iGEMS – RESOURCE-ORIENTED SYSTEM

FOR TELEROBOTICS TESTBEDS

By

NHAT D. NGUYEN

Bachelor of Science in Computer Science

Oklahoma State University

Stillwater, Oklahoma

2009

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2011

iGEMS – RESOURCE-ORIENTED SYSTEM

FOR TELEROBOTICS TESTBEDS

Thesis Approved:

Dr. Johnson Thomas

Thesis Adviser

Dr. Subhash Kak

Dr. David Cline

Dr. Sheryl A. Tucker

Dean of the Graduate College

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

NOMENCLATURE

JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
ROA	Resource-Oriented Architecture
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language

CHAPTER I

INTRODUCTION

1.1 Motivations

Currently, robotics is a very popular and interesting research area. Many scientists and researchers are spending much time and effort in this field and the field is rapidly evolving. Robotics and its sub area Telerobotics both are not new but they have great potential and promising applications. Applications of robots are presently everywhere, from home to outer space.

Although the future appears to be bright, there exist two big problems which may limit further developments in this field. Firstly, the cost of robotic experiments remains very high. Although there are many kinds of robots in the market today, the cheaper ones are usually not sophisticated enough for research purposes. It is difficult for educational institutions to conduct large-scale robotic experiments which require many robots and other facilities as the total cost is huge. For example, experiments for swarm intelligence usually require hundreds or even thousands of robots which make the cost prohibitive. Similarly, advanced intelligent humanoid robotic experiments conducted with some fifteen thousand dollar robots are out of the range of affordability for most educational institutions

In addition to the excessive costs, technology differences are another disadvantage. In the world of robotics, there exist many differences among robotic development platforms, interfaces, programming languages, controllers and other related technologies. A researcher does not have enough time to learn all of the different technologies to take advantage of all the advanced robotic options and features that are available for his experiments. He needs a lot of time to understand these robots at a low level or sometimes has to do reverse engineering to reveal their communication interfaces. A researcher faces additional challenges in the field of robotics as he is not well versed in hardware issues.

These two above serious problems of finance and technology will limit the application of robotics in the future; as more different varieties of robots are produced, the more confusion they cause to robotic scientists. Though, there are some efforts to standardize both in the software and hardware domains, they only partly solve the problems. For example, Microsoft Inc. presents Microsoft Robotics Developer Studio as a solution to provide a common environment for many kinds of robots through a common platform and programming language. However, it is still difficult to use and lacks strong support from the robotic industry. Currently, there are only standards for industrial robots, while standards in other robotic areas are not defined transparently. Because of this, in next section, we propose a solution to solve the two above critical problems to create a better environment to do research for robotics scientists, both from a financial and technology perspective.

1.2 Proposed Solution

In this thesis, we propose a Resource-Oriented System for Telerobotics Testbeds to address the above problems of cost and technology. This system creates a layer of abstraction for all robotic and other devices connected to it as resources. When validated by the system, these facilities can be exposed to users as a set of resources adapting to a model called REST (Resourced-Oriented

Architecture and REST as will be defined and explained in detailed in Chapters 2 and 3). An abstract layer is created as a set of secure and responsive web services which when consumed by client programs can provide many benefits. Different kinds of development platforms, development environments, and programming languages can seamlessly use these resources regardless of the kind of final applications created. There is no difference in delay time or communication protocols between web applications, web services, mobile applications or traditional desktop applications. The proposed framework enables scientists to conduct experiments without having to own real robots or spend much time to learn about robotic interfaces. All they have to do is to log in to the system, schedule their time, connect their programs to the resources, and start conducting their experiments. This system also opens collaborative opportunities for robotic scientists to combine their local robotic facilities with remote testbeds or to use remote testbeds.

1.3 Research Objective

The objective of this thesis is to design, implement, and test the telerobotics System with the RESTful Model. We propose to conduct some interesting demonstrations of the system on real testbeds to prove its convenience and efficiency. The research objectives are as follows:

- Review existing similar systems and identify their deficiencies.
- Compare and show the benefits of the proposed Resource-Oriented Architecture and RESTful Model over other architectures and models.
- Present the significant steps of RESTful web services design.
- Analyze the architecture and design of the system.
- Deploy the system on real servers.
- Demonstrate some typical scenarios on the system.
- Prove the convenience and efficiency of the system on real testbeds.

1.4 Research Contributions

The proposed pure Resource-Oriented Restful model is a new and convenient solution for a Telerobotics system. This system creates a convenient and efficient environment for scientists to do their experiments within their available financial resources and limited understanding of technology. The system will provide scientists at small institutions an opportunity to access more advanced robotic facilities at other universities using a logical schedule. This will decrease the cost and build a friendly collaborative environment to take advantage of robotic testbeds from different institutions in a harmonic and effective way. Moreover, the system can assure secured communication channels to protect research results.

1.5 Outline of the Thesis

This thesis is organized as follows: Chapter two reviews past and current similar Telerobotics systems and their applications in many areas. Chapter two also analyzes and identifies the disadvantages of current Telerobotics systems compared to our system in two specific aspects, convenience and efficiency; Chapter three presents the detailed software engineering process for our overall system. This chapter is strongly focused on the construction phase. Chapter four describes the installation of the system on servers and deployment of robotic testbeds. Chapter five validates the convenience and efficiency of the system with some typical scenarios on real robotic testbeds. Chapter six explores some findings during development and the testing phase of the system. Chapter seven concludes the thesis with some potential future work.

CHAPTER II

REVIEW OF LITERATURE

2.1 Introduction to Robotics and Telerobotics

2.1.1 Robotics

The word *robotics* and *robot* are originated from the word *robota* in Czech which means labor. This word is coined by Czech writer Karel Capek (1890-1938) in 1921 in his play *Rossum's Universal Robots* ("Who did," 2011). In the modern world, "Robotics is the branch of technology that deals with the design, construction, operation, structural disposition, manufacture and application of robots." ("Robotics," Oxford, 2011) "Robotics is also a technology dealing with the design, construction, and operation of robots in automation." ("Robotics," Webster, 2011) So the very premise concept of the robot is fiction, not scientific. Writers are the first people who took the fundamental steps for this research area. The most important robotic rules are Issac Asimov's Three Laws of Robotics which are stated in his famous science fiction novel, Runaround. These rules can be compared to Issac Newton's Three Laws of Motion in physics. ("Three Laws," 2011).

- "A robot may not injure a human being or, through inaction, allow a human being to come to harm.

- A robot must obey any orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Law." ("Three Laws," 2011)

The above three laws show that a robot should not be harmful to itself and anything else around it, and the robot must not disobey its creators and owners, human beings. In this meaning, robots are still machines; they should not be highly intelligent (at least not more than human beings).

Today, robots are divided into two groups, mobile robots and industrial robots. Mobile robots are ones which can move themselves by using wheels (2, 3 or 4 wheels are most popular), tracks, propellers, fins, jet engines, etc. Mobile robots (Figure 2) are automatic machines that are capable of movement in a given environment. They have the capability to move around in their environment and are not fixed to one physical location. Meanwhile, industrial robots mostly are jointed robotic arms which are attached to fixed surfaces. These robotic arms (Figure 1) usually serve specific tasks such as grinding, welding, painting, etc. In this thesis, both kinds of robots are implemented on the system with some different scenarios. ("Mobile Robot," 2011)

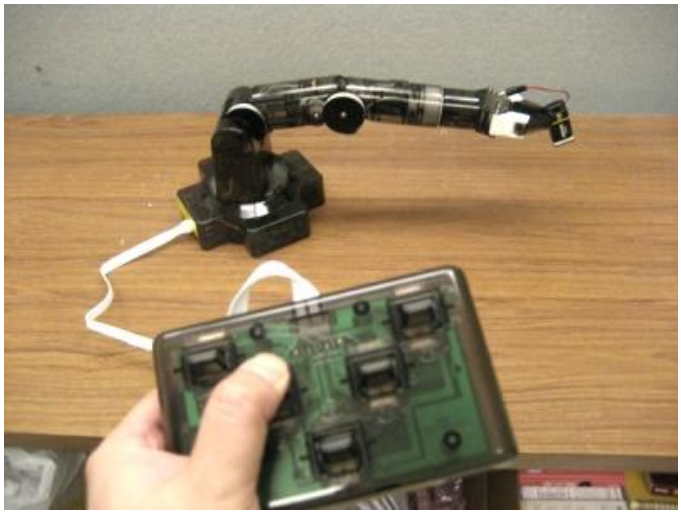

Figure 1: OWI007 Robotic Arm with remote control

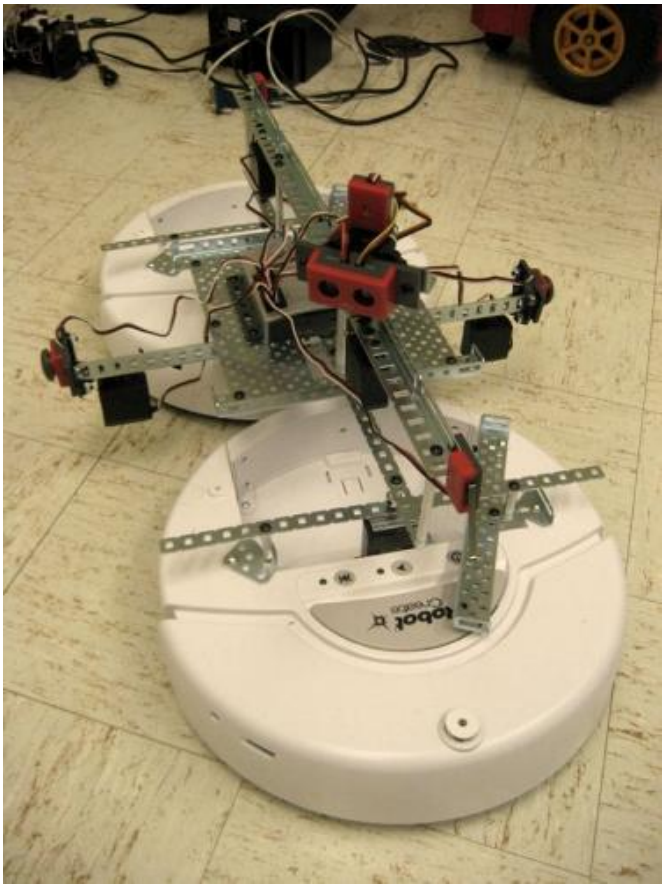

Figure 2: A mobile robot chained from 2 iRobot Creates and 1 VEX robot

2.1.2 Telerobotics

Originally, "Tele" is a Greek prefix meaning "distant". Telerobotics is the area of robotics concerned with the control of robots from a distance, chiefly using wireless connections. Telerobotics is a vague concept because it is not clear how far a distance has to be to be defined as "Tele" and whether appearance of real robots in sight of human operators is optional. Actually, in telerobotics, distance is not required to be long to called tele. This concept becomes relative because in some area like Tele-micro-surgery, this distance can be very small. As a matter of fact, telerobotics is a combination of two major subfields, teleoperation and telepresence; so to comprehend telerobotics, we should explain these two sub-concepts. ("Telerobotics," 2011)

Teleoperation is similar to remote control, in that, a teleoperator controls a teledevice remotely. "If such a device has the ability to perform autonomous work, it is called a telerobot. If the device is completely autonomous, it is called a robot." ("Telerobotics," 2011) These devices can be connected to their controllers tethered or wirelessly. The most significant problem with teleoperation is latency due to long distance communication. This leads to difficulties in real-time tracking of devices. The controller should have a visual tracking system to give visual response of devices when it is in motion. The delay in visual representation is not only because of communication distance but also because of encoding and decoding video signals. This problem will be analyzed in Chapters 4 and 5 of this thesis.

