JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 4, May – Juni 2006

Modeling a Grid of Traffic Lights -A Case Study Using WinForms, Event Handling and Observer Pattern

Richard Wiener, Editor-in-Chief, JOT, Associate Professor, Department of Computer Science, University of Colorado at Colorado Springs

The goal of this case study is to demonstrate the use of event-handling, winforms, simple graphics and the observer pattern and show these in action in an interesting and fun simulation.

The simulation of a traffic grid consisting of 16 lanes of cars (8 roadways with two lanes in each roadway) controlled by 15 traffic lights forms the basis of this case study. The traffic grid is shown in Figure 1 below.

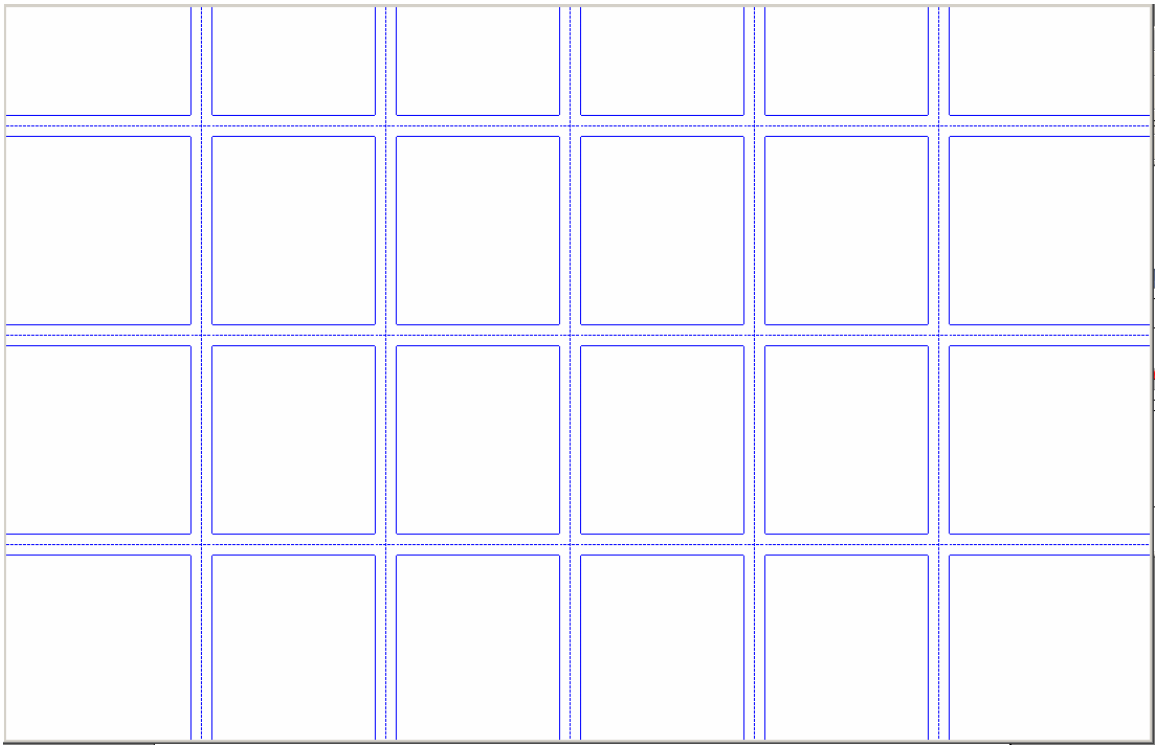


Figure 1 – Traffic Control Grid With 16 lanes and 15 intersections

Cite this column as follows:Richard Wiener: “Modeling a Grid of Traffic Lights -A Case Study Using WinForms, Event Handling and Observer Pattern”, in *Journal of Object Technology*, vol. 5, no. 4, Mai - June 2006, pp. 28-58 http://www.jot.fm/issues/issue_2006_05/column4

The quality of a computer simulation, from a scientific perspective, is based on the degree to which the mathematical models that are used to approximate "reality" capture the essence of that reality. The application program (simulation) must of course faithfully implement the mathematical models that describe the behavior of the system being simulated. For a traffic grid, the aspects of reality that must be modeled are:

1. The arrival pattern of cars into each of the 16 lanes. Here stochastic modeling is appropriate. That is the arrivals of cars is a random phenomena governed by known laws of statistics.
2. The car-following dynamics that determines the speed of a following-car with respect to a leading car directly in front of it.
3. The timing pattern of the 15 traffic signals (the starting time for green, its duration, the duration of amber and the duration of red).

1 MATHEMATICAL MODEL OF THE ARRIVAL PATTERN OF CARS INTO THE 16 LANES

In the world of traffic control modeling, a Poisson probability distribution has long been used to represent the arrival pattern of cars on a roadway.

One of the most important and fundamental aspects of a Poisson process is that the number of events (arrival of cars into a given lane in this case) that occur within a specified time interval is totally independent of the number of events that have occurred in a previous time interval. The only factor that influences the number of events that occur within a specified time interval is the average number of arrivals – the single parameter that completely characterizes the Poisson distribution. This is often referred to as the "memoryless" property of a Poisson distribution. For a more detailed look at the mathematical properties of Poisson distribution, see http://en.wikipedia.org/wiki/Poisson_process.

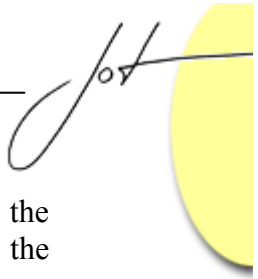
As is well known, if the probability of the number of arrivals within a specified interval of time is given by a Poisson probability distribution function (pdf), the time between successive arrivals (cars in a given lane in this case) is given by an exponential probability distribution.

