# Distributed Control Lab

Andreas Rasche, Bernhard Rabe, Peter Tröger, and Andreas Polze

Hasso-Plattner-Institute, University of Potsdam, P.O. Box 900460,
14440 Potsdam, Germany,

**Abstract.** The Distributed Control Lab (DCL)[10] provides an open infrastructure for conducting robotics and control experiments from the Internet. It is based on web service technologies and offers a wide range of frontend applications. Within the DCL environment we focus on safety mechanisms in order to prevent malicious code from damaging experimental equipment. This includes source code analysis, runtime observation and the dynamic replacement of faulty control algorithms.
Within the paper we present an overview of our architecture and explain the implemented frontends in detail. We introduce the installed experiments and report our experiences from operation in the last years. In addition we will describe our latest integration of grid computing technologies, which allows the outsourcing of computing-intensive simulation tasks.

## 1 Introduction

The 'Distributed Control Lab' developed at the Operating Systems and Middleware Chair at Hasso-Plattner-Institute at University of Potsdam realizes a remotely controlled testbed for interconnected middleware-based components and embedded mobile systems.

The job-based environment of our Distributed Control Lab allows the user to conduct experiments from a variety of client devices. In the context of this paper we use the term *experiment* to denote its physical installation and its specific control software.

Our management architecture includes user authentification, experiment management, result management and the management and queuing of jobs. After a successful authentification at the DCL, the user can submit code to a chosen experiment. Afterwards he can analyze execution results, when the experiment execution has finished. The implemented architecture allows the management of multiple users accessing a variety of experiments, while all users and experiments can be distributed among different locations. Figure 1 shows a general overview of the lab architecture. A variety of experiments have already been integrated, including the control of *Lego Mindstorm* robots, a *Foucault's Pendulum* and the *Higher Striker*. Furthermore a number of different user interfaces have been implemented. There is a ASP.NET web frontend, a command line interface, a Windows CE mobile client and access to the DCL has also been integrated into the *Visual Studio .NET* development environment. Publishing experiments over the

Rasche A., Rabe B., Tröger P. and Polze A. (2004).
Distributed Control Lab.
In *Proceedings of the First International Workshop on e-Learning and Virtual and Remote Laboratories*, pages 150-159
DOI: 10.5220/0001149501500159

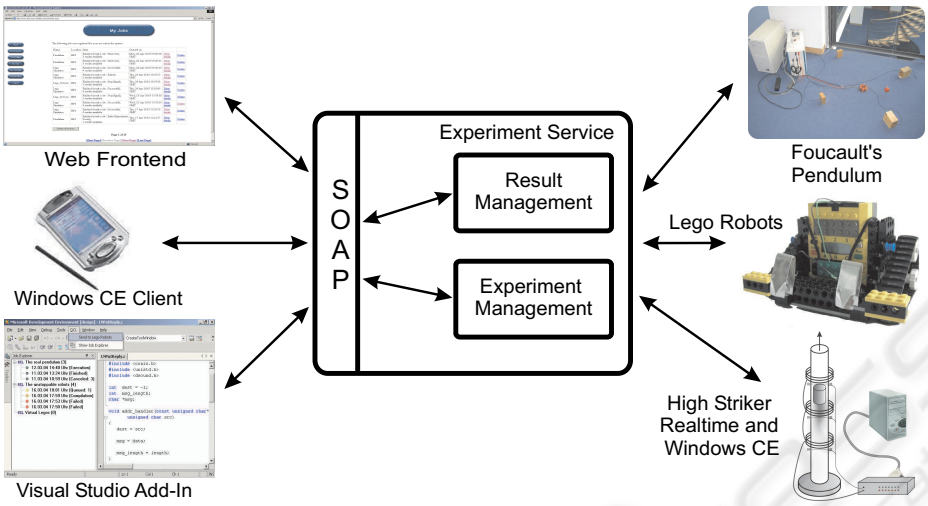


**Fig. 1.** Distributed Control Lab Overview

Internet bears the risk that malicious code from unknown sources can possibly damage physical experiment equipment. Within the DCL, we investigate and implement techniques that prevent malicious code from damaging the experiments. We use source code analysis to detect malicious code. Because not all potential errors can be found, we have also introduced special languages that forbid usage of pointers and recursion, which can easily be used to attack our experiments. In the *Foucault Pendulum* experiment we use analytic redundancy - user control component are observed during runtime. In case of abnormal behavior, it can be replaced by a verified safety controller using dynamic reconfiguration. Details about this experiments can be found in [12, 13].