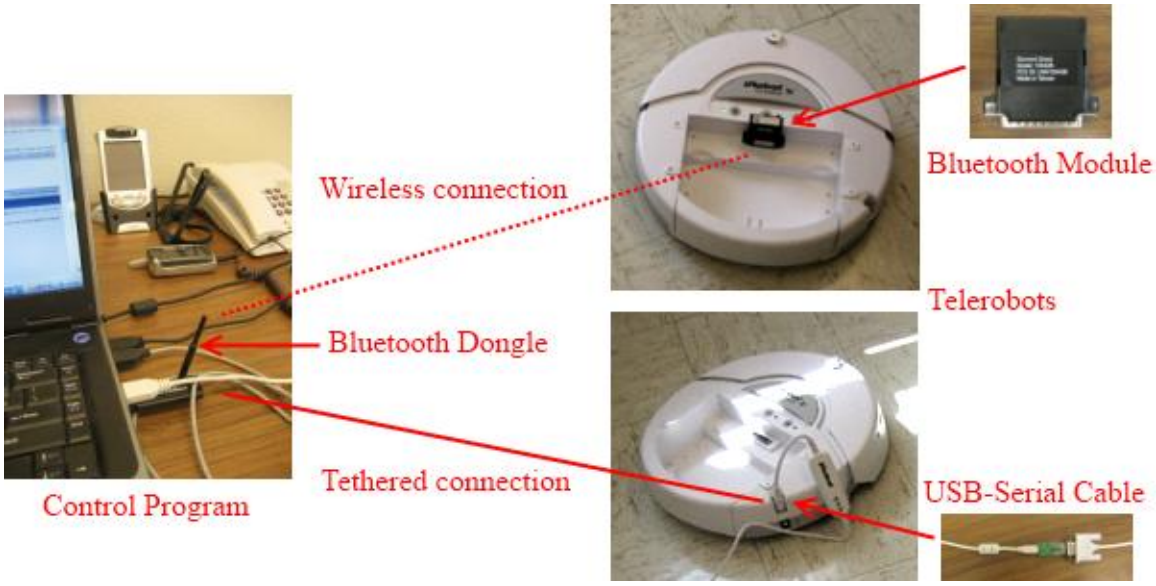

Figure 3: Typical teleoperation scenario in telerobotics system

Another major concept of telerobotics is tele-presence. Telepresence means "feeling like you are somewhere else" ("Telerobotics," 2011). "Telepresence refers to a set of technologies which allow a person to feel as if they were present, to give the appearance of being present, or to have an effect, via telerobotics, at a place other than their true location." ("Telepresence," 2011) Telepresence is more advanced than traditional video conferencing systems, webcam chatting,

office and home security system, and webcam daycare. Advanced video conferencing systems or robotic avatar systems are examples of applications of telepresence. They offer the capability of simulated presence of users at distant locations. The real users can feel as if they are actually at those places via sophisticated sensing mechanisms. In this thesis, we try to concentrate on a video tracking system to show some aspects of telepresence in a telerobotics system.

2.2 Applications of Telerobotics

2.2.1 Educational Applications

There has been much effort to create telerobotics systems for educational purposes over the last several years. In fact, there are many systems for remote education via telerobotics technology for the classroom or lab environment. Most of these recent systems are web-based to take advantage of the Internet as the main communication medium. Some of the very first ones were built as peer-to-peer systems; though these systems are more efficient than web-based ones, they are much less convenient. Web-based telerobotics systems similar to our system are the current trend because with the rapid evolution of high-speed Internet, this kind of system is much more convenient than others while not nearly as efficient. Educational telerobotics systems usually aim to serve some specific robotic experiments, platforms or interfaces which are limited when compared to our system.

2.2.2 Space Exploration Applications

Applications of telerobotics in space exploration are at a very early stage. In fact, all post-Apollo space programs use telerobotics and current satellite controls are mainly via telerobotics and similar technologies. Most popular telerobotics applications in space are related to outer-space robotic arms. These arms have redundant degrees of freedom which are required to conduct sophisticated motions and controls in space such as grasp, launch and recall of some kinds of satellites, space telescopes or other space modules. Without gravity, some kinematics rules

applied to these arms have to be computed in different ways. Practice with these arms on the ground requires complex supporting mechanisms to simulate a zero-gravity environment. In addition, due to long distances in space and other interference factors, communication with these arms can have a significant time delay leading to many control difficulties.

Figure 4: A NASA space robotic arm

2.2.3 Under-marine Exploration Applications

"Underwater robotics constitutes one of the most representative application fields of telerobotics. Over the last few decades, Remotely Operated Vehicles (ROVs) have played a very important role in undersea exploration/intervention, reducing the need of manned submersibles. In the future, Autonomous Underwater Vehicles (AUVs) equipped with acoustic modems and modern telerobotics technologies will do the job." (Ridao & Carreras, 2007) In reality, this is the most difficult area for telerobotics applications due to the limitations imposed by an under-marine environment. Communication, motion and detection are not like in air or outer-space because water does not allow communication means like electromagnetic signals or lasers to go through. The only possible medium is based on acoustic technologies which reveal many disadvantages.

Though these technologies already exist and have been around for a long time, this area still poses many challenges.

2.2.4 Surgery Applications

Telerobotics surgery or telesurgery is "surgery performed at a distance from the patient, which is enabled by advanced robotics, computer technology and, when distances from the surgeon to the patient are great, robust formats for data transmission." ("Telesurgery," 2011) Telesurgery will be popular in the future when surgical robots become more precise and affordable for most medical institutions; when this occurs, hospitals in rural areas will be able to afford sophisticated operations with telesurgery provided by remote high-skill professionals from other big medical centers. Telesurgery does not have to be from a long distance; sometimes it can apply to micro-surgery in which doctors use some special devices to do operations without really touching patients' internal organs.

2.2.5 Hazardous Environment Applications

Telerobotics technology is extensively used in hazardous environments such as nuclear plants, toxic chemical plants or areas, weaponry plants, or in extremely high or low temperature environments. These robots can operate some specific daily tasks such as picking or moving some toxic, radiated or hot materials with some direct control from human beings. Some robots work as rescuers when crises happen, like going to a place of fire to find and rescuing people or detecting survivors under collapsed structures after earthquakes or tsunamis.

2.2.6 Military Applications

These days, the military uses many kinds of robots and most of them are teleoperated. These robots can be very big and expensive such as the Unmanned Aerial Vehicles (UAVs) to do surveillance and launch missiles, or small ones to clear roadside bombs or a mine field.

Applications of telerobots and telerobotics in wartime save human lives and money for the army. These robots are gradually becoming more intelligent and in the future it is very possible that they may replace human soldiers to fight battles.

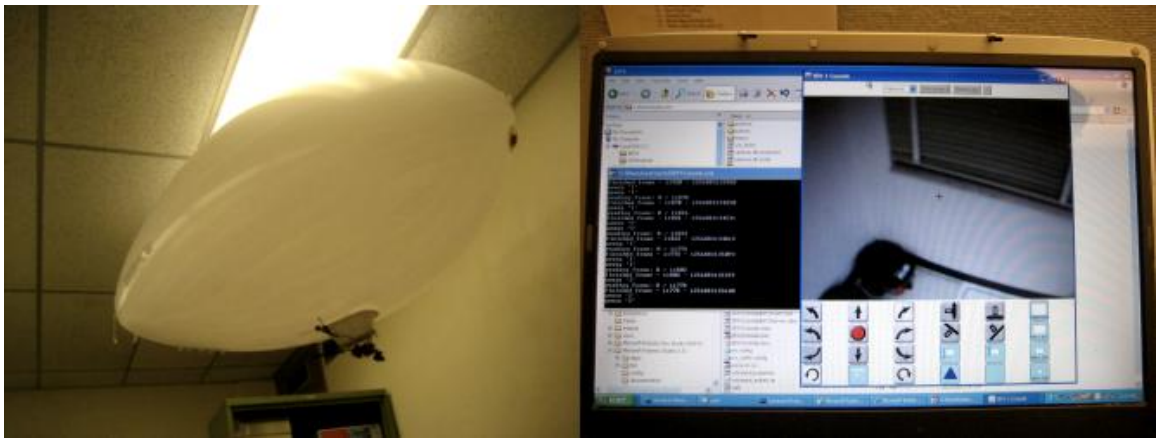

Figure 5: Zeppelin robot Blimp – A typical UAV with camera system

2.3 Some fundamental web concepts

Our system is RESTful (Representational State Transfer), and is based on the World Wide Web architectural style. Hence we review some basic web concepts to enforce the core ideas of REST.

2.3.1 HTTP

"Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web." ("Hypertext Transfer," 2011) Web and programmable web use HTTP in many different ways. Pre-REST web and programmable web do not usually take advantage of all HTTP methods or use them in the right way. Especially, RPC-style tends to ignore HTTP methods or use HTTP as a formal envelope to contain other labeled envelopes inside. (Richardson & Ruby, 2007) HTTP is a request-response protocol adapting to the client-server model, in which clients can be browsers for human-readable web and other kinds of applications for programmable web,

while servers are usually server-side web applications or web services. Hypertext seems to be obsolete as nowadays there are many kinds of media on the web, so hypermedia should be the right name for this protocol. However, because this is the base for the World Wide Web, any changes to it is difficult; only updates are acceptable and the current update specification of HTTP 1.1 is on July 11, 2011 (Lafon, 2011).

