Predictable Interactive Control of Experiments in a Service-Based Remote Laboratory

Andreas Rasche
Hasso-Plattner-Institute
University of Potsdam
Potsdam, Germany
andreas.rasche@hpi.uni-potsdam.de

Frank Feinbube
Hasso-Plattner-Institute
University of Potsdam
Potsdam, Germany
frank.feinbube@hpi.uni-potsdam.de

Peter Tröger
Blekinge Institute of Technology
Ronneby, Sweden
peter.troger@bth.se

Bernhard Rabe
Hasso-Plattner-Institute
University of Potsdam
Potsdam, Germany
bernhard.rabe@hpi.uni-potsdam.de

Andreas Polze
Hasso-Plattner-Institute
University of Potsdam
Potsdam, Germany
andreas.polze@hpi.uni-potsdam.de

ABSTRACT

Remote and virtual laboratories are commonly used in electronic engineering and computer science to provide hands-on experience for students. Web services have lately emerged as a standardized interfaces to remote laboratory experiments and simulators. One drawback of direct Web service interfaces to experiments is that the connected hardware could be damaged due to missed deadlines of the remotely executed control applications.

Within this paper, we suggest an architecture for predictable and interactive control of remote laboratory experiments accessed over Web service protocols. We present this concept as an extension of our existing Distributed Control Lab infrastructure. Using our architecture, students can conduct complex control experiments on physical experiments remotely without harming hardware installations.

1. INTRODUCTION

The Distributed Control Lab (DCL) is a virtual laboratory at the Hasso-Plattner-Institute in Potsdam, which enables the remote usage of experiments for teaching purposes. It comprised an infrastructure that provides access to remote real-time control experiments via the Internet. Authenticated users can submit control programs for an experiment by the help of different front-ends, such as the Web interface, a development environment plugin or a command-line interface. Each control program by a particular user is called a *job*, which is executed by a matching *experiment controller* that steers the according physical experiment hardware. Students usually implement control algorithms, which process data from physical hardware installations and generate according control signals.

The DCL infrastructure is responsible of distributing incoming jobs to available experiments. Multiple experiments of the same type, being able to handle the same kind of job, are called *experiment types*. The infrastructure supports both physical experiments and simulations for the same experiment type. Simulations are intended to help out in case of high load on experiments, e.g. before a student assignment deadline. Simulations can act as full replacement for the real experiment in most cases, since students submit most of their jobs for checking the correctness of their control application. This only demands mainly a compiler run for the control program in the particular runtime environment, but not a real execution of physical activities [10, 14].

Beside the support for teaching activities, research in the DCL project covers the question of protecting the infrastructure against malicious code, which can potentially harm experiment hardware or execution nodes. It utilizes techniques such as automated source code analysis, run-time monitoring and dynamic adaptation for protecting the experiment infrastructure [12].

Within the DCL several experiments have already made accessible. Among others, there are the control of Lego Mindstorm NXT robots, the control of a Foucault-style pendulum, a model railway track and an industrial control scenario. Further details on the existing DCL experiments can be found in [13, 14].

The DCL provides a batch mode experiment interaction style. The user implements a control algorithms and submits the source code to the lab infrastructure. The DCL than identifies a free experiment instance, compiles the provided source code and runs the control program on the particular experiment. After successful execution, the user can view results in the form of diagrams, text or videos. Each ex-

Preprint of the ACM version

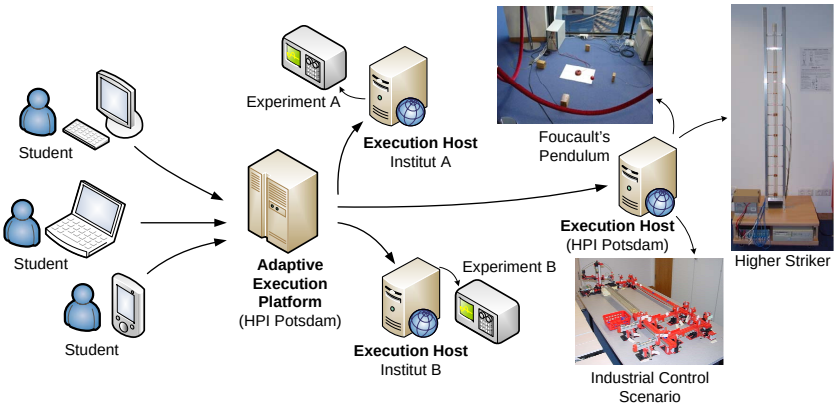

Figure 1: The Distributed Control Lab

periment run comprises only one direct interaction with the experiment. In case of multiple users, the lab infrastructure serializes access to the experiments, potentially queuing up jobs. The extensible architecture of the lab also supports multiple experiment instances of one type, allowing for the load balancing of jobs and the toleration of experiment outages.

