# Foucault's Pendulum in the Distributed Control Lab

Andreas Rasche, Peter Tröger, Michael Dirska and Andreas Polze
University of Potsdam
14482 Potsdam, Germany
{andreas.rasche|peter.troeger|michael.dirska|andreas.polze}@hpi.uni-potsdam.de

## Abstract

*The 'Distributed Control Lab' [6] at Hasso-Plattner-Institute, University of Potsdam allows experimentation with a variety of physical equipment via the web (intra and internet), among them the Lego Mindstorm robots and Foucault's Pendulum. In order to conduct control experiments, students may write programs, which are validated, run on a simulator, and eventually downloaded on the actual control device. We use online replacement of software components (dynamic re-configuration) as a safeguard mechanism to avoid damage to our hardware.*

*Our research focuses on the extension of middleware concepts to embedded devices. The component-based architecture of the laboratory in conjunction with given timing and safety constraints dictated by the experiments make our infrastructure an ideal candidate for studying system predictability, availability and security in context of middleware-based dynamic control systems. Within this paper we are going to describe our extensible architecture for hosting physical control experiments and focus on Foucault's Pendulum as a case study. For the Pendulum we have implemented a dynamic reconfiguration algorithm, which is able to replace erroneous user-supplied control programs with a verified safety controller at runtime. In addition we are going to discuss the design of custom-built controller hardware which allows us to meet the timing constraints of the Pendulum experiment with a commercial-off-the-shelf (COTS) operating system and middleware. Architectural characteristics of our hardware and software as well as a performance evaluation of the reconfiguration process will be discussed in some detail.*

## 1 Introduction

The 'Distributed Control Lab' currently under development at the Operating Systems and Middleware Chair at the Hasso-Plattner-Institute at University of Potsdam realizes a remotely controlled testbed for interconnected middleware-based components and embedded mobile systems. The primary point of interest is how to reach predictable system behavior in an unstable environment as well as the extension of middleware concepts and platforms to embedded devices.

The job-based environment of our Distributed Control Lab allows the user to conduct experiments from a variety of client devices (see Figure 1). In the context of this paper the term *experiment* denotes the physical installation and its specific control software. The user can write code for a chosen experiment and submit it after successful authentification to the job management system. The experiment management service queues the code as a job object until an instance of the chosen experiment type is available. Once the experiment is completed the user is able to view and analyze results of his control algorithm using our graphical web interface or any other type of client interface (mobile device, command line application). The open architecture of the overall system allows not only several client application types to co-exist but also supports enhancements to examine new research topics.
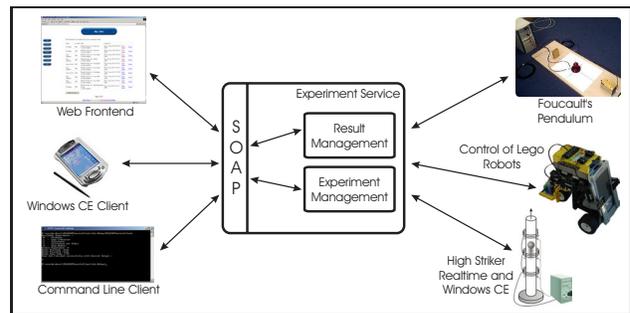


**Figure 1. Distributed Control Lab Overview**

Publishing experiments for varying kinds of remote clients may cause problems related to non-functional application properties such as fault-tolerance, resource usage or real-time constraints. Another problem with the open publication of experiment services, e.g. over the Internet, is that malicious code from unknown sources can possibly damage

physical experiment equipment. We use analytic redundancy to deal with those issues and have implemented several observation techniques to monitor behavior of user code including replacement strategies in case of malicious code. Dynamic reconfiguration is being used as a safe-guard mechanism for code submitted by external users, so that damage to the experiment can be avoided.

This paper concentrates on our Foucault's Pendulum experiment. Within this experiment an user is able to practice writing control algorithms for real-time control problems. The goal of the experiment is to accelerate a iron pendulum with an electro magnet having only information from 2 orthogonal light barriers in the center of the oscillation (see figure 4). Within this paper we also cover some technical details concerning driver and hardware architecture of the pendulum controller. The results of the experimental evaluation as well as the measurement results of reconfiguration times will indicate the appropriateness of our approach.