2.3.2 SOAP

"SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks." ("SOAP," 2011). It is an envelope-style protocol similar to HTTP but is based on XML; not only is it similar to HTTP but SOAP is dependent on HTTP because an XML-based SOAP envelope has to be put inside an HTTP unlabeled envelop to address. (Richardson & Ruby, 2007) This seems to be complicated but this is the architecture of RPC-style web services. HTTP envelops only the role of a container while SOAP envelop is a presentation of the inner resource. SOAP is very important to web services and is still very popular though there is a newer and better RESTful model. In the next chapter, we will show a comparison between traditional SOAP-based web services and RESTful web services.

2.3.3 URI

Uniform Resource Identifier (URI) or sometimes called Uniform Resource Locator (URL) defines the scope of the web. Though URL is more popular, URI seems to be more precise and natural to web essence. "A Uniform Resource Identifier (URI) is a string of characters used to identify a name of a resource on the Internet. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols. Schemes specifying a concrete syntax and associated protocols define each URI." ("Uniform Resource," 2011) In the pre-REST era, URIs were not friendly and optimal but REST

has changed the way the programmable web uses URIs. Planning, defining and optimizing URIs are important tasks in RESTful web services. One purpose of REST is to make these URIs friendly and uniform. The way web services use URIs is also different than the normal web. "A RESTful, resource-oriented service exposes a URI for every piece of data the client might want to operate on." (Richardson & Ruby, 2007) We will show step by step how to define resources and do those tasks with URIs in the next chapter.

2.3.4 XML-RPC

"It's a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet." ("What is," 2011) XML-RPC is based on XML for encoding messages which are then put in HTTP envelops for transportation. ("XML-RPC," 2011) This is the oldest web services styles and though it has many disadvantages, it still exists in many legacy web services. XML-RPC defines signatures for functions calls and follows the RPC style.

2.3.5 Programmable web

When users enter URLs on browsers, requests are sends to servers. Servers process these requests and appropriately return responses as web pages. The users surf these pages and find resources as they need. These responses are in a human-readable web form because they are rendered on the browsers' screens with logical layout and user friendly design which follow web standards. In contrast, the programmable web is different; a response from the programmable web is not for normal users to read. "The programmable web usually serves stark, brutal XML documents." (Richardson & Ruby, 2007) In fact, the programmable web can return other formats such as binary data or JSON though XML is more popular now. Except for return data format, other mechanisms of the programmable web are similar to the human-readable web.

The programmable web concept includes many kinds of web services and similar things, such as web crawlers, web bots, etc. Web services are the most typical applications in the world of programmable web right now while other similar ones are back behind the scenes. Web services can be classified based on their technologies, designs or architectures. Currently, technology is the most popular criterion to group web services because it is easy to identify it from the appearances of the web services. (Richardson & Ruby, 2007) Programmable web and web services are mainly for web programmers and web developers; they cannot read these services but they can consume them and use them in their applications.

2.4 Introduction to REST and Resource-Oriented Architecture

2.4.1 REST

REST stands for Representational State Transfer, which is a set of design criteria based on the architectural style of the World Wide Web (Richardson & Ruby, 2007, Flanders, 2009). The term REST was coined by Roy T. Fielding at the University of California Irvine in 2000 in his Doctoral Thesis, "Architectural Styles and the Design of Network-based Software Architecture", in which, REST is described as a series of architectural constraints that exemplify how the web's design emerged (Richardson & Ruby, 2007). REST mainly adapts to the client-server model of World Wide Web with core protocol HTTP. In this model, clients send requests, and servers process and respond to them accordingly. These messages in request-response communication are a transfer of representations of resources. Resources in this context are something useful, meaningful and relevant to the needs of the clients and available on the servers. Representations of resources at some phases reflect current states or future states of those resources. Changes in representations of resources are transferred from some states to other states which is the essential concept of REST. REST is not firmly attached to HTTP but can be applied to other protocols. However, in this thesis, we only focus on REST on HTTP and HTTPS.

Anything that conforms to REST constraints is referred to as being "RESTful" ("Representational," 2011). These architectural constraints are not standard rules but they are unofficial instructions so web services usually use them partly. As a matter of fact, it is rare for web services to strictly follow all constraints defined by REST. The reason for this can be explained by the comparison between REST and SOAP in the following table.

TABLE 1

COMPARISON BETWEEN SOAP AND REST

SOAP	REST
Protocol-independence	Best support for HTTP
A Specification	A set of constraints
Standardized by W3C	Unofficial use as "guidelines"
Not an architectural style	An architectural style
Based on service vocabulary	Based on resources
Use one URI for each server	Use many uniform URIs
Concern with verbs (actions)	Concern with nouns (resources)
Complicated interfaces	Simple and well-defined interfaces

2.4.2 Resource-Oriented Architecture

Resource-Oriented Architecture (ROA) was first defined by Leonard Richardson and Sam Ruby in their book RESTful Web Services in 2007. Resource-Oriented Architecture is built upon REST and they cannot be separated because currently there is no model better than REST for ROA. Actually, "the ROA is a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web, and the programmers will enjoy using." (Richardson & Ruby, 2007) REST can be used for other architectures because it is a set of

general design criteria; however, it is most appropriate for web services and ROA. REST itself is not architectural but ROA is. Though REST can be applied in general, possibly for other architectures or upon many other protocols, there is only one REST defined clearly by Roby Fielding in 2000. To understand ROA we have to mention its four key characteristics: addressability, statelessness, connectedness, and the uniform interface which will be explained below. (Richardson & Ruby, 2007)

2.4.2.1 Addressability. Web services provide resources to clients and if they can do it through URIs from parts of its data set in an interesting, appropriate and useful way, they are addressable. Web services can have an unlimited number of URIs which correspond to their resources. During the time of using web services, clients are concerned about addressability of web services the most. Therefore addressability is vital to web services and it should be considered first when designing web services. Addressability is the most flexible part of web services as it allows users to consume and use services in creative ways. Addressability is the nature of web services which are RESTful.

2.4.2.2 Statelessness. Statelessness is the next key feature of ROA. In reality, statelessness is not only an important feature of the ROA but it is the nature of the Web. Statelessness means a request-response process is absolutely completed in its own domain. The response is only for its closest previous request, it does not reply to any other requests before. Theoretically, there is no information left on servers after each request-response process. All information needed for the server to process and return a response is already included inside the request. There are neither connections between requests nor orders among them. Each request is an independent entity and all concepts of transactions or sessions are only simulations or "pseudo".

2.4.2.3 Connectedness. Connectedness is a sliding concept when it mentions about client-sides sessions and states but not "cookies". In this meaning, it builds contexts in which users create

sessions on their own and maintain their states by following the flow of their needs. The web with its hyperlinks and hypermedia arranged in logical ways will help users to do this harmoniously while still adapted to the statelessness rule. Hyperlinks become engines of application state and in the case of web services resources should link to each other in their representations. (Richardson & Ruby, 2007)

2.4.2.4 Uniform Interface. Most of non-RESTful web services do not have uniform interfaces though all use Uniform Resource Identifiers. Uniform interface means using HTTP methods in the same way and the right way. In each web service, one HTTP method should be used in the same way for any requests and on any resources. The following table shows the six most popular and important HTTP methods, their functions and properties.

TABLE 2

HTTP METHODS, THEIR FUNCTIONS AND PROPERTIES

Method	Function	Safe?	Idempotent?	Cacheable?
GET	Retrieve a resource	Yes	Yes	Yes
POST	Create a new resource	No	No	No
PUT	Update an existing resource	No	Yes	No
DELETE	Remove an existing resource	No	Yes	No
HEAD	Retrieve a metadata-only representation of a resource	Yes	Yes	Yes
OPTIONS	Check allowable options of a resource	Yes	Yes	No

(Richardson & Ruby, 2007, Flanders, 2009, Allamaraju, 2010)

2.5 Analysis on some Typical Telerobotics Systems

2.5.1 Non-web-based Telerobotics Systems

Non-web-based Telerobotics systems use two popular models - Peer-to-peer and Client-server. “Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads among peers. Peers are equally privileged, equipotent participants in the application. They are said to form a Peer-to-Peer network of nodes.” (“Peer-to-peer,” 2011) The “client–server model of computing is a distributed application that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.” (“Client-server,” 2011) Peer-to-peer and client-server are natural for distributed systems via UDP or TCP. In Peer-to-peer, there is no server or the server plays a very minor role and it is never a service provider. In contrast, client-server strongly concentrates on server systems because these are resource and service providers and are very loaded, when usually serving multiple client requests simultaneously. Non-web-based distributed telerobotics systems on Peer-to-peer or Client-server were very popular in the early days of the Internet due to the narrow bandwidth and limits of web technologies. (Graves, Ciscon, & Wise, 1994) designed and created a distributed telerobotics system in a project supported by NASA to connect robotics testbeds among four university campuses; this system can be deployed and run over many processors via a wide area network; it follows an object oriented and data driven architecture which actually adapts to the peer-to-peer model. (Bottazzi, Caselli, & Amoretti, 2002) developed a telerobotics framework based on Real-Time COBRA which took advantage of Asynchronous Method Invocation and exploited the Client-server model; this system presented the concepts of Robot server, Robot client and distributed sensor data. (Park & Park, 2003) propose a solution to Internet random time delays for non-web-based telerobotics systems built on a Client-server architecture using Event-based teleoperation. (Munasinghe, 2002) set up a telerobotics experimental environment for an industrial telemaipulator on a Peer-to-peer model.

Non-web-based telerobotics systems were good choices when Mega and Giga-bit Internet were not available though they have many disadvantages and inconveniences. These systems are still

useful if they are very specific to some narrow research areas or applications. Usually they required installation of client programs on workstations and a handshaking phase between peers or between clients and servers. The interface of these systems is often complicated and it takes a long time for scientists to masters their API.

2.5.2 User-web-interface-based Telerobotics Systems

Web-based systems include web-user-interface-based and programmable-web-based systems. Our system belongs to the latter category. Web-user-interface-based systems are based on the normal web or a human-readable web which is loaded using web browsers and is easily understood by everybody, whilst the programmable-web is to be consumed by machines and client programs and not directly by human eyes. This kind of web popularly appears as web services and other things like web bots, web spiders, etc. In this part we will discuss about User-web-interface telerobotics systems and compare them to our system and non-web-based ones.

