The Design and Implementation of an Object-Oriented Process Control Loop Framework

Taewoong Jeon[1], Sungwhan Roh[1], Hyonwoo Seung[2], and Sungyoung Lee[3]

[1] Dept. Computer Science, Korea University
{jeon, shroh}@selab.korea.ac.kr
208 Seochang-Dong, Chochiwon, Choongnam, Korea, 339-700

[2] Dept. Computer Science, Seoul Women's University, hwseung@swu.ac.kr
[3] Dept. Computer Engineering, Kyunghee University, sylee@oslab.kyunghee.ac.kr

Abstract. Control loop is an essential part of the process control system that must control physical processes in which it is difficult or impossible to compute correct output value with input values alone. In this paper, we describe the design and implementation of a highly reusable object-oriented control loop framework to support the efficient development of real time process control applications. The basic building block in our control loop framework is the Point class. The Point class encapsulates process variables of a control loop together with control algorithms so that it can be easily adapted and extended to process control applications that have various structures and behaviors. The core of this paper is the design pattern of event/time-triggered Point class that can be used for flexible implementation of monitor and control functions required of target processes through the interaction of point objects performing continuous re-computation.

1. Introduction

The process control system produces the desired output from the input resources of a process responding adequately to the constantly changing environment, or continuously monitors and controls a process in order to maintain required relations among the objects in the process. In such a system, it is difficult or impossible to compute the correct output value with input values alone, and the time constraint is usually accompanied. A control loop is an essential part of the process control system.[1]

An object-oriented framework provides an architecture that can be commonly used for the development of application programs that belong to a specific area or functional group. The architecture includes classes that can be easily adapted and extended. Accordingly, a complete application system can be easily built by modifying some of the classes in a framework and appending additional functions to the framework, then connecting them to the building blocks of the framework.[2]

On the other hand, much research has been done recently for the development of real-time systems based on the object-oriented methodology. S. Faulk et al. in [3],

tried to make requirement specifications in the real-time software development mathematically more rigid and easier to share, by putting object-orientation, graphic representation and standardized approaches together. B. Selic et al., in [4], provided more accurate and simple system modeling strategies by graphically representing the real-time system architecture through object-oriented methodologies. The system model, practicable at all the levels of abstraction, helps find out requirement or design drawbacks at the early stage of development.

While the development of general real-time systems is treated in [3] and [4], this paper focuses on the development of a process control framework with higher reusability. That is, the purpose of this paper is to make development of a process control system more efficient and easier by studying the methods to design common parts of process control application programs as an object-oriented framework with higher reusability and flexibility. In such a framework, the structure of a control loop can not only be modified dynamically during the execution of an application, but it can also be easily extended to a single control loop with many process variables or a complicated control loop application with many single control loops. The core of this paper is the design pattern of the Point class which flexibly supports the continuous re-computation of process variables of a control loop. A control loop framework is composed of frozen parts and hot spots. The frozen parts implement common functions of various control loops and can be used without modification during an application development. The hot spots represent the variability between control loops. A control loop application can be completed by adapting and extending the hot spots of the framework according to the system requirements, and then compounding them to the frozen parts of the framework.

This paper is organized as follows. Section 2 analyzes the domain of the control loop framework to be developed, explains the control loop model, and reviews requirements to be considered in order to develop a highly reusable and flexible framework. In Section 3, we propose a control loop framework designed to be easily adapted and extended to process control application systems with various structures and behaviors to meet the requirements presented in Section 2. In Section 4, an example of implementing a control loop application using the proposed control loop framework is explained. Section 5 reviews previous researches on the process control software designed as an object-oriented framework. Section 6 presents our conclusion and future work.

2. Domain Analysis of the Control Loop System

2.1 Control Loop Model

A process control system consists of a process and a control which controls and monitors the process. The current status of the process can be represented by process variables. There are four types of process variables: controlled variable, input variable, manipulated variable and reference variable. A controlled variable represents an actual measured value of an object to control. An input variable, not an object to

control, is a process variable which represents an input value needed to control the process. Both variables are used as process variables monitored by the control. That is, they are monitored variables of the process. A manipulated variable can be directly modified by the control. A reference variable represents a setpoint, that is, a target value of the controlled variable.[1]

Since the external environment of a process control system is indeterministic, unstable and constantly fluctuating regardless of the process, it is difficult or impossible to determine the correct output value with values of input and internal state alone. Therefore, in order to meet or maintain the requirements of the process outputs specified by the reference variables, the control computes the values of manipulated variables based on the reference and monitored variables and sends them to the process. The process, in turn, controls the monitored variables according to the values of the manipulated variables entered from the control. Such a process control system can be regarded as a control loop model.[Figure 1] A control loop model is composed of two components, a control and a process, and two connectors which provide paths of the asymmetric and cyclic flow of data.

