The Consumer Coffee Maker

Purchasing and Design Heuristics through Modeling and Simulation

Neal DeBuhr and Andre Baptiste

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ, USA, 85281

January 11, 2018

Contents

1 Introduction

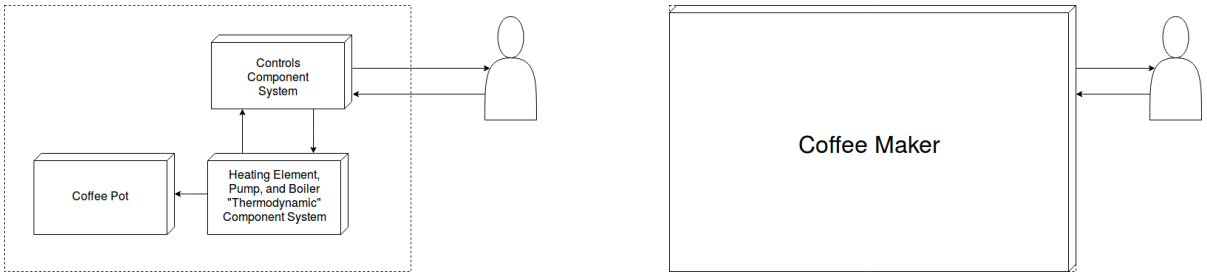
Globally, annual coffee production last year was 9 million metric tons (Coffee market...) - this is equivalent to about a million African elephants (Frei) in weight, and fills 150 million 60 kg coffee sacks. Coffee is clearly loved around the world. For some, the drink is an experience - a treat to be enjoyed when a suitable occasion arises. For others, coffee is a necessary foundation of a productive day of work and activity. Regardless of the reasons for drinking, very few people understand the mechanics of coffee brewing. This is evident by uninformed consumer decisions regarding coffee makers. Despite an appearance of simplicity, coffee makers can be understood through a wide variety of experimental frames, levels of system specification, and modeling paradigms. Through modeling and simulation, valuable insights about coffee brewing can be uncovered.

1.1 Experimental Frames

The system can be framed from the hydrological/chemical perspective, thermodynamic perspective, control systems perspective, or the user perspective (to name just a few). The experimental frame serves as a lens for any modeling and simulation effort. For this report, the system will be approached primarily in a thermodynamic and control systems experimental frame. Using this frame enables the capturing of core user IO behavior, as well as the fundamental coffee brewing thermodynamic behavior. Other possible experimental frames will be largely reduced to assumptions. For example, the uid mechanics will be reduced to an assumption of constant mass ow rate between the boiler and coffee pot during brewing. Further, the chemical composition of the water will be treated as a parameter, rather than a variable.

1.2 Levels of System Specification

There is value in developing multiple specifications of the model, at varying levels of specification. However, the "coupled component" (level four) system specification (Zeigler) will serve as the primary lens on the system during analysis. This level of specification combines component systems through IO behavior, and is particularly intuitive/meaningful for the coffee maker system. A representation of the system is provided in Figure 1a.



(a) Coupled Decomposed System Diagram

(b) Atomic Composed System Diagram

Figure 1: System Diagrams at Different Levels of Decomposition

Presenting these components as a larger system provides an alternative perspective on the system (Figure 1b). This in turn lends itself to analysis of the behavioral, as opposed to structural, dynamics of the system. In the sense that this representation of the system does not capture structure and component system coupling, it is a lower level system specification (level three). However, being that the composed model exhibits closure under coupling, the same tools for analysis of the atomic component models can be used to analyze the composed model. This presents opportunities for additional insight and analytical flexibility for the modeling and simulation effort.
System inference and system design are outside the scope of this project. Rather, this project takes a systems analysis perspective on modeling and simulation. As such, the foundational model for this project is a high level system specification. From there, lower level system specifications are generated and interpreted.

1.3 Modeling Formalism

Most modern coffee makers utilize digital, discrete-time systems for measurement, control, and interacting with the user. However, the underlying thermodynamic, chemical, and electromechanical behavior lends itself to continuous-time representation. This plurality of possible time bases, along with a multitude of formats for quantifying input, state, and output (eg. "dispensing" vs. +13 mL/sec), gives rise to a great number of modeling formalism possibilities. In this project, the discrete event system specification (DEVS) formalism is used for simulation. This modeling paradigm captures the system dynamics of the control system with relative ease. However, the thermodynamic behavior is a bit more difficult to capture with this modeling formalism. The boiler heating element, pump, and coffee pot fluid mixing would most naturally be quantified through a differential equation system specification. However, by discretizing the differential equations (more details on this in Section 4.2.2), a discrete time system specification arises. This way, the "DEVS Wrapper" approach can be employed to simulate the discrete time behavior within a DEVS environment. The DEVS Bus approach brings power and flexibility to the modeling effort, mainly due to the "universality of the DEVS representation and the ability of DEVS bus to support the basic system formalisms" (Zeigler). The only externality of the formalism embedding described above is the creation of a "time step" parameter. With smaller values for the time step parameter, the DEVS simulation results will converge with the differential equation system specification behavior. For this modeling project, the DEVS Suite software will serve as the simulator for the DEVS modeling described above.

2 System Classification

The coffee maker system being modeled is in some sense a cyber-physical system. Cyber-physical systems

> "combine computing and networking with physical dynamics. Cyber-physical systems require model combinations that integrate the continuous dynamics of physical processes (often described using differential equations) with models of software. Diverse models are most useful in applications where timed interactions between components are combined with conventional algorithmic computations," (Ptolemaeus).

While networking details are not considered in this analysis, this system classification is otherwise fitting. Cyber-physical systems are often at the heart of modeling and simulation activities. As such, the modeling and simulation insights generated by this project may be adaptable to a wide variety of other interesting systems.

3 Problem Description

3.1 Questions

This report approaches the coffee maker system from the thermodynamic and electronic perspective. Valuable and actionable insights is the objective of the modeling and simulation effort. Providing coffee consumers with additional insight into the coffee brewing process will relieve some of the information asymmetry in the market - a key problem detailed in Section 1. While a holistic and open-minded perspective is applied to the interpretation of the simulation results, some possible questions addressed by the modeling and simulation effort include:

- How does the selection of thermodynamic components influence the behavior of the brewer?
- Does the controls system latency give rise to any instability, or otherwise problematic behavior?
- What issues arise at the intersection of the digital control system electronics and the analog electromechanical system?
- How does erroneous user behavior impact coffee maker performance?
- Do stability and/or safety issues arise for simple heating systems and simple safety shutoff systems?

3.2 Assumptions

The following assumptions support a simplification of the problem and a focus on the desired experimental frame:

- Constant mass flow rate during coffee pot filling

- Perfectly insulated boiler and coffee pot - no convection or conduction heat loss
- Absence of any radiative heat transfer
- Constant thermodynamic properties[1]
- No mass loss through steam or other mechanisms
- Ideal operation of boiler heating element - constant heat flux
- Complete absence of impurities in water - water is considered distilled
- No consideration for chemical reactions between the water and brewer components (e.g. rusting)

4 Approach

4.1 Model Development

The essential construct of the system includes a boiler system component model, a coffee pot component model, and modeled digital electronics for user interaction and coffee maker control. The system is open with respect to defined user inputs (eg. user pressing the "On" button) and outputs (eg. indicator lights to inform the user of brewing state).

