AN ABSTRACT OF THE THESIS OF

Ji Seok Liew for the degree of Master of Science in Electrical and Computer Engineering presented on March 17, 2003.

Title: Web-Based Distributed Applications for Cytosensor

Abstract approved: Redacted for privacy

Wojtek J. Kolodziej

To protect the environment and save human lives, the detection of various hazardous toxins of biological or chemical origin has been a major challenge to the researchers at Oregon State University. Living fish cells can indicate the presence of a wide range of toxins by reactions such as changing color and shape changes. A research team in Electrical and Computer Engineering Department is developing a hybrid detection device (Cytosensor) that combines biological reaction and digital technology. The functions of Cytosensor can be divided into three parts, which are real-time image acquisition, data processing and statistical data analysis.

User-friendly Web-Based Distributed Applications (WBDA) for Cytosensor offer various utilities. WBDA allow the users to control and observe the local Cytosensor, search and retrieve data acquired by the sensor network, and process the acquired images remotely using only a web browser. Additionally, these applications minimize the user’s exposure to dangerous chemicals or biological products.

This thesis describes the design of a remote controller, system observer, remote processor, and search engine using JAVA applets, XML, Perl, MATLAB, and Peer-to-Peer models. Furthermore, the implementations of image segmentation technique in MATLAB and the Machine Vision Algorithm in JAVA for independent web-based processing are investigated.
Web-Based Distributed Applications for Cytosensor

by

Ji Seok Liew

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 17, 2003
Commencement June 2003

APPROVED:

Redacted for privacy 3-31-03

Major Professor, representing Electrical and Computer Engineering

Redacted for privacy 4-7-03

Head of the Department of Electrical and Computer Engineering

Redacted for privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Ji Seok Liew, Author
ACKNOWLEDGEMENTS

I appreciate my advisor, Dr. Wojtek J. Kolodziej, for his patience and valuable guidance throughout my graduate study. His criticism and advice made me grow not only as an engineer, but also as a person. I would also like to thank the members of my thesis committee for their helpful feedback.

Thanks to Nicolas Roussel for his insights and helpful comments on the image segmentation of living cells as well as a wide range of subjects in the image processing field. Thanks to Bertrand Boichon and Voranon Kiettrisalpipop for their insights and helpful comments on the Peer-to-Peer model, XML-RPC based command scenario and Cytosensor application. Thanks to Joseph Sam Lesmana for helping me with JAVA programming as a master. Thanks to the entire Cytosensor team and the Catalyst Foundation.

I’d like to thank my family for their endless support. I would like to thank my parents for patiently listening to my thoughts on my future and advising me. I would like to thank my wife for conversing with me about many different issues of my life. Whenever I look at Naomi, my first baby in the world, her cute face stimulates me to work hard.

