2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

A Web Lab for Mobile Robotics Education

Paulo R.S.L. Coelho[1], Rodrigo F. Sassi[1,2], Eleri Cardozo[1], Eliane G. Guimarães[2], Luis F. Faina[3], Alex Z. Lima[1], Rossano P. Pinto[1]

***Abstract*— This paper presents an architecture for building remote access laboratories (or Web Labs) following the service-oriented computing approach. In this architecture the application's building blocks are services that can be recursively composed resulting in more comprehensive services. Remote access laboratories can benefit of this approach. Every lab resource (physical or logical) is modeled and implemented as a service (in our case, a Web service) and lab experiments are assembled by composing these services. A Web Lab built according to this architecture is presented with examples of remote experiments in the field of mobile robotics.**

I. Introduction

Remote Access Laboratories or *Web Labs* have been proposed as powerful tools for the sharing of expensive equipment and for bringing experimentation into distance learning. The challenge regarding Web Labs implementations is to provide an infrastructure where remote experiments are easily assembled and modified.

This paper proposes a service-oriented computing approach for building Web Labs. Considered as an evolution of distributed computing, service-oriented computing is defined as "the approach that uses services as fundamental elements in application development" [1]. Using composition mechanisms (or orchestration), more comprehensive services can be built from existing ones. Since the composed service is itself a service, it can take part in further compositions [2].

Implementing Web Labs according to service-oriented computing brings some remarkable benefits. Firstly, lab resources (physical and logical) are modeled and implemented as services, for instance, a robot exports a set of services, each one performing an specific function (sensing, navigation, and so on). Secondly, experiments can be synthesized as a composition of services. In this way, new experiments may use existing ones as units of composition, and the updating of an experiment is restricted to its composition's logic. Finally, the concept of federation of services allows Web Labs to use resources maintained by other Web Labs located in different administrative domains.

This paper describes an architecture based on composition and federation of services for building Web Labs. A complete implementation of this architecture using the Web services technology and a Web Lab in mobile robotics built according to the architecture are presented as well. The paper is organized as follows. Section II presents some related work; Section III presents a service-oriented architecture for Web Labs; Section IV provides an implementation of this architecture; Section V describes a Web Lab in mobile robotics developed according to the implemented architecture; and Section VI closes the paper with conclusions and future work.

[1] State University of Campinas, Campinas-SP, Brazil; [2] Renato Archer Research Center, Campinas-SP, Brazil; [3] Federal University of Uberlândia, Uberlândia-MG, Brazil pcoelho@dca.fee.unicamp.br

II. Related Work

Web Labs were initially developed with well known Web technologies, such as Java *applets* on the client side [3] and server-side extensions based on CGI (Common Gateway Interface), ASP (Active Server Pages), or JSP (Java Server Pages) [4][5]. Distributed objects approaches based on Java RMI (Remote Method Invocation) and CORBA (Common Object Request Broker Architecture) were proposed as well [6]. Other implementations require platform dependent software in the client terminal [7]. All of these approaches have in common the limited software reuse, customization, and interoperability. These limitations are due to the low granularity of software artifacts and the adoption of interaction protocols that are usually blocked by network firewalls.

More recently, Web Lab architectures based on software components were proposed [8]. Although components increase software reuse and customization, component-based architectures still lack interoperability among different platforms and administrative domains. Architectures based on Web services are an attempt to build Web Labs that fulfill the requirements of software reuse, customization, and interoperability [9][10].

The related work closer to the described in this paper is the iLAB project [11]. iLAB identifies a set of administrative functionalities such as user subscription, authorization and authentication, group management, and access control. Such tasks are common to all Web Labs and are decoupled from the domain specific functionalities the Web Labs support. The administrative tasks are managed by a *Service Broker* that mediates the access to the associated Web Labs. The Service Broker supports in some way the federated operation of Web Labs. The architecture proposed in this paper shares some similarities with iLAB, for instance:

- decoupling between use and management of Web Labs;

- use of Web services for management and access to Web Labs;
- federated operation of Web Labs.

iLAB makes no restriction about the implementation of the Web Labs, except the interfaces for attaching to the Service Broker.

