REAL-TIME PROCESS CONTROL WITH CONCURRENT JAVA

Webjørn Rekdalsbakken[1], Arne Styve[1]

[1] Aalesund University College, Institute of Technology and Nautical Science, N-6025 Aalesund, Norway.

wr@hials.no (Webjørn Rekdalsbakken)

Abstract

This paper presents examples from a student course on Bachelor level in real-time programming and dynamic process control. A main part of the course comprises a project with the goal of designing a control system for a dynamic process. The chosen process may be either a real physical system or a simulated model. Three examples of these projects with different applications of real-time control of dynamical systems are presented in the paper. The design and programming have been performed with the use of the Java Concurrency Model (JCM) and with the aid of the Real-Time Specification for Java (RTSJ). Dynamic process control in real-time is considered as a complicated field in engineering education, especially at the Bachelor level. This paper intends to show that with appropriate simulation tools and programming languages it is fully possible to achieve satisfactory results. The three examples presented here comprise the balancing of a flexible inverted pendulum by an autonomous vehicle, the direction control of a ship search light, and the control of a heave compensated winch over a distributed network. The projects have been chosen in cooperation with local industry partners and have been accomplished as student projects in groups of 2-3 students under surveillance of the teachers responsible for the real-time course. The course was originally taught using C/C++ and a real-time kernel, but has now been completed twice for about twenty students using the JCM. The experience is that in almost all cases the project groups succeed in building a complete control system.

Keywords: The Java Concurrency Model, embedded process control, distributed real-time simulation.

Presenting Author's biography

Webjørn Rekdalsbakken is MSc. in Physics from the Norwegian Institute of Technology. He has been head of the med. tech. dept. at a general hospital, and a senior engineer in process control at Hydro Aluminium. He has been Assistant Professor in computer science at Aalesund University College (AUC), and also rector at AUC. Now he is Assistant Professor/program leader of the Cybernetics program at AUC.
He has been doing research on nautical simulators and published papers on the dynamic control of motion platforms. He is a member of the Norwegian Automation Society.

ISBN 978-3-901608-32-2 *Copyright © 2007 EUROSIM / SLOSIM*

1 General

1.1 The real-time course

Third year bachelor students in cybernetics have a mandatory course in real-time programming comprising 15 ECTS credits. The course gives an introduction to concurrent programming and real-time principles. Half of the course is theory with exercises in real-time programming, the other half consists of a project where the students are required to design and implement a complete control system for a dynamic process. The programming tool is concurrent Java and implementation is done on a PC or an embedded microcontroller. The dynamic processes to be controlled have been chosen from realistic situations in local industry, mainly maritime related companies. AUC has specialized in the building of nautical simulators and many small scale models of motion platforms and manipulator arms have been built in the laboratory. Different kinds of control strategies have been tested on these models. Also, models of dynamical systems have been simulated i Java. A special communication bus, called Common Simulation Interface (CSI) has been utilized to integrate distributed processes through a network. An important objective has been to organize the projects so that they reflect a complete development cycle for a control system. In the solution, both the real-time aspect of the problem and the control strategy shall be considered. This will include such topics as process communication, driver software, independent tasks, task priority, control methods, and user interface. The student groups are required to work as independently as possible, however, with support from the teachers whenever necessary. This is a demanding situation for the students, which challenges both their skills and their personal qualities. This paper presents a survey of selected student projects from this real-time course. The projects have been chosen to give a representative overview of the course.

1.2 The Java Concurrency Model

The JCM allows program threads to run concurrently and independently. The Thread class has a *start()* method that is called once for each thread and a *run()* method that executes the thread content. Each object in Java has a *mutual exclusion* lock. The central device is the *synchronized* method modifier. A method or a block of code that is labelled with the *synchronized* modifier, can only be accessed after the corresponding object lock has been obtained. Other objects that try to get access to the *synchronized* method, will enter the *locked* state and will have to wait for the access. The methods *wait()* and *notify()* have opposite functions; *wait()* is used by the current thread to temporarily release the lock of the object and enter a wait state. Access to the corresponding synchronized method is thus allowed. *notify()* signals the waiting objects that access is again permitted. In this way *wait()* and *notify()* constitute the basic mechanism for the synchronization of threads in Java. The programmer must, however, implement the concept of *conditional variables* to discriminate between the different synchronized methods. With these variables it will be possible for a thread to identify the resource (object) it is waiting for. In addition to this mechanism a relative time wait is also implemented with the method *sleep()*. This scheme is the basis of the JCM. By these mechanisms all common real-time devices can be implemented, like semaphores, signals, event flags, blackboards etc. As a help in secure implementation of these concurrency utilities, they have been introduced into the release of Java 1.5 in classes of the package *java.util.concurrent* and subclasses to these. For information on the JCM, see [1,2,3].