Finally, I’d like to thank my God for always being there and listening to my countless prayers in my good times and bad times. I could not have completed my graduate study without Him.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. COMMUNICATION - OBSERVER AND REMOTE CONTROLLER</td>
<td>3</td>
</tr>
<tr>
<td>2.1. Overview of the remote controller and observer</td>
<td>3</td>
</tr>
<tr>
<td>2.2. Needs and specifications</td>
<td>6</td>
</tr>
<tr>
<td>2.3. Software design – The kernel technique for the server side</td>
<td>7</td>
</tr>
<tr>
<td>2.3.1. Command handling</td>
<td>8</td>
</tr>
<tr>
<td>2.3.2. Distributing information using XML</td>
<td>12</td>
</tr>
<tr>
<td>2.3.3. P2P (Peer to Peer) - Data sharing and search</td>
<td>14</td>
</tr>
<tr>
<td>2.4. Software design - The kernel technique for the client side</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1. General Function of the JAVA applet</td>
<td>17</td>
</tr>
<tr>
<td>2.4.2. Reading information using XML (Alfred parsing) in applet</td>
<td>20</td>
</tr>
<tr>
<td>2.4.3. Future procedures and possible improvement</td>
<td>22</td>
</tr>
<tr>
<td>3. REMOTE PROCESSING</td>
<td>25</td>
</tr>
<tr>
<td>3.1. Overview of the remote processing</td>
<td>25</td>
</tr>
<tr>
<td>3.2. Needs and Specifications</td>
<td>26</td>
</tr>
<tr>
<td>3.3. Processing unit – MATLAB Web Server Tool Box</td>
<td>28</td>
</tr>
<tr>
<td>3.3.1. Image processing with MATLAB</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2. MATLAB Web Server Tool Box</td>
<td>30</td>
</tr>
<tr>
<td>3.4. Remote processing with HTML page containing applet</td>
<td>31</td>
</tr>
<tr>
<td>3.4.1. MATLAB and XML- Upload &amp; Processing Indicator</td>
<td>31</td>
</tr>
<tr>
<td>3.4.2. Display the processing results</td>
<td>32</td>
</tr>
<tr>
<td>3.4.3. P2P (Peer to Peer) – Data sharing and search</td>
<td>34</td>
</tr>
<tr>
<td>TABLE OF CONTENTS (CONTINUED)</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4. IMAGE PROCESSING AND MACHINE VISION IN JAVA</td>
<td>35</td>
</tr>
<tr>
<td>4.1. Remote image processing in JAVA applet</td>
<td>35</td>
</tr>
<tr>
<td>4.1.1. JAVA Image processing</td>
<td>35</td>
</tr>
<tr>
<td>4.1.2. Image processing - Implemented by Java Applet</td>
<td>36</td>
</tr>
<tr>
<td>4.2. Machine Vision in Java</td>
<td>39</td>
</tr>
<tr>
<td>4.2.1. Machine Vision technique for image segmentation</td>
<td>39</td>
</tr>
<tr>
<td>4.2.2. Machine Vision technique for object detection</td>
<td>40</td>
</tr>
<tr>
<td>5. UNDERSTANDING IMAGE ANALYSES IN MATLAB</td>
<td>42</td>
</tr>
<tr>
<td>5.1. Image Analyses in MATLAB - Geometrical analyses</td>
<td>42</td>
</tr>
<tr>
<td>5.2. Image Analyses in MATLAB - Color analysis</td>
<td>44</td>
</tr>
<tr>
<td>6. CONCLUSION</td>
<td>46</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>47</td>
</tr>
<tr>
<td>APPENDIX (SCREEN SHOTS)</td>
<td>48</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1: Cytosensor and Diagram Overview</td>
<td>3</td>
</tr>
<tr>
<td>2.2: Overview of Web-Based Distributed Applications for Cytosensor</td>
<td>4</td>
</tr>
<tr>
<td>2.3: Peer-to-Peer scenario to search for and retrieve data</td>
<td>5</td>
</tr>
<tr>
<td>2.4: Command line using Socket programming</td>
<td>8</td>
</tr>
<tr>
<td>2.5: XML-RPC</td>
<td>9</td>
</tr>
<tr>
<td>2.6: Multi-threading example</td>
<td>10</td>
</tr>
<tr>
<td>2.7: Thread control Flow</td>
<td>11</td>
</tr>
<tr>
<td>2.8: XML files containing information and image properties</td>
<td>12</td>
</tr>
<tr>
<td>2.9: Scenario of P2P Model</td>
<td>15</td>
</tr>
<tr>
<td>2.10: General overview of JAVA applet</td>
<td>18</td>
</tr>
<tr>
<td>2.11: Screen Capture of Controller application</td>
<td>19</td>
</tr>
<tr>
<td>2.12: Menu bar and properties of images in Observer</td>
<td>20</td>
</tr>
<tr>
<td>2.13: XML document and displaying panel</td>
<td>21</td>
</tr>
<tr>
<td>2.14: Perl Search Engine on the Web Page</td>
<td>22</td>
</tr>
<tr>
<td>2.15: General overview of future JAVA applet</td>
<td>23</td>
</tr>
<tr>
<td>2.16: Sample code of XML-RPC</td>
<td>24</td>
</tr>
<tr>
<td>3.1: General overview of remote processing</td>
<td>25</td>
</tr>
<tr>
<td>3.2: The steps of remote processing using web pages</td>
<td>27</td>
</tr>
<tr>
<td>3.3: Overview of image processing of Cytosensor project</td>
<td>29</td>
</tr>
<tr>
<td>3.4: Procedure of remote processing</td>
<td>30</td>
</tr>
<tr>
<td>3.5: Uploading, starting to process, and processing Indicator</td>
<td>32</td>
</tr>
<tr>
<td>3.6: Screen capture of CytoMat</td>
<td>33</td>
</tr>
<tr>
<td>3.7: Scenario of P2P based search model</td>
<td>34</td>
</tr>
<tr>
<td>4.1: Overview of the JAVA structure for the Image Processing</td>
<td>35</td>
</tr>
<tr>
<td>4.2: Integer arrays in the RGB color model</td>
<td>36</td>
</tr>
<tr>
<td>4.3: To read pixel information in JAVA</td>
<td>37</td>
</tr>
<tr>
<td>4.4: Sobel Edge Detector using a 3x3 mask</td>
<td>38</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4.5</td>
<td>Real time processing using stand-alone JAVA applet</td>
</tr>
<tr>
<td>4.6</td>
<td>Watershed Segmentation</td>
</tr>
<tr>
<td>4.7</td>
<td>Dilatation and Erosion</td>
</tr>
<tr>
<td>4.8</td>
<td>Overall view of cell detection</td>
</tr>
<tr>
<td>5.1</td>
<td>Geometrical Features using JAVA applet and XML</td>
</tr>
<tr>
<td>5.2</td>
<td>MajorAxisLength, MinorAxisLength, and Orientation</td>
</tr>
<tr>
<td>5.3</td>
<td>Statistical color analysis using JAVA applet and XML</td>
</tr>
</tbody>
</table>
WEB-BASED DISTRIBUTED APPLICATIONS FOR CYTOSENSOR

1. INTRODUCTION

User Friendly Web-Based Distributed Applications (UFWBDA or WBDA) offer solutions for the problems regarding remote communication, remote control, remote processing, and complicated computer network usage through the standard interface. In particular, this application is used by the Cytosensor project. Various features of WBDA are illustrated using the Cytosensor as an example. The Cytosensor is an automated, living cell based sensor that detects a variety of biological and chemical toxins. The Cytosensor operation is based on the observation of a living cell, which changes its shape and color when exposed to various toxins. The faculty and students from several departments in the colleges of science and engineering are involved in this interdisciplinary project. The role of the Electrical and Computer Engineering Department is to provide the data acquisition and computational platform and to develop algorithms for automated data analysis. Since the cell behavior in the presence of toxins is best described visually, the image processing techniques are the most suitable. The software and hardware are integrated for image acquisition and processing. The software functionality can be divided into three parts: real-time image acquisition, image processing and statistical data analysis. A high-definition camera captures the images.

WBDA offers a tremendous help for users of various professional backgrounds to use the Cytosensor. First, when they conduct experiments with multiple of acquired images and processed results, WBDA allows them to control and observe the local Cytosensor to search and retrieve data acquired by the sensor network, and to process the output of Cytosensor using only a web browser. Second, the programmers can maintain and upgrade the sensor software remotely. The third purpose is the security and safety issue. The security of the internet is not only powerful but also effective to deny access to unknown users. With regard to safety, the researchers can be protected from dangerous chemicals or biological hazards.
These applications are implemented by JAVA applets [14] - [15] and XML [4] (Extensible Markup Language), with both of them easily integrated. A JAVA applet is a special program which one can embed in a web page so that the applet gains control over a certain part of the web page. XML is in many ways nothing more than a standard method to describe data and make itself compatible with the P2P (Peer-to-Peer) model [7] for data search and retrieval. The MATLAB Webserver toolbox [16], JAVA applet, and XML are mainly used here for the remote processing part. To operate MATLAB [16], the web server software and a PERL program [8] are employed as a data provider and a trigger of the processing device. With a single command in the JAVA applet in the user’s web browser, a sequence of images will be uploaded by the FTP server or P2P model as soon as the MATLAB Webserver receives a request from the user. During this operation, MATLAB writes XML files [5] in order to provide information from the image processing results, such as, cell-detection, color segmentation, calculation of cell area, etc. This action is ‘Full Duplex’. However, MATLAB Webserver has some weak points like long processing time as compared to independent JAVA Image processing programs [11]. Further work should be to translate MATLAB code to JAVA. This is discussed in Chapters 4 and 5 of this thesis. Another possible contribution of this work is to develop new advanced algorithms for image processing in JAVA [12].

