2005

Real-Time Control System Framework

Nishant Thakkar

Computer Science Capstone Thesis

Real-Time Control System Framework

Author	Nishant Thakkar	Computer Science
Advisor	James Vallino	Software Engineering
Committee Chairman	James Vallino	Software Engineering
Reader	Leonid Reznik	Computer Science
Observer	Juan Cockburn	Computer Engineering

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: Real-Time Control Systems Framework

Name of author: Nishant Thakkar
Degree: Masters of Science
Program: Computer Science
College: College of Computing and Information Services

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, Nishant Thakkar, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: _____ Date: 3/14/05

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, _____, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: _____ Date: _____

Table of Contents

List of Figures

List of Equations

List of Examples

1) Abstract

Computer based controllers are being used increasingly in every aspect of modern life and have been integrated into everything from simple thermostats to the most complicated fighter jets. Multiple methodologies can be used within these controllers including differential equations, discrete difference equations, and fuzzy logic. Also, they can run on platforms ranging from scaled down versions of consumer operating systems to directly on embedded processors. What's missing is a low cost framework that links the variety of methodologies used in controllers and the variety of platforms that controllers can run on. My thesis is that it is possible to create a real-time control systems framework that is portable across platforms, is extensible, and has a low cost. The approach to make the framework multi-platformed and to allow it to use multiple control methodologies was to build a software system with a layered design. This was done so that the platform specific information could be isolated in its own layer, the control methodology information could be isolated in its own layer, and the services provided by the framework could be loosely couple from both of them. To solve the problem of price it was decided to make the project open source, which had the added benefit of allowing students, scientists and even corporations to improve upon the system in the long run. The solution that was implemented is called RTCSF, Real-Time Control Systems Framework. It runs on two different platforms, RTLinux, and a preemptable version of the base Linux 2.6 kernel. Two control methodologies were also implemented in RTCSF, one uses discrete difference equations, and the other uses fuzzy logic. It reached the goals of being multi-platformed, multi-methodology, and low cost, although it has limitations such as being only soft real-time on the 2.6 preemptable kernel and when running the fuzzy logic methodologies.

2) Introduction

The Real-Time Control Systems Framework (RTCSF) is a framework that allows for different types of real-time controllers to be deployed upon different real-time platforms. This software was developed to facilitate solving problems for three classes of people: the control engineer with a small budget, the researcher trying to create a new type of control system, and the programmer trying to port controller code to a different real-time OS.

For a control engineer that has a small budget and who needs to develop a control system, RTCSF implementation allows the running of a control system in real-time without the aid of a real-time programmer. This reduces the number of people on a team needed, and allows control engineers to concentrate on the details of creating the best controller for their product instead of the details of implementing the controller. Since all components of a control system can be modeled by the same methodologies as the controller, a whole control system can be built using the same framework, either in one embedded system, or across multiple systems connected to each other.

An additional benefit to the Control Engineer is a shortened time for verification. Since RTCSF is built using a layered approach, each layer, such as the hardware, OS, framework, and control system, can be verified independently of each other layer. Therefore verification of multiple control systems, which all use RTCSF and the same platform would require only the control logic to be verified multiple times, not the whole system.

For a researcher trying to develop a novel control system methodology, RTCSF is an open framework in which new control modules can be plugged into. This encourages

the development of new control methodologies by allowing the researcher to concentrate on the transform instead of the details of how that transform interacts with the outside world.

For a programmer who needs to port a real-time control system to another platform (combination of OS and hardware), RTCSF includes a platform abstraction layer (PAL), which encapsulates all the platform specific information of the framework. The PAL needs to be written only once for each platform, and much of it will be specific to the OS with very little being specific to the hardware itself. This allows any control system written for RTCSF to be able to be run on any system for which a PAL has been written for. So the programmer only needs to rewrite a small portion of the code, and can ignore the rest, instead of having to look through each piece of code in detail.

3) Available Control System Frameworks

3.1) MATLAB

According to it's creator MathWorks, "MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation." [21]

3.1.1) Pros

MATLAB allows for different methodologies of control systems to be run in real-time on a variety of different real-time operating systems. In addition it allows control engineers to design a wide variety of control systems, such as linear time invariant discrete time controllers, continuous time controllers, rule based controllers, fuzzy controllers, etc., using an intuitive graphical interface, called Simulink. It is able to be extended by adding new toolboxes, so if a new type of control system were to be developed it could be added to MATLAB. In order to execute it in real-time, packages can be purchased that allow MATLAB models to be executed in real-time on a variety of platforms, such as RTLinux, VxWorks, etc. [15]

3.1.1.1) Simulink

Users can use Simulink to graphically layout parts of control systems, like they would if they were designing them by hand, and let the tool determine how to mathematically calculate that graphical representation. Simulink is such a good tool for designing control systems that control systems used in airplanes and military grade guidance systems are developed using it. [15]

3.1.2) Cons

The only benefits of RTCSF over MATLAB are cost and openness. Since RTCSF is free and MATLAB is expensive people may decide that they do not wish to pay the money for the added features of MATLAB. RTCSF is also open source, so if people wish to extend its functionality they can. People can create modules to extend MATLAB, but only the company that owns the copyright, MathWorks, can modify and add to the underlying source code. Also because of the openness of RTCSF, if people wish to run it on a platform that isn't supported, they can add support themselves, but with MATLAB if people want to add support for a different platform, they need to go through MathWorks and try and get them to add support. [15]

3.2) Open Real-Time Control Services

The Open Real-Time Control Services (Orocos) is a control systems kernel, which is a framework that provides "a control oriented interface to the low level operating system primitives." [23] This project started as part of the Open Robot Control System project, but that project was split off recently. It is open source, but is mainly being developed by Dr. ir. Herman Bruyninckx at Katholieke Universiteit Leuven in Belgium. This project is very similar to RTCSF, but it is grander in scale, and was not published until after work began on RTCSF. [24]

3.2.1) Pros

Orocos allows for distributing real-time continuous time controllers across a

network. It can use commodity hardware, and is open source, which allows for a low cost solution. It has been implemented on both RTAI and the vanilla 2.6 Linux kernel, and is designed to be able to be easily ported to other RTOSs. [23] [24]

3.2.2) Cons

A drawback of Orocos is that it is not a full framework in its own right. One must still build on top of Orocos in order to have a framework that can be used directly by control engineers. This is less of an issue since such a framework has already been built, and is in use in applications. Although, since there is only one existing framework, and it is limited to only continuous time controllers, currently there is no support for other types of controllers, which is a major drawback. [24]

3.3) The Real-Time Controls Laboratory

The Real-Time Controls Laboratory (RTiC-Lab) is a control systems framework built in collaboration by individuals from FSM Labs, Inc., the creators of RTLinux, and the University of Virginia. [22]

3.3.1) Pros

RTiC-Lab allows for distributing real-time continuous time controllers across a network from one host system to many control systems. It can use commodity hardware, and is open source, which allows for a low cost solution. From a single host system, RTiC-Lab has the built in ability to gather input from and modify variables on the distributed network of control systems [22]

3.3.2) Cons

The main drawbacks of RTiC-Lab are that it is coupled with RTLinux and it has no built in support for control methodologies other then continuous time controllers. Since it is open source, it may be modified to fit one's particular needs, but it may be difficult to port or implement new types of controllers. [22]

3.4) Real-Time Frameworks

In addition to the before mentioned control system frameworks real-time frameworks exist that simplify the development of real-time software and increase portability. Two such frameworks, Real-Time Java and The Embedded Machine, are discussed below along with the reasons that they were not used.

3.4.1) Real-time Java

The Real-Time Specification for Java (RTSJ) is an extension to Java that provides for real-time thread execution, deterministic memory allocation, and other services that a real-time OS would provide. Any application that's written to work with the RTSJ should be able to run on any Real-Time JVM, and therefore be portable to any OS that has a Real-Time JVM. [19]

RTSJ was not used in the implantation of RTCSF due to some limitations of Real-Time Java. In order to run Real-Time Java code a Real-Time JVM must exist for that platform. Although RTSJ runs on over 80 platforms, which include 8 different processor architectures, RTSJ does not run on every platform and a Real-Time JVM must first be

ported, which increased development time and cost [20]. Since RTSJ runs within a JVM, it has an increased memory footprint, which means that there is an increased cost for every instance of the application. Additionally, when writing Real-Time Java code, one is unable to use some of the beneficial features of Java, such as garbage collection and full memory abstraction. Those are the reasons why RTCSF was not implemented using RTSJ. [19]

3.4.2) The Embedded Machine

The Embedded Machine is a portability layer that was built to allow real-time applications to be portable between platforms. The Embedded Machine also uses a virtual machine, which means that the applications will have an increased memory footprint, which translates to an increased cost for every instance of the application. Also, this particular VM is not used by many groups, so it does not currently run on many platforms, which means that porting may still be difficult and costly if a VM does not already exist for the target platform. [2]

4) Background on Control Systems

A control system consists of many parts. The crucial component being the **Plant**, which is what is being controlled. Another major part of a control system is the controller. The controller takes some desired value, known as the **Reference Input**, from the user of the system and uses that to control the plant's output, known as the **Controlled Variable**. [14]

In order to control the plant, the controller needs to regulate the plant's state, which is a set of variables that together with the control signal define the operation of the plant at any point in time. The controller does this by sending data, known as **Control Signals**, to the plant's actuators, which are often small motors within the plant that handle things such as opening and closing of valves. Because communications in physical systems are not completely isolated, disturbances often occur within the control signals. These **Disturbances** have the potential to change the values of the control signals. Due to delay of the signal and disturbances, the resulting plant state is not the exact desired outcome. The components discussed thus far make up a control system. [14]

This is an imperfect system, since there is a potential for problems due to delay, disturbances, and a lack of a perfect model of the plant. To mitigate these problems an addition to the control system was developed. This added the ability for the controller to use information about the current state of the plant. Sensors throughout the plant gather data and when analyzed collectively this creates the **Subset of State**. Similar to disturbances, **Noise** can affect the signals sent from the sensors; therefore, the signals that reach the controller are not precise. The **Feedback Signal** is the subset of the plants state after noise has been taken into account. A control system that also has a feedback signal

is called a feedback control system. Figure 3.1 shows how the components of a feedback control system interact. [14]

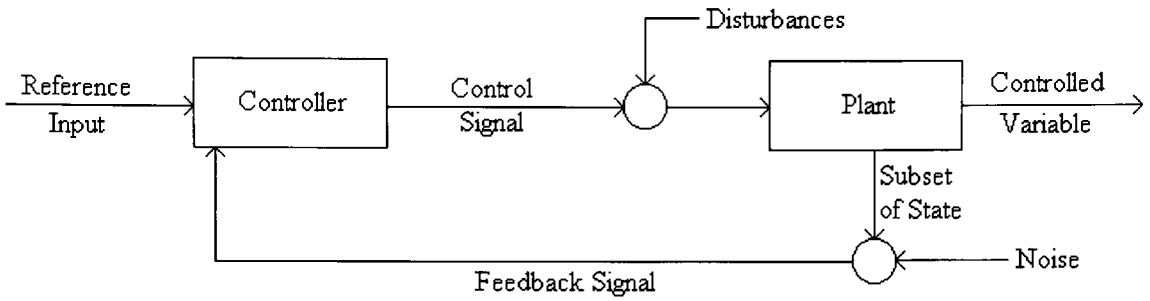


Figure 4.1: The parts of a control system and how they interact.

Feedback control systems can be found in every aspect of modern day life. The diversity of these applications range from simple to complex and include household thermostats, anti-lock breaks, cruise control systems, and even missile guidance system. Figure 3.2 below, shows an example of the parts that make up a feedback cruise control system.

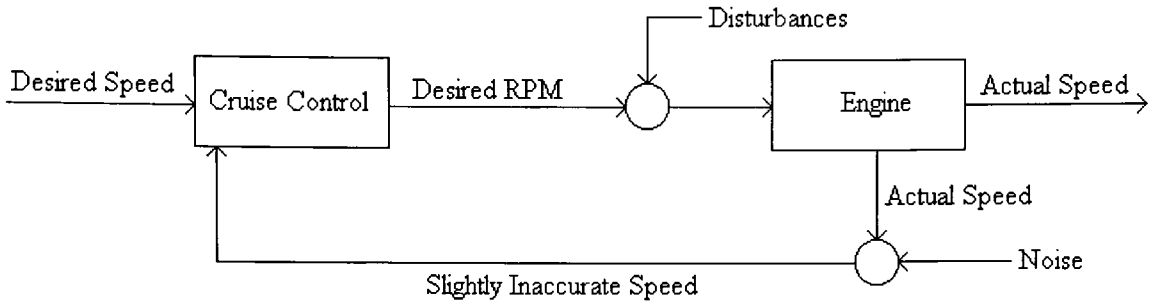


Figure 4.2: An example of a cruise control system.

The controller and plant need not be physical devices. Both can be simulated on computers, or may be computer programs themselves.

4.1) Controllers and Plants in a Single Device

In order to simulating plants and controllers they must be modeled mathematically. Since the inputs and outputs of controllers and plants are different, it may initially seem that they would have different models, but this is not the case. One can create a model which fits both controllers and plants. The benefit of having this one single system is that if someone were to create a new method to building controllers, the new method could also be applied to plant simulations. It also means that RTCSF can work with one system and not need to differentiate between controller and plant.

This is possible because in essence both a controller and a plant work by converting their inputs into outputs. Because of that a model can be created that has only a single set of inputs and a single set of outputs. Figure 3.1.1 shows the controller, the plant, and how each of them maps to the model.

The plant input and controller output can be directory mapped correctly to the

model because of their singular nature; however, controller input and plant output require mathematical manipulation. In order to map the controller input to the model, it is necessary to convert the controller's two sets of input to the model's single set of inputs. This can be done by taking the union of the controller's input sets. The same mapping can be performed with the plant outputs.

For all practical purposes disturbances and noise need not be distinguished from each other for the purpose of computing their impact on the model since both of them affect the output of one system and the input of the other.

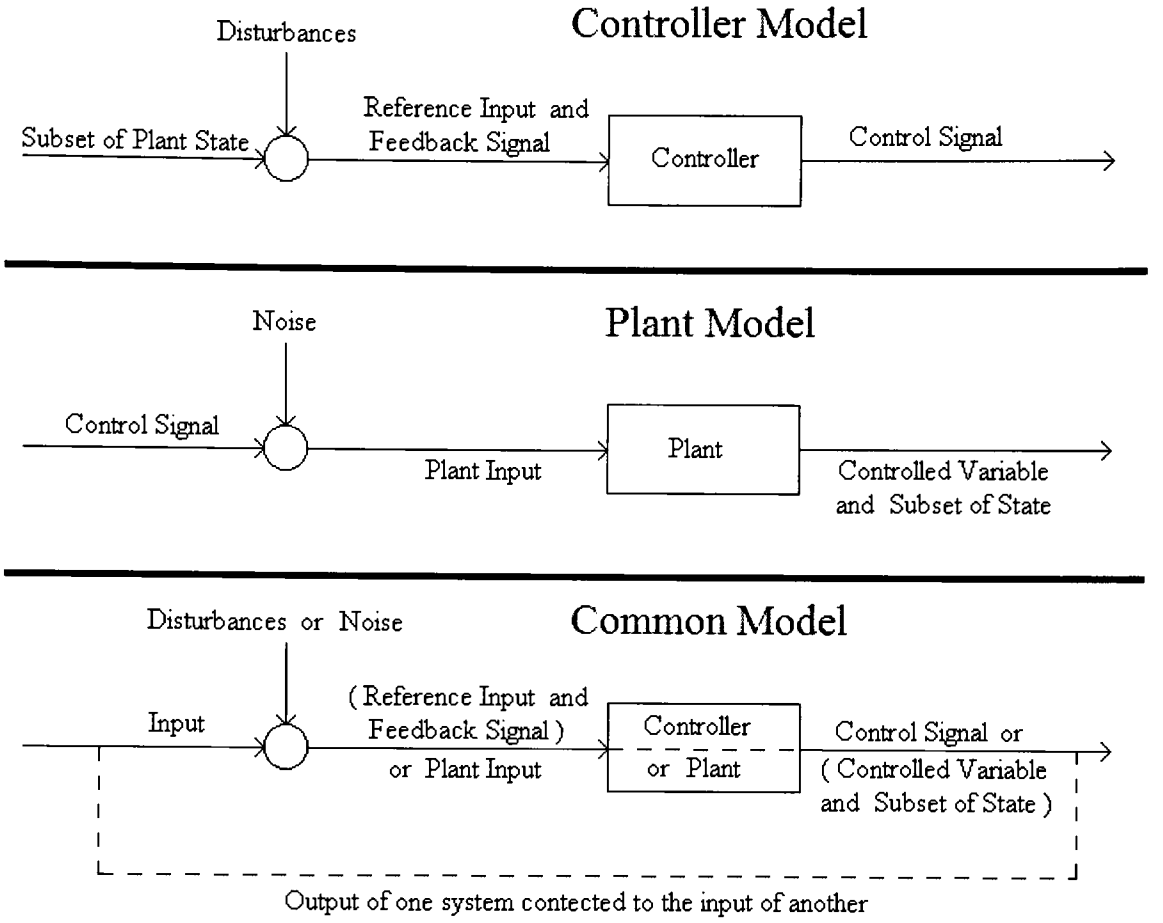


Figure 4.1.1: A visualization of controller, a plant, and their common model.

4.2) Control Methodologies

There are many methods that one can use to design a controller, such as discrete time difference equations, continuous time differential equations, fuzzy logic, and various other methodologies. Some of these methodologies are better for certain situations. For example, discrete controllers are the easiest to implement on computers, while differential controllers are the easiest to implement with basic electrical components.

Discrete and differential controllers have been around for a long time and are very well defined, and can be both linear and non-linear. Rules-based and fuzzy controllers tend to make it be easier to implement systems that are currently being controlled by people, because it is easier to map human actions to rules instead of equations [3].

Due to the shared model of plants and controllers, methodologies used to for one can often be used for the other. Therefore, in order to simplify the rest of this section refers only to controllers.

4.2.1) Linear Time Invariant Discrete Time Controllers

Linear time invariant discrete controllers [8] are traditionally described using discrete transfer functions, such as:

Equation 4.2.1.1 $\mathbf{y}(z) = \mathbf{K}(z)*\mathbf{u}(z)$

$\mathbf{y}(z)$ is the z transform of the output signal
$\mathbf{u}(z)$ is the z transform of the output signal
$\mathbf{K}(z)$ is the z transform of the output signal
z is the z-transfer variable, which is complex

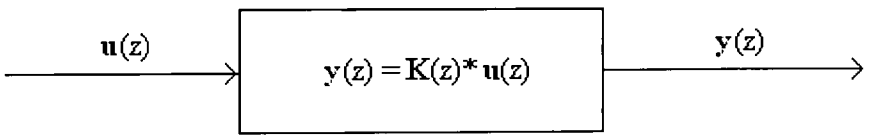


Figure 4.2.1.1: A block diagram representation of a discrete transfer function.