In order to be able to integrate experiment installations from different organizations, the latest version of the DCL relies on commonly accepted Web service standards to interconnect different experiment installations. Each experiment therefore implements a Web service interface providing the necessary methods for job execution and monitoring. We rely our architecture on the explicit notion of *logical service instances*, an abstraction concept for the stateful interaction of DCL users with the infrastructure. A logical service instance combines the relevant methods for uploading the job control logic, checking the experiment run properties and for fetching the experiment run results. Each client therefore can operate its own logical instance (or session) for an experiment. We give a short introduction of the underlying concepts in the following paragraphs, details can be found in [16].

In our service-oriented version of a remotely accessible virtual laboratory, each control program for one experiment run is represented as logical service instance. Calls to logical instances from a DCL user are dynamically mapped on *physical service instances*, representing the end point for either physical or simulated experiment hardware. A logical service instance therefore represents a stateful entity to the client, but does not necessarily need to be realized by only one physical service instance on a particular server. All clients communicate with a coordination layer that routes SOAP requests to a matching *execution host*, a computer running the physical service instances. Matching logical and physical instances relate to the same *service implementation*, which is realized as typical binary Web service component (e.g. Java Servlet, .NET Assembly) with its according WSDL information.

In order to create a new experiment run based on some provided control logic, the DCL user contacts a *factory service* to get the reference to a new logical service instance. The logical instance is represented as SOAP endpoint, described by a *WS-Addressing* [5] XML document. A logical service instance has query-able state properties, which can be read and written through standardized SOAP operations [4].

The set of existing logical service instances is managed by the *coordination layer* in the DCL, which schedules and manages the incoming requests on the matching physical service instances. These physical instances share a common state storage, which allows to route requests to the same logical instance to different physical instances of the same implementation.

If the client queries its logical instance for some current attribute value, the coordination layer can provide the latest data made available by the experiment service out of the central database. The data is written during the experiment run by the particular physical instance.

Figure 1 gives an overview of the service-based DCL architecture. Students already can use laptops, PCs or even mobile devices with Windows CE for accessing the lab experiments. The coordination layer routes service request from clients to the execution host containing the physical services. The physical services then runs the provided control program on the experiment hardware.

2. INTERACTIVE EXPERIMENT USAGE

Both the classical and the service-oriented version of our DCL infrastructure assume a batch processing mode, where each experiment run is performed independent from the client. In a single step conducted before the experiment run, control logic is transmitted to the experiment and/or experiment-specific parameters are configured. Because of this basic architectural style, it is not possible to interact with the experiment while a control algorithm is executed. This prevents the interactive access to measurement values and experiment results while the experiment is running. Even though this strict decoupling enabled straight-forward fault tolerance and scalability features, it also prevents typical interactive usage patterns, especially the debugging or fine-tuning of experiment-parameters during the execution of a job by a lab user.

Our primary objective for supporting an interactive mode was the usage of standard development tools such as LabVIEW, Eclipse or Visual Studio in the existing DCL infrastructure. One example is an experiment setup [7] at Technical University of Darmstadt, where FPGA boards can be programmed remotely. After burning a new FPGA image, test sequences are sent to the board and output data is sampled and transferred to the lab user. This transfer of test samples also requires support of an interactive mode. Within the Leonardo Da Vinci Project *Vet-TREND*, we needed to integrate this experiment in the DCL infrastructure.

Another example of an experiment is the control of moving robots, as already provided in our lab. In the so far applied batch mode processing of requests, a control program can be uploaded to the experiment and controls the robot from there on autonomously. For an updated experiment scenario, it might be interesting to process control commands from the user while the robot is already moving. This would allow students to implement a control algorithm for the robot and steer it afterwards remotely.

In order to support an interactive mode, the original service architecture had to be extended slightly. We introduced a special class of *interactive logical instances*, where the infrastructure binds the particular logical instance to a physical one, avoiding the dynamic routing of requests. We also added support for *asynchronous logical instance invocations*, in order to allow the parallel execution of an experiment run – represented by a call – and querying of values – also represented by a call.