The remainder of the paper is structured as follows: The subsequent Section 2 describes the management architecture of our *Distributed Control Lab*. Section 3 outlines details of the pendulum control experiment including hardware and driver details. Section 4 covers our approach for online observation and reconfiguration of control applications with stringent safety constraints. Related work is provided in Section 5. The final Section concludes the paper and discusses future work.

## 2 The Distributed Control Lab Architecture

The architecture of our Distributed Control Lab (DCL) consists of a frontend interface for different types of end user applications and the framework management backend part.

The DCL offers Web Services-based (SOAP) interfaces to allow several types of user interface applications to connect regardless of their hardware and software architecture. Until now we have implemented three different user interfaces including a web-based client, a command line client and a graphical Windows CE client.

At the Web Services interface, requests for an experiment usage are abstracted as *jobs*. The internal representation of a job object contains the requested experiment type, a reference to user details and the code to be executed. Our framework ensures the serial execution of jobs on managed experiments. Results of experiment runs are post-processed and stored in the framework.

The DCL backend consists of three main parts. Authentication of users is realized in a separate component - the ticket server. The second component - the experiment manager - queues request for experiment usages from users and manages instances of experiments. A result manager keeps track of execution states of jobs and stores results of experiment runs. The central components of the DCL have been implemented on Microsoft's .NET plattform.
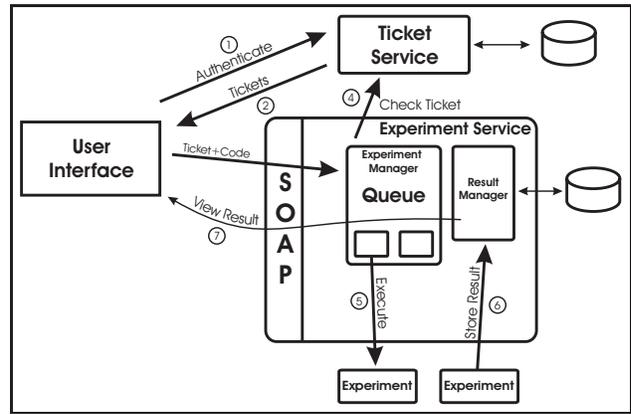


**Figure 2. Message Flow in DCL**

Figure 2 shows the parts of the DCL architecture including timely ordered message flow. Behind the SOAP interface the main components of the DCL perform their communication with .NET Remoting mechanisms. Experiments are attached to the management system by registration, using a .NET Remoting interface.

The three main components will be described in more detail in the following.

### 2.1 Ticket Server

The Ticket Server as part of the DCL management architecture is responsible relating executed jobs with a physical end user. The ticket server implements user authentification mechanisms and will eventually implement other user related functionalities such as billing or role settings.

All jobs in the DCL system are represented by unique *tickets*. All other parts of the DCL architecture (Experiment Controller, Experiment Manager, Result Manager) use those tickets. The decoupling of the user identity offers a flexible solution since authentification mechanisms and functional extensions are independent from the rest of the system.

Every user in the DCL system can own a list of tickets. Each ticket represents an experiment job that was queued by a particular user. For every start of a new experiment run he needs a fresh ticket, which is returned and added to his list if the authentification was successful. The retrieval of the list of allocated tickets is also protected through authentification. The typical steps in an experiment run are illustrated in figure 2 :

1. The user authenticates himself in the DCL frontend, which sends the data to the ticket server. This happens when a user starts a new experiment (new ticket required) or if the user wants to see his current list of queued experiments (ticket list required).

2. After a successful authentification the ticket server sends the requested information back to the frontend.

3. Based on this information the frontend is now able to use the *Experiment Service* SOAP interfaces for the chosen action.

4. The *Experiment Service* checks the given ticket for its validity, e.g. the ticket expiration status. This could also be used for other kinds of external user restrictions.

