Design and Implementation Issues for Modern Remote Laboratories

Eliane G. Guimarães, Eleri Cardozo, Daniel H. Moraes, and Paulo R. Coelho

Abstract—The design and implementation of remote laboratories present different levels of complexity according to the nature of the equipments operated by the remote laboratory, the requirements imposed on the accessing computers, the network linking the user to the laboratory, and the type of experiments the laboratory supports. This paper addresses the design and implementation of remote laboratories employing web technologies, both at the client and the server side. These types of remote laboratories are called WebLabs, and can be deployed over different networks such as the public internet, campuswide networks, or high-speed private networks. Although most published works on WebLabs focus on their functional and operational aspects, nonfunctional requirements related to security, quality of service, and federated operation of WebLabs have received little attention. This paper addresses how these requirements can be incorporated into WebLab design, and discusses the most appropriate web technologies to fulfill such requirements.

Index Terms—Remote laboratories, WebLabs, federated WebLabs, mobile robotics.

1 Introduction

Most web-based laboratory (WebLab) implementations focus on the interaction facilities for manipulating WebLab resources over the network. Such facilities try to reproduce on the user's computer the same interaction mechanisms employed in the in situ operation of the resources. For example, a WebLab experiment that employs a spectrum analyzer can offer an interface that displays to perfection the equipment's console with its screens, push buttons, knobs, switches, and lights. Through this interface, a remote user can operate the equipment in much the same way as if he/she were physically present in the lab.

Although interaction is a key issue in WebLab design, it is not the only one. As Internet connection speed grows steadily and high-speed academic networks reach more and more students and researchers, WebLabs are expected to become valuable tools for practical experimentation in learning and research activities. In this scenario, institutions may also take an interest in sharing WebLabs in order to increase the spectrum of laboratorial facilities offered to their students and researchers. Issues such as security, quality of service, and operation across organizational borders become crucial for the success of future WebLab projects.

This paper details the topics of security, quality of service, and federated operation in WebLab design and deployment. A domain-independent platform for WebLabs addressing these topics is presented. A WebLab in the field of mobile robotics built above this platform is presented as well. The contributions of this paper to the field of remote labs are: 1) to propose a reference model with which remote labs can be designed and compared; 2) to discuss subjects rarely found in the remote lab literature: quality of service, security, and federated operation, as well as to propose technical solutions for incorporating these subjects into the development of remote labs; 3) to identify a set of domain-independent services for building software platforms to support secure remote labs.

The paper is organized as follows: Section 2 presents a reference model for WebLabs that helps to identify and classify the components required by the the federated operation of WebLabs. Section 3 discusses how security can be incorporated into WebLab design in order to protect the resources accessible through the network. Section 4 addresses quality of service in WebLabs. Section 5 discusses federated operation of WebLabs, i.e., the sharing of WebLabs among multiple organizations. Section 6 presents REALabs, a platform for WebLabs, and REALabs-BOT, a WebLab in mobile robotics. Section 7 presents the usage of REALabs-BOT in a graduate course on mobile robotics. Section 8 evaluates some related works, and Section 9 concludes the paper.

2 A Reference Model for WebLabs

Fig. 1 shows a reference model for WebLabs in a Unified Modeling Language (UML) simplified class diagram. This model is an extension of a reference model proposed by the authors in [1]. The purpose of such a model is to identify the major functions that a modern remote lab adopting a service-oriented architecture must support. The model is neutral in terms of implementation technology and application domain. The central entity of the model is the WebLab itself. This component operates a set of resources, both physical

- *E.G. Guimarães is with the Robotics and Computer Vision Division, Information Technology Center Renato Archer, Rodovia Dom Pedro I (SP-65) Km. 143,6, Campinas, SP 13069-901, Brazil.*
  E-mail: eliane.guimaraes@cti.gov.br.
- *E. Cardozo, D.H. Moraes, and P.R. Coelho are with the School of Electrical and Computer Engineering, University of Campinas, DCA-FEEC-UNICAMP, Av. Albert Einstein, 400, Campinas, SP 13083-852, Brazil.*
  E-mail: {eleri, dmoraes, pcoelho}@dca.fee.unicamp.br.

Manuscript received 31 Mar. 2009; revised 15 Nov. 2009; accepted 27 July 2010; published online 12 Aug. 2010.
For information on obtaining reprints of this article, please send e-mail to: lt@computer.org, and reference IEEECS Log Number TLTSI-2009-03-0064.
Digital Object Identifier no. 10.1109/TLT.2010.22.

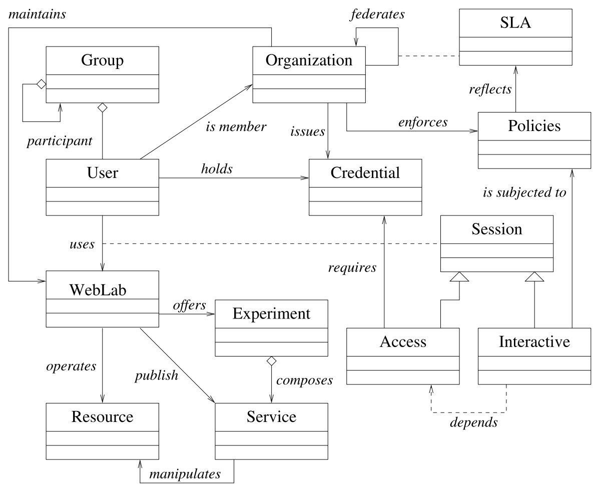


Fig. 1. A reference model for WebLabs.

(equipments, devices, machines, etc.) and logical (software systems). Resources are remotely manipulated by services offered by the WebLab. Services hide the particular characteristics of a resource, for instance, the programming language and communication protocol employed to handle it. The mapping between services and resources is arbitrary. Services may aggregate multiple resources into a single manipulation unit (e.g., a camera and a microphone as a communication resource), and multiple services can manipulate a single resource (e.g., each service offering a different mode of operation for the resource). Services can execute in the resources' embedded processors (as in network cameras and robots) or in external processors connected to the resource, known as lab servers. However, no matter how services and resources are mapped or where the services run, the model requires that the manipulation of resources be done solely through well-defined services.

Experiments offered by a WebLab are performed through a composition of services. This composition can be supplied by the WebLab or by the user. For instance, a WebLab in mobile robotics can offer services to manipulate a mobile robot, such as a locomotion service to drive the robot, a perception service to acquire data from the robot's sensing devices, and an action service to configure pre-programmed actions on the robot. More specialized services such as localization, mapping, and autonomous navigation can be offered as well. In this context, a teleoperation experiment can be prepared by composing a locomotion and a perception services in order to allow the user to drive the robot with the aid of an on-board camera and a sonar or laser map updated as the robot moves.

WebLabs are maintained by organizations for the benefit of their members (registered users). Organizations can form federations in order to share among their members the WebLabs they maintain. This sharing is regulated by contracts or service-level agreements (SLA). SLAs state the conditions governing the use of a WebLab by members of other organizations. For example, through an SLA, an organization can offer a WebLab it maintains to members of another organization, subject to restrictions and privileges such as maximum reservation time, maximum number of accesses per day, and quality of service. SLAs are described by a set of usage policies expressed as a number of condition/action statements (rules) that establish how experiments and services behave according to the restrictions and privileges stated in the SLAs.

Organizations issue credentials to their registered users after the user is authenticated. Credentials are assertions (facts) about the user such as his/her identity, authentication method, and credential expiring date. Credentials are usually digitally signed by the issuing organization in order to assure its origin and integrity. In a federation, credentials must be understood by all the federated organizations.

In order to access an experiment or service maintained by a WebLab, the user must establish an access session with the WebLab. This process checks whether the credentials presented by the user suffice for granting him/her access to the experiment or service. An access session holds state about the user accessing the WebLab, and lasts until the user explicitly terminates the access, or until some termination condition holds (e.g., the amount of time reserved for the access has expired).

Once an access session is successfully established, the user can initiate interactive sessions with the WebLab. An interactive session holds state about the usage of the resources, and must assure that the resources assigned to the session are in consistent state before the session begins and after the session ends. Examples of interactive sessions include manipulation of multimedia devices, scientific instruments, and software systems such as simulators and emulators. Interactive sessions require authorization. Authorization consists in checking whether the access complies with the policies enforced by the organization for that WebLab (e.g., the user is accessing an experiment or service anticipated in an SLA established between the user's and WebLab's organizations).