The architecture proposed in this paper differs from iLAB in three important aspects:

1) the scope of the federation of Web Labs is enlarged by allowing Web Labs to share resources (not only subscribed users);
2) instead of rely on a centralized access control (Service Broker), each federated Web Lab sets its own access policies;
3) the federation imposes that each federated Web Lab must implement a minimum security infrastructure.

Composition plays a key role in this proposed architecture for Web Labs. Intra-domain composition allows a Web Lab to combine its resources within experiments. Inter-domain composition allows a Web Lab to combine its own resources with resources offered by other Web Labs. Inter-domain composition enlarges the range of experiments a Web Lab can offer and constitutes the truly motivation for a federated operation of Web Labs.

The architecture also gives each Web Lab the freedom to set its owns access rules. Policies is a flexible way to set the condition upon which users can access the Web Lab. For instance, a policy must require that foreign users be authenticated, present credentials issued by the Web Lab he/she was subscribed, and has a time slot already reserved to perform experiments. The flexibility of policies allows a Web Lab to take part of multiple federations by defining a policy set for each federation it takes part.

Finally, the federation imposes a minimum set of security constraints to its federated Web Labs. For instance, each Web Lab must support an infrastructure for authenticating users and services. This infrastructure is domain-independent and can be supplied by the federation and integrated into existing Web Labs.

III. A Service-oriented Architecture for Web Labs

The service-oriented architecture for building Web Labs defines a reference model for Web Labs and a family of services for supporting the model elements.

The reference model for Web Labs is shown in Fig. 1. The UML concept diagram shows the main elements of the model as well as the relationships among them. The two central elements are *Participant* and *Web Lab*. A participant can be an individual (*User*) or a *Group*. A group is a collection of participants formed by users or (sub)groups. The line connecting participants to Web Labs represents the usage relationship. To use a Web Lab a participant must have assigned the proper *Credentials* and establish one or more *Sessions* with the Web Lab. Notice that credentials and sessions refer to a specific participant accessing a specific Web Lab. Examples of credentials include the user's identification, roles (e.g., student, instructor), permissions (e.g., usage, management), and privileges (e.g., group leader).

Sessions are responsible for managing the interaction between a participant and a Web Lab. Sessions maintain state information related to the interaction such as the identification of the participant, the remaining access time, and the actions the participant has performed.

Web Labs publish the *services* they offer. Services are units of composition on interaction with the Web Lab employed for purposes such as access, interaction, communication, and management. *Experiments* offered by Web Labs are modeled as a composition of services. For instance, an experiment in environmental mapping can compose a service of locomotion (e.g., random walk) with a service of telemetry (e.g., sonar readings). A Web Lab maintains a set of *Resources*, both physical such as robots and cameras, and logical such as robot simulators and navigation maps. Resources are manipulated remotely through services. A Web Lab has a self-relationship that models a federation of Web Labs. The model does not impose a particular mechanism for building such a federation.

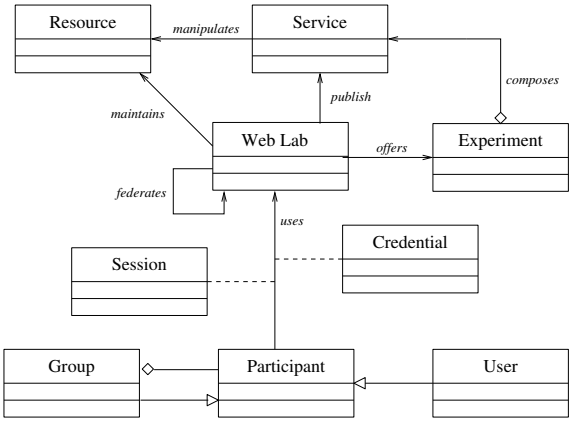

Fig. 1. A reference model for Web Labs.