When control engineers design linear time invariant discrete time controllers, they

often design transforms in the form of equation 4.2.1.1 to define them. On the other hand it is simpler for computers to compute equations in terms of time rather than the z-transfer variable. Functions in terms of z-transfer variables can be transformed into state space equations, which are in the time domain, via an inverse z-transform, thereby simplifying computation.

In the time domain the state-space equations consist of two equations for every transfer function that is in the z domain. This can be attributed to the added need of the state-space equations, which require that the current state of the system, be calculated. State space equations are written in the following form:

Equation 4.2.1.2 $\mathbf{x}(t+1) = \mathbf{A}*\mathbf{x}(t) + \mathbf{B}*\mathbf{u}(t)$

Equation 4.2.1.3 $\mathbf{y}(t) = \mathbf{C}*\mathbf{x}(t) + \mathbf{D}*\mathbf{u}(t)$

$\mathbf{x}(t+1)$ is the next state of the system
$\mathbf{x}(t)$ is the current state of the system
$\mathbf{x}(t)$ is the current input vector
$\mathbf{x}(t)$ is the current output vector
t is an integer, representing the current time
A is a coefficient matrix, with $|\mathbf{x}|$ rows and $|\mathbf{x}|$ columns
B is a coefficient matrix, with $|\mathbf{x}|$ rows and $|\mathbf{u}|$ columns
C is a coefficient matrix, with $|\mathbf{y}|$ rows and $|\mathbf{x}|$ columns
D is a coefficient matrix, with $|\mathbf{y}|$ rows and $|\mathbf{u}|$ columns

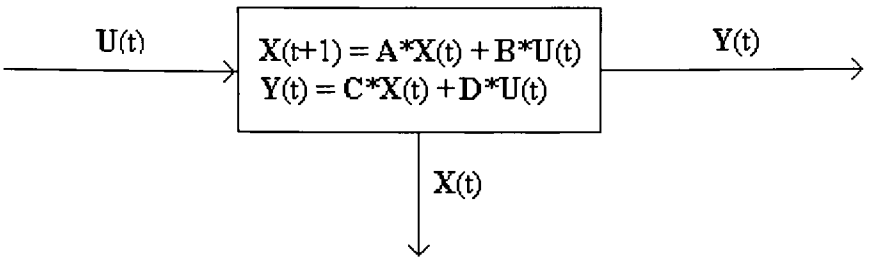


Figure 4.2.1.2: A block diagram of discrete state space equations.

Assuming that the initial conditions are zero, the conversion from the time domain to the z domain is shown below.

The z-transform of equation 4.2.1.2 results in:

Equation 4.2.1.4 $z*\mathbf{x}(z) = \mathbf{A}*\mathbf{x}(z) + \mathbf{B}*\mathbf{u}(z)$

Solving for $\mathbf{x}(z)$ gives:

Equation 4.2.1.5 $\mathbf{x}(z) = ((z\mathbf{I} - \mathbf{A})^{-1}) * \mathbf{B} * \mathbf{u}(z)$

The z-transform of equation 4.2.1.3 results in:

Equation 4.2.1.6 $\mathbf{y}(z) = \mathbf{C} * \mathbf{x}(z) + \mathbf{D} * \mathbf{u}(z)$

Substituting equation 4.2.1.5 into equation 4.2.1.6 gives:

Equation 4.2.1.7 $\mathbf{y}(z) = \mathbf{C} * ((z\mathbf{I} - \mathbf{A})^{-1}) * \mathbf{B} * \mathbf{u}(z) + \mathbf{D}*\mathbf{u}(z)$

Solving in terms of $\mathbf{u}(z)$ results in:

Equation 4.2.1.7 $y(z) = ((\mathbf{C} * ((z\mathbf{I} - \mathbf{A})^{-1}) * \mathbf{B}) + \mathbf{D}) * \mathbf{u}(z)$

Let $\mathbf{K}(z)$ be define as:

Equation 4.2.1.8 $\mathbf{K}(z) = ((\mathbf{C} * ((z\mathbf{I} - \mathbf{A})^{-1}) * \mathbf{B}) + \mathbf{D})$

Substituting equation 4.2.1.8 into equation 4.2.1.7 yields:

Equation 4.2.1.9 $\mathbf{Y}(z) = \mathbf{K}(z) * \mathbf{U}(z)$

Equation 4.2.1.1 equals 4.2.1.9, thereby completing the conversion from state space equations to the discrete transfer function.

It is possible to convert a discrete transfer function to discrete state space equations as illustrated by "Minimal state-space realization in linear system theory: An overview" by B. De Schutter [5].

Using the state space equations it is a matter of calculating the state and the output, since **A**, **B**, **C**, **D**, and the initial state are all user-defined. These types of controllers are very well suited for computers since they are not continuous and only require the periodic calculation of two linear equations when calculated in real-time, or the calculation of a recursive function when the inputs have already been determined.

4.2.2) Continuous Time Controllers

The following two equations are how state space differential controllers [8] are defined:

Equation 4.2.2.1 $d\mathbf{x}/dt(t) = \mathbf{A}*\mathbf{x}(t) + \mathbf{B}*\mathbf{u}(t)$

Equation 4.2.2.2 $\mathbf{y}(t) = \mathbf{C}*\mathbf{x}(t) + \mathbf{D}*\mathbf{u}(t)$

d**x**/dt is the change in state
u is the input vector
v is the output vector
t is a real number, representing the current time
A is a coefficient matrix, with | d**x**/dt | rows and |**x**| columns
B is a coefficient matrix, with | d**x**/dt | rows and |**u**| columns
C is a coefficient matrix, with |**y**| rows and |**x**| columns
D is a coefficient matrix, with |**y**| rows and |**u**| columns

From the state space equations it is possible to calculate the differential transfer function, which is defined as:

Equation 4.2.2.3 $\mathbf{y}(s) = \mathbf{K}(s) * \mathbf{u}(s)$

The conversion follows the same steps as the conversion that was done in the previous section, 4.2.1, with the only major difference being that Laplace transforms must be used instead of z-transforms. The $\mathbf{K}(s)$ that would be found is:

Equation 4.2.2.4 $\mathbf{K}(s) = ((\mathbf{C} * ((s\mathbf{I} - \mathbf{A})^{-1}) * \mathbf{B}) + \mathbf{D})$

It is possible to convert a differential transfer function to state space equations as illustrated by "Minimal state-space realization in linear system theory: An overview" by B. De Schutter [5].

4.2.3) Rule Based Controllers

Rule based controllers [3] are controllers that decide on output using rules instead of just equations. A rules based controller is defined by a set of rules that are activated by a set of inputs, and that modify a set of outputs. Each rule consists of one or more conditions and one or more resulting outputs and has the form of:

Equation 4.2.3.1 If (Conditions) then (Results)

If (q < v) then (let n = n + 2s*(v-q))
If (q = v) then (let n = n + c)
If (q > v) then (let n = N s*(v-q))
q is the speed obtained from the sensors
v is the desired speed

n is the current force of the engine

s is a conversion constant to relate error in speed to engine force

c is a constant that allows for maintaining speed

Example 4.2.3.1: An example of a rule based controller based on the cruise control example in Figure 4.2.

These controllers are slightly more complicated to implement than the previous methods of control, but are also more intuitive, since humans tend to work better with a series of rules then with a set of equations.

4.2.4) Fuzzy Controllers

Fuzzy controllers [3] are in essence rule based controllers that use fuzzy logic instead of standard set theory. To understand fuzzy controllers, one must first understand the basics of fuzzy logic, which is based on set theory. In standard in set theory, an object either belongs to a set or does not, but with fuzzy sets objects can partially belong to sets. Think about this, if someone's hair falls out one strand at a time, when does the person move from the set of people with hair to the set of bald people? With normal set theory it is impossible to tell precisely when the transition between sets occurs, but with fuzzy logic the person always belongs to both sets with a varying degree. When the person has a full head of hair, he belongs to the hair set 100% and the bald set 0%, as time goes on he belongs to the hair set less and the bald set more, and eventually he will be at hair 0% and bald 100% [3].

Fuzzy controllers are an extension of rules based controllers. They consist of a series of rules that use fuzzy values instead of exact values. Therefore the exact inputs must be converted into fuzzy inputs using a process called fuzzification. Fuzzification is done by having a membership function for each fuzzy set, which is used to calculate the degree (or percentage) of membership to that set [3].

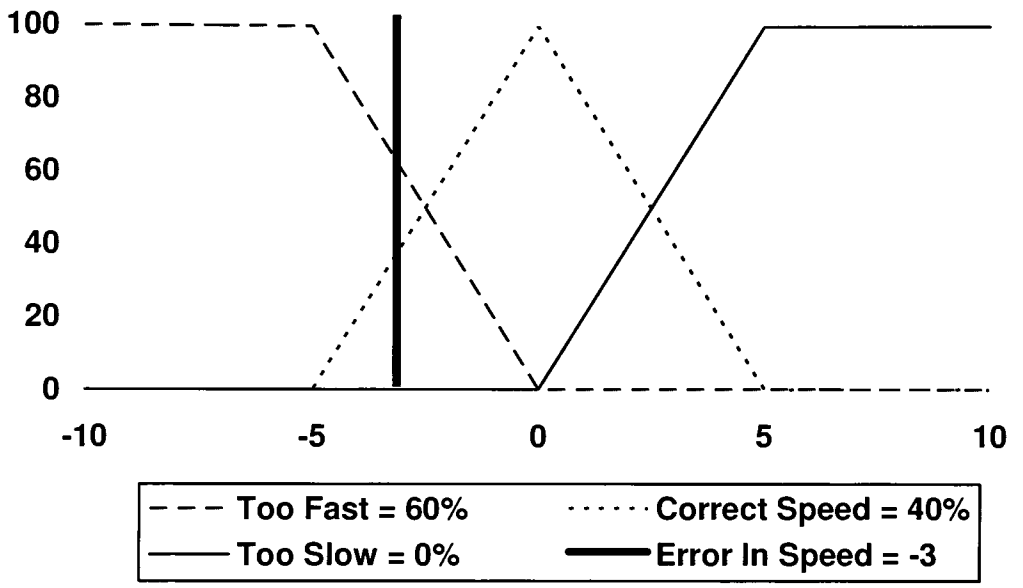


Figure 4.2.4.1 Membership Function for error in speed, which is desired speed - slightly inaccurate speed. As it can be seen, the error of -3 means that the degree of Too Fast is 60%, the degree of Correct Speed is 40% and the degree of Too Slow is 0%.

After the fuzzy values are determined, the rules portion of the fuzzy controller occurs. Like a rule-based controller, each rule consists of one or more conditions, but this time the conditions are always, is the value a member of the set. They take the form of If belongs to A then Result, where the result is a mathematical function of the input

variables [3].

```
If Too Fast
        then ( desired RPM = minRPM + decelerateConst *  ErrorInSpeed )
If Correct Speed
        then ( desired RPM = minRPM + coastConst *  ErrorInSpeed )
If Too Slow
        then ( desired RPM = minRPM + accelerateConst *  ErrorInSpeed )
```

Example 4.2.4.1: An example of the rules of a fuzzy controller using the sets in Figure 4.2.4.1 and based on the cruise control example in Figure 4.2.

Each rule is calculated, and then to find the final result the sum of the results weighted by the degrees of their conditions is calculated.

```
( Too Fast Result * 0.6 + Correct Result * 0.4 + Too Slow Result * 0 ) / ( 0.6 + 0.4 + 0 ) = Final Result
```

Example 4.2.4.2: The calculation for the final result based on example 4.2.4.1.

In summary the fuzzy control has three parts, first the fuzzification, then the rule execution, then the final result calculation.

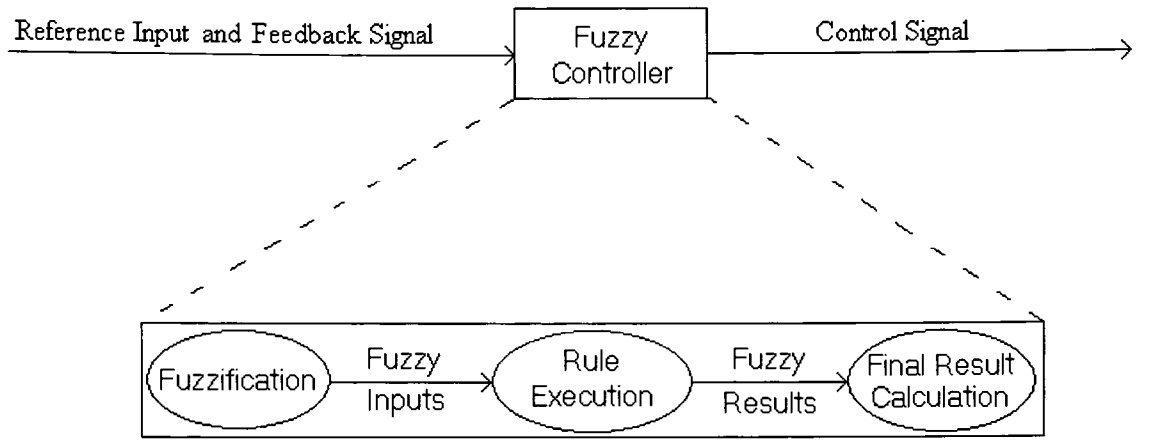


Figure 4.2.4.2 The block diagram of a fuzzy controller.

That is not the only way to execute fuzzy controllers. There are multiple types of fuzzy controllers, two of which are called the Mamdani-type and the Sugeno-type. The type that is described in this section is the Sugeno-type. Since RTCSF currently uses Sugeno controllers for its Fuzzy controllers, the details of Mamdani controllers will not be discussed here [3].

Once again it is important to remember that these methodologies can be applied to plants as well as controllers due to their common model.

5) Background on Real-Time Systems

"[Real-time] describes an application which requires a program to respond to stimuli within some small upper limit of response time (typically milli- or microseconds)." [4] In order to run control systems in real-time, the services of a real-time operating system are required. The specific services that are required are I/O, task management, hardware abstraction, and deterministic execution. This section will elaborate upon these services and discuss a few real-time operating systems.

5.1) Soft and Hard Real-Time

A real-time system can be mapped onto two orthogonal axes, hardness and criticality. The criticality of a system is simply how important it is that the system work properly, and ranges from non-essential to extremely essential. The hardness of a system is how exact is must meet deadlines, and ranges from soft or being able to miss some deadlines by a small amount to hard or being required to make every deadline. If a deadline is missed, then the results are practically useless for a hard real-time system, whereas with a soft real-time system the value is only diminished.

Critical hard real-time systems are what people tend to think of when they think of real-time, but real-time systems exist throughout the plane that is defined by criticality and hardness. Examples of highly critical hard real-time systems are cruise control systems in cars, and auto-pilots in planes. If either of those miss a deadline then it is possible that human lives can be lost, that is why it is very important for hard real-time systems to always finish their calculations within the deadline. An example of a non-critical hard real-time system is a first person shooter style computer game. If the game

does not meet its deadlines then the player may not be able to react fast enough and might die, but since it is a virtual life it is not very essential. An example of a soft real-time non-critical system is a software DVD player, since if it misses a deadline it can still display part of the results and the user will probably just notice a portion of the screen is pixelated, which is not nearly as bad as someone dying. An example of a soft real-time critical system is a stock trading system. If a stock trade occurs a little late, it is still useful, but being late could have an adverse financial impact. In contrast, if the trade does not go through at all it could significantly adversely affect the investor and the reputation of the investment firm.

	Critical	Not-Critical
Hard	Car cruise control systems	Computer games
Soft	Stock trading systems	Consumer audio/video players

Figure 5.1.1: A table of examples of real-time systems

5.2) Requirements for Control Systems

Control systems can fit anywhere on the plane of criticality and hardness. Often they are safety critical components that must run in hard-real time, but they can also be in things as simple as refrigerators, which can be done in soft real-time and are not very critical. So control systems may or may not need hard real-time, but a hard real-time system would be able to satisfy the requirements of all control systems.

5.3) Real-Time Operating Systems

Real-time operating systems (RTOSs) are needed for real-time tasks for the exact

same reason regular operating systems are needed for non-real-time programs, such as hardware abstraction, task management, and other services. RTOSs have to do a few extra things as well, such as having system calls be predictable and deterministic and keeping time at a very fine grain for improved scheduling. Having selectable OS components also helps since it means only the required OS components are taking up resources. Many RTOSs are bound to particular hardware or languages, but others are programmable in any language and are portable across many hardware platforms. [6]

5.3.1) Hardware Abstraction

By using a real-time OS one is able to have useful abstractions for the hardware, which simplifies the application itself and allows hardware changes without requiring changes to the application it self. Also with a RTOS multiple real-time threads can run on the same hardware, and when that happens the RTOS is required to manage memory, processor time, and other common resources. [6]

5.3.2) Task Management

RTOSs must manage all real-time tasks that are running on the system and if the OS is sufficiently advanced it will also be able to manage non-real-time tasks. A major part of managing tasks is scheduling them so that they meet their deadlines and have as little jitter as possible. This can be done using a predetermined schedule, or dynamic schedule based on predetermined priorities or other means. To make sure that tasks are scheduled properly, most RTOSs are able to preempt running tasks to allow other higher priority tasks to run and minimize the effects of lower priority tasks holding on to

resources needed by higher priority tasks. [6]

Real-time tasks often have to communicate with each other and to do so synchronization primitives and/or message passing are needed. Synchronization primitives make sure that one task does not modify resources that are being accessed by other tasks.

The outside world is crucial to real-time applications, if it was not, then there would be no need for the application to be real-time. Do to interactions with the outside world, all RTOSs must provide services that allow tasks to respond to interrupts and outside events. Interrupts can occur for many reasons, I/O arrival, I/O completion, time keeping, and alarms. RTOSs often have priority based interrupts so that key interrupts can be processed immediately. The operating system must have well-documented response times so that the application developer can know what to expect. [6]

5.3.3) Services

Another benefit of RTOSs is that they have many services ordinary operating systems have, that way each application does not have to contain all of that functionality within itself. Communication is crucial to modern programs whether they are real-time or not, that is why RTOSs often support communication, both on the physical layer and on higher networking layers, such as TCP/IP. Other examples of services that RTOSs provide because they are required by most real-time tasks are memory management, file access, device handling, and the ability to connect to interrupt handlers. [6]

5.4) Specific Real-Time Operating Systems

5.4.1) Real-time Linux

There are two ways to make the open source operating system Linux into a real-time operating system [7]. RTLinux and RTAI take the approach of adding a layer below the kernel which can handle interrupts and can schedule the kernel as a low priority real-time process [7]. These variants of real-time Linux are suitable for hard real-time applications such as real-time control systems.

The other approach is to improve the scheduler, the granularity of the clock, and make the kernel preemptable. That approach is taken by two major groups, TimeSys and Montavista, and is included in the following Linux variants: Linux-SRT, QLinux, Linux/RK, RedHawk Linux, ART-Linux, KURT, SMART-Linux, RED-Linux, and the official 2.6 Linux Kernel [11] [17]. These variants are more suited for soft real-time applications like multimedia instead of hard real-time control systems [12].

Since RTLinux and the 2.6 kernel cover the two distinct types of real-time Linux and no other variant offers significant advantages this document will not discuss those variants.