User-web-interface-based teleorobotics systems are the current trend and becoming more popular. These systems have many advantages over non-web-based ones and exploit all benefits of modern web technologies and high-bandwidth Internet. Web-based telerobotics are especially useful on large-scale distributed systems which have to satisfy diversity of clients and homogeneous environment. Most scientists currently are forsaking the classic non-web-based design because it has many limitations. (Yu, Tsui, Zhou, & Hu, 2001) built a web-based telerobotics system on standard networking protocols and an interactive human-machine interface; users can remotely control and program mobile robots on their web browsers via a user web interface; besides, users are able to track their experiments with visual feedback and a perceptual map. (Doulgeri, 2006) created a web telerobotics system for teaching robotics based on real industrial robotic tasks; his system provided a high degree of real time interaction using an intuitive interface to guarantee operability and safety. (Terbuc, Uran, Rojko, & Jezernik, 2004)

introduced an effective methodology to teach students robotics and mechatronics by using 3D web-based virtual robotics lab for remotely control robots. (Safaric, Debevc, Parkin, & Uran, 1999) presented a way to train people working on expensive robotic facilities; this system has 2 phases, the first one uses off-line Virtual Environment planning and the latter one exports them to real remote robots to remotely execute via the Internet; this approach improved possibilities of education and training with low cost.

Our Telerobotics system is called iGEMS and is built on ROA. iGEMS is basically a web-based system but is not based on a user-web-interface. Our system is strongly focused on web services. Table 3 shows the comparison between other non-web-based, user-web-interface-based and our system iGEMS.

TABLE 3

COMPARISON BETWEEN NON-WEB-BASED, USER-WEB-INTERFACE-BASED AND IGEMS

Non-web-based	User-web-interface-based	iGEMS
Advantages		
▪ Appropriate for slow Internet	▪ Easily to use and develop ▪ Feasible to popularize	▪ Feasible to popularize ▪ Very flexible for development environments and platforms ▪ Natural to telerobotics and testbeds with resource-oriented ▪ Relieved from browser compatibilities ▪ Relieved from generation programming
Disadvantages		

▪ Required to install client programs ▪ Difficult to popularize ▪ Complicated in programming interface and API ▪ Inflexible for development environments and platforms	▪ Limited in control interface ▪ Inflexible for development environments and platforms ▪ Unreliable on browser compatibilities	▪ Required intermediate or professional level to consume and code

CHAPTER III

ARCHITECTURE AND DESIGN

3.1 Architecture

iGEMS is built on a Resource-Oriented Architecture. The reason we choose this architecture is because it matches the purposes of our telerobotics system:

- ROA is based on REST which can be adapted by both web-user-interfaces and web services.
- ROA is ideal for a system to provide resources as services.
- ROA is supported by many web technologies and patterns such as Microsoft WCF, Microsoft MVC, etc.
- ROA suggests natural and friendly service interface to clients.
- Systems built on ROA can be easily maintained and extended.

iGEMS is simple with 3 layers including 5 key modules; the highest layer consists of 2 interface modules, Web User Interfaces and Web Services; the middle layer is composed of 3 modules, Core, Controllers and Data Adaptor; the lowest layer is the Microsoft .NET Framework and related Microsoft libraries. The middle layer is the most important with the most sophisticated and critical modules of the system. These layers will be explained in detail next and the 5 key modules will be discussed in the next part of the system design.

Figure 6: iGEMS' Architecture

3.1.1 Public Interfaces

This highest layer has two modules, Web User Interfaces and Web Services. These two modules combined reveal the interfaces of the system to users, client programs, and controllers. The difference between these two modules is the former provides public interfaces for users in human-readable format while the latter is only consumed by client programs or machines. Except for this layer, all other lower layers are system internals. Some features on these layers can directly reference the lowest layer of the Architecture without going through the middle layer. However, most of the time, modules on this layer will heavily access and use modules on the middle layer.

3.1.2 Internal Libraries

The middle layer includes the typical and important modules of the system. Core is the most important one because it is a set of controller classes (this will be explained in next chapter) which define resources and create a resource management mechanism. Controllers are set of libraries and programs which support connections to robots such as iRobot Create, OWI007 Arm, etc. Data Adaptor is based on the Microsoft Entity Framework to access databases and create relational data models in memory. The modules on these layers will take advantage of features provided by the lowest layer of the Architecture.

3.1.3 Framework and other libraries

This layer contains Microsoft .NET Framework and other libraries such as Microsoft DSSP & CCR. This layer, in some aspects, can be considered to be external to our system because these framework and libraries are not built by us . We will return to this topic in the next part of system design. Microsoft Decentralized Software Services Protocol (DSSP) & Concurrency and Coordination Runtime (CCR) are libraries that belong to Microsoft Robotics Developer Studio and we use these in our controllers to control robots like iRobot Creates. DSSP & CCR are very good for providing loosely coupled robotic services for many kinds of robots. In this thesis, we use these services mostly to talk to iRobot Create to implement the mobile robots testbed.

3.2 Overall Design

Based on the 3-layer architecture, our system has 5 main modules: User-Web-Interfaces, Web Services, Core, Controllers, and Data Adaptor.

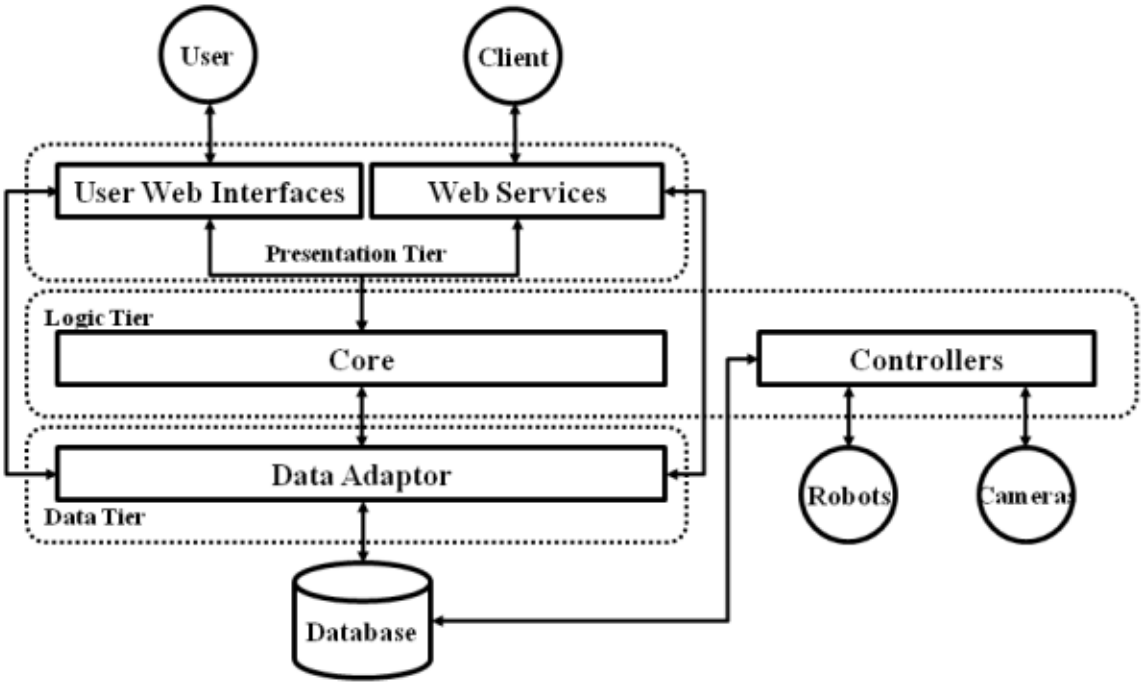

Figure 7: iGEMS' Overall Design

Our Overall Design includes 3 tiers, which means each tier can be deployed on different machines. The layers on the architecture are logical when tiers on the design are physical. The former one shows how code is organized and how close it is to end-users and clients. The latter presents deployment of code on real machines and how it is distributed. In addition, the database can be considered the fourth tier because it is possible to place the database on a separate server.

On the Overall Design diagram (Figure 7), the end-users are computer scientists, who usually use a User-Web-Interface to access the system via web browsers to conduct trivial actions such as logon, log-off, schedule experiments, check experiment results, start experiments at some specific times, etc., whist, clients are programs which are created by computer scientists and will consume web services to connect to the system's resources. Most of the time, one experiment sessions will involve both User-Web-Interfaces and Web Services because each provide different features and complement each other. End-users and client programs do not have to care about internal modules or .NET Framework, they only need to learn, get used to, and master User-Web-Interfaces and Web Services to use our system.

3.2.1 User Web Interfaces

The User Web Interfaces support the following functions for an end-user:

- Register for new user account: a user can register for a new account on our system by providing user-name, password, and appropriate CAPTCHA phrase.
- Log on the system: a registered user can log on our system to start using functions and connecting their programs to our external web services.
- Check and update account information: a registered and authenticated user can change their password.

- Start a new experiment: a registered and authenticated user can start a new experiment based on current available resources. The system currently does not support scheduling experiments in advance.
- Review and download previous experiments' results: a registered and authenticated user can review his previous experiments and download results.
- Log off the system: a registered and authenticated user can log off our system and stop all current connections and experiments.