Looking at the reference model presented in Fig. 1 it is possible to aggregate the concepts around the participant (user, group, credential), the Web Lab (resource, experiment), and the usage of the Web Lab by a given participant (session). This suggests a need for services for managing participants, Web Labs, and sessions.

The service in charge for managing participants offers functionalities for the management of users and groups. User management functions include subscription and unsubscription, and assignment of credentials with proper roles, permissions, and privileges. Group management functions include the creation, updating, and removing of groups.

The service in charge for managing Web Labs offers functionalities for the management of resources and ex-

periments available in the Web Lab. Resource management functions include resource registration and usage restrictions such as periods of availability. Experiment management functions include activation of experiments and assignment of restrictions to experiments such as reservation restrictions (e.g., maximum reservation time) and accessing restrictions (e.g., minimum set of credentials).

The participant and lab management services hold long-term information and provide both programatic and human-operated interfaces. The use of services provided by these interfaces are subject to authentication and authorization.

The services in charge for managing sessions expose interfaces for the management of interactions between a participant and the Web Lab. The architecture identifies three basic session types: access, interaction, and communication sessions. Access sessions manage the access of a participant to the Web Lab according to its credentials. Interaction sessions manage the remote execution of experiments maintained by the Web Lab. Communication sessions manage the exchange of information between the users and the Web Lab.

Access management functions include authentication and authorization. Authentication establishes the identity and credentials of the user while authorization decides if the credentials suffice for granting access to the Web Lab resources. A set of usage policies define the rules for accessing the Web Lab resources and experiments as a function of the user's credentials.

Interaction sessions provide the correct flow of service invocation demanded by an experiment. This flow is specified as a composition of services supported by the Web Lab. Interaction management functions include the initiation and termination of communication sessions, and logging functions. Interaction sessions are subjected to the existence of a previously established and valid access session.

Communication management functions include configuration and connection of producers and consumers of information, management of quality of service for multimedia communication, support for person-to-person communication, and logging of information.

Figure 2 illustrates the major components of the architecture in a UML package diagram. The five packages represent the general components for supporting Web Labs and offer a hint for aggregation of services according to the reference model described above.

IV. Architecture Implementation

The five service classes shown in Fig. 2 were implemented using the Web services technology. Participant and Web Lab Management Services are traditional database-centric services and will not be detailed. Access Management and Communication Services are domain independent services and are detailed in the sequence.

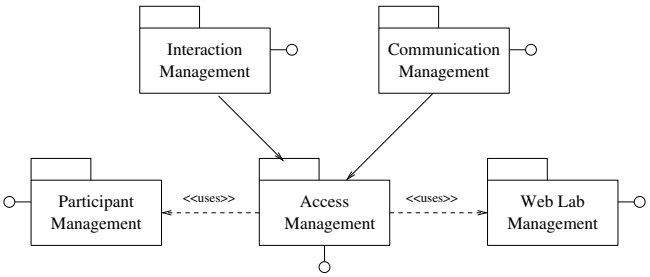

Fig. 2. Major components of a service-oriented architecture for managing Web Labs.

Interaction Management Services are domain dependent services and will be detailed in Section V.

A. Access Management Service

The access management service implements policy-based access control according to the Extensible Access Control Markup Language (XACML) [12]. XACML allows a Web Lab to define access policies for each experiment it supports. When a user attempts to access an experiment, the access management service is invoked. The service requests user authentication and invoke the participant and Web Lab management services in order to get the user's credentials and experiment restrictions such as credentials necessary for execution and current reservation data (reserving user, reserved time slot, etc.). User and experiment information is then submitted to the XACML engine for evaluation. XACML policies typically check if the credentials suffice for accessing the experiment, and if the user holds a valid reservation for executing it. If the checking succeeds, the user is allowed to execute the experiment.