4.1.1 System Diagram

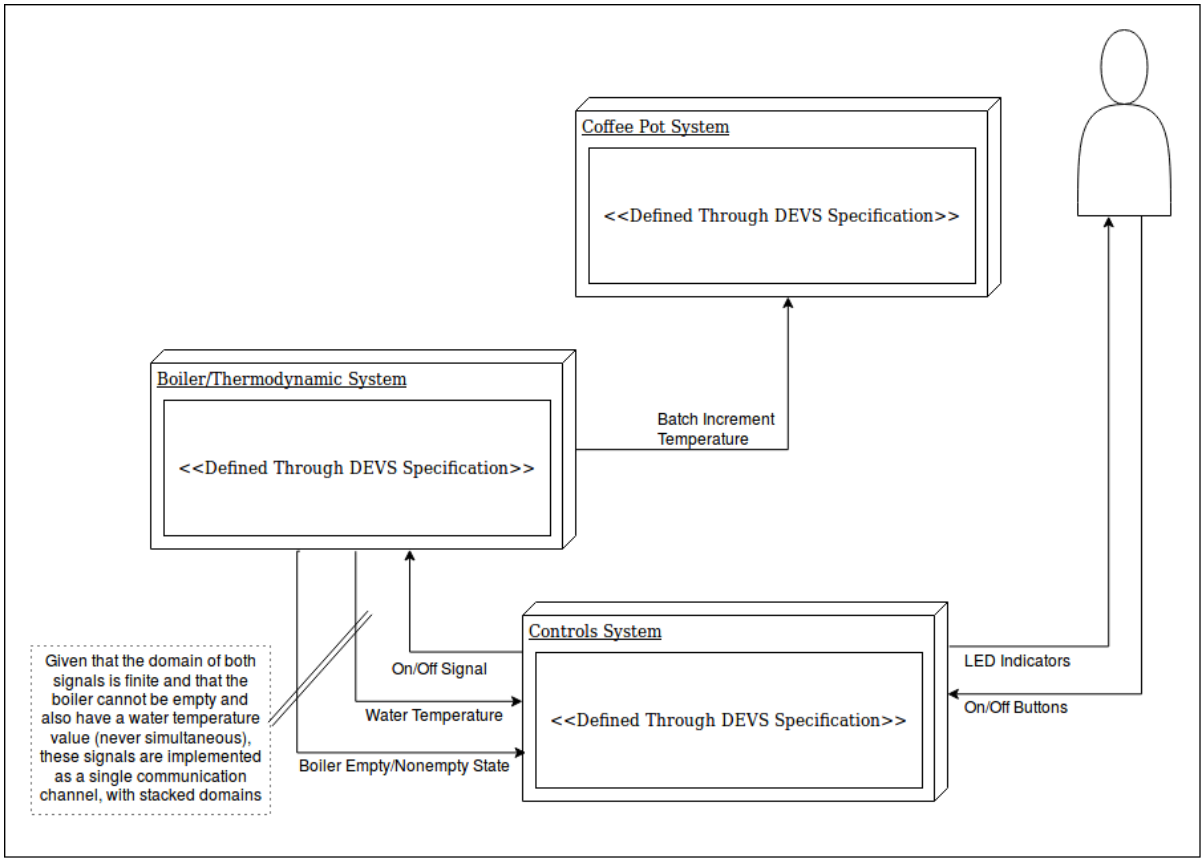


Figure 2: System Diagram for Coffee Maker Model and Simulator

[1]While thermodynamic properties can vary by temperature and pressure, the change is relatively small for environments close to room temperature and sea level pressure - these relatively small changes can be ignored with negligible impact on results

4.2 System Specification

Given that this project is a systems analysis initiative, the foundational definition of the system is a high level system specification. As the analysis proceeds, lower level system specifications are produced.

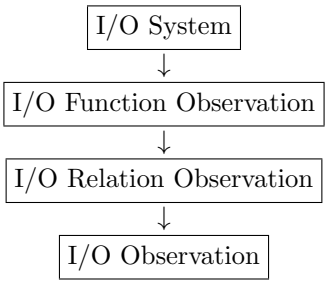


Figure 3: System Specification Hierarchy

4.2.1 Model Components Qualitative Description

Thinking about the system holistically is an important predecessor to a more rigorous mathematical definition. As such, this section seeks to provide some context and focus for component system structure and behavior. Of particular importance is considering how the component systems will interact, and ports/constructs by which the system structure translates to system behavior.

Thermodynamic Component System
Quantized DESS Embedded in DEVS
Signals are Discrete Events

- Input
  - Brew Signal - Coupled with Controls System
  - Off Signal - Coupled with Controls System
- Output
  - Water Temperature - Coupled with Controls System
  - Boiler Emptiness - Coupled with Controls System
  - Incremental Brew Volume - Coupled with Coffee Pot System
- Functions
  - Heat Flux - Internal State Transition
  - Brewing Flow - Internal State Transition
  - Brew Initiation - External State Transition
  - Brew Termination - External State Transition
- Parameters
  - Water Volume
  - Heating Element Power
  - Specific Heat Capacity of Water
  - Fluid Density
- State Variables
  - Boiler Phase
  - Boiler Water Temperature
  - Fluid Portion in Pot
  - Time Until Next Event (σ)

Figure 4: Thermodynamic System Interfaces and Qualitative Structure

Controls Component System
DEVS
Signals are Discrete Events

- Input
  - On/Off Buttons - Inputs from User
  - Water Temperature - Coupled with Thermodynamic System
  - Boiler Emptiness - Coupled with Thermodynamic System
- Output
  - User LED Indicators - External Output to the User
  - Brew Signal - Coupled with Thermodynamic System
  - Off Signal - Coupled with Thermodynamic System
- Functions
  - User Commands - External State Transition
  - Temperature Safety Shutoff - External State Transition
  - Brew Completed Shutoff - External State Transition
  - Safety Timeout - Internal State Transition
  - Indicators Timeout - Internal State Transition
- State Variables
  - Controls Phase
  - Time Until Next Event (σ)

Figure 5: Controls System Interfaces and Qualitative Structure

Coffee Pot Component System
DEVS
Signals are Discrete Events

- Input
  - Incremental Brew Coffee Temperature - Coupled with Thermodynamic System
- Output
  - Coffee Pot Temperature - Coupled with Thermodynamic System
- Functions
  - Process Incremental Brew - External State Transition
  - Passivate After Processing - Internal State Transition
- State Variables
  - Phase (Passive or Reading)
  - Number of Incremental Brews Received
  - Coffee Pot Temperature
  - Received Incremental Brew Temperature
  - Time Until Next Event (σ)

Figure 6: Coffee Pot System Interfaces and Qualitative Structure

4.2.2 Building Up The Thermodynamic System Specification

While the thermodynamic component system is ultimately implemented with a DEVS modeling formalism, developing the foundational DESS and DTSS specifications for parts of the system structure provides some additional insight on the modeling process and potential sources of error.

DESS Specification
The temperature increase of the water is based on the following differential equation:

$$
\begin{array}{rll}
\frac{dQ}{dt} & = m * c * \frac{dT}{dt} & \\
Q & = Energy\ Transferred & Joules \\
m & = Mass & Kilograms \\
c & = SpecificHeatCapacity & Joules * Kilograms^{-1} * Celsius^{-1} \\
T & = Temperature & Celsius
\end{array}
$$

Rather than a value for total energy Transferred, heating elements are defined with respect to a heat flux. That is, $\frac{dQ}{dt}$ is known, not Q. The units for $\frac{dQ}{dt}$ are Joules per second. Using this definition in the equation, we arrive at:

$$q = m * c * \frac{dT}{dt}$$

Leaving the possibility for q to vary over time, this equation is of the linear differential form $x = A * y + B * \frac{dy}{dt} + C * \frac{d^2y}{dt^2} + ...\ A,\ B,\ and\ C$ in this equation are constant coefficients.

DTSS Specification

Embedding the differential equation system specification in a discrete time system specification utilizes numerical methods to discretize the thermodynamic functions.

For this project, the Euler approach is employed to discretize the problem. The Euler approximation suggests that for sufficiently small step sizes h,

$$\rho(t+h) = \rho(t) + h * \frac{d\rho}{dt}$$
$$\frac{d\rho}{dt} = \frac{\rho(t+h) - \rho(t)}{h}$$

A causal implementation of this Euler approach (one that uses only current and past values) will enable the thermodynamic behavior to be handled with the DEVS Suite.

$$\frac{d\rho}{dt} = \frac{\rho(t) - \rho(t-h)}{h}$$

To handle the "startup problem" for the differential equation discretization, the water in the boiler is initialized to 25 degrees Celsius. While the Euler approximation may be less accurate than alternative approaches, it shines in the area of stability.

Using the discretized differential equation in the heat flux equation results in the following function:

$$
\begin{array}{rl}
q & = m * c * \frac{T(n) - T(n-h)}{h} \\
T(n) & = T(n-h) + h * \frac{q}{m*c}
\end{array}
$$

Finally, we make a substitution for the mass value, as water density (ρ) and water volume (V) values are more intuitive and meaningful then a mass value.

$$T(n) = T(n-h) + h * \frac{q}{\rho * V * c}$$

This $T(n)$ equation will serve as the basis for the DEVS specification detailed in 4.2.3.