The list of tickets is hold in a database table. Data integrity is assured through regular database mechanisms (transactions, backup functions). There are two relevant tables, one with the tickets already allocated and another table with the possible users.

The user authentification is performed through checks to a Windows 2000 Active Directory [2]. This prevents the DCL architecture from managing the user accounts dedicated from the surrounding authentification environment (in our case the DISCOURSE VPN network [1]). This would otherwise lead to the usual problems of duplicated user databases (e.g. password synchronization). In case of an authentification request the ticket server first looks into his user table to check if this user is generally allowed to work with the DCL. In the positive case the password is checked against the Active Directory and the requested data is returned.

## 2.2 Experiment Manager

The *Experiment Manager* is the central component of the DCL management backend. It manages experiments and serializes access to them. Single instances of experiments can be grouped to allow load balancing for the experiment instances. Groups of experiments are known as experiment types. Experiments have to register at the *Experiment Manager* and are able to set experiment details, e.g. the time that is needed by the experiment to recover to its initial state between two job runs. When a job request arrives at the *Experiment Manager* and an experiment of the requested type is available the code of the job is send via the interface *IDCLControl* to the experiment. *IDCLControl* contains methods to download code, start, stop experiment execution and methods to query state information of the experiment. If no experiment is available the request is queued as a job in a per experiment type data structure. The code will be compiled as part of the physical experiment. In case of compilation errors compiler messages will be send to the result manager and can be viewed by the users. After successful installation on the physical experiment the experiment execution is started.

The execution results, recorded during the experiment, will be finally send to the result manager. Runtime error messages are stored and provided to the user.
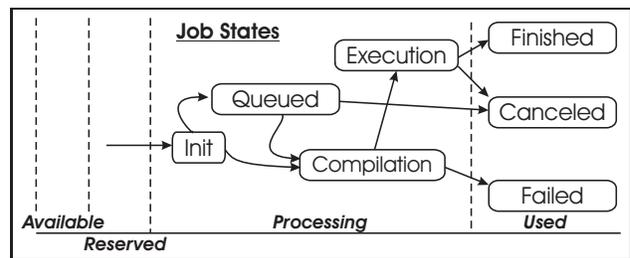


**Figure 3. Job States in the DCL**

Figure 3 shows the states a job cycles through.

## 2.3 DCL Experiment Controller

An *Experiment Controller* manages one physical experiment. At startup it registers itself at the *DCL Experiment Manager* passing its logical name, experiment type and an *Unified Resource Locator* (URI) as unique identifier for .NET remoting calls.

Within the *Experiment Controller* user programs are executed and observed for correct behavior. In order to prevent malicious code from damaging our physical equipment several mechanisms for control of user programs are applicable. The mechanisms can be grouped into categories. Each category can be applied at certain times during the experiment execution - some of them before code is compiled, others during runtime of the experiment.

### 2.3.1 Code-based security enabling methods

Before a user program is compiled it can be checked for dangerous code segments. In most cases a lot of errors can be found using this methods, but e.g. pointer operation in C-style languages are still a problem. This can be solved by a limitation of language facilities, since the prohibition of pointers and recursion enables a better pre-compilation check of user code. We have implemented and integrated a language framework called *Robot Control Language (RoCoL)* to program Lego Mindstorm Robots. Using a special syntax a lot of potential user code errors can be found in advance.

### 2.3.2 Simulation of experiments

Before the execution on the physical hardware an experiment could be run on a simulator. We have implemented simulator for our pendulum experiment, but an exact simulation of the magnetic field of the coil is very complex. For the Lego robots we also have implemented a simulator and are able to recognize in advance if the robot leaves a defined operation area. The simulation approach looks very promising and will be focused in future work.

### 2.3.3 Runtime-based security and fault tolerance methods

The most powerful method is the runtime observation of user control algorithms. In case of abnormal behavior of user code it can be replaced by a well-known verified safety controller. We have developed a framework for dynamic reconfiguration that is able to realize such a replacement. This technique will be discussed in more detail in section 4. The reconfiguration times realized with our framework are satisfactorily enough to allow valuable runtime security checks.