A group of users can access a WebLab simultaneously. In this case, all group members are allowed to establish access and interactive sessions at the same time. For this usage model, the WebLab must allow reservations to be shared among group members. The WebLab must also supply concurrency control mechanisms that prevent misoperation of the resources under concurrent access.

3 WEBLAB SECURITY

Security is of prime importance for WebLabs, independently if they are accessible through the public Internet or a private network. Security consists basically of authentication and authorization.

Usually, authentication consists of a challenge presented to the user. In its simpler form, the challenge is a request of a password. More elaborated schemes may demand the user to present a certificate issued by the organization to which he or she belongs. A certificate is a text identifying the user and signed with the organization's private key. Once the user is authenticated, the organization issues a set of credentials with assertions about the user. Credentials are also signed with the organization's private key and inspected on subsequent interactions with the WebLab. Once the user is successfully authenticated, it is said that the user has established a valid access session with the WebLab.

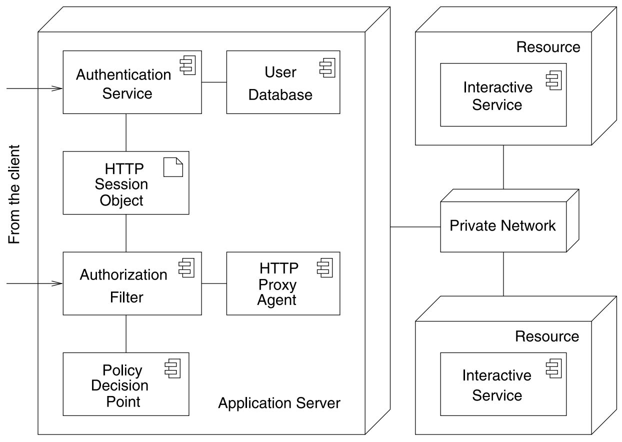


Fig. 2. Web-based authentication and authorization.

In web-based applications such as WebLabs, if the authentication is mediated by an application server, the server maintains a Hypertext Transfer Protocol (HTTP) session for a period of time. As long as the HTTP session is valid, the user is considered authenticated by the server. Fig. 2 shows a web-based scheme of authentication and authorization for WebLabs in a UML deployment diagram.

An HTTP session object maintains state about the session, including the user's credentials. A web-based authentication service can employ a database for storing the registered users and HTTP session objects holding information about the authenticated users. The access session of our WebLab model can be implemented as an HTTP session object maintained by the application server.

Authorization is the process of checking whether an authenticated user is allowed to establish an interactive session with the WebLab. This is usually based on access policies. An access policy condition checks many access parameters such as user's identity, credentials, location, resource being accessed, etc. An access policy action allows or denies the access to the resource. As an example, an access policy can state that administrative services may only be invoked by users holding administrative credentials. Another example is an access policy stating that administrative services may only be invoked from machines inside the organization.

On web-based applications, authorization can be implemented by an HTTP filter. HTTP filters are extra code incorporated into the application server. This code is able to inspect the HTTP request before it is processed by the server's core, and has the power to stop the request processing and return an error to the client. Authorization filters extract some parameters from the HTTP request and from other sources (e.g., HTTP session objects), and submit these parameters to the policies. In policy terminology, the authorization filter becomes a policy enforcement point. The entity that holds and processes the policies is called a policy decision point. This entity can be colocated with the authorization filter, or it can be a separate entity (e.g., a rule-based engine). The authorization filter may install a state related to the interactive sessions in order to prevent policy checking at each HTTP operation.

Authorization must also prevent a malicious user from bypassing this process and interacting directly with a service or resource. Resources and services are unaware of the need of establishing access and interactive sessions in order to interact with them. For example, if the user is interacting with a service supplied by a network camera and discovers the camera's Internet Protocol (IP) address, e.g., by sniffing the HTTP traffic, he/she can directly operate the camera through this address even after the interactive session has ended. In order to prevent such vulnerability, resources such as robots and cameras that are directly operated through the network must be placed on a private network. By assigning private addresses to resources, their direct operation by the user becomes impossible. In this case, services manipulating resources must be accessible solely from proxies reached after the request has passed through the authorization process. Another benefit of proxies is to help applications meet the "same origin" security policy imposed by Java and Asynchronous Javascript and XML (AJAX) technologies.

4 Quality of Service in WebLabs

Quality of service (QoS) consists of a set of control and management functions allowing the network to guarantee some end-to-end metrics such as delay and jitter for the traffic flows generated by the applications. A traffic flow is identified by network packets generated at the same origin and targeted to the same destination. Origin and destination can be defined in terms of applications, machines, networks, or even domains. In order to assure end-to-end metrics, the control and management functions must be performed on each router along the flow path. This requirement is rarely fulfilled due to network operation practices, mainly when the flows cross multiple administrative domains. However, QoS techniques can be applied in more restricted scenarios, contributing to avoid performance degradation of the distributed applications when network resources are limited. Two QoS techniques can improve the performance of WebLabs: flow management and experiment adaptation.

Flow management consists in establishing relative priorities among the flows and limiting the network resources (bandwidth) consumed by the flows. Flow management requires routers able to identify packets belonging to a given flow, mark the packets according to the priority assigned to the flow, and forward the packets based on their marks. A WebLab can prioritize and constrain the flows it produces or consumes. For example, flows related to equipment control must have higher priority than those related to user interaction (e.g., video from panoramic cameras). Although such management actions are circumscribed to the WebLab domain, they avoid the degradation of experiments as perceived by the user on low-bandwidth Internet connections.

As an example of the benefits of flow management, consider Figs. 3 and 4. These figures show a laser-based navigation experiment where the robot detours an obstacle in its path using an autonomous navigation algorithm based on potential fields. The experiment manipulates two flows, one produced by the robot's laser scanner, and another by the robot's on-board camera. The trajectory in Fig. 3 was obtained without flow prioritization, with video

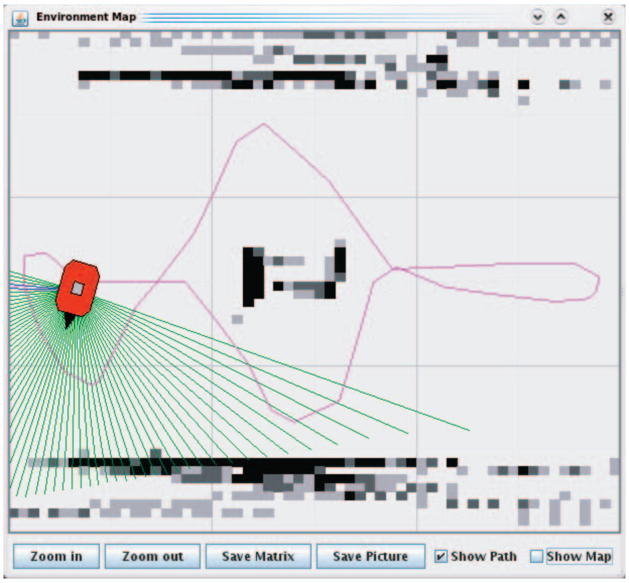


Fig. 3. Autonomous navigation with low telemetry priorities.

flow consuming nearly 1 Mbit/s. In Fig. 4, the telemetry flow was given higher priority than the video flow. In addition, video flow was limited to 200 Kbit/s. Visually, by looking at the stability (smoothness) of the trajectories, it is easy to notice the impact of flow management on WebLab experiment results. In [2], [3], we detail the use of flow management in WebLab experiments.

Experiment adaptation adjusts the experiment behavior according to the available network resources. Indirectly, adaptation acts on the flows, reducing or increasing them according to the actual network capacity. The adaptation process requires a controller that acts on the experiment according to a control strategy. As an example of adaptation, consider a WebLab experiment in mobile robotics where the robot must follow a colored stripe on the floor using its on-board camera. The vision-based navigation

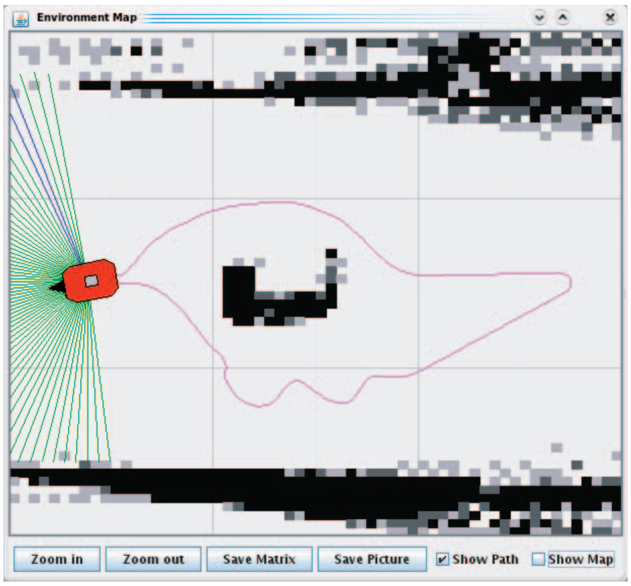


Fig. 4. Autonomous navigation with high telemetry priorities.