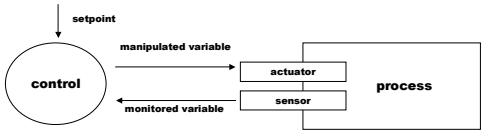

Fig. 1. The Control Loop Model

2.2 Requirements of the Control Loop Framework

Following are the requirements considered in this paper to enhance reusability and flexibility of the control loop framework.

- By adapting and extending hot spots of the control loop framework, it should be possible to easily build up a control loop application which implements interactions of process variables and control algorithms required in a specific system. It should be also possible to dynamically modify the configuration of the control loop during execution of the application.
- It should be possible to extend to a complex process control application which includes many control loops, not just one.
- The control loop application should be both time-triggered and event-triggered.
-Simulations of the control loop application should be possible, and once they are done, the application should be easily migrated to the real process control environment.
- If the user of the framework wants alarming or watching function, it should be added with ease.

3. Design of the Control Loop Framework

3.1 Point Class

The control loop controls the process through an actuator so that the state of the process can be kept within the setpoint, and monitors the state of the control through a sensor. The control loop system timely reacts to the value changes of the monitored variables entered from the sensor and recomputes the values of the manipulated variables. Then, it sends the result to the actuator to change the values of the controlled variables of the process. The control loop framework proposed in this paper is designed to have the Point class as its basic building block which encapsulates process variables together with control algorithms in order to flexibly constitute the interrelationship among the process variables.

A point can either be an input point or a computed point according to the way its value is determined. While the value of an input point comes from an external data source, a computed point gets its value from the other points inside the system. That is, a computed point depends on input points or other computed points to determine its value, and registers itself as a destination point(toPoints) to those points. If the value of a point changes, it notifies all the registered toPoints of the change. A toPoints, then, reads the values of all the source points(fromPoints) to recompute its value using its own formula. Continuous re-computation is performed as the change of point values is propagated to all the toPoints connected through the dependency chain. Figure 2 shows the relation between a point P and fromPoints/toPoints. The point P becomes toPoints1 from the viewpoint of fromPoints1, and fromPoints2 from the viewpoint of toPoints2.

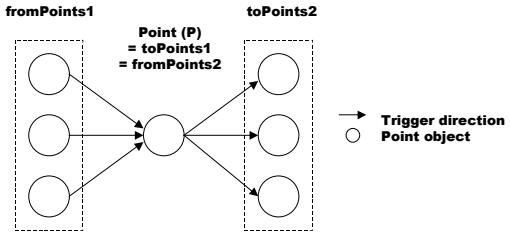

Fig. 2. Relation between the Point P and fromPoints/toPoints

Loose coupling among the objects must be maintained in order to easily adapt and extend the process control framework to a process control application system that has various structures and behaviors. The point objects can be loosely coupled by representing control relationships among them using the Observer Pattern and the Composite Pattern[5] in design.

Figure 3 shows the design pattern of the Point class and depicts the relationship among the Point class and its subclasses, InputPoint and ComputedPoint class. The ComputedPoint class responds to the value changes of the process variables in the Point class whose destination point(toPoints) is the Computed Point. It plays the role of observers. The ComputedPoint also creates a Composite pattern in which it has other point objects as its components through the fromPoints reference. The

InputPoint gets its value directly from the outside through the setValue member function, while the ComputedPoint computes its value using the values of the other points affecting it. Therefore, the fromPoints reference in Figure 2 can be either an InputPoint object or a ComputedPoint object, but the toPoints reference can be a ComputedPoint object only. In Figure 4, when a value is set to anInputPoint, its notifyPoints member function is called, which, in turn, calls the update functions of all the toPoints of the InputPoint. After the update function of the invoked aComputedPoint1 re-computes its point value through the formula class(aFormula1), it calls the update functions of its toPoints(aComputedPoint2). Since values of the monitored variables entered from the sensor end up with starting the actuator through the chain reaction of the point re-computations and update calls, an expected control effect can be achieved.