In the classical batch mode operation, experiment jobs are queued if no free physical instance (meaning experiment) is available. This represented a feasible approach, since the duration of single jobs is typically limited and result are analysed after the finished execution of a job. In the interactive mode, end users are directly involved in the execution of jobs. This prolongs the execution time of jobs and makes queuing and the calculation of queuing times more difficult. We therefore changed to an external time-slot-based reservation system, as already suggested by related work in the area of virtual laboratories [17, 2]. The user registers for a time-slot through a separate interface. The reservation is persisted, and every time a logical instance operation is invoked, the infrastructure checks if the authenticated user is allowed to use the bound physical service instance. The reservation and accounting is handled externally and can be configured e.g. by a teacher.

Figure 2 illustrates the steps happening when an interactive job is executed. After the experiment service implementation has been deployed and configured (step 1), the user registers for a time-slot for that experiment. In the third step, the user creates a logical service instance for an interactive experiment, opening his own session with the physical experiment in an indirect manner. Afterwards, in step 4, the client sends an experiment-related request to the logical service instance. The coordination layer checks (4.1) if the user is authorized to perform that action. In the case of an unauthorized request, the coordination layer responds with a SOAP fault reply message. If a user has a valid reservation

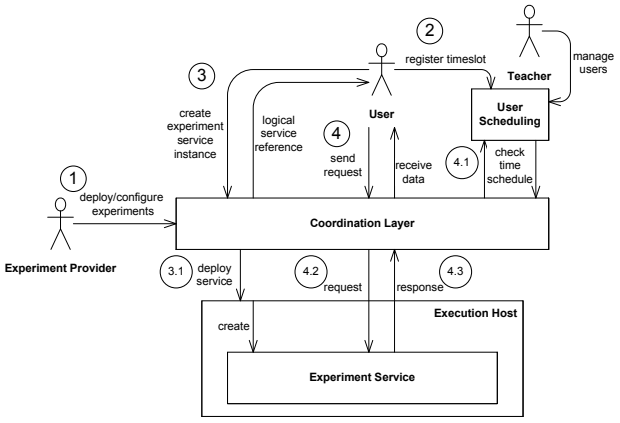

Figure 2: Interactive mode: usage scenario

for the interactive experiment, it might still be needed to deploy the service implementation to the execution host in order to get a physical instance. This shows that even with interactive logical instances, our basic concepts of dynamic resource usage still hold. During the reserved time slot, the particular physical instance is exclusively reserved for one user. Requests for the same experiment type by other users are handled by other physical instances for the same type, if available.

The described interactive mode of our infrastructure can be used for Web-service-based interactions, but also custom interaction protocols might be used. During the reserved time slot, no other user can interact with the physical service instance for the experiment, since all requests are routed through the coordination layer. Alternative protocols then can perform some kind of 'out-of-band' connection to the experiment, like with a native debugging session. Since there may be multiple execution hosts available for the experiment type, the target addresses for the custom protocol must be configured dynamically. The host address of the used experiment host can be acquired by a dedicated logical instance operation. In order to prevent other users from connecting to the experiment computer via the custom protocol, a small proxy could be implemented on the infrastructure management hosts that simply forwards the request to the experiment computer and additionally checks again the reservation database.

Beside our realization of proprietary protocol access, there is also support for a flexible SOAP-based interaction between user and experiment. Two possibilities for the service interfaces exists: On the one hand a generic interaction interface, illustrated in listing 1 can be used. The generic methods *SendData* and *ReceiveData* allow to to transmit experiment specific data, such as proprietary monitoring values.

```
interface IInteractive{
 void SendData(Byte[] data);
 Byte[] ReceiveData();
}
```

Listing 1: Generic interface

On the other hand, experiment-specific SOAP operations can be used. One example is shown in listing 2. This interface is used to control one of the robots in our laboratory environment interactively. Depending on the invoked operation, the submitted control logic must make the robot move in a correct way.

```
interface IRobot{
 void Move(int speed);
 void Turn(Angle degree);
 void Stop();
}
```

Listing 2: Special interface

With an interactive mode in place, users are now able to influence a running experiment. This introduces a new problem class regarding the timing behaviour of experiment runs, which is discussed and solved for our infrastructure in the following section.