This exponential pdf is given by:

$$f(t) = \bullet e^{-\bullet t} \text{ for } t > 0.$$

The cumulative distribution function (cdf) is given by:

$$1 - e^{-\bullet t}$$



Here λ represents the average number of cars that arrive / second. The cdf provides the probability that an arrival occurs within t seconds of the previous arrival. The higher the arrival rate λ, the more likely (probability closer to 1) an arrival for a given value of t.

2 COMPUTER IMPLEMENTATION OF POISSON ARRIVAL PROCESS

To generate an exponentially distributed interarrival time generate a uniformly distributed random number from 0 to 1, say U (0 < U < 1). See, http://engineering.dartmouth.edu/~eric/engs027/outlines/19DiscrEvSim.pdf#search='Simulating%20a%20Poisson%20process'

Solve the equation,

$$U = 1 - e^{-\bullet t}$$

for t.

The solution is:

$$t = -1 / \bullet * \ln(1 - U)$$

Since (1 – U) has the same distribution as U, we can simplify the result ,

$$t = -1 / \bullet * \ln(U)$$

This leads to the method,

```
public void SetTimeOfNextArrival(double currentTime,
                                 double arrivalRate) {
   timeOfNextArrival = currentTime + Math.Log(
                       Constants.rnd.NextDouble()) / (-arrivalRate)
}
```

3 CAR FOLLOWING MODEL

The following description is quoted from the paper:

http://tomfotherby.com/Contents/Education/Project/finalReport.pdf

"The simplest mathematical car-following model is linear. This means that as long as there's no obstruction (another car or a red light) in front of a vehicle, it will travel at the speed limit of *k* kilometres per hour.

If a car gets within a distance of *L* metres from an obstruction, it will reduce its speed proportionally to that distance. If a car gets within a distance of *l* metres from an obstruction, it will stop altogether.

Let *x* denote the distance in metres between two cars (measured from the front of one to the back of another, or between the front of one car and a red light ahead).

The speed *v* of the car is:

```
V = k                    if x >= L
0 <= v <= k;     if l < x < L
0                   if x <= l
```

In the second case, $l < x < L$, we're assuming the car changes speed proportionally to changes in distance, so that *v* is given by a "linear" function of *x* (the graph of *v* as a function of *x* is a straight line). This means that $v = mx + c$ for two constants *m* and *c*. These constants can be determined from the two conditions:

1. v=0 when x=l
2. v=k when x=L;

These three numbers (*k*, *l*, and *L*) therefore, completely determine the traffic behavior. The design decision to use the simplest model possible allows the project a wider scope in other aspects, however the model suffers in realistic factors such as drivers' reaction times, or limits to how fast a car can brake or accelerate."

4 COMPUTER IMPLEMENTATION OF CAR-FOLLOWING MODEL

The following method is based on the linear car-following model presented above. Some modifications have been made.

```
// Designed for the first horizontal lane (left to right)
// Returns speed of car based on carAhead
private double CarFollowingModel1a(Car car, Car carAhead) {
   bool skip = false;
   if (car.Speed == 0 && car.ReleaseTime == 0) {
      car.ReleaseTime = currentTime + timeDelayFromStandstill;
      return 0.0;
   } else if (car.Speed == 0 && currentTime >= car.ReleaseTime ||
              car.Speed > 0.0) {
      car.ReleaseTime = 0.0;
      double distanceAhead = 0.0;
      double carDistanceAhead = carAhead.Position.X - car.Position.X;
      // Find the closest light and the distance of car to it
      double lightDistanceAhead = -1.0;
      int lightNumber = 4;
```

```
        for (; lightNumber >= 0; lightNumber--) {
          lightDistanceAhead = lights[lightNumber].Position.X - 20 -
                                          car.Position.X;
          if (lightDistanceAhead < 0 && lightNumber < 4) {
            lightNumber++;
            lightDistanceAhead = lights[lightNumber].Position.X - 20 -
                                          car.Position.X;
            skip = true;
            break;
          }
        }
        if (!skip && lightDistanceAhead > 0 && lightNumber < 4) {
          lightNumber++;
        }
        if (lightDistanceAhead > 0 &&
              (lights[lightNumber].Color == LightColor.red ||
              lights[lightNumber].EWYellow)) {
           distanceAhead = lightDistanceAhead < carDistanceAhead ?
                           lightDistanceAhead : carDistanceAhead;
        } else {
          distanceAhead = carDistanceAhead;
        }
        if (distanceAhead >= longDistanceThreshold) {
          return car.DesiredSpeed;
        } else if (distanceAhead >= shortDistanceThreshold) {
          return car.DesiredSpeed / (longDistanceThreshold -
            shortDistanceThreshold) * distanceAhead -
            car.DesiredSpeed * shortDistanceThreshold /
            (longDistanceThreshold - shortDistanceThreshold);
         } else {
         car.ReleaseTime = 0.0;
          return 0.0;
        }
      }
      return 0.0;
    }
```

5 THE TIMING PATTERN OF THE 15 TRAFFIC SIGNALS

There are two options that will be used to control the traffic signals. The fixed cycle option has all 15 lights turning green at the same time, remaining green for a fixed and specified duration, and then turning amber for a short fixed period and then turning and staying red for a specified duration. This is typical of many intersections. The only real control is the specification of the duration of the green and red signals in each direction.