4.2.3 DEVS System Specification for Component Atomic Models

Thermodynamic System

$$
\begin{aligned}
X &= \{0, 1\} \\
Y &= \{y \mid 0 \leq y\} \\
S &= \{Phase, Temp, BrewedPart, \sigma\},\ where \\
&\quad Phase\ \epsilon\ \{\text{“}Passive'', \text{“}Brewing''\} \\
&\quad 0 \leq Temp \\
&\quad 0 \leq BrewedPart \leq 100 \\
&\quad \sigma\ \epsilon\ \mathbb{R}_0^+
\end{aligned}
$$

$$
\delta_{ext}(Phase, Temp, BrewedPart, \sigma, e, x) = \begin{cases} (\text{“}Brewing'', Temp, BrewedPart, TIME\_STEP) & if\ x = 1 \\ (\text{“}Passive'', Temp, BrewedPart, \sigma - e) & otherwise \end{cases}
$$

$$
\delta_{int}(\text{“}Brewing'', Temp, BrewedPart, \sigma) = \begin{cases} (\text{“}Passive'', Temp, BrewedPart, \infty) & if\ 100 \leq BrewedPart \\ (\text{“}Brewing'', \\ Temp + TIME\_STEP * \frac{HEAT\_FLUX}{VOLUME * DENSITY * HEAT\_CAPACITY}, \\ BrewedPart + \frac{TIME\_STEP}{FLOW}, \\ TIME\_STEP) & otherwise \end{cases}
$$

$$
\lambda(\text{“}Brewing'', Temp, BrewedPart, \sigma) = \begin{cases} Temp & if\ BrewedPart < 100 \\ 0 & otherwise \end{cases}
$$

$$
ta(Phase, Temp, BrewedPart, \sigma) = \sigma
$$

Some notes on the specification above:

- Temperature and emptiness values should never be sent simultaneously - an empty boiler has no water temperature value
- Given the bullet above and the domain limits for both temperature and emptiness, these outputs can be stacked in a single communication channel
  - An output value of 0 indicates the boiler is empty, whereas any value above 0 indicates the water temperature
  - The temperature can never be 0 (frozen solid water would not brew)

Controls System

$$
\begin{array}{rcll}
InPorts & = & \{User, Thermo\},\ where & \\
 & & X_{User} = \{0, 1\} & \\
 & & X_{Thermo} = \{x \mid 0 \ \leq \ x\} & \\
 & & & \\
X & = & \{(p, v) \mid p \ \epsilon \ InPorts,\ v \ \epsilon \ X_p\} & \\
 & & & \\
OutPorts & = & \{User, Thermo\},\ where & \\
 & & Y_{User} = \{Blank, Brewing, Ready\} & \\
 & & Y_{Thermo} = \{0, 1\} & \\
 & & & \\
Y & = & \{(p, v) \mid p \ \epsilon \ OutPorts,\ v \ \epsilon \ Y_p\} & \\
 & & & \\
S & = & \{Phase, \sigma\},\ where & \\
 & & Phase \ \epsilon \ \{Passive, BrewRequest, BrewProgress, & \\
 & & \qquad BrewReady, BrewTerminate, Standby\} & \\
 & & \sigma \ \epsilon \ \mathbb{R}_0^+ & \\
 & & & \\
\delta_{ext}(Passive, \sigma, e, (User, v)) & = & (BrewRequest, REGISTER\_TIME) & if\ v = 1 \\
 & & (Passive, \sigma - e) & otherwise \\
 & & & \\
\delta_{int}(BrewRequest, \sigma) & = & (BrewProgress, SAFETY\_TIMEOUT) & \\
\lambda(BrewRequest, \sigma) & = & (Thermo, 1)\ \&\ (User, Brewing) & \\
 & & & \\
\delta_{ext}(BrewProgress, \sigma, e, (User, v)) & = & (BrewProgress, \sigma - e) & if\ v = 1 \\
 & & (BrewTerminate, REGISTER\_TIME) & otherwise \\
\delta_{ext}(BrewProgress, \sigma, e, (Thermo, v)) & = & (BrewTerminate, REGISTER\_TIME) & if\ MAX\_SAFE\_TEMP < v \\
 & & (BrewReady, DISPLAY\_TIME) & if\ v = 0 \\
 & & (BrewProgress, \sigma - e) & otherwise \\
\delta_{int}(BrewProgress, \sigma) & = & (BrewTerminate, REGISTER\_TIME) & \\
 & & & \\
\delta_{int}(BrewTerminate, \sigma) & = & (Passive, \infty) & \\
\lambda(BrewTerminate, \sigma) & = & (Thermo, 0)\ \&\ (User, Blank) & \\
 & & & \\
\delta_{int}(BrewReady, \sigma) & = & (Standby, INDICATOR\_TIMEOUT) & \\
\lambda(BrewReady, \sigma) & = & (Thermo, 0)\ \&\ (User, Ready) & \\
 & & & \\
\delta_{int}(Standby, \sigma) & = & (Passive, \infty) & \\
\lambda(Standby, \sigma) & = & (User, Blank) & \\
 & & & \\
ta(Phase, \sigma) & = & \sigma &
\end{array}
$$

Some notes on the specification above:

- The standard phase progression is $Passive \rightarrow BrewRequest \rightarrow BrewProgress \rightarrow BrewReady \rightarrow Standby \rightarrow Passive$
- Events which could result in behavior other than the standard phase progression include:
  - Mid-brew user "Off" request
  - Emergency temperature overheat shutoff
  - Emergency timeout shutoff

Coffee Pot System

$$
\begin{aligned}
X &= \{x \mid 0 \leq x\} \\
Y &= \{y \mid 0 \leq y\} \\
S &= \{Phase, TempCount, PotTemp, InTemp, \sigma\},\ where \\
&\quad Phase\ \epsilon\ \{\text{“}Passive'', \text{“}Reading''\} \\
&\quad 0 \leq TempCount \\
&\quad 0 \leq PotTemp \\
&\quad 0 \leq InTemp \\
&\quad \sigma\ \epsilon\ \mathbb{R}_0^+
\end{aligned}
$$

$$\delta_{ext}(\text{“}Passive'', TempCount, PotTemp, InTemp, \sigma, e, x) = (\text{“}Reading'', TempCount + 1, PotTemp, x, 0)$$

$$\delta_{int}(\text{“}Reading'', TempCount, PotTemp, InTemp, \sigma) = (\text{“}Passive'', TempCount, \tfrac{PotTemp*(TempCount-1)+InTemp}{TempCount}, InTemp, \infty)$$

$$\lambda(\text{“}Reading'', TempCount, PotTemp, InTemp, \sigma) = PotTemp$$

$$ta(Phase, Temp, BrewedPart, \sigma) = \sigma$$

Some notes on the specification above:

- The system keeps a running average of the received temperature values
- This specification would need to change if each brewed increment was not of the same volume
- The system immediately outputs the temperature of the coffee pot after receiving a new increment of coffee
- The specification assumes perfect and immediate mixing of the new coffee increment into the previously brewed coffee

Confluence Function
To handle simultaneous events, the confluence function is defined below. This particular specification runs the internal state transition function before the external state transition function - an approach applied to all the component models.

$$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$$

4.3 Simulation Strategy

From a system architecture perspective, there is an important distinction between simulator and model. For this project, the DEVS Suite is used as the simulator, while the model is developed in Java and conforms to the DEVS Suite simulator standards. The capabilities of the DEVS Suite enable the stimulation of the coffee brewer model, as well as the processing of outputs. Further, the flexible and modular construction of the coffee brewer model enable model reusability, extensibility, and adaptability. A screenshot of the model, within the DEVS Suite simulator, is provided below:

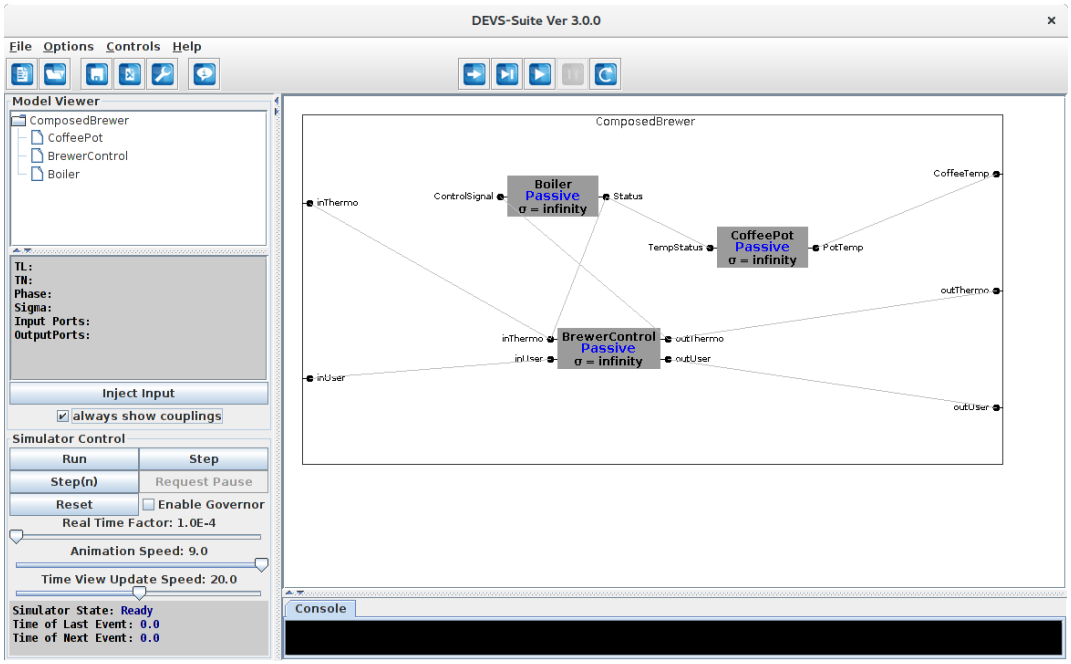


Figure 7: Coffee Brewer Coupled Model Within DEVS Suite Simulator

This project employs a sequential approach to simulation execution. While the premise of a parallel and/or distributed strategy seems attractive, the scope of the project, low anticipated computational demand of the simulation, and availability of high performance computers suggest that a parallel/distributed strategy does not meet the cost-benefit threshold. That said, the distinction between model and simulator in the project documentation and code provides a foundation for further extension of the project (outside of the scope of the course) into the realm of distributed simulation models.

4.3.1 Experimental Frame

The experimental frame for any simulation captures the lens for the experimentation and is a key component of simulation strategy. By building Requester and Transducer models, the experimental frame can be formalized. For this project, user inputs and artificial thermocouple readings are the most important means to stimulate the model. As such, these capabilities are built into the Requester model. The most important metric for discerning coffee quality is the temperature of the coffee pot. With this in mind, the Transducer is integrated into the model and coupled in order to record this important value.

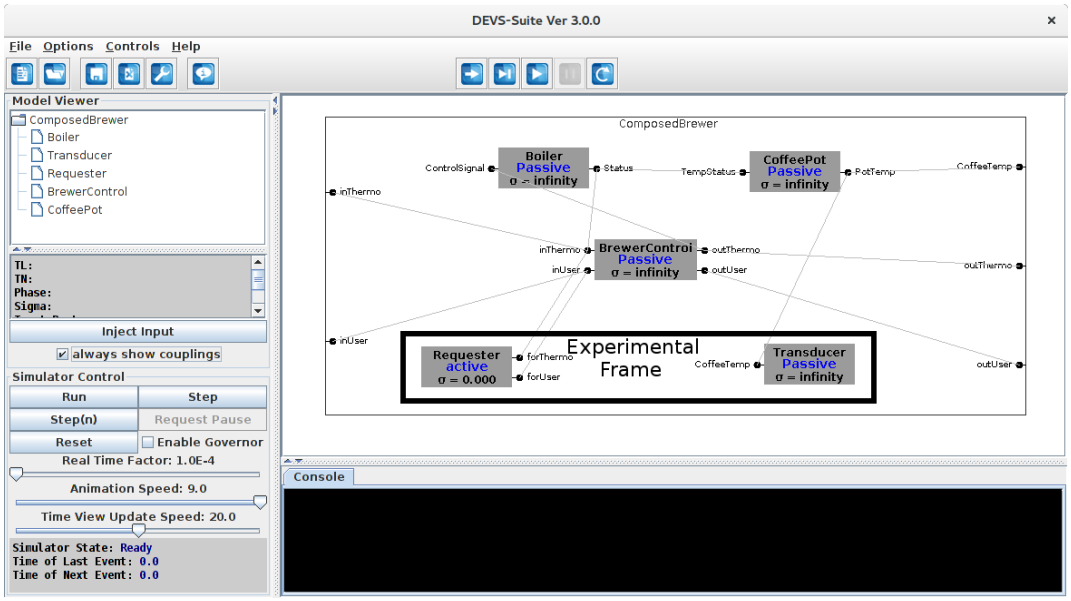


Figure 8: Experimental Frame Built Into Model

4.3.2 Animation

An important aspect of simulation strategy is the approach to interpreting simulation results. Seeing as the coffee brewer system has an intuitive and natural physical representation, animation is of substantial value. While this provides no new data, it does provide more in the way of insights/value. Additionally, animation enables a wider audience, particularly those with a non-technical background, to understand and interpret results.
For this project animation was implemented using a separate program, developed from scratch in Python. During execution, the DEVS system was programmed to output key value pairs to regular system files within the project directory. The Python animation program is started separately from the DEVS simulation. Through a continuous refresh, the Python animation can track the progress of the DEVS simulation as the user "steps" through the model execution. A screenshot of the animation program is provided below.

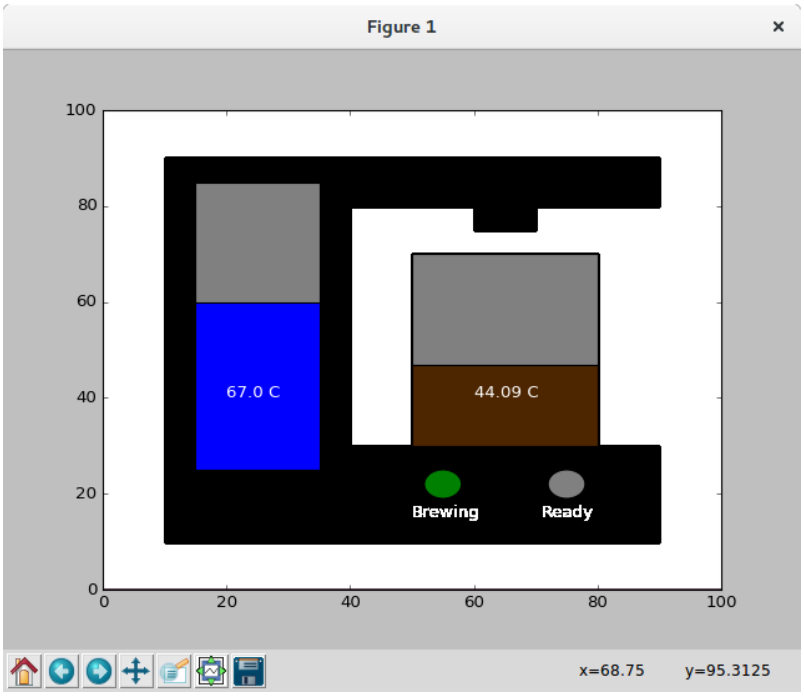


Figure 9: Coffee Brewer Animation

4.4 Experimental Details

The modeling and simulation of the coffee maker was primarily focused on more quantitative safety and functional features. This included the safety shutoff feature, the overheat prevention shutoff, and the temperature of the coffee. Tests were conducted using variation in heat flux and vertical flow. The temperature within the boiler and coffee pot was collected at each time interval. The difference between those temperatures at different heat flux values will be compared, along with whether or not a particular shut down mechanism was activated.