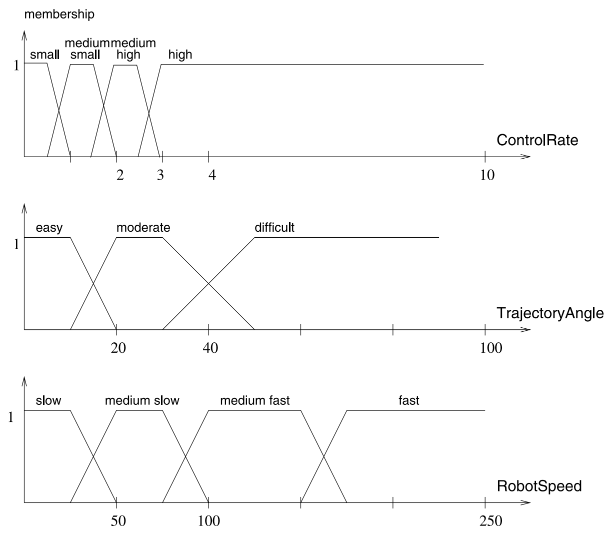


Fig. 5. Fuzzy sets for control cycle rate, angle of the trajectory, and robot's speed.

algorithm runs on the user's computer and performs the following cycle:

1. Acquire an image from the robot's on-board camera.
2. Using image filtering and segmentation, identify the stripe in the image.
3. Compute the angle necessary to orient the robot in order to center the stripe in the image.
4. Turn the robot through the angle computed in step 3 and go to step 1.

The problem with this procedure is to set a proper robot speed. If the speed is set too high, the robot looses the stripe on angled trajectory segments. If the speed is set too low, the robot takes a long time to complete the trajectory. This speed is a function of the trajectory's smoothness and the rate of cycles the algorithm performs.

For this experiment, we employed an adaptation module (controller) based on fuzzy logic. The controller receives two input parameters and computes an appropriate robot speed. The parameters are the number of actuations per second and the angle of the last actuation. The first parameter, control rate, reflects the network delay and the power of the user's computer. The second parameter, trajectory angle, reflects how difficult (angled) the trajectory is in the vicinity of the robot. Input and output parameters are expressed in terms of fuzzy sets shown in Fig. 5. For each control variable (X-axis), fuzzy sets (small, high, etc.) are defined in terms of membership functions that specify in the Y-axis the degree of membership (from 0 to 1) of the variable in the set. The controller employs fuzzy rules such as *"if ControlRate is small and TrajectoryAngle is difficult then RobotSpeed is slow."*

Table 1 shows the average robot speed as a function of the accessing network. Note that depending on the accessing network, average robot speed changes seven times from a slow to a high-speed network.

Other control strategies can be employed for adaptation. For example, Saltzmann et al. [4] describe an adaptation scheme for remote experimentation based on an integral controller.

TABLE 1
Maximum Speeds for Different Accessing Networks

Accessing Network	Speed (mm/s)
Residential Cable Modem	30
Campus Ethernet	90
WebLab internal Ethernet	170
Gigabit Optical	200

5 Federated Operation of WebLabs

A federation of WebLabs allows WebLabs to share experiments and resources. Users subscribed at one organization can use WebLabs offered by their own organization and by other federated organizations. Federated WebLabs must offer a set of management functions for the model elements of Fig. 1 such as those listed below:

- management of users,
- management of resources,
- management of SLA, policies, and QoS,
- management of access.

The management of users allows an organization to subscribe/unsubscribe users and group of users, and to assign attributes to users such as identities, credentials, and profiles (preferences).

The management of resources allows an organization to present to other organizations the resources it maintains, the policies assigned to its resources and experiments, and the resource reservation schedule. The management of resources must offer a service for resource reservation subjected to the local policies.

The management of SLAs allows an organization to model the contracts it has established with other organizations of the federation. Usually, contracts are expressed by electronically signed documents (e.g., in XML format) stating, for instance, the parties to the agreement, the services covered by the contract, QoS parameters, responsibilities, roles, and pre/postconditions for service invocation.

The management of policies allows an organization to establish and enforce access and usage policies for the WebLabs it maintains, both for its own subscribed users and for those subscribed at other federated organizations. Policies can be modeled by business rules that reflect the SLAs established by the organization.

The management of QoS can employ flow management and experiment adaptation as described in Section 4. For each experiment, an SLA can state the relative priorities and maximum bandwidth for each flow produced or consumed by the resources assigned to the experiment.

The management of access comprises federated authentication and authorization. Federated authentication, usually known as Single Sign On [5], allows users to be authenticated only once and by their respective organizations in order to access resources of any federated organization. Single Sign On assures that the information about the user is stored only in his/her home organization. Federated authorization is a process by which the identity and credentials of a user are recognized by all federated organizations when the user tries to access any WebLab in the federation.

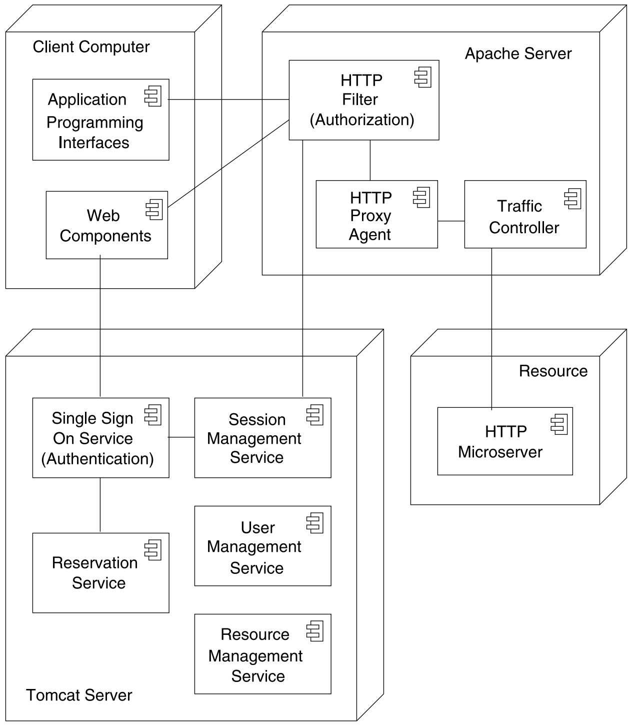


Fig. 6. Main components of the REALabs platform.

Federated access requires standards for the secure exchanging of authentication and authorization data among the organizations, as well as a common semantics (ontology) for describing and negotiating SLAs.

6 The REALabs Platform

REALabs is a domain-independent service-oriented platform for WebLab implementations. It follows the WebLab reference model of Fig. 1, and implements the management services listed in Section 5. In addition, the platform offers a set of web-based utilities relevant for WebLab development, deployment, and operation. The services follow the Representational State Transfer (REST) [6] interaction style, and are invoked by HTTP operations (GET or POST) carrying, when applicable, XML documents as payloads.

The REALabs platform is implemented with the J2EE technology, mainly servlets and Java Server Pages as offered by the Apache Tomcat application server. Management of users and resources are classical database-centered applications and will not be further detailed.

The SLA model element (Fig. 1) supported by REALabs is very simple: users with the same credentials have the same privileges, no matter to which organization they belong. QoS management is implemented for all experiments via traffic prioritization and bandwidth limitation. Once the resources have their flows characterized, the platform generates a shell script for installing packet filters and traffic control queues on the routers and servers serving the WebLab. The script calls *tc*, a traffic controller facility available in Linux.

Fig. 6 shows the platform in an UML deployment diagram. The management of access comprises a reservation service, a Single Sign On service, and a session management service. The reservation service allows users to reserve a time slot to perform an experiment. Reservation is possible only during the periods when the experiment is open for reservation, and requires user authentication and authorization.

The Single Sign On service is fundamental for federated access. This service relies on OpenSAML, an open system for creating, signing, and validating Security Assertion Markup Language (SAML) documents. SAML is an XML-based standard for exchanging authentication and authorization data between organizations. The Single Sign On service is entirely implemented in Java with servlets and HTTP filters. This service supports access sessions according to the model of Fig. 1.