3. REAL-TIME WEB SERVICES

When steering real-time experiments through remote interface technologies such as SOAP, the problem of *timelininess* must be solved. The miss of deadlines in hard real-time experiment could potentially damage expensive hardware installations. One example is the control of a moving robot - a missing command to stop the robot in front of an obstacle could lead to a sub-optimal experiment result. Another example is the higher-striker experiment operated in the DCL, where students have to enable magnetic coils connected to a car battery to accelerate an iron cylinder. Enabling the coils for more than 200 milliseconds can burn the wires and therefore render the whole experiment unusable.

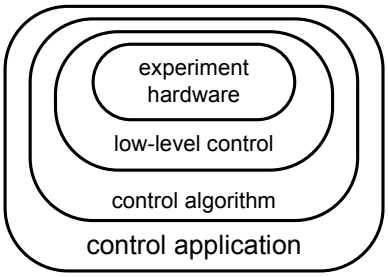

Figure 3: Typical experiment control

Figure 3 shows the typical control setup for a remotely accessible experiment in a batch-processing approach. The control application is executed on the experiment control computer (called execution host in our infrastructure). The hardware can be accessed directly by executing the according I/O instructions in the control program. The execution behaviour of the control application can be determined in advance, since all relevant parameters are known. The experiment execution is decoupled from external influences, which allows to keep all relevant deadlines given by the physical circumstances of an experiment. As already argued, a drawback of this decoupled approach are restrictions for debugging and monitoring.

In order to use the features offered by modern development tools as presented before, at least some parts of the experiment control logic have to be executed on the user's computer. This leads to the notion of a distributed application, where the control flow is partitioned between the users computer and the experiment computer. The part running on the user's computer interacts with the application part residing on the experiment computer. When analysing timing requirements, a new factor has to be considered now: the communication within the distributed control application.

The problem turns out to be even worse with the usage of Web service technologies and wide-area Internet connections, due to their unpredictable timing behaviour. In the Internet different load situations can be found, making a precalculation of transmission times difficult. Also the processing of XML messages in the Web service stacks can lead to unpredictable timing behaviour within the distributed control application. The usual approach is the introduction of an according fault tolerance mechanism, which reacts on timing faults caused by the usage of unpredictable communication protocols. The experiment defines the maximum time between two Web service invocations to ensure a safe control of the experiment. In case the timing requirements are not hold - a deadline is missed - a fault handling mechanism must be activated.

We propose a new approach for preventing missed deadlines in such a scenario, by applying the *composite object pattern* first introduced by Polze et al. [9, 8]. The composite object pattern decouples unreliable communication mechanisms from hard real-time applications. By separating real-time and non-real-time activities within an object, all timing requirements of the real-time activities can be hold due to prioritization. In addition non-real-time activities are observed and in the case of a missed deadline, a default handler residing in the real-time part can tolerate timing faults. The composite object pattern has been originally developed for the CORBA middleware, and is applied on interactive experiment Web service in the context of our DCL infrastructure.

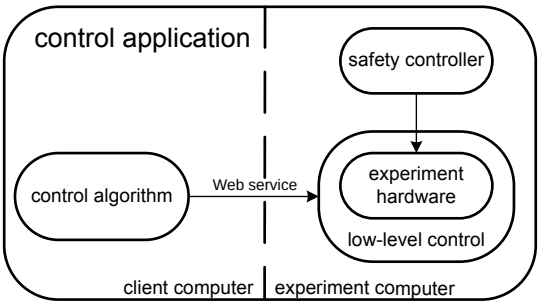

Figure 4: Distributed control infrastructure

Figure 4 shows the architecture of a distributed control application when applying the pattern. The application part running on the user's computer controls the experiment by invoking Web service operations on the experiment computer. On the experiment computer, a safety controller is executed in parallel to the low-level control application part. The safety controller observes the fulfilment of the given real-time requirements for the Web service invocations. In

case a deadline is missed, the safety controller reacts using default actions to stabilize the experiment. The important aspect here is the decoupling of the non-real-time Web service interface from the real-time low-level I/O control and the safety controller process.

