

MODELING WITH ACUMEN: A HANDS-ON TUTORIAL

Walid Taha,
Rice University, Houston, TX, USA.

2009-11-15

Copyright Walid Taha and Rice University, 2009.

CONTENTS

1	Introduction	4
2	Disclaimer, Known Bugs, and Feedback	4
3	Getting Started	5
3.1	Installation	5
4	Continuous Behaviors	6
4.1	Physical Quantities and Differential Equations	6
4.2	An Acumen Model for the Mass/Spring Example	6
4.3	Virtual Experiments with Simple Mechanical Systems	7
4.4	Principles and Methods for Systematic Modeling	8
4.5	Pendulum Using Newton's Equations	8
4.6	Pendulum and Pendulum/Spring Using Euler-Lagrange Equation	9
4.7	Pendulum Using Hamilton's Equations	10
5	Discrete Events	10
5.1	Digital and Analog Computers	10
5.2	Quantization and Discretization	10
5.3	Difference Equations	11
5.4	Event-driven Equations	11
5.5	An Acumen Model for an Alternator	11
5.6	Virtual Experiments with Simple Discrete Components	12
6	Hybrid Models	13
6.1	Open-Loop Pendulum Behavior	13
6.2	Closed-Loop Control	14
7	Acumen in Server Mode	15
7.1	OCaml Client Example	15
7.2	Java Client Examples	16
8	Tutorial Activities	16
A	Error Messages	18
B	Copy of Acumen's INSTALL file	18
C	Copy of Acumen's README file	20
D	Credits	20

1 INTRODUCTION

Increasingly, software and hardware are interacting directly with their physical environments. As a result, it is becoming more natural to specify the intended behavior from these components in terms of their actions on their physical environments. In addition, it is also the case that novel physical devices, such as robotic systems, are being designed simultaneously with their digital computing and control components. Such systems, which are sometimes described as being *cyberphysical systems*, require that we model both physical (often continuous) as well as digital (discrete) components in an integrated manner. Enabling this type of integration and facilitating the modeling of the continuous parts are among the key design goals of Acumen.

This document provides the notes for the tutorial part of a short course on modeling in Acumen. It starts with introducing Acumen's continuous modeling language, then moves to the discrete modeling language, and then to the integration of the two. The last section of the tutorial introduces Acumen's server-mode functionality, which is designed to make it easy to interact with Acumen from external application codes.

The short course took place at Halmstad University on 2009/11/17.

These course notes are intended as an introduction to both the Acumen language and the process of modeling physical systems. The course was written for Acumen version 2009-11-15.

2 DISCLAIMER, KNOWN BUGS, AND FEEDBACK

The current release of Acumen is a research prototype, and the implementation is still undergoing significant changes. Such an early release is made available only to facilitate the dissemination of the ideas underlying Acumen, and not to provide a working product. For example, know that the system:

- Can fail by a segmentation fault for a variety of reasons, the main one being that the simulation engine is implemented using several naive algorithms that use resources that are quite limited in the underlying implementation (such as the call stack).
- Currently requires that the name used for any event is always "clock". This is another artifact of using an old and simplistic solver engine.
- Has default GNUplot behavior that does not necessarily automatically plot all variables; also, sometimes it is not even possible to explicitly list all variables.

No doubt, as you work through the notes and try out different examples, you will encounter bugs. The next section will include instructions for the best way to collect your feedback as you use the system. When you do that, this will help us to make the prototype more usable for you and for others. Thank you in advance for helping us in this way.

3 GETTING STARTED

3.1 Installation

The current release is available online, and it should be easy to install on Unix-like operating systems, including Linux, Mac OS X, and Cygwin (a Unix environment for Windows).

1. Download and untar the `.tar.gz` file containing the source distribution from `http://www.cs.rice.edu/~taha/Acumen/Acumen.tar.gz`.
2. As soon as you have untarred the Acumen distribution, enter the Acumen directory and create a subdirectory `firstname-lastname`. At the end of the course, you will tar this directory and send it back to the instructor.
3. Inside the directory you just created, we want you to make a file with a specific name for jotting down your observations about the system. For this purpose, create and use a file called `journal.txt`. Please be sure to include your name and e-mail address in the file, which will be the place to make note of any bugs, strange behaviors, or things that you particularly liked about Acumen. Please also include any comments that you may have encountered during installation.
4. Follow the installation instructions in the `INSTALL` file. A copy of that file made at the time of creating this document is available as an appendix to this document.