The session management service is part of the authorization process offered by the platform. Once the user is authenticated, the experiment logic must call this service to initiate and maintain interactive sessions. The service installs a cookie in the user's web browser that is carried in all subsequent HTTP interactions. This cookie is checked during the authorization process. Interactive sessions are soft state, meaning that they are terminated if the experiment does not maintain them by calling this service periodically. This behavior is obtained by issuing cookies with a short lifetime. Once a cookie expires, the web browser stops sending it, causing a failure in the authorization process. Soft state will close all inactive interactive sessions such as those left open when the user ceases interaction with the WebLab.

The authorization process is complemented by an HTTP filter for the Apache HTTP server. The filter inspects whether the HTTP request carries a valid cookie installed by the interactive session management service. Authorization filters need to be extended for policy checking. The only policy (Policy element in Fig. 1) currently enforced by the REALabs is "usage upon reservation": in order to establish an interactive section, the user must have previously established an access session that, in turn, requires a reservation for the experiment. A more flexible policy management system is under development.

In the REALabs platform, HTTP proxying is performed by the Apache HTTP server. In this server, HTTP proxying is available by configuration. This platform was employed in the development of two WebLabs in mobile robotics. The first implementation is reported in [1], and the latest implementation is described in the sequence.

6.1 The REALabs-BOT WebLab

REALabs-BOT is a WebLab for mobile robotics experiments built above the REALabs platform [7]. It was designed to run over high-speed networks, although it can operate over the public Internet and campuswide networks as well. In REALabs-BOT, experiments run at the user's computer (although the REALabs platform does not require this). Although many WebLabs allow the uploading and execution of user-developed experiments on the WebLab's servers, we favor the execution of such experiments on the user's own computer for the following reasons related to convenience and security:

- Users can employ their favorite Integrated Development Environment (IDE) and programming language to code, debug, and run the experiments they developed.
- Execution of user-developed code on the WebLab servers is insecure, except if the code runs inside a sandbox. However, sandboxes restrict the code in terms of access to the file system, invocation of system calls, and consumption of processor resources such as CPU and memory. Moreover, user and WebLab must share exactly the same version of compilers, libraries, and operating systems.
- The WebLab need not offer a workspace for storing user's programs and output files. The workspace is the user's own computer.
- The penalty in performance decreases as the user's computer and Internet connection capacities increase.

Returning to the model of Fig. 1, REALabs-BOT offers a set of interactive services in the mobile robotics domain. The remaining model elements are provided by the REALabs platform. Currently, four classes of interactive services are implemented:

1. Locomotion service: allows operations for moving the mobile robots and its accessories such as grippers and arms.
2. Action service: allows the installation of preprogrammed actions on the mobile robots. Examples of such actions are obstacle avoidance, collision recovery, and stall recovery.
3. Telemetry service: allows the acquisition of sensory data from the mobile robots.
4. Vision service: allows the positioning and image capturing from both panoramic and on-board cameras.

REALabs-BOT offers both physical and logical resources. Physical resources consist of Pioneer P3-DX mobile robots from MobileRobots Inc., with different configurations, and pan-tilt-zoom network cameras from Axis Communications. Logical resources are Java/Javascript-based software components that can be added to experiments. An example of logical resource is SmartCam, a component that allows a panoramic camera to follow a moving robot.

In REALabs-BOT, interactive services are entirely based on HTTP and XML, and follow the REST interaction style. Every operation over a resource consists of an HTTP operation that may return an XML document. Operation name and parameters are identified in the Uniform Resource Locator (URL). For example, by issuing an HTTP GET operation to the URL */move?dist=1000*, the robot moves 1,000 mm synchronously. The operation returns the HTTP 200 OK code when the move has been completed. Another example: an HTTP GET operation over the URL */telemetry?sensor=laser&range=-30:30:3* returns the HTTP 200 OK code with an XML document containing the data read from the laser scanner in the range of -30 to $+30$ degrees from the robot's longitudinal axis in steps of 3 degrees. Although XML over HTTP is not efficient in terms of bandwidth consumption, some advantages justify its use:

- HTTP is not blocked by firewalls,
- XML is neutral in terms of programming languages and platforms,

TABLE 2
Results for the Centralized Scenario (Milliseconds)

	ARIA/OpenCV			REALabs		
	Op1	Op2	Op3	Op1	Op2	Op3
Mean	0.00087	0.01151	34.58	0.191	1.929	34.41
SD	0.00042	0.00074	12.18	0.042	0.054	10.40
CI	0.00007	0.00012	2.00	0.007	0.009	1.72

TABLE 3
Result for the Distributed and Secure Scenarios

	Distributed			Secure		
	Op1	Op2	Op3	Op1	Op2	Op3
Mean	4.83	6.23	32.91	35.05	34.76	48.29
SD	1.64	0.44	3.91	4.09	3.37	3.08
CI	0.27	0.07	0.64	0.67	0.56	0.51

- Requests can be easily intercepted for authentication and authorization purposes,
- Both HTTP and XML can be handled inside a web browser through Javascript.

The usage model in REALabs-BOT consists in offering users robots and panoramic cameras with different configurations, plus a set of "utilities" that simplifies the task of developing their own mobile robotics experiments. These utilities include:

- web-based interfaces for login, experiment reservation, and interactive session establishment,
- web-based interfaces for direct manipulation of the resources via their respective interactive services, as shown in the left-hand side of Fig. 11,
- application programming interfaces (APIs) for invoking the interactive services in a programmatic way,
- a set of preprogrammed mobile robotics algorithms (e.g., autonomous navigation and environment mapping), and
- a set of software components to enhance the experiments.

The aim is to let the user develop or configure a mobile robot algorithm using the provided APIs (Java, C++, Python, Matlab, and HTTP-based). The development can be from scratch, from a base code, or from a fully operational code, depending on the complexity level desired for the experiment. The algorithm is first tested on a simulated environment and, after debugging and calibration, the simulated resources are replaced by physical resources for execution in the real environment.

6.2 Quantitative Evaluation of REALabs-BOT WebLab

In order to evaluate the performance of REALabs-BOT, we instrumented experiments performing operations on sensor reading, image gathering, and robot control. The sensor reading operation (Op1) requests high-resolution range data from the robot's laser scanner. A 60 Kbytes XML document is returned with 180 readings in the robot and ground coordinates. The image gathering operation (Op2) requests a 320×240 JPEG picture from the on-board camera. The picture returned in reply averages 20 Kbytes in size. The robot control operation (Op3) simply sets a linear speed on the robot. A small 60-byte XML document with an acknowledgement is returned by this operation. These operations were conducted in three scenarios, each exhibiting certain overheads. The overheads were estimated by comparing the time necessary to complete the operation and the time necessary to act on the robot. The robot's embedded HTTP microserver employs the ARIA robotic framework to acquire data and act on the robot, and OpenCV for capturing images from the internal framegrabber. One hundred measurements for each operation were taken in each scenario. Mean, standard deviation (SD), and a confidence interval (CI) of 90 percent were determined. All results are presented in milliseconds.

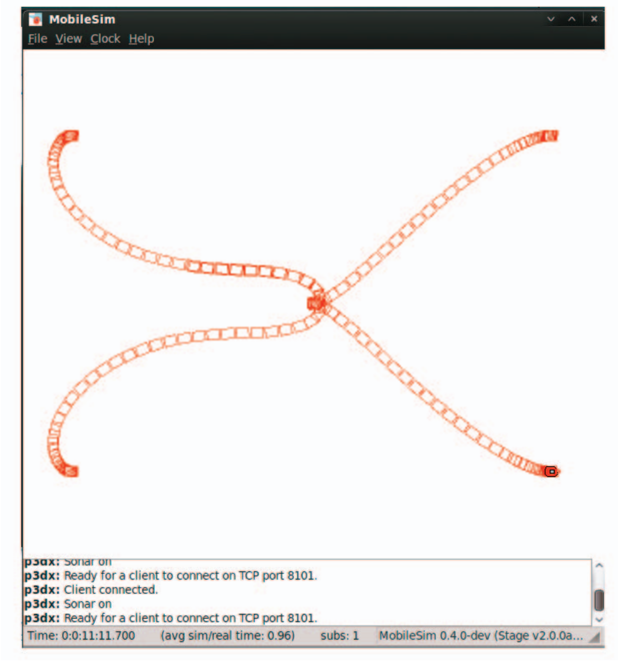
6.2.1 Evaluation Scenarios

In a centralized scenario, the test code performing the three operations runs on the robot's on-board processor, an Intel Pentium M. The protocol overhead (due to HTTP and XML processing) is estimated in this scenario. Table 2 displays the results. Operations 1 and 2 show that protocol overheads range from 0.2 to 2 milliseconds according to the size of XML data returned. Operation 3 shows that HTTP processing overheads are negligible when the HTTP payload is large and needs no XML parsing (such as binary images).

