SOA Meets Robots - A Service-Based Software Infrastructure For Remote Laboratories

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

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

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

Robert Wierschke
Hasso-Plattner-Institute
at University of Potsdam, Germany
robert.wierschke@hpi.uni-potsdam.de

Abstract

With the ongoing internationalization of virtual laboratories, the integration of such infrastructures becomes more important. The meanwhile commonly accepted 'glue' for such legacy systems are service-oriented architectures, based on standardized and accepted Web service standards.
We present our concept of the 'experiment as a service', where the idea of service-based architectures is applied to virtual remote laboratories. In our laboratory middleware, experiments are represented as stateful service implementations. We discuss performance, reliability and security in this approach, and show how our solution - the Distributed Control Lab - is applied in the European VetTrend project.

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. 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.
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 [4, 6].

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 [5].

Within the DCL, several real-time control experiments have already been integrated. Figure 1 gives an overview of some experiments connected to the DCL. Foulcault's Pendulum as an experiment imitates Leon Foucault's famous experiment for measuring the earth rotation. An iron ball is used for the pendulum that can be accelerated using an electro-magnet connection to a control-PC via USB. Two orthogonal laser-based light barriers provide position information about the swinging ball. In this experiment, students have to implement an algorithm which evaluates the light barriers and switches the magnet on and off to keep the pendulum swinging.

A second experiment is the Higher Striker, which works like a linear motor. Seven electro-magnets are placed around a tube of glass and can be used to accelerate an iron cylinder. Light barriers among the tube can be used to determine the position of the cylinder. The task of the experiment is to analyze the data-stream sampled from the light-barriers and generate a control data-stream for the magnets to move the cylinder to the top of the tube. The electro-magnets and the light barriers are sampled by a control-PC with a fre-

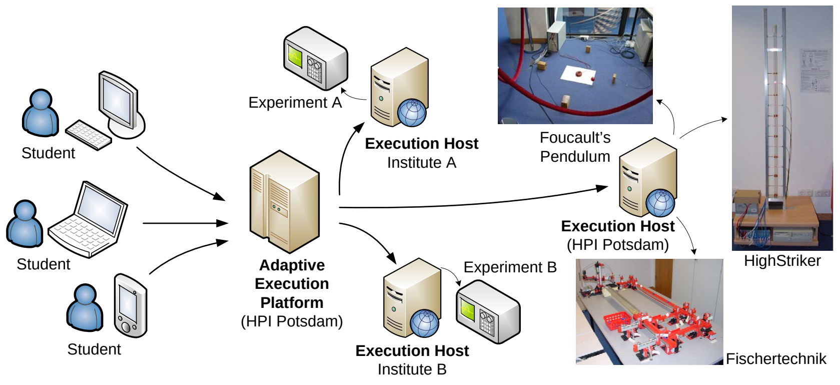

Figure 1. The Distributed Control Lab

quency of 38,4 KHz. We use this experiment to teach the programming of embedded real-time control applications and students compare different real-time operating systems on the control-PC.
Another experiment shown in Figure 1 is a model of an assembly line controlled by a programmable logic controller (PLC). In addition to the control program, which has to be implemented in a PLC-language (IEC 61131), monitoring and HMI components are implemented as Java and .NET programs, which be uploaded via our laboratory infrastructure. Students can use this experiment to experience heterogeneous embedded control systems. Further experiments also include the programming of Lego NXT robots and various simulators for our physical experiment installations.

1.1 Motivation

For several years, the DCL installation at HPI was based on a proprietary distributed .NET application. Both the experiment controllers and the job scheduler where realized with the .NET 1.1 framework, where components interact through a proprietary remoting technology. Meanwhile, new intended experiment types require the experiment controllers to be implemented in Java and other languages that are either not or badly supported by the .NET environment. This requirement motivated the usage a service-oriented middleware, in order to integrate different execution platforms for the experiment controllers.
Within the Vet-Trend project, we are also investigating the integration of experiment installations from different organizations. Since different technologies and execution platforms are in use in the field, a way must be found to integrate these heterogeneous systems. This requirement motivated the usage of commonly accepted Web Services standards to access the experiment installations.
Another motivation to improve the architecture of the DCL was the separation of compile and execution step for single experiment runs. In the old architecture both steps have been coupled together, causing many users to wait for a compilation, while the physical experiment was in use by a long running job.
Our new implementation of the DCL middleware builds upon earlier research results in the area of service-oriented architectures, which are briefly described in the following section.