The thesis organization is as follows: Chapter 2 describes the communication between Cytosensor and a user’s web browser using the Controller and Observer. Also, P2P technique [7] and the merit of JAVA applets are revisited with details in this chapter. Chapter 3 presents the remote image processing using MATLAB Webserver and user-friendly HTML pages containing JAVA Applets. Chapter 4 is devoted to the Machine Vision Algorithm in JAVA as well as future works for independent remote processing using JAVA Advanced Image Processing [12]. Chapter 5, ‘Image Analysis in MATLAB’ [16], illustrates the concepts of Chapter 3 to present algorithms for image processing. Finally, Chapter 6 provides the conclusion for this thesis.
2. COMMUNICATION - OBSERVER AND REMOTE CONTROLLER

2.1. Overview of the remote controller and observer

We use the Cytosensor to illustrate how a specific device can be controlled and observed using the Web-Based Distributed Application. Cytosensor [Figure 2.1] is a biological detector of toxins. It is a specialized system that can be made portable or stand-alone. Software and hardware are integrated on an embedded Windows image acquisition and processing computer (Matrox 4Sight). The software functionality can be divided into three parts: real-time image acquisition, image processing and statistical data analysis. The image acquisition is based on a high definition camera. We call this system a ‘Biological Sensor’ and named Cytosensor. The use of biotechnology in a computational device implies that there is a biological component which interacts with the data processing.

Figure 2.1: Cytosensor and Diagram Overview
Using WBDA, it is possible for a user to remotely control and observe the Cytosensor, process the acquired data, and search and retrieve all experimental results. Even if a typical Cytosensor user running biological experiments doesn’t have a specific knowledge of computer programming, he needs to process and interpret the acquired data. The results of image processing are estimated from complex probability equations and calculations. Moreover, the user would like to know the information for every single image. The information includes properties of the acquired images, the results of image processing, the properties of the sensor’s environment, etc. The environment could be the status of cameras, acquisition rate, error reports, light parameters, the location of the files, or the user’s information.

Web-Based Distributed Applications allow users to handle every function easily using a web browser. They can conduct an experiment, study the result of image processing, or search and retrieve the data without complex manuals for operation or installation.

![Overview of Web-Based Distributed Applications for Cytosensor](image)

**Figure 2.2:** Overview of Web-Based Distributed Applications for Cytosensor
Furthermore, when the Cytosensor software developer wants to test or improve the sensor operation while seated in front of a personal computer, or when several users want to run experiments at the same time but not at the same place, the remote processing and remote control provide a powerful solution. There are also the security and safety issues. Internet security is not only powerful but also an effective and proven concept. Remote access provides a safety feature while using dangerous chemical or biological toxins.

Each experiment generates a large amount of data. This makes it difficult to search and retrieve specific images and their processing results, which are distributed among various sensor storage spaces. We discuss here the new search engine using the P2P model [7], which is a popular technique used by modern applications [Figure 2.3].

![Figure 2.3: Peer-to-Peer scenario to search for and retrieve data](image)

To realize the solution, this thesis presents the Controller and Observer. All functions of the Cytosensor can be remotely controlled and observed from the user web browser. The screen captures of Controller and Observer are attached in the appendix. Controller is simply an application which can control the local Cytosensor with the same interface on a web browser, and Observer allows the user to watch image sequences which were acquired during an experiment like a slide show [14].
The P2P model for the new search system is not yet established as a JAVA applet. It can be an applet on the same web page, and this task will be my critical job in the near future.

2.2. Needs and specifications

Certain criteria must be met to create Web-Based Distributed Applications for remote access and control:

- From a non-technical user's point of view, the interface on the web page has to be friendly and easy to use. (One click solution)
- The sensor has to be controlled and observed.
- Every complex procedure must be a transparent one. The output of image processing and acquired image data should be obtained easily.
- From a programming point of view, special regard has to be paid to the multi threading.
- Cytosensor has to have a function as a server.
- Cytosensor must not be overloaded.
- The JAVA applet on the web browser has to be associated with Cytosensor for stable and accurate remote control.
- All information has to be easy to publish, track and retrieve.
- Both the sensor and the Web-Based Distribution Application have to know and share statuses with each other.

The web browser provides the user-friendly interface. Most people comfortably use the internet web browsers. It is common, but has unlimited possibilities when it is used with JAVA applet. Most of today's popular browsers support JAVA applet. It is able to embrace and extend the existing functions of a web browser. The JAVA applet is the key to running Web-Based Distributed Applications, which provide effective communication to control and observe the sensor.
In this thesis, the commands from a user to Cytosensor are based on socket programming using TCP (Transmission Control Protocol) [14]. JAVA has simple class packages to open socket and send commands to the server with only a ten-line program. At this moment, Cytosensor has to have a built-in server to get the commands from the user. Additionally, the security issue can be solved, because every user has to tell his IP address to the server side whenever he tries to log in. If the IP address is not registered, the server side will not open a socket for an unknown user. The efficiency of thread-parallel JAVA programming for real-time systems is also important to manage the commands. The future method for sending user commands is based on XML-RPC. It is a set of implementations that allows software running on disparate operating systems, running in different environment, to make procedure calls over the internet. This is remote procedure calling using HTTP for the transport and XML for the encoding.

The advanced search engine described in this thesis is based on the Peer-to-Peer (P2P) model [7]. A central server receives the keyword from a user and sends the query to every single sensor that can search the proper data without external help. The sensor and user are ‘Peers’ in this case.