## 2.4 Result Manager

The *Result Manager* stores status information and results from experiment runs in the file system and a database. Results can be as simple as console outputs of the result of a calculation but can also be complex state flow recordings or videos recording during experiment execution. The result manager handles all of them as simple byte streams, whereas the frontend of the DCL is responsible for generating viewable output. The result manager also keeps track of the states of experiment runs. A low priority thread thread writes these internal structures to a database to be able to recover the management service in case of faults.

## 2.5 Frontends for the DCL

The open frontend interface of the DCL, based on the widely adopted SOAP standard, facilitates several types of client applications. Depending on the properties and technical restrictions of the client regarding to graphical, computational and bandwith capabilities there should be always an appropriate display of job submission interfaces and experiment results.

The primary frontend is realized as an ASP.NET web interface. ASP.NET is a web application environment based on the .NET framework by Microsoft. Because of its seamless integration of web services functionality it can act as powerful client for our DCL management architecture. The actual version contains several features :

- Overview and detailed informations for all experiment types

- Live movie stream from the experiment area for immediate user feedback

- Support of incremental code development for the experiment, alternately usage of existing example code snippets

- State informations for already finished or queued jobs

- Result presentation in several ways regarding to the experiment type (data value files, state diagrams, Flash movie)

- Special administration mode for job cleanup and other maintenance tasks

- Screen layout adaption to fulfill differing user needs

As part of a students' project, we have implemented a client for a Pocket PC running Microsoft Windows CE 3.0. This application offers most of the features that are available in the web interface, even with the restricted conditions of the device regarding to screen size, missing keyboard support and networking bandwidth.

Additionally there is a command line based interface with the main purpose to enable the fast and semi-automated submission of many job requests in a bunch. With the help of such an interface the DCL can be tested for robustness and availability in the case of high-load situations.

We are planning to develop more client interfaces on top of other mobile hardware through the usage of SOAP capable frameworks like the Java MicroEdition for mobile phones. Ongoing work concentrates on support of live video on Windows CE devices.

## 3 Pendulum Control Experiment

This section will cover one of the first experiments installed in the DCL. It is a pendulum control experiment, following the idea of a Foucault's Pendulum [5, 4]. The experiment consists of a pendulum with an iron ball swinging over an electromagnet. Two orthogonal light barriers provide information about the balls position. The goal of the experiment is to switch the magnet on or off at the correct moment in order to keep the pendulum swinging. Users can practice to find algorithms optimized for minimum energy consumption or accelerating to a maximum amplitude within a given time. Figure 4 shows a first installation of the pendulum at the HPI. One is able to see the red iron ball, the coil and the laser beams.

In the following part the hardware, driver and control architecture of the pendulum will be explained.

### 3.1 Driver / Hardware Architecture of the Pendulum Controller

In the pendulum experiment feedback is given by two photoelectric sensors using laser beams positioned above the magnet perpendicular to each other. The user program is provided with timing information about when the iron sphere at the end of the pendulum arm enters and leaves the light beams. Exact timing is achieved by using special hardware
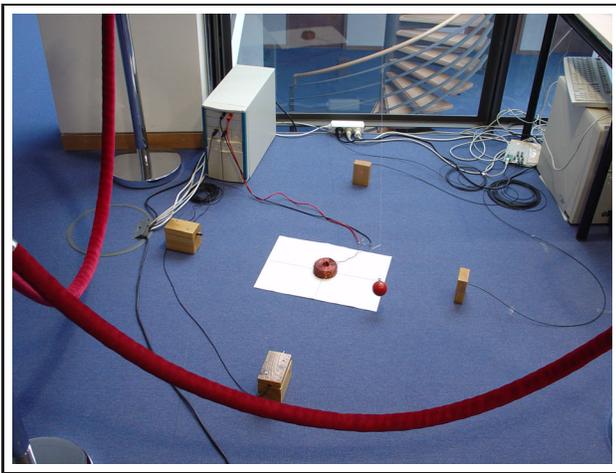
**Figure 4. Pendulum Experiment at HPI**