In a distributed scenario, network overheads can be estimated by accessing the robot on a private network through the HTTP proxy agent. A high-speed network was employed in the tests with the results shown in Table 3. Note that network and HTTP proxy overheads add up to about 5-6 milliseconds. Image transferring generates a negligible overhead as the major overhead is on image capture.

We compared the overhead obtained in the distributed scenario with the Microsoft Robotics Studio (MSRS) [8] performing the same sensor readings operation. We set a .NET DSS (Decentralized Software Service) service acquiring the sonar readings using the MSRS facilities and an HTTP GET Service Handling for returning an XML document with the range data. The server side runs on a Dell 510 notebook with Windows XP. The client program was the same used in our tests. We obtained an average of 7.33 milliseconds with standard deviation and confidence interval of 0.73 and 0.14, respectively. The similarity between the REALabs and MSRS protocol overheads indicates that the cost of distribution employing open protocols is limited to a few milliseconds. Also, the 6 milliseconds is one order of magnitude lower than the 70 milliseconds of round-trip time from a cable modem residential Internet access used in the QoS experiment reported in Section 4.

Finally, in a distributed and secure scenario, it is possible to estimate the security overheads by performing the operations via HTTP Secure (HTTPS) protocol. Table 3 shows these results, with the observed overheads of about 30 milliseconds imposed by HTTPS. Again, overheads imposed on image transferring are proportionally lower than those imposed on XML documents. Due to the high overhead imposed by HTTPS, this protocol should be employed only in situations where end-to-end security is a strong requirement.



```
p3dx: Sonar on
p3dx: Ready for a client to connect on TCP port 8101.
p3dx: Client connected.
p3dx: Sonar on
p3dx: Ready for a client to connect on TCP port 8101.
Time: 0:0:11:11.700     (avg sim/real time: 0.96)     subs: 1     MobileSim 0.4.0-dev (Stage v2.0.0a...
```

Fig. 7. Trajectories toward a goal point from different robot positions.

7 Usage Scenario

An introductory graduate course in mobile robotics is being taught by the first two authors at the School of Electrical and Computer Engineering (FEEC), University of Campinas. The main reference book adopted is by Roland Siegwart and Illah Nourbakhsh, *Introduction to Autonomous Mobile Robots* [9]. It covers four major topics:

1. kinematics,
2. perception,
3. localization and mapping, and
4. path planning and navigation.

We proposed four practical experiments addressing these topics as described in the sequence.

7.1 Proposed Experiments

The practical activities are initially conducted on a simulated environment and then on the real robots through the REALabs-BOT WebLab. Students are asked to compare the results from simulated and real environments, and to propose improvements on the mobile robotic algorithms they implemented. For performing the experiments, REALabs-BOT provides the user with a virtual machine for the VirtualBox virtualizer. The virtual machine contains:

- the latest version of Ubuntu (a Linux release),
- a set of developing tools (gcc C++ compiler, Java platform, Firefox web browser, etc.),
- low-level mobile robot APIs (ARIA and Player),
- mobile robot simulators (MobileSim and Stage),
- HttpIpthru, an HTTP microserver that drives both real and simulated mobile robots,
- C++, Java, Python, and Matlab APIs for interacting with HttpIpthru from these programming languages,
- a set of sampling code.

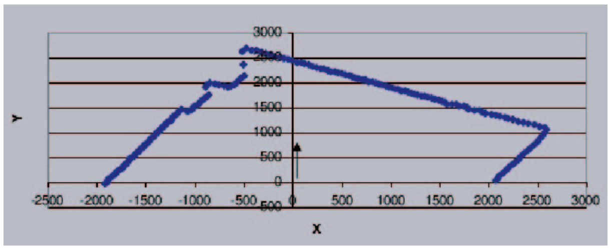


Fig. 8. Laser reading from the robot environment.

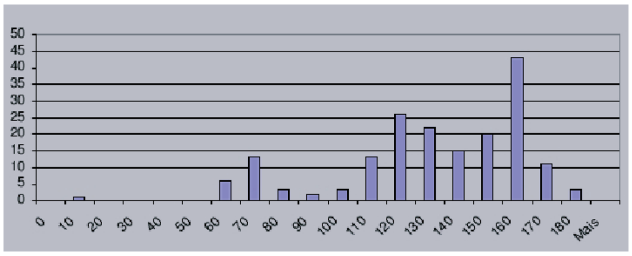


Fig. 9. Angle histogram for feature extraction.

7.1.1 Experiment on Kinematics

In the first experiment, students implement a proportional control algorithm that drives the robot from an initial point to a goal point. The algorithm employs a kinematic model presented in the textbook. Students are asked to employ a mobile robot API and a simulator. ARIA and MobileSim are, respectively, an open source C++ API and a simulator from the robot manufacturer. Player and Stage are, respectively, an open source C++ API and a simulator that support a wide range of mobile robots. ARIA/MobileSim and Player/Stage are suggested for this first experiment. The experiment must be coded in C++ as required by ARIA or Player.

Fig. 7 shows the trajectories toward a goal point from different robot positions obtained by a student in the simulated environment employing ARIA and MobileSim.

For the three remaining activities, students are instructed to employ the APIs offered by the REALabs-BOT WebLab. Students are free to use any of the four supported programming languages: Java, C++, Python, and Matlab.

7.1.2 Experiment on Perception

The experiment on perception aims to estimate the characteristics of the sonar and laser rangefinders installed on the mobile robot. The experiment is conducted with the robot at rest. A series of measurements is acquired from the robot and stored in the student's computer. The measurements are analyzed with tools such as Gnuplot, Excel, and Matlab in order to extract statistical parameters about the sensors. The main parameters are sensor variance, sensor resolution, and sensor precision.

The measurements are also used for feature extraction. Fig. 8 shows the plotting of 180 laser readings obtained through the REALabs-BOT WebLab. By constructing an angle histogram from these points (Fig. 9), walls and other straight-lined obstacles are detected. In the histogram, peaks (Y-axis) correspond to line length, and angles (X-axis) correspond to line orientation.

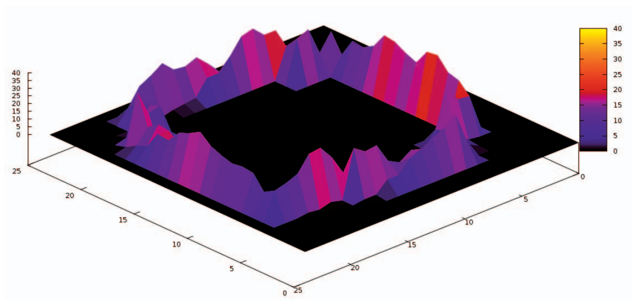


Fig. 10. Occupancy grid obtained through REALabs-BOT WebLab.

7.1.3 *Experiment on Localization and Mapping*

The experiment on localization and mapping aims to derive a map of the robot environment employing a simplified occupancy grid algorithm. The robot wanders around the environment acquiring sonar data and building a map of the environment by marking the obstacles in a bidimensional array. Fig. 10 shows an occupancy grid map plotted with Gnuplot. Peaks represent the number of sonar hits on the respective grid cell. The higher the peak, the higher the likelihood of an obstacle being located at that cell.