You should now be ready to use Acumen. As a “sanity check,” execute the following command at the command line:

```
./acumen examples/misc/aardvark.acumen
```

The system should respond at the console with:

```
gnuplot script is examples/misc/aardvark.gnuplot
gnuplot data is examples/misc/aardvark.dat
Output generated is examples/misc/aardvark.pdf
```

In addition, a three-page PDF document should automatically open on your screen. If that does not happen, please notify the instructor.

4 CONTINUOUS BEHAVIORS

This section reviews basic concepts that arise in modeling physical phenomena using differential equations, and it introduces the features of Acumen that support this type of modeling.

4.1 Physical Quantities and Differential Equations

To model physical processes mathematically, it is useful to write equations on real-value, time-varying values. This means that we work primarily with values of type $\mathbb{R} \rightarrow \mathbb{R}$, where \mathbb{R} is the set of real numbers and \rightarrow is the constructor for the space of continuous functions from its domain to its range¹. For notational convenience, it is also common to use a point-free notation, where instead of writing $f(t) = m * a(t)$ we simply write $f = m * a$. We also need to express the fact that certain values model the rates of change of other values. For example, $a = dv/dt$, or, in the point-free form, $a = v'$.

Example 4.1 (Mass/Spring) *Consider the motion of a mechanical system that consists of a mass $m = 1$ that can move only vertically, that is attached to a spring from below, and that is subject to gravity. Assume that the spring has a natural length $l = 10$ and a coefficient of compression $k = 10$. The dynamics of such a system is described by the following equations:*

$$f = ma, \quad a = x'', \quad f = k(l - x) - mg.$$

4.2 An Acumen Model for the Mass/Spring Example

Acumen allows us to express the equations above directly. To simulate the behavior of the system, we need to also specify the initial state of the system, typically called *boundary conditions* in the context of differential equations. For this example, we can assume that the initial position is $x(0) = 1$, and the initial speed is $x'(0) = 0$. Finally, we will assume that we are interested in the behavior of the system from time $t = 0$ to time $t = 5$, with a simulation step size of $\Delta t = 0.001$. In Acumen, this problem would be specified as:

```
continuous
  f = m * a; a = x''; f = k * (l-x) - m * g;
  l = 10; m = 1; g = 9.81; k = 10;
boundary conditions
  x with x(0) = 1, x'(0) = 0;
```

¹An important technical question is how the set of reals is represented, as this has implications for what operations can be defined precisely on this set. Currently, the Acumen prototype uses floating points. However, we expect that this will change in the future.

```
simulation
    time_start = 0; time_end = 5; step_size = 0.001;
```

Acumen will compile this model into simulation code, run the simulation code, generate a PDF summarizing the process, and automatically open a the file on your screen. Please read further before trying to run this program!

4.3 Virtual Experiments with Simple Mechanical Systems

This section describes a series of problems that are essentially experiments that you can conduct using Acumen.

Problem 4.1 (Mass/Spring) *For the example presented in the previous section, write down what you expect the behavior of x , x' , and x'' to be. Once you have done that, run the command above and note any differences between what you expected and what the simulation produced.*

The easiest way to execute this simulation code is as follows. Assuming that the above code is stored in a file called `mass-spring.acumen`, the easiest way to execute the simulation is to type the following at the command line while in the Acumen directory:

```
./acumen examples/physics/mass-spring.acumen
```

Problem 4.2 (Free Piston and Air Chamber) *The example above can be easily changed to model a piston attached to an air-tight chamber and free to move vertically. In particular, this would require replacing the term $(l-x)$ by another term that reflects how an air-tight chamber reacts to compression. For compressible gases, the key principle is that the result of multiplying the pressure by the volume produces the constant. For a piston, the volume of the underlying chamber is proportional to the height of the piston x .*

Before changing your Acumen program, write down how you expect the behavior of x , x' , and x'' to be different in this new setting in which the spring is replaced by an air chamber. Modify your Acumen model and simulate it. You may need to adjust the simulation time to see periodic behavior. Note how the results you get compare with what you expected.