Another problem with publicly available experiments is the unpredictable number of users accessing a single experiment or even the overall framework. The usual solution for this scalability issue is the simulation of experiment code execution. Users can test their code before submitting it to physical experiment. This leads to an increased need for computing resources. In a consequence the response time of the lab is influenced by heavy usage. Therefore we use grid technologies to compensate these issues by utilizing computing power from grid resources.

The remainder of the paper is structured as follows: The subsequent Section 2 describes the management architecture of our *DCL* including an overview of implemented DCL frontends. Section 3 briefly outlines experiments installed in the Lab. Section 4 presents our simulation infrastructure for the Lab experiments, including details about the usage of grid services. 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 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. The existing frontends for the DCL will be discussed in section 2.1 in detail.

At the Web Services interface, requests for 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 requests for experiment usage from users and manages instances of experiments. A result manager keeps track of execution state of jobs and stores results of experiment runs. The central components of the DCL have been implemented on the Microsoft .NET platform. Figure 2 illustrates central parts of the DCL architecture, in-

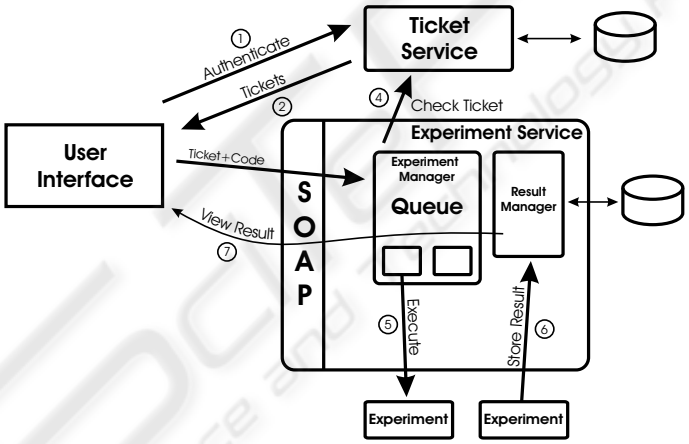


**Fig. 2.** Message Flow in DCL

cluding 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 *Ticket Server* component is responsible for relating executed jobs with a physical end user. 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. Tickets can be acquired from the *Ticket Server*, which checks user data against a Windows 2000 Active Directory [8].

A user can select an experiment and send a valid ticket and the code to be executed to the *Experiment Manager*. If a free experiment instance of the selected type is available, the code will be transferred to an *Experiment Controller* instance, where it will be compiled and installed on the physical experiment. Within the *Experiment Controller*, user code is checked and observed during runtime and in case malicious code is detected, it will be removed. After experiment execution has finished, results are transferred to the *Result Manager*. Experiment results can be acquired via the SOAP interface. Some of our implemented user interfaces are able to visualize this data as diagrams or display video streams recorded during experiment execution.

### 2.1 Frontends for the DCL

The open architecture of the DCL, based on the widely adopted Web Services using the SOAP standard, facilitates several types of client applications. Since the start of the DCL in 2002 various types of frontends were developed, offering different subsets of functionality. At the moment there are 4 frontends (see figure 1): a ASP.NET Website, a command line based client, a Windows CE client and a Visual Studio .NET Add-In. The following Sections will present the important features and requirements of the implemented frontends.

**ASP.NET Web Interface** The first and our primary frontend is realized as ASP.NET interface. ASP.NET is the web application environment of the Microsoft .NET platform. Because of its seamless integration of web services technology it can act as powerful client for the DCL architecture. The actual version contains the following features:

- Overview and detailed information for all experiment types
- Live video stream from the experiment area for immediate user feedback
- Support of incremental code development
- State information of all jobs, including currently running and queued onces
- Job management (cancel, delete, or start jobs)
- Result presentation in several ways regarding to the experiment type (raw data, state diagrams, Flash movie, animated video, detailed error description, compile errors)

**Command Line Client** The command line client frontend is mainly used for performance and load tests of the DCL framework. It can work in an interactive mode with a step-by-step flow and in a silent mode without user interaction. Results and errors are only available as text messages. With the set of call parameters and the silent mode complex batch processes are possible including multiple execution of jobs on a selected experiment.

**Windows CE Client** The the full featured ASP.NET web interface is optimized for desktop computers and could not be displayed by today's Pocket PCs with it's restricted capabilities. Therefore, as part of a students' project, we have implemented a limited feature set for a Pocked PC running at least Microsoft Windows CE 3.0. The mobile frontend was implemented according to the devices' screen size, missing keyboard support and networking bandwidth. Entering program code is simplified by pointing selectable code fragment templates.