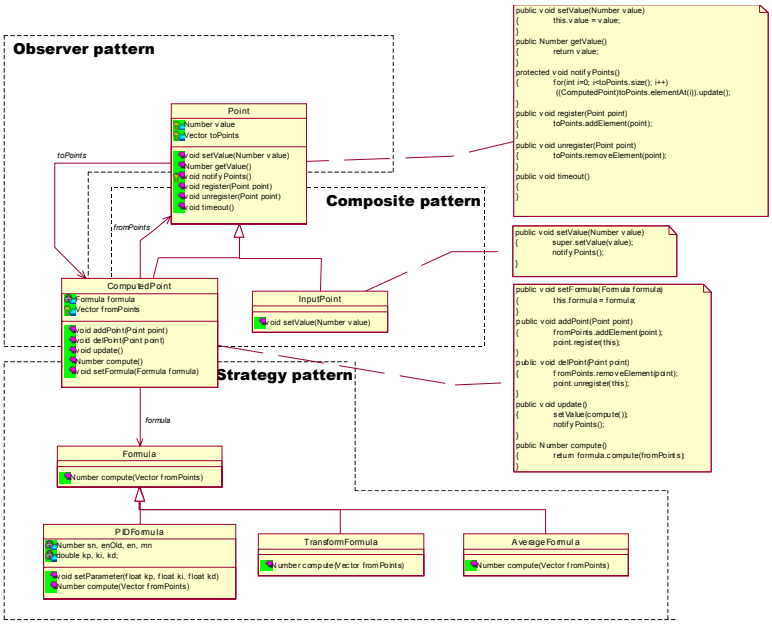

Fig. 3. The Point Class

3.2 Formula Class

The manipulated variables, the monitored variables and the actuator can be regarded as computed points. Their structures are the same, but the behaviors of their formula functions are different. So, we applied the Strategy pattern[5] to encapsulate the formula in a separate class in designing a computed point.

The computation algorithm of a computed point is determined by a Formula class object which is dynamically replaceable according to the Strategy pattern. The computed point class has a formula which is a reference to the Formula class object. The Formula class provides a compute member function which is a common interface to the various algorithms a computed point may have. Using the compute function, a

computed point invokes an algorithm defined in a subclass of the Formula class. If a computed point object represents a manipulated variable using the PID control algorithm, it references to the PID Formula. If a computed point object represents a monitored variable that contains an average value of many sensors, it references to the Average Formula class object. If a computed point object represents an actuator, it references to the Transform Formula. The compute member function of the PID Formula class contains PID control algorithm, and the setParameter member function reads parameter values required for the PID control algorithm when the program starts. By making the reconfiguration of parameters possible, the system can be easily modified without rebuilding the program.

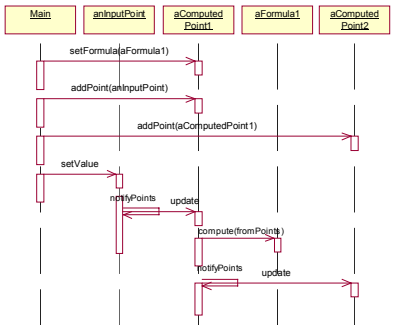

Fig. 4. Interactions of Point Objects

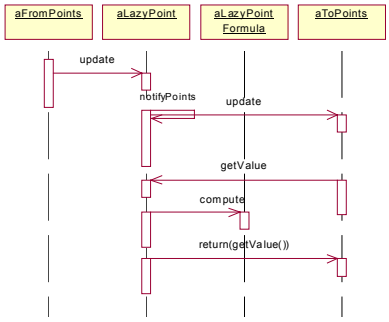

Fig. 5. Interaction of the LazyPoint

3.3 Lazy Point Class

If a value of a certain computed point is used infrequently, it is a waste of time to re-compute its value every time the values of points it is dependent on change. It is more efficient to re-compute when the value is requested. In order to meet such a requirement, a LazyPoint class[8] is incorporated in this paper as a subclass of the ComputedPoint class. A LazyPoint re-computes its value only when it is used by another point, while a ComputedPoint is re-computed whenever the values of fromPoints change.