Problem 4.3 (Mass/Spring/Damper) *A damper is a device that creates a force against movement, and that force is proportional to the speed. Using a coefficient of half, modify the equation $f = k * (l - x) - m * g$ to model a damper being attached to the mass. Think carefully about the direction of this force. Write down how you expect the behavior to change. Then, run your simulation. Finally, write down how the result compared with what you expected.*

Problem 4.4 (Mass/Charge) *Opposite electric charges repel with a force inversely proportional to the square of the distance between the two objects.*

Change the term $(l - x)$ in the mass /spring example one more time to model this kind of behavior. As usual, document your expectations about the behavior of the new system, run the simulation, and compare the results with what you expected.

4.4 Principles and Methods for Systematic Modeling

Historically, math, science, and engineering have always been closely coupled. Math seems to focus on abstracting principles away from the details of particular application domains, with the hope that these principles can be understood as clearly as possible and be made as widely applicable as possible. Science seems to focus on understanding natural phenomena and the laws that govern them. Often, these laws are stated mathematically. Engineering seems to focus on building complex structures using what science and engineering tradition tells us. To build structures, engineering also involves analyzing structures and understanding their behavior. To do that, effective engineering has always depended heavily on systematic methods for analysis. Often, such methods are so systematic that they are highly mechanizable. Good modeling languages should provide support for such methods, and Acumen strives to be a good modeling language.

To give you a taste of what such engineering methods are like, we will focus on modeling a class of mechanical systems, called multi-body systems. We will illustrate how one simple system can be modeled using a basic method, namely, writing Newton's equations, then two more systematic methods, namely, using the Euler-Lagrange equation and Hamilton's equations. These examples motivate why Acumen supports vectors, matrices, partial derivatives, and families of equations.

4.5 Pendulum Using Newton's Equations

Consider a pendulum carrying a mass m suspended with a rod of length l and subject to a horizontal force F . Let us call the angle of the pendulum from the vertical line θ . The following Acumen code captures this situation:

```
I = m * l^2;  
F*l*cos(theta) - m*g*l*sin(theta) = I * theta'';
```

The first defines the moment of inertia around the point at which the pendulum is fixed, and the second equation is essentially Newton's second law of motion ($f = ma$) in rotational form. Such a formulation of the dynamics of the system can be written by inspecting the system that we are analyzing. This is the style in which we solve problems in a basic physics class, and possibly early on in an engineering class. However, in virtually all engineering disciplines, more systematic methods are found that enable a more systematic method of collecting the needed information in the form of equations.

4.6 Pendulum and Pendulum/Spring Using Euler-Lagrange Equation

The Euler-Lagrange equation is a generic template that can be instantiated to a mechanical system consisting of multiple different mechanical equations. Formally, the method consists of defining on vector q as a sequence of all state variables (or “degrees of freedom”), inspecting the system to write down the equation for the total kinetic energy T and total potential energy V , and then describing the behavior of the whole system by the Euler-Lagrange equation, which is defined as follows:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = 0$$

First, we note that this “equation” is not really one equation but rather a family equations, one for each value of i , which will range over the indices of values in q . Second, note the use of partial derivatives. These partial derivatives do not mean that we have a *partial differential equation*, because in fact some fairly simple computations can be used to eliminate these partials for the type of mechanical systems in which we are interested.

In the case of a pendulum, there is only one value in q because we have only one component (with one degree of freedom), but we can already see that the equations look different, and thus we can use a different “language” than what we may be familiar with from physics. In Acumen, the Euler-Lagrange formulation of the pendulum can be expressed as follows:

```
I = m * l^2; (* Stays the same *)
q = (theta,); (* Vector of state variables *)
T = 1/2 * I * theta'^2; (* Kinetic E. *)
V = 1/2 * m * g * a * (1-cos(theta)); (* Potential E. *)
L = T - V; (* Lagrangian *)
for each i in length (q) begin
  (L' (q'_[i]))' - L' (q_[i]) = 0; (* Euler-Lagrange Eq. *)
end;
```

Now, if we modify the system so that the pendulum is no longer attached to a wall, but rather attached to another mass that can itself move horizontally and that is attached to a vertical wall by a spring, this more complex system can be described with only a small change to the model, and, in particular, we only need to change q , T , and V into:

```
q = (x, theta);
T = 1/2 * (M + m) * x'^2
    + m * a * x' * theta' * cos(theta)
    + 2/3 * m * a^2 * theta'^2;
V = 1/2 * k * x^2 + m * g * a * (1 - cos(theta));
```

Problem 4.5 (Derivation by Hand) *Derive, by yourself, the terms of potential energy and kinetic energy, and compare them with the ones given*

above. Make note of any discrepancies in what you derived initially and what is presented above.