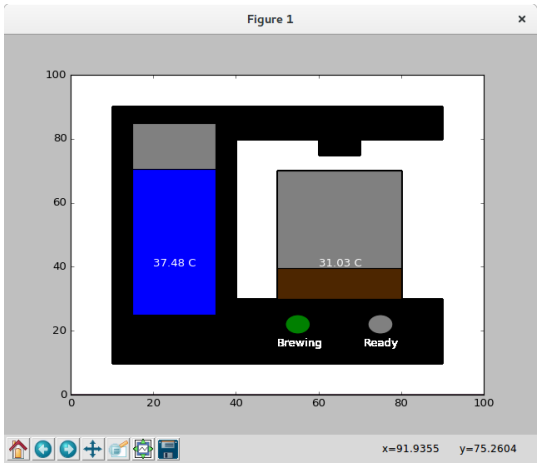
4.4.1 Regular Brew

Boiler Setup

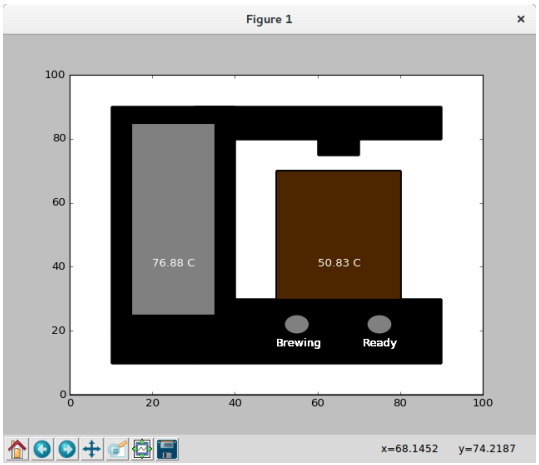
- TimeStep = 1 sec
- HeatFlux = 900 W
- Volume = 0.001 m^3 (1 Liter)
- Density = 1000 $\frac{kg}{m^3}$
- HeatCapacity = 4180 $\frac{J}{kg*C}$
- Flow = 240 sec (to fill pot)

Controller Setup

- RegistrationTime = 1.5 sec (User Input Latency)
- SafetyTimeout = 600 sec
- DisplayTime = 0.5 sec (User Output Latency)
- IndicatorTimeout = 300 sec
- MaxSafeTemp = 100 $degrees\ Celsius$



(a) Coffee Maker After One Minute



(b) Coffee Maker End State

Figure 10: Regular Brew Status Snapshots

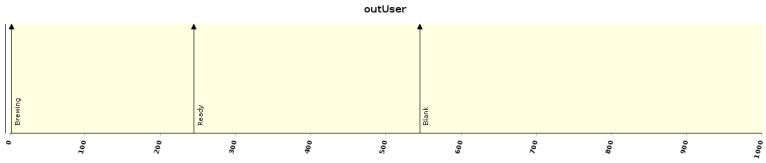


Figure 11: User Indicator Progression

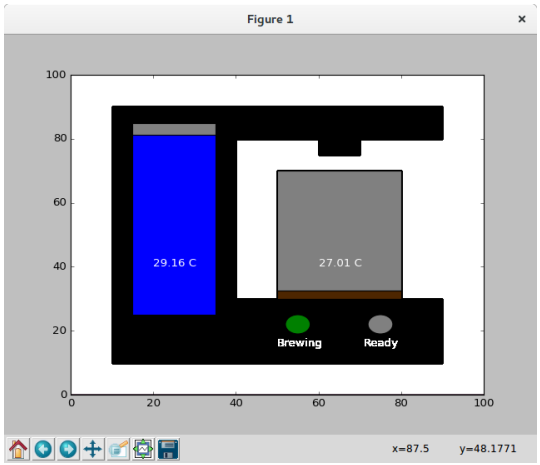
4.4.2 Safety Timeout

Boiler Setup

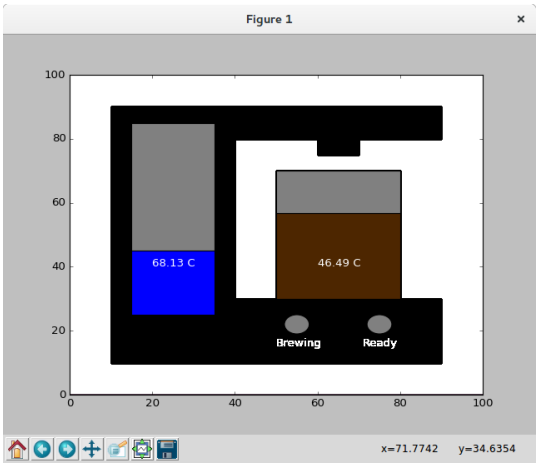
- TimeStep = 1 *sec*
- HeatFlux = 300 W
- Volume = 0.001 m^3 (1 Liter)
- Density = 1000 $\frac{kg}{m^3}$
- HeatCapacity = 4180 $\frac{J}{kg*C}$
- Flow = 900 *sec* (to fill pot)

Controller Setup

- RegistrationTime = 1.5 *sec* (User Input Latency)
- SafetyTimeout = 600 *sec*
- DisplayTime = 0.5 *sec* (User Output Latency)
- IndicatorTimeout = 300 *sec*
- MaxSafeTemp = 100 *degrees Celsius*



(a) Coffee Maker After One Minute



(b) Coffee Maker End State

Figure 12: Safety Timeout Status Snapshots

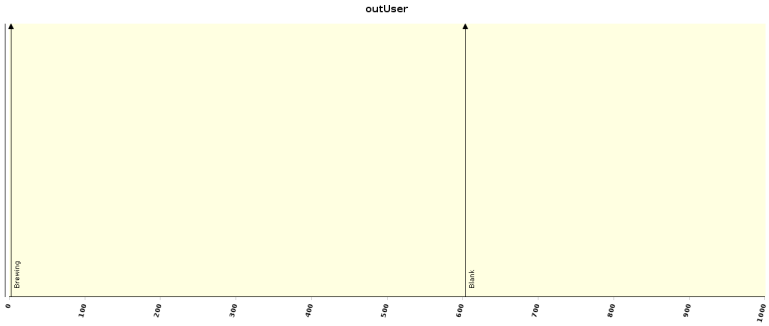


Figure 13: User Indicator Progression

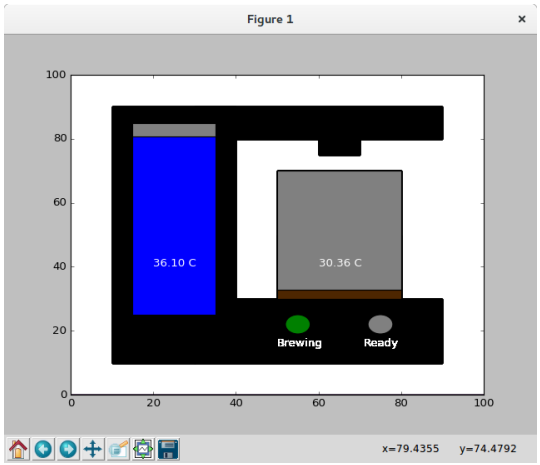
4.4.3 Overheat Shutoff

Boiler Setup

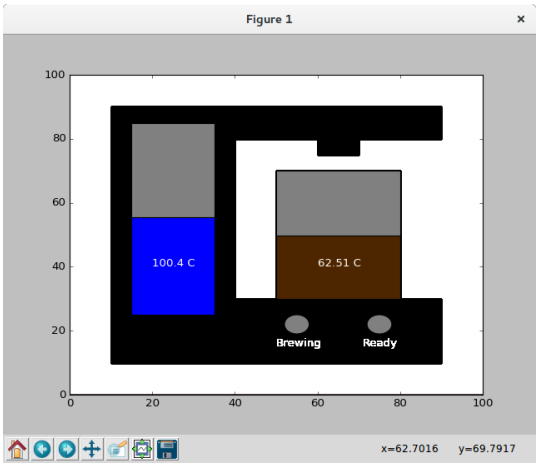
- TimeStep = 1 *sec*
- HeatFlux = 800 W
- Volume = 0.001 m^3 (1 Liter)
- Density = 1000 $\frac{kg}{m^3}$
- HeatCapacity = 4180 $\frac{J}{kg*C}$
- Flow = 800 *sec* (to fill pot)

Controller Setup

- RegistrationTime = 1.5 *sec* (User Input Latency)
- SafetyTimeout = 600 *sec*
- DisplayTime = 0.5 *sec* (User Output Latency)
- IndicatorTimeout = 300 *sec*
- MaxSafeTemp = 100 *degrees Celsius*