which samples the light sensors and outputs magnet activity information at a constant rate of 10kByte per second. The device, which is connected via USB, contains FIFO memory which has to be filled (and emptied) by a Windows 2000 user space process. The FIFO size is 1kByte in both directions, so the operating system has to be able to schedule the servicing task at least every 100ms to write (read) 1kByte. This is routinely achieved with a .NET program running on a Pentium III 800MHz Windows 2000 Workstation computer which has no other task to perform. Unfortunately, the FIFO memory introduces a time latency in read and write direction of about 0.1 seconds each. For a real-time control application like the pendulum experiment this is a big disadvantage, because the user program cannot act directly on events received by the photo sensors - the latency is too large. In this case, the algorithm calculates the magnet switching commands for the next pass of the iron sphere.

## 3.2 Controlling the pendulum

```
public class ControlEvent
{
    public int nr;          // sensor or actuator identifer
    public int state;       // actuator 1 ON - 0 OFF
                            // ligth barrier  1 ligth -> dark
                            //                0 dark -> light
    public long timestamp; // global time stamp
}

public interface Pendulum
{
    // Dequeue next event
    // Blocks if no event present until next event occures
    public ControlEvent GetNext();

    // Queue next event to put energy on / off
    public bool SendEvent(ControlEvent input);

    // Get global time stamp 1 micro seconds logical resolution
    public long GetTime();
}
```

**Figure 5. Interface of the Pendulum Controller**

Users can access the pendulum hardware via a special interface, which is provided by the user space process that fills the FIFO memory of the hardware device. The process translates streaming FIFO-data into events, which are stored in a queue and vice versa. In the pendulum experiment events are changes of the light barrier and commands to the electro magnet (on/off).

Figure 6 shows the interface of the pendulum controller. The user of the experiment is able to get events from the light barriers using the *GetNext()* method. The method returns data indicating the source of the event, on/off, and the time the event occurred. Using the function *SendEvent* one is able switch the magnet on and off. A timestamp indicates when the magnet has to be switched. Depending on the control architecture this must be at least 10 ms in future because of the described buffering. The function *GetTime()* returns the current time counter of the system. The pendulum interface provides an easily usable access to the experiment. So users can concentrate on the development of effective control algorithms.

The pendulum experiment controller allows to compile C#-code given by users of the DCL and supports a connection to the physical control architecture. Figure 5 demonstrates the usage of the pendulum interface. The program excerpt demonstrates how the magnet can be controlled using information from the light barriers. The code will enable the magnet for about a quarter second right after the pendulum enters the light barrier.

```
while(true)
{
    // get next event
    ControlEvent ev = pendulum.GetNext();
    if(ev.state == 1) // pendulum enters light barrier
    {
        // switch magnet on
        pendulum.SendEvent(new ControlEvent(ev.timestamp+1000,0,1));
        // switch it off after 5/23 seconds
        pendulum.SendEvent(new ControlEvent(ev.timestamp+6000,0,0));
    }
}
```

**Figure 6. Example: Pendulum Control**

After submission of control code, the user can view a lot results of his experiment run including its output to console and a state flow of important variables.

Figure 7 shows the result of an executed user control algorithm. The blue line represents the ground speed over time during user control. One can see that it increased intensely in the beginning. At the end of time increase of ground speed was weaker. The red line indicates the energy that has been used over time. One can see in the diagram, that the used algorithm constantly accelerated the pendulum with linear energy usage.