Hamilton's method is very helpful for modeling many mechanical systems. However, it is not the ideal method for all systems, and other methods exist for making it easier to model different classes of physical systems and to capture more clearly different features of these systems.

4.7 Pendulum Using Hamilton's Equations

Another classical method for modeling mechanical systems is to write Hamilton's equation. As with the Euler-Lagrange equation, this method also requires the modeling language to support vectors, partials, and equation families. Here is what Hamilton's equation looks like for the pendulum example:

```
H = p^2 / (2 * m * l) - m * g * l * cos (theta);  
theta' = H' (p);  
p' = -H' (theta);
```

In the general form, p is replaced by p_i , and θ is replaced by q_i .

5 DISCRETE EVENTS

In this section, we motivate the need for modeling discrete events and introduce Acumen's language for modeling event-driven systems.

5.1 Digital and Analog Computers

While physical phenomena are naturally modeled by continuous values, digital computers implement discrete models of computation. It is interesting to note in this context that before the digital computing revolution started, analog computers did exist. Analog computing was performed by putting together circuits that computed certain analog functions, so that their output signal was the result of the computation. Because analog circuit elements are sensitive to a host of environmental factors (temperature, radio waves, etc.), their results were susceptible to a wide range of errors. Digital computers seemed, and still do, to have an advantage in terms of predictably reproducing the results of their computations.

5.2 Quantization and Discretization

Compared with the physical view of the world, discrete models typically have two key features: *quantization* and *discretization*. Quantization is

the finite representation of all values, including ones that represent real-valued quantities. Discretization is the transition from a continuous notion of time to a discrete one. These two changes can be viewed, respectively, as replacing the \mathbb{R} by \mathbb{N} in the domain and range of the type of functions (representing time-varying values) with which we are working.

5.3 Difference Equations

Acumen provides special support for describing discretized computations. The design of Acumen's language reacting to discrete events is motivated by the notion of *difference equations*. In contrast to differential equations, the values we work with have type $\mathbb{N} \rightarrow \mathbb{R}$. For values of this type, we can write equations such as $x_{n+1} = -x_n$. This equation specifies that whatever the value of x_n is, the value of x_{n+1} will have the same magnitude but with the opposite sign. The equation can be viewed as implicitly defining the difference between the value at time n and at time $n + 1$. In this case, that difference would always be either $+2x_n$ or $-2x_n$.

5.4 Event-driven Equations

Acumen's language for describing event-driven behavior, which is called E-FRP, uses essentially the same idea with one variation. Instead of using a single notion of a time-tick, any of a number of *events* can trigger movement to the next-time step, and the reaction (or difference) can depend on which event occurred. For example, the equation above can be expressed as:

```
x = init 1 {clock => -x later};
```

The code above specifies that the value of x at time 0 is 1 and that whenever the event `clock` occurs, the next value for x will have the same magnitude by the alternate sign.

5.5 An Acumen Model for an Alternator

The code to implement an alternator based on the above difference equation is expressed in Acumen as follows:

```
discrete efrp
  writes x;
  observes event clock rate t = .1;
  begin
    x = init 1 {clock => -x later};
  end
```

By itself, this is a complete E-FRP program that defines the behavior that we want for x providing a definition for the periodic event `clock`. In the definition above, we specify that we want this event to occur automatically each 0.1 time units.

The `later` annotation in the above code is crucial. In particular, E-FRP components are allowed to refer to current values of other variables. The `later` annotation at the end of an expression means that the `x` will only be updated with the new value at the end of the process of responding to the current event handler.

E-FRP components such as the above can be used to generate code that is guaranteed to use bounded space and to respond to any event in finite time.

In the current implementation of Acumen, if we want to simulate the behavior of an E-FRP component, the surrounding continuous environment needs to be declared, and the output of an E-FRP component needs to be used in the definition of this environment.