As shown in Figure 5, a LazyPoint simply records an update and delays the re-computation of its value until it is requested, while a ComputedPoint re-computes its value every time its update function is called, and notifies the registered points of the change. Only when a record of an update exists, its value is re-computed, otherwise, the stored point value is sent. So, time waste for the unnecessary point re-computation can be avoided.

3.4 Time Triggered Point Class

In the control loop system, not only the event triggered approach, in which the system reacts to the state change of the process, but also the time triggered approach must be utilized, so that the system can reach or maintain the setpoint value of the process within a required time by monitoring the control state of the process at specific points in time regardless of the state change of the process. A Timer class is defined in the system to meet such requirement.

As explained above, the manipulated variable point of the control loop computes its value based on the values of the setpoint and the monitored variable, and sends the value to the actuator. If the value of the monitored variable does not reach the setpoint value after a specific period of time, it might be necessary to raise an alarm condition. To represent such a manipulated variable, the ManipulatedVariablePoint class which needs a timer is provided as a subclass of the ComputedPoint class. General manipulated variables which do not need alarming function are created from the ComputedPoint class.

The TimeTriggeredSensor is periodically activated by the Timer, and the EventTriggeredSensor is activated in case the value of the sensor changes. Both sensors are created by inheriting from the InputPoint. The Point objects that need a timer such as the time triggered sensors, manipulated variables with alarming functions, or the Simulation Point objects register themselves and the duration of time to the Timer. The Timer, at specific intervals of time, invokes the timeout member function in the registered Point object and notifies the registered time has passed. The manipulated variable whose registered time has passed checks if the monitored variable has reached the setpoint, and raises an alarm condition if it has not.

3.5 Simulated Process

The SimulationPoint class represents a simulated process by simulating the interaction between an actuator and a sensor. The effect caused by the actuator is sent to the SimulationPoint object, which, in turn, transfers the simulated value based on the effect to the sensor after a certain feedback delay time. Therefore, when an application which is made out of a framework is to run under the actual operational environment, the SimulationPoint is only to be replaced by the actual process. The starting point of the SimulationPoint is the actuator, and the destination point is the sensor. Once the actuator object invokes update function of the SimulationPoint object, the SimulationPoint object computes a simulated value through the SimulationFormula object it references to. When the Timer object of the SimulationPoint invokes timeout function after the feedback delay time passes, the simulated value is sent to the sensor.

3.6 Architecture of Entire Classes of the Control Loop Framework

Figure 6 shows the inheritance and composition relationship among the classes of the control loop framework as a whole. Since the fundamental concept of the framework

design in this paper is the continuous re-computation of point values, the various classes that consist of the control loop framework are inherited from the Point class.

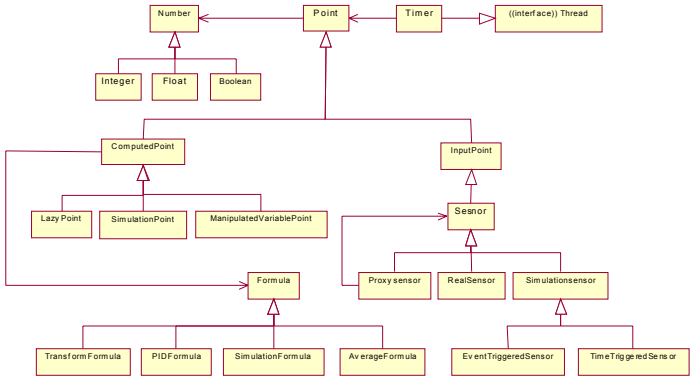

Fig. 6. The Control Loop Framework

4. An Application using the Control Loop Framework

Using the control loop framework proposed in this paper, it is possible to design a flexible control loop software component based on the control loop model in Figure 1. To implement a control loop application, the control loop system developer using the framework can build the classes his/her application requires by inheriting and extending the Point class or its subclasses. This section explains a control loop application implemented using the Java programming language.

Figure 7 shows a heat control loop model which controls room temperature. The control is a heat control to maintain the desired room temperature as a setpoint, and the process is a heater monitored and controlled by the heat control. The monitored variable represents the current room temperature taken by the sensor. The temperature is a feedback value to the control and used to compute the value of the manipulated variable, which, in turn, is sent to the actuator.