The reasons why we chose the XML and HTTP server scheme are because XML is apt for web browsers and is a web-friendly programming language. Even though the programming language is different from JAVA, this application can read the data from a JAVA applet using a parsed XML file. An HTTP server is a fundamental and flexible equipment to send the JAVA applet and XML files to the user’s browser [9]. Thus, a web browser can be associated with the sensor for remote access, control, and observation. It is also involved in P2P (Peer-to-Peer) technique to search and retrieve the data.

2.3. Software design – The kernel technique for the server side

The future of Cytosensor is not only to provide acquisition data but also to distribute the results of experiments to users on the network In particular, it has to be
remotely controlled by Web-Based Distributed Applications, which is written in a JAVA applet that will be downloaded from the server. The server program needs to be constructed as part of the sensor to satisfy the above requirements. The sensor can be called the ‘Server Side’ in this chapter. The Apache HTTP Server was selected from among several server programs since it works well with Windows and it has been the most popular web server on the Internet. The August 2002 Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined [15].

### 2.3.1. Command handling

Embedded systems and devices are becoming more common and powerful. In order to be a real-time multi-task system, it must be guaranteed that sensor functions perfectly with multiple running tasks (or threads). As the system becomes more complex, more threads are needed to manage various events. From the programmer’s perspective, managing these threads can be burdensome and onerous. Fortunately, tools have been available for the multi-thread programming in JAVA. Java has built-in support for multi-threading and a standard threads interface [2].

The command handling proposed in the thesis is based on socket connection [Figure 2.4]. Network programming in JAVA [14] is quite easy. When the user runs a web browser to access the server, the socket program in the JAVA applet opens connections. The JAVA applet uses the ‘Input Stream’ method to extract data from or sends commands to the server.

![Figure 2.4: Command line using Socket programming](image)
Another improved method is the use of XML-RPC [4]. XML-RPC is a simple, portable way to make remote procedure calls over HTTP. It can be used with Java, C++ and many other programming languages. Compared to a simple transfer of command with socket programming, greater efficiency can be achieved when local machines are controlled or managed by a command list format with XML. The Cytosensor network has a complex structure to operate its job upon receiving only one command. It is obvious that XML-RPC (Remote Procedure Call) is more efficient than socket-programming in a chatting format, particularly because Cytosensor operates multiple channels, through which various combinations of command lists would be delivered at the same time. HTTP is used for the transmission mechanism. XML is the container of the information. HTTP is apt for Internet and standardized, and XML is flexible and extendable. The most important character of XML-RPC is interoperability.

Figure 2.5: XML-RPC

On the XML-RPC server side, there is a top-level program that sits in a loop waiting for something to happen, and then distributes control to a hierarchy of procedures that respond. This is possible because there is no difference between a local procedure call and a remote one. The user’s command allows the server to execute one of its methods, which can have various parameters. A procedure call
receives a returned value. This is the answer that the procedure sends back to the origination procedure. XML-RPC also has a standardized cross-platform approach. The common language that allows communication between different platforms is XML.

A more important part of this project is the use of multi-threading to handle the commands. The commands have to be accepted no matter what the sensor is doing. This requires a flexible and modular program. `ClientListener` is the monitor concept (synchronized-wait-notify) program written in JAVA and it can elevate the performance criteria such as receiving commands accurately and executing operations correctly [15]. It is ready to listen to the client’s command constantly while running other tasks requiring huge amount of memory and long processor times. In the implementation of the Java monitor, threads are added to the end of a single synchronized queue when the monitor has already been claimed by another thread. This function can be used for the JAVA applet of the client side also, because the JAVA applet on the client’s web browser has to be ready to parse XML file, which contains information from the server side.

```java
public void run() {
    synchronized(key) {
        try {
            System.out.println("Thread ID? "+this.toString());
            if(theThread.getMessage() != null) {
                if(theThread.getMessage().equals("go")) {
                    goSomeWhere();
                } else if(theThread.getMessage().equals("stop")) {
                    stopGoing();
                }
            }
            key.wait();
        } catch (InterruptedException ie) {
            System.out.println("Thread ID? "+this.toString());
        }
    }
}
```

```java
//This observing method is inside of java.util package
//To use this, just type 'import java.util.*;
public void useThis() {
    String str = theThread.getMessage();
    // To use AddObserver, just call other class
    // such as 'Play Video', 'StartGrab' and so on..
    AddObserver(new ClassForDoingSome());
    while (str != null){
        setChanged();
        //notify str to other class
        notifyObservers(str);
    }
}
```

Figure 2.6: Multi-threading example

Figure 2.6 illustrates an example of the thread method. The run() method makes `ClientListener` run until the user orders ‘stop’. To avoid checking continuously
the thread sleeps if there is no change of command. It causes the server disruption because ClientListener is holding one memory slot inside of processor while running.

Whenever ClientListener gets a new command, it wakes up and says ‘Hey! I got a new command! Take it now!’ to the executing programs inside of the same machine. However, the sensor (server side) doesn’t need to make as many threads. When a new command arrives at the sensor, the sensor makes a new thread and gives it a new ID. All commands are of the ‘go and stop’ type. When the user sends a ‘stop’ command, the thread which is already running will be terminated. The sensor can tell which task it needs to end up according to the thread’s ID. For example, assume that two threads are running already, and when client sends another ‘go’ command to the sensor, the sensor makes new thread and gives a new ID to the new thread. This new thread falls asleep as soon as the sensor receives the command and executes another operation [Figure 2.7].

![Figure 2.7: Thread control Flow](image)

This idea is revisited again in Chapter 3. MATLAB Web Server Toolbox is started by a PERL [8] script on the user’s web browser. Once MATLAB starts, the user cannot easily start, stop, or execute other tasks. Furthermore, the processing unit handles a huge amount of image sequences and calculates complex algorithms to provide the resulting data. This thread system can effectively lighten the load of the processing unit.
2.3.2. **Distributing information using XML**

Between the server side (*Cytosensor*) and the user side, there is a large amount of transferred and stored data. There are a great number of total frames of all acquired image files, since the sensor has 12 channels. Each channel acquires images every 4 sec, and experiments usually take at least 30 minutes. Every image file has unique information. Also, the environment of each sensor’s camera would be different from the others. We need a proper and accurate method to record in a document format.