Example 5.1 (Wrapper for Observing Discrete Components) *To simulate a discrete component such as the above, extend the code as follows to provide a continuous context and some simulation parameters:*

```
continuous
  y' = x;
  boundary conditions
  y with y(0) = 0;

(* ... Insert "discrete efrp" code here ... *)

simulation
  time_start = 0; time_end = 1; step_size = .001;
```

And we would have a program that Acumen can simulate.

Problem 5.1 (Alternator) *Document the behavior you expect to see in x , y , and y' values, then run the above code using the command:*

```
./acumen examples/discrete/alternator.acumen
```

Document how the output compares with your expectations.

5.6 Virtual Experiments with Simple Discrete Components

Problem 5.2 (Counter) *Modify the model for the alternator to create a counter that starts at -2 and counts up to 2. Once it has reached 2, in the next time step it is reset to -2.*

Problem 5.3 (Zigzag) *Modify the model for the counter to have the clock occur each 0.01 time units. Make the initial value for x be -20, and let the maximum value be 20. Add another variable d to keep track of direction, and use it to create a counter that counts up to 20 and then down to -20, and repeats.*

It is easy to see that, while the E-FRP language is simple, it is sufficiently expressive to express a range of important types of behaviors that a discrete component can affect. The primary motivation for the simplicity of the language is to easily ensure that controllers expressed in it are easily synthesizable into code that will use bounded space and terminate. It is also expressive enough to express important kinds of bad behaviors that a discrete control can affect on a physical context. With these things in mind, it is also useful to note that any other programming language for modeling the discrete components can, in principle, be incorporated in Acumen’s modeling framework.

6 HYBRID MODELS

Hybrid models combine both continuous and discrete behaviors. In a sense, we have already seen this in the previous section in the way that y' , a continuous time value, can be equated with x , a value that can change abruptly at discrete time steps. What we have not seen are examples in which the discrete component not only affects the physical context but can also only observe some of its values. In control theory, if we consider the physical system to be the “plant” and the discrete component to be the “controller,” closing the loop allows us to achieve use feedback in controlling the physical system.

6.1 Open-Loop Pendulum Behavior

We discussed above the model of a pendulum. A controller that performs no action to affect this pendulum has the form:

```
discrete efrp
  writes    F;
  observes  event clock rate t = 0.1;
begin
  F = 0;
end
```

Simulating this controller allows us to observe the natural behavior of this pendulum when there is no controller acting on it.

Problem 6.1 *A model for the pendulum, including initial conditions, as well as the controller, can be found in:*

`examples/hybrid/pendulum-direct-discrete-controller-0.acumen`

Run this simulation and note the first peak angle. Note any unexpected features of this simulation, and suggest a possible explanation. (Hint: What laws of physics do you expect the behavior of this system to enjoy?)

6.2 Closed-Loop Control

Now, we are ready to explore the design of controllers for this system. While there is a vast space of possible galls for controlling a mechanical system, as well as possible approaches to doing that effectively, to understand cyber-physical systems, and especially the *process of innovative cyberphysical systems*, it is very helpful to experiment and “tinker.”

As an example goal, a basic goal that we often desire from a controller is to stabilize a system, which generally means reducing its energy. Intuitively, a good way of reducing energy is to have the controller activate a force against the direction of any motion². For a controller to determine the direction of motion, it has to observe the angle of the pendulum and keep a copy of the angle it observes at the current time for making decisions in the next time step as well. The following is a simple controller that uses these ideas:

```
discrete efrp
  reads      theta;
  writes     F;
  observes   event clock rate t = 0.1;
  begin
    F = init 0
      { clock => if theta > last then -1.0 else 1.0 };
    last = init 0 { clock => theta later};
  end
```

In this example, `last` is used to store the value of `theta` in the previous time step.

Problem 6.2 *Postulate the behavior that you expect to see given this controller’s action on the pendulum, and note it down. You can see it simulate this situation by running the model in:*

```
examples/hybrid/pendulum-direct-discrete-controller-1.acumen
```

Note the value for the new first peak, and compare it with what you observed in the previous problem. Compare the behavior of `theta` with what you expected. What issues are revealed by studying the trajectories of the other parameters that are graphed in the output?

Problem 6.3 *Suggest different approaches to reduce the involvement of the controller in the system. After you have finished doing so, explain the idea behind adding the condition `... < pi/6000` in the file:*

```
examples/hybrid/pendulum-direct-discrete-controller-2.acumen
```

²Of course, control theorists also know that we can prove that this intuition is a valid principle.