5.4.1.1) Preemptable Kernel

On a 1GHz Pentium 3 the base 2.6 Linux kernel has a worst case interrupt response of under 0.2ms, with an average case of under 0.02ms. [10] On the same machine the task response time with this kernel is 4.5ms in the worst case and under 0.15ms in the average case. The kernel is preemptable and has an efficient scheduler that

is able to schedule processes in constant time This kernel supports standard POSIX threads just like RTLinux making it easier to port applications between the two. Since this is the official Linux kernel it already has been ported to many different architectures including x86, x86-64, PPC (32 or 64bit), and a host of embedded devices that used to use a specialized micro Linux kernel.

5.4.1.2) RTLinux

RTLinux is not really a real-time version of Linux, it is more of a real-time operating system that is also able to run Linux.

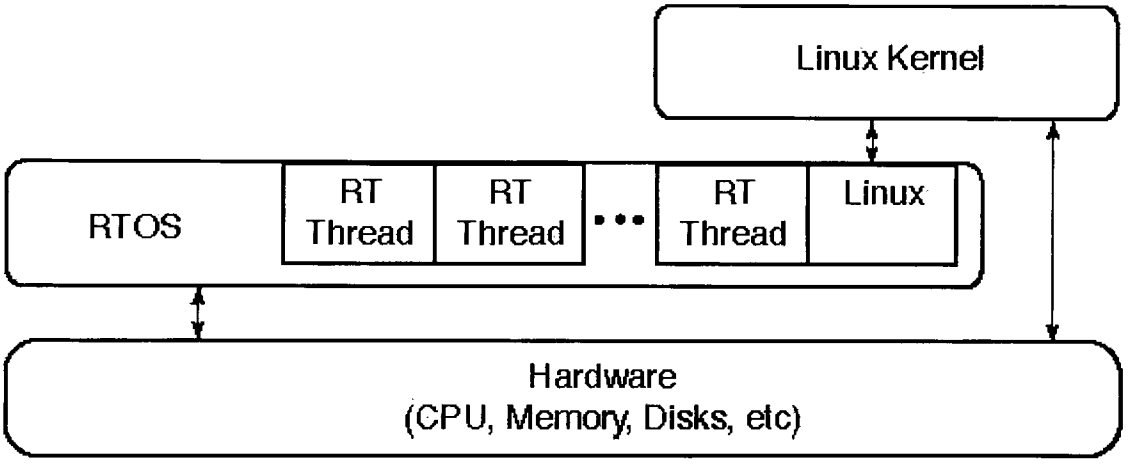


Figure 5.4.1.2.1: A diagram of how RTLinux works. It shows that in RTLinux the Linux Kernel is just another real-time thread that happens to be an OS.

Picture taken directly from reference 9

RTLinux programs are written as kernel modules. This causes some problems since it means that real-time programs do not have memory protection, are unable to use

most library routines, and are not even able to access many kernel functions. On an 1GHz Pentium 3 RTLinux had a task response time of under 0.02ms in the worst case with the average case and interrupt response times under 0.01ms.

5.4.2) VxWorks AE 1.1

VxWorks is a RTOS by WindRiver that is used in many commercial situations. It is not based on a standard OS and was created exclusively to be a RTOS. Because of that it lacks the software library and driver support of the Linux based RTOSs have, even though it does support a large array of hardware. Because it was designed to be an RTOS it has task and interrupt response times comparable to RTLinux even when running on hardware that is a generation older. [13]

5.4.3) Windows

Windows, the popular OS by Microsoft, has many versions that have the potential to obtain hard real-time performance, with Windows CE being the only one that obtains it out of the box. Windows NT, 2000, XP and XP Embedded can all obtain hard real-time performance with extensions from Venturcom and other companies [16]. The benefits of having a real-time version of windows are similar to having a real-time version of Linux. A lot of hardware is already supported, and people are used to interfacing with it. Also since Windows has a lot of brand recognition, using a real-time version tends to go over well with management. [18]

5.4.4) QNX Neutrino

QNX, a name well known in the embedded market, also has their own RTOS called Neutrino. Neutrino, like VxWorks was designed from the start to be a RTOS, but QNX took the approach of making Neutrino as a microkernel OS. This allows Neutrino to restart parts of the kernel when they fail and allows for better scalability. Neutrino was also designed to be POSIX compliant so that code written on Unix and Linux could be easily ported, and it supports over a half a dozen different architectures. [1]

6) Framework Details

The overall architecture of this framework is split into three parts; the platform abstraction layer (PAL), the control methodology layer (CML), and the framework services module (FS), which is specific to control systems in general but not to any one methodology. The PAL fully abstracts platform details from the other layers, and the only communication it initiates with the other modules is during the creation of the FS module. The FS module is tightly coupled to the interface of both the PAL and the CML, but knows specifics about neither the current platform nor the methodologies that can be executed. Although the CML fully hides the details of the control methodologies, both the CML and the specific methodologies use the FS module and the PAL.

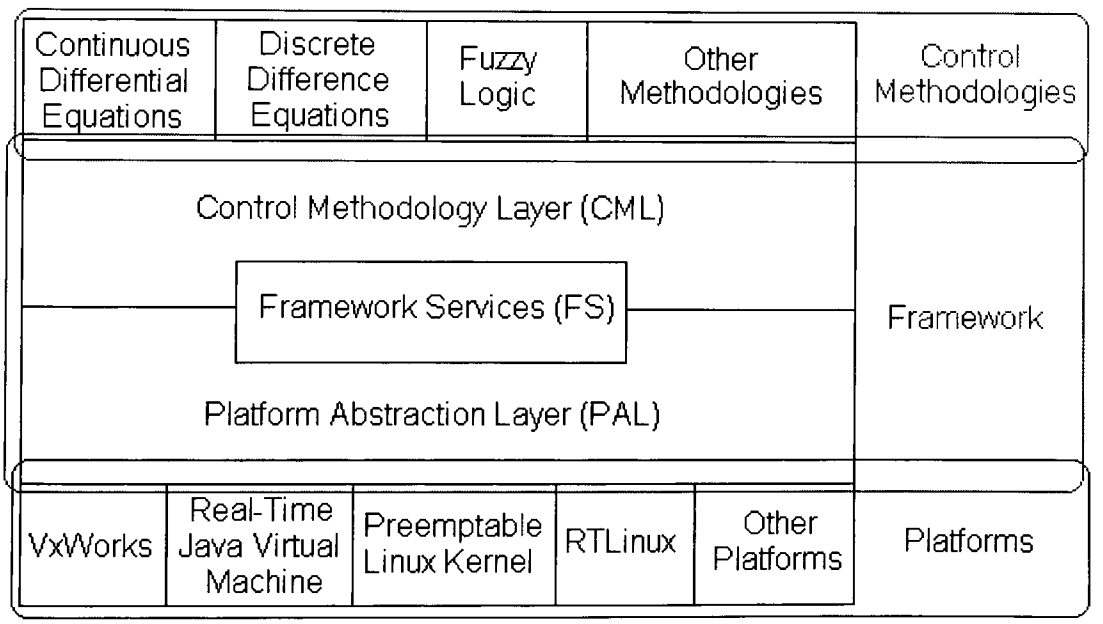


Figure 6.1: The overall architecture of RTCSF.

6.1) Framework Services

The framework services (FS) module is the largest of the three modules and consists of classes that work with scheduling and signal IO. The FS consists of three major parts: the scheduler, which controls the input, execute, output loop (IEO loop) needed to execute control systems; the result generator, which handles the execute portion of the before mentioned input, execute, output loop; and the signal manager, which handles the input and output of signals from the PAL. All of these classes and their interactions are shown in the following diagram, figure 6.1.1.

The result generator is represented by the ResultGenerator class. The class has only one purpose; to execute each of the ControlMethodology objects that registered with it each time its execute() function is called.

Values of inputs and outputs to the system are wrapped by the Signal class so that their manipulation can be controlled. This class provides a getter and setter for a floating point value elsewhere in the system with an additional attribute; the set does not actually take hold until the Signal's updateVal() function is called. The function getNewVal() has been added so that one can access the set value before updateVal() is called. The setSignal() function connects the Signal class to the floating point value that it wraps.

Virtual signals have been added to allow two control systems that are both internal to the framework to communicate with each other. These signals will act exactly like an output signal that is connected to an input signal, but have the benefit of not actually using hardware resources.

The SignalManager class has three purposes: it calls to IODriver to update the signals, it updates the virtual signals, and it allows other classes access to the signals.

When its execute() function is called, SignalManager runs the following operations in order: update all the output Signals; call the IODriver to output the signals in the hardware; cycle the virtual Signals; call the IODriver to read the new input values from the hardware.

Schedulable objects are objects within the framework services layer that execute a loop. Within that loop two things happen, first the thread blocks until another thread wakes it, second a pure virtual function, execute(), is called, which does the work for the period. The classes ResultGenerator and SignalManager both inherit from Schedulable.

The scheduler is represented by the class Scheduler and it has the following responsibilities: ensure SignalManager completes before ResultGenerator starts; output an error if SignalManager or ResultGenerator does not complete within allocated time; call MethodologyCreator to create the methodology instances. It can be modified to act as a watchdog timer for ResultGenerator and SignalManager to make sure that they are restarted if they seem locked. Schedulder uses the Schedulable class to schedule ResultGenerator and SignalManager. Scheduler also inherits from Schedulable, but Scheduler uses the an internal timer to wake itself up to reduce jitter.

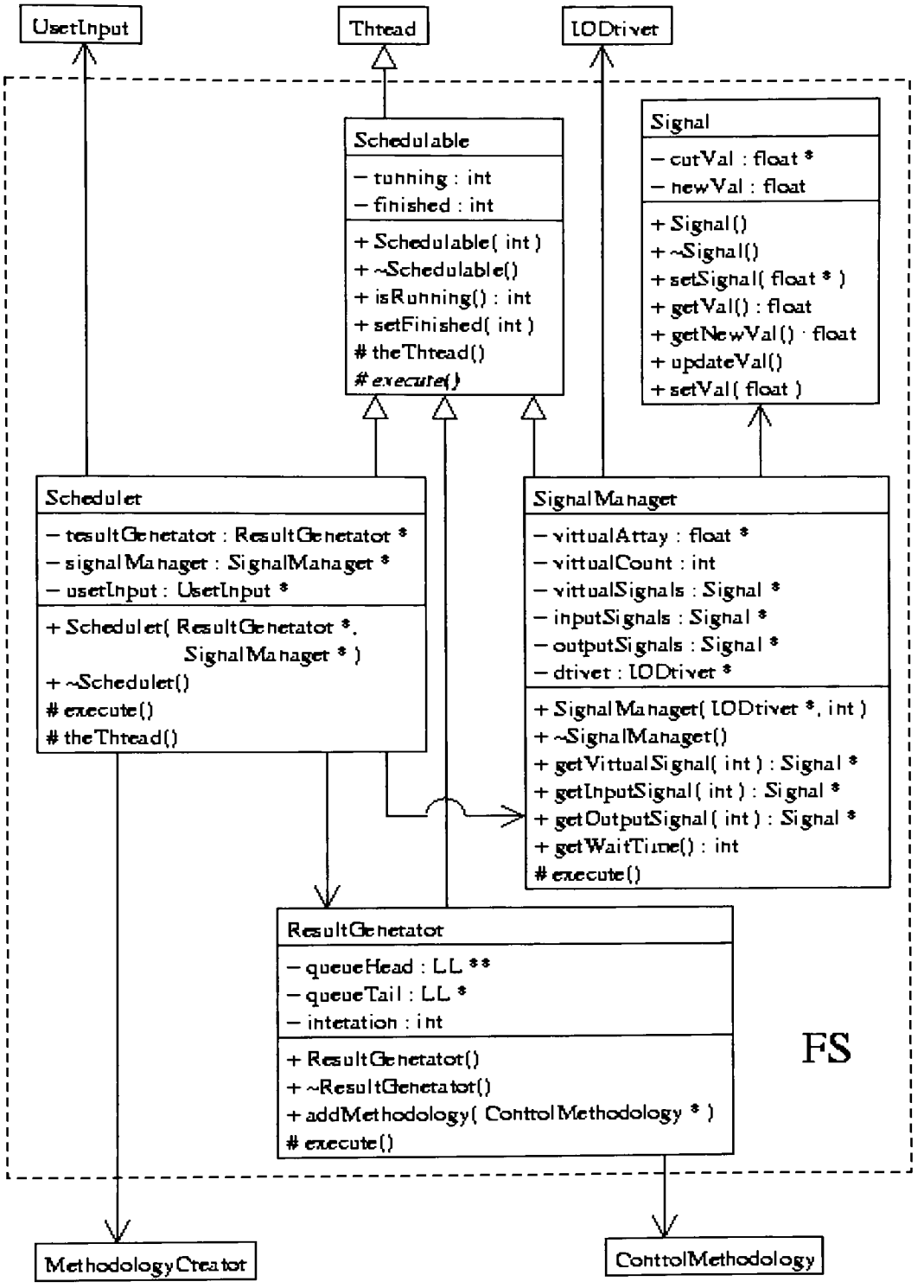


Figure 6.1.1: The class diagram of the FS. This figure gives details of the objects Scheduler, ResultGenerator, SignalManager, Signal, Schedulable, and IODriver.

6.2) *PAL (Platform Abstraction Layer)*

The platform abstraction layer (PAL) is an interface used to separate the specifics of the hardware and OS from the rest of the framework. The PAL is complete enough that it can be implemented on hardware running no OS, although the services that would have been provides by the OS would have to be coded into that instance of the PAL. This means that the FS and CML don't know, or even care, if the PAL is running on top of VxWorks, RTLinux, or even the bare hardware.

The PAL is required to abstract all the functionality required by the higher layer. This functionality can be broken up into three categories: Threads and synchronization, user input and output, and signal input and output.

The thread and synchronization category exposes commonly used threading concepts, which allow for thread creation and deletion, priorities, mutexes, waiting, sleeping, and waking threads that are waiting (both periodically and through other threads).

The user input and output category is required to get information into the program, and to send the results back out. This is split up into two portions, the input and the output. The input allows one to read initialization data from the user, such as the specifics of control systems. This is flexible enough that if the programmer so desired the input could be read in from constants linked in with the code itself. The output allows for multiple levels of data to be outputted so that error messages, normal messages, and other types of messages can be outputted to the user in different ways.

The signal input and output category is required to communicate with external controllers and plants. This is usually done through data acquisition devices, but can be done other ways, or even completely virtually. For example, in the specific implementation there is a set of signals that are nothing more than data in the computer's memory. This is done so that if multiple control systems are running within the same machine, they can communicate without having to use the limited resources on the attached data acquisition device(s).

In addition to those three categories, code is needed to start and stop the system and to hold global defines. The Platform class is what handles those things. Platform.h contains any platform specific information that needs to be included in all files, defines what platform is being used, and defines the priorities of the scheduler, the signal manager, and the thread that executes all the control systems. Platform.cpp contains a small piece of platform specific code that is used to start and stop the program. Platform may or may not actually be a class, it depends on what the specific platform requires to start and stop.

One class handles the thread synchronization, and that is the Thread class. Thread is a virtual class and any other class that wants to be in a separate thread must inherit from Thread.

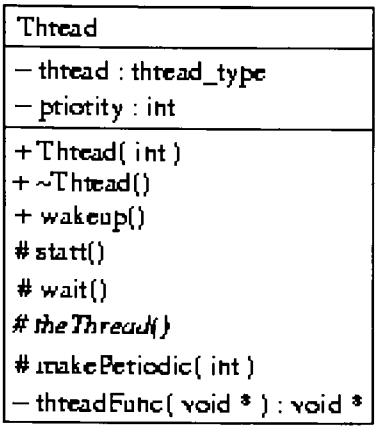


Figure 6.2.1: The class diagram for the Thread class.

Two classes deal with user input and output. UserInput has three functions preparing for input, reading integers, and reading floating point numbers. OutputFifo handles the outputting of data back to the user. Its only responsibility is to output data in real-time so that that the user can access it later. There are three levels of the OutputFifo; the critical FIFO, which is used for critical error messages, the message FIFO, which is used for outputting the status of the control systems, and the log FIFO, which is used for outputting basic log information that can be used later to track down problems. Since every class uses OutputFifo, it is not shown in any of the class diagrams except for figure 6.2.2.

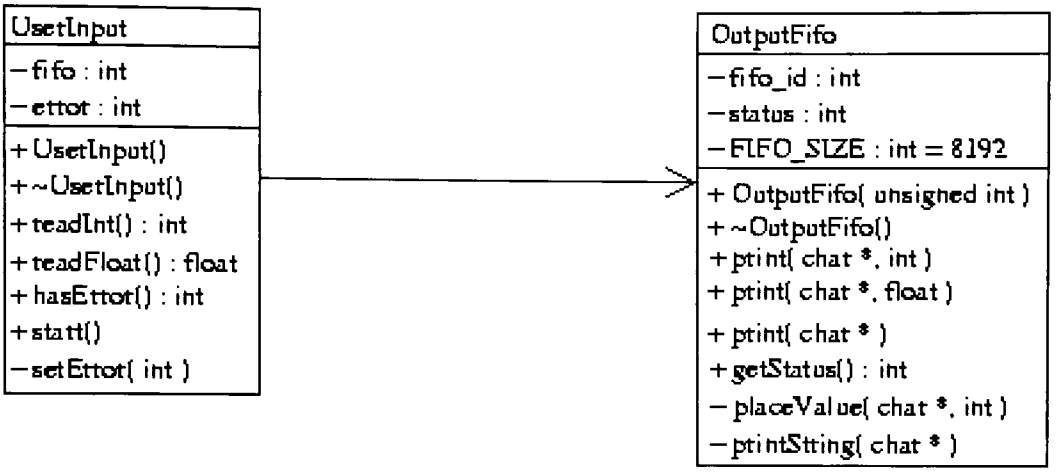


Figure 6.2.2: The class diagram for the UserInput and OutputFifo classes.

The classes that handle the signal input and output is called IODriver. IODriver is meant to communicate with specific data acquisition devices and has functions to read the input signals, set the output signals, and access the arrays that store the signal values. IODriver is tied to the specific data acquisition unit and may not need to change even when the underlying OS is changed, but often will need to change if a different data acquisition unit is used. SignalManager from the FS handles virtual signals so that is not needed to be done within IODriver. Since the IODriver code is specific to more than the OS, the signal input and output category will not be discussed within the specific OS sections.

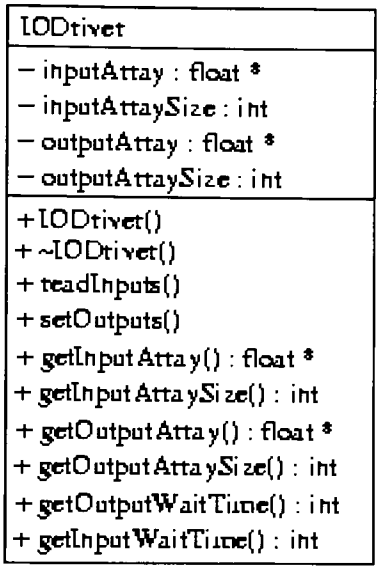


Figure 6.2.3: The class diagram for the IODriver class.