**Visual Studio Add-In** In the presence of errors in the source code of a job, it is difficult to manage debug parameter with the introduced DCL clients. The integration of the DCL functionality in the Microsoft Visual Studio .NET 2003 IDE is the latest frontend that also originated as students' project. The important features in comparison with the web interface are:

- A experiment specific structural list of jobs in a tool window
- Display of source code errors in the code editor
- Direct selection of available results
- Familiar environment for developer

The Add-In allows developing experiment programs in a IDE manner, because the well known Visual Studio work flow was kept. After writing the program code the target experiment will be selected (e.g real robot or simulation) and possible errors will be listed in the task list of the Visual Studio. This Add-In combines the functionality of the DCL and the powerful tool chain of the Visual Studio .NET.

## 3 DCL Experiments

### 3.1 Lego Mindstorm Robots

One year ago the control of Lego Mindstorm Robots has been integrated into the *Distributed Control Lab*. The lab user can write C-style programs in order to conduct robot control experiments within a pre-defined operation area. After each experiment run, the robot has to be driven to its charging dock. The operation area is observed by a camera. An image tracking software provides position information of the robot. A safety controller, installed on the robot, is able to find the robot's home and to dock onto the charging ramp. The safety controller is activated as well, if a faulty user control algorithm tries to leave the operation area, or experiment hardware could be damaged.

We have learned from this experiment, that stable environmental conditions must be established. Changing light conditions influence tracking quality and operation area ground properties change results of the control experiments. We have installed the robot experiment in a basement room with constant light conditions, which has no windows.

We also use a special control language that prevents malicious code constructs, which could damage the hard- and software. This language forbids the usage of pointers and recursions. In addition only calls to pre-defined functions are permitted.

A simulator for this experiment has been developed as well. The robot simulator is able to execute programs together for the real experiment and produces a trajectory of the robots way, which can be visualized by some of the DCL frontends.

### 3.2 Foucault's Pendulum

This 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 time in order to keep the pendulum swinging. Users can practice to find algorithms, optimized

for minimum energy consumption or acceleration to a maximum amplitude within a given time.

In order to control the pendulum, a user can write C# programs, which are executed by a commercial-off-the-shelf X86 PC running the *Windows 2000* operating system. Because of real-time constraints of the experiment hardware, which samples the light barriers with a frequency of 23,4 kHz, a custom hardware buffers sampled data, which can be transferred into the PC via the *Universal Serial Bus* interface. A device driver transforms the experiment data into a queue-based structure and is able to generate events for a user control program. The user program itself can switch the magnet by sending events to the device driver.

### 3.3 Higher Striker Experiment

The *Higher Striker* is a hard real-time control experiment. In this experiment a user can write control programs, that accelerate an iron cylinder in a tube of glass, using six electro magnets, which have been placed along the tube. Seven light barriers between the magnets provide position information of the cylinder.

The experiment consists of a custom control hardware connected via the parallel port interface to a control computer, and the physical installation, with electro magnets, light barriers and the described tube. Data from the light barriers and to the electro magnets are sampled with a frequency of 38,4 kHz. A special hardware ensures that the electro magnets can not be activated more than 500 ms to avoid heat problems.

Data from the light barriers and to the magnets can be buffered before they are read or write via the parallel port. The user control program can define how much of the 256 byte buffer should be used. The smaller the buffer size, the faster a control program has to read or write the data. Smaller buffers guarantee faster reaction times onto light barrier events, but demand stricter timing constraints of the control program.

Because of the high speed, the cylinder can reach, a real-time operating system must be installed at the controlling PC. We have already used Windows CE.NET 4.2 [9] and RT-Linux [2] as basis of our control programs. A DCL user can write a Windows CE.NET console application and access the experiment via a low level API, that supports the configuration of the buffer size, and provide an atomic read/write operation, which is needed in order to keep light barrier and magnet data synchronous.

This experiment can be used to practice writing effective control algorithms, because physics behind the movement of the cylinder is quite complex. We are using the *Higher Striker* experiment in our lectures of embedded system development.

## 4 Simulation Infrastructure

The following Section explains our extension of the DCL architecture, in order to allow the utilization of grid computing technologies for computing intensive simulation tasks.