The second option involves automatic control of each traffic signal. Here, each traffic light operates independently of all the other traffic lights. The duration of green in the east-west/west-east directions, assumed to carry most of the traffic volume is based on an algorithm developed by Richard Wiener. This algorithm is given as follows:

```
For a given light that is green in the EW/WE direction:
      (1) Must stay green for at least 15 seconds.
      (2) Cannot stay green for more than 90 seconds.
      (3) If the number of cars waiting in either red direction equals
          10 or more or the total number of cars in the red directions
          equals 16 or more, changes from green to red.
      (-) A platoon is a sequence of three or more cars where the
          separation from the front of one car to the front of the car
          ahead of it is equal or less than 1.5 longDistanceThreshold
          (48 feet).
      (4) After the light has been green for its minimum of 15 seconds
          (exactly 15 seconds) in the EW/WE direction:
          identify platoons upstream of the light but after the
          previoius light. Identify the last car in the most upstream
          platoon in each dirction and mark these cars.  Look in both
          green directions in making this determination. When both cars
          get through the intersection, the light turns red (assuming
          that constraints 2 and 3 are met). If no platoons are
          identified in either direction upstream at 15 seconds into
          the green cycle, the green immediately turns red.
          Note:  Platoon identification is performed only once at
          exactly 15 seconds into the green cycle.
          In the NS/SN direction, the light stays green for only 12
          seconds independent of traffic since the traffic flow is
          assumed to be much lighter in these directions.
```

6 IMPLEMENTATION OF TRAFFIC SIGNAL ALGORITHM AND OTHER SUPPORTING CLASSES

Method *SensorLightControl*, shown below, implements the algorithm presented in Section 5. The properties LightColor.green and LightColor.red pertain to the lights in the EW/WE direction by default. The array, *grid*, contains 16 objects of type *Lane*.

```
private void SensorLightControl() {
  for (int lightNumber = 0; lightNumber < 15; lightNumber++) {
     TrafficLight light = lights[lightNumber];
     if (light.Color == LightColor.green && currentTime >=
           light.TurnedGreenEW + 90.0) {
        // Light has been green for more than 90 seconds
        light.TurnGreenNS(currentTime);
     } else if (light.Color == LightColor.green && currentTime >
                 light.TurnedGreenEW + 15.0 &&
                 (light.CarsWaitingNS >= 10 ||
                 light.CarsWaitingSN >= 10 ||
                 (light.CarsWaitingNS + light.CarsWaitingSN) >= 16)) {
        // Light has been green for more than 15 seconds and
        // sufficient queue in red direction
        light.TurnGreenNS(currentTime);
```

```
      } else if (light.Color == LightColor.green &&
                  currentTime >= light.TurnedGreenEW  + 15.0 &&
                  !light.DelaySet) {
         if (lightNumber < 5) {
           double delay1 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[0],
                                light.Position.X, Direction.EW);
                     double delay2 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[1],
                                light.Position.X, Direction.WE);
           light.Delay = (delay1 > delay2) ? delay1 : delay2;
         } else if (lightNumber < 10) {
           double delay1 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[12],
                                light.Position.X, Direction.EW);
           double delay2 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[13],
                                light.Position.X, Direction.WE);
           light.Delay = (delay1 > delay2) ? delay1 : delay2;
         } else {
           double delay1 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[14],
                                light.Position.X, Direction.EW);
           double delay2 =
             ComputeTimeDelayToClearUpstreamPlatoon(grid[15],
                                light.Position.X, Direction.WE);
           light.Delay = (delay1 > delay2) ? delay1 : delay2;
         }
         if (light.Delay == 0) {
           light.TurnGreenNS(currentTime);
         }
      } else if (light.Color == LightColor.green &&
            currentTime > light.TurnedGreenEW + 15.0 &&
            currentTime > light.TurnedGreenEW + + 15.0 +
                                light.Delay) {
         light.TurnGreenNS(currentTime);
      }

      if (light.Color == LightColor.red &&
            currentTime >= light.TurnedGreenNS + 12.0) {
         // Light has been green for more than 12 seconds
         light.TurnGreenEW(currentTime);
      }
   }
}
```

Class *Lane* is given as follows:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace TrafficSimulation {
```

```
// Models a lane of cars
public class Lane {

    // Fields
    private List<Car> cars = new List<Car>();
    private double timeOfNextArrival;


    // Indexer
    public Car this[int index] {
        get {
            return cars[index];
        }
    }

    // Properties
    public List<Car> Cars {
        get {
            return cars;
        }
        set {
            cars = value;
        }
    }

    public double TimeOfNextArrival { // Read-only
        get {
            return timeOfNextArrival;
        }
    }

    // Commands
    public void AssignCar(Car car) {
        cars.Add(car);
    }

    public void RemoveCar() {
        cars.RemoveAt(1); // Remove first car
    }

    public  void  SetTimeOfNextArrival(double  currentTime,  double
                                                    arrivalRate) {
        timeOfNextArrival=currentTime+
           Math.Log(Constants.rnd.NextDouble()) / (-arrivalRate);
    }