6.2.1) RTLinux Specifics

In RTLinux, to start a program one needs to insert a kernel module with the function int init_module() defined. When that module is inserted, the function is immediately executed. To stop, the module that started the program must be removed from the kernel, and the function void cleanup_module() is invoked. Platform.h simply includes rtl_cpp.h, which is required when classes are used within RTLinux.

RTLinux is able to use non-portable OS primitives to handle waiting, waking up threads, and making threads periodic, but it uses POSIX compatible functions for handling priorities and starting the functions themselves.

UserInput in RTLinux is implemented using RTLinux's real-time FIFO. It is accessible through the file system at /dev/rtf63, and can be accessed like a file. Since the standard C libraries are not available, readInt and readFloat parse the data coming in and convert the characters into either an int or float respectively. UserInput::start must be called within a running thread because it makes a call to put the currently running thread to sleep after installing a signal handler to read in the data asynchronously. The data is read asynchronously to prevent the system from hanging if the process that is filling the FIFO has a lower priority then the thread UserInput::start is running within, which is a common occurrence since UserInput::start is running within a real-time thread. The thread is woken by the signal handler when the end of the initialization data is reached, allowing the program to now utilize the initialization data.

	Outside of the program	RTCSF	UserInput:start	Input Handler
1)		Call UserInput:start		
2)			Load input handler	
3)			Block current thread	
4)	Data passed into FIFO			
5)				Read data until the string "END" is reached
7)				Uninstall handler
8)				Unblock thread (from step 3)
9)			Return	

Figure 6.2.1: This table shows how the UserInput class reads input in RTLinux.

OutputFifo in RTLinux also uses RTLinux's real-time FIFOs. It uses /dev/rtf02 for critical messages, /dev/rtf05 for normal messages, and /dev/rtf08 for log messages; all of which should be emptied by user space processes. In addition to writing critical messages to the FIFO, it is also written to the system console using RTLinux's rtl_printf function.

6.2.2) Preemptable Linux Kernel Specifics

Real-time programs written for Linux with the preemptable kernel extensions, such as the 2.6 kernel, are just programs with their priorities set below the constant

MAX_RT_PRIO. Since RTCSF is just a normal program, when it is started its main function is called, which starts the platform agnostic code and then blocks waiting for user input from the terminal that spawned it. The program ends when the user sends input to the process through that same terminal. Platform.h includes signal.h for signal handling, thread.h for thread support, and sys/time.h for sleep and timer support.

The thread support in the 2.6 kernel is similar to RTLinux, but a few things are different. Specifically the handling of priorities, the start function and threads being started with C functions with the prototype void *func(void *) are the same. The waiting, waking up, and handling of periodic threads are different though. Waiting is done by telling the thread to wait on a condition, and it is woken up by signaling that same condition. Periodic wakeups are done through adding an interval timer, which calls a signal handler after every interval (period), and that handler then just calls wakeup on the thread.

UserInput is simple with this platform, all it does is open a file, inputData, and use the function scanf to parse it.

OutputFifo is more complicated. Each instance of OutputFifo creates a low priority thread that communicates with the OutputFifo through an array in memory that's used as a queue. This way information can be written the OutputFifo in time that's constant relative to the amount of data written, and conversion operations, which are not real-time\, can be done without affecting the real-time threads. While all the queues are written out to files to be reviewed later, the messages from the critical FIFO are also printed out to the console.

6.2.3) Porting to another OS

To add a new OS to RTCSF, one has to create a build environment and implement the headers IODriver.h, UserInput.h, OutputFifo.h, and Thread.h, which are described in section 6.2. The developer must also create the class Platform where Platform.h includes any platform specific information that all files must know about and Platform.cpp has platform specific start and stop code.

6.3) CML (Control Methodology Layer)

The control methodology layer consists of three parts, which are tightly coupled. The first part consists of modules that represent the control methodologies themselves. The second part is the interface of those modules to the rest of the system, which is the class ControlMethodology. The third part is a factory to create instances of the methodologies, which is the class MethodologyCreator.

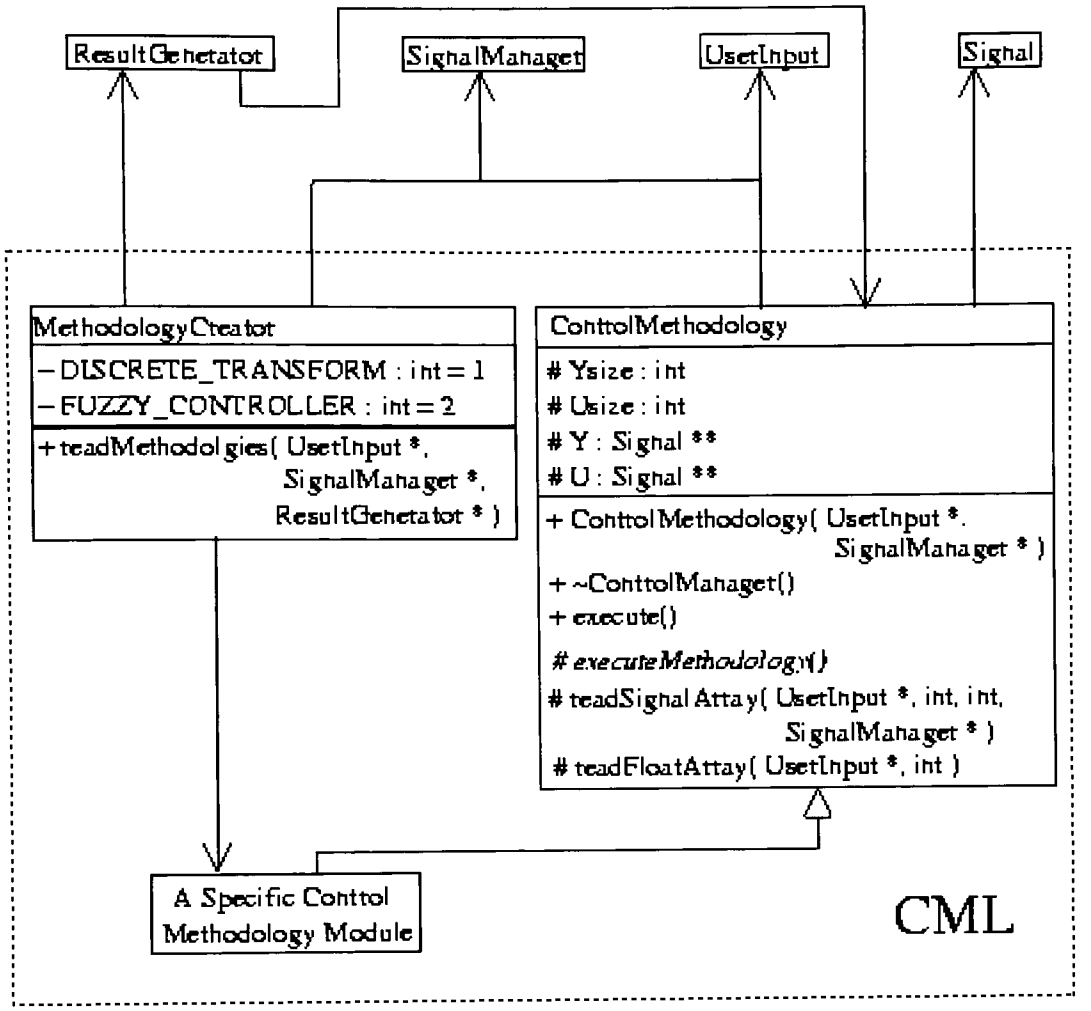


Figure 6.3.1: A diagram of how the control methodology layer works. FuzzyController and DiscreteTransform are modules that represent those types of methodologies.

The modules that represent the methodologies must inherit from ControlMethodology. Each methodology must read in its own information in its constructor, clean itself up in its destructor, and must transform the input to the output during the executeMethodology function. How the methodology goes about doing that is

completely up to the programmer as long as the executeMethodology function can be executed in real-time.

Control Methodology exists for two reasons. The first reason is so that ResultGenerator can have a common interface to the different methodologies. The second reason is to provide common functionality to the methodologies including reading in signal numbers and outputting the signal values to a CSV file during each iteration.

The MethodologyCreator is the only class that knows about all the methodologies that exists within the system. What it does is read in integers from the UserInput class and instantiate different methodology modules depending on the value returned.

Currently two methodologies have been created, one for discrete transforms, and the other for fuzzy controllers.

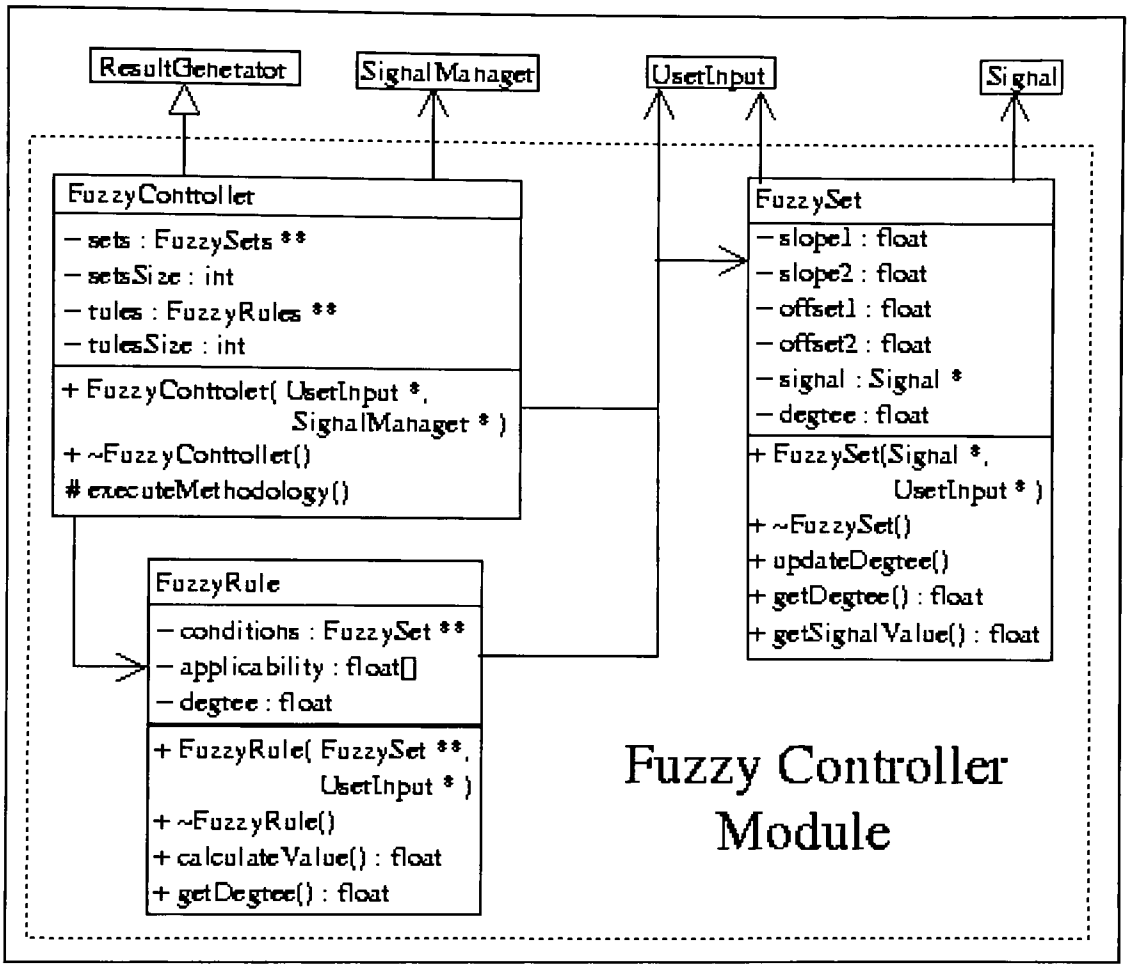
6.3.1) Discrete

The discrete transform module, which is used to develop linear time invariant discrete time controllers, consists of only one class, DiscreteTransform. During the function executeMethodology it executes equations 4.2.1.2 and 4.2.1.3. During its initialization it reads in which signals are used for the state, the initial values of the state, and the A, B, C, and D matrices.

6.3.2) Fuzzy

The fuzzy controller methodology consists of three classes, FuzzyController, FuzzySignal, and FuzzyRule. FuzzyController is the class that inherits from

ControlMethodology, initializes instances of the other two classes, and calculates the final results from the results of each rule.



Figured 6.3.2.1: The interactions of the classes in the fuzzy controller module along with details of the classes in the module itself.

FuzzySet is a fuzzy logic wrapper for Signal. It represents the membership of a signal in a fuzzy set. For example, in Figure 3.5.1 there would be three FuzzySets, the first would have a value of 60%, the second would have a value of 40%, and the third

would have a value of 40%

6.3.3) Adding new methodologies

To add a new methodology two things must be done. First the programmer must edit the MethodologyCreator class and add in the ability to create an instance of the new methodology. Second, the program must create the classes of the module, one of which must inherit from ControlMethodology and implement the executeMethodology function.

6.4) Real-time Contracts

In order for RTCSF to be real-time, each thread that makes up the framework must run in real-time. RTCSF guarantees this through real-time contracts. A real-time contract is a function guaranteeing under what condition it is real-time. This contract is given within the documentation of the function, and consists of "@Real-time", which is followed by the conditions under which the function is real-time. Each function that is executed within the context of a thread running in real-time, must present a real-time contract, as shown in example 6.4.1. If the line is not in the comment block of a function then it is to be assumed that the function is not real-time. If the line is within the comment block of a function then it is to be assumed that the function is deterministic if, and only if, the conditions listed on the line are met.

```
/**
* Function: print( char *string )
* This function prints the string in real-time
* @Real-time This is deterministic with respect to the length of string,
*       when it is the same value, the function will execute in the same
*       amount of time.
**/
```

*Example 6.4.1: An example of the real-time contract for the function print(char *string).*

Each function must verify that everything it does is real-time subject to the conditions that are documented. When that is done, any function that wishes to be real-time and use a real-time function, knows what conditions must be met. This means that each function can be broken down into parts, and if everything that is done locally is real-time, and all function calls are real-time, then the function itself must be real-time.

If all of these threads are deterministic, then the execution of the system is deterministic. The whole system is real-time if it is not only deterministic, but can also meet its deadlines.

6.5) Directory Structure

The files of RTCSF can be separated into two categories, the shared files and the OS specific files. The shared files consist of the interfaces and implementations of both

the Framework Services modules and the Control Methodology Layer, the interfaces and implementations of each of the methodology modules (fuzzy, discrete, and future modules), and the interfaces for the Platform Abstraction Layer. The shared files can all be found within the shared subdirectory of RTCSF. The OS specific files consist of Platform.h, an OS specific implementation of the PAL, and an OS specific build environment. It can either be found in either the root of the RTCSF directory or within subdirectories, except for the shared subdirectory.

7) User Manual

RTCSF is a program that allows users to run a variety of types of control systems on a variety of different platforms. At this point the system requirements are a Linux system running a preemptable Linux kernel with virtual memory turned off and kernel preemption turned on, or a system running RTLinux. The minimum hardware requirements have yet to be determined, but it is recommended that the machine be running a processor that's equivalent or better than a 6^{th} generation x86 processor (Pentium 2) at 400MHz, and that the system contains a minimum of 256MB of RAM and have at least 2GB of hard drive space.

7.1) Installation

To install RTCSF one must first install and configure the underlying OS. For RTLinux one may follow the kick start guide, which may be found at http://www.linuxdevices.com/files/article032/kickstart.pdf. When configuring RTLinux, debugging support is not needed unless you plan on making changes to the underlying C++ code. For installing an OS with the preemptable Linux kernel, a distribution of Linux that uses the 2.6 kernel, such as RedHat Fedora Core 2, may be used. Although, with RedHat Fedora Core 2, one must recompile the kernel with virtual memory turned off and kernel preemption turned on.

After installing the underlying system, determine where you want RTCSF to be installed, making sure to remember that a user with access to the data acquisition device is required to execute it. Once you have chosen a destination, create a directory called RTCSF and then untar RTCSF-version.tar, where version stands for the current version

number, into that directory.

RTCSF is now installed, but it is only setup to run the default control systems. To make RTCSF run the control systems that you desire, you must modify inputFile, and to do that you need to know the input format.

7.2) Input Format

All input is done through numbers, either integers or floats, with each number being separated by some white space, new lines or spaces. The overall format is shown below.

Period time in microseconds (an integer)

N, the number of control systems (an integer)

N control system blocks, which are defined below

The string "END", which is the only exception to the rule of all input being numbers.

A control system block is:

T, the type of control system (an integer)

Usize, the number of input signals

{Signal}^Usize

Ysize, the number of output signals

{Signal}^Ysize

Followed by a fuzzy controller block if type is 2 or a discrete controller block if type is 1.

A discrete controller block is:

Xsize, the number of state signals

{Signal}^Xsize

{float}^Xsize, the initial values of the state

{float}^(Xsize*Xsize)

{float}^(Usize*Xsize)

{float}^(Xsize*Ysize)

{float}^(Usize*Ysize)

A fuzzy controller block is:

{setsBlock}^Usize

{rulesBlock}^Ysize

A setsBlock is:

numSets, the number of sets for that particular input signal (an integer)

{setBlock}^numSets

A setBlock is:

slope1, the slope of the 1st equation that defines the set, the m in y=mx+b (a float)

offset1, the offset of the 1st equation that defines the set, the b in y=mx+b (a float)

slope2, the slope of the 2nd equation that defines the set, the m in y=mx+b(a float)

offset2, the slope of the 2nd equation that defines the set, the b in y=mx+b(a float)

A rulesBlock is:

numRules, the number of rules for that particular output signal (an integer)

{ruleBlock}^numRules

A ruleBlock is:

numConditions, the number of conditions in the rule (an integer)

{condition}^numConditions

{float}^(numConditions+1), the applicability of each condition (a float ranging from 0 to 1)

A condition is:

signalNumber, the signal (from the input) that is involved (an integer)

setNumber, the id of the fuzzySet of the above signal (signalNumber) (an integer)

A Signal is:

{integer}, an id number of a signal within the data acquisition device

OR

-{integer}, an id number of a virtual signal

Input Example	*Explanation*
1000	*Period Time*
1	*Number of control systems*
1	*Control system #1: type, discrete*
2	*Control system #1: # of input signals*
2 3	*Control system #1: input signal ids*
1	*Control system #1: # of output signals*
1	*Control system #1: output signal id*
2	*Control system #1: # of state signals*
-1-2	*Control system #1: state signal ids*
0	*Control system #1: initial state value*
0 0 0 0	*Control system #1: A matrix*
1 0 0 1	*Control system #1: B matrix*
0.5 0.5	*Control system #1: C matrix*
0 0	*Control system #1: D matrix*
END	*Ending token*

Example 7.2.1: The input format of a discrete time system that has 2 inputs, and 1 output. The system latches the input values in the state, and after 1 second it outputs the average of them.

7.3) How to start/stop

To start RTCSF the user must be running as root and then execute RTCSF making sure that the file inputFile is in the same directory and has the initialization data which is explained in section 7.2.