### 4.1 DCL Usage Scenarios

Due to the fact that a possibly high number of client machines may try to work with a fixed set of experiments, we refer to this scenario as *many-to-one* model. We extend

this model within the DCL to a *many-to-many* scenario by using grid computing technologies for simulation purposes. This extension was motivated by the daily operation in the last years. In general we have experienced three practical usage scenarios for our virtual lab:

**few jobs on all experiments** : Most of the time our virtual lab operates at normal load, serving only a small number of users, which work on all kinds of experiments. In this situation the underlying hardware (computers, experimental hardware) works in normal operation mode. All jobs are executed directly after their submission.

**many jobs on one experiment** : This situation occurs in case when a new experiment is introduced to a group of people, for example in the context of a lecture tutorial. The subsequent submission of many jobs by different users leads sometimes to unsatisfying timeouts for particular users. Another reason for high load on experiments is the development of a new algorithm, for example to drive a robot back to its battery-charging base station. In this situation, only one user submits a lot of different jobs in short intervals to a particular experiment. For several experiment types with a long reset time this leads to annoying delays between experiment runs.

**many jobs on all experiments** : Sometimes the DCL has a high overall load, typically because of a preceding demonstration to a larger audience. After such demonstrations many parallel users try to submit jobs to all kinds of experiments. This leads to a maximum workload on the software framework as well as on the experiments. Users have to wait an unsatisfying long time before they can submit their code to an experiment.

To provide a solution for these restrictions, the DCL relies (like many other virtual lab infrastructures) on the simulation of experiments. Simulation can reduce the impact of many parallel users and increases the responsiveness of the overall framework. The simulation can basically substitute the functionality of the experiment, which increases the availability for the overall experiment type. The availability of the physical counterpart can be influenced by several factors: Some of the experiments could have a long initiation phase (robots), others may need a long execution time in general (pendulum). There are also experiments that need periodic breaks during their usage, for example to recharge batteries (robots) or to cool down hardware parts (higher striker). However, the simulation relies mainly on available computing capacity.

A primary constraint for the usefulness of such a simulation is execution time. If the overall execution time of a simulation run exceeds the time on the physical experiment, the simulation gives the user no benefit. Therefore it must be always ensured that the simulation can be executed fast enough. For most institutions in this situation it is not worthwhile to spend a lot money for better simulation machines, since this large computing capacity is only needed several times during high-load phases. The availability of a virtual lab in the Internet makes this kind of load peaks even more unpredictable.

### 4.2 Grid Computing Backend

There exist several approaches in the *grid computing* research field to cope with the demand-driven access to computing resources. On of the primary aims in this area is the transparent offering of resources over platform-independent interfaces [5].

For the purposes of our simulation tasks we consider current computational grid technologies. They allow us an on-demand outsourcing of computing-intensive DCL tasks. The current DCL installation uses the HPI grid testbed as background resource, which consists of nearly 50 machines in a Condor [6] cluster, several Itanium SMP machines and some other heterogeneous systems (Sun, MacOS X, Windows). We made a clear decision not to use proprietary interfaces, we rely on the *Distributed Resource Management Application API* (DRMAA)[11], a standard job submission interface defined by a working group at Global Grid Forum (GGF). Additionally we make use of Globus [4], an open source toolkit for building grid infrastructures. Since the job submission interface of Globus does not rely on standards defined by the GGF, we decided to utilize only resource information interfaces from Globus for our DCL framework.

### 4.3 Simulation in the Grid

If we assume the availability of multiple machines in the virtual lab grid-backend, two usage scenarios can be identified: using idle machines for self-contained tasks, or using idle machines to compute a distributed algorithm.

The first case applies to simulations where the simulated time can be faster than the real time, for example in the Lego robot experiment. We developed a .NET application that takes a user-written program as input and simulates the robot's driving, including the boundaries of the area. This simulation can be executed at the highest possible speed, since no dependencies to real world parameters must be considered. We try to integrate all relevant parameters (like battery charging status or ground quality) into the simulation in order to provide meaningful results for the user. The grid-backend of the DCL takes such a simulation job request, and finds an appropriate free machine for execution. After finishing, the resulting data (e.g. driving route) is transfered back to the DCL server. The grid-backend can assign multiple machines for a parallel execution of different simulation tasks. This could become the case within an overload situation on the overall framework.

