for drawing scales and grid lines. As you navigate inside a 2D graphical component, the scales and grid lines are adapted automatically.

Graphical components provide a parameter mechanism. Annotations inside the graphical components define, that the values of certain variables are parameters of the graphical components. By pressing the settings button, the user can interactively change their values.

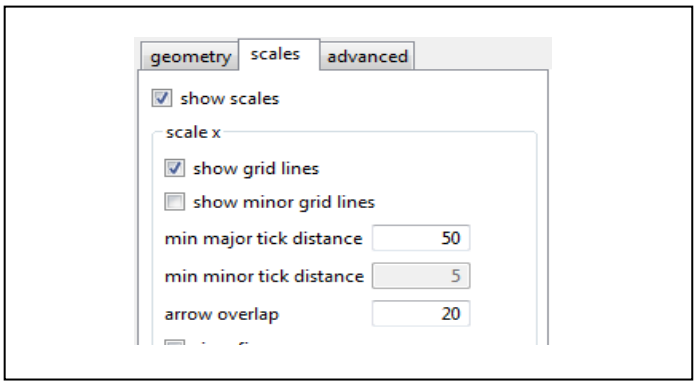

FIGURE VI. GRAPHICAL COMPONENT PARAMETERS

You can program a graphical component that is tailored to a specific physical system. Alternatively, you can implement a generic graphical component that is designed for a specific domain and that can be used by different kinds of physical system from that domain. Figure V shows a generic graphics component for two dimensional point masses. Any physical system working with two dimensional point masses can use this graphics component without any extra programming work.

Generic graphical components make use of the built-in structure API. The structure API allows the graphical component to read the structure of the physical system. In the example: How many point masses are in the physical system? What are their names? What are their positions and speeds? The above example also makes use of the recorder. Figure V not only shows the current location of the point masses, but also their paths. The paths are drawn using data from previous points in time.

VII. PHYSICAL EVENTS

Physical variables usually are continuous and differentiable functions of time. In physics, this is true for many domains. In such domains, physical systems are often described using ordinary differential equations.

However, there are also some physical effects, where the values of variables change abruptly. The change happens within one point in time. This sudden change results in a discontinuity of the physical variables. This kind of physical effect shall be called a physical event. Examples for physical events are mechanical impacts and breaks, electronic switching operations and radioactive decay events. In all this cases, the physical model defines, that the event happens in a single point in time. In such a case the formulas of the physical model describe a functional relationship between the state of the physical system right before the event and the state of the physical system right after the event.

Physolator supports physical events. An event oriented programming style is used to deal with physical events. The method g from class PhysicalSystem implements physical events. It is overwritten by the users physical system. The method is used to detect physical events and to handle them. Handling the event means mapping the state right before the event to the state right after the event. The Physolator framework iteratively detects points in time where physical events occur. For this iteration Physolator makes use of method g. Then the physical system is run until event time is reached. At that point in time the event handler is fired.

The following pictures shows some balls flying in a box. The arrows indicate velocities. The balls are once a while colliding against other balls or against the walls of the box. The picture on the left shows the state right before the two balls on the top hit against one another. The right hand side picture shows the state right after the impact. Due to the impact, the speed of the two balls changes abruptly. The relationship between the speeds before the impact event and the speed right after the impact event is described by physical formulas fired inside g.

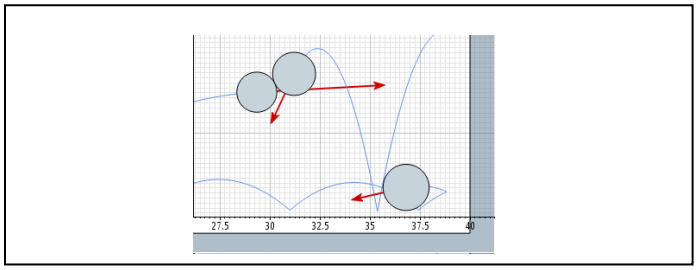

FIGURE VII. Physical Events

VIII. PERFORMANCE MONITORING

With an increasing complexity, performance matters. Especially, when it comes to real time simulations. As soon as you are faced with a performance issue, you have to analyze the CPU load in order to find the bottleneck. Depending on the physical model, the time consumption for the numerical procedures could be responsible for the performance problem, but it could also be the program code for visualization. To find the bottleneck, Physolator comes with built-in performance analysis tools.

ACKNOWLEDGMENT

This article could only give an overview of the Physolators architecture. More details and more examples can be found on the web page *www.physolator.de*. Currently, some more examples are being developed. I am also intending to provide some educational material for use in a classroom scenario. Comments are welcome. Feel free to contact me.

REFERENCES

[1] D. Eisenbiegler, "Physolator – Getting Started", Video Tutorial, http://www.physolator.de/joomla/index.php/en/manual#GettingStarted

[2] D. Eisenbiegler, "Physolator Programming", Video Tutorial, http://www.physolator.de/joomla/index.php/en/manual#PhysolatorProgramming

[3] D. Eisenbiegler, "Objektorientierte Modellierung und Simulation physikalischer Systeme mit dem Physolator", BoD, 2015, ISBN 978-3738606829

[4] B. Eckel, "Thinking in Java", Prentice Hall, 2006

[5] R. Warburton, "Java 8 Lambdas: Pragmatic Functional Programming", O'Really, 2015

[6] L. Vogel, "Eclipse IDE: Eclipse IDE based on Eclipse 4.2 and 4.3", vogella series, 2013

[7] G. M. Sellers et. al., "OpenGL Superbible", Pearson Education, 2015.