Problem 6.4 *Suggest different wave forms that can be added to model different types of “noise” or “external disturbance” to the pendulum that can affect our pendulum, and that we would like our controller to nevertheless be able to deal with. For example, what could be a reasonable model for a wind blowing the mass from one side to the other, or a human changing its position. Naturally, if there is enough disturbance, any controller would be out of luck. Given that that’s the case, what are reasonable criteria to restrict the types of disturbance that our controller should be able to deal with?*

7 ACUMEN IN SERVER MODE

To facilitate the use of Acumen in a language-independent way, Acumen provides support for use in server mode, in which a client starts Acumen (initializing it with a particular Acumen model source file) and then communicates with Acumen using TCP/IP sockets. When the Acumen program starts, it sends back its complete simulation state in XML format and waits for a revised format to be sent back so that it can perform the next simulation step. This enables the client both to observe all state variables and to modify them.

The distribution contains several examples of simple clients that can be used as templates for building larger clients, and for integrating Acumen into other tools. They can be found in the `client-examples` directory. The subdirectories are named after the language in which the clients are written. Currently, examples are provided for `ocaml` and `java` clients. For all of these examples, the client code first needs to be built from the command line by typing:

```
make
```

A `Makefile` is provided to illustrate what needs to be done for each example. Once the example client is built, it can be started from the command line by typing:

```
./run
```

Again, for each example, we provide a script called `run` that spells out the steps needed to start both the server (which is Acumen, running in server mode), and the client. The script also coordinates finding a port that is available for the client and the server to use to communicate. By default, after a client session ends, the server closes. In addition, it produces the same type of reports that you have seen in previous exercises with Acumen.

7.1 OCaml Client Example

The OCaml client example `ocaml/echoclient` simply prints to the console the full state of the simulation as it is sent from the server during each

simulation step, and before it is sent back (unmodified) by this client.

7.2 Java Client Examples

The Java client example `java/EchoClient` is similar to the OCaml example described above.

The Java client example `java/MouseClient` opens a window and records the x and y positions on the window for the duration of a pre-set simulation time. This example provides a brief illustration of how external input can be incorporated into simulations in the current implementation.

8 TUTORIAL ACTIVITIES

The following are suggested exercises. If you would like them to be reviewed by the instructor, please collect all materials in one directory (including your `journal.txt` file) and e-mail it to the instructor as one archival file.

1. **Tutorial Problems.** Limiting yourself to three hours, solve as many of the problems included in the tutorial as you can. Include both your `.acumen` files and generated PDF files in what you hand in. For virtual experiment problems, include both your before- documentation (hypothesis) and after-documentation (conclusions) as comments in your Acumen file. If you run into any difficulties with any of the problems, please document these in your `journal.txt` file.
2. **Modeling Project.** Spend up to one hour modeling a physical system of your choice in Acumen. Your system can be a simple mechanical device, such as a simple robot, a particle moving in a field, a multi-body problem, a heat-transfer problem, or any such example of your choice. Feel free to use any of the examples discussed in this tutorial as a starting point. Define a target behavior (output) that you would like the physical system to perform. For example, if it was an analog electrical circuit, your desired behavior can be following a sine curve of a frequency and magnitude of your choosing. Develop a discrete controller that modifies one of the inputs to attain that goal.

If you don't finish your design within one hour, submit what you have and explain the difficulties you encountered.
3. **Summary.** Write a 1,000-word summary of what you learned in this course. Identify what you see as the key strengths and weaknesses of Acumen, and compare Acumen to similar tools that you may be familiar with. Also, compare the ideas presented in the course to the research literature that you are familiar with, and explain what you see as the strongest potential application for Acumen. Outline what

you view as the key directions for improving both the underlying theory and the engineering work that is needed to make Acumen a better tool.

4. **Optional Client Project.** Limiting yourself to one hour, build a client application for visualizing and interacting with your physical system and its controller.

Please feel free to ask the instructor if you would like any clarification about any aspect of these exercises.

A ERROR MESSAGES

This section lists error messages that may be encountered when Acumen is invoked from the command line using the `acumen` script.

Usage: acumen filename. This error is produced when no filename is provided. Make sure that you provide a filename right after the command `acumen` on the command line.