The following section covers how a controllable state of the pendulum can be assured at each time, even in the presents of erroneous user code.
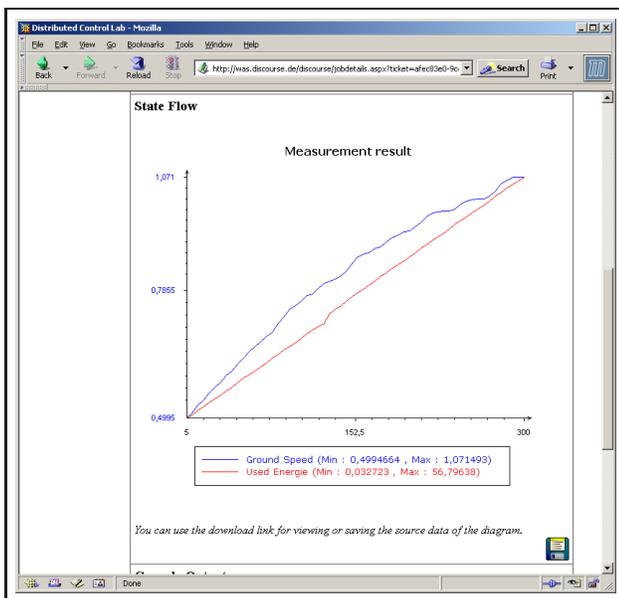
**Figure 7. Results of a Pendulum Experiment**

## 4  Dynamic Reconfiguration for Fault Tolerance and Security

In order to prevent malicious code damaging our experiment equipment, executed user programs are monitored during their runtime. If a dangerous situation is detected the user program has to be replaced with a verified safety controller which is able to preserve a stable state of the experiment.

The introduced pendulum experiment can get out of control in several ways: If the amplitude of oscillation decreases too much, no correct position information of the pendulum can be delivered, because the ball would block light barriers all the time and could not trigger change events. The control of the pendulum would be lost. In addition too long runtimes of the electromagnet could cause heat damage of physical equipment. The runtime of user control programs must be limited as well because of fairness issues to other users.

The safety controller of the pendulum experiment has to be activated in the following situations:

- the available time has expired

- the pendulum falls below a critical amplitude (energy) - so it could not be kept swinging

- user cancels program execution

- the user program terminates out of any reason

- the available energy has expired

If one of these dangerous situation is detected the user code will be replaced by the safety controller using dynamic reconfiguration.

We have developed a framework for dynamic reconfiguration using a XML-based configuration description languages. Profiles can be defined to map various system conditions and environmental settings to defined application configurations. Our reconfiguration framework is able to change an application configuration during runtime if significant changes are detected. The main part of the framework, the configuration manager is able to monitor environmental setting using reusable observer components. In addition it is able to monitor states of components and is able to detected component crashes. This is realized by the observation of the runtime structure of the component (process,thread).

The used algorithm for dynamic reconfiguration is based on an approach of M. Wermelinger [12]. The algorithms allows for a consistency preserving transition from one configuration of an component-based application to another. In our work we know a configuration of a component-based application to denote the set of its parameterized components and the connections between them.

In order to reconfigure an application, it has to be turned into a reconfigurable state by blocking connections between involved components in a special order. The algorithm is able to replace, remove and add components as well as to change properties of components during runtime. A detailed description of the used reconfiguration framework can be found in [8, 9]. The latter also contains a detailed performance analysis of the framework based on the Microsoft .NET Platform.

Concerning the pendulum experiment the environmental properties that have to be monitored are the pendulums amplitude, used energy and expired time. The state of the executed user program must be observed as well. We have defined an application profile that assigns a configuration of the pendulum control application to these environmental settings. Within this profile the user-code-control has to be replaced if the monitored settings leave a specified range. The range can be expressed by the specification of min/max-values.

We have implemented a safety controller that runs if no user program component is available. The safety controller executes an effective control algorithm that is supposed to be very stable, but not optimized for low energy consumption or fast acceleration.

Figure 8 shows possible configurations of the pendulum control application. *Configuration 1* represents the control by a safety controller that is known to be working safely. In this configuration only the safety controller is connected to the USB-controller.