    // Queries
    public Car CarAt(int index) {
        return cars[index];
    }

    public int Size() {
        return cars.Count;
    }
}
```

```
}
```

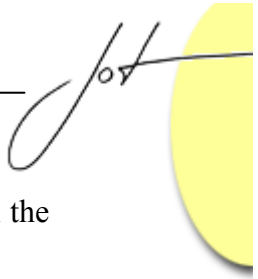
Each *Lane* object contains a generic *List* of *Car* objects.

Class *Car* is given as:

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

namespace TrafficSimulation {

    // Models speed function
    public delegate double CarFollowing(Car car, Car carAhead);

    public class Car {
        // Fields
        private Position position;  // In ft
        private double speed; // In ft/sec
        private CarFollowing speedCurve;
        private double releaseTime; // In seconds
        private double desiredSpeed; // In ft/sec
        private double timeEntersGrid;
        private Hashtable lightsStoppedAt = new Hashtable(); // Holds
                            // the positions of the lights stopped at

        public Car(Position initPosition, CarFollowing speedCurve,
                        double speedLimit, double time) {
            this.speedCurve = speedCurve;
            position = initPosition;
            releaseTime = 0.0;
            desiredSpeed = (0.75 + 0.5 * Constants.rnd.NextDouble()) *
                                    speedLimit;
            timeEntersGrid = time;
            Thread.Sleep(5);
        }

        // Properties
        public Position Position {
            get {
                return position;
            }
            set {
                position = value;
            }
        }

        public CarFollowing SpeedCurve {
            get {
                return speedCurve;
            }
```

```
            }

            public double Speed {
                get {
                    return speed;
                }
                set {
                    speed = value;
                }
            }

            public double ReleaseTime {
                get {
                    return releaseTime;
                }
                set {
                    releaseTime = value;
                }
            }

            public double DesiredSpeed {
                get {
                    return desiredSpeed;
                }
                set {
                    desiredSpeed = value;
                }
            }

            public double TimeEntersGrid {
                get {
                    return timeEntersGrid;
                }
                set {
                    timeEntersGrid = value;
                }
            }

            public int LightsStoppedAt {
                get {
                    return lightsStoppedAt.Count;
                }
            }

            // Commands
            public void SetLightsStoppedAt(int pos) {
                lightsStoppedAt[pos] = pos;
            }
        }
    }
```

Class *Car* has a field of type *CarFollowing*, a delegate that models the many car-following methods that are used.



The field *lightsStopped* at is of type *Hashtable* since duplicates are not allowed in the Hashtable collection.

When a car is constructed its desired speed is computed by,

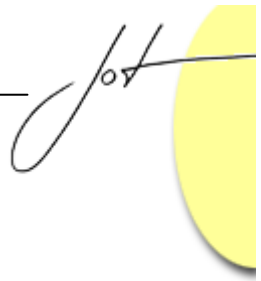
```
desiredSpeed = (0.75 + 0.5 * Constants.rnd.NextDouble()) *
                                                    speedLimit;
```

The desired speed is uniformly distributed from 75 percent to 125 percent the speed limit. The variance of desired speed is what leads to the formation of platoons.

Class *TrafficLight* is given as follows:

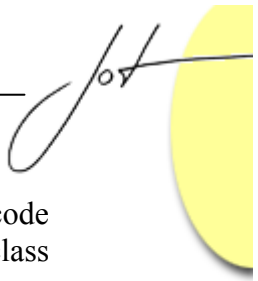
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace TrafficSimulation {

    public enum LightColor { red, green }; // In east-west direction
    public class TrafficLight {

        // Fields
        private LightColor color = LightColor.green;
        private Position position;
        private double redDuration;    // Seconds
        private double greenDuration;  // Seconds
        private double yellowDuration = 4.0; // Seconds
        private bool ewYellow;
        private bool nsYellow;
        private double timeTurnsRed = 1000000.0;
        private double timeTurnsGreen = 0.0;
        private int carsWaitingEW, carsWaitingWE, carsWaitingNS,
                    carsWaitingSN;
        private double turnedGreenEW = 0.0;  // Used only for
                                             automatic
                                             // light control
        private double turnedGreenNS = 1000000; // Used only for
                                             // automatic light
                                             control
        private double delay; // Based on upstream platoon
        private bool delaySet;

        public TrafficLight(Position pos) {
            position = pos;
            TurnGreenEW(0.0);
        }

        public void TurnGreenEW(double time) {
            turnedGreenEW = time;
            turnedGreenNS = 100000;
            color = LightColor.green;
            nsYellow = false;
            ewYellow = false;
```

```
        delay = 0.0;
        delaySet = false;
    }

    public void TurnGreenNS(double time) {
        turnedGreenNS = time;
        turnedGreenEW = 100000;
        color = LightColor.red;
        nsYellow = false;
        ewYellow = false;
        delay = 0.0;
        delaySet = false;
    }

    // Properties
    public LightColor Color {
        get {
            return color;
        }
        set {
            color = value;
        }
    }

    public Position Position {
        get {
            return position;
        }
    }

    public double RedDuration {
        get {
            return redDuration;
        }
        set {
            redDuration = value;
        }
    }

    public double GreenDuration {
        get {
            return greenDuration;
        }
        set {
            greenDuration = value;
        }
    }

    public double YellowDuration {
        get {
            return yellowDuration;
        }
        set {
            yellowDuration = value;
```



```
        }
    }

    public double TimeTurnsRed {
        get {
            return timeTurnsRed;
        }
        set {
            timeTurnsRed = value;
        }
    }

    public double TimeTurnsGreen {
        get {
            return timeTurnsGreen;
        }
        set {
            timeTurnsGreen = value;
        }
    }

    public bool EWYellow {
        get {
            return ewYellow;
        }
        set {
            ewYellow = value;
        }
    }

    public bool NSYellow {
        get {
            return nsYellow;
        }
        set {
            nsYellow = value;
        }
    }

    public int CarsWaitingEW {
        get {
            return carsWaitingEW;
        }
        set {
            carsWaitingEW = value;
        }
    }

    public int CarsWaitingWE {
        get {
            return carsWaitingWE;
        }
        set {
            carsWaitingWE = value;
        }
    }
```