Is there any well-described and efficient documentation for information searching? The answer is XML (Extensible Markup Language) [4], which is in many ways nothing more than a standard method to describe data. XML can also be used as an intermediary for communication between two different software program languages. Because of these merits, XML files can be used as observation tools to share each side’s status, and can also be used for P2P, which is a file search and retrieval model. In order for a JAVA applet to read XML, parsing is required.

![Figure 2.8: XML files containing information and image properties](image)

For the parsing, we can use the SAX (Simple API for XML) package supported by JAVA SUN [15]. Basically, a parser makes a JAVA applet or application understand what XML says. A parser has two parts, which are lexical analysis and grammatical analysis. The purpose of lexical analyzers is to take a stream of input characters and decode them into higher-level tokens that a parser can understand.
Grammatical analysis is the rule for creating real documentation. For example, words and sentences found in authentic contexts are organized by writing grammatical descriptions with strict rules. Using these two parts, the parser checks an XML file and if the file follows a certain pattern called DTD (Document Type Definition), the parser works out what function each piece of the document plays, and writes the new information in memory. A DTD is a formal description in XML Declaration Syntax of a particular type of document. It defines out what names are to be used for the different types of element, where they may occur, and how they all fit together.

The Document Object Model (DOM) is an XML model. 'DOM Core Level 1' [5] presents writing, reading, modifying and deleting of XML files. Its specifications are determined by W3C® and it can allow DOM to implement XML as user wish. There is no restriction among the computer languages. Microsoft, IBM, and Sun support DOM. Interfaces are provided for the management of elements, attributes, text nodes, and the other XML node types. When speaking of these entities in general, we will call them document nodes, and they will contain the information of all image data, which can be acquired images, the results of image processing, or processed graphical data. The information can also be the environment of the Cytosensor such as acquisition rate, the status of cameras, error reports, light parameters, the location of the files, or the user's information. Cytosensor writes the location of the images in the XML document using adding and removing an attribute as a DOM element while acquiring images. With the parent and child nodes, XML holds various information about images or statuses of Cytosensor.

In the same manner, an XML file can be a search list for the Peer-to-Peer search model. The user doesn't need to type in search words term by term. He can simply write all properties of the image or processing result which you would like to retrieve. Then, the server finds a match data and tells the user the location of the file. Whenever Cytosensor is notified of changes of environment, it rewrites and updates the XML file. On the user side, the primary purpose of the Java DOM API is to allow Java applets to interact with and modify the document in which they are embedded. Moreover, the 'DOM Level 2' [6] specification defines events which can be triggered
by physical user actions such as GUI (Graphical User Interface) events which modify the document structure and content. The future work is ‘Concentrating on DOM Level 2’, because a JAVA applet or Cytosensor can produce various effects as soon as it reads an XML file without additional action from the user. This eliminates one unnecessary step.

It is also straightforward to use XML over the Internet. An web browser must be able to read XML documents as quickly and easily as HTML documents. The HTTP server provides this feature. XML files will be created in the web server folder, and a user retrieves them like HTML, JPG, and GIF files. The properties of XML are much more useful than those of HTML files while both formats’ file publishing schemes are identical.

2.3.3. P2P (Peer to Peer) - Data sharing and search

At the present stage of this project, XML files are stored in the ‘cgi-bin’ folder. A PERL program search engine and a simple FTP server work together to allow users to find and download data. In order to meet the previously stated specifications, the above search engine was used as a simple solution. However, it now appears to be non-versatile and primitive. A high performance search model is required, since several sensors running continuously can acquire a large amount of data. When seeking another method that differs from a common structure established between the client and server, a P2P based-model is a good solution because its dependency on the server will be reduced, so Cytosensor will have a more reasonable work load [7]. Furthermore, we have to run at least one HTTP server to allow downloading of JAVA applet files and html files and to run Web-Based Distributed Applications for Cytosensor on client side. We can use this server as a P2P server.

Users can directly connect with other users to download the data without involving servers. In this model, every client has a JAVA application named ‘Arbiter’ on his personal computer. With the application, the user sends his requests and receives the Arbiter’s response. The steps explaining how this system works are as
follows: First, if a peer needs to search and retrieve information, instead of communicating with another peer, it contacts the server. Second, the server is searching for shared resources among the sensors. The server asks the sensors which one has the proper data. Finally, the user knows which sensor to connect with him to get the data, according to the server's response. Then, the user requests a download with a unique ID, which can be the IP address in this case. At this moment, the Arbiter in the sensor allows the user to take its data and another Arbiter in the user's computer will open the browser to receive shared data. Certain criteria must be met to create this application. Search options should be clear, the application should sort well, and a user-friendly interface for file downloading is required.

Figure 2.9: Scenario of P2P Model

The server maintains a simple database that holds the registered information of all logged-in sensors, including their IP addresses. It is important to notice that all components are written in XML. We have chosen XML as the communication medium because it is legible to all contemporary programming languages. There are two prewritten components used for handling XML in this application; these are the XML parser and the XML creator which used for Controller and Observer. Based
on this application’s P2P computing architecture, we categorize the relationships of the four components in the following way: user to server relationship, server to sensor relationship, server to user relationship, and user to sensor relationship. The user opens the local Arbiter browser and attempts to connect with the ‘main server’ which collects responses from the sensors and users. Through the authorized login ID, the user sends the key word to the main server.

The main server queries to every registered sensor whether the sensor has the data or not, using the key word. The Arbiter of each sensor reads and checks all the XML documents, which were written when sensor has acquired the images. Next, each Arbiter sends the response ‘Yes’ or ‘No’ to the main server. Because the main server is just sending and receiving a short response to and from each sensor, this task doesn’t require much of the processing time. If a certain sensor sends a ‘No’ response to the main server, the main server will delete its IP address automatically on the checking list to avoid redundant tasks such as repeated searches. If a certain sensor has the XML file containing the key word, which the user is seeking, the Arbiter of the sensor sends ‘Yes’ to the main server and the main server identifies the IP address of the sensor using basic socket programming.