*Configuration 2* shows a simple possibility of a control application with a user code code control algorithm. Only
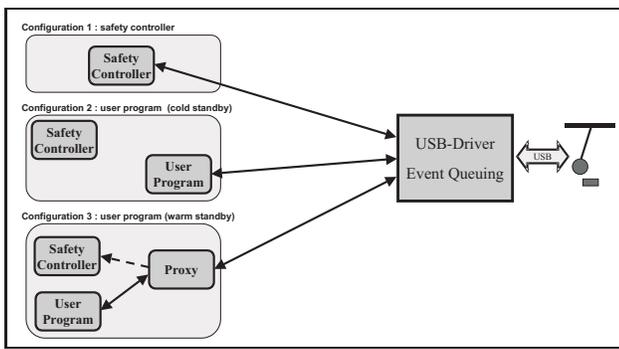
**Figure 8. Configurations of Pendulum Control**



**Figure 9. Histogram of Reconfiguration Times to React on Failed User Programs**

the user program is connected to the hardware. The problem of this configuration is that if the user program fails in some way the time needed to return control back to the safety controller is very long. The safety controller must be started first, has to be connected and restore the current state information of the experiment before it can perform correct control signals.

*Configuration 3* demonstrates a better way for a user control algorithm. The safety controller is still connected (while the user program runs) to the experiment but in a passive mode. A special proxy component forwards only control signals from user code to be send to the experiment but all events from the light barriers are send to the safety controller and the user program. In the case of a failure of the user program the safety controller can immediately take control of the experiment.

All involved components run within separate processes. So component crashes can be detected easily.

### 4.1 Measurements

To evaluate the appropriateness of our approach we have investigated a worst case scenario for the pendulum control application. We measured the time needed by our reconfiguration framework to restore control of the safety controller (producing correct control events) in case the user control component crashes. We realized this scenario by killing the user process at a defined point in time. We used the *high resolution performance counter* of the *Win32-API* for our measurements. Looking back to figure 8 we tested the transition from *Configuration 3* to *Configuration 1*.

Figure 9 shows our measurement results. The diagram shows the measurement distribution of about 1000 measurements. One can see that the take-over takes averagely about 46 ms. The values scatter about 15 ms around this average value. This scatter is caused by the detection of the killed process - the observation of certain values need some kind of polling. The value of 15 ms is exactly the the minimal
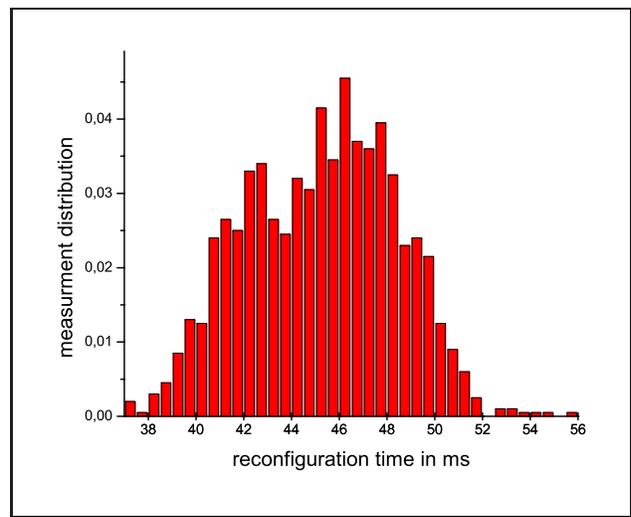
poll time reasoned by the resolution of the system clock.

We have also measured transition from *Configuration 2* to *Configuration 1*. The need to load the safety controller causes additional 700 ms. Furthermore the safety controller would have to measure the current state of the experiment first, which takes at least a complete period of the oscillation of the pendulum.

Our measurements showed that our reconfiguration framework provides a sound foundation for online observation of user control algorithms in our *Distributed Control Lab*.

## 5   Related Work

Web-based eLearning and online laboratories became more and more focus of research in the last years. There are several real world examples available. Most of them concentrate on the provision of experiments with expensive control hardware.

The VVL (german: Verbund Virtuelles Labor) project at University Reutlingen / Germany [11] focuses on an eLearning environment in the context of automation systems. They offer experiment with CAN bus installations, industrial robots and Java-based control logic simulations. The project is primarily intended for automation engineering students and for computer science students.

The Virtual Lab at University of Hagen [10] offers educational experiments in control engineering.The project is a cooperation over three german universities (Bochum, Dortmund, Hagen). At the moment there are several robot based experiments. One of their primary goals is to avoid costly transportation of teaching equipment to the participating universities.