The second possibility for grid usage is the distributed execution of an application. We currently use a rendering program for the visualisation of simulation results in the higher striker experiment. The simulator itself runs on the same hardware and operating system as the real controller machine (Windows CE, X86). In case of a simulation run, the control commands for the physical hardware are exchanged with a call to our simulated higher striker hardware. The main reason for this differing architecture are the strict timing constraints in the experiment - it is not possible to ignore influences from operating system and hardware behavior. Therefore it is not possible to build an adequate self-contained simulation. In practical usage this is not a big problem, since the code execution in the entire experiment takes only several seconds.

From the given reasons we decided to use the grid resources in an alternative way here. Our framework uses the output from the simulator in order to render a 3D visualisation of an higher-striker experiment run. We use the popular Povray [1] package for the rendering tasks. The binary is executed in a distributed manner over the available grid resources, simply by segmenting the rendering tasks for the resulting movie.

We experienced in practical exercises that non-technical users, for example high-school students, are very pleased if the user interface depicts the results. Additionally

it is possible to augment the resulting video with additional information, for example power relationships between attraction and rejection of the metal cylinder in the higher-striker experiment.

Both simulation experiments use the grid resources in a similar manner. The Povray rendering tasks as well as the Lego simulation activity are initiated by the experiment controller for a particular user job. The experiment controller uses the DRMAA interface for submitting a grid job, which consists of the binary to be executed and the input parameters. The execution is coordinated by the implementation. It checks the number of available cluster nodes before starting the grid job submission. After finishing, all nodes return their result values to the DCL server, which itself stores them as standard experiment results. The grid infrastructure is responsible for the complete execution of the job and the reliable transfer of input and output files.

## 5 Related Work

Web-based eLearning and online laboratories became more and more focus of research in the last couple of 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 [15] focuses on an eLearning environment in the context of automation systems. They offer experimentation 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 [14] 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 to be carried out. 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 robotic experiments. The user accesses the system through a downloadable Java-applet. This applet offers a graphical 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. In contrast to most of this work, which focuses on particular experiment hardware, our work at DCL addresses software engineering aspects of interconnecting middleware and embedded devices as well.

## 6 Conclusions and Future Work

Within the paper we have presented the architecture of our remote laboratory and introduced available experiments. We have made good experience with the daily operation in teaching and research, regarding availability and safety of our infrastructure. We make use of source code analysis, runtime observation and dynamic reconfiguration to prevent malicious code from disturbing the integrity of our lab.

We have explained the usability of simulation techniques to cope with high-load situations. We utilize grid computing technologies for out-sourcing computing intensive tasks. Currently we have already used this approach within two experiments: the simulation of robot movement and the graphical rendering of simulation result.

In the future we plan to extented our lab with new experiments, as well as new frontend applications. We are going to integrated experiment from geographically distributed locations. In addition we will apply latest web service standards, such as WS-Security, to improve the interoperability of our frontend interfaces.

## References


1. Persistence of vision raytracer. http://www.povray.org, 2004.
2. Michael Barabanov. A linux-based realtime operating system. Master's thesis, New Mexico Institute of Mining and Technology, 1997.
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. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
5. Ian Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.
6. M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, pages 104–111, 1988.
7. Microelectronics WebLab at MIT. http://www-mtl.mit.edu/ alamo/weblab/index.html. 2003.
8. Microsoft. Microsoft Active Directory. *http://www.microsoft.com/ad*, 2003.
9. Microsoft. Windows CE.NET 4.2. http://msdn.microsoft.com/embedded, 2004.
10. OSM Group. The Distributed Control Lab. http://www.dcl.hpi.uni-potsdam.de, 2004.
11. Hrabri Rajic, Roger Brobst, Waiman Chan, Fritz Ferstl, Jeff Gardiner, Andreas Haas, Bill Nitzberg, and John Tollefsrud. Distributed Resource Management Application API Specification 1.0. http://forge.ggf.org/projects/drmaa-wg/, 2004.
12. Andreas Rasche and Andreas Polze. Configuration and Dynamic Reconfiguration of Component-based Applications with Microsoft .NET. In *International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, pages 164–171, Hakodate, Japan, May 2003.
13. Andreas Rasche, Peter Tröger, Michael Dirska, and Andreas Polze. Foucault's Pendulum in the Distributed Control Lab. In *Procceedings of IEEE Workshop on Object-Oriented Realtime Dependable Systems*, pages 299–306, Capri Island, Italy, October 2003.
14. Real Systems in the Virtual Lab. http://prt.fernuni-hagen.de/virtlab/info_e.html. 2003.
15. Verbund Virtuelles Labor - Automation Systems and Computer Science. http://robo16.fh-reutlingen.de/english/index.html. 2003.