1.3 The RTSJ specification

While the JCM may give an adequate platform for parallel programming, it does not specifically meet the requirements of real-time programming. The RTSJ is developed on basis of the requirements for Real-Time Java (RTJ) from the US National Institute of Standards and Technology (NIST). It concerns the implementation requirements of the actual Java Virtual Machine (JVM) and gives specifications for the use of the concurrency model. The crucial point is the determinism of the real-time system. In short RTSJ deals with timing requirements for the different actions and the priorities of the schedulable objects. In addition it contains specifications for asynchronous event handling and transfer of control. These properties are provided as special classes in RTSJ. However, the mechanisms behind are buried in the interaction between system software and the hardware. For a specific operating system and hardware an application programmer often has little influence on these matters aside from following the recommendations for the use of the concurrency model. For more information on RTSJ, see [1,4,5].

2 The inverted pendulum

2.1 Hardware design

This project represents an original solution to the classical problem of balancing an inverted pendulum. A dedicated autonomous vehicle was build with basis in a 1:10 model of a Ferrari car. The chassis and motor were kept intact and supplied with measuring devices for car position and pendulum angle. In addition a motor control system was designed. The vehicle was supplied with a Javelin Stamp microcontroller equipped with the necessary peripherals to control the vehicle operation. The purpose of the vehicle is to keep the pendulum upright at a given vehicle reference position. A collage of the vehicle and some of its components is shown in Fig. 1. All the necessary electronic circuits for the vehicle operation were assembled on a common board. The board is shown in Fig. 2.