(a) Coffee Maker After One Minute



(b) Coffee Maker End State

Figure 14: Overheat Shutoff Status Snapshots

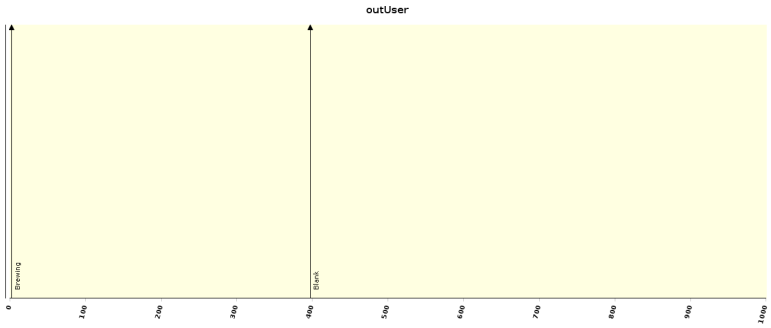


Figure 15: User Indicator Progression

5 Results

The simulation was run under the regular brew, safety timeout, and overheat shutoff settings. In the regular brew settings, the temperature of the output coffee was below the safety threshold and was lower that what would typically be expected for coffee by about 20^{o}C, sitting at 50^{o}C. Under the safety timeout settings, the coffee does not completely brew. About 2/3 of the pot is brewed in 10 minutes before the shutoff occurs. In the overheat shutoff test, the boiler had exceeded the acceptable safety threshold temperature within 6 minutes. This also made it impossible to complete the pot.

There is some evidence of strong results verification for the simulation runs. For example, the emergency timeout occurs right at the expected 10 minute mark during that simulation run. Further, the temperature barely reaches 100^{o}C during the overheat shutoff case and the brew time matches expectation in the baseline simulation. The simulator and modeling approach have been successfully adapted to a wide variety of projects, which provides some additional confidence with respect to verification. Validation on the other hand is a more difficult assessment. The timing and temperatures for the coffee brewing are within a physically intuitive range. However, structuring the system so that the pump and heating element always operate together appears to generate some behavior that doesn't align with real world experience. It may be preferable to model the system in such a way that the pump does not operate until the water is at a sufficiently high temperature. Independent operation of the pump and heating element is an option for future exploration and an approach which is likely to result in better model validation.

6 Conclusions

There is an intuitive relationship between the heat flux, fluid flow rate, and the coffee brewers ability to complete the pot without exceeding temperature and time limits. Particularly, even with a relatively low heat flux, a low volumetric flow will result in the brewer shutting down before filling the pot. It makes logical sense for a low rate of transfer from the boiler to delay the brewing beyond a reasonable time frame. What would be worthy of examination in the future would be the use of a higher volumetric flow in order to manage a greater heat flux. This may prevent overheating while allowing the pot to fill. There would likely be an increase in the speed in which it fills as well. Therefore, it would be beneficial to look further into the degree of heat flux allowed in relation to volumetric flow.

7 References

Coffee market ends 2016/17 coffee year in deficit for the third consecutive year. (2017, July 31). Retrieved October 16, 2017, from http://www.ico.org/prices/po-production.pdf

Frei, G. (2016, April 11). Gewicht und Grsse der Elefanten. Retrieved October 16, 2017, from https://de.upali.ch/gewicht-und-grosse/

Ptolemaeus, C. (2014). System design, modeling, and simulation: using Ptolemy II. Berkeley: Ptolemy.org.

Sarjoughian, H. (2017). Chapter 1 - Modeling Concepts WorldView Applications [Powerpoint slides]. Retrieved from https://myasucourses.asu.edu/bbcswebdav/pid-16267679-dt-content-rid-105794536_1/xid-105794536_1

Sarjoughian, H. S., & Sundaramoorthi, S. (2015, April). Superdense time trajectories for DEVS simulation models. In Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (pp. 249-256). Society for Computer Simulation International.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2010). Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Amsterdam: Acad. Press

8 Appendices

8.1 Appendix A - ComposedBrewer Model

```
/*
 *
 *   Author      : Neal DeBuhr
 *   Version     : DEVS-Suite 3.0.0
 *   Date        : 12-19-2017
 */
package CoffeeBrewin;
import java.awt.*;

import GenCol.*;


import model.modeling.*;
import model.simulation.*;

import view.modeling.ViewableAtomic;
import view.modeling.ViewableComponent;
import view.modeling.ViewableDigraph;
import view.simView.*;

public class ComposedBrewer extends ViewableDigraph
{
    public ComposedBrewer(){
        super("ComposedBrewer");

        ViewableAtomic controller = new BrewerControl("BrewerControl", 1.5, 600,
                                                      0.5, 300, 100);
        ViewableAtomic requester = new Requester();
        ViewableAtomic boiler = new Boiler("Boiler", 1, 300, 0.001,
                                           1000, 4180, 900);
        ViewableAtomic pot = new Pot("CoffeePot");
        ViewableAtomic transducer = new Transducer("Transducer");

        add(controller);
        add(requester);
        add(boiler);
        add(pot);
        add(transducer);

        addInport("inThermo");
        addInport("inUser");
        addOutport("outThermo");
        addOutport("outUser");
        addOutport("CoffeeTemp");

        addCoupling(pot, "PotTemp", transducer, "CoffeeTemp");

        addCoupling(this, "inThermo", controller, "inThermo");
        addCoupling(this, "inUser", controller, "inUser");

        addCoupling(requester, "forThermo", controller, "inThermo");
        addCoupling(requester, "forUser", controller, "inUser");

        addCoupling(controller, "outThermo", this, "outThermo");
        addCoupling(controller, "outUser", this, "outUser");

        addCoupling(controller, "outThermo", boiler, "ControlSignal");
        addCoupling(boiler, "Status", controller, "inThermo");

        addCoupling(boiler, "Status", pot, "TempStatus");
        addCoupling(pot, "PotTemp", this, "CoffeeTemp");

        addTestInput("inThermo", new entity("4"), 1);
        addTestInput("inThermo", new entity("8"), 5);
        addTestInput("inThermo", new entity("150"), 8);
        addTestInput("inThermo", new entity("210"), 4);
        addTestInput("inUser", new entity("Off"), 0);
        addTestInput("inUser", new entity("On"), 1);
        addTestInput("inUser", new entity("Off"), 4);
        addTestInput("inUser", new entity("On"), 8);
        addTestInput("inUser", new entity("Off"), 12);
    }

    /**
     * Automatically generated by the SimView program.
     * Do not edit this manually, as such changes will get overwritten.
     */
    public void layoutForSimView()
    {
```

```
        preferredSize = new Dimension(591, 269);

    }
}
```

8.2 Appendix B - Pot Model

```
/*
 *
 *  Author      : Neal DeBuhr
 *  Version     : DEVS-Suite 3.0.0
 *  Date        : 12-19-2017
 */
package CoffeeBrewin;

import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;
import view.modeling.ViewableComponent;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public class Pot extends ViewableAtomic {
    // ViewableAtomic is used instead
    // of atomic due to its
    // graphics capability
    protected double pot_temp;
    protected double received_temp;
    protected double temp_count;

    public Pot() {
        this("CoffeePot");
    }

    public Pot(String name) {
        super(name);
        addInport("TempStatus");
        addOutport("PotTemp");
        }

    public void initialize() {
        phase = "Passive";
        // Possible phases include Passive and Reading
        sigma = INFINITY;
        temp_count = 1;
        pot_temp = 25; //Room temperature
        received_temp = 0; //Changed after input event
        super.initialize();
    }

    public void deltext(double e, message x) {
        Continue(e);

        System.out.println("The elapsed time of the Pot is" + e);
        System.out.println("**************************************");
        System.out.println("External-Phase before: " + phase);

        if (phaseIs("Passive")) {
            for (int i = 0; i < x.getLength(); i++)
                if (messageOnPort(x, "TempStatus", i)) {
                    if (x.getValOnPort("TempStatus", i).toString() != "0")
                        received_temp = Double.parseDouble(x.getValOnPort("TempStatus", i).toString());
                        holdIn("Reading", 0);
                }
        }
        System.out.println("External-Phase after: " + phase);
    }

    public void deltint() {

        System.out.println("Internal-Phase before: " + phase);
        if (phaseIs("Reading")) {
            temp_count++;
            pot_temp = (pot_temp * (temp_count-1) + received_temp)/temp_count;
            phase = "Passive";
            sigma = INFINITY;
        }
        try {
            PrintWriter writer = new PrintWriter("pot-vals.txt", "UTF-8");
            writer.println("PotTemp:" + Double.toString(pot_temp));
            writer.close();
        } catch (FileNotFoundException|UnsupportedEncodingException e) {
        }
        System.out.println("Internal-Phase after: " + phase);
    }
```

```
    public void deltcon(double e, message x) {
        System.out.println("confluent");
        deltext(0, x);
        deltint();
    }

    public message out() {
        message m = new message();

        String average_temp = Double.toString(pot_temp);
        m.add(makeContent("PotTemp", new entity(average_temp)));

        return m;
    }

    public void showState() {
        super.showState();
    }

    public String getTooltipText() {
        return super.getTooltipText() + "\n" + "␣Pot␣Temp:␣" + Double.toString(pot_temp);
    }
}
```