```
        public int CarsWaitingNS {
            get {
                return carsWaitingNS;
            }
            set {
                carsWaitingNS = value;
            }
        }

        public int CarsWaitingSN  {
            get {
                return carsWaitingSN;
            }
            set {
                carsWaitingSN = value;
            }
        }

        public double TurnedGreenEW {
            get {
                return turnedGreenEW;
            }
            set {
                turnedGreenEW = value;
            }
        }

        public double TurnedGreenNS {
            get {
                return turnedGreenNS;
            }
            set {
                turnedGreenNS = value;
            }
        }

        public double Delay {
            get {
                return delay;
            }
            set {
                delay = value;
                delaySet = true;
            }
        }

        public bool DelaySet {
            get {
                return delaySet;
            }
        }
    }
}
```



Class *Simulation* comprises the core of the simulation model. It is over 1000 lines of code and therefore too voluminous to include in its entirety. Only portions of this important class are presented.

Portions of Class Simulation

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace TrafficSimulation {

    public enum Direction { EW, WE, NS, SN };

    public class Simulation {

  // Instance used to communicate with UI class
        public delegate void UpdateView(Lane[] lanes, double time,
                                double systemTimeEW, int carsProcessedEW,
                                double systemTimeNS, int carsProcessedNS,
                                int lightsStoppedAtEW,
                                int lightsStopeedAtNS);

        // Fields
        private Lane[] grid = new Lane[16];
        private Thread runThread;
        private double widthGrid = 3276.0;
        private double heightGrid = 2100.0;
        private bool stopSimulation = false;
        private double timeBetweenTicks = 0.25; // seconds
        private double timeDelayFromStandstill = 0.75; // seconds
        private double currentTime = 0.0; // seconds
        private double speedLimit = 44.0; // ft / sec
        private double longDistanceThreshold = 36;
        private double shortDistanceThreshold = 22;
        private double arrivalRateEW = 0.2; // cars / sec
        private double arrivalRateNS = 0.1;
        private double arrivalRateWE = 0.3;
        private double arrivalRateSN = 0.05;
        private TrafficLight[] lights = new TrafficLight[15];
        private int carsProcessedEW = 0;
        private double totalTimeInSystemEW = 0.0;
        private int carsProcessedNS = 0;
        private double totalTimeInSystemNS = 0.0;
        private int lightsStoppedEW = 0;
        private int lightsStoppedNS = 0;
        private double greenDuration;
        private double redDuration;
        private bool autoLightControl = false;
        private CarFollowing[] carFollowingModel =
                                    new CarFollowing[16];
        private Direction[] directionOfFlow = new Direction[16];
        private double [] position = new double[16];
```

```
// Events
private event UpdateView updateMethod;

// Constructor
public Simulation() {
    Initialize();
}

// Properties
public TrafficLight this[int index] {
    get {
        return lights[index];
    }
}

// Other properties not shown here

// Commands
public void RegisterView(UpdateView update) {
    updateMethod += new UpdateView(update);
}

public void SetLightDuration() {
    for (int i = 0; i < 15; i++) {
        lights[i].RedDuration = redDuration;
        lights[i].GreenDuration = greenDuration;
    }
}

public void ResetSimulationStatistics() {
    totalTimeInSystemEW = 0.0;
    totalTimeInSystemNS = 0.0;
    carsProcessedEW = 0;
    carsProcessedNS = 0;
    lightsStoppedEW = 0;
    lightsStoppedNS = 0;
    currentTime = 0.0;
    Initialize();
}


public void StopSimulation() {
    stopSimulation = true;
}

public void StartSimulation() {
    runThread = new Thread(new ThreadStart(Run));
    runThread.Start();
}

public void Run() {
    // Notify the UI
    updateMethod(grid, currentTime, totalTimeInSystemEW,
```