7.1.4 *Experiment on Planning and Navigation*

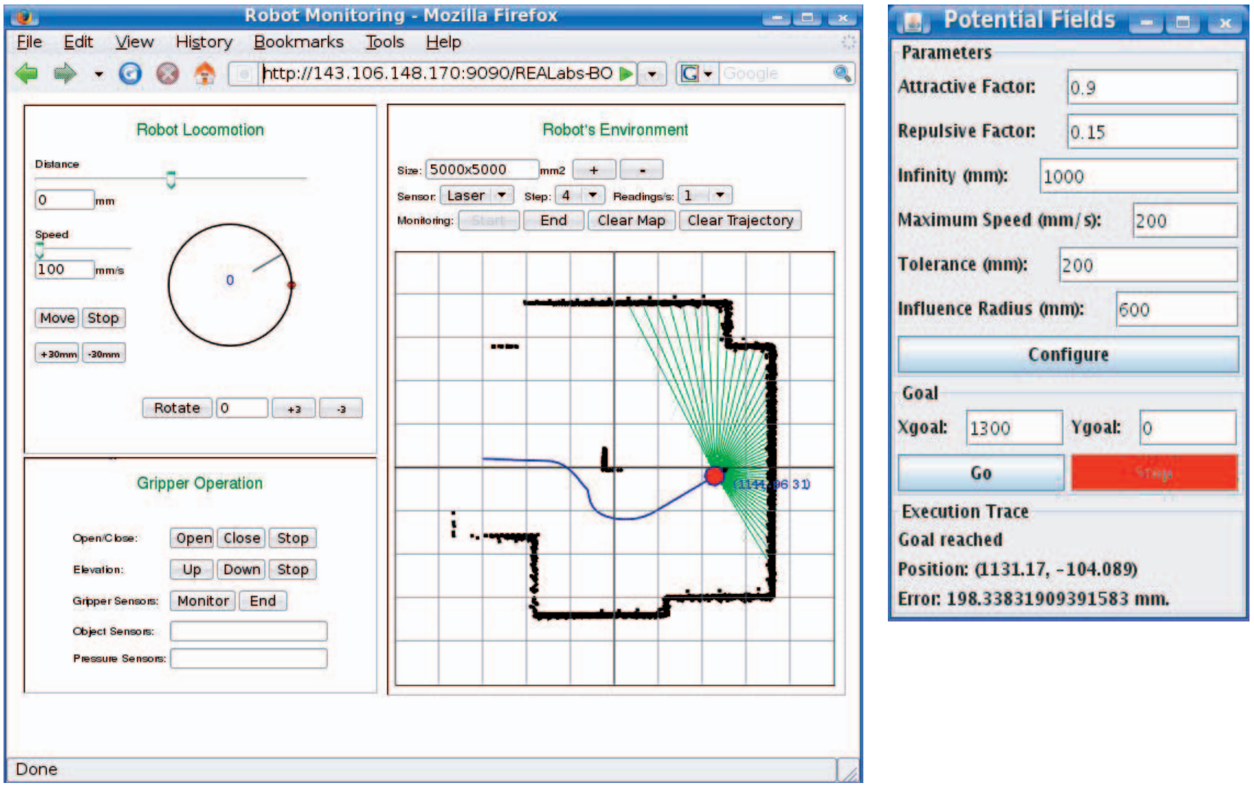
In the planning and navigation experiment, students can choose one of two activities. In the first activity, a potential field algorithm is proposed. Fig. 11 shows a typical trajectory with a navigation based on potential fields, where the robot is attracted to the goal and repelled from obstacles. In this figure, the screen on the left is a web utility for robot monitoring supplied by REALabs-BOT, and the screen on the right is a Java-based interface for configuring the navigation algorithm as developed by a student. In the second activity, the student implements the A^* search algorithm to compute an optimal trajectory between an initial and a goal point, based on a given environment map. In both the activities, the student must devise a control algorithm in order to drive the robot according to the computed trajectory.

7.2 WebLab Infrastructure

The WebLab infrastructure, shown in Fig. 12, is spread over two domains: FEEC and Information Technology Center Renato Archer (CTI). Each domain operates a Pioneer P3-DX mobile robot, an Axis 214 PTZ camera, and a Dell PowerEdge server. The domains are connected through KyaTera, a high-speed academic network deployed in the state of São Paulo, Brazil.

All robots and cameras have private IP addresses accessible only through an HTTP proxy. The proxy runs in the FEEC server, and is able to forward HTTP requests to the CTI via the KyaTera network. From the user's standpoint, there is no perceived difference when operating a robot at FEEC or CTI, although all the Internet access reaches the FEEC first and is then proxyed to the CTI when necessary. Packet filters and queues for traffic prioritization and limitation are installed for each experiment on the Dell server at FEEC using the shell script generated by the REALabs platform.

All students are subscribed at FEEC. We plan to test the Single Sign On infrastructure in the next offering of the course.



http://143.106.148.170:9090/REALabs-BO

Distance: 0 mm
Speed: 100 mm/s
Rotate: 0

Open/Close: Open Close Stop
Elevation: Up Down Stop
Gripper Sensors: Monitor End
Object Sensors:
Pressure Sensors:

Size: 5000x5000 mm2
Sensor: Laser Step: 4 Readings/s: 1
Monitoring: Start End Clear Map Clear Trajectory

Parameters
Attractive Factor: 0.9
Repulsive Factor: 0.15
Infinity (mm): 1000
Maximum Speed (mm/s): 200
Tolerance (mm): 200
Influence Radius (mm): 600

Goal
Xgoal: 1300 Ygoal: 0

Execution Trace
Goal reached
Position: (1131.17, -104.089)
Error: 198.33831909391583 mm.

Fig. 11. Potential fields navigation experiment in a real environment.

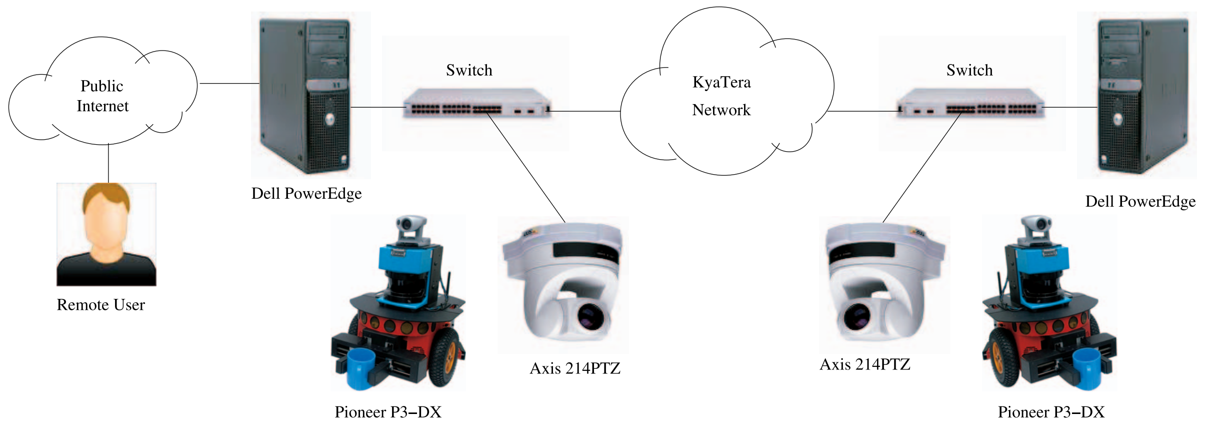


Fig. 12. Infrastructure of the REALabs-BOT WebLab.

7.3 Evaluation by Students

Although we have not conducted a systematic evaluation of the WebLab, inputs from the 15 students were positive, particularly with regard to 1) the ability to develop and test the experiments first on a simulated environment and then on the real robots, 2) the fact that the code supplied as a VirtualBox virtual machine avoids the problems of software installation, and runs above the student's favorite operating system, and 3) the ability to conduct practical experiments remotely from home or office as allowed by the WebLab.

Three students complained that their Internet connection bandwidth was too low for remote experimentation. In these cases, they reduced the quality of the panoramic camera to a minimum, or did not even open the camera in order to avoid long delays in the interaction with the mobile robot. More recently, we implemented an adaptation mechanism in the panoramic camera component in which size and frame rate are adjusted according to the round-trip time of a video frame request.

As most of the students own notebooks, the deployment of access points connected to the KyaTera network was suggested. From these access points, FEEC students could connect directly to the WebLab through a high-speed network improving significantly the quality of interaction.

There was a consensus that the APIs provided by REALabs-BOT are much simpler to use than the ARIA and Player low-level APIs. The freedom of choosing among different programming languages was also noted, as ARIA and Player demand C++ programming. Students also recognized the value of the teaching assistants (TAs) aiding the usage of the WebLab. The dialog between the students and TAs was conducted via MSN.

Finally, students reported no failures during the usage of REALabs-BOT. This robustness of the design and implementation of the WebLab is, in part, due to the RealLabs platform that provides a set of common services already implemented and tested.

7.4 Access Statistics

As the REALabs platform logs every reservation and access into the database, it is straightforward to build a tool (e.g., using PHP) that provides access statistics based on logged data. Log information consists of user and group identification, WebLab and experiment accessed, reserved timeslot, access session starting and ending times, and the event causing session termination (logout, inactivity, or expired reservation). Log analysis provides useful information for next course offerings, such as average amount of time to execute a particular experiment, relationship between reservation time and usage time, reservation without access (no show), and so on.