SunXACML, a public domain implementation of XACML from Sun Microsystems was employed as XACML engine [13].

B. Communication Management Services

The Communication Management Service is implemented as a information diffusion service according to the publish/subscribe model. The service propagates XML documents generated by producers to the subscribed consumers. The service offers a Web service management interface for subscribing and unsubscribing producers and consumers, and for submitting XML documents for diffusion.

During the subscription, a consumer may supply a XPath expression that is evaluated when documents are submitted. The result of this evaluation is propagated to the consumers if it results in a valid document (the submitted one or part of it). If a consumer supplies a notification interface, the service invokes the operation *push* in this interface in order to transfer the document as soon as it was processed (push style). Alternatively, the consumer may inquire the service for new submitted documents (pull style).

If a producer states a lifetime for a submitted XML document, the document persists on the service during this lifetime, otherwise, the service discards the document after it was delivered. The kind of information present in these documents can serve several purposes, for instance, notification (e.g., alarms); configuration (e.g., formats and sizes of audio and video); user generated information (e.g., chat messages); synchronization messages (e.g., whiteboard updates); and logging information (e.g., session-related data).

A media streaming service was implemented above the diffusion service. Media producers publish their capabilities, media ports, and negotiation interfaces. Media consumers access these informations and, if the case, negotiate a proper media quality and format through the producer's negotiation interface. The media flow is not transferred through the diffusion service.

Other Web Lab implementations may choose different approaches for supporting communication. Both commercial and open source products supporting media streaming, conferencing, and messaging are available from many sources and can serve as a basis for the Communication Management Service. In this case, a Web service wrapper to control these products must be implemented.

V. GigaBOT Web Lab

GigaBOT is a Web Lab for mobile robotics education implemented following the architecture described above. Being domain independent, access and communication services don't need any adaptation to the mobile robotics domain. The interaction services, otherwise, have been developed for this specific domain and will be described next.

A. Lab Infrastructure

The GigaBOT Web Lab operates two ActivMedia's Pioneer P3-DX robots with sixteen sonars and protection bumpers. One robot is fitted with on-board processor, wireless network interface (802.11g), and a Canon VC-C4 on-board camera with PTZ (*pan/tilt/zoom*) and connected to framegrabber. The second robot does not have on-board processor, being controlled via serial port by an HP IPaq H5555 handheld with 802.11b wireless network. The handheld runs the Familiar Linux v0.8.1 operating system. This robot is fitted with an Axis 206W wireless network camera. An Axis 213 PTZ network camera is available to allow a panoramic view of the execution of experiments.

The software infrastructure at the lab's side is composed of a Web service container (Apache Axis Java), a JSP/Servlet container (Apache Tomcat), a UDDI (Universal Description Discovery Integration) server (jUDDI), and a relational database (MySQL). Axis, Tomcat, and jUDDI are supplied by the Apache Software Foundation [14]. Containers and databases execute on a dual-Xeon processors running the GNU/Linux operating system. The infrastructure is complemented by the robots' application programming interface (API).

Figure 3 illustrates the infrastructure components and the interaction protocols on the server side. The components on the client side consist of common Web browsers, Java clients (Java processes that run on the client desktop) loaded from the Java Web Start application launcher, and MPEG-4 media players such as MPlayer or Windows Media Player. Java clients discover Web services by querying the UDDI service and invoke them through the SOAP (Simple Object Access Protocol) protocol.

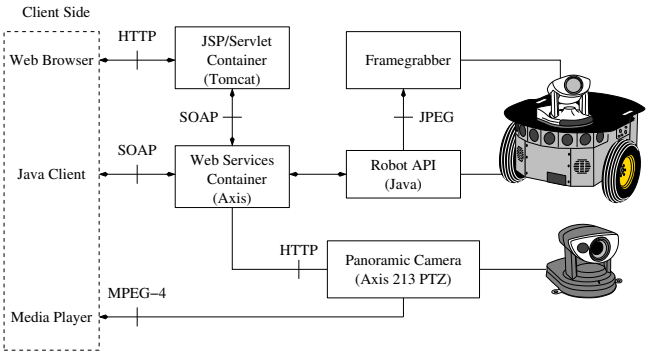

Fig. 3. GigaBOT Web Lab Infrastructure.