Arbiters of the sensors are also turned on by the built-in server, which is used for the remote control. Moreover, the sensor writes XML documents continuously. This document is no more than the list of files. With the parsing technique and DOM behavior, Arbiter can search the XML file list document easily and find out whether the sensor has a requested file. As soon as the Arbiter responds ‘Yes’ to the main server, Arbiter writes the new XML file, which contains the found file location matching the key word. This XML file is ready to be published in a specific directory. The connection between the main server and, the sensor has to remain established at all times and both can access this connection without any authorization. In this case, the main server has to have a static IP address. This IP address is delivered to the user from the main server. Such a system flow has merits, because the main server doesn’t do a complete search by itself. Every single sensor’s Arbiter participates in the
searching task. Since there are many sensors, the overall search time can be significantly reduced.

In the case of an acquiring sensor, when the user requests a connection, we can use the common internet login procedure of the internet. Up to this point, we can use a simple socket programming to send and receive information that consists of few bytes messages such as key-words, ‘Yes’, ‘No’ and IP addresses. Now, the two machines are able to communicate with each other. The sensor sends the XML document that contains the information file list of the search result. Here we are using a program to download the data from the sensor to the user machine. Simply puts, the sensor manages its output in a specific shared folder.

Since the XML file specifies the location of the data, the user simply accesses this location and transfers the required data. Unfortunately, the applet on the web page doesn’t have the permission to write any data on a local hard drive. For this purpose, an installable package has designed. This package must be downloaded and installed by the user and it has full permission to write and read the data on local hard drive. This program can be improved and will be developed further.

For now, the sensors can map their shared directory for each other, and the user who runs the web browser can transfer the files remotely from the sensor to the main server using one of Controller’s functions. Here, the main server is an HTTP server, and the user applet can download the data from the mapped directory. This solution is not fully satisfactory, since the sensor has to send data to the main server and the user retrieves it to his local storage. This procedure causes a detour between the sensor and user.

2.4. Software design - The kernel technique for the client side

2.4.1. General Function of the JAVA applet

To run Web-Based Distributed Applications for Cytosensor, the user needs a browser which is capable of running Java. Using a *.jar file to compress JAVA classes, the user’s web browser downloads the JAVA applet automatically when it connects to
the server. Suitable browsers include Microsoft Internet Explorer version 3 or higher, and Netscape Navigator version 3 or higher. While using the Microsoft Internet Explorer version 6 browser, one needs to install Microsoft's Java Virtual Machine (JVM) to run the applets. This is done automatically; once the browser detects an applet program, it checks the right version of JAVA and recommends installing JVM to user [14]. The JVM can be viewed as an abstract computer. In other words, it is implemented in software on top of a "real" hardware platform and operating system. Java programs are compiled for the JVM instead of for the system [Figure 2.10]. That means that the programmer writes his Java program and the Java compiler translates that into language that the JVM implements. These instructions are then executed to generate the desired output.

![Figure 2.10: General overview of JAVA applet](image)

Controller and Observer are the two main JAVA applets of the WBDA. Controller provides all functions of the local Cytosensor from the user's web browser. The Cytosensor functions are executed using GUI type buttons. With the Controller JAVA applet running in the browser, users can see and control the same interface of the Cytosensor, including the location of all the buttons. Each button of the applet sends a different command message to the Cytosensor, which contains a built-in server. Controller commands to the server through an internet TCP-IP sockets connection [9].
The Java API includes extensive support for sockets and network programming, and yet is simple to use. Both Controller and the server can send and receive data streams with each other. When the user clicks the button on the JAVA applet interface [Figure 2.11], the JAVA applet send short message to the local Cytosensor and the Cytosensor executes a corresponded operation.

![Figure 2.11: Screen Capture of Controller application](image)

Changing the tab above the display screen lets the user meet Observer. The screen will be shifted to 'Observer' mode, and a new control menu will be shown. Observer doesn’t exist in the local Cytosensor. The remote user who controls Cytosensor will be anxious to watch whether or not it works well. Also, the user will want to know whether or not the sensor is acquiring good quality images and saving the image sequence properly. The roles of the Observer provide a real-time display
and a slide show. The slide show allows the user to watch previously acquired image sequences starting at a certain point of time to satisfy the user's will. It not only displays images, but also proffers detailed image properties and information such as the name of the camera (channels), status, date, parameters of the light system, and error report to the user's web page.

*Cytosensor* provides XML files to the user's applet. An XML document describes changes of environment such as acquisition rate, JPEG compression rate, initialization of a new experiment or terminating of a running experiment, and error reports when *Controller* changes the status of *Cytosensor*. The XML file is parsed and written in the list, and the user can select a specific acquired image name. The information (properties) of each image is available beside the list window. Also, the XML file informs the user's browser of all locations of the image files, and the HTTP server provides compressed images [Figure 2.12].

![Figure 2.12: Menu bar and properties of images in Observer](image)

2.4.2. Reading information using XML (Alfred parsing) in applet

The XML parser is an important tool to translate XML documents into the language which JAVA applet can understand. Here, we consider a certain parser which enables *Controller* and *Observer* to run efficiently. There are several considerations for any 'good' parser.
- It must be as small as possible, so that it doesn't add too much to your applet download time.
- It must be compatible with most or all Java implementations and platforms.
- It must use as little memory as possible, so that it does not take away resources from remaining programs for real-time broadcast in future.
- It must run fast, so that it does not slow down the rest of the programs.

The AElfred parser (Ælfred, Microstar Software Ltd.) [4] is a small, fast, DTD-aware Java-based XML parser, and is especially suitable for use in Web-Based Distributed Applications for Cytosensor. AElfred consists of only two class files, with a total size of about 24K, and requires very little memory to run. AElfred is now free for both commercial and non-commercial use. Also, this parser runs successfully on Microsoft Windows.