2 Stateful Service Concept

In order to realize the DCL as service-based distributed environment, we applied results from our earlier *Service Infrastructure* research [8] to the domain of remote/virtual laboratories. Our stateful service approach extends the idea of stateless Web services with the explicit notion of *service instances*. Client applications (such as workflow engines in typical SOA environments) are enforced to perform an explicit service instantiation through a factory operation. The factory returns a reference to a *logical service instance*, which is described as *WS-Addressing*-compliant XML document [2]. This document is then used by the client for subsequent service invocations, which are all automatically related the initially created session between client and server.
A logical service instance 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. This slightly extends the idea of standard Web service frameworks, where services are referenced by an endpoint URI for a particular service instance on a particular machine. Instead, all clients communicate with an *coordination layer* that routes SOAP requests (specifically the SOAP body) to a matching execution host. All logical and physical instances relate to their according service implementation, which is realized as binary Web service component, such as a Java Servlet or a .NET Assembly. The kind of implementation for the particular service is transparent to the client in this case, and depends only on the available kind of execution hosts.
In our stateful service concept, a logical service instance has query-able state and monitoring information, expressed by uniformly accessible attributes. Our

implementation uses the specifications for the *Web Services Resource Framework (WSRF)* to allow interaction with stateful SOAP implementations. WSRF combines the *WS-ResourceProperties (WS-RP)* specification, which defines read, write and list operations for Web service attributes [1], and the *WS-ResourceLifetime (WS-RL)* specification, which defines operations and WS-RP attributes for managing the lifetime of a service instance [7]. Since both the attribute access and the lifetime management is independent from the particular service implementation, clients can access and utilize this functionalities in all cases. The according query and update operations become automatically part of the service interface, as defined in the according standards.

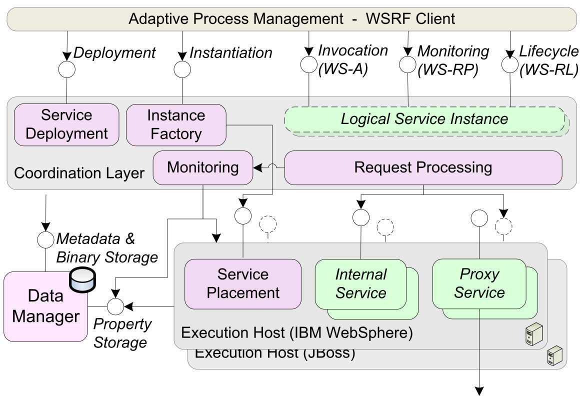

Figure 2. Service Infrastructure

A first implementation of this infrastructure model was realized and tested in the *Adaptive Services Grid* project for a period of 18 months. Figure 2 shows how the logical service instances for the client are managed by a coordination layer, which schedules and manages the incoming requests for the available set of physical service instances. New service implementations can be deployed at runtime, in which case the coordination layer chooses the right execution host for the binary (*service placement*). Service access is monitored by the request processing components, which supports the unified gathering of monitoring information for all service types. Service implementations can use a central storage facility to provide attribute values to the client or to save their own state between invocations. The atomic services can either implement some functionality by them self or act as proxy for external functionality.

2.1 Experiments as a Service

The experience with the existing Service Infrastructure from the ASG project showed the relevance of some architectural patterns for the identified DCL problems:

- Due to the usage of standardized SOAP-based protocols, client and infrastructure implementation can rely on different platforms.
- Stateful interaction with services is made available to the client in an interoperable and standardized manner.
- For scalability and fail-over issues, new services can be deployed during runtime. They can utilize additional execution resources on demand, without effecting active requests and their clients.
- Clients and services are loosely coupled. Service access and state data access do not relate to a particular execution host.