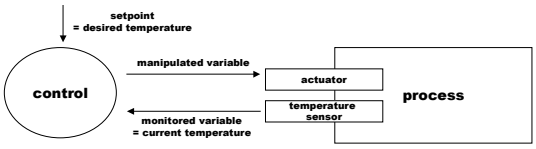

Fig. 7. The Heat Control Loop

Figure 8 shows a configuration of the heat control loop in Figure 15 constructed using the Point classes in the control loop framework. The arrows represents trigger directions between the connected Point objects. The setpoint and the sensor are InputPoint objects of the framework. The manipulated variable, monitored variable, actuator and the simulation point are all ComputedPoint objects, and each references to its own Formula object.

The control mechanism of the heat control loop system is as follows: The sensor is triggered by the sensor timer at specific periods of time, and sends the measured

temperature to the monitored variable. The manipulated variable takes the monitored value from the monitored variable and computes the manipulated value using the PID control algorithm of the PID formula. The computed manipulated value is sent to the actuator. Then, the value is reflected on the simulated process. The manipulated variable timer notifies the manipulated variable that a certain period of time has elapsed. Then, the manipulated variable, with the monitored value and the setpoint value, judges whether the control is in the normal state or not.

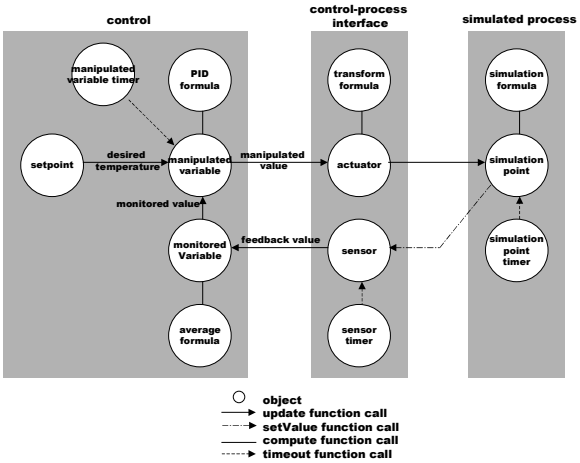

Fig. 8. The Heat Control Application

The value of the actuator is sent to th simulation point. As soon as the simulation timer notifies the simulation point that the feedback delay time has elapsed, the simulation point sends the current temperature to the sensor. The value of the sensor is, in turn, sent to the monitored variable to make a loop.

Figure 7 is a single control loop system where there is only one point object for each of the actuator, sensor, monitored variable and manipulated variable. There exists a single control loop system with many process variables, or a control loop system with many single control loops. Such complicated control loop systems can also be easily implemented using the control loop framework.

Figure 9 shows a control loop system which consists of two single control loops. An application of such a system can be constructed using Point classes of the control loop framework as shown in Figure 10.

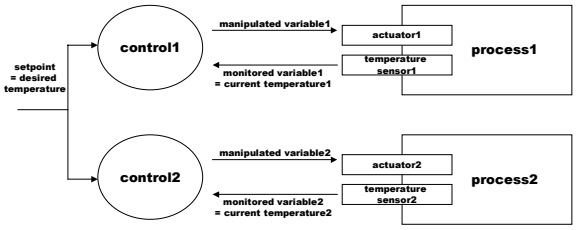

Fig. 9. A Control Loop System with Two Single Control Loops

5. Related Works

Current researches on the design of the process control software in an object-oriented framework are reviewed in this chapter.

Per Dagermo and Jonas Knutsson[8] regarded the process control as a continuous re-computation of a number of output values as the response to changes to a number of input values. A value in [8], which is an abstraction of a process variable, can either be an input value or a computed value. When a value changes, it notifies all the computed values registered as its dependents. When a dependent value is notified, it re-computes itself.

P. Molin and L. Ohlsson[9] designed a framework for fire alarm systems, where the logical behavior of the input/output devices such as sensors and actuators at the interface are standardized and defined as Points [9]. The concept of Point was applied in this paper as the Point class.

Jan Bosch[10] designed an object-oriented framework for the measurement system which measures the quality of manufactured products and picks out low-quality products. The Strategy pattern and the Composite Pattern were mainly utilized in the design of the system. The control loop of the measurement system differs from the one proposed in this paper in that the measured values are not used directly to control the manufacturing process. In this paper, the measured value, that is, the monitored variable is used as the controlled variable to control the process.