To stop RTCSF in RTLinux simply input a character and then press return.

7.4) Features and limitations

- Period time is limited by 3 things, platform, data acquisition unit, and amount & complexity of control systems. Each platform has a minimum period time that will be used if the period time is set too short, for the 2.6 kernel that is 2.5ms, and for RTLinux that is 0.01ms.
- Each control system on the same machine must have the same period time. For the 2.6 kernel implementation there is a work around, just execute the program multiple times with different data acquisition units and each execution can have its own period.
- The input that each control module can receive is currently limited to only integers and floats.
- The user cannot communicate with the framework or any module while the program is running, except for through signals.

8) Conclusion

RTCSF is able to solve the problems of the three before mentioned classes of people: the control engineer with a small budget, the researcher trying to create a new type of control system, and the programmer trying to port controller code to a different real-time OS. The control engineer has an inexpensive way to build a controller out of commodity hardware. The researcher has a fully open and free framework that is suitable for designing new and different types of controllers and plants. The programmer has an abstraction layer that simplifies porting from one real-time platform to another.

9) Appendix: Source Code

This section contains the source code of the project. It is split up into three major layers as described in section 6; the Platform Abstraction Layer, the Framework Services, and the Control Methodology Layer. The following sub-sections contain the source code of each of those layers. Note that all code provided is open source since it is copyrighted under the GNU Public License (GPL) unless otherwise noted. Each subsection contains a title page to separate it and then the files that are in the specified module.

9.1) Framework Services

```
/**
 * Name:     ErrorHandler.h
 * Authors:  Nishant Thakkar
 * Date:     September 30, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#ifndef ERRORHANDLER_H
#define ERRORHANDLER_H

#include "../Platform.h"

class ErrorHandler {

public:
        static ErrorHandler *Instance();
        void setError();
        int hasError();

protected:
        ErrorHandler();

private:
        static ErrorHandler *_instance;
        int criticalError;

};

#endif
```

```
/**
 * Name:     ErrorHandler.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 30, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* includes */
#include "ErrorHandler.h"

ErrorHandler *ErrorHandler::_instance = NULL;

ErrorHandler *ErrorHandler::Instance() {
        if( _instance == NULL ) {
                _instance = new ErrorHandler();
        }
        return _instance;
}

ErrorHandler::ErrorHandler() {
        criticalError = 0;
}

void ErrorHandler::setError() {
        criticalError = -1;
}

int ErrorHandler::hasError() {
        return criticalError;
}
```

```
/**
 * Name:     ResultGenerator.h
 * Authors:  Nishant Thakkar
 * Date:     September 23, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef RESULTGENERATOR_H
#define RESULTGENERATOR_H

#include "../Platform.h"
#include "Schedulable.h"
#include "ControlMethodology.h"
#include "SignalManager.h"
#include "UserInput.h"

typedef struct LL {
        ControlMethodology *methodology;
        struct LL *next;
} LL;

class ResultGenerator : public Schedulable {

public:
        ResultGenerator();
        ~ResultGenerator();
        void addMethodology( ControlMethodology *cm );

protected:
        void execute();

private:
        LL *queueHead;
        LL **tail;
        int iteration;

};

#endif
```

```
/**
 * Name:     ResultGenerator.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 23, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "ResultGenerator.h"
#include "FuzzyController.h"
#include "DiscreteTransform.h"

ResultGenerator::ResultGenerator() : Schedulable( RESULT_GENERATOR_PRIORITY ) {
        queueHead = NULL;
        tail = &queueHead;
        iteration = 0;

        LOGFIFO->print ("ResultGenerator created\n");

        start();
}


ResultGenerator::~ResultGenerator() {
        LL *queuePtr = queueHead;
        while( queuePtr != NULL ) {
                queuePtr = queuePtr->next;
                delete ( queueHead->methodology );
                delete queueHead;
                queueHead = queuePtr;
        }

        LOGFIFO->print("ResultGenerator destroyed\n");
}

void ResultGenerator::execute() {
        iteration++;

        MESSAGEFIFO->print( "%d", iteration );

        LL *queuePtr = queueHead;
        while( queuePtr != NULL ) {
                queuePtr->methodology->execute();
                queuePtr = queuePtr->next;
        }

        MESSAGEFIFO->print( "\n" );

        LOGFIFO->print( "ALL EXECUTED\n" );
}

void ResultGenerator::addMethodology( ControlMethodology *cm ) {
        *tail = new LL;
        (*tail)->next = NULL;
```



```
        (*tail)->methodology = cm;
        tail = &((*tail)->next);

        LOGFIFO->print( "Methodoogy Added\n" );
}
```

```
/**
 * Name:      Schedulable.h
 * Authors:   Nishant Thakkar
 * Date:      September 23, 2003
 * Revision: 1.1
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef SCHEDULABLE_H
#define SCHEDULABLE_H

#include "../Platform.h"
#include "Thread.h"
#include "ErrorHandler.h"
#include "OutputFifo.h"

class Schedulable   public Thread {

public:
        Schedulable( int priority );
        virtual ~Schedulable();
        int isRunning();
        void setFinished( int status );

protected:
        virtual void theThread();
        virtual void execute() = 0;


private:
        int running;
        int finished;

};

#endif
```

```
/**
 * Name:     Schedulable.h
 * Authors:  Nishant Thakkar
 * Date:     September 23, 2003
 * Revision: 1.1
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "Schedulable.h"

Schedulable::Schedulable( int priority ) : Thread( priority ) {
        running = 0;
        finished = 0;
}


Schedulable::~Schedulable() {
}

int Schedulable::isRunning() {
        return running;
}

void Schedulable::setFinished( int status ) {
        finished = status;
}

void Schedulable::theThread() {
        while (!finished) {
          wait();

          running = -1;
          execute();
          running = 0;
        }
}
```

```
/**
 * Name:     Scheduler.h
 * Authors:  Nishant Thakkar
 * Created:  September 23, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#ifndef SCHEDULER_H
#define SCHEDULER_H

#include "ResultGenerator.h"
#include "SignalManager.h"
#include "Schedulable.h"
#include "UserInput.h"
#include "MethodologyCreator.h"
#include "../Platform.h"

class Scheduler : Schedulable {

public:
        /**
         *  Initialize and start the scheduler
         */
        Scheduler( ResultGenerator *e, SignalManager *sm );

        /**
         *  Stop and clean up the scheduler
         */
        ~Scheduler();

protected:
        void execute();
        void theThread();

private:
        void printTime( char *string );

        ResultGenerator *resultGenerator;
        SignalManager   *signalManager;
        UserInput       *userInput;

};
#endif
```

```
/**
 * Name:     Scheduler.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 23, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#include "Scheduler.h"

void Scheduler::theThread() {

        int schedulePeriod = userInput->readInt();
        if( schedulePeriod < PLATFORM_MIN_SCHEDULE_PERIOD ) {
                schedulePeriod = PLATFORM_MIN_SCHEDULE_PERIOD;
        }

        MethodologyCreator::readMethodologies( userInput, signalManager, resultG
enerator );

        makePeriodic( schedulePeriod );

        printTime( "Schedule loop starting" );

        Schedulable::theThread();
}

void Scheduler::execute() {

  while( resultGenerator->isRunning() ) {
    LOGFIFO->print( "calculations taking too long, wait 10us\n" );
    usleep( 10 );
  }

  printTime( "IO Start" );
  signalManager->wakeup();
  usleep( signalManager->getWaitTime() );

  while( signalManager->isRunning() ) {
    LOGFIFO->print( "IO taking too long, wait 10us\n" );
    usleep( 10 );
  }
  printTime( "Result Generator Start" );
  resultGenerator->wakeup();

}


/**
 *  Initialize the transports
 *  And starts them running
 */
Scheduler::Scheduler( ResultGenerator *g, SignalManager *sm ) :
        Schedulable( SCHEDULER_PRIORITY ) {
```



```
        resultGenerator = g;
        signalManager = sm;

        userInput = new UserInput();

        LOGFIFO->print( "Scheduler created\n" );

        start();
}


/**
 *  Stop all of the transports
 */
Scheduler::~Scheduler() {
        delete userInput;

        LOGFIFO->print( "Scheduler destroyed\n" );
}

void Scheduler::printTime( char *string ) {
        LOGFIFO->print( string );
        LOGFIFO->print( "\n" );
}
```

```
/**
 * Name:     Signal.h
 * Authors:  Nishant Thakkar
 * Date:     March 1, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#include "../Platform.h"
#include "OutputFifo.h"

#ifndef SIGNAL_H
#define SIGNAL_H

class Signal {

public:
        Signal();
        ~Signal();
        void setSignal( float *signal );
        float getVal();
        float getNewVal();
        void updateVal();
        void setVal( float val );


private:
        float *curVal;
        float newVal;

};

#endif
```

```
/**
 * Name:     Signal.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 1, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "Signal.h"

Signal::Signal() {
        curVal = NULL;
//        logFifo->print("Signal created\n");
}

Signal::~Signal() {
//        logFifo->print("Signal destroyed\n");
}

void Signal::setSignal( float *signal ) {
        curVal = signal;
}

float Signal::getVal() {
        return *curVal;
}

float Signal::getNewVal() {
        return newVal;
}

void Signal::updateVal() {
        *curVal = newVal;
}

void Signal::setVal( float val ) {
        newVal = val;
}
```

```
/**
 * Name:     SignalManager.h
 * Authors:  Nishant Thakkar
 * Date:     March 2, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef SIGNALMANAGER_H
#define SIGNALMANAGER_H

#include "../Platform.h"
#include "Signal.h"
#include "IODriver.h"
#include "Schedulable.h"

class SignalManager · public Schedulable (

public:
        SignalManager( IODriver *d, int virtualCount );
        ~SignalManager();
        Signal *getVirtualSignal( int val );
        Signal *getInputSignal( int val );
        Signal *getOutputSignal( int val );
        int getWaitTime();

protected:
        void execute();

private:
        float *virtualArray;
        int virtualCount;
        Signal *virtualSignals;
        Signal *inputSignals;
        Signal *outputSignals;
        IODriver *driver;

);

#endif
```

```
/**
 * Name:     SignalManager.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 2, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "SignalManager.h"

SignalManager::SignalManager( IODriver *d, int virtualSize ) :
        Schedulable( SIGNAL_MANAGER_PRIORITY ) {

        driver = d;

        virtualCount = virtualSize;
        virtualArray = new float[virtualCount];
        virtualSignals = new Signal[ virtualCount ];
        for( int i = 0; i < virtualCount; i++ ) {
                virtualSignals[i].setSignal( virtualArray + i );
        }

        LOGFIFO->print( "Input size = %d\n",  driver->getInputArraySize() );

        inputSignals = new Signal[ driver->getInputArraySize() ];
        for( int i = 0; i < driver->getInputArraySize(); i++ ) {
                inputSignals[i].setSignal( driver->getInputArray() + i );
        }

        LOGFIFO->print( "Output size = %d\n",  driver->getOutputArraySize() );

        outputSignals = new Signal[ driver->getOutputArraySize() ];
        for( int i = 0; i < driver->getOutputArraySize(); i++ ) {
                outputSignals[i].setSignal( driver->getOutputArray() + i );
        }

        LOGFIFO->print( "SignalManager created\n" );

        start();
}

SignalManager::~SignalManager() {

        if( virtualCount > 0 ) {
                delete virtualSignals;
                delete virtualArray;
        }

        if( driver->getInputArraySize() > 0 ) {
//              delete inputSignals;
        }

        if( driver->getOutputArraySize() > 0 ) {
//              delete outputSignals;
```



```
        }

        LOGFIFO->print( "SignalManager destroyed\n" );
}

Signal *SignalManager::getVirtualSignal( int val ) {
        val--;//since val is > 0, we can reduce space by having it point 1 lower
        if( val >= virtualCount ) {
                ErrorHandler::Instance()->setError();
        }
        return virtualSignals + val;
}

Signal *SignalManager::getInputSignal( int val ) {
        if( val >= driver->getInputArraySize() ) {
                ErrorHandler::Instance()->setError();
        }
        return inputSignals + val;
}

Signal *SignalManager::getOutputSignal( int val ) {
        if( val >= driver->getOutputArraySize() ) {
                ErrorHandler::Instance()->setError();
        }
        return outputSignals + val;
}

void SignalManager::execute() {
        for( Signal *ptr = virtualSignals; ptr < virtualSignals + virtualCount;
             ptr++ ) {
                ptr->updateVal();
        }

        for( Signal *ptr = outputSignals;
             ptr < outputSignals + driver->getOutputArraySize(); ptr++ ) {
                ptr->updateVal();
        }
        driver->setOutputs();

        for( Signal *ptr = inputSignals;
             ptr < inputSignals + driver->getInputArraySize(); ptr++ ) {
                ptr->updateVal();
        }
        driver->readInputs();

}

int SignalManager::getWaitTime() {
        int virtualWaitTime = virtualCount * 1/*usec*/;
        int outputWaitTime = driver->getOutputWaitTime();
        int inputWaitTime = driver->getInputWaitTime();
        return virtualWaitTime + outputWaitTime + inputWaitTime;
}
```

9.2) Platform Abstraction Layer

```
/**
 * Name:      IODriver.h
 * Authors:   Nishant Thakkar
 * Date:      September 25, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef IODRIVER_H
#define IODRIVER_H

#include '../Platform.h"
#include "ErrorHandler.h"
#include "OutputFifo.h"

class IODriver (

public:
        IODriver();
        ~IODriver();
        void readInputs();
        void setOutputs();
        float *getInputArray();
        int getInputArraySize();
        float *getOutputArray();
        int getOutputArraySize();
        int getOutputWaitTime();
        int getInputWaitTime();

private:
        void communicateWithUnit( unsigned char *buffer );
        float *inputArray;
        int inputArraySize;
        float *outputArray;
        int outputArraySize;
        char state;
        void *dev;
);

#endif
```

```
/**
 * Name:     OutputFifo.h
 * Authors:  Nishant Thakkar
 * Date:     September 28, 2003
 * Revision: 2.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef OUTPUTFIFO_H
#define OUTPUTFIFO_H

#include "../Platform.h"
#include "ErrorHandler.h"

#define MAXFIFO 9
#define CRITICAL_FIFO_ID 2
#define MESSAGE_FIFO_ID 5
#define LOG_FIFO_ID 8
#define CRITICALFIFO OutputFifo::Instance( CRITICAL_FIFO_ID )
#define MESSAGEFIFO OutputFifo::Instance( MESSAGE_FIFO_ID )
#define LOGFIFO OutputFifo::Instance( LOG_FIFO_ID )

#define INTEGER 1
#define FLOAT 2
#define NOARG 0

class OutputFifo (

public:
        static OutputFifo *Instance( unsigned int id );
        static void destroyAll();
        void print( char *string, int val );
        void print( char *string, float val );
        void print( char *string );
        int getStatus();

private:
        OutputFifo( unsigned int id );
        ~OutputFifo();
        void placeValue( int type, char *string, int size );
        void printString( char *string );
        unsigned int fifo_id;
        int status;

        static const int FIFO_SIZE = 8192;
        static OutputFifo **fifoArray;
        char buffer[FIFO_SIZE];
        int buffer_pos;
        thread_type thread;
        void *arg;
};

#endif
```

```
/**
 * Name:      Thread.h
 * Authors:   Nishant Thakkar
 * Date:      March 24, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef THREAD_H
#define THREAD_H

#include "../Platform.h"
#include "ErrorHandler.h"
#include "OutputFifo.h"

class Thread {

public:
        Thread( int thePriority );
        virtual ~Thread();
        void wakeup();

protected:
        void start();
        void wait();
        virtual void theThread() = 0;
        void makePeriodic( int usecond );

private:
        static void *threadFunc( void *arg );

        thread_type thread;
        int priority;

#ifdef PLATFORM_2_6
        pthread_cond_t cond;
        pthread_mutex_t mutex;
        pthread_mutexattr_t mutex_attr;
#endif

};

#endif
```

```
/**
 * Name:     UserInput.h
 * Authors:  Nishant Thakkar
 * Date:     March 1, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#ifndef USERINPUT_H
#define USERINPUT_H

#include "../Platform.h"
#include "ErrorHandler.h"
#include "OutputFifo.h"

class UserInput {

public:
        UserInput();
        ~UserInput();
        int readInt();
        float readFloat();
        int hasError();
        void start();

private:
        void setError();

        int fifo;
        int error;
};

#endif
```