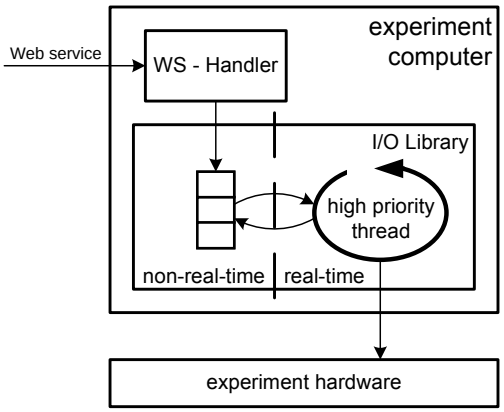

Figure 5: Composite objects and safety control

Figure 5 shows the resulting architecture of a timing-fault resilient Web service implementation. A Web service handler processes the incoming service operation request. Instead of directly interacting with the hardware, control commands are then written into a shared memory buffer by the handler implementation. A second real-time priority thread cyclically reads the commands from the buffer and interacts accordingly with the physical experiment installation. It also reads sensor data from the hardware and writes them into the read buffer of the Web service implementation, which enables the retrieval of sensor values for the client without disturbing the timing behaviour of the overall experiment.

The important issue in our approach is to isolate the deterministic real-time (rt) part from the non-deterministic non-real-time (non-rt) part in the activities triggered by a client request. A technique proposed by one the original authors [8] is the usage of a horizontal timing firewall. The technique relies on the usage of a real-time operating system and uses two classes of thread priorities for non-rt and rt-tasks. By using higher priorities for the real-time threads the isolation can be realized. Another approach is to use a multiprocessor system (SMP) and use a dedicated CPU for real-time activities. This technique has already been implemented in the form of CPU-Shielding [3] in a real-time version of a Linux operation system.

For our remote laboratory environment, we use the the real-time extension for Linux (rt-Linux) [1]. Non-RT activities are executed as normal Linux processes being allowed to access all functionality available in the operating system. Real-time activities are executed in a real-time kernel module with restricted functionality. The synchronization between both parts is implemented via a shared memory region. In a second future scenario, we will use the Windows CE real-time operating system and implement a horizontal timing firewall. We are therefore porting parts of the DCL infrastructure to the Windows Mobile 6.0 R2 operating system, in order to run execution hosts on RT-enabled operating systems.

Depending on the requirements of the lab experiments different consistency protocols for buffer synchronization can be used in the implementation. In the real-time Web service scenario three types of variables are used. There are *non-shared real-time variables* only used by rt-tasks. These variables must be pinned in memory and ensured to not being swapped out to secondary storage by the operating system. *Non-shared non-real-time variables* are only used by the Web service handler and may be swapped out. The most important type are *shared variables*, which are used by real-time and non-real-time tasks.

Shared variables can be synchronized following a strong consistency protocol. This requires the synchronization of real-time and non-real-time tasks via common synchronization primitives. To avoid the priority inversion problem, a priority ceiling protocol is typically used. Since the access time to data by non-real-time threads is restricted, time spend in such a critical section must be calculated when analyzing the feasibility of the real-time system.

Among others, we are using .NET- and Java-based web service implementations. This leads to a problem when calculating worst case execution times of non-rt critical sections, since the garbage collection could stop the whole application any time - causing unpredictable delays. For this reason, we rely on weak consistency protocols instead to implement shared variables, where they are handled as real-time variables with a shadow copy accessed by non-real-time tasks. The values of the real-time variables are copied with configurable interval timings to the non-real-time shadow buffer. This also means that the copy temporarily diverges. This is no problem, since the copy interval can be configured as being much smaller than the arrival rate of Web service requests. In many configurations, the usage of old data is also no problem. On example is the control of a moving robot, where a late move right command does not automatically expose damage to the robot. In a dangerous situation, the safety controller can still take over control.

One advantage of the proposed solution is that it can also be used to tolerate incorrect computation and even byzantine and omission faults. One example are erroneous or even malicious control algorithms implemented by students. The safety controller component is not implemented by a lab user, but by the operating authority. In this case, the safety controller is denoted to be *analytically redundant* to the user control component. The safety controller therefore solves a common control problem with restricted, but proven functionality. More details on the efficient dynamic reconfiguration of control applications can be found in [14, 11].

4. APPLICATION SCENARIO

Figure 6 shows an example application of the proposed concept on one of our DCL experiments. The Fischertechnik assembly line shown on the picture is a model of a ball sorting assembly line, consisting of rotating conveyor belts, pneumatic actuators and proximity sensors. The assembly line can be controlled by a Beckhoff programmable logic controller (PLC) running on top of the Windows CE operating system.