3.2.2 Web Services

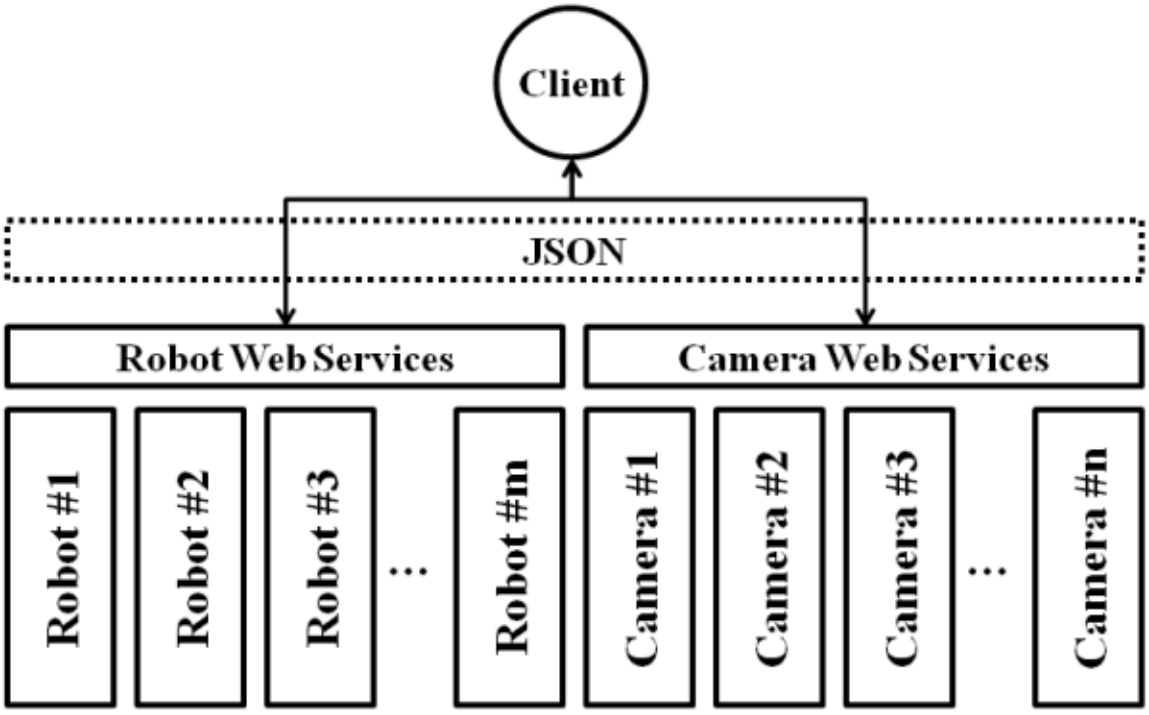

Figure 8: iGEMS' Web Services

There are 2 main web services in our system, robot web services and camera web services. All are RESTful web services and expose these resources as RESTful API. The robot web services allow client programs to connect to and control robots, while receiving feedback from robots' sensors. The feedback from robots' sensors is not enough for experimental tracking so camera

web services help with visual tracking channels from many perspectives. These web services are the main aspects of our system that are based on ROA and REST. Through these web services design and resource selections, the efficiency and convenience of REST constraints are manipulated and proved.

3.2.3 Core

The Core is a set of logics and filters to define most of the main features of our system:

- Define User Web Interface with filters for input and output data via Data Adaptor
- Create Web Service with filters for input and output data via input arguments and Data Adaptor
- Authorize users to use the User Web Interfaces and connect to External Web Services
- Specify special users representing attached external resources to the system
- Check input actions and redirect them if necessary

3.2.4 Robot Controllers

Currently, our system integrates controllers for iRobot Create and the OWI007 Robotic Arm. Controllers are developed and deployed on local workstations and connect to corresponding robots. These controllers are desktop programs to connect to robots via cables or wireless. These controllers have to be programmed specifically for each kind of robot because usually these kinds of robots support very different communication interfaces. Each robot will be controlled by one control program and these robots are differentiated from each other by their Ids.

Each controller is written as a desktop program with user interface. We choose to do this because this provides an emergency control pad to steer or stop the robot in urgent or emergency cases. Many things can happen to the robots in testbeds and there do not exist 100% autonomous robots. The Controller for iRobot Creates is created using Microsoft Robotics Developer Studio. This

controller allows steering this two wheeled robot forward, backward, to the left, to the right, and stopping the robot. The Controller for the OWI007 robotic arm uses an external library provided by the OWI007 manufacturer with a very simple interface to open/close the gripper, turn left/right the wrist, move up/down the elbow, move up/down the shoulder, and turn left/right the base.

3.2.5 Data Adaptor and Databases

The Data Adaptor is a module to map between persistent data in the database to corresponding objects in our system. The Data Adaptor is a middle layer between the database and the rest of the system. This module plays an important role in making the data abstract to the system. For any system, data is the most important thing and it should be carefully manipulated. The Data Adaptor currently only supports Microsoft SQL Server and Microsoft SQL Server Express. This module connects two databases; the first one contains data about user accounts and roles; this database is automatically created by Microsoft with a default database structure and table definitions; the second database is defined by us to store data about experiments and resources of the system.

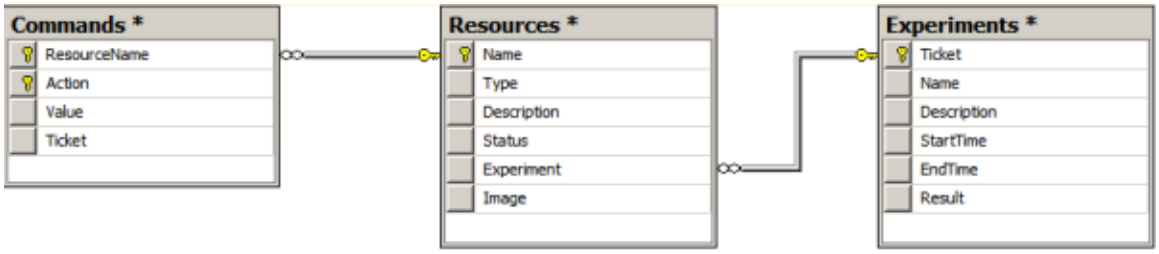

Figure 9: iGEMS' Database Structure

CHAPTER IV

IMPLEMENTATION

4.1 RESTful Web Services

We use web services in our system to communicate mostly with robots. Hence we follow some RESTful design instructions suggested by (Richardson & Ruby, 2007) to create a skeleton of real Resource-Oriented web services for iGEMS. We want to focus on RESTful web services for robots only so other features such as user account registration, and experiment management will be served by User-Web-Interfaces.

4.1.1 Data Set

We have 5 iRobot Creates attached to the system so the data set related to these robots are named as follows: iRobotCreate1, iRobotCreate2, iRobotCreate3, iRobotCreate4, and iRobotCreate5. For each iRobot Create, its data set consists of status, current speed, motion type, motion period, motion angle, and status of bumper; motion type includes moving forward, moving backward, turning left and turning right; motion period is an integer number greater than zero which represents the number of milliseconds the robot will move; motion angle is between 1 and 360.

iGEMS is attached to two OWI007 robotic arms so the data sets related to these arms are named as OWI007_1 and OWI007_2. Each robotic arm has five joints and its data set is consisted of status, joint, motion type, and motion period.

TABLE 4

IROBOT CREATE DATA SET

Data Item	Value
Status	N/A, available
Speed	Double value between 0.0 and 1.0
Motion Type	Move forward, move backward, turn left turn right
Motion Period	Integer value greater than zero
Motion Angle	Integer value between 0 and 360
Bumper	Touched and untouched

TABLE 5

OWI007 ROBOTIC ARM DATA SET

Data Item	Value
Status	N/A, available
Joint	Gripper, wrist, elbow, shoulder, base
Motion Period	Integer value greater than zero
Gripper	Open, close
Wrist	Turn clockwise, turn counterclockwise
Elbow	Life up, move down
Shoulder	Life up, move down
Base	Turn clockwise, turn counterclockwise

4.1.2 Split the Data Set into Resources

In this step we will decide which data set will be exposed as resources. Based on the dataset that we defined in the last step, we want all the data items in these data sets to be able to serve as resources for the system.

4.1.3 Name the Resources with URIs

We have the data sets and decision on which data set to expose as described above. We now proceed to name these resources logically to serve clients the best. To simplify the URIs, we encode them hierarchically as following:

TABLE 6

IROBOT CREATES RESOURCE URIS

Resource	URI
List of iRobot Create	http://server:port/iRobotCreate
Status of iRobot Create X	http://server:port/iRobotCreate/X
Get Current Speed	http://server:port/iRobotCreate/X/Speed/
Set Current Speed	http://server:port/iRobotCreate/X/Speed/Value
Move Forward	http://server:port/iRobotCreate/X/Forward
Move Forward in a period of time	http://server:port/iRobotCreate/X/Forward/Time
Move Backward	http://server:port/iRobotCreate/X/Backward
Move Backward in a period of time	http://server:port/iRobotCreate/X/Backward/Time
Turn Left	http://server:port/iRobotCreate/X/LeftTurn
Turn Left in a period of time	http://server:port/iRobotCreate/X/LeftTurn/Time
Turn Right	http://server:port/iRobotCreate/X/RightTurn
Turn Right in a period of time	http://server:port/iRobotCreate/X/RightTurn/Time
Bumper	http://server:port/iRobotCreate/X/Bumper

TABLE 7

OWI007 ROBOTIC ARM RESOURCE URIS

Resource	URI

List of OWI007 Robotic Arm	http://server:port/OWI007
Status of OWI007 X	http://server:port/OWI007/X
Gripper current Position	http://server:port/OWI007/X/Gripper
Gripper Open indefinitely	http://server:port/OWI007/X/GripperOpen
Gripper Open in period of time	http://server:port/OWI007/X/GripperOpen/Time
Gripper Close indefinitely	http://server:port/OWI007/X/GripperClose
Gripper Close in a period of time	http://server:port/OWI007/X/GripperClose/Time
Wrist current Position	http://server:port/OWI007/X/Wrist
Wrist Turn Clockwise indefinitely	http://server:port/OWI007/X/WristLeft
Wrist Turn Clockwise in a period of time	http://server:port/OWI007/X/WristLeft/Time
Wrist Turn Counterclockwise indefinitely	http://server:port/OWI007/X/WristRight
Wrist Turn Counterclockwise in a period of time	http://server:port/OWI007/X/WristRight/Time
Elbow current Position	http://server:port/OWI007/X/Elbow
Elbow Lift Up indefinitely	http://server:port/OWI007/X/ElbowUp
Elbow Lift Up in a period of time	http://server:port/OWI007/X/ElbowUp/Time
Elbow Move Down indefinitely	http://server:port/OWI007/X/ElbowDown
Elbow Move Down in a period of time	http://server:port/OWI007/X/ElbowDown/Time
Shoulder current Position	http://server:port/OWI007/X/Shoulder
Shoulder Lift Up indefinitely	http://server:port/OWI007/X/ShoulderUp
Shoulder Lift Up in a period of time	http://server:port/OWI007/X/ShoulderUp/Time
Shoulder Move Down indefinitely	http://server:port/OWI007/X/ShoulderDown
Shoulder Move Down in a period of time	http://server:port/OWI007/X/ShoulderDown/Time
Base current Position	http://server:port/OWI007/X/Base
Base Turn Clockwise indefinitely	http://server:port/OWI007/X/BaseLeft

Base Turn Clockwise in a period of time	http://server:port/OWI007/X/BaseLeft/Time
Base Turn Counterclockwise indefinitely	http://server:port/OWI007/X/BaseRight
Base Turn Counterclockwise in a period of time	http://server:port/OWI007/X/BaseRight/Time

4.1.4 Expose a Subset of the Uniform Interface

The decision in this step is simple because we have defined the same static resources in our system like robot names, robot motion types, etc. Clients can only change parameters such as periods of times, speed, motion angle. The URIs are already uniform and clients are not able to create any new URIs.