In order to transfer the Service Infrastructure concepts to the virtual laboratory application scenario, we compared the DCL concepts for the old infrastructure and the stateful service concepts of the ASG infrastructure. The resulting mapping is shown in Table 1.
Each DCL experiment controller, the software component to execute jobs on physical experiment hardware, can be represented by an experiment execute service implementation. It provides the necessary interfaces to execute jobs and query job results. Since most of the experiments expect the source code of a control application as input, we also introduced the notion of a compile service. A compile service is specific to an experiment type (as the execute service) and transforms source code to an executable binary, which can be directly passed to an execution service of a given experiment type.
The decoupling of compilation and job execution improves the scalability of single experiment types. Execute services act as proxy for the physical hardware, and can therefore not be duplicated to multiple physical instances on multiple computers. In contrast, the compiler service acts as self-contained functional unit, and can be replicated over multiple execution hosts.
As described in Table 1, every job for an experiment type can be represented by creating a logical instance for an execute service. The mapping between logical instances and physical instances is managed by the coordination layer. Each client therefore can operate its own logical instance (or session) for an experiment. The central request processing module queues the incoming requests for the available physical instances.
The standardized support for service attributes allows a unified representation of job results. Each physical instance can store job results from the experiment run as attribute values. The infrastructure relates such saved attribute values to the logical instance triggering the operation, and stores the value and related logical service instance identifier in a central database. If the client now queries its logical instance for some current attribute value, the coordination layer can provide the latest data made available by the execution service or the compile service. This decouples write and read operations for experiment data, and decouples clients from particular execution hosts for compilation or experiment services.

Distributed Control Lab	Stateful Service Concept
Experiment controller daemon	Execute service implementation
Experiment compiler daemon	Compile service implementation
Experiment simulation daemon	Experiment service implementation
Job for an experiment type	Logical service instances of compiler and execute service
Compiling a job	Operation on logical instance of compiler service
Running a job	Operation on logical instance of execute service
Job results	Attributes on logical instance of execute service
List of all available experiments	List of all usable execute services
Status of users job	Attribute on logical service instance
Queue per physical experiment	Queue per physical service instance

Table 1. Mapping of DCL concepts on stateful services

In the following section, we will now describe the implementation of the updated DCL based on the stateful service concept.

3 Implementation Details

In the current implementation of the DCL infrastructure, new available experiments must be announced by providing an implementation of execute and compile service for a particular experiment type. The experiment provider uploads a *service package* as binary file, containing the service implementation and a *deployment descriptor* (see figure 3). The deployment descriptor contains meta-data such as scheduling configurations, a description of the experiment for the front-end display, properties of the service and the experiment type that is used to group compile and execute services. During the registration process, the WSDL of the compile and the execute service is augmented with the necessary operations defined by the WSRF standards for property and life-time management.

To use an experiment a logical service instance for a compile service and for a execute service has to be created by the front-end on behalf of the end user. The created instances allow the usage of an experiment by invoking the standardized `ExecuteExperiment` or `CompileExperiment` method on the logical instance. The DCL coordination layer ensures that a working physical service instance exists for any logical instances being successfully created. If necessary, it places a new service by loading the according service package to a remote host. Results of the experiment runs are centrally stored and can be accessed via the resource properties of the logical service instance. Experiment hardware cannot be shared among multiple jobs. Therefore, the coordination layer has to support the serialization of invocations for physical service instances.

Listing 1 shows a sample implementation of an execute service for the Lego NXT robot experiment. Users can write control programs for robot movement, and submit it to the infrastructure in order to see the resulting physical activities of the experiment hardware. In the implementation, the class `NxtExecuteService` derives from the `WebService` base class, indicating the implementation of a new Web Service. Each execute service must implement the `ExecuteExperiment` method, which receives the program image to be executed as binary array. The method is called by the coordination layer, based on the next request to be handled from the queue of pending logical instance calls.

```
[WebService(Namespace="http://hpi-web.de/esvc")]
public class NxtExecuteService : WebService
{
  [WebMethod]
  public void ExecuteExperiment(byte[]
       experimentProgram)
  {
    WebCam.Record();

    NXTBluetooth.UploadProgram(experimentProgram);
    NXTBluetooth.StartProgram();

    // save experiment result
    PropertySupport.SetInstanceProperty
      ("Video", WebCam.GetVideo());
    PropertySupport.SetInstanceProperty
      ("DisplayOutput", NXTBluetooth.GetLog());
  }
}
```