9.2.1) PAL: RTLinux

```
all: _Control.o _ControlMethodology.o _DiscreteTransform.o _ErrorHandler.o \
     _IODriver.o _OutputFifo.o _ResultGenerator.o _Schedulable.o _Scheduler.o \
     _Signal.o _SignalManager.o _UserInput.o _Platform.o _Thread.o\
     rtl_cpp.o

include ../../debugger/rtl.mk

CRTBEGIN=`g++ -print-file-name=crtbegin.o`
CRTEND=`g++ -print-file-name=crtend.o`

_Control.o: shared/Control.o
        $(LD) -r -o _Control.o $(CRTBEGIN) shared/Control.o $(CRTEND)

_ControlMethodology.o: shared/ControlMethodology.o
        $(LD) -r -o _ControlMethodology.o $(CRTBEGIN) shared/ControlMethodology.
o $(CRTEND)

_DiscreteTransform.o: shared/DiscreteTransform.o
        $(LD) -r -o _DiscreteTransform.o $(CRTBEGIN) shared/DiscreteTransform.o
$(CRTEND)

_ErrorHandler.o: shared/ErrorHandler.o
        $(LD) -r -o _ErrorHandler.o $(CRTBEGIN) shared/ErrorHandler.o $(CRTEND)

_IODriver.o: IODriver.o
        $(LD) -r -o _IODriver.o $(CRTBEGIN) IODriver.o $(CRTEND)

_OutputFifo.o: OutputFifo.o
        $(LD) -r -o _OutputFifo.o $(CRTBEGIN) OutputFifo.o $(CRTEND)

_Platform.o: Platform.o
        $(LD) -r -o _Platform.o $(CRTBEGIN) Platform.o $(CRTEND)

_ResultGenerator.o: shared/ResultGenerator.o
        $(LD) -r -o _ResultGenerator.o $(CRTBEGIN) shared/ResultGenerator.o $(CR
TEND)

_Schedulable.o: shared/Schedulable.o
        $(LD) -r -o _Schedulable.o $(CRTBEGIN) shared/Schedulable.o $(CRTEND)

_Scheduler.o: shared/Scheduler.o
        $(LD) -r -o _Scheduler.o $(CRTBEGIN) shared/Scheduler.o $(CRTEND)

_Signal.o: shared/Signal.o
        $(LD) -r -o _Signal.o $(CRTBEGIN) shared/Signal.o $(CRTEND)

_SignalManager.o: shared/SignalManager.o
        $(LD) -r -o _SignalManager.o $(CRTBEGIN) shared/SignalManager.o $(CRTEND
)

_Thread.o: Thread.o
        $(LD) -r -o _Thread.o $(CRTBEGIN) Thread.o $(CRTEND)

_UserInput.o: UserInput.o
        $(LD) -r -o _UserInput.o $(CRTBEGIN) UserInput.o $(CRTEND)


clean:
        rm *.o
        rm shared/*.o

start: all
```



```
        @sync
        @echo "This is a test of the scheduler"
        @dmesg -c > /dev/nul
        @insmod rtl_cpp.o
        @insmod _ErrorHandler.o
        @insmod _OutputFifo.o
        @insmod _IODriver.o
        @insmod _Thread.o
        @insmod _Schedulable.o
        @insmod _Signal.o
        @insmod _SignalManager.o
        @insmod _UserInput.o
        @insmod _ControlMethodology.o
        @insmod _DiscreteTransform.o
        @insmod _ResultGenerator.o
        @insmod _Scheduler.o
        @insmod _Control.o
        @insmod _Platform.o
        @sleep 1s
#       cat inputer/outBin > /dev/rtf63
        cat testFile > /dev/rtf63

stop:
        @rmmod _Platform
        @rmmod _Control
        @rmmod _Scheduler
        @rmmod _ResultGenerator
        @rmmod _DiscreteTransform
        @rmmod _ControlMethodology
        @rmmod _UserInput
        @rmmod _SignalManager
        @rmmod _Signal
        @rmmod _Schedulable
        @rmmod _Thread
        @rmmod _IODriver
        @rmmod _OutputFifo
        @rmmod _ErrorHandler
        @rmmod rtl_cpp

include $(RTL_DIR)/Rules.make
```

```
/**
 * Name:      IODriver.cpp
 * Authors:   Nishant Thakkar
 * Date:      September 25, 2003
 * Revision: 1.0
 */

/* File Includes */
#include "shared/IODriver.h"

extern "C" {
        #include "nishdriver/labjack.h"
}

IODriver::IODriver() {

        unsigned char sendBuffer[8];
        int result;

        dev = NULL;
        state = 0x00;

        LOGFIFO->print( "Open %d\n", labjack_open(&dev) );

        memset(sendBuffer, 0, 8);
        sendBuffer[0] = 0x08;
        sendBuffer[1] = 0x09;
        sendBuffer[2] = 0x0A;
        sendBuffer[3] = 0x0B;
        sendBuffer[4] = 0x00;
        sendBuffer[5] = 0xC0;
        sendBuffer[6] = 0x00;
        sendBuffer[7] = 0x00;

        result = labjack_write(dev, sendBuffer, 8);
        LOGFIFO->print( "Write %d\n", result );

        outputArraySize = 2;
        outputArray = new float[outputArraySize];
        outputArray[0] = 4.5;
        outputArray[1] = 2.5;

        inputArraySize = 4;
        inputArray = new float[inputArraySize];
        inputArray[0] = 0;
        inputArray[1] = 0;
        inputArray[2] = 4.5;
        inputArray[3] = 2.5;

        LOGFIFO->print ("IODriver created\n");

}


IODriver::~IODriver() {

        LOGFIFO->print( "Release %d\n", labjack_release( dev ) );
        LOGFIFO->print ("IODriver destroyed\n");

]

void IODriver::communicateWithUnit( unsigned char *buffer ) (
```



```
        int result;

        result = labjack_write(dev, buffer, 8);
        while( result < 8 ) {
//              LOGFIFO->print( "f" );
                usleep( 1000 );
                result = labjack_write(dev, buffer, 8);
        }
//      LOGFIFO->print( "Write %d\n", result );

        memset(buffer, 0, 8);
        result = labjack_read(dev, buffer, 8);
        while( result < 8 ) {
//              LOGFIFO->print( "f" );
                usleep( 1000 );
                result = labjack_read(dev, buffer, 8);
        )
//      LOGFIFO->print( "Read %d\n", result );
}

void IODriver::readInputs() (
        unsigned char buffer[8];
        int intArray[4];
        int val1Mask = 0x000000F0;
        int val2Mask = 0x0000000F;


        memset(buffer, 0, 8);
        buffer[0] = 0x08;
        buffer[1] = 0x09;
        buffer[2] = 0x0A;
        buffer[3] = 0x0B;
        buffer[4] = state;
        buffer[5] = 0xC0;
        buffer[6] = 0x00;
        buffer[7] = 0x00;
        state = ~state;

        communicateWithUnit( buffer );

        intArray[0] = buffer[3];
        intArray[0] += ( ( buffer[2] & val1Mask ) << 4 );
        intArray[1] = buffer[4];
        intArray[1] += ( ( buffer[2] & val2Mask ) << 8 );
        intArray[2] = buffer[6];
        intArray[2] += ( ( buffer[5] & val1Mask ) << 4 );
        intArray[3] = buffer[7];
        intArray[3] += ( ( buffer[5] & val2Mask ) << 8 );


        for( int i = 0; i < 4; i++ ) (
                inputArray[i] = ((5.0*intArray[i])/1023.0)-10.0;
//              messageFifo->print( "%x: val: %x float: %d\n", i, intArray[i], (
int)(1000 * inputArray[i]) );
        }

        LOGFIFO->print( "Inputs have be read in\n" );
}

void IODriver::setOutputs() (
        unsigned char buffer[8];
        int bin0, bin1;
```

```
        int bit0 = 0x00000001;
        int bit1 = 0x00000002;
        int bit2 = 0x00000004;
        int bit3 = 0x00000008;

        memset(buffer, 0, 8);

        bin0 = (int)((1023.0F * (outputArray[0] / 5.0F)));
        if( bin0 > 1023 ) {
                bin0 = 1023;
                MESSAGEFIFO->print("value trimmed\n");
        }
        if( bin0 < 0 ) {
                bin0 = 0;
                MESSAGEFIFO->print("value trimmed\n");
        }
        if (bit1 & bin0) buffer[5] |= bit3;
        if (bit0 & bin0) buffer[5] |= bit2;
        buffer[6] = (unsigned char)(bin0/4);  //upper 8 bits of pwm command

        bin1 = (int)((1023.0F * (outputArray[1] / 5.0F)));
        if( bin1 > 1023 ) {
                bin1 = 1023;
                MESSAGEFIFO->print("value trimmed\n");
        }
        if( bin1 < 0 ) {
                bin1 = 0;
                MESSAGEFIFO->print("value trimmed\n");
        }
        if (bit1 & bin1) buffer[5] |= bit1;
        if (bit0 & bin1) buffer[5] |= bit0;
        buffer[7] = (unsigned char)(bin1/4);  //upper 8 bits of pwm command


//        MESSAGEFIFO->print( "bin0: %x\n", bin0 );
//        MESSAGEFIFO->print( "float: %d\n", (int)(1000*(outputArray[0])));
//        MESSAGEFIFO->print( "bin1: %x\n", bin1 );
//        MESSAGEFIFO->print( "float: %d\n", (int)(1000*(outputArray[1])));

        communicateWithUnit( buffer );
        LOGFIFO->print( "Outputs have been set\n" );
}

float *IODriver::getInputArray() {
        return inputArray;
}

int IODriver::getInputArraySize() {
        return inputArraySize;
}

float *IODriver::getOutputArray() {
        return outputArray;
}

int IODriver::getOutputArraySize() {
        return outputArraySize;
}
```

```
/**
 * Name:     OutputFifo.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 28, 2003
 * Revision: 1.0
 */

/* File Includes */
#include "shared/OutputFifo.h"

#define MAX_PRINT_FIFO 10


OutputFifo *OutputFifo::Instance( unsigned int id ) {
        if( id > MAXFIFO ) {
                return NULL;
        }

        if( fifoArray == NULL ) {
                fifoArray = new (OutputFifo *)[MAXFIFO+1];
                for( int i = 0; i < MAXFIFO + 1; i++ ) {
                        fifoArray[i] = NULL;
                }
        }

        if( fifoArray[id] == NULL ) {
                fifoArray[id] = new OutputFifo(id);
        }

        return fifoArray[id];

}

void OutputFifo::destroyAll() {
        if( fifoArray != NULL ) {
                for( int i = 0; i < MAXFIFO + 1; i++ ) {
                        if( fifoArray[i] != NULL ) {
                                delete fifoArray[i];
                        }
                }
        }
}


void OutputFifo::print( char *string, int val ) {
if( fifo_id < MAX_PRINT_FIFO ) {
rtl_printf( string, val );
}
/*
//block
        printString( string );
        placeValue( (char *)&val, sizeof(val) );
//unblock
*/
}

void OutputFifo::print( char *string, float val ) {
if( fifo_id < MAX_PRINT_FIFO ) {
rtl_printf( string, val );
}
/*
//block
```



```
        printString( string );
        placeValue( (char *)&val, sizeof(val) );
//unblock
*/
}

void OutputFifo::print( char *string ) {
if( fifo_id < MAX_PRINT_FIFO ) {
rtl_printf( string );
}
/*
//block
        printString( string );
//unblock
*/
}

OutputFifo::OutputFifo( unsigned int id ) {
        fifo_id = id;
        status = rtf_create( fifo_id, FIFO_SIZE );
        if( status ) {
                rtl_printf( "FIFO id=%d cannot be created, status = %d\n",
                        fifo_id, status );
                rtl_printf( "ENODEV = %d\n", -ENODEV );
                rtl_printf( "EBUSY = %d\n", -EBUSY );
                rtl_printf( "ENOMEM = %d\n", -ENOMEM );

                ErrorHandler::Instance()->setCriticalError();
        }
        else {
                rtl_printf( "FIFO id=%d was sucessfully created.\n", id );
        }
}

OutputFifo::~OutputFifo() {
        rtf_destroy( fifo_id );
}

void OutputFifo::placeValue( char *string, int size ) {
        char value[size+1];
        char *value_ptr = value;

        *(value_ptr++) = '\0';
        while( value_ptr < (value + size + 1) ) {
                *(value_ptr++) = *(string++);
        }

        status = rtf_put( fifo_id, value, size + 1 );
        if( status < 0 ) {
//              ErrorHandler::Instance()->setMinorError();
        }
}

/**
 * This places string into the fifo.
 */
void OutputFifo::printString( char *string ) {
        char *string_ptr = string;

        while( *(string_ptr++) != '\0' );

        int count = string_ptr - string;
```

```
        status = rtf_put( fifo_id, string, count );
        if( status < 0 ) {
                ErrorHandler::Instance()->setMinorError();
        }

}

int OutputFifo::getStatus() {
        return status;
}


OutputFifo **OutputFifo::fifoArray = NULL;
```

```
#include <rtl_cpp.h>

typedef pthread_t thread_type;

#ifndef NULL
        #define NULL 0
#endif

#define SCHEDULER_PRIORITY 2
#define SIGNAL_MANAGER_PRIORITY 3
#define RESULT_GENERATOR_PRIORITY 3
```

```
/**
 * Name:     Platform.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 24, 2004
 * Revision: 1.0
 */

/* includes */
#include "Platform.h"
#include "shared/Control.h"

Control *c;

int init_module(void)
{
        __do_global_ctors_aux();

        c = new Control();

        return 0;
}

void cleanup_module (void) {
        delete c;

        __do_global_dtors_aux();
}
```

```
/*
 * RTLinux C++ support
 * Written by Michael Barabanov, baraban@fsmlabs.com
 * (C) 2000 FSMLabs, GPL
 *
 */
#include <rtl_printf.h>
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/module.h>

int stderr = 0;

int fputs(const char *s, void *stream)
{
        rtl_printf(s);
        return 0;
}

void __pure_virtual (void)
{
        rtl_printf("__pure_virtual called\n");
}

void __this_fixmap_does_not_exist(void)
{
        rtl_printf("__this_fixmap_does_not_exist called\n");
}

void __assert_fail (char *assertion, char *file, unsigned int line,
                char *function)
{
        (void) rtl_printf ("rtl: assertion %s failed.\n", assertion);
}

int __isnan(double x)
{
        return 0;
}
```

```
/**
 * Name:     Thread.h
 * Authors:  Nishant Thakkar
 * Date:     September 23, 2003
 * Revision: 1.1
 */

/* File Includes */
#include "shared/Thread.h"

Thread::Thread( int thePriority ) {
        priority = thePriority;
        thread = 0;
}


Thread::~Thread() {
        if( thread != 0 ) {
                pthread_delete_np( thread );
        }
}

void Thread::start() {
        if( thread != 0 ) {
                ErrorHandler::Instance()->setCriticalError();
                CRITICALFIFO->print( "Thread %d, thread already started\n", (int)this );
        }
        else {
                struct sched_param sched;
                sched.sched_priority = priority;

                pthread_attr_t attr;
                pthread_attr_init( &attr );
                pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_DETACHED );
                pthread_attr_setschedparam( &attr, &sched );

                pthread_create(&thread, &attr, threadFunc, this);

                pthread_attr_destroy( &attr );

        }
}

void Thread::wakeup() {
        if( thread == 0 ) {
                ErrorHandler::Instance()->setCriticalError();
                CRITICALFIFO->print( "Thread %d told to wake up before start was called\n", (int
)this );
        }
        else {
                pthread_wakeup_np( thread );
        }
}

void Thread::makePeriodic( int useconds ) {
        if( thread == 0 ) {
                ErrorHandler::Instance()->setCriticalError();
                CRITICALFIFO->print( "Thread %d told to make periodic before start was called\n",
(int)this );
        }
        else {
                pthread_make_periodic_np (thread, gethrtime(), useconds*1000);
```



```
        }
}

void Thread::wait() {
        if( thread == 0 ) {
                ErrorHandler::Instance()->setCriticalError();
                CRITICALFIFO->print( "Thread %d told to wait before start was called\n", (int)t
his );
        }
        else {
                pthread_wait_np();
        }
}

void *Thread::threadFunc( void *arg ) {
        Thread *thread = (Thread *)arg;

        thread->theThread();

        return NULL;
}
```

```
/**
 * Name:     UserInput.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 2, 2003
 * Revision: 1.0
 */

/* includes */
#include "shared/UserInput.h"

#define FIFO_ID 63

#include <stdlib.h>

char buffer[1024*16]; //16k buffer
char *buffer_start = buffer;
char *buffer_end = buffer;
int finished = 0;
pthread_t thread;

int handler( unsigned int fifo ) {
        if( finished == 0 ) {
                int status = rtf_get( fifo, buffer_end, 1 );
                while( status == 1 ) {
                        buffer_end++;
                        if( buffer_end[-1] == '\n' ) {
                                if( buffer_end - buffer_start > 3 ) {
                                        if( strncmp( buffer_end-4, "end", 3 )
                                            == 0 ) {
                                                finished = 1;
                                                rtl_printf( "EOF\n" );
                                                pthread_wakeup_np( thread );
                                                break;
                                        }
                                }
                        }
                        status = rtf_get( fifo, buffer_end, 1 );
                }
        }
        return 0;
}


void UserInput::start() {
        thread = pthread_self();
        rtf_create_handler( fifo, handler );
        pthread_wait_np();   //handler will wake this process up
}

int isDigit( char ch ) {
        if( ch >= '0' && ch <= '9' ) {
                return 1;
        }
        else {
                return 0;
        }
}

void read_int_helper( int *v, int *n ) {
        int val = 0;
        int neg = 1;
```



```
        //ignore any non-digits
        while( !( isDigit( *buffer_start ) || ( *buffer_start == '-' ) ) &&
                ( buffer_start < buffer_end ) ) {
                buffer_start++;
        }

        //if the number < 0, set neg to -1 so it can be negeted later
        if( ( *buffer_start == '-' ) && ( buffer_start < buffer_end ) ) {
                neg = -1;
                buffer_start++;
        }

        while( isDigit( *buffer_start ) && ( buffer_start < buffer_end ) ) {
                val *= 10;
                val += *buffer_start - '0';
        //      rtl_printf( "val = %d\n", val );
                buffer_start++;
        }

        *n = neg;
        *v = val;
}


int UserInput::readInt() {
        int val;
        int neg;
        read_int_helper( &val, &neg );
        val *= neg;

//      val = (int)strtol( buffer_start, &buffer_start, 10 );

/*
        if( rtf_get( fifo, &val, sizeof(int) ) != sizeof(int) ) {
                CRITICALFIFO->print( "read error in readInt\n" );
                setError();
        }

*/

        LOGFIFO->print( "readInt()=%d\n", val );

        return val;
}

float UserInput::readFloat() {
        float val;
        int ival;
        int neg;
        read_int_helper( &ival, &neg );
        val = ival;

        if( *buffer_start == '.' ) {  //there is more to this float
                buffer_start++;
                float position = 0.1;
                while( isDigit( *buffer_start ) &&
                       ( buffer_start < buffer_end ) ) {
                        float tmp = *buffer_start - '0';
                        tmp *= position;
                        position /= 10;

                        val += tmp;
```

```
                //      rtl_printf( "val = %f\n", val );
                        buffer_start++;
                }

        }



        val *= neg;

//      val = (float)strtod( buffer_start, &buffer_start );

/*
        if( rtf_get( fifo, &val, sizeof(float) ) != sizeof(float) ) {
                CRITICALFIFO->print( "read error in readFloat\n" );
                setError();
        }
*/

        LOGFIFO->print( "readFloat()=%d\n", (int)(val*10000) );

        return val;
}

UserInput::UserInput() {
        error = 0;
        int status;
        fifo = FIFO_ID;
        status = rtf_create( fifo, 8192 );

        if( status != 0 ) {
                CRITICALFIFO->print( "Input FIFO couldn't de created,    );
                switch( status ) {
                        case -ENODEV:
                                CRITICALFIFO->print( "invalid FIFO number\n" );
                                break;
                        case -EBUSY:
                                CRITICALFIFO->print( "FIFO already in use\n" );
                                break;
                        case -ENOMEM:
                                CRITICALFIFO->print( "not enough memory for FIFO\n" );
                                break;
                        default:
                                CRITICALFIFO->print( "unknown error\n" );
                                break;
                }
                setError();
        }
        else {
                LOGFIFO->print( "FIFO id=%d was sucessfully created.\n", fifo );
        }

        LOGFIFO->print( "UserInput created\n" );
}

int UserInput::hasError() {
        if( error ) {
                LOGFIFO->print( "HAS ERROR\n" );
        }
        else {
                LOGFIFO->print( "NO ERROR\n" );
        }
```



```
        return error;
}

void UserInput::setError() {
        LOGFIFO->print( "ERROR IS SET\n" );
        error = 1;
        ErrorHandler::Instance()->setCriticalError();
}

UserInput::~UserInput() {

        rtf_destroy( fifo );

        LOGFIFO->print( "UserInput destroyed\n" );
}
```

```
25000
1
 1
  1 0
  1 1
  3 0
  0
  0.95
  0.25


end
```

```
/**
 * Name:      reader.c
 * Authors:   Nishant Thakkar
 * Date:      May 1, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX_PRINT_FIFO 3
#define INTEGER 1
#define FLOAT 2

int main( int argc, char **argv) {

        FILE *file = fopen( argv[1], "w" );

        int fifoFD = open( argv[2], O_RDONLY );

  char tmp_buffer[8192];
  int num_zeros = 0;
  int i = 0;
  char *val;
  int type = -1;


  unsigned char curChar = '\0';

  while( read( fifoFD, &curChar, 1 ) != 0 ) {
//    printf( "i = %d; curChar = %x\n", i, (unsigned int)curChar );
    if( curChar != '\0' ) {
      num_zeros = 0;
    }
    else {
      num_zeros++;

      if( type < 0 ) {
        val = tmp_buffer + i + 2;
        //      printf( "type i = %d\n", i );
        type = 0;
      }

      if( num_zeros == 8 ) {
        type = val[-1];
        //      printf( "msg = '%s' type = %d\n", tmp_buffer, type );

        //      printf( "HERE" );
        if( type == INTEGER ) {
```



```
          fprintf( file, tmp_buffer, *(int *)val );
        }
        else if( type == FLOAT ) {
          fprintf( file, tmp_buffer, *(float *)val );
        }
        else {
          fprintf( file, tmp_buffer );
        }
        fflush( file );


        i = 0;
        type = -1;
      }

      if( num_zeros > 100 ) {
        close( fifoFD );
        exit(0);
      }

    }

    tmp_buffer[i++] = curChar;

    if( tmp_buffer[0] == 0 ) {
      i = 0;
      type = -1;
    }


  }

//  printf( "END\n" );
}
```

```
#
# Makefile for labjack
#
# Copyright (c) 2003 Eric Sorton <ericsorton@cfl.rr.com>
#
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the Free
# Software Foundation; either version 2 of the License, or (at your option)
# any later version.
#
# NOTE: You must change KERNEL_HEADERS to match the location of the kernel
#       headers on your system.  This varies by distribution and no path is
#       correct for all systems.
#

KERNEL_HEADERS=/usr/src/linux/include

CFLAGS= -D__KERNEL__ -DMODULE -Wall -Wstrict-prototypes -O2 -fomit-frame-pointe
r \
        -fno-strict-aliasing -pipe -fno-strength-reduce \
        -I$(KERNEL_HEADERS)

labjack.o    labjack.c
```

```
extern int labjack_open (void **dev_ptr);
extern int labjack_release (void *dev_ptr);
extern ssize_t labjack_read (void *dev_ptr, unsigned char *buffer, size_t count)
;
extern ssize_t labjack_write (void *dev_ptr, const unsigned char *buffer, size_t
 count);
```

```
extern int labjack_open (void **dev_ptr);
extern int labjack_release (void *dev_ptr);
extern ssize_t labjack_read (void *dev_ptr, unsigned char *buffer, size_t count)
;
extern ssize_t labjack_write (void *dev_ptr, const unsigned char *buffer, size_t
 count);
```