4.1.5 Design the Representation Accepted from the Client

As the URIs in 4.2.3, the presentations, in this case are URIs, to accept data from clients are simple. They adapt to the hierarchical rule and intuitively figure out how to use and follow these URIs. The convenience and efficiency of this design will be discussed in the next chapter.

4.1.6 Design the Representation Served to the Client

The representation accepted from the client is already stated on the URIs of the above two tables. We now discuss the representation served to the client from web services. This representation is the data format which is returned by the web services. There are many kinds of data formats but we choose to only return JSON because this format is more popular and is easily consumed by clients.

"JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.

JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language." ("JSON", 2011). In our system, we use JSON as the main format to return data from web services. For example, following is the list of iRobot Creates and their status return from the system in JSON format:

[{"ResourceName":" iRobot Create 1", "ResourceStatus": "N/A"},

{"ResourceName":" iRobot Create 2", "ResourceStatus": "Available"},

{"ResourceName":" iRobot Create 3", "ResourceStatus": "N/A"},

{"ResourceName":" iRobot Create 4", "ResourceStatus": "Available"},

{"ResourceName":" iRobot Create 5", "ResourceStatus": "Available"}]

JSON format is based on three structures:

- JSON overall data structure is an ordered list of objects. A list is bounded in a pair of square brackets, one for opening and one for closing.
- JSON object data structure is an unordered set of name/value pairs. An object is placed in a pair of curved brackets, one for opening and one for closing. Name/value pairs are separated from each other by commas.
- JSON name/value pair is structure of two parts which is separated by a colon. Name is a string and value can be Boolean, integer, float, or string.

In each programming language, the above structures become concrete data structures with different names and can be implemented in very different ways. Currently, most modern programming language support I/O in JSON format naturally, especially as JavaScript and JQuery dominate the world of web today.

We use Microsoft ASP.NET MVC 3 to create the web services. The reason we choose this technology and model because it is very natural to REST and it has very good routing module to

support generating many URI patterns. According to (Freeman & Anderson, 2011), URIs patterns in Microsoft ASP.NET MVC 3 have two key behaviors:

- "URL patterns are *conservative*, and will match only URLs that have the same number of segments as the pattern. This can be seen in the fourth and fifth examples in the table.
- URL patterns are *liberal*. If a URL does have the correct number of segments, the pattern will extract the value for the segment variable, whatever it might be."

The model for routing and general URIs in Microsoft ASP.NET MVC 3 is very flexible and logical. We can take advantage of this to generalize any URI pattern we want. In this project, we exploit this to create a set of URIs for our web services to expose our resources as we already defined in step 3.

4.1.7 Error Conditions

The following are some error conditions which can happen when clients request for our web services:

- The entered URIs are in wrong format, they can be too long, too short or have mistaken typo. The response is 400 (Bad Request)
- The entered URIs do not exist on the system. The response is 400 (Bad Request)
- The entered values are invalid, e.g. clients provide negative speed, negative time values, etc. The response is 400 (Bad Request)
- The resources which clients request are busy or not available at that time. The response is 503 (Service Unavailable)
- The server is down. The response is code 500 (Internal Server Error)

4.2 User Web Interfaces

In addition to Web Services, our system also supports User Web Interfaces. The main purpose of the User Web Interfaces is to support features like registering for a new account, logging on and logging off the system, starting a new experiment and reviewing previous ones. These functions are not the main features of our system and so we decided not to provide them as web services. We also use Microsoft ASP.NET to create Web User Web Interfaces because this provides many new features and models to create good User Web Interfaces.

Figure 10: iGEMS Register Page

Figure 11: iGEMS Log On Page

After authentication and authorization, a user has his Home Page with two functions to proceed:

- Start a New Experiment: a user can start a new experiment by providing its name and description and selecting resources from the available ones.
- Review Previous Experiments: a user is able to review his previous experiments and can decide to download their results if possible.

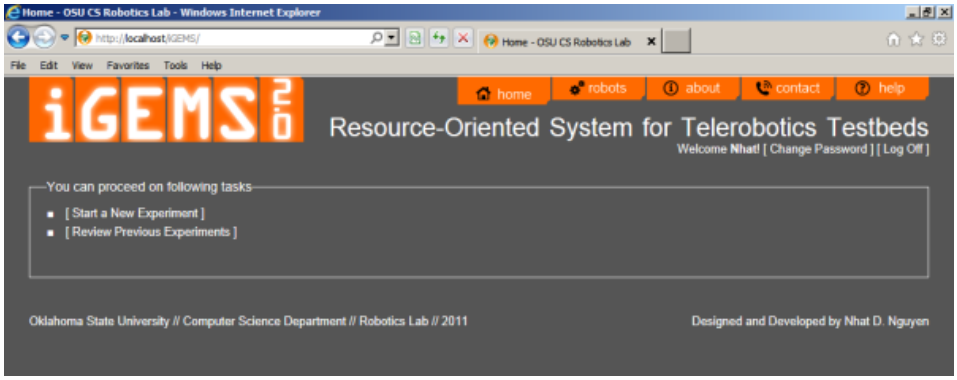

Figure 12: iGEMS Home Page

Figure 13: iGEMS Start a New Experiment Page

The Robot Page shows the current list of resources attached to our system and their status. A user can review this page to decide whether he is interested in these resources to start his new

experiment. This list is updated in almost real time to reflect the current resource repository of the system.

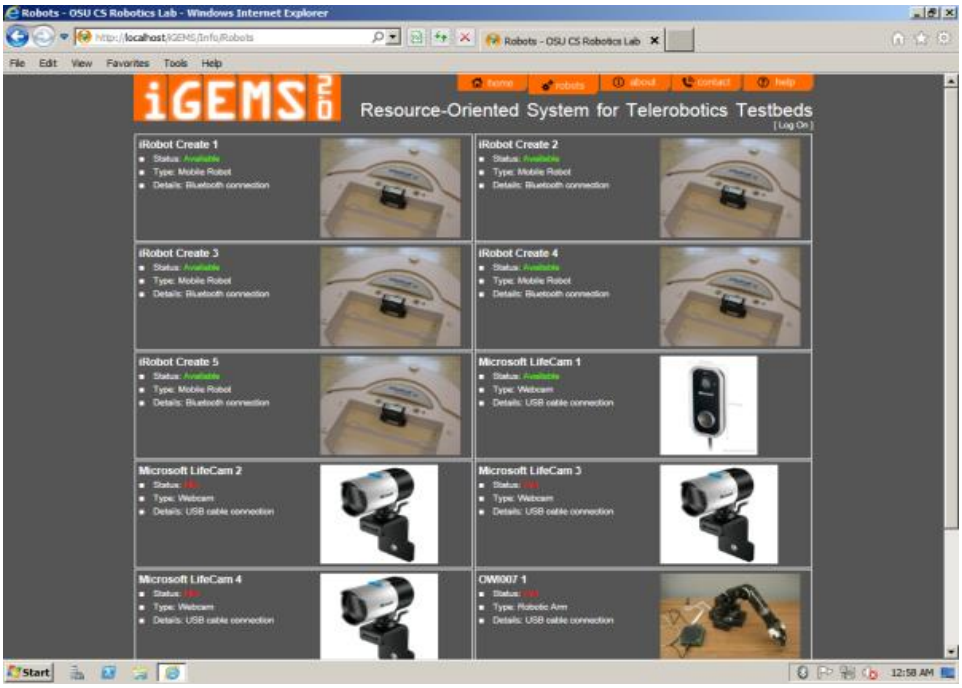

Figure 14: iGEMS Robots Page

4.3 Robot Controllers

We build two control programs; one for iRobot Create and one for OWI007 Robotic Arm. iRobot Create controller is created based on Microsoft Robotics Developer Studio and Microsoft Windows Form. This controller allows steering a iRobot Create, changing its speed and provides feedback about its bumper status. This program takes advantage of Microsoft robotic services to talk to iRobot Create via Bluetooth connection. We attach 5 iRobot Creates to our system. The server connection point is a Bluetooth dongle while each robot connection point is a Bluetooth Adaptor Module (BAM). Each robot communicates with the server on a different COM port and there is no conflict or interference in communication channels.

The control program of OWI007 Robotic Arm is developed on Microsoft Windows Form and OWI007 USB 076 Interface. This interface allows the controllers to set the robotic arm joints servos (robotic motors) in three states: left, right and teisi. Left state makes these motors shaft turn In addition to the excessive costs, technology is another disadvantage. In the world of robotics, left, right state makes them turn right and teisi means they are stopped.

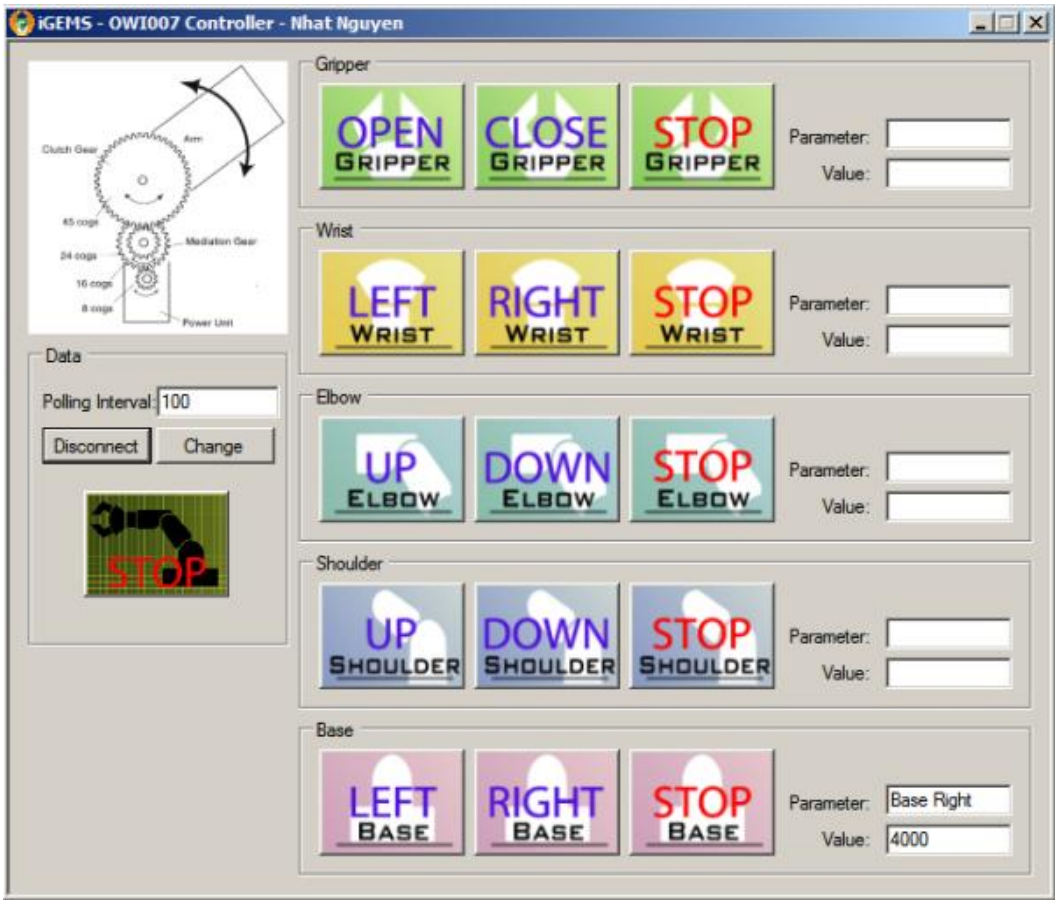

Figure 15: OWI007 Robotic Arm Control Program

These control programs are constantly polling the database to check and obtain input commands and arguments to proceed with actions. Commands are differentiated from each other by unique tickets generated by the system.