8 Related Work

The control of remote devices through the Internet is not a new topic, and early publications on the subject date from the midnineties. For instance, [10], published in 1995, addresses teleoperation of robots via the World Wide Web. Since then, several architectures and technologies have been proposed for the design and implementation of remote operation of devices through computer networks in general, mainly the Internet. Architectures based on distributed objects, on software components, and on services are found in the literature. In [11], [12], and [1], the authors describe remote laboratories employing these architectures.

On the server side, remote laboratories have used technologies such as Common Gateway Interface (CGI), Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), Java servlets, and Microsoft's .NET platform. On the client side, Java has been the choice of many projects due to its graphic capabilities and its ability to run on web browsers as Java applets. Communication between client and server components can employ many TCP-based protocols such as HTTP, IIOP (CORBA's Internet Inter-ORB Protocol), RMI's Java Remote Method Protocol (JRMP), and Simple Object Access Protocol (SOAP).

Recent publications on remote laboratories point to the service-oriented architecture (SOA) as a trend. Web Services, a comprehensive set of standards from the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium, are strongly supported by the major software companies and open software consortia. On the server side, web services can be deployed on a wide range of modern and extensible application servers. Such application servers support virtualization, load balancing, programming of new core functions, and a comprehensive set of functions that are enabled via configuration. On the client side, web services are supported by applications running standalone or on the web browser.

The authors implemented a WebLab in mobile robotics employing SOA/web services [1]. Two motivations drove this development. The first was to replace CORBA due to its lack of client-side support, and the fact that IIOP protocol messages are constantly blocked by NAT boxes and firewalls. The second motivation was to employ service composition in the design of remote experiments. Business Process Execution Language (BPEL) is an XML-based workflow language for service composition supported by many BPEL execution engines such as ActiveBPEL and JDeveloper. Although we achieved an elegant architecture and sound implementation, problems soon emerged:

- processing of SOAP messages on the client side is cumbersome for small computers,
- efficiency of the SOAP protocol is far from adequate for real-time control of remote devices,
- SOAP implementations from different sources still lack interoperability,
- developing composite services with BPEL is not an easy task,
- specialized services (the WS-*) are specified but lack implementations.

In order to achieve more simplicity and efficiency on both client and server sides, our current WebLab architecture replaces SOAP by the REST interaction style (XML over HTTP). Although SOAP has its role in business-to-business web services, it was never adopted by successful services offered through the Internet (e.g., Google Maps). On the client side, we favor interfaces built over the web browser's native XML-based technologies. For example, the web-based teleoperation interface shown in Fig. 11 (left) employs AJAX for HTTP/XML interaction, XML DOM (an XML parser) for processing the telemetry data from the robot, and Scalable Vector Graphics (SVG) for the graphic components.

Finally, applications running on multiple organizations is a trend set by grid/cloud computing and Web 2.0 communities. We incorporated such a trend into the REALabs platform via a SAML-based Single Sign On authentication procedure. This implementation has an architecture inspired on Shibboleth [13], a production-grade Single Sign On system that requires both a servlet container (e.g., Tomcat) and the Apache HTTP server. In addition, Shibboleth runs a daemon process on the HTTP server's host. Our Single Sign On solution is similar to Shibboleth's in that both rely on SAML, but ours needs only a servlet container to operate. Shibboleth also requires the port 80 to be open for authentication, a restriction absent in our solution.

8.1 Related Projects

Although many remote laboratory projects share the service-oriented architecture adopted by REALabs-BOT (e.g., the MoCoLab project [14]), what we are considering related projects here are those that offer a domain-independent WebLab infrastructure. Three projects offer such infrastructure: the University of Deusto's WebLabs, the MIT's iLAB, and the Polytechnic University of Madrid's Cyclope WebLab.

WebLab-Deusto [15] employs a service-oriented architecture for supporting WebLabs in different application domains. The authors of WebLab-Deusto describe four generations of WebLab architectural styles [16]. The most restrictive style, called "Socket and Applet-based" by the authors, employs proprietary protocols. Interoperability and licensing issues are the major drawback of such architectural styles. The first version of WebLab-Deusto and the early versions of the authors' REAL WebLab [11] employ this architectural style.

The second architectural style, "web-based," employs web protocols and Java-based interfaces on the client side. Security issues such as Java code signing and required processing power at the client side are the major drawbacks. The third architectural style, "AJAX and web-based," replaces Java on the client side by the web browser with AJAX/Javascript. XML messages over SOAP or HTTP are the communication mechanism between the client and server sides. These three architectural styles rely on a single server connecting the WebLab resources via RS-232, GPIB, USB, etc., a solution that impacts on scalability and dependency of proprietary protocols on the server side. The version 2.0 of WebLab-Deusto and the previous version of REALabs-BOT [1] employ this architectural style.

Finally, the fourth architectural style, "Micro-server AJAX-based," retains AJAX on the client side, and connects most of the WebLab devices directly on the network. The advantage of this solution is to allow an interface running on the user's web browser to access a resource without the need of an intermediate server. Current versions of WebLab-Deusto (3.0) and REALabs-BOT employ this architectural style, although the former retains SOAP while the latter employs HTTP as interaction protocol. In REALabs-BOT, robots and cameras are fitted with micro-HTTP servers.

WebLab-Deusto implements a WebLab-independent layer with functions such as registry and locator, management of sessions, and user authentication. In REALabs-BOT, such an independent layer is provided by the REALabs platform. For security and for conserving public IP address, both WebLab-Deusto and REALabs-BOT employ private IP addresses and proxies, as shown in Fig. 6.

iLab [17] is a pioneer project developed at the Massachusetts Institute of Technology that employs web services for communicating with the WebLab. Until recently, iLab supported only "batched" (noninteractive) experiments. Interactive experiments were added by extending the functions of a central element in the iLab's architecture, the Service Broker, and adding new services for reservation scheduling, secure access, and storage of data generated by the experiments.

iLab enforces secure access via authentication and authorization, both centered in the Service Broker. Authorization for interacting with the lab servers requires credentials (tickets) issued by the Service Broker to the client application. Tickets are issued only if the user has a reservation scheduled for the time of access, and are valid during the reservation period.

Service Brokers in different domains communicate among themselves when the user on a domain wants to perform a remote experiment in another domain. This is a

TABLE 4
Comparison of WebLabs Employing the Reference Model

Model Element	REALabs-BOT	iLab	Ciclope
Federation	yes	yes	no
SLA	very simple	none	none
Group	yes	no	yes
Credential	roles	tickets	privileges
Service	REST	Web Service	SSH
Session	supported	supported	supported
Policies	reservation	reservation	reservation
Experiment	many	many	many

simple form of federation, requiring installation of the iLAB architecture in each domain.

The interactive version of iLab's architecture and the REALabs platform have some services in common: user and resource management, secure access based on authentication and authorization, and simple authorization policy (usage upon reservation). A difference is that iLab employs SOAP as the interaction client-server, while REALabs employs XML directly over HTTP. Another difference is in the architecture: REALabs employs a fully decentralized architecture without the need of lab servers or mediators as the iLab's Service Broker. The iLab's architecture follows the "web-based" style described above.

The Ciclope project [18] developed a WebLab architecture composed of a set of modules. The core module supports WebLab administration with facilities for user and group management, resource management, and resource reservation. These services are also supported by the REALabs platform. In addition to the core module, a set of lab modules provides access to lab facilities. For example, Ciclope Robot [19] supports the remote access of a four-degree-of-freedom manipulator for teaching real-time programming in practice. Ciclope Robot employs Java applets on the client side and a lab server connecting the lab devices. Client and server communicate through the SSH (Secure Shell) protocol. This is a "Socket and Applet-based" architecture.

The reference model presented in Section 2 provides a framework for comparing WebLab implementations, as it identifies the major functions that a modern remote lab must provide. For example, Table 4 compares REALabs-BOT, iLab, and Ciclope according to this reference model.

Finally, we can clearly identify some important points shared by the WebLabs presented above:

- the choice of a service-oriented architecture,
- the use of standard, "firewall-friendly" communication protocols,
- a set of domain-independent functions that can be reused across multiple WebLabs, and
- usage policy based on user subscription, authentication, and reservation.

9 Conclusions and Future Work

WebLab developers must take into account a series of requirements that few distributed applications have simultaneously. This paper presented some issues that we consider very relevant in WebLab design based on our experiences in the development of WebLabs employing many architectures and technologies. We summarize below the requirements that a WebLab designer must pursue in order to achieve a useful learning and research tool.

The first issue is access control. A WebLab must enforce user management and usage upon reservation, if its experiments operate over physical devices. Access control must also log user activities for usage statistics, failure detection, auditing, and, in some cases, accounting. User management is a database-centered application and can employ well-established technologies such as J2EE, .NET, PHP, etc.