9.2.2) PAL: Preemptable Linux Kernel

```
all: RTCSF

classes=shared/MethodologyCreator.o shared/ControlMethodology.o \
        shared/DiscreteTransform.o shared/ErrorHandler.o \
        shared/ResultGenerator.o shared/Schedulable.o shared/Scheduler.o \
        shared/Signal.o shared/SignalManager.o shared/FuzzyController.o \
        IODriver.o OutputFifo.o UserInput.o Thread.o shared/FuzzySet.o \
        shared/FuzzyRule.o
RTCSF: $(classes) Platform.cpp Platform.h
        g++ $(CXXFLAGS) -lpthread -o RTCSF Platform.cpp $(classes)

start: all
        ./RTCSF

CRTBEGIN=`g++ -print-file-name=crtbegin.o`
CRTEND=`g++ -print-file-name=crtend.o`
CFLAGS= -g
CXXFLAGS = -g

clean:
        rm *.o *~ shared/*~ shared/*.o RTCSF
```

```
/**
 * Name:     IODriver.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 25, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "shared/IODriver.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>


#define AD_RANGE 5.0
#define MAX_OUTPUT 4.3
#define DA_REF 3.75
#define DEVICE "/dev/ttyS1"
#define BAUDRATE B9600

//the # in READ_SIGNALS must be 1 less then inputArraySize
#define READ_SIGNALS "!0RA3"

float stepSize;

int device;


IODriver::IODriver() {
  inputArraySize = 4;
  inputArray = new float[inputArraySize];
  outputArraySize = 2;
  outputArray = new float[outputArraySize];

  stepSize = AD_RANGE / 4095.0;


  device = open(DEVICE, O_RDWR | O_NOCTTY );
  if (device <0) {
    CRITICALFIFO->print("could not open serial port\n");
//  ERROR_HANDLER;
  }

  struct termios newtio;
  memset( &newtio, 0, sizeof(newtio) );
  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR;
  newtio.c_oflag = 0;

  /* set input mode (non-canonical, no echo,...) */
  newtio.c_lflag = 0;
```



```
  newtio.c_cc[VTIME]    = 0; /* inter-character timer unused */
  newtio.c_cc[VMIN]     = 1; /* blocking read until 1 char received */

  tcflush(device, TCIFLUSH);
  tcsetattr(device,TCSANOW,&newtio);


  LOGFIFO->print ("IODriver created\n");

}


IODriver::~IODriver() {
  close( device );

  delete inputArray;
  delete outputArray;

  LOGFIFO->print("IODriver destroyed\n");

}

void IODriver::readInputs() {
  //This tells the device to read the A/D channels
  //subtract 1 from the size to remove the nul byte
  /*
  write( device, READ_SIGNALS, sizeof( READ_SIGNALS ) - 1 );

  for( int i = inputArraySize - 1; i >= 0 ; i++ ) {
    unsigned int intval;
    unsigned char tmp;

    read( device, &tmp, 1 );
    intval = tmp << 8;

    read( device, &tmp, 1 );
    intval += tmp;

    inputArray[i] = intval * stepSize;
  }
  */
  LOGFIFO->print( "Inputs have be read in\n" );
}

void IODriver::setOutputs() {
  unsigned char outputString[] = "!0SV##";//when wrting, use sizeof - 1
  /*
  for( int i = 0; i < outputArraySize; i++ ) {
    if( outputArray[i] > MAX_OUTPUT ) {
      CRITICAL_FIFO->print( "Output #%d trimmed\n", i );
    }

    unsigned int intval = ( outputArray[i] * ( 256 / 2 ) ) / DA_REF;
    char data;
    data = i << 6;  //put the position number in the top 2 bits
    data &= 0x20;   //turn on the multiplier bit
    data += intval >> 3;  //shift the integer bits into the correct position
    outputString[5] = data; //and put all of that into position

    //put the rest of the interger bits into position
```

```
    data = intval << 5;
    outputString[6] = data;

    write( device, outputString, sizeof( outputString ) - 1 );
  }
*/
  LOGFIFO->print( "Outputs have been set\n" );
}

float *IODriver::getInputArray() {
        return inputArray;
}

int IODriver::getInputArraySize() {
        return inputArraySize;
}

float *IODriver::getOutputArray() {
        return outputArray;
}

int IODriver::getOutputArraySize() {
        return outputArraySize;
}

int IODriver::getInputWaitTime() {
        return 100;
}

int IODriver::getOutputWaitTime() {
        return 100;
}
```

```
/**
 * Name:     OutputFifo.cpp
 * Authors:  Nishant Thakkar
 * Date:     September 28, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "shared/OutputFifo.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_PRINT_FIFO 3

OutputFifo *OutputFifo::Instance( unsigned int id ) [
        if( id > MAXFIFO ) {
                return NULL;
        }

        if( fifoArray == NULL ) [
                fifoArray = new (OutputFifo *)[MAXFIFO+1];
                for( int i = 0; i < MAXFIFO + 1; i++ ) {
                        fifoArray[i] = NULL;
                }
        }

        if( fifoArray[id] == NULL ) {
                fifoArray[id] = new OutputFifo(id);
        }

        return fifoArray[id];

}

void OutputFifo::destroyAll() [
        if( fifoArray != NULL ) [
                for( int i = 0; i < MAXFIFO + 1; i++ ) {
                        if( fifoArray[i] != NULL ) {
                                delete fifoArray[i];
                        }
                }
        }
}


void OutputFifo::print( char *string, int val ) [
        printString( string );
        placeValue( INTEGER, (char *)&val, sizeof(val) );
}

void OutputFifo::print( char *string, float val ) [
        printString( string );
        placeValue( FLOAT, (char *)&val, sizeof(val) );
```



```
}

void OutputFifo::print( char *string ) [
        printString( string );
        placeValue( NOARG, NULL, 0 );
}

struct threadArg [
  char *buffer;
  int *buffer_pos;
  int size;
  FILE *file;
};


OutputFifo **OutputFifo::fifoArray = NULL;

/*
read into a temp buffer until you reach 8 consecutive 0s (mark the 1st 0)

printf the tmp buffer with the pointer at the byte 1 after the mark

read until you read a non-0
*/

void *printThread( void *arg ) {


  char *buffer = ((struct threadArg *)arg)->buffer;
  volatile int *buffer_pos = ((struct threadArg *)arg)->buffer_pos;
  int size = ((struct threadArg *)arg)->size;

  char tmp_buffer[size];
  int cur = 0;
  int num_zeros = 0;
  int i = 0;
  char *val;
  int type = -1;
  FILE *file = ((struct threadArg *)arg)->file;


printf( "starting with buffer = %x\n", buffer );

  while( 1 ) {
    while( cur == *buffer_pos ) { usleep( 100 ); }
    //    printf( "i = %d; cur = %d; buffer_pos = %d, char = %d\n", i, cur, *buf
fer_pos, (unsigned int)buffer[cur] );
    if( buffer[cur] != '\0' ) [
      num_zeros = 0;
    }
    else {
      num_zeros++;

      if( type < 0 ) {
        val = tmp_buffer + i + 2;
        //      printf( "type i = %d\n", i );
        type = 0;
      )

      if( num_zeros == 8 ) [
        type = val[-1];
        //      printf( "msg = '%s' type = %d\n", tmp_buffer, type );
```

```
        //      printf( "HERE" );
        //it might be an int or a float, but printf will figure it out for me
        //printf( "oldmark = %d\n", old_mark );
        if( type == INTEGER ) {
          fprintf( file, tmp_buffer, *(int *)val );
        }
        else if( type == FLOAT ) {
          fprintf( file, tmp_buffer, *(float *)val );
        }
        else {
          fprintf( file, tmp_buffer );
        }

        i = 0;
        type = -1;
      }
    }

    tmp_buffer[i++] = buffer[cur];
    cur = ( ( cur + 1 ) % size );

    if( tmp_buffer[0] == 0 ) {
      i = 0;
      type = -1;
    }


  }
  //  printf( "END" );


}


OutputFifo::OutputFifo( unsigned int id ) {
                printf( "FIFO id=%d was sucessfully created.\n", id );

                buffer_pos = 0;
                memset( buffer, 0, FIFO_SIZE );

                arg = malloc( sizeof( struct threadArg ) );
                ((struct threadArg *)arg)->buffer = buffer;
                ((struct threadArg *)arg)->buffer_pos = &buffer_pos;
                ((struct threadArg *)arg)->size = FIFO_SIZE;

                if( id < MAX_PRINT_FIFO ) {
                        ((struct threadArg *)arg)->file = stdout;
                }
                else {
                  if( id > 7 ) {
                        sprintf( buffer, "Log%d", id );
                  }
                  else {
                        sprintf( buffer, "Message%d", id );
                  }

                  FILE *file = fopen( buffer, "w" );
                  if( file == NULL ) {
                    perror( buffer );
                    exit(1);
                  }
```



```
                  ((struct threadArg *)arg)->file = file;
                }


                pthread_attr_t attr;
                pthread_attr_init( &attr );
                pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_DETACHED );
                pthread_create(&thread, &attr, printThread, arg);

                pthread_attr_destroy( &attr );
}

OutputFifo::~OutputFifo() {

  pthread_cancel( thread );

  if( ((struct threadArg *)arg)->file != stdout &&
      ((struct threadArg *)arg)->file != stderr ) {
        fclose( ((struct threadArg *)arg)->file );
  }

  free(arg);

}

void OutputFifo::placeValue( int type, char *string, int size ) {
        buffer[buffer_pos] = type;
        buffer_pos = ( buffer_pos + 1 ) % FIFO_SIZE;

        for( int i = 0; i < size; i++ ) {
          buffer[buffer_pos] = *(string++);
          buffer_pos = ( buffer_pos + 1 ) % FIFO_SIZE;
        }

        for( int i = 0; i < 8; i++ ) {
          buffer[buffer_pos] = 0;
          buffer_pos = ( buffer_pos + 1 ) % FIFO_SIZE;
        }

        //      printf( "in placeValue\n" );
}

/**
 * This places string into the fifo.
 */
void OutputFifo::printString( char *string ) {
  //  printf( "PRINTING '%s'\n", string );

  do {
    buffer[buffer_pos] = *string;
    buffer_pos = ( buffer_pos + 1 ) % FIFO_SIZE;
  } while( *(string++) != '\0' );

  //  printf( "in printString\n" );

}

int OutputFifo::getStatus() {
        return status;
}
```

```
/**
 * Name:     Platform.h
 * Authors:  Nishant Thakkar
 * Date:     March 24, 2004
 * Revision: 2.1
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

#include <unistd.h> //needed for usleep
#include <pthread.h>
#include <signal.h>
#include <sys/time.h>

#ifndef PLATFORM_H
#define PLATFORM_H

typedef pthread_t thread_type;

#define PLATFORM_2_6

#ifndef NULL
        #define NULL 0
#endif

#define RESULT_GENERATOR_PRIORITY 50
#define SIGNAL_MANAGER_PRIORITY 50
#define SCHEDULER_PRIORITY 25

#define PLATFORM_MIN_SCHEDULE_PERIOD 25000

#endif
```

```
/**
 * Name:     Platform.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 24, 2004
 * Revision: 2.1
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* includes */
#include "Platform.h"
#include "shared/IODriver.h"
#include "shared/ResultGenerator.h"
#include "shared/SignalManager.h"
#include "shared/Scheduler.h"
#include "shared/ErrorHandler.h"
#include "shared/OutputFifo.h"
#include <stdio.h>

int main() {
        CRITICALFIFO->print( "Program starting\n" );

        IODriver *d = new IODriver();

        ResultGenerator *g = new ResultGenerator();

        SignalManager *sm = new SignalManager( d, 10 );

        Scheduler *s = new Scheduler( g, sm );

        int tmp;
        scanf( "%d", &tmp );

        delete s;

        delete sm;

        delete g;

        delete d;

        CRITICALFIFO->print( "Program ending\n" );

        usleep(100000);

        OutputFifo::destroyAll();

        return 0;
}
```

```
/**
 * Name:      Thread.h
 * Authors:   Nishant Thakkar
 * Date:      September 23, 2003
 * Revision: 1.1
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "shared/Thread.h"
#include "stdio.h"

static pthread_key_t key;

void handler( int sig ) {
        Thread *thread = (Thread *)pthread_getspecific( key );
//      printf( "3Key: %d, Thread: %x\n", key, thread );
        thread->wakeup();
}

Thread::Thread( int thePriority ) {
  priority = thePriority;
  thread = 0;

  pthread_cond_init( &cond, NULL );
  pthread_mutexattr_settype( &mutex_attr, PTHREAD_MUTEX_ADAPTIVE_NP );
  pthread_mutex_init( &mutex, &mutex_attr );
}

void Thread::start() {
        if( thread != 0 ) {
                ErrorHandler::Instance()->setError();
                CRITICALFIFO->print( "Thread %d, thread already started\n", (int)this );
        }
        else {
                struct sched_param sched;
                sched.sched_priority = priority;

                pthread_attr_t attr;
                pthread_attr_init( &attr );
                pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_DETACHED );
                pthread_attr_setschedparam( &attr, &sched );

                pthread_create(&thread, &attr, threadFunc, this);

                pthread_attr_destroy( &attr );

        }
}


Thread::~Thread() {
        signal( SIGALRM, SIG_DFL );

        pthread_cancel( thread );
```



```
}

void Thread::wakeup() {
LOGFIFO->print( "woken\n");
  pthread_cond_signal( &cond );
}

void Thread::makePeriodic( int useconds ) {
        signal( SIGALRM, handler );

        struct itimerval itv;
        itv.it_interval.tv_sec = itv.it_value.tv_sec = useconds / 1000000;
        itv.it_interval.tv_usec = itv.it_value.tv_usec = useconds%1000000;
        setitimer( ITIMER_REAL, &itv, NULL );

LOGFIFO->print( "interval sec = %d\n", (int)itv.it_interval.tv_sec );
LOGFIFO->print( "interval usec = %d\n", (int)itv.it_interval.tv_usec );

}

void Thread::wait() {
                pthread_cond_wait( &cond, &mutex );
}

void *Thread::threadFunc( void *arg ) {
        Thread *thread = (Thread *)arg;

//      printf( "1Key: %d, Thread: %x\n", key, thread );

        pthread_key_create( &key, NULL );
        pthread_setspecific( key, thread );

//      printf( "2Key: %d, Thread: %x\n", key, thread );


        struct sched_param p;
        p.sched_priority = 2;
        pthread_setschedparam( pthread_self(), SCHED_FIFO, &p );

        thread->theThread();

        return 0;
}
```

```
/**
 * Name:     UserInput.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 2, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* includes */
#include "shared/UserInput.h"
#include <stdio.h>

FILE *input;

int UserInput::readInt() {
  int val;

  if( fscanf( input, "%d", &val ) != 1 ) {
    CRITICALFIFO->print( "read error in readInt\n" );
    setError();
  }

  LOGFIFO->print( "readInt()=%d\n", val );

  return val;
}

float UserInput::readFloat() {
  float val;

  if( fscanf( input, "%f", &val ) != 1 ) {
    CRITICALFIFO->print( "read error in readFloat\n" );
    setError();
  }

  LOGFIFO->print( "readFloat()=%f\n", val );

  return val;
}

UserInput::UserInput() {
  error = 0;

  input = fopen( "inputFile", "r" );
  if( input == NULL ) {
    CRITICALFIFO->print( "Input FIFO couldn't de created\n" );
    setError();
  }
  else {
    LOGFIFO->print( "UserInput\n" );
  }

}

void UserInput::start() {
```



```
      //nothing to do in this implementation
}

int UserInput::hasError() {

  if( error ) {
    LOGFIFO->print( "HAS ERROR\n" );
  }
  else {
    LOGFIFO->print( "NO ERROR\n" );
  }
  return error;

}

void UserInput::setError() {

  LOGFIFO->print( "ERROR IS SET\n" );
  error = 1;
  ErrorHandler::Instance()->setError();

}

UserInput::~UserInput() {

  fclose( input );

}
```