 Copyright © 2007 EUROSIM / SLOSIM

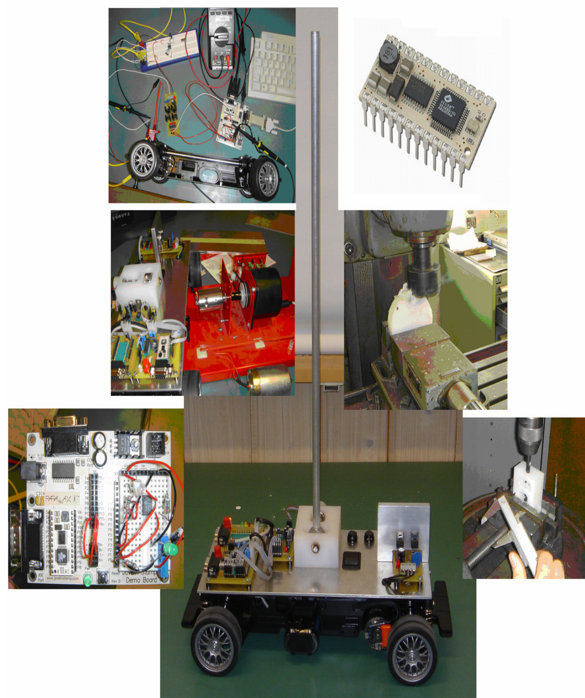

Fig. 1 The pendulum vehicle

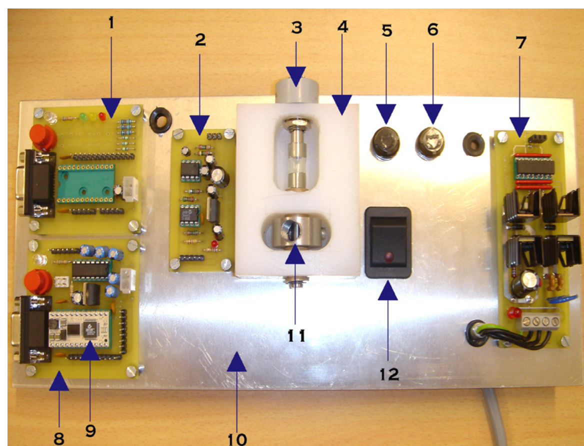

Fig. 2 The vehicle component board

2.2 The Javelin Stamp microcontroller

The Javelin Stamp is a microcontroller with a Java Virtual Machine included. The controller has 16 digital I/O-lines. The controller is also equipped with five so called Virtual Peripherals (VP), i.e. firmware that serves the hardware peripherals on the stamp controller. These peripherals include a UART, a PWM output, a 32-bit timer, a 1-bit DAC, and an 8-bit ADC. The VPs can run independently in the background of the main program in time slices of 8.68 microseconds, or they can be called directly from the foreground process. The Javelin JVM supports no garbage collection, but this is no problem in a small dedicated application, rather a benefit for the determinism of the system. The Javelin Stamp is supported by an Integrated Development Environment (IDE) for programming in Java on a PC. The IDE includes properties for easy downloading and debugging of the code on the stamp controller through a RS232 interface. For information on the Javelin Stamp microcontroller, see [6,7].

2.3 Measuring the state variables

To control the pendulum the necessary state variables have to be measured. In this case the state variables are the pendulum angle, the angular velocity and the vehicle position. The control algorithm can easily be extended to include the integrals of the angle and the vehicle position as well. The angular velocity and the integrals must be calculated from successive measurements of the primary variables. The pendulum angle is measured by a potentiometer through an external 12-bits AD converter connected to the Javelin Stamp via a Serial Peripheral Interface (SPI). The potentiometer signal is first filtered through a third order low pass Butterworth-filter, which was realized by operational amplifiers and implemented on the electronic board. The vehicle position is measured with a modified serial mouse (Microsoft 2-button serial mouse) through the Javelin Stamp's UART. The program is based on an example found on the Parallax Internet site. The example program was much reduced and modified to fit to the embedded application.

2.4 Motor control

The motor is a DC servo motor powered by a 7.2 V battery pack. The motor is controlled by a PWM signal generated by a VP on the Javelin Stamp. To be able to change motor direction and to make the motor turn fast enough for the control of the pendulum, an H-bridge circuit was designed and implemented as a Motor Control Unit (MCM) on its own board with sufficient cooling capacity. The source of the PWM signal is the control algorithm running on the Javelin Stamp. The drawing of H-bridge is shown in Fig. 3 and the physical board is shown in Fig. 4.