Security must be a major focus—no matter to which access network the WebLab is connected. Physical resources must always be placed behind a firewall, and have their access subjected to stringent authentication and authorization procedures. Overheads from authorization are unavoidable, but smart designs can reduce them to acceptable values.

Quality of service, another topic addressed by this paper, can be incorporated in specific contexts. On the public Internet, we suggest flow management and experiment adaptation to maximize the benefits of the available, and usually low, bandwidth.

We also consider that WebLabs become more attractive under federated operation. A federation of WebLabs allows experiment and resource sharing. Users subscribed at one organization can use WebLabs offered by their organization as well as by the remaining organizations. Single Sign On authentication and federated authorization are the mechanisms for achieving this.

On the client side, the ideal situation is to have the access to the WebLab entirely based on a web browser. Although AJAX and other technologies available on modern web browsers are replacing Java applets and plugins, they are not sufficient in most cases. We are particularly interested in situations where the user develops his/her own experiments, evaluates and tunes them on simulated environments, and then tests the experiments on physical resources maintained by the WebLab. In such cases, the user may need a programming language more adequate than those available on web browsers (e.g., Javascript and VBScript). Languages such as Java, Python, and Matlab seem more appropriate for coding experiments in the engineering domain. Access from powerful mobile devices is also a trend, and some WebLabs already offer it to their users [20].

Finally, in terms of architecture, we agree that a "Microserver AJAX-based Solution" as described in [16] is the state-of-the-art architecture for WebLabs. We speculate that only web-enabled devices will be part of a WebLab physical resources portfolio in the near future.

9.1 Future Work

We are currently evaluating the Internet Multimedia Subsystem (IMS) architecture as a framework for WebLab developments. IMS has a comprehensive set of functions related to user management, session management, and federated authentication and authorization. We are considering porting part of the REALabs functions to the OpenIMS developed by the Fokus Institute [21]. From the standpoint of the IMS, WebLab experiments are IMS applications.

Another focus of our research is federated authorization. The definition of ontologies to achieve a common semantics for describing credentials, resources, and usage and reservation restrictions, among others, is a necessary step for federated authorization. We are currently evaluating Apache's Imperius and JBoss' Drools frameworks to model authorization policies. Imperius is based on Simplified Policy Language (SPL) from the Distributed Management Task Force, while Drools is based on the Java Rule Engine API (JSR-94) from the Java Community Process.

ACKNOWLEDGMENTS

The authors would like to thank Fapesp, RNP, and CAPES for supporting this research. Thanks to the graduate students who contributed to the implementation and operation of the REALabs-BOT WebLab. Thanks also to the reviewers for their relevant suggestions.

REFERENCES

[1] P. Coelho, R. Sassi, E. Cardozo, E. Guimarães, L. Faina, R. Pinto, and A. Lima, "A Web Lab for Mobile Robotics Education," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '07),* Apr. 2007.

[2] D. Moraes, P. Coelho, E. Cardozo, E. Guimarães, T. Johnson, and F. Atizani, "A Network Architecture for Large Mobile Robotics Environments," *Proc. Second Int'l Conf. Robot Comm. and Coordination (ROBOCOMM '09),* Apr. 2009.

[3] P. Coelho, D. Moraes, E. Cardozo, E. Guimarães, T. Johnson, and F. Atizani, "A Network Architecture for Mobile Robotics," *Proc. 27th Brazilian Symp. Computer Networks,* May 2009.

[4] C. Salzmann, D. Gillet, and P. Mullhaupt, "End-to-End Adaptation Scheme for Ubiquitous Remote Experimentation," *Personal and Ubiquitous Computing,* vol. 13, no. 3, pp. 181-196, Mar. 2009.

[5] "Guide to Open-Source Identity Management Software,"white paper, Sun Microsystems, Apr. 2009.

[6] L. Richardson and S. Ruby, *RESTful Web Services.* O'Reilly, 2007.

[7] E. Cardozo, E. Guimarães, F. Paolieri, and V. Pinto, "REALabs-BOT: A WebLab in Mobile Robotics over High Speed Networks," *Proc. IFAC Workshop Networked Robotics (NetRob '09),* Oct. 2009.

[8] J. Jackson, "Microsoft Robotics Studio: A Technical Introduction," *IEEE Robotics and Automation Magazine,* vol. 14, no. 4, pp. 82-87, 2007.

[9] R. Siegwart and I. Nourbakhsh, *Introduction to Autonomous Mobile Robots.* MIT, 2004.

[10] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, "Desktop Teleoperation via the World Wide Web," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '95),* May 1995.

[11] E. Guimarães, A. Maffeis, J. Pereira, B. Russo, E. Cardozo, M. Bergerman, and M. Magalhães, "REAL: A Virtual Laboratory for Mobile Robot Experiments," *IEEE Trans. Education,* vol. 46, no. 1, pp. 37-42, Feb. 2003.

[12] E. Guimarães, A. Maffeis, R. Pinto, C. Miglinski, E. Cardozo, M. Bergerman, and M. Magalhães, "REAL—A Virtual Laboratory Built from Software Components," *Proc. IEEE,* vol. 91, no. 3, pp. 440-448, Mar. 2003.

[13] Internet2 Middleware Initiative, Shibboleth, http://shibboleth.internet2.edu, Mar. 2009.

[14] C. Buiu and N. Moanta, "Using Web Services for Designing a Remote Laboratory for Motion Control of Mobile Robots," *Proc. AECE World Conf. Educational Multimedia, Hypermedia and Telecomm.,* June 2008.

[15] J. García-Zubia, P. Orduña, I. Angulo, J. Irurzun, and U. Hernández, "Towards a Distributed Architecture for Remote Laboratories," *Int'l J. Online Eng.,* vol. 4, special issue 1, http://www.i-joe.org, pp. 11-14, 2008.

[16] J. Garcia-Zubia, D. López-Ipinã, and P. Orduña, "Evolving Towards Better Architectures for Remote Laboratories: A Practical Case," *Int'l J. Online Eng.,* vol. 1, no. 2, http://www.i-joe.org, 2005.

[17] J. Hardison, K. DeLong, P. Bailey, and V. Harward, "Deploying Interactive Remote Labs Using the iLab Shared Architecture," *Proc. 38th ASEE/IEEE Frontiers in Education Conf.,* Oct. 2008.

[18] R. Cedazo, D. Lopez, F. Sanchez, and J. Sebastian, "Ciclope: FOSS for Developing and Managing Educational Web Laboratories," *IEEE Trans. Education,* vol. 50, no. 4, pp. 352-359, Nov. 2007.

[19] D. Lopez, R. Cedazo, F. Sanchez, and J. Sebastian, "Ciclope Robot: A Remote Laboratory for Teaching Embedded Real Time Systems," *Proc. IEEE Int'l Symp. Industrial Electronics (ISIE '07),* 2007.

[20] R. Martin, L. Nomdeleu, P. Wirz, and P. Sanz, "Robotics Internet Tele-Lab: Programming Using Mobile Devices," *Proc. Int'l Conf. Multimedia and Information and Comm. Technologies (ICT '09) on Education,* Apr. 2009.

[21] Fraunhofer Fokus Inst., Open IMS Core, http://www.openimscore.org, Mar. 2009.

Eliane G. Guimarães received the BS degree in computer science and the MS and PhD degrees in electrical engineering from the University of Campinas in 1977, 1990, and 2004, respectively. She is a research scientist at the Robotics and Computer Vision Division, Information Technology Center Renato Archer, Campinas, Brazil. Her research interests include networked robotics, WebLabs, and distributed systems.

Eleri Cardozo received the BS degree in electrical engineering from the University of São Paulo, Brazil, in 1978, the MS degree in electrical engineering from the Technological Institute of Aeronautics, Brazil, in 1981, and the PhD degree in electrical engineering from Carnegie Mellon University in 1987. He is a professor of electrical and computer engineering at the School of Electrical and Computer Engineering, University of Campinas, Brazil. His research interests include distributed systems, computer networks, and software engineering.

Daniel H. Moraes received the BS degree in computer engineering from the University of Campinas, Brazil, in 2004. He is currently working toward the MS degree in the School of Electrical and Computer Engineering, University of Campinas, Brazil.

Paulo R. Coelho received the BS and MS degrees in computer engineering from the University of Campinas, Brazil, in 2004 and 2007, respectively. Currently, he is an assistant professor at the Federal University of Uberlândia, Brazil.