B. Interaction Services

Interaction services are services supporting the interaction with the Web Lab domain-specific resources. The GigaBOT Web Lab provides six interaction services: locomotion, telemetry, vision, panoramic camera, actions, and code submission services.

The locomotion service exports operations for moving and turning the robot to a relative or absolute position and for adjusting and limiting the speed and acceleration of the robot. Operations supporting robot locomotion can be synchronous (blocking while the movement completes) or asynchronous.

The telemetry service allows the obtaining of telemetry data such as current robot position, sonars readings, robot physical dimensions, and voltage supplied by the battery.

The vision service allows to control the robot's on-board camera. The control parameters includes the absolute and relative values of pan (horizontal movement), tilt (vertical movement), zoom, and focus. The service also allows the capture of a static image in JPEG format.

The panoramic camera service provides access to the Axis 213 PTZ resource. The service supports the same operations as the vision service plus operations for video recording.

The action service wraps some of the built-in robot actions into Web services. An action establishes a set of basic operations which, together, perform a specific task. Each action added to the robot is inserted into a queue

with a given priority. Actions are performed in a sequence given by their priorities. Nine actions are available: six related to prevention and recover from collisions (e.g., moving with frontal and lateral obstacle avoidance), and three actions related to locomotion (e.g., moving the robot to a giving point).

Finally, the code submission service allows users to submit and execute C++, Java, or Python code on a server connected to the robot. The submitted code must control the robot by calling the methods supplied in the robot's application programming interface (API). This service also provides methods for controlling the execution of the submitted program, for instance, retrieving its process identifier (pid) and standard output printouts, and controlling its state of execution (suspend, resume, kill).

C. Lab Experiments

The experiments provided by the GigaBOT Web Lab are programmed using the composition of the interaction, access, and communication services. The level of difficulty of a proposed experiment can vary from the entire coding of a robot navigation algorithm to the adjusting of parameters of a supplied algorithm. Intermediate levels include the coding of a new action. An experiment can be executed on the user's computer or on a server located at the Web Lab and connected to the robot.

Six experiment classes were developed for the Web Lab:

1) Basic Telemetry: experiments that allow to control the robot as well as to inspect the sonar readings. Control can be performed
   - by issuing basic commands (move and turn) via a graphical user interface;
   - by drawing a route with the mouse on a navigation map;
   - by combining a set of pre-programmed actions (e.g., random walking with obstacle avoidance).
2) Navigation on Structured Environments: experiments that allow the mapping of the robot's environment and navigation based on environmental maps.
3) Navigation on Non Structured Environments: experiments that allow navigation and environmental mapping simultaneously.
4) Vision-based Navigation: experiments that allow navigation using the robot's vision system.
5) Code Submission: allows the user to write navigation code and submit it directly to the robot.
6) Cooperative Robotics: experiments employing two robots developing cooperative tasks.

Figure 4 shows the interface for the Basic Telemetry experiments. The interface consists of two panels with images from the on-board and panoramic cameras (left); a navigation map (top right) with the actual robot position, the sonar readings represented as lines emanating from the robot, and the already detected obstacles; and a tabbed panel for navigation (bottom right).