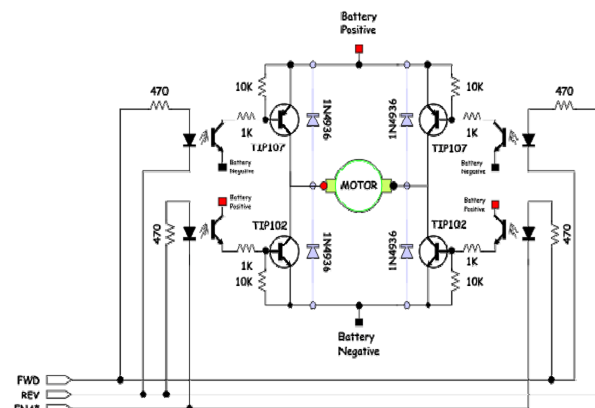

Fig. 3 Drawing of the H-bridge

 Copyright © 2007 EUROSIM / SLOSIM

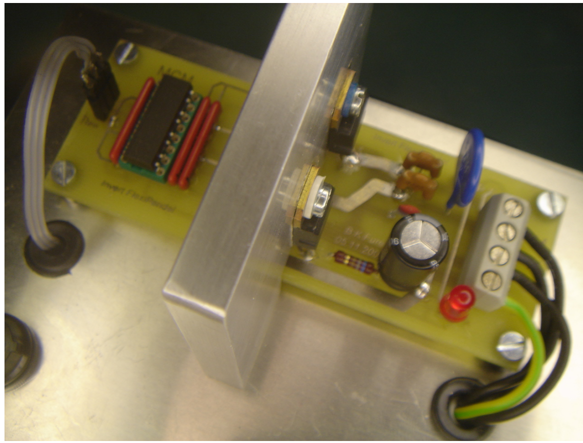

Fig. 4 Motor Control Unit board

2.5 Control strategy

The control strategy is based on a pole placement regulator using the control law. A state-space model of the pendulum is derived, and the pendulum is simulated with Simulink.[1] On the modelling and control of the inverted pendulum, see [8,9]. The necessary bandwidth of the control loop was stipulated, and the poles were located according to a Butterworth polynomial. Ackermann's formula is used to calculate the **G**-matrix. Actually several control algorithms were explored with order 3, 4 and 5 depending on the number of state variables used in the feedback. The best result was obtained with a fourth order regulator in the variables pendulum angle, angular velocity, angle integral and vehicle position. The experience was that the feedback constants of the **G**-matrix had to be systematically tuned for the real situation and could be quite far from the ones found by simulation.

2.6 Software design

The software was developed in Java with the Javelin Stamp IDE. On the Javelin Stamp only one program thread can run in the background, so there is basically no concurrency. However, with support of the peripheral firmware and the use of the timer, the VPs can operate independently of each other as service routines in the foreground. The software application therefore works as a foreground/background system with a simple form of concurrency. The program system is composed of five classes named as InvertPendel, Regulator, MCM, MCP3202, and MS2ButtonSerial. The InvertPendel class includes the *main()* method. In this class all static variables are initialized and an object of each of the other classes is created. The *main()* method then initializes the system peripherals including the timer function and enters an everlasting loop with a 100 millisecond time interval. The loop is a sequence of events that first reads the angle and position and calculates the angular velocity and the integral of the angle. Then it calculates the control signal for the motor, determining the motor speed and direction. Then the motor is started to run for a given interval. This procedure is performed by calls to methods in the four other classes. The methods of the classes MCM, MCP3202, and MS2ButtonSerial respectively control the Motor Control Unit, the angle measurement unit and the position measurements of the serial mouse. These classes perform their tasks by the appropriate services of the VPs in the Javelin Stamp firmware. The concurrency lies in the property that the VPs can perform their services independently of the running thread.

[1] Simulink Toolbox is a product of Mathworks Inc.

3 The ship search light

3.1 Hardware design

The hardware design consists of a mechanical frame for the installation of a search light on a ship. The assembly includes three AC servo motors with accompanying gears. The search light is controlled by the servos to hold and focus on an object floating on the sea. Two of the motors respectively control the horizontal (yaw) and the vertical (pitch) directions of the light beam. The third motor controls the zoom (focus) of the beam. The reference inputs to the system are given by an operator of a wireless joystick. The joystick receiver is connected to the PC via an USB port. The operator can alternatively choose to control the search light from a GUI on the PC. This option is, however, mostly used for test procedures. The mechanical assembly with motors and joystick is shown in Fig. 5.