Listing 1. Experiment implementation

As first step in the example implementation, a camera recording is started to save a video of the robot's movements during the job execution. Then the binary program image is transferred to the NXT via a Bluetooth connection. After the execution of the job, which is indicated over the Bluetooth connection, results of the experiments are saved in the infrastructure. The DCL implementation automatically determines the related logical instance being called and can therefore provide a generic attribute access library for experiment services. This simplifies the programming model, since the integrators of new experiments don't need to consider the logical instance handling of the coordination layer.

Listing 2 shows a shortened example for the implementation of a client using an experiment and fetching the results afterwards. In the listing, logical instances

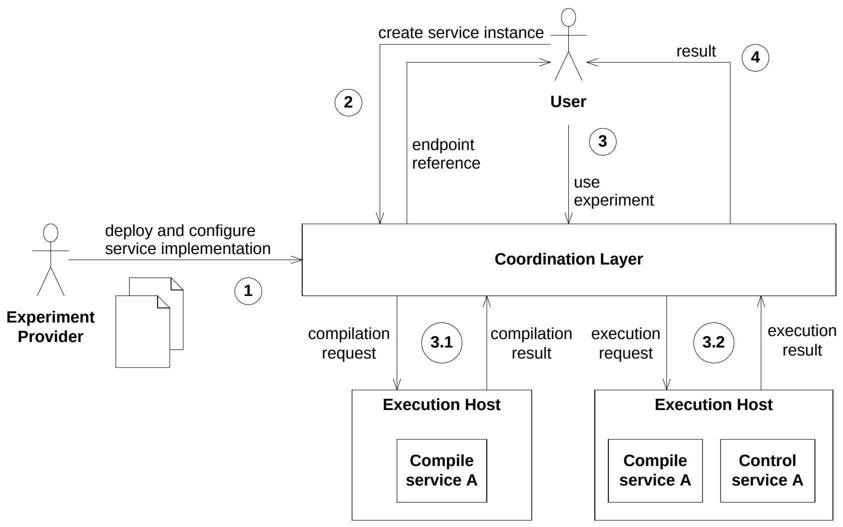

Figure 3. Workflow of activities in the Distributed Control Lab

```
// retrieve service instances
EndpointReference compileService =
  serviceFactory.CreateServiceInstance(
       CompileServiceId);
EndpointReference executeService =
  serviceFactory.CreateServiceInstance(
       ExecuteServiceId);

// compile code and execute experiment
byte[] compilationResult =
  compileService.CompileExperiment(code);
executeService.ExecuteExperiment(compilationResult)
     ;

// retrieve experiment result
byte[] experimentResult =
  executeService.GetResourceProperty("Result");
```

Listing 2. Experiment client

for compile and execute service are created first. Each service instance is represented by an endpoint reference object, in accordance to the WSRF specification. The `ExecuteExperiment` method returns after the finalization of the control program, or after the maximum time allowed for execution has been expired. The result of the experiment run can be acquired by accessing the according service attributes with the WS-RP operations.

All DCL experiments are currently accessible over the Internet. Therefore it must be ensured that only authorized users can access the experiments. The DCL therefore requires authentication data to be present in a SOAP request as described by the *WS-Security Username Token Profile* [3]. Furthermore, clients and experiment developers are free to use other WSS compliant mechanisms to protect the message body.

3.1 Scheduling

The classification and different treatment of incoming requests is another demand on virtual laboratory middleware, identified in the practical experiences with the classical DCL infrastructure. An open access to researchers, students and guest users at the same time demands a prioritization of specific requests according to the user identity.

In order to schedule Web service requests in our infrastructure according to an assigned priority, two problems needed to be solved. First, the priority decision value that is encoded in the SOAP message needs to be accessed. As most Web service stacks abstract from the communication handling, the access to priorities is usually not possible before the request processing starts. This prevents a re-ordering of incoming requests according to some priority setting. The problem was solved by intercepting the SOAP processing in the Web service stack, and analyzing the incoming raw XML data in a custom preprocessing handler. This step also covers the reaction on WSRF-compliant request messages, for example for the attribute access, which is not covered by the service implementation itself. The solution provides fast access to the priority values and allows the correct routing of the result messages.