8.3 Appendix C - Boiler Model

```
/*
 *
 *  Author      : Neal DeBuhr
 *  Version     : DEVS-Suite 3.0.0
 *  Date        : 12-19-2017
 */
package CoffeeBrewin;

import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;
import view.modeling.ViewableComponent;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public class Boiler extends ViewableAtomic {
    // ViewableAtomic is used instead
    // of atomic due to its
    // graphics capability
    protected double time_step;
    protected double heat_flux;
    protected double volume;
    protected double density;
    protected double heat_capacity;
    protected double flow;
    protected double boiler_temp;
    protected double brewed_part;

    public Boiler() {
        this("Boiler", 0.5, 100, 10, 1, 1, 0.25);
    }

    public Boiler(String name, double Time_step, double Heat_flux,
                  double Volume, double Density, double Heat_capacity,
                  double Flow) {
        super(name);
        addInport("ControlSignal");
        addOutport("Status");

        time_step = Time_step;
        heat_flux = Heat_flux;
        volume = Volume;
        density = Density;
        heat_capacity = Heat_capacity;
        flow = Flow;
        }

    public void initialize() {
        phase = "Passive";
        // Possible phases include Passive and Brewing
        sigma = INFINITY;
        boiler_temp = 25; // Room temperature in celsius
        brewed_part = 0; // Nothing yet brewed
        super.initialize();
    }

    public void deltext(double e, message x) {
        Continue(e);

        System.out.println("The elapsed time of the Boiler is" + e);
        System.out.println("**************************************");
        System.out.println("External-Phase before: " + phase);

        if (phaseIs("Passive")) {
            for (int i = 0; i < x.getLength(); i++)
                if (messageOnPort(x, "ControlSignal", i)) {
                    if (x.getValOnPort("ControlSignal", i).toString() == "True")
                        holdIn("Brewing", time_step);
                    else if (x.getValOnPort("ControlSignal", i).toString() == "False")
                        holdIn("Passive", sigma-e);
                }
        } else if (phaseIs("Brewing")) {
            for (int i = 0; i < x.getLength(); i++)
                if (messageOnPort(x, "ControlSignal", i)) {
                    if (x.getValOnPort("ControlSignal", i).toString() == "True")
                        holdIn("Brewing", sigma - e);
                    else if (x.getValOnPort("ControlSignal", i).toString() == "False")
                        holdIn("Passive", INFINITY);
                }
```

```
        }
        System.out.println("External-Phase after: " + phase);
    }

    public void deltint() {

        System.out.println("Internal-Phase before: " + phase);
        if (phaseIs("Brewing")) {
            if (brewed_part >= 1) {
                phase = "Passive";
                sigma = INFINITY;
            } else {
                boiler_temp = boiler_temp + time_step*(heat_flux/(volume*density*heat_capacity));
                brewed_part = brewed_part + (time_step/flow);
                holdIn("Brewing", time_step);
            }
        }
        try {
            PrintWriter writer = new PrintWriter("thermo-vals.txt", "UTF-8");
            writer.println("BoilerTemp:" + Double.toString(boiler_temp));
            writer.println("PortionBrewed:" + Double.toString(brewed_part));
            writer.close();
        } catch (FileNotFoundException|UnsupportedEncodingException e) {
        }
        System.out.println("Internal-Phase after: " + phase);
    }

    public void deltcon(double e, message x) {
        System.out.println("confluent");
        deltext(0, x);
        deltint();
    }

    public message out() {
        message m = new message();
        if (phaseIs("Brewing")) {
            if (brewed_part < 1) {
                String current_temp = Double.toString(boiler_temp);
                m.add(makeContent("Status", new entity(current_temp)));
            } else {
                m.add(makeContent("Status", new entity("0")));
            }
        }
        return m;
    }

    public void showState() {
        super.showState();
    }

    public String getTooltipText() {
        return super.getTooltipText() + "\n" + " Boiler Temp: " + boiler_temp
            + "\n" + " Portion Brewed: " + brewed_part;
    }
}
```

8.4 Appendix D - BrewerControl Model

```
/*
 *
 *  Author      : Neal DeBuhr
 *  Version     : DEVS-Suite 3.0.0
 *  Date        : 12-19-2017
 */
package CoffeeBrewin;

import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;
import view.modeling.ViewableComponent;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public class BrewerControl extends ViewableAtomic {
    // ViewableAtomic is used instead
    // of atomic due to its
    // graphics capability
    protected double register_time;
    protected double safety_timeout;
    protected double display_time;
    protected double indicator_timeout;
    protected double max_safe_temp;

    public BrewerControl() {
        this("BrewerControl", 0.4, 30, 0.7, 12, 200);
    }

    public BrewerControl(String name, double Register_time, double Safety_timeout,
                         double Display_time, double Indicator_timeout, double Max_safe_temp) {
        super(name);
        addInport("inThermo");
        addInport("inUser");
        addOutport("outThermo");
        addOutport("outUser");

        register_time = Register_time;
        safety_timeout = Safety_timeout;
        display_time = Display_time;
        indicator_timeout = Indicator_timeout;
        max_safe_temp = Max_safe_temp;
        }

    public void initialize() {
        phase = "Passive";
        // Possible phases include Passive, BrewRequest, BrewProgress,
        // BrewReady, BrewTerminate, and Standby
        sigma = INFINITY;
        super.initialize();
    }

    public void deltext(double e, message x) {
        Continue(e);

        System.out.println("The elapsed time of the CoffeeControl is" + e);
        System.out.println("**************************************");
        System.out.println("External-Phase before: " + phase);

        if (phaseIs("Passive")) {
            for (int i = 0; i < x.getLength(); i++)
                if (messageOnPort(x, "inUser", i)) {
                    if (x.getValOnPort("inUser", i).toString() == "On")
                        holdIn("BrewRequest", register_time);
                    else if (x.getValOnPort("inUser", i).toString() == "Off")
                        sigma = sigma - e;
                }
        } else if (phaseIs("BrewProgress")) {
            for (int i = 0; i < x.getLength(); i++)
                {
                    if (messageOnPort(x, "inUser", i)) {
                        if (x.getValOnPort("inUser", i).toString() == "On")
                            holdIn("BrewProgress", sigma - e);
                        else if (x.getValOnPort("inUser", i).toString() == "Off")
                            holdIn("BrewTerminate", register_time);
                    }
                    if (messageOnPort(x, "inThermo", i)) {
                        if (Double.parseDouble(x.getValOnPort("inThermo", i).toString()) > max_safe_temp)
                            holdIn("BrewTerminate", register_time);
```

```
                        else if (Double.parseDouble(x.getValOnPort("inThermo", i).toString()) == 0)
                            holdIn("BrewReady", display_time);
                        else
                            holdIn("BrewProgress", sigma);
                    }
                }
        }
        System.out.println("External-Phase after: " + phase);
    }

    public void deltint() {

        System.out.println("Internal-Phase before: " + phase);
        if (phaseIs("BrewRequest")) {
            holdIn("BrewProgress", safety_timeout);
        } else if (phaseIs("BrewProgress")) {
            holdIn("BrewTerminate", register_time);
        } else if (phaseIs("BrewTerminate")) {
            phase = "Passive";
            sigma = INFINITY;
        } else if (phaseIs("BrewReady")) {
            holdIn("Standby", indicator_timeout);
        } else if (phaseIs("Standby")) {
            phase = "Passive";
            sigma = INFINITY;
        } else if (phaseIs("Passive")) {
            phase = "Passive";
            sigma = INFINITY;
        }
        System.out.println("Internal-Phase after: " + phase);
    }

    public void deltcon(double e, message x) {
        System.out.println("confluent");
        deltext(0, x);
        deltint();
    }

    public message out() {
        message m = new message();
        String out_user = "Blank";
        if (phaseIs("BrewRequest")) {
            m.add(makeContent("outThermo", new entity("True")));
            m.add(makeContent("outUser", new entity("Brewing")));
            out_user = "Brewing";
        } else if (phaseIs("BrewTerminate")) {
            m.add(makeContent("outThermo", new entity("False")));
            m.add(makeContent("outUser", new entity("Blank")));
            out_user = "Blank";
        } else if (phaseIs("BrewReady")) {
            m.add(makeContent("outThermo", new entity("False")));
            m.add(makeContent("outUser", new entity("Ready")));
            out_user = "Ready";
        } else if (phaseIs("Standby")) {
            m.add(makeContent("outUser", new entity("Blank")));
            out_user = "Blank";
        }
        try {
            PrintWriter writer = new PrintWriter("control-vals.txt", "UTF-8");
            writer.println("OutUser:" + out_user);
            writer.close();
        } catch (FileNotFoundException|UnsupportedEncodingException e) {
        }
        return m;
    }

    public void showState() {
        super.showState();
    }

    public String getTooltipText() {
        return super.getTooltipText() + "\n" + " Temp Safety Cutoff: " + max_safe_temp
            + "\n" + " Safety Timeout: " + safety_timeout;
    }
}
```