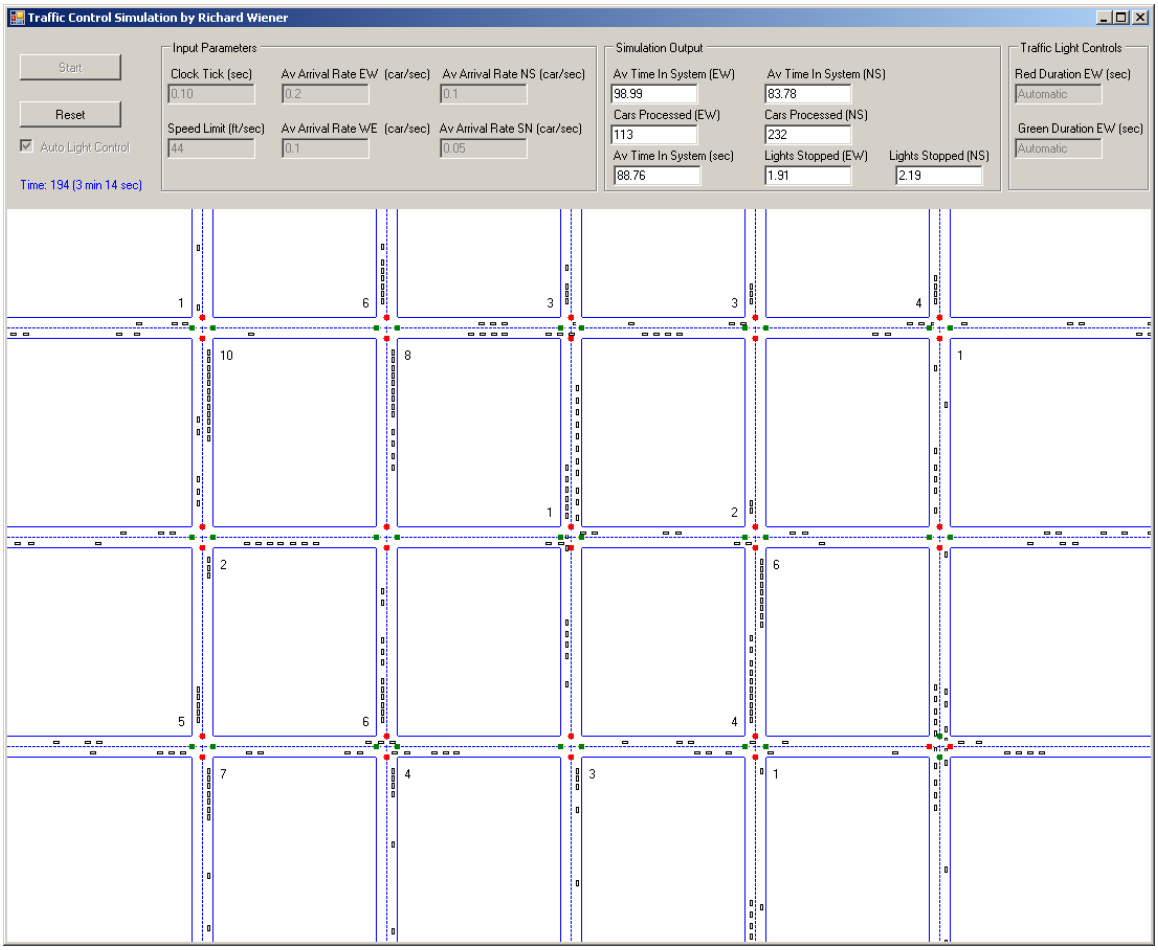
```
                    carsProcessedEW,
                    totalTimeInSystemNS, carsProcessedNS,
                    lightsStoppedEW, lightsStoppedNS);
        while (!stopSimulation) {
            Thread.Sleep((int)(1000 * timeBetweenTicks));
            currentTime += timeBetweenTicks;
            UpdateCarsWaiting();

            if (!autoLightControl) {
                FixedLightControl();
            } else {
                SensorLightControl();
            }

            for (int i = 0; i < 16; i++) {
                SetSpeedAndPositionOfCars(grid[i],
                carFollowingModel[i], directionOfFlow[i],
                position[i]);
            }
            // Notify the UI
            updateMethod(grid, currentTime, totalTimeInSystemEW,
                    carsProcessedEW,
                    totalTimeInSystemNS, carsProcessedNS,
                    lightsStoppedEW, lightsStoppedNS);
        }
    }

    public void Initialize() {
        // Construct an array of method instances
        carFollowingModel[0] = CarFollowingModel1a;
        carFollowingModel[1] = CarFollowingModel2a;
        carFollowingModel[2] = CarFollowingModel3a;
        carFollowingModel[3] = CarFollowingModel4a;
        carFollowingModel[4] = CarFollowingModel3b;
        carFollowingModel[5] = CarFollowingModel4b;
        carFollowingModel[6] = CarFollowingModel3c;
        carFollowingModel[7] = CarFollowingModel4c;
        carFollowingModel[8] = CarFollowingModel3d;
        carFollowingModel[9] = CarFollowingModel4d;
        carFollowingModel[10] = CarFollowingModel3e;
        carFollowingModel[11] = CarFollowingModel4e;
        carFollowingModel[12] = CarFollowingModel1b;
        carFollowingModel[13] = CarFollowingModel2b;
        carFollowingModel[14] = CarFollowingModel1c;
        carFollowingModel[15] = CarFollowingModel2c;

        directionOfFlow[0] = Direction.EW;
        directionOfFlow[1] = Direction.WE;
        for (int i = 2; i <= 10; i += 2) {
            directionOfFlow[i] = Direction.NS;
        }
        for (int i = 3; i <= 11; i += 2) {
            directionOfFlow[i] = Direction.SN;
        }
        for (int i = 12; i <= 14; i += 2) {
            directionOfFlow[i] = Direction.EW;
```

```
            }
            for (int i = 13; i <= 15; i += 2) {
                directionOfFlow[i] = Direction.WE;
            }

            lights[0] = new TrafficLight(new Position(528, 400));
            lights[1] = new TrafficLight(new Position(1056, 400));
            lights[2] = new TrafficLight(new Position(1584, 400));
            lights[3] = new TrafficLight(new Position(2112, 400));
            lights[4] = new TrafficLight(new Position(2640, 400));

            lights[5] = new TrafficLight(new Position(528, 1000));
            lights[6] = new TrafficLight(new Position(1056, 1000));
            lights[7] = new TrafficLight(new Position(1584, 1000));
            lights[8] = new TrafficLight(new Position(2112, 1000));
            lights[9] = new TrafficLight(new Position(2640, 1000));

            lights[10] = new TrafficLight(new Position(528, 1600));
            lights[11] = new TrafficLight(new Position(1056, 1600));
            lights[12] = new TrafficLight(new Position(1584, 1600));
            lights[13] = new TrafficLight(new Position(2112, 1600));
            lights[14] = new TrafficLight(new Position(2640, 1600));

            position[0] = lights[0].Position.Y - 45.0;
            position[1] = position[0] - 30.0;
            position[2] = lights[0].Position.X + 15.0;
            position[3] = position[2] + 30.0;
            position[4] = lights[1].Position.X + 15.0;
            position[5] = position[4] + 30.0;
            position[6] = lights[2].Position.X + 15.0;
            position[7] = position[6] + 30.0;
            position[8] = lights[3].Position.X + 15.0;
            position[9] = position[8] + 30.0;
            position[10] = lights[4].Position.X + 15.0;
            position[11] = position[10] + 30.0;
            position[12] = lights[6].Position.Y - 45.0;
            position[13] = position[12] - 30.0;
            position[14] = lights[11].Position.Y - 45.0;
            position[15] = position[14] - 30.0;
            grid = new Lane[16];

            for (int i = 0; i < 16; i++) {
                grid[i] = new Lane();
            }

            for (int i = 0; i < 16; i++) {
                CreateLaneAndAssignFirstCar(grid[i],
                     directionOfFlow[i], carFollowingModel[i],
                     position[i]);
            }
        }

        private void SetSpeedAndPositionOfCars(Lane lane, CarFollowing
            carFollowingModel, Direction direction, double position) {
            if (direction == Direction.EW) {
```

```
                if (currentTime >= lane.TimeOfNextArrival) {
                    lane.AssignCar(new Car(new Position(0, position),
                        new CarFollowing(carFollowingModel),
                        speedLimit, currentTime));
                    lane.SetTimeOfNextArrival(currentTime,
                                              arrivalRateEW);
                }
                // Set the speed of all the cars in the lane
                for (int index = 1; index < lane.Size(); index++) {
                    lane[index].Speed =
                        lane[index].SpeedCurve(lane[index],
                                               lane[index - 1]);
                }
                // Set the position of all the cars in the lane
                for (int index = 1; index < lane.Size(); index++) {
                    double changeInXPosition =
                        (int)lane[index].Speed * timeBetweenTicks;
                    double oldXPosition = lane[index].Position.X;
                    lane[index].Position =
                        new Position(oldXPosition + changeInXPosition,
                                                    position);
                    if (lane[index].Position.X > widthGrid) {
                        totalTimeInSystemEW += currentTime -
                                               lane[index].TimeEntersGrid;
                       carsProcessedEW++;
                       lightsStoppedEW += lane[index].LightsStoppedAt;
                       lane.RemoveCar();
                    }
                }
            } else if (direction == Direction.WE) {
                // No further details shown

            }
        }

        // No further details shown
    }
```