9.3) Control Methodology Layer

9.3.1) CML: General CML Code

```
/**
 * Name:     ControlMethodology.h
 * Authors:  Nishant Thakkar
 * Date:     March 3, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef CONTROLMETHODOLOGY_H
#define CONTROLMETHODOLOGY_H

#include "Signal.h"
#include "UserInput.h"
#include "SignalManager.h"
#include "OutputFifo.h"
#include '../Platform.h"

class ControlMethodology {

public:
        ControlMethodology( UserInput *input, SignalManager *signalManager );
        virtual ~ControlMethodology();
        void execute();
        virtual void executeMethodology() = 0;

protected:
        Signal **readSignalArray( UserInput *input, int size, int isInput,
                                  SignalManager *signalManager );
        float *readFloatArray( UserInput *input, int size );

        int Ysize;
        int Usize;

        Signal **Y;
        Signal **U;
};

#endif
```

```
/**
 * Name:     ControlMethodology.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 3, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "ControlMethodology.h"

Signal **ControlMethodology::readSignalArray( UserInput *input,
                                              int size, int isInput,
                                              SignalManager *signalManager ) {

        Signal **array = new (Signal *)[size];  //an array of Signal pointers

        //for each element in the array, set to a Signal
        for( Signal **arrayPtr = array; arrayPtr < array + size; arrayPtr++ ) {

                int val = input->readInt();

                if( val < 0 ) {
                        *arrayPtr = signalManager->getVirtualSignal( -1 * val );
                }
                else if( isInput ) {
                        *arrayPtr = signalManager->getInputSignal( val );
                }
                else {
                        *arrayPtr = signalManager->getOutputSignal( val );
                }

        }

        return array;
}

float *ControlMethodology::readFloatArray( UserInput *input, int size ) {
        float *array = new float[size];

        for( float *arrayPtr = array; arrayPtr < array + size; arrayPtr++ ) {
                *arrayPtr = input->readFloat();
        }

        return array;
}

ControlMethodology::ControlMethodology( UserInput *input,
                                        SignalManager *signalManager ) {

        Usize = input->readInt();
        U = readSignalArray( input, Usize, 1, signalManager };
        Ysize = input->readInt();
        Y = readSignalArray( input, Ysize, 0, signalManager );
```



```
        for( int i = 0; i < Usize; i++ ) {
                MESSAGEFIFO->print( ",U%d", i );
        }

        for( int i = 0; i < Ysize; i++ ) {
                MESSAGEFIFO->print( ",Y%d", i );
        }

        LOGFIFO->print( "ControlMethodology created\n" );
}


ControlMethodology::~ControlMethodology() {
        delete Y;
        delete U;

        LOGFIFO->print("ControlMethodology destroyed\n");
}

void ControlMethodology::execute() {

        for( Signal **uPtr = U; uPtr < U + Usize; uPtr++ ) {
                MESSAGEFIFO->print( ",%f", (*uPtr)->getVal() );
        }

        executeMethodology();

        for( Signal **yPtr = Y; yPtr < Y + Ysize; yPtr++ ) {
                MESSAGEFIFO->print( ",%f", (*yPtr)->getNewVal() );
        }
}
```

```
/**
 * Name:     MethodologyCreator.h
 * Authors:  Nishant Thakkar
 * Date:     April 13, 2003
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef METHODOLOGYCREATOR_H
#define METHODOLOGYCREATOR_H

#include "../Platform.h"
#include "MethodologyCreator.h"
#include "ResultGenerator.h"
#include "SignalManager.h"
#include "UserInput.h"

class MethodologyCreator {



public:
/*
        MethodologyCreator();
        ~MethodologyCreator();
*/
        static void readMethodologies( UserInput *userInput,
                                SignalManager *signalManager,
                                ResultGenerator *generator );
protected:
        static const int DISCRETE_TRANSFORM = 1;
        static const int FUZZY_CONTROLLER = 2;


};

#endif
```

```
/**
 * Name:     MethodologyCreator.cpp
 * Authors:  Nishant Thakkar
 * Date:     April 13, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "MethodologyCreator.h"
#include "FuzzyController.h"
#include "DiscreteTransform.h"

void MethodologyCreator::readMethodologies( UserInput *userInput,
                                            SignalManager *signalManager,
                                            ResultGenerator *resultGenerator ){
        LOGFIFO->print( "Read Methodology Starting\n" );

        userInput->start();

        MESSAGEFIFO->print( "Iteration" );

        int methodologyCount = userInput->readInt();
        LOGFIFO->print( "Methodology Count = %d\n", methodologyCount );

        for( int i = 0; ( i < methodologyCount ) & !( userInput->hasError() );
             i++ ) {

                int type = userInput->readInt();
                LOGFIFO->print( "TYPE %d\n", type );

                if( !( userInput->hasError() ) ) {
                        if( type == DISCRETE_TRANSFORM ) {

                                DiscreteTransform *dt =
                                        new DiscreteTransform( userInput,
                                                               signalManager );
                                resultGenerator->addMethodology( dt );

                        }
                        else if( type == FUZZY_CONTROLLER ) {
                                FuzzyController *fc =
                                        new FuzzyController( userInput,
                                                             signalManager );
                                resultGenerator->addMethodology( fc );

                        }
                        else {
                                CRITICALFIFO->print( "Incorrect Methodology Type\n" );
                                break;
                        }
                }

        )
```



```
        MESSAGEFIFO->print( "\n" );
        LOGFIFO->print( "Read Methodology Finished\n" );

}
```

9.3.2) CML: Discrete Module

```
/**
 * Name:     DiscreteTransform.h
 * Authors:  Nishant Thakkar
 * Date:     Febuary 11, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef DISCRETETRANDFORM_H
#define DISCRETETRANDFORM_H

#include "Signal.h"
#include "UserInput.h"
#include "SignalManager.h"
#include "ControlMethodology.h"
#include "../Platform.h"

class DiscreteTransform : public ControlMethodology {

public:
        DiscreteTransform( UserInput *input, SignalManager *signalManager );
        ~DiscreteTransform();
        void executeMethodology();

private:
        void matrixMultAndAdd(  int resultSize, Signal **result,
                                float *Xcoefficient, float *Ucoefficient );

        int Xsize;
        Signal **X;
        float *A;
        float *B;
        float *C;
        float *D;
};

#endif
```

```
/**
 * Name:     DiscreteTransform.cpp
 * Authors:  Nishant Thakkar
 * Date:     Febuary 11, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "DiscreteTransform.h"

void DiscreteTransform::matrixMultAndAdd(  int resultSize, Signal  **result,
                           float *Xcoefficient, float *Ucoefficient ) {
        int i, j;
        for( i = 0; i < resultSize; i++ ) {
                float tmpVal = 0;

                Signal **x = X;
                for( j = 0; j < Xsize; j++ ) {
                        tmpVal += (*x)->getVal() * (*Xcoefficient);
                        x++;
                        Xcoefficient++;
                }

                Signal **u = U;
                for( j = 0; j < Usize; j++ ) {
                        tmpVal += (*u)->getVal() * (*Ucoefficient);
                        u++;
                        Ucoefficient++;
                }

                (*result)->setVal( tmpVal );

                result++;
        }
}

DiscreteTransform::DiscreteTransform( UserInput *input,
                                      SignalManager *signalManager ) :
        ControlMethodology( input, signalManager ) {

        Xsize = input->readInt();
        X = readSignalArray( input, Xsize, 0, signalManager );
LOGFIFO->print( "Signals read in\n" );
      for( Signal **xPtr = X; xPtr < X + Xsize; xPtr++ ) {
                (*xPtr)->setVal( input->readFloat() );
        }
LOGFIFO->print( "X read in\n" );

        A = readFloatArray( input, Xsize * Xsize );
LOGFIFO->print( "A read in\n" );
        B = readFloatArray( input, Usize * Xsize );
LOGFIFO->print( "B read in\n" );
        C = readFloatArray( input, Xsize * Ysize );
LOGFIFO->print( "C read in\n" );
```



```
        D = readFloatArray( input, Usize * Ysize );
LOGFIFO->print( "D read in\n" );

        LOGFIFO->print( "DiscreteTransform created\n" );
}


DiscreteTransform::~DiscreteTransform() {
        delete A;
        delete B;
        delete C;
        delete D;

        delete X;

        LOGFIFO->print("DiscreteTransform destroyed\n");
}

void DiscreteTransform::executeMethodology() {

        //X(t+1) = A*X(t) + B*U(t)  <-- this happens since the signal's value
        //                          doesn't get set until the output stage
        matrixMultAndAdd( Xsize, X, A, B );

        //Y(t) = C*X(t) + D*U(t)
        matrixMultAndAdd( Ysize, Y, C, D );

}
```

9.3.3) CML: Fuzzy Module

```
/**
 * Name:     FuzzyController.h
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef FUZZYCONTROLLER_H
#define FUZZYCONTROLLER_H

#include "Signal.h"
#include "UserInput.h"
#include "SignalManager.h"
#include "ControlMethodology.h"
#include "FuzzySet.h"
#include "FuzzyRule.h"
#include '../Platform.h"

class FuzzyController : public ControlMethodology {

public:
        FuzzyController( UserInput *input, SignalManager *signalManager );
        ~FuzzyController();
        void executeMethodology();

private:
        FuzzySet ***sets;
        FuzzyRule ***rules;

};

#endif
```

```
/**
 * Name:     FuzzyController.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "FuzzyController.h"


FuzzyController::FuzzyController( UserInput *input,
                                  SignalManager *signalManager ) :
  ControlMethodology( input, signalManager ) {

  sets = new (FuzzySet **)[Usize];

  for( int i = 0; i < Usize; i++ ) {

    //read in number of possible values, and equations
    int numSets = input->readInt();
    sets[i] = new (FuzzySet *)[numSets + 1];

    for( FuzzySet **setArrayPtr = sets[i];
         setArrayPtr < sets[i] + numSets;
         setArrayPtr++ ) {
      *setArrayPtr = new FuzzySet( U[i], input );
    }

    (sets[i])[numSets] = NULL;

  }


  rules = new (FuzzyRule **)[Ysize];

  for( int i = 0; i < Ysize; i++ ) {

    //read in number of possible rules for this output signal
    int numRules = input->readInt();
    rules[i] = new (FuzzyRule *)[numRules + 1];

    for( FuzzyRule **ruleArrayPtr = rules[i];
         ruleArrayPtr < rules[i] + numRules;
         ruleArrayPtr++ ) {
      *ruleArrayPtr = new FuzzyRule( sets, input );
    }


    (rules[i])[numRules] = NULL;
  }

  LOGFIFO->print( "FuzzyController created\n" );
}
```



```


FuzzyController::~FuzzyController() {
  for( int i = 0; i < Ysize; i++ ) {

    for( FuzzyRule **ruleArrayPtr = rules[i];
         *ruleArrayPtr != NULL;
         ruleArrayPtr++ ) {

      delete *ruleArrayPtr;

    }

    delete rules[i];
  }
  delete rules;

  for( int i = 0; i < Usize; i++ ) {

    for( FuzzySet **setArrayPtr = sets[i];
         *setArrayPtr != NULL;
         setArrayPtr++ ) {
      delete *setArrayPtr;
    }

    delete sets[i];
  }

  delete sets;


  LOGFIFO->print("FuzzyController destroyed\n");
}

void FuzzyController::executeMethodology() {
  for( int i = 0; i < Usize; i++ ) {
    for( FuzzySet **setArrayPtr = sets[i];
         *setArrayPtr != NULL;
         setArrayPtr++ ) {

      (*setArrayPtr)->updateDegree();

    }
  }

  for( int i = 0; i < Ysize; i++ ) {

    //this calculates a weighted average of the fuzzy rules
    float value = 0;
    float degrees = 0;
    for( FuzzyRule **ruleArrayPtr = rules[i];
         *ruleArrayPtr != NULL;
         ruleArrayPtr++ ) {
      //I'm splitting this into 2 lines since calculate value also calculates th
e degree
      float tmpVal = (*ruleArrayPtr)->calculateValue();
      float tmpDegree = (*ruleArrayPtr)->getDegree();

      value += tmpVal + tmpDegree;
      degrees += tmpDegree;
    }
```

```
    if( degrees != 0 ) {
      Y[i]->setVal( value / degrees );
    }
    else {
      CRITICALFIFO->print("no rules for fuzzy output signal %d\n", i );
    }

  }

  LOGFIFO->print("FuzzyController executed\n");
}
```

```
/**
 * Name:     FuzzyRule.h
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef FUZZYRULE_H
#define FUZZYRULE_H

#include "FuzzySet.h"
#include "UserInput.h"
#include '../Platform.h"

class FuzzyRule {

public:
        FuzzyRule( FuzzySet ***sets, UserInput *input );
        ~FuzzyRule();
        float calculateValue();
        float getDegree();

private:
        FuzzySet **conditions;
        float *applicability;
        float degree;
        int numConditions;
};

#endif
```

```
/**
 * Name:     FuzzyRule.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "FuzzyRule.h"

FuzzyRule::FuzzyRule( FuzzySet ***sets, UserInput *input ) {
  numConditions = input->readInt();

  conditions = new (FuzzySet *)[numConditions];
  for( int i = 0; i < numConditions; i++ ) {
    int signalNum = input->readInt();
    int setNum = input->readInt();
    conditions[i] = (sets[signalNum-1])[setNum-1];
  }

  applicability = new float[numConditions + 1];
  for( int i = 0; i < numConditions + 1; i++ ) {
    applicability[i] = input->readFloat();
  }

}


FuzzyRule::~FuzzyRule() {
  delete conditions;
  delete applicability;
}

float FuzzyRule::calculateValue() {
  float value = applicability[numConditions]; //the offset is stored at the end
  degree = 1;

  for( int i = 0; i < numConditions; i++ ) {
    value += conditions[i]->getSignalValue() * applicability[i];
    degree *= conditions[i]->getDegree();
  }

  return value;
}

float FuzzyRule::getDegree() {
  return degree;
}
```

```
/**
 * Name:     FuzzySet.h
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */


#ifndef FUZZYSET_H
#define FUZZYSET_H

#include "Signal.h"
#include "UserInput.h"
#include  ../Platform.h"

class FuzzySet {

public:
        FuzzySet( Signal *sig, UserInput *input );
        ~FuzzySet();
        void updateDegree();
        float getDegree();
        float getSignalValue();

private:
        float slope1;
        float slope2;
        float offset1;
        float offset2;
        Signal *signal;
        float degree;

);

#endif
```

```
/**
 * Name:     FuzzySet.cpp
 * Authors:  Nishant Thakkar
 * Date:     March 7, 2004
 * Revision: 1.0
 *
 * Copyright (c) 2004 Nishant Thakkar <nat0001@rit.edu>
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 */

/* File Includes */
#include "FuzzySet.h"

FuzzySet::FuzzySet( Signal *sig, UserInput *input ) {
  signal = sig;
  slope1 = input->readFloat();
  offset1 = input->readFloat();
  slope2 = input->readFloat();
  offset2 = input->readFloat();
}


FuzzySet::~FuzzySet() {
  //no cleanup needed
}

void FuzzySet::updateDegree() {
  float val = getSignalValue();

  float eq1 = slope1 * val + offset1;
  float eq2 = slope2 * val + offset2;
  float eq3 = 1;

  //need to find the minimum of the 3 equations
  if( eq1 > eq2 ) {
    degree = eq2;
  }
  else {
    degree = eq1;
  }

  if( degree > eq3 ) {
    degree = eq3;
  }

  if( degree < 0 ) {
    degree = 0;
  }

}

float FuzzySet::getDegree() {
  return degree;
}

float FuzzySet::getSignalValue() {
  return signal->getVal();
```



```
}
```

10) References

1) QNX Realtime Operating System Overview. QNX. 16 April 2004<http://www.qnx.com/products/rtos/>.

2) Henzinger, Thomas A. and Kirsch, Christoph M. "The embedded machine: preditcatble, portable, real-time code." Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation (2002) : 315-326

3) Reznik, Leonid. Fuzzy Controllers. Oxford: Newnes, 1997.

4) "REAL-TIME." HyperDictionary 2003. 23 Feb. 2004 <http://www.hyperdictionary.com/dictionary/real-time>.

5) B. De Schutter, "Minimal state-space realization in linear system theory: An overview." Journal of Computational and Applied Mathematics, Special Issue on Numerical Analysis in the 20th Century - Vol. I: Approximation Theory, vol. 121, no. 1-2 (Sept. 2000) : 331-354

6) Shaw, Alan C. Real-Time Systems and Software. New York:Wiley Text Books, 2001.

7) Ripoll, Ismael. RTLinux vs. RTAI. 2002. 3 Feb. 2004 <http://bernia.upv.es/rtportal/comparative/rtl_vs_rtai.html>.

8) CJC. "CTM: Control Tutorials for Matlab." Regents of the University of Michigan. 18 Aug. 1997. 10 Feb. 2003 <http://www.engin.umich.edu/group/ctm/>.

9) Dieter, William R. Real-Time Linux (RTLinux) Intro. 14 Nov. 2003 <http://www.engr.uky.edu/~ee599/examples/rtlinux_intro.pdf>.

10) White, Brandon. "Linux 2.6: A Breakthrough for Embedded Systems." DeviceForge LLC. 9 Sept. 2003. 3 Feb. 2004 <http://www.linuxdevices.com/articles/AT7751365763.html>.

11) Wurmsdobler, Peter. The Real-Time Linux Foundation Website. Home Page. 2001. Real-Time Linux Foundation. 3 Feb. 2004 <http://www.realtimelinuxfoundation.org/>.

12) "The Real-time Linux Software Quick Reference Guide." DeviceForge LLC. 13 June 2003. 3 Feb. 2004 <http://www.linuxdevices.com/articles/AT8073314981.html>.

13) Beneden, Bart Van. "Examining the VxWorks AE 1.1 RTOS." Dr. Dobb's Journal. November 2002. 28 Feb. 2004 <http://www.ddjembedded.com/resources/articles/2002/0211f/0211f.htm>

14) Warwick, Kevin. Control Systems: An Introduction. New York: Prentice Hall, 1989.

15) The MathWorks – MATLAB and MATLAB for Technical Computing. Home Page. 6 April 2003. MathWorks. 12 April 2004. <http://www.mathworks.com>.

16) "Achieving hard real-time on Windows XP, XP Embedded." Device Forge LLC Oct. 2003. 16 April 2004

<http://www.windowsfordevices.com/articles/AT2503923807.html>.

17) "Preemptable kernel patch makes it into Linux kernel 2.5.4-pre6." DeviceForge LLC 10 Feb. 2002. 24 Feb. 2004 <http://www.linuxdevices.com/news/NS3989618385.html>.

18) "The Real-time Windows Embedded Quick Reference Guide." Device Forge LLC 6 Oct. 2003. 16 April 2004 <http://www.windowsfordevices.com/articles/AT5376962137.html>.

19) "The Real-Time Specification for Java." CollabNet, Inc 12 Nov. 2001. 26 Sept. 2004 <https://www.rtj.org/ >.

20) "Java Reference Implementation (RI) and Technology Compatibility Kit (TCK)." TimeSys Corp. 2004. 26 Sept 2004 <http://www.timesys.com/index.cfm?bdy=java_bdy_ri.cfm>.

21) The MathWorks – MATLAB – The Language of Technical Computing. MATLAB Product Page. MathWorks. 28 Sept 2004. <http://www.mathworks.com/products/matlab/>.

22) E. Hilton, V. Yodaiken, M. Humphrey, and P. Allaire. "The Real Time Controls Laboratory, an Open Source, Hard Real Time, Controls Implementation Platform." Proceedings of Second Real-Time Linux Workshop , Orlando, FL (November, 2000)

23) H. Bruyninckx, P. Soetens, and B. Koninckx. "A Software Framework for

Component-based Distributed Feedback Control Kernels." K.U.Leuven—PMA Technical Report (January 2004) <http://www.orocos.org/documents/motconframe.pdf>.

24) The Orocos Project - Orocos. Home Page. 23 Sept. 2004. Herman Bruyninckx. 30 Sept. 2004. <http://www.orocos.org/>.