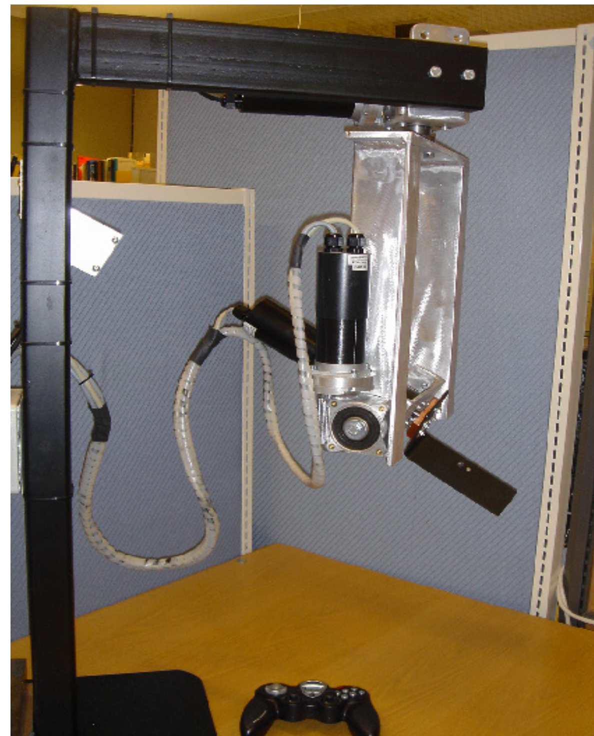

Fig. 5 Mechanical assembly of the search light

ISBN 978-3-901608-32-2

Copyright © 2007 EUROSIM / SLOSIM

3.2 Motor control

All three motors are of the type MAC140 from JVL. It is a 3-phase AC motor of 134 W and a continuous torque of 0.32 Nm. The motor has the electronics for the servo drive and communication logic embedded inside the motor housing. The power supply is optional between 12-48 VDC. In this case a 36 VDC power supply was chosen giving a maximum rotational speed of 3000 rpm. The motors are coupled to planetary gears with a transmission of 1:100, and an accuracy of 15 arc seconds. The communication with the motors is over a serial line, either RS232 or RS422 using a proprietary protocol called MacTalk. All motors are connected to the same communication line, and each motor has its own unique address. The motors can be controlled in either velocity mode or position mode over the serial line. For information on the MAC motors, see [10].

3.3 Input reference signals

The input reference signals are provided by a wireless joystick control of the type Saitek P3000. The joystick communicates with a receiver connected to a USB port on the PC. It operates on the 2.4 GHz frequency and has a reach of 10 meters. The stick on the control gives the yaw and pitch angles of the search light, and two buttons set the focus of the light beam. The Java application communicates with the joystick through the USB port with the help of a special joystick API program package. For information on the joystick API, see [11].

3.4 Software design

The application software comprises six kernel classes and two auxiliary classes. The kernel classes are named GUI, Controller, ThreadMaker, ThreadRunner, Interface and Semaphor. The GUI class contains the *main()* method. It does the initialization of the system and provides a user interface for the operator of the system. In *main()* an object of the Controller class is created. Then an object of the ThreadMaker class is created given a reference to the Controller object. The ThreadMaker object creates an object of the Interface class. It then creates three objects of the ThreadRunner class giving each a reference to itself, and it starts these threads. ThreadMaker also sets the allowed range of motion for each of the three motors. ThreadRunner is a thread class, and three objects of this class run as concurrent independent threads, each of them controlling one of the servo motors. The Controller class constitutes the interface to the joystick control. It uses an object of the auxiliary class Joystick to read the joystick input. The threads call a method *getValue()* in the Controller class to obtain the joystick values. The *getValue()* method has an argument specifying which input to be read from the joystick, and the Joystick object keeps track of these inputs. In this way there will be no concurrency problem here with an eventual deadlock between the three threads. Each of the threads calculates the new speed of the respective motor and uses an object of the Interface class to control the motor. The Interface class opens the communication with the motors through the auxiliary class MacCom with a call to the method *get_DMacComm()*. DMacComm is an ActiveX interface to the COM port. The Interface object takes input from each of the three threads and controls the position of the corresponding motor. Because there are three motors on one communication line, the Interface object has to use the Semaphore class. The Interface object creates an instance of the Semaphore class and uses it to get mutual exclusive access to the communication line.

3.5 Motor communication

Driver software may be a problem in Java. Proprietary software drivers are most commonly written in C++ and are found as DLL libraries or ActiveX components. To use these methods from Java one has to make use of the Java Native Interface (JNI) mechanism, and write wrapper classes in Java that take care of the software interface to the driver functions. This is done for the motor communication. To communicate with a COM port from Java, a software package called EZ Jcom is used to generate a wrapper class. This is done in the class Interface by creating an instance of the wrapper class MacComm. Through the MacComm class the Interface class has access to methods in a library of ActiveX components that controls the COM port. For info on EZ Jcom, see [12].

3.6 Further development

In cooperation with local industry the search light control system has been further developed with compensation for the ship movements. Most ships have installed a Motion Reference Unit (MRU). The MRU provides the measurements of the ship position in 3D space with six independent axes, consisting of three angles and the relative translation in three directions. The roll and pitch angles in addition to the heave position is used by the search light platform to compensate for the ship movements and keep the platform in a stable position. The search light beam is then aimed at the direction of the object with correction for the heave position of the platform. This is quite a comprehensive system which is very important for the security in many naval operations.