You should compile Acumen first. You attempted to invoke Acumen before the sources were compiled. To compile the sources, type `make` at the command line. Make sure that you do that from within the directory from which the Acumen distribution was expanded. This directory is usually called `Acumen`.

Fatal error: exception Sys.error("...: No such file or directory"). The file name provided to the `acumen` command could not be found. The name may have been spelled incorrectly, the file may be missing, or it may not have appropriate permissions for Acumen to read it.

Error in checking (append): No lower bound available for StaticTpVar. Currently, this error message may arise for a variety of reasons, including the fact that the file you are passing to Acumen may be empty. The error message currently arises primarily due to bugs in the implementation.

B COPY OF ACUMEN'S INSTALL FILE

PREREQUISITES

Acumen requires:

- Cygwin (for windows only)
- Ocaml
- XML-light (this is an OCaml XML library used for server mode functionality)
- LaTeX
- GNUplot

Once you have downloaded and installed each of these, you can build and install Acumen by following the instructions below.

MAC OS:

- The latest ocaml version (3.11.1) can be obtained from <http://caml.inria.fr/>
- The latest MacTeX (latex) version can be obtained from <http://www.tug.org/mactex/2009/>

- The latest gnuplot version (4.2.6) can be obtained
from <http://www.gnuplot.info/>

- The latest xml-light version (2.2) can be obtained
from <http://tech.motion-twin.com/xmllight.html>

MAC OS SNOW LEOPARD:

- Before installing gnuplot, you need to configure gnuplot as follows:

```
./configure --with-readline=bsd \  
--x-include=/usr/include/X11 --x-libraries=/usr/X11/lib
```

WINDOWS:

- Cygwin can be downloaded from <http://www.cygwin.com/>

- Ocaml, latex (tetex), and gnuplot (4.2.4) packages can be installed
while installing cygwin

- You might need to install ocaml from sources
from <http://caml.inria.fr>
instead of installing it as a cygwin package

- The latest xml-light version (2.2) can be obtained
from <http://tech.motion-twin.com/xmllight.html>

- Before installing xml-light, you need to edit its Makefile
to replace the following section:

```
.mly.ml:  
    ocamlyacc $\  
with:  
%.mli %.ml: %.mly  
    ocamlyacc $<
```

INSTALLATION

From the top directory, do:

```
make
```

To start Acumen do:

```
./acumen <acumen-file>
```

To start Acumen in server mode do:

```
./acumen -p <port-number> <acumen-file>
```

and then you should start one of the Acumen clients under
Acumen/client-examples For more information go to one of the client
folders and follow the readme instructions there

[OPTIONAL] EMACS MODE INSTALLATION INSTRUCTIONS

Assuming you have installed Acumen in /Acumen then to install emacs
mode for Acumen, edit your .emacs file to add the following lines:

```
;;Mode for Acumen  
(setq load-path (cons "/Acumen/emacs" load-path))  
(setq auto-mode-alist (cons ('("\\.acumen" . acumen-mode)  
                             auto-mode-alist))  
(autoload 'acumen-mode "acumen-mode"  
          "Major mode for editing Acumen code" t)
```

C COPY OF ACUMEN'S README FILE

\$Id: README 9848 2007-12-29 05:09:42Z ji2 \$

OVERVIEW:

Acumen is an environment for modeling and simulating cyberphysical systems.

CONTENTS:

INSTALL	instructions for installation
LICENSE	license file
Makefile	main Makefile
README	this file
tests	tests files
examples/	example files
client-examples/	example ocaml and Java clients
src/	source files
src/lib	library source code
src/parser	parser files

INSTALLATION: See the file INSTALL

AVAILABILITY:

The Acumen distribution can be accessed at

<http://http://www.cs.rice.edu/~taha/Acumen/Acumen.tar.gz>

BUG REPORTS AND USER FEEDBACK:

Send your bug reports by E-mail to:

Walid Taha <maroneal@gmail.com>

D CREDITS

Several people have contributed to the implementation of Acumen:

Design:

Walid Taha

Development:

Adam Wulf
Alexandre Chapoutot
Angela Zhu
Cherif Salama
Edwin Westbrook
Jun Inoue
Marisa Peralta
Roumen Kaiabachev
Stefan Monnier
Travis Martin
Walid Taha