Within the existing DCL experiment, students can imple-

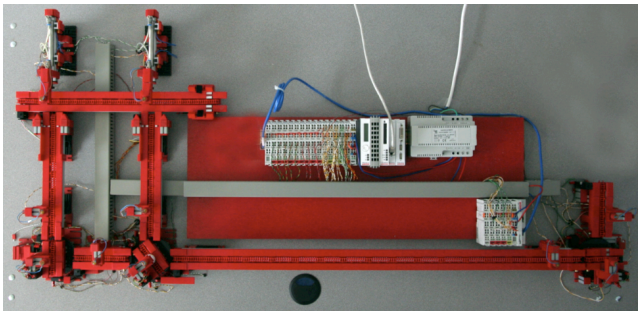

Figure 6: DCL experiment: Fischertechnik assembly line

ment .NET-based control applications for accessing the PLC's I/O-modules. The .NET code is executed directly on the PLC using Microsoft's .NET Compact Framework.

For a new version of the experiment, we want students to use TwinCAT, a PLC standard development tool, to implement control applications using a typical IEC-61131 programming language. TwinCAT (figure 7) requires multiple interactions with the experiment controlling PLC during the execution of a control application. This is due to debugging, but also for the configuration of the hardware mapping. The experiment has both hard real-time requirements and the demand for an interactive control. If the conveyor belts are not stopped in a correct timely manner, the balls could be pushed from the table by the pneumatic ram.

We applied the described concept for the interactive mode by running a PLC-based safety controller in a high priority task. It observes missed deadlines by the users' control application, as well as faulty behaviour by observing external experiment parameters such as the position of the balls and the speed of the conveyor belt. First experiments show that the extended experiment controller architecture fits into our service-oriented remote laboratory infrastructure, while ensuring that the real-time experiment hardware is not harmed by incoming control programs.

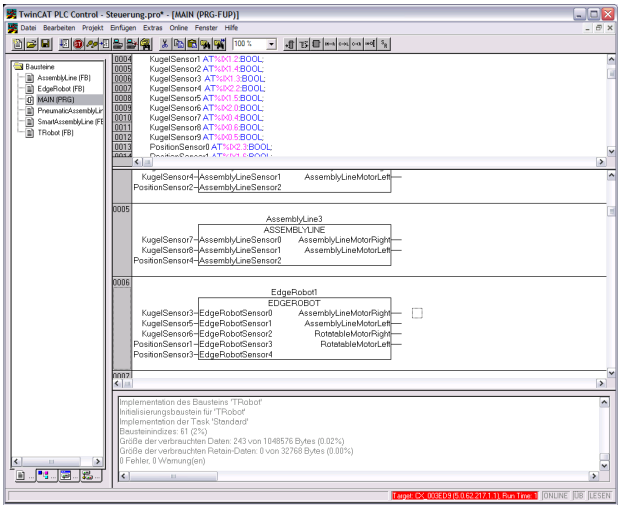

Figure 7: Standard PLC development tool

Another experiment we plan to implement is the control of Lego NXT robots. The user can upload a control application, which reacts on steering signals generated through calls on an interactive logical instance. If the robot leaves the pre-defined operation area the safety controller takes over control and moves the robot to its charging station. The uploaded control application must implement the interactive commands for the Lego, using a Bluetooth connection from the experiment host to the Lego robot.

5. RELATED WORK

The management of timing properties for real-time experiments is based on the composite object pattern first used in the RemoteLab [15], which was a cooperation between Humboldt Universität zu Berlin and the University of Illinois Urbana-Champaign. Within the lab a Khepera robot was controlled using the CORBA middleware. Real-time behaviour of the robot was ensured by using the composite object pattern.

An interactive mode for virtual/remote laboratory experiments has also been implemented in other remote lab environments, such as the MIT iLab project [6]. Within iLab an open source framework was created, providing common functionality for the operation of a virtual/remote laboratory. iLab relies on a three-tier Web architecture including client applications, intermediate service brokers and lab servers. iLab also used Web services for communication between the tiers, but in contrast to our work there is no integrated session concept for accessing and checking an experiment run. iLab's interactive mode gives the frame for common services such as distributed logging and access control, but leaves the protocol used for interaction as experiment-specific. This paper suggests an interaction protocol based on Web services or tool-specific protocols, but concentrates on the assurance of real-time requirements.

6. CONCLUSIONS

The contribution of this paper is a new architecture for providing interactive and real-time control of experiments for non-deterministic communication middleware such as Web services. The usage of analytic redundancy allows to tolerate timing faults raised by the communication layers. This allows users of remote laboratories to execute control applications interactively, for example with debug and monitoring features, while still getting the advantages of modern service-based access mechanisms. The proposed architecture also enables the usage of standard development tools such as LabView directly for the remote laboratory users.