4 A heave-compensated winch

4.1 The Common Simulation Interface

This project comprises a distributed control system realized by a special network and communication framework called the Common Simulation Interface (CSI bus). The CSI bus is based on the High Level Architecture (HLA) standard and is designed for distributed simulation of complex systems. The bus is developed by the Norwegian Marine Technology Research Institute (Marintek) in cooperation with

 Copyright © 2007 EUROSIM / SLOSIM

Rolls Royce Marine Dept. and is written in Java.[2] It is used mainly for simulating marine operations, but can be used as a general system bus for distributed simulations. The CSI bus communicates by XML streams over a TCP socket connection and consists of one server and one or more clients. The server is a multithreaded Java application with one thread serving each client. Each client consists of an application called a *federate*. The federate can be written in any language supporting socket connections. In this project Java and the JCM is used. A federate consists of *objects* which contain *parameters* and *attributes* that are published to the server. An *object* can subscribe on attributes in objects in the other federates via the server. The parameters of an object can be changed by the other federates, while an attribute is read-only, and only set by the owning object. The federate configuration is given by an XML file specifying the federation of federates and their properties (attributes and parameters) and the communication details. Each federate exists as an independent process in interaction with all other federates on the CSI bus, and the system can be modified or extended with new federates as required. The server takes care of the synchronization between federates. However, the synchronization time is only relative between the federates, and not an absolute clock. Each federate has to tell the server at what time it wants to start executing again. In this way the server will know the chronology of the federates. For the system to operate in real time one federate must have access to a hardware clock to set an absolute pace of the execution. It is also possible to let a federate execute asynchronously of the other federates. For information on the CSI bus, see [13].

4.2 The winch simulation system

Heave-compensation is an important subject in many operations on board a ship. In the handling of all kinds of cargo with ship winches this problem must be considered. The principle of heave-compensation is sketched in Fig. 6. The load of the winch must be in the correct position independent of the movements of the ship. The simulated system for the heave-compensated winch consists of three federates, each of which implements a simulation model. These models represent the winch, the MRU, and the regulator. The MRU supplies signals for the ship movements, represented by the angles roll and pitch, and the heave motion, in addition to the velocities of these variables. The winch is represented by a second order model of a DC servo motor with gear. The input to the motor model comes from the regulator and the output is the length of the unwound cable. The regulator consists of a PID controlled feedback loop for the length of unwound cable, in addition to a feed forward signal from the instant vertical position of the winch derived

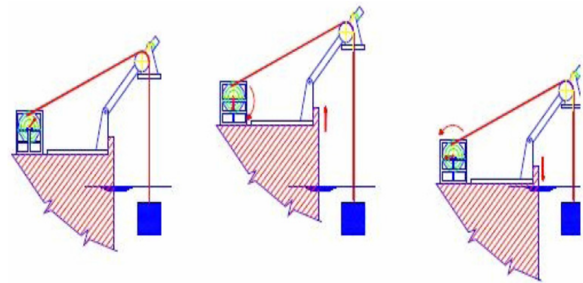

Fig. 6 The principle of heave-compensation

from the MRU signals. The output from the regulator is a speed reference signal to the winch federate.

4.3 Software design

The software was designed using the CSI Framework and the concept of federates. The winch federate has two kernel classes; SimulatedWinch and DCMotor, which run as two independent threads. In addition the federate has an auxiliary class called Integrator. SimulatedWinch uses the PIDController class that belongs to the regulator federate, but communicates with the SimulatedWinch object over the CSI bus. The PIDController provides the speed reference for the winch motor and decides when to start and stop the motor. The SimulatedWinch thread gets the actual motor speed from DCMotor and calculates the deviation from the reference value and sends the speed control signal to DCMotor. It keeps track of the length of used winch cable by help of the Integrator class. SimulatedWinch publishes the length of used cable on the CSI bus for use by the PIDController class. The DCMotor thread simulates the motor function and provides the actual motor speed after a call from the SimulatedWinch thread. DCMotor contains the methods to get and set the motor speed. These methods are used by both the threads, so they have to be identified as synchronized to avoid corruption of data.

The MRU federate has one kernel class called SimulatedMRU with an internal class called SimulationTask and the auxiliary class MRUWindow. SimulatedMRU contains the methods and attributes necessary to provide the ship motion variables for publication on the CSI bus. SimulationTask is a subclass to the Java TimerTask and is run as an independent thread. An instance of SimulationTask is used to establish a timer for the system, and the *run()* method of this thread publishes the MRU data on CSI bus at regular time intervals. Thus the MRU federate is responsible for the system real time clock. The MRUWindow class is run as a thread and provides for the GUI of the federate. MRUWindow starts the federate by reading initial values from the operator and creating the SimulatedMRU object.