![Image of XML document and displaying panel](location)

Figure 2.13: XML document and displaying panel
With the advanced programming of the JAVA applet, which can handle the XML file more efficiently using this parser, the compatibilities of Controller and Observer can be enhanced. XML can be used for the P2P search model using the same technique. Even though more element nodes in the XML file will be created more to hold other properties, the JAVA applet can read XML files faster using the AEIfred parser. We can handle longer and more complicated documentation without being concerned about the translating time. The JAVA applet can register an event when one of the image names is selected. In this case, information will appear on the properties window in response to the selection event in JAVA programming [Figure 2.13].

2.4.3. Future procedures and possible improvement

There are several tasks which we need to address in the near future. In general, applets are prevented from reading and writing files on the user's system. Furthermore, in the case of network connections, the JAVA applet can only connect with the originating host (in our case the local Cytosensor), because of the applet security policy. When we need to install an executing program onto the user's machine, the program has to be user-friendly. It should be quickly installed with one click. Similar JAVA programs exist in the internet world and this is not a complicated or difficult job.

In this form, you can find the specific data (images) following method. Every image file has same format such as '001-Mar-2003-0001-0001' which means 'Camera No.' - 'Month' - 'Year' - 'Hour & Minute' - 'Sec & Micro Sec'.

Figure 2.14: Perl Search Engine on the Web Page
Currently in this project, a PERL program search engine and a simple FTP server work together to allow the users to find and download data [8]. The user requests the search from the server, and the server displays a dialog box [Figure 2.14]. Most internet users are familiar with this kind of search engine. However, in order to meet the previously stated specifications, this search engine was only initially used in our project. The PERL programming writes a new HTML document and user’s web browser reads it. An HTML document is not active and it has only simple functions. A high performance search model is required since several sensors run at the same time conducting parallel experiments, The solution is the P2P model and detailed explanation of its network modeling is presented in Chapter 3.

The Slide show allows the user to preview the acquired image sequence. The user can choose a specific time period to view. Observer can present captured still images as compressed JPG Files.

![Figure 2.15: General overview of future JAVA applet](image)

Another option of the Slide show is to play a movie clip. When the camera starts data acquisition, the local Cytosensor can record *.avi files and provide them to the user. A future applet of this project will be able to broadcast a movie to the user’s web page in real time. In this case, more advanced control is required to handle complex functions. To achieve this goal, the Java TM Media Framework API (JMF)
[15] can be used to enable video in Java applications and applets. The current local Cytosensor JAVA version is using this new technology. With this package, we can capture, play back, and stream data; extending the multimedia capabilities of the Cytosensor.

Another improved method to send commands is XML-RPC. Sending a single command from the user to the server is inefficient. The user can send various parameters to the server in one step. XML-RPC is a portable way to make remote procedure calls over HTTP. It also has a standardized cross-platform approach. The common tongue that allows communication between different platforms is XML. It can be used with Java, C++ and many other programming languages. The most important character of XML-RPC is interoperability. We are testing new remote control model based on XML-RPC with Apache XML-RPC. By adding the parameters in “some parameter” [Figure 2.16], the user can control the local Cytosensor efficiently and can receive a response.

```java
XmlRpcClient xmlrpc = new XmlRpcClient("http://localhost:8080/RPC2");
Vector params = new Vector();
params.addElement("some parameter");
// this method returns a string
String result = (String) xmlrpc.execute("method.name", params);
```

Figure 2.16: Sample code of XML-RPC
3. REMOTE PROCESSING

3.1. Overview of the remote processing

The purpose of remote processing is to allow the user to process results, using only the internet web browser. The processing of *Cytosensor* data is just an example of this remote processing model, as this model can accomplish various kinds of processing. Here, processing implies image processing and statistical data analysis such as cell detection, color segmentation, calculation of cell area etc. The image processing is based on complex algorithms, and it is assumed that most users of Web-Based Distribution Applications are not interested in the details of such a process. Thus the data processing for a typical user of Web-Based Distributed Application is a transparent procedure. The main user of this remote processor may not care about the complicated procedure.

The main parts are the processing unit and the user's web browser. Components of the processing unit are MATLAB®, MATLAB Web Server Tool Box and HTTP Server. MATLAB is a high-level matrix language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It includes facilities for managing the variables in its workspace and importing and exporting data. On the user side, the main technical parts are the JAVA applet and XML. The realized abilities of these this chapter are similar to things presented in Chapter 2.

The overall scenario shows how the user who is at a remote location handles the Web-Based Distributed Application on his or her web browser [Figure 3.1].

![Figure 3.1: General overview of remote processing](image-url)
However, this model doesn’t achieve good enough transmission speed if the network condition is not favorable. It provides the data to web pages using the HTTP server after saving the results to the server. Therefore, independent JAVA applet processors could be a good solution because their processing time is usually faster than that of MATLAB. It skips the file transmission step. Development of the new JAVA applet will be discussed in Chapter 4.

3.2. Needs and Specifications

The following are the basic requirements for the web control:

- From a non-technical user’s point of view, the interface and web page should be user-friendly. ("One click solution")
- Complicated and difficult processing procedures must be transparent.
- The processing unit has to be controlled and observed.
- The user can access the output of image processing and the acquired image data.
- It should operate on a complicated sensor network, and all information has to be easy to publish, track, archive, and retrieve.

Ideally, the user should be able to run an application and obtain the result by clicking just one button. The web pages I built for this project have a Web-Based Distributed Application for remote processing (One-button operation), the manual session, file transferring session, search engine page, An Indicator that observes the processor’s status, and the display session [Figure 3.2].

The processing unit, which executes the commands from the web, has the HTTP Server. Of course, the user shouldn’t have to worry about network complexes and just needs to know how to retrieve the results from the server. The JAVA applet named CytoMat displays the processing results. The capabilities of CytoMat are similar to the
Observer's functions [Chapter 2]. Also, XML is used as an information container and the method of parsing XML files is the same as the one used by Observer.