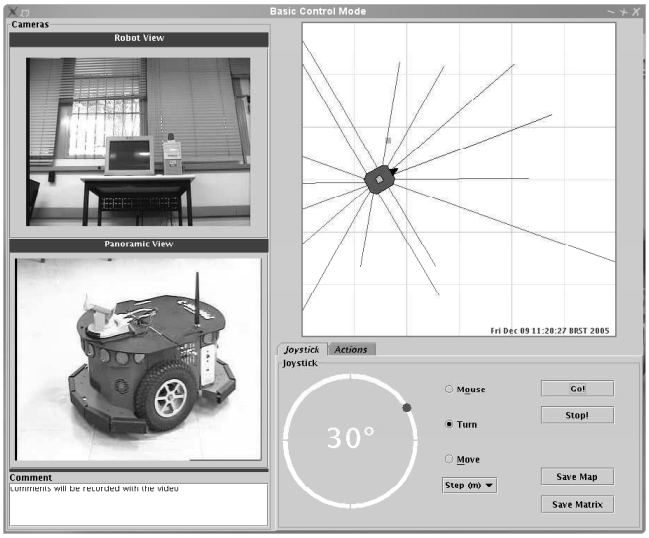

Fig. 4. Interface for the basic telemetry experiments.

Experiments are built by composing the six interactive services. The composition logic can be coded in general purpose programming languages such as Java or C++, or expressed in specialized composition languages such as BPEL (Business Process Execution Language) [15]. BPEL allows the specification of workflows of service calls in XML. The advantage of BPEL is the availability of sophisticated engines supporting graphical edition as well as execution and debugging of BPEL scripts.

As an example of composition with BPEL consider an experiment in the Vision-based Navigation class where the robot must follow a colored strip on the floor using solely its on-board camera. This experiment composes the vision, motion, and communication services with a image processing algorithm written using the *Java Advanced Imaging* (JAI) system [16]. Figure 5 illustrates the interface of this experiment. The image on the left shows the image captured from the on-board camera (highlighted in black). The image on the center shows the detected borders of the strip, and the image on the right shows the trajectory computed from the other two images. The interface allows the user to adjust the maximum speed in which the robot follows the strip as well as some parameters of the image processing algorithm. A movie of the experiment is recorded from the panoramic camera.

The composition logic employs a cycle with the following steps:

1) invoke the communication service to start the movie recording from the panoramic camera;
2) invoke the vision service to acquire an image from the robot's on-board camera;
3) invoke the image processing algorithm to identify the strip. If no strip is identified do: (i) invoke the locomotion service to stop the robot; (ii) invoke the

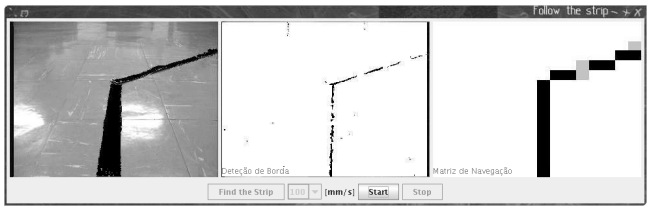

Fig. 5. Navigation using the robot vision.

communication service to stop recording; (iii) end of experiment.

4) invoke an heuristic to compute a movement that will position the strip in the center of the image;
5) invoke the locomotion service to perform the movement synchronously;
6) go to step 2.

Since this experiment runs on the client side, the network bandwidth and delay strongly impacts on the performance of the experiment. In order to assess the bandwidth and delay demanded by the Web Lab, the vision-based navigation experiment was performed from four different access networks: local area, campus, residential ADSL (Asymmetric Digital Subscriber Line), and virtual private networks. Local area and campus networks are Ethernet-based networks; ADSL service is limited to 2 Mbit/s; and virtual private network (VPN) is a dedicated network with Gigabit Ethernet links. Table I shows the maximum speed the robot is still able to follow the strip and the network bandwidth consumed by the experiment for these four networks.

Access Network	Max. Speed (mm/s)	Bandwidth (Kbps)
Local Ethernet	170	23,500.00
Campus Ethernet	90	1,250.00
ADSL	30	970.00
VPN	200	23,700.00

TABLE I
EXPERIMENT PERFORMANCE IN DIFFERENT ACCESS NETWORKS.