The proposed extension of the Distributed Control Lab, operated at Hasso-Plattner-Institute, are currently in use for different teaching and research demonstration activities. Future work will concentrate on the further investigation of real-time experiment integration in a dynamic service-oriented remote laboratory.

7. ACKNOWLEDGMENTS

This work has been sponsored by the Leonardo Da Vinci Programme of the European Union within the VET-TREND-project (RO/06/B/F/NT175014).

8. REFERENCES

[1] M. Barabanov. A Linux-based Real-Time Operating System. Master's thesis, New Mexico Institute of Mining and Technology, 1997.

[2] M. Basso and G. Bagni. ARTIST: A Real-Time Interactive Simulink-based Telelab. In *Proceedings of IEEE CCA/ISIC/CACSD Conference*, Taipei, Taiwan, September 2004.

[3] S. Brosky. Shielded CPUs: Real-Time Performance in Standard Linux. *Linux Journal*, May 2004.

[4] S. Graham and J. Treadwell. Web Services Resource Properties 1.2 (WS-ResourceProperties). OASIS Open, April 2006.

[5] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - Core. World Wide Web Consortium (W3C), May 2006.

[6] J. Harward, T. T. Mao, and I. Jabbour. iLab Interactive Services – Overview. http://icampus.mit.edu/iLabs/Architecture, 2006.

[7] L. S. Indrusiak, M. Glesner, and R. A. da Luz Reis. Lookup-based Remote Laboratory for FPGA Digital Design Prototyping. In *Intl. Workshop on e-learning and Virtual and Remote Laboratories (VIRTUAL-LAB 2004)*. INSTICC Press, August 2004.

[8] A. Polze and L. Sha. Composite Objects: Real-Time Programming with CORBA. In *Proceedings of 24th Euromicro Conference, Network Computing Workshop*, volume II, pages 997–1004. IEEE Computer Society, 1998.

[9] A. Polze, K. Wallnau, and D. Plakosh. A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain. Technical Report CMU/SEI-97-TR-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, 1997.

[10] A. Rasche and A. Polze. Configuration and Dynamic Reconfiguration of Component-based Applications with Microsoft .NET. In *Proceedings of the International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, pages 164–171. IEEE Computer Society, May 2003.

[11] A. Rasche and A. Polze. Redac - dynamic reconfiguration of distributed component-based applications with cyclic dependencies. In *Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), to appear*. IEEE Computer Society, May 2008.

[12] A. Rasche, M. Puhlmann, and A. Polze. Heterogeneous Adaptive Component-Based Applications with Adaptive.Net. In *Proceedings of the International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, pages 418–425. IEEE Computer Society, May 2005.

[13] A. Rasche, B. Rabe, M. von Löwis, J. Möller, and A. Polze. Real-Time Robotics and Process Control Experiments in the Distributed Control Lab. In *IEE Software, Special Edition on Microsoft Research Embedded Systems RFP*, pages 229– 235. IEE Proceedings Software, October 2005.

[14] A. Rasche, P. Tröger, M. Dirska, and A. Polze. Foucault's Pendulum in the Distributed Control Lab. In *Procceedings of IEEE Workshop on Object-Oriented Real-time Dependable Systems*, pages 299–306. IEEE Computer Society, October 2003.

[15] J. Schwarz, A. Polze, K. Wehner, and L. Sha. Remote Lab: A Reliable Tele-Laboratory Environment. In *Proceedings of the International Conference on Internet Computing*, pages 55–62. CSREA Press, June 2000.

[16] P. Tröger, A. Rasche, F. Feinbube, and R. Wierschke. SOA Meets Robots - A Service-Based Software Infrastructure For Remote Laboratories. In *Proceedings of the 2nd International Workshop on elearning and Virtual and Remote Laboratories*, pages 57–62. Hasso-Plattner-Institut, Universitätsverlag Potsdam, Germany, February 2008.

[17] M. Wulff, P. Lauer, and T. Braun. Content management and architectural issues of a remote learning laboratory. In *Proceedings of the 2nd International Workshop on elearning and Virtual and Remote Laboratories*, pages 13–19. Hasso-Plattner-Institut, Universitätsverlag Potsdam, Germany, February 2008.