8.5 Appendix E - Requester Model

```
/*
 *
 *  Author      : Neal DeBuhr
 *  Version     : DEVS-Suite 3.0.0
 *  Date        : 12-19-2017
 */
package CoffeeBrewin;

import java.util.Random;
import GenCol.*;
import model.modeling.*;
import view.modeling.ViewableAtomic;

public class Requester extends ViewableAtomic {

    // variable for sequencing order of internal transition states and thus outputs
    protected int count;

    public Requester() {
        this("Requester");
    }

    public Requester(String name) {
        super(name);
        addOutport("forUser");
        addOutport("forThermo");
    }

    public void initialize() {
        holdIn("active", 0);
        count = 0;
        super.initialize();
    }

    public void deltext(double e, message x) {
        Continue(e);
    }

    //time scheduler for producing outputs
    public void deltint() {
        Random randomGenerator = new Random();
        int next_event_time = randomGenerator.nextInt(50);
        if (count < 20) {
            holdIn("active", next_event_time);
        } else {
            passivate();
        }
        count = count + 1;
    }

    public message out() {

        //creating empty message
        message m = new message();

        //random selection of output message
        Random randomGenerator = new Random();
        int next_event_type = randomGenerator.nextInt(100000);

        String next_port;
        String next_val;
        if (next_event_type % 5 == 0) {
            next_port = "forUser";
            next_val = "On";
        } else if (next_event_type % 5 == 1) {
            next_port = "forUser";
            next_val = "Off";
        } else {
            next_port = "forThermo";
            next_val = Integer.toString(randomGenerator.nextInt(230));
        }

        content con = makeContent(next_port, new entity(next_val));
        System.out.println("--------Count =" +count);

        m.add(con);
        return m;
    }

    public void showState() {
        super.showState();
    }
```

```
    public String getTooltipText() {
        return super.getTooltipText() + "\n"  + " count: " + count;
    }

}
```

8.6 Appendix F - Transducer Model

```
/*
 *
 *  Author      : Neal DeBuhr
 *  Version     : DEVS-Suite 3.0.0
 *  Date        : 12-19-2017
 */
package CoffeeBrewin;

import GenCol.*;
import model.modeling.*;
import model.simulation.*;
import view.modeling.ViewableAtomic;
import view.simView.*;
import view.modeling.ViewableComponent;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public class Transducer extends ViewableAtomic {
    // ViewableAtomic is used instead
    // of atomic due to its
    // graphics capability

    public Transducer() {
        this("Transducer");
    }

    public Transducer(String name) {
        super(name);
        addInport("CoffeeTemp");
        }

    public void initialize() {
        phase = "Passive";
        sigma = INFINITY;
        super.initialize();
    }

    public void deltext(double e, message x) {
        Continue(e);
    }

    public void deltint() {
    }

    public void deltcon(double e, message x) {
        deltext(0, x);
        deltint();
    }

    public message out() {
        message m = new message();
        return m;
    }

    public void showState() {
        super.showState();
    }

    public String getTooltipText() {
        return super.getTooltipText();
    }
}
```

8.7 Appendix G - Animator Code

```
import time
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle, Circle

# while(True):
for i in range(0,100000):

    with open('../thermo-vals.txt') as f:
        content = f.readlines()
        content = [x.strip('\n') for x in content]
    for line in content:
        the_key = line.split(':')[0]
        the_value = line.split(':')[1]
        if the_key == 'BoilerTemp':
            BoilerTemp = float(the_value)
        if the_key == 'PortionBrewed':
            PortionBrewed = float(the_value)

    with open('../control-vals.txt') as f:
        content = f.readlines()
        content = [x.strip('\n') for x in content]
    for line in content:
        the_key = line.split(':')[0]
        the_value = line.split(':')[1]
        if the_key == 'OutUser':
            OutUser = the_value

    with open('../pot-vals.txt') as f:
        content = f.readlines()
        content = [x.strip('\n') for x in content]
    for line in content:
        the_key = line.split(':')[0]
        the_value = line.split(':')[1]
        if the_key == 'PotTemp':
            PotTemp = float(the_value)

    plt.figure(1)

    plt.axis([0,100,0,100])
    plt.ion()
    plt.show()

    currentAxis = plt.gca()
    currentAxis.add_patch(Rectangle((10, 20), 30, 70, facecolor="black"))
    currentAxis.add_patch(Rectangle((30, 10), 60, 20, facecolor="black"))
    currentAxis.add_patch(Rectangle((10, 10), 20, 10, facecolor="black"))
    currentAxis.add_patch(Rectangle((30, 80), 60, 10, facecolor="black"))
    currentAxis.add_patch(Rectangle((60, 75), 10, 5, facecolor="black"))
    currentAxis.add_patch(Rectangle((15, 25), 20, 60, facecolor="grey"))
    currentAxis.add_patch(Rectangle((15, 25), 20, 60*(1-PortionBrewed), facecolor="blue"))
    currentAxis.add_patch(Rectangle((50, 30), 30, 40, facecolor="grey"))
    currentAxis.add_patch(Rectangle((50, 30), 30, 40*(PortionBrewed), facecolor=[77/255, 38/255, 0]))

    if PortionBrewed < 100:
        if len(str(BoilerTemp)) > 5:
            temp_txt = plt.text(20, 40, str(BoilerTemp)[0:5] + ' C', color="white")
        else:
            temp_txt = plt.text(20, 40, str(BoilerTemp) + ' C', color="white")
    if PortionBrewed > 0:
        if len(str(PotTemp)) > 5:
            pot_txt = plt.text(60, 40, str(PotTemp)[0:5] + ' C', color="white")
        else:
            pot_txt = plt.text(60, 40, str(PotTemp) + ' C', color="white")

    if (OutUser == 'Brewing'):
        currentAxis.add_patch(Circle((55,22), 3, facecolor="green"))
        currentAxis.add_patch(Circle((75,22), 3, facecolor="grey"))
    elif (OutUser == 'Ready'):
        currentAxis.add_patch(Circle((55,22), 3, facecolor="grey"))
        currentAxis.add_patch(Circle((75,22), 3, facecolor="green"))
    else:
        currentAxis.add_patch(Circle((55,22), 3, facecolor="grey"))
        currentAxis.add_patch(Circle((75,22), 3, facecolor="grey"))

    plt.text(50, 15, 'Brewing', color='white')
    plt.text(71, 15, 'Ready', color='white')

    plt.plot([0,100],[0,0])
    plt.draw()
    plt.pause(0.1)
```

```
    if PortionBrewed < 100:
        temp_txt.remove()
    if PortionBrewed > 0:
        pot_txt.remove()
```