VI. Conclusions

A reference model for Web Labs was introduced in this paper with the objective of identifying the major components a Web Lab must implement. A domain independent service-oriented architecture adhering to this model was also detailed. In this architecture, lab experiments are designed as a composition of services offered by the Web Lab. This approach favors coarse-grained reusability when building new experiments or adapting existing ones to new contexts. The sophisticated composition engines available today reduce drastically the effort for deploying remote accessible experiments.

The architecture also supports the federated operation of Web Labs. The federation relies on inter-domain authentication for both users and service calls. A policy-based authorization scheme provides flexibility on the access control of Web Labs. These schemes take advantage of XML-based standards for identity control (SAML) and policy-based access control (XACML).

Finally, the next step in Web Lab design and operation is to devise novel federation mechanisms. The proposed architecture is an step towards this goal. Currently we are building a federation of Web Labs in the field of Control Systems and Robotics.

Acknowledgments

This project is supported by the Brazilian National Teaching and Research Network (RNP) through the Giga Project. The authors thank the undergraduate students Victor Perticarrari, Danielle Santos, and Ewerton Barbosa for helping in software development.

References

[1] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing: Introductions. *Communication of the ACM*, 10(46):24–28, Oct 2003.
[2] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The Next Step in Web Services. *Communication of the ACM*, 10(46), Oct 2003.
[3] J. Hua and A. Ganz. Web Enabled Laboratory (R-LAB) Framework. In *ASEE/IEEE Frontiers in Education Conference*, Boulder, USA, Nov 2003.
[4] A. Khamis, D. M. Rivero, F. Rodríguez, and M. Salichs. Pattern-based Architecture for Building Mobile Robotics Remote Laboratories. In *IEEE International Conference on Robotics and Automation*, Taiwan, Sept 2003.
[5] J. A. del Alamo, L. Brooks, C. McLean, J. Hardison, G. Mishuris, V. Chang, and L. Hui. The MIT Microelectronics WebLab: a Web-Enabled Remote Laboratory for Microelectronic Device Characterization. In *World Congress on Networked Learning in a Global Environment*, Berlim, Germany, May 2002.
[6] E.G Guimarães, A.T. Maffeis, J.L. Pereira, B.G. Russo, E. Cardozo, M. Bergerman, and M. Magalhães. REAL: A Virtual Laboratory for Mobile Robots Experiments. *IEEE Transactions on Education*, 46(1), Feb 2003.
[7] D.Z. Denis, A. Bulancak, and G. Ozcan. A Novel Approach to Remote Laboratories. In *ASE/IEEE Frontiers in Education Conference*, Boulder, USA, Nov 2003.
[8] E.G. Guimaraes, A.T. Maffeis, R.P. Pinto, C.A. Miglinski, E. Cardozo, M. Bergerman, and M. Magalhaes. REAL: A Virtual Laboratory Built from Software Components. *Proceedings of the IEEE*, 91(3), March 2003.
[9] M. Casini, D. Prattichizzo, and A. Vicino. The Automatic Control Lab. *IEEE Control Systems Magazine*, 25(1), June 2004.
[10] A. Rasche, B. Rabe, M. von Lowis, J. Moller, and A. Polze. Real-time robotics and process control experiments in the Distributed Control Lab. *IEE Proceedings - Software*, 152(4), Oct 2005.
[11] MIT. iCampus/iLAB Project, 2007. `http://icampus.mit.edu/ilabs`.
[12] OASIS. eXtensible Access Control Markup Language (XACML), May 2006. Available at `http://www.oasis-open.org`.
[13] Sourceforge. *Sun's XACML Implementation*, 2007. `http://sunxacml.sourceforge.net`.
[14] Apache Software Foundation. *Apache Project*, 2007. `http://www.apache.org`.
[15] OASIS. Web Services Business Process Execution Language Version 2.0, May 2006. Available at `http://www.oasis-open.org`.
[16] Sun Microsystems. Java Advanced Imaging (JAI) API, January 2007. Available at `https://jai.dev.java.net/`.