Some of the experiment hardware requires a recovery phase between successive jobs. This is implemented by according queuing strategies in the coordination layer implementation.

Using our central scheduling approach, we are able to tolerate crash-faults of experiment controller machines, by having execution hosts installed on redundant computers. Before executing a job, the coordination layer checks whether the chosen physical service instance is still operational. If this is not the case, a physical service instance on another host is used. If no more physical instances are available, the coordination layer can also decide to deploy the service implementations to another empty machine. The infrastructure supports the addition of new execution hosts at run time, which allows the immediate reaction on failures without down time for the whole infrastructure.

In a future step, we plan to delegate jobs to multiple physical service instances in parallel and choose a result according to a voting mechanism. This mechanism is independent from the location of the execution host, and can therefore support fail-over scenarios between multiple interconnected virtual labs. For the sake of extensibility, the coordination layer itself is not aware of the deployment format of a service package. At the moment, our infrastructure contains two different types of service containers – one type to process .NET Web services, and one type for JAX-WS Web services. Since all incoming requests contain the information about the logical service instance, successive jobs need not to be processed by the same physical instance. This supports both load balancing and fault tolerance for a particular experiment type, under the assumption of reliable central data storage.

4 Operational Experiences

Within the VET-Trend project, we started a first pilot effort for integrating experiment installations at the Technical University Darmstadt and at the Hass-Plattner-Institute. TU Darmstadt is operating a remote laboratory for reconfigurable hardware modules, which can be programmed and tested by according tools. In order to perform the integration, TU Darmstadt is going to implement a service interface for their experiments, which will be used by our infrastructure. Beside the batch mode programming of hardware modules, we also identified the need for an interactive mode, which is required to perform test cycles at the downloaded hardware configuration. The interactive mode will realized as stream-based interaction with an experiment during the execution of a job.

Practical tests showed that the usage of SOAP messaging to query information about experiment runs is the most time consuming task in the new infrastructure. Since most of the job-related information remains constant during their life-time, we integrated several caches in parts of the infrastructure. With this technique, the number of fully processed service invocations at the coordination layer was dramatically reduced.

In the current implementation, we successfully interconnected an ASP.NET frontend with the coordination layer written in Java 6. Current execution host are programmed both in Java and .NET, and initial experiments already showed the possibility also for other platforms. In general, the usage of mature Web service standards and the consideration of WS-I regulations has shown to be helpful in order to achieve true interoperability in a heterogeneous middleware environment.

5 Conclusion

The utilization of service-oriented software architectures for remote/virtual laboratories is a promising approach for solving the typical problems of cross-organizational access, scalable behavior and dynamic resource usage. We presented our concept of an 'experiment as a service', were physical service instances provide access to the experiment hardware and logical service instances represent according user jobs. The application of mature Web service technologies allows to establish a transnational virtual laboratory environment, which integrates experiments and users from different sites all over Europe. First steps toward such an infrastructure already have been taken.
We have successfully used our laboratory infrastructure in courses on embedded systems, held at the Hasso-Plattner-Institute and the Blekinge Institute of Technology in Sweden. The integration of new experiments from other organizations has just started. Future work we will concentrate on the integration of further experiments, and on the improvement of our service-oriented laboratory middleware according to user and integrator feedback.

References

[1] S. Graham and J. Treadwell. Web Services Resource Properties 1.2 (WS-ResourceProperties). OASIS Open, June 2004.

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

[3] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Services Security UsernameToken Profile 1.1. OASIS Open, Feb. 2006.

[4] A. Rasche and A. 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, May 2003.

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

[6] 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 Realtime Dependable Systems*, pages 299–306, Oct. 2003.

[7] L. Srinivasan and T. Banks. Web Services Resource Lifetime 1.2 (WS-ResourceLifetime). OASIS Open, June 2004.

[8] P. Tröger, H. Meyer, I. Melzer, and M. Flehmig. Dynamic Provisioning and Monitoring of Stateful Services. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies (WEBIST 2007)*, pages 434–438, Mar. 2007.