The GUI is constructed using Visual Studio .NET 2005 using simple and frequently used components. Key input parameters are available for the user to modify with useful default values supplied. Output is presented as the simulation evolves.

A screen shot of the GUI and the simulation in action is shown below.



Portions of class *TrafficLightsUI* are presented below.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;

namespace TrafficSimulation {

    public partial class TrafficLightsUI : Form {

        // Fields
```

```
private Graphics panelGraphics;
private Simulation simulation;
private int feetPerPixel = 3;
private StreamWriter logFile = new StreamWriter(
        new FileStream("Stats.txt", FileMode.Create,
                                FileAccess.Write));
private double time;

public TrafficLightsUI() {
    InitializeComponent();
    panelGraphics = panel.CreateGraphics();
    simulation = new Simulation();
    simulation.RegisterView(Update);
    Control.CheckForIllegalCrossThreadCalls = false;
}

public void Update(Lane[] lanes, double time,
                   double systemTimeEW, int carsEW,
                   double systemTimeNS, int carsNS,
                   int stoppedEW,
                   int stoppedNS) {
    this.time = time;
    if (!sensorsCheckBox.Checked && (int)time == 600) {
        // Change the light duration every 10 minutes
        WriteStatistics();
        double redDur = simulation.RedDuration;
        double greenDur = simulation.GreenDuration;
        simulation.ResetSimulationStatistics();
        simulation.RedDuration = redDur - 5.0;
        if (simulation.RedDuration <= 5.0) {
            simulation.StopSimulation();
            MessageBox.Show("Simulation has ended.");
            Application.Exit();
        }
        simulation.GreenDuration = greenDur + 5.0;
        simulation.SetLightDuration();
        redDuration.Text = "" + simulation.RedDuration;
        greenDuration.Text = "" + simulation.GreenDuration;
        averageWaitTimeEW.Text = "";
        averageWaitTimeNS.Text = "";
        overallAverage.Text = "";
        carsProcessedEW.Text = "";
        carsProcessedNS.Text = "";
        lightsStoppedEW.Text = "";
        lightsStoppedNS.Text = "";
        return;

    }
    if (carsEW > 0) {
        carsProcessedEW.Text = "" + carsEW;
        averageWaitTimeEW.Text = String.Format("{0:f}",
                                   systemTimeEW / carsEW);
        lightsStoppedEW.Text = String.Format("{0:f}", (double)
                                   stoppedEW / carsEW);
    }
    if (carsNS > 0) {
```

```
            carsProcessedNS.Text = "" + carsNS;
            averageWaitTimeNS.Text = String.Format("{0:f}",
                                        systemTimeNS / carsNS);
            lightsStoppedNS.Text = String.Format("{0:f}",
                                   (double) stoppedNS / carsNS);
        }
        if (carsEW + carsNS > 0) {
            overallAverage.Text = String.Format("{0:f}",
           (systemTimeEW + systemTimeNS) / (carsEW + carsNS));
        }

        int currentTime = (int) time;
        int minutes = currentTime / 60;
        int seconds = currentTime - 60 * minutes;
        timeLbl.Text = "Time: " + currentTime + " (" +
                    minutes + " min " + seconds + " sec)";

        for (int i = 0; i < 16; i++) {
            DisplayLanes(lanes[i], simulation.DirectionOfFlow[i]);
        }

        // Display traffic lights
        for (int light = 0; light < 15; light++) {
            DisplayLight(light);
        }
    }


    private void DisplayLanes(Lane lane, Direction direction) {
        if (direction == Direction.EW ||
                direction == Direction.WE) {
            List<Car> cars = lane.Cars;
            Car c = cars[0];
            panelGraphics.FillRectangle(new
                SolidBrush(Color.White), 0, (int)(c.Position.Y /
                                feetPerPixel), panel.Width, 3);
            foreach (Car car in cars) {
                panelGraphics.DrawRectangle(new Pen(Color.Black),
                    (int)(car.Position.X / feetPerPixel),
                    (int)(car.Position.Y / feetPerPixel), 5, 2);
            }
        } else {
            List<Car> cars = lane.Cars;
            Car c = cars[0];
            panelGraphics.FillRectangle(new
                SolidBrush(Color.White),
                (int)(c.Position.X / feetPerPixel), 0, 3,
                panel.Height);
            foreach (Car car in cars) {
                panelGraphics.DrawRectangle(new Pen(Color.Black),
                 (int)(car.Position.X / feetPerPixel),
                 (int)(car.Position.Y / feetPerPixel), 2, 5);
            }
        }
    }
```