The process variables as well as sensors and actuators in this paper are all defined as Point classes or its subclasses performing continuous re-computation. While data types, computation algorithm, and control of the computation are all encapsulated in the Value class in [8], they are separated in different classes in this paper. As the data type of a process variable is separated from its control and defined in a Number class, the operations of the process variable are performed polymorphically. This allows the control loop to have multiple types for its point values. We defined computation algorithms in separate classes according to the Strategy pattern so that the algorithms used by computed points are easily replaceable at runtime.

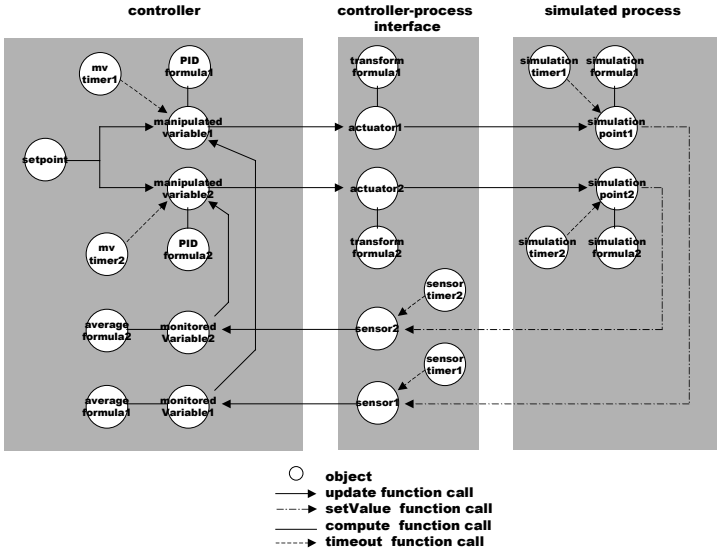

Fig. 10. An Application of the System in Figure 9

6. Conclusions and Future Work

The design and implementation of a highly reusable object-oriented control loop framework for process control systems was proposed in this paper. The core of the control loop framework is the design pattern of the Point class which encapsulates state values of the process together with the control mechanism. Since the Point classes are designed using the Observer and Composite patterns, the Point objects are loosely coupled one another and can be easily reconfigured. Furthermore, connection and disconnection of objects can also be easily done during the execution. Therefore, control loop application developers using the framework can complete their applications more efficiently and flexibly by adapting and extending the Point classes to their requirements and connect them to the framework.

Most of the process control systems require parallel and real-time processing. Although the control loop framework proposed in this paper utilizes Java's multi-thread function for the parallel processing, it does not directly support real-time scheduling. It assumes a real-time kernel to be used when it is extended to an application. Enhancing to a control loop framework that can be easily extended to a specific real-time operating system environment is the primary work to be done in the future.

References

[1] Mary Shaw, "Beyond Objects: A Software Design Paradigm Based on Process Control", ACM Software Engineering Notes, Vol 20, No 1, Jan, 1995

[2] Gregory F. Rogers, "Framework-Based Software Development in C++", Prentice Hall PTR, 1997

[3] S.Faulk, J. Brackett, P.Ward, and J.Kirby Jr, "The Core Method for Real-Time Requirements", IEEE Software, pp 22-23, Sept. 1992.

[4] B.Selic, G.Gullekson, and, P.T. Ward, "Real-Time Object-Oriented Modeling", John Wiley and Sons, 1994.

[5] Erich Gamma and et al, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Publishing Company, 1995.

[6] Stuart Bennett, "Real-time Computer Control: An Introduction", 2nd Edition, Prentice Hall International (UK) Limited, 1994.

[7] Bobby Woolf, "The Abstract Class Pattern"

[8] Per Dagermo, Jonas Knutsson, "Development of an Object-Oriented Framework for Vessel Control Systems", Technical Report, Dover Consortium 1996.

[9] P.Molin and L. Ohlsson, "Points & Deviations - A Pattern Language for Fire Alarm Systems", Pattern Languages of Program Design 3, Addison-Wesley Publishing Company, 1995

[10] Jan Bosch, "Design of Object-Oriented Framework for Measurement Systems"

[11] Ken Arnold, James Gosling, and David Holmes, The Java Programming Language, 3rd edition, Addison-Wesley, 2000