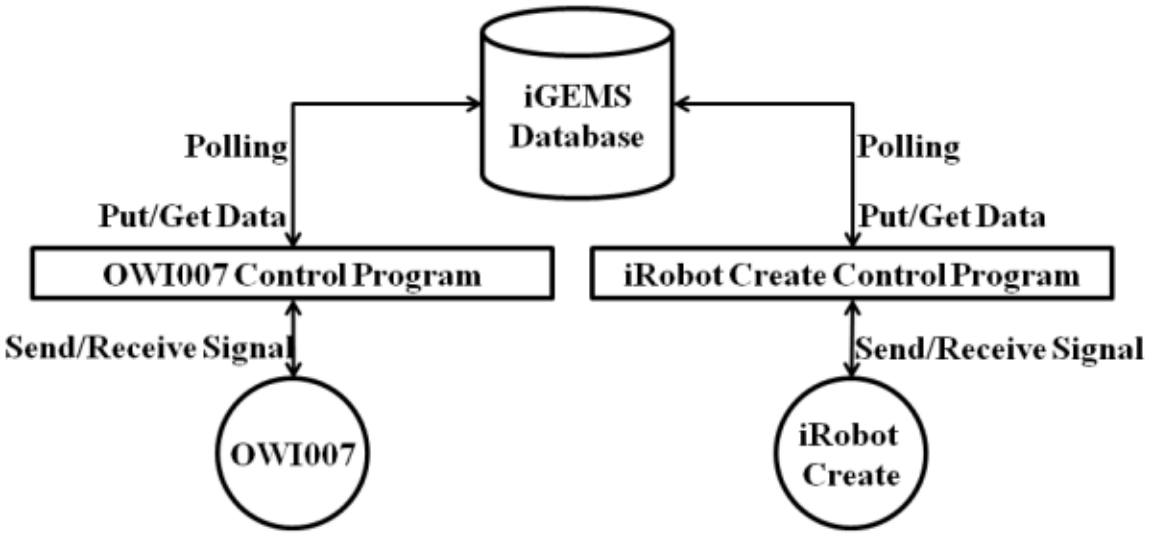

Figure 16: Database Polling Model for Robotic Control Programs

4.4 Core

The Core module of our system is implemented by Controllers in Microsoft ASP.NET MVC 3. These Controllers are different from the Controllers or Control Program in the previous section. These Controllers are the ones that will directly receive requests from and send responses to clients. Controllers together create a layer of logics which define I/O rules for the overall system. Controllers are enforced when equipped with filters. Filters are I/O rules which are applied to each Controller or each Action of a Controller. These filters help controllers validating inputs and selecting appropriate actions.

The big difference between the traditional web framework and Microsoft ASP.NET MVC is each URI in ASP.NET MVC corresponds to one Action in one Controller, not to a web page like the traditional case. An Action is similar to a handler function to process proper requests which are sent to it by clients and reply with an appropriate response.

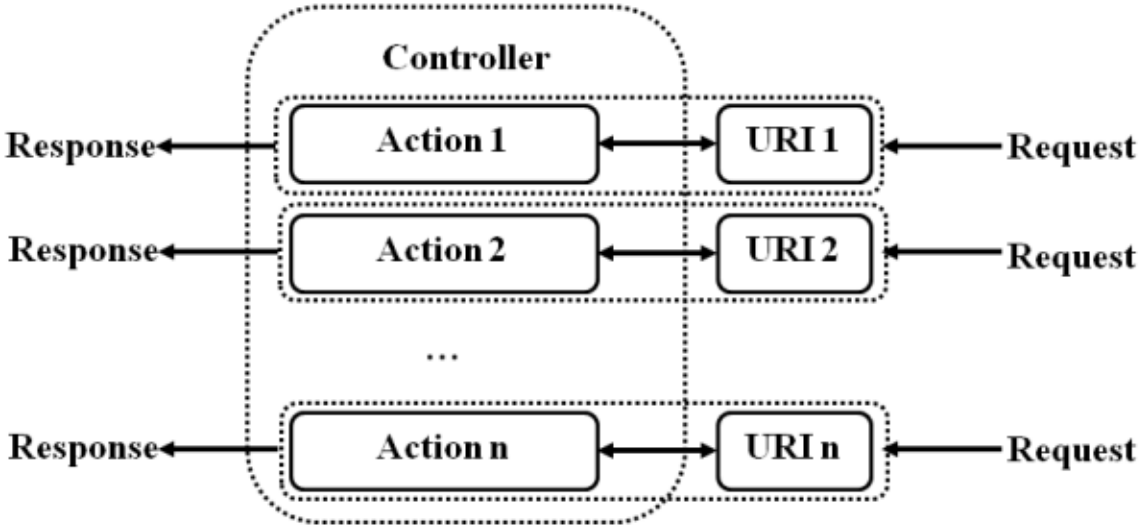

Figure 17: Controllers and Actions

4.5 Data Adaptor and Databases

We use Microsoft Entity Framework to map to the database that we already outlined in the last chapter. Entity Framework is simple when creating a data object model in memory to access anytime. This data model maps to tables (or relations) in iGEMS' database. We have 3 data objects:

- Experiment: this entity maps to the Experiments table in the database to access data about experiments such as name, description, start time, end time.
- Resource: this entity maps to Resources table in the database to access data about resources of the system such as name, type, description, status.
- Command: this entity maps to Commands table in the database to access data about commands sent to robots such as action, arguments' values.

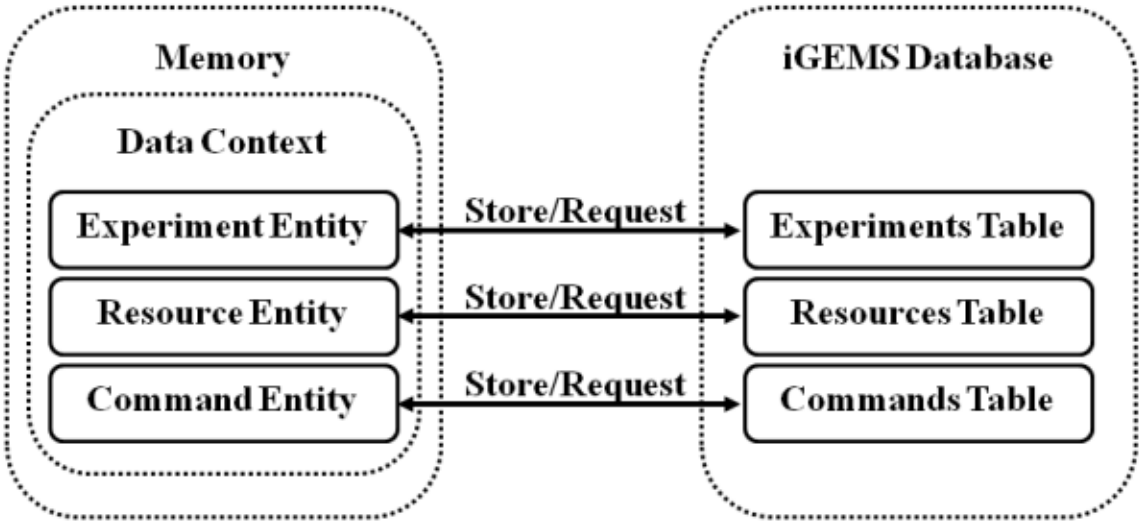

Figure 18: Mapping between Data Entity Objects and Database Tables

Besides, the database that we designed and created for iGEMS, we also take advantage of a built-in database by Microsoft for authentication and authorization of users. This database is generated automatically when we first create our project by Microsoft Visual Studio and access the account feature of a Microsoft ASP.NET MVC 3 application.

4.6 Video Streaming Services

We built video streaming services based on Windows Media Services and Microsoft Expression Encoder. Each web cam will be attached and encoded and streamed by Microsoft Expression Encoder to pre-defined publishing points. The web services will provide these publishing points to clients to connect to and obtain the streaming. The streaming process will suffer lags due to delays of encoding and decoding process and latency of the network.

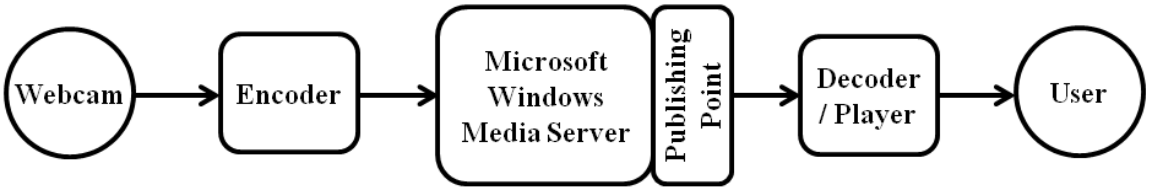

Figure 19: Video Streaming Process from a Live Source

Figure 20: Microsoft Expression Encoder with Live Source from a Webcam

Below are the typical parties participating in a video streaming process, which starts at a webcam live source and ends at the user who will receive video frames on his player. Our webcam Web Services will return the publishing points to clients so they can connect to and with their appropriate decoders, they can receive their video frames from our webcams.

CHAPTER V

FINDINGS

5.1 Current State of Implementation

We have implemented iGEMS using the development tools listed in the previous chapter. Currently, iGEMS includes all 5 key modules, namely, User Web Interfaces, Web Services, Core, Robot Controllers and Data Adaptor. We also tested this system on real robots including iRobot Create and OWI007 Robotic Arm and the overall system performs as expected; the robots receive commands almost instantly and the video feeding delay time is acceptable (20 ms – 40 ms). The last thing that we have to do is to integrate iGEMS into OKGems and the Geni project with a common resource management and security mechanism.