Another example is the iLab project (WebLab) at MIT [7] in cooperation with Microsoft Research allows remote experiments with microelectronic devices. The main purpose is the measurement of the characteristics of those devices. It is possible to access the full programming capabilities of the semiconductor measuring units, voltage measuring units, or voltage source units.

At University of Pisa the Tele-Laboratory [3] offers a graphical user interface for robot experiments. The user accesses the system through a downloadable Java-applet. This applet offers a graphic multi target robot language with a basic instruction set. It acts as an intuitive programming interface for easy access from undergraduate and high-schools.

In general there are a lot of web-based remote lab solutions available. Our work focuses on the interconnection of standard middleware with devices. We concentrate on online replacement strategies in order to realize safety mechanisms in unstable environments.

## 6 Conclusions and Future Work

With the Distributed Control Lab at Hasso-Plattner-Institute at University Potsdam, we are offering an environment for remote experiment access. The software architecture of our DCL is based on commercial-off-the-shelf middleware and operating systems. First experiments have been implemented and their safety against malicious user code has been demonstrated. Within the DCL we have already integrated a robotic experiment and the control of a Foucault's Pendulum. In this paper we have described an online replacement strategy using dynamic reconfiguration as a safeguard mechanism to protect our physical equipment. We have presented experimental results showing the timely behavior of our reconfiguration algorithm in the pendulum experiment.

From our pendulum experiment we have learned, that real-time control applications can be realized using Windows 2000 and the .NET Framework. However, in order to cope with the soft real-time semantics of Windows 2000 and .NET, we had to develop custom hardware and device drivers. Future work will consider porting the pendulum control software to a the Windows CE real-time operating system.

Currently, our Distributed Control Lab can be used from within the Discourse Network [1] - a virtual private network - among the four universities in Berlin and Potsdam. We are going to extend the reach of the network to incorporate additional schools, among them the University of Pisa / Italy. We are constantly adding new experiments to our DCL, actually there is ongoing work concerning the integration of a model-railway installation. We are working together with industrial and research partners to offer a universal test platform for various control algorithms for embedded devices.

## References

[1] DISCOURSE Project. *http://www.discourse.de*, 2003.

[2] Microsoft Active Directory. *http://www.microsoft.com/ad*, 2003.

[3] A. Bicchi, A. Coppelli, F. Quarto, L. Rizzo, F. Turchi, and A. Balestrino. Breaking the lab's walls: Tele-laboratories at the university of pisa. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1903–1908, Seoul, Korea, 2001.

[4] California Academy of Sciences. http://www.calacademy.org/products/pendulum.html. 1999.

[5] Foucault's Pendulum. http://www.abc.net.au/science/k2/-pendulum/pendulum.htm. 2003.

[6] Homepage of Distributed Control Lab. http://www.dcl.hpi.uni-potsdam.de. *Operating systems and middleware chair - Hasso-Plattner-Institute, University of Potsdam*, 2002.

[7] Microelectronics WebLab at MIT. http://www-mtl.mit.edu/ alamo/weblab/index.html. 2003.

[8] A. Rasche and A. Polze. Configurable Services for mobile Users. In *Procceedings of IEEE Workshop on Object-Oriented Realtime Dependable Systems*, pages 163–171, San Diego, CA, Januar 2002.

[9] A. Rasche and A. Polze. Configuration and Dynamic Reconfiguration of Component-based Applications with Microsoft .NET. In *International Symposium on Object-oriented Realtime distributed Computing (ISORC)*, page to appear, Hakodate, Japan, May 2003.

[10] Real Systems in the Virtual Lab. http://prt.fernuni-hagen.de/virtlab/info_e.html. 2003.

[11] Verbund Virtuelles Labor - Automation Systems and Computer Science. http://robo16.fh-reutlingen.de/english/index.html. 2003.

[12] M. Wermelinger. A hierarchic architecture model for dynamic reconfiguration. In *Proceedings of the Second International Workshop on Software Engineering for Parallel and Distributed Systems*, pages 243–254, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1997. IEEE.