The regulator federate consists of six kernel classes and two auxiliary classes. The Controller is the central class; it controls and supervises the regulation of the winch with the use of the other classes. It implements

[2] The CSI bus is a proprietary product of Marintek

the *Runnable* interface to be run as an independent thread. When an object of this class is created, all regulator and winch parameters are initialized for the given configuration. The other kernel classes are PIDController, CSICommunicator, ControllerData, PositionCalculator, and MissingAttributeExeption. CSICommunicator and ControllerData take care of the communication with the CSI bus and provide the data from the MRU. PositionCalculator uses the MRU data to calculate the correct position of the winch. PIDController uses the cable reference value and the winch position to calculate the speed reference value. This value is published on the CSI bus. The MissingAttributeExeption class contains an exception thrown by CSICommunicator if the attributes from the CSI bus are not available. The auxiliary classes MainWindow and PlotPanel represent the GUI of the federate. PlotPanel is run as an independent thread.

4.4 Further development

The CSI bus is an excellent platform for the simulation of complex systems. The federates run as independent systems, and new federates for extended functionality can be added when needed. At AUC ship simulators and visualizing systems have been developed based on, and using the CSI bus. The simulator environment is continually upgraded and extended, and the existing federates are available for interaction and testing of new models. This gives a unique opportunity to simulate and test expensive and complex equipment before the hardware design, and when modifications are going to take place. The heave-compensated winch is one of the systems tested in this environment. Based on these simulations a physical test model has been built, and the technology is already implemented in products delivered by a local company.

5 Discussion and conclusion

The experience as responsible instructors for these projects is that real-time process control is a demanding challenge for bachelor students. The real-time course probably represents the professional peak of this education and pushes the students' intellectual limit. However, motivated and determined students need challenges. It is important that a part of their education touches the real situation in their future professional career. The use of the Java platform has been successful. With the JCM and the RTSJ, the students can use a standard, open, freely available, and very widely used programming language. Since the concurrency model is a part of the standard Java SDK, no third-party libraries may be needed. The course has also shown that very much can be gained through teamwork and by good cooperation between teachers and students. The motivation lies primarily in the feeling of mastering a developmental process; the students realize that they can design a real, quite complex control system. By building physical models of real equipment and simulate realistic situations, they get the feeling of doing proper and important work. They also know that they are working with modern tools and methods that will affect and maybe decide their professional situation after having finished their education. The accomplishment of these quite demanding projects shows that the mature bachelor students can master an in-depth course like real-time programming.

6 References

[1] A. Wellings. Concurrent and Real-Time Programming in Java. John Wiley & Sons, Ltd. England. 2004. ISBN 0-470-84437-X.

[2] J. Magee, J. Kramer. Concurrency. State models and Java programming. Wiley & Sons, Ltd. England. 2006. ISBN-13: 978-0-470-09356-6.

[3] B. Goetz. Java Concurrency in Practice. Addison-Wesley. 2006 Pearson Education, Inc. ISBN 0-321-34960-1.

[4] http://jcp.org/en/jsr

[5] http://www.rtsj.org

[6] Parallax. Javelin Stamp user's manual. Version 1.0. Parallax Inc., 2002.

[7] http://www.parallax.com/Javelin

[8] K. Ogata. Designing Linear Control Systems with Matlab. Matlab Curriculum Series. Prentice Hall. 1994.

[9] W. Rekdalsbakken. Feedback Control of an Inverted pendulum with the use of Artificial Intelligence. In *Proceedings of ICCC 2006, IEEE International Conference on Computational Cybernetics*, pp. 75-80, IEEE Catalog Number of Printed Proceedings: 06EX1270.

[10] http://www.jvl.dk

[11] http://www.sourceforge.net

[12] http://www.ezjcom.com

[13] N. Husteli. Common Simulation Interface. Documentation and Tutorial. Marintek Report, Trondheim, Norway, 2005.

ISBN 978-3-901608-32-2

Copyright © 2007 EUROSIM / SLOSIM