5.2 Potential factors of RESTful Web Services for Large-Scale Experiments

This system is aimed at lending the robotics resources to users as Web Services. These Web Services we built are RESTful and the overall system is Resource-Oriented. Robotic resources are provided via Web Services in a concrete manner as URIs. These URIs follow defined patterns which have two parts:

- Required part: this part is the static resources of the system such as a name of a robot, a sensor of a robot, a joint of a robotic arm, a name of camera, etc.
- Optional part: this part is parameters to be assigned to some resources defined in the required part of the URI, like speed of a robot, period of time in motion of a joint of a robotic arm, etc.

This is the general format of an URI: http://server:port/(required part)/[optional part]. We show some examples:

- http://server:port/(OWI007/1/ElbowUp)/[Time]
- http://server:port/(OWI007/2/BaseLeft)/[Time]

The required part is not absolutely strict because it describes a general set of resources where each resource can be referenced by its Id. This system of Ids are very powerful when applied to a huge set of resources and even when there are new resources attached to the system, the general interfaces for these Web Services do not have to be changed.

The above characteristics of RESTful Web Services are very good for large-scale experiments because we do not have to re-write Controllers and Action or re-define the URIs when implementing to testbeds with a few resources or with one with a large number of resources. If users want to attach new resources, the system only sets new Ids for them; if users desire to provide more parameters for resources, they can define them in the optional part of URIs.

5.3 Friendliness and Intuition of RESTful URIs

The patterns of URIs are uniform because its resources belong to one category; they will reveal the same URIs with the only difference being in Ids. If users know how to connect and consume one resource then they can easily figure out to do the same with other resources in the same category. Even if the resource is in a different category, the pattern of its URIs is very similar.

Besides, the hierarchical patterns of URIs are intuitive for users to understand the meanings of each level and their roles in the current referenced resource. For example:

http://server:port/OWI007/1: this URIs refers to the OWI007 Robotic Arm with Id = 1; the user can learn this pattern and refer to other OWI007 Robotic Arms in the system by only changing the Id on URIs.

http://server:port/OWI007/2/Gripper: this refers to a joint called Gripper of an OWI007 Robotic Arm and this level will refer to each joint of an arm.

The RESTful Web Services URIs are a combination of friendly and intuitive factors to serve best patterns to clients so users can save time when from learning the interfaces, compared to non-RESTful Web Services which have a very complicated vocabulary of remote functions.

CHAPTER V

CONCLUSION

iGEMS is the first of a new kind of Resource-Oriented systems for Telerobotics Testbeds which exploit REST to provide a convenient, friendly and efficient environment to do robotics research. When some robotic resources are out of reach for some small educational institutions, our system provides a framework to create bridges and collaboration between robotics scientists in the computer science area. The reason to choose Web Services as the means to expose remote robotic resources is because our system aims at a large amount of users have very different backgrounds and use different tools to work with robots. The two significant problems of finance and technology in the world of robotics are solved in our system. Though iGEMS is the first of its kind, it can be extended and improved in the future to overcome its current limitations; in fact, the autonomy of the system is not perfect, it still requires human interference to help robots when they get stuck and humans need to check their mechanical parts' conditions; it would be much better if we can significantly reduce the latency for video streaming to simulate real-time experiments

REFERENCES

1. *Who did actually invent the word "robot" and what does it mean?* (1933, Dec 24) Retrieved August 25, 2011 from Dominik Zunt Web site: http://capek.misto.cz/english/robot.html, (Date of access: August 25, 2011)

2. Robotics. (n.d.). In *Oxford online dictionary*. Retrieved August 22, 2011, from http://oxforddictionaries.com, (Date of access: August 22, 2011)

3. Robotics. (n.d.). In *Webster online dictionary and thesaurus*. Retrieved August 22, 2011 from http://merriam-webster.com, (Data of access: August 22, 2011)

4. Three Laws of Robotics. (n.d.). In. Retrieved August 25, 2011, from The Free Online Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Three_Laws_of_Robotics, (Date of access: August 25, 2011)

5. Mobile Robot. (n.d.). Retrieved August 25, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Mobile_robot, (Date of access: August 25, 2011)

6. Telerobotics. (n.d.). Retrieved August 25, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Telerobotics (Date of access: August 25, 2011)

7. Telepresence. (n.d.). Retrieved August 25, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Telepresence, (Date of access: August 25, 2011)

8. Ridao P., Carreras M., Hernandez E., & Palomeras H. (2007). Underwater Telerobotics for Collaborative Research. *Springer Tracts in Advanced Robotics*, 31, 347-359. Retrieved from http://www.springerlink.com/content/0123282405663604/, (Date of access: September 27, 2011)

9. Telesurgery. In *the online Free Dictionary*. Retrieved August 25, 2011, from http://medical-dictionary.thefreedictionary.com/Telerobotic+Surgery, (Date of access: September 27, 2011)

10. Richardson L., & Ruby S. (2007). *RESTful Web Services*. Sebastopol, CA: O'reilly.

11. Allamaraju S. (2010). *RESTful Web Services Cookbook*. Sebastopol, CA: O'reilly.

12. Flanders J. (2009). RESTful .NET: Build and Consume RESTful Web Services with .NET 3.5. Sebastopol, CA: O'reilly.

13. Representational State Transfer. (n.d). Retrieved August 26, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Representational_state_transfer, (Date of access: August 26, 2011)

14. Lafon, Y. (n.d.). *HTTP – Hypertext Transfer Protocol*. Retrieved August 28, 2011, from http://www.w3.org/Protocols/, (Date of access: August 28, 2011)

15. Hypertext Transfer Protocol. (n.d.). Retrieved August 28, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, (Date of access: August 28, 2011)

16. SOAP. (n.d.). Retrieved August 28, 2011, from the Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/SOAP, (Date of access: August 28, 2011)

17. Uniform Resource Identifier. (n.d.). Retrieved August 29, 2011, from the Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Uniform_Resource_Identifier, (Date of access: August 29, 2011)

18. Unknown. (n.d.). *What is XML-RPC?* Retrieved August 29, 2011, from http://ww.xmlrpc.com, (Date of access: August 29, 2011)

19. XML-RPC. (n.d.). Retrieved August 29, 2011, from the Free Encyclopedia Wikipedia:http://en.wikipedia.org/wiki/XML-RPC, (Date of access: August 29, 2011)

20. Peer-to-peer. (n.d.). Retrieved September 7, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/Peer-to-peer, (Date of access: August 29, 2011)

21. Client-server. (n.d.). Retrieved September 7, 2011, from The Free Encyclopedia Wikipedia: http://en.wikipedia.org/wiki/ Client%E2%80%93server_model, (Date of access: September 7, 2011)

22. Graves S., Ciscon L., & Wise J.D. (1992, May 12-14). *A Modular Software System for Distributed Telerobotics*. Proceedings IEEE International Conference on Robotics and Automation, Nice, France. doi: 10.1109/ROBOT.1992.220013

23. Bottazzi S., Caselli S., Reggiani, & Amoretti M. (2002). *A Software Framework based on Real-Time COBRA for Telerobotics Systems*. Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems. doi: 10.1109/IRDS.2002.1041730

24. Park J.H., & Park J. (2003, Oct 27-31). *Real Time Bilateral Control for Internet based Telerobotic System*. Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems. doi: 10.1109/IROS.2003.1248792

25. Munasinghe S. R. (2002, Nov). *Design and Implementation of an Experimental Test Bed for Telerobotics Research Between Saga University and Kurume National College of Technology*. Retrieved from www.ent.mrt.ac.lk/~rohan/career/projects/telegolf/telerob.pdf, (Date of access: October 10, 2011)

26. Yu, L., Tsui P. W., Zhou Q., & Hu H. (2001). *A Web-based Telerobotics System for Research and Education at Essex*. Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics. doi: 10.1109/AIM.2001.936427

27. Doulgeri, Z. (2006, May). *A Web Telerobotic System to Teach Industrial Robot Path Planning and Control*. Journal of IEEE Transactions on Education, 49(2), 263-270. doi: 10.1109/TE.2006.873975

28. Terbuc M., Uran S., Rojko A., & Jezernik K. (2004). *Web-based Education of Robotics and Mechatronics at the University of Maribor*. Retrieved from www.ro.feri.uni-mb.si/~martin/predst/caita.pdf, (Date of access: October 22, 2011)

29. Safaric R., Debevc M., Parkin R. M., & Uran S. (1999) *Telerobotics Experiments via Internet. Journal of IEEE Industrial Electronics*, 48(2), 424-431. doi: 10.1109/ISIE.1999.801802

30. Masse M. (2011). *REST API Design Rulebook.* Sebastopol, CA: O'reilly.

31. Fowler M., & Robinson I. (2011). *Service Design Patterns Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Sebastopol, CA: O'reilly.

32. JSON. (n.d.). *Introducing JSON*. Retrieved Oct 12, 2011, from http://json.org, (Date of access: November 3, 2011)

33. Freeman A., & Sanderson S. (2011). *Pro ASP.NET MVC 3 Framework*. New York City: Apress.

VITA

Nhat Dong Nguyen

Candidate for the Degree of

Master of Science

Thesis: IGEMS - RESOURCE-ORIENTED SYSTEM FOR TELEROBOTICS TESTBEDS

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December, 2011.

Completed the requirements for the Bachelor of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in 2009.

Name: Nhat Dong Nguyen Date of Degree: December, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: IGEMS - RESOURCE-ORIENTED SYSTEM FOR TELEROBOTICS TESTBEDS

Pages in Study: 51 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: system design with empirical methodology.

Findings and Conclusions:
Although robotics is currently a very popular and interesting research area, there still exist two main problems of finance and technology that create barriers for roboticists and computer scientists to have access to sufficient robotic resources to conduct their experiments. In this thesis, we design and create a Resource-Oriented System for Telerobotics Testbeds which allow scientists to connect and use remote robotic resources from other institutions where they may be available. This will help solve the financial problem for small education institutions with a small budget. Besides, the system provides the attached resources as Web Services which can be connected and consumed by diverse technologies and platforms. This overcomes the second big problem of the technology barrier. Our system is implemented on a real server. Our Web Services followed by URIs patterns are very friendly and intuitive to users. The design of URIs enables our framework to be applied for large-scale experiments with a huge amount of attached resources.

ADVISER'S APPROVAL: DR. JOHNSON THOMAS