```
// No further details shown
```

White rectangular slivers are used to surgically erase all cars traveling either EW/WE or NS/SN. By not erasing the traffic lane lines and middle-of-lane dashed line, the simulation proceeds without flicker.

The *Simulation* class determines when and what is displayed in the *TrafficLightUI* class. The *Update* method of *TrafficLightUI* is registered with the *simulation* object held by the UI class. This is a classic use of the Observer Pattern. In this case the UI class is updated after each clock tick (every 0.1 seconds by default). The current position of each car is depicted to scale on the UI panel that is used to support

Simulation Results

Some simulation results are shown below. The grid operates much more efficiently when each light is controlled independently using the traffic control algorithm designed by the author. Simulation statistics are shown for various fixed light cycles and the variable cycle. Clearly the variable cycle automatic control is significantly more effective.

60 second cycle

Time: 600 (10 min 0 sec)

EW Green: 30 seconds EW Red: 30 seconds

EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2

WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1

NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1

SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05

EW/WE Cars Processed: 376

EW/WE Av Wait Time: 152.87

NS/SN Cars Processed: 386

NS/SN Av Wait Time: 90.97

Lights Stopped At (EW): 2.46

Lights Stopped At (NS): 1.56

Overall wait time for car in system: 121.51

Time: 600 (10 min 0 sec)

EW Green: 35 seconds EW Red: 25 seconds

EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2

WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1

NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 424
EW/WE Av Wait Time: 137.29
NS/SN Cars Processed: 430
NS/SN Av Wait Time: 100.96
Lights Stopped At (EW): 2.15
Lights Stopped At (NS): 1.74
Overall wait time for car in system: 119.00

Time: 600 (10 min 0 sec)
EW Green: 40 seconds EW Red: 20 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 466
EW/WE Av Wait Time: 113.23
NS/SN Cars Processed: 327
NS/SN Av Wait Time: 140.55
Lights Stopped At (EW): 1.66
Lights Stopped At (NS): 2.43
Overall wait time for car in system: 124.50

Time: 600 (10 min 0 sec)
EW Green: 45 seconds EW Red: 15 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 471
EW/WE Av Wait Time: 103.74
NS/SN Cars Processed: 315
NS/SN Av Wait Time: 164.51

Lights Stopped At (EW): 1.43
Lights Stopped At (NS): 2.88
Overall wait time for car in system: 128.09

Time: 600 (10 min 0 sec)
EW Green: 50 seconds EW Red: 10 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 491
EW/WE Av Wait Time: 96.97
NS/SN Cars Processed: 299
NS/SN Av Wait Time: 180.75
Lights Stopped At (EW): 1.26
Lights Stopped At (NS): 2.97
Overall wait time for car in system: 128.68

Time: 600 (10 min 0 sec)
EW Green: 50 seconds EW Red: 10 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 491
EW/WE Av Wait Time: 96.97
NS/SN Cars Processed: 299
NS/SN Av Wait Time: 180.75
Lights Stopped At (EW): 1.26
Lights Stopped At (NS): 2.97
Overall wait time for car in system: 128.68

40 second cycle

Time: 600 (10 min 0 sec)
EW Green: 20 seconds EW Red: 20 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 386
EW/WE Av Wait Time: 157.04
NS/SN Cars Processed: 369
NS/SN Av Wait Time: 93.90
Lights Stopped At (EW): 3.55
Lights Stopped At (NS): 2.18
Overall wait time for car in system: 126.18

Time: 600 (10 min 0 sec)
EW Green: 25 seconds EW Red: 15 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 435
EW/WE Av Wait Time: 116.00
NS/SN Cars Processed: 354
NS/SN Av Wait Time: 114.09
Lights Stopped At (EW): 2.48
Lights Stopped At (NS): 2.77
Overall wait time for car in system: 115.14

Time: 600 (10 min 0 sec)
EW Green: 30 seconds EW Red: 10 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1

NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 427
EW/WE Av Wait Time: 105.62
NS/SN Cars Processed: 355
NS/SN Av Wait Time: 119.80
Lights Stopped At (EW): 2.18
Lights Stopped At (NS): 2.90
Overall wait time for car in system: 112.05

30 second cycle

Time: 600 (10 min 0 sec)
EW Green: 15 seconds EW Red: 15 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 536
EW/WE Av Wait Time: 157.50
NS/SN Cars Processed: 1015
NS/SN Av Wait Time: 90.32
Lights Stopped At (EW): 4.71
Lights Stopped At (NS): 2.70
Overall wait time for car in system: 113.54

Time: 600 (10 min 0 sec)
EW Green: 20 seconds EW Red: 10 seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 550
EW/WE Av Wait Time: 125.14

NS/SN Cars Processed: 964
NS/SN Av Wait Time: 106.95
Lights Stopped At (EW): 3.44
Lights Stopped At (NS): 2.94
Overall wait time for car in system: 113.55

Automatic Signal Control

Time: 5798 (96 min 38 sec)
EW Green: Automatic seconds EW Red: Automatic seconds
EW Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.2
WE Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
NS Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.1
SN Av Arrival Rate: System.Windows.Forms.TextBox, Text: 0.05
EW/WE Cars Processed: 5177
EW/WE Av Wait Time: 93.75
NS/SN Cars Processed: 4380
NS/SN Av Wait Time: 73.53
Lights Stopped At (EW): 1.34
Lights Stopped At (NS): 1.84
Overall wait time for car in system: 84.48

About the author

Richard Wiener is Associate Professor of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 22 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, just published by Thompson, Course Technology in April 2006, is entitled *Modern Software Development Using C#/.NET*.