Figure 3.2: The steps of remote processing using web pages

The advanced search engine presented in this thesis is based on the Peer-to-Peer model. The central server receives the keyword from the user and sends the query to every sensor. Each sensor can search its own database without any external help while using the JAVA application. Here, the search engine model is not a 'Pure P2P'. In this case the user is the only "Peer" which communicates with the server to receive and send data. The speed and accuracy have to be positively guaranteed. Another JAVA applet program called Indicator monitors continuously the condition of remote processor. JAVA Multi-Thread programming provides the means for the proper operation of the Indicator by using ClientListener and the AElfred parser.
3.3. Processing unit – MATLAB Web Server Tool Box

3.3.1. Image processing with MATLAB

In this project, the main image processing tasks are concerned with individual cell detection and the color analysis of a cell’s behavior in a time series. All of the algorithms were prototyped using MATLAB and next translated into MATLAB Web Server Toolbox code for use in the real-time software architecture.

The proper answers for “Why MATLAB is selected for the processing”, split into two parts. These are “programming for user” to create a complete user-friendly interface, and “programming for developer” to create quick and dirty throwaway programs. The advantage for the user is that data analysis can be visualized with scientific graphics, and a GUI (Graphical User Interface) can be designed. Graphic data is easier to understand than text data for common users. From the developer point of view, the reasons why MATLAB was selected are:

- It is used in a variety of application areas including signal and image processing.
- It provides core mathematics and advanced graphical tools for data analysis, visualization, and algorithm and application development.
- Toolbox collections, which are highly optimized, application-specific functions, can extend MATLAB functionality. In particular, for signal and image processing, several toolboxes are available.
- MATLAB interfaces well with programs written in other programming languages, such as C++ or JAVA. In the case of Java, MATLAB can call Java classes and methods.

Using the image compression techniques, the images and analysis results are stored as JPEG format files. This is specific to the Cytosensor project, and a detailed investigation of image processing for the Cytosensor project will be presented in
Chapter 5 of this thesis. Of course, similar MATLAB programming for other projects can do different processes. To present remote processing in this thesis, the *Cytosensor* project will be the example model. In the case of the *Cytosensor* Project, the user checks the condition of the cell area where the toxin is injected, and observes it as time progresses. Generally, the area of the cell is diminishing, and the user investigates some biological reaction based on the change in the cells' total area.

![Figure 3.3: Overview of image processing of Cytosensor project](image)

The above figure shows the actual output images. On the left, there is the original image of the cells acquired by *Cytosensor*. The middle image shows the detected cell frames, and the image on the right side shows that the area of the cell has decreased at the end of experiment. This figure illustrates the changes of cell properties in time as represented by an image time series (x-axis).
3.3.2. **MATLAB Web Server Tool Box**

The original MATLAB Webserver Tool Box has limited capabilities, which include receiving several parameters from the web page, processing data, and displaying the results (mainly graphical) on the same web page. Once started, the processing can not be stopped. The maximum amount of HTML data you can receive from MATLAB is just 256 KB. However, the MATLAB Web Server Toolbox capability can be extended using the concepts presented in this chapter. First, using a MATLAB M-file that can receive the data entered in the HTML input format, we can program an active M-file, which is able to write and read XML documents. Fortunately, MATLAB can call the JAVA classes and methods using the DOM [Chapter 2] to read and write XML files. After reading the XML document, the active M-file can start or stop running other M-files for multi tasking following the commands placed in the XML document by the user. These documents can be received by the HTTP web server built in the same machine. The user’s WBDA can also read and write XML. The procedure by which the MATLAB Web Server Toolbox can send processing results to user’s web browser is shown below [Figure 3.4].

![Figure 3.4: Procedure of remote processing](image-url)
The user can upload the acquired image sequence from Cytosensor to the processing unit using FTP or P2P, and process them using the MATLAB program. The command 'Start' comes from the user's web browser through CGI (Gateway Interface). CGI extracts data from HTML documents and transfers them to the MATLAB server. When user calls *matweb.exe*, which establishes communication with MATLAB, it executes the specific M-file. To send the commands to the server, we need to set the proper arguments in the HTML document, such as the name of the M-file. As soon as the server receives it, *matweb.exe* recognizes the name of the file to run. The processing results, are placed in a particular folder and HTTP server delivers them to the user's web browser.

3.4. Remote processing with HTML page containing applet

3.4.1. MATLAB and XML- Upload & Processing Indicator

There are two methods to retrieve the images acquired by Cytosensor. The first one is the direct connection method. To use this method, press the 'FTP' button on the Controller GUI [Chapter 2]. The files are automatically transferred to the processing unit. Figure 3.5 shows the second method.

The user can easily upload and download data using the FTP server and the above user-friendly interface. Clicking one of the channel-buttons causes the new popup window to appear on the user's screen. Using simple operations ('drag and drop' or 'copy and paste'), this interface allows the user to transfer the acquired image data from the local Cytosensor to the remote processing unit. After checking the images using any kind of image viewer, the user can upload them to the processing unit directly within this web page. To upload the data, 'drag and drop' or 'copy and paste' it into the white box.

Indicator displays every step of the processing procedure. This JAVA applet program reads the XML file, which the server sends when there is a change. XML parsing and multi-threading techniques are used for this applet [Chapter 2].
3.4.2. Display the processing results

The JAVA applet named CytoMat serves not only in displaying the image, but also provides details of each image properties such as cell area, major axis length, eccentricity, euler number, solidity, and mean color to the user's web page using XML document. Cytosensor is an image capturing device and a remote unit processes the output of the Cytosensor. The results of this process are acquisition of live cell images, detection of cells, and data analysis the data. Therefore, the outcomes are images of the cells and graphs of certain processing results. The main
functions of *CytoMat* are arranging the graphical images and displaying images and graphs at the same time to compare and investigate. Figure 3.6 shows the functions of the interface of *CytoMat*.

![Figure 3.6: Screen capture of CytoMat](image)

In the future, *CytoMat* can become the WBDA not only for display but also for processing. Currently, *CytoMat* depends on the MATLAB Web Server to do the processing. Thus it is a weak point because the processing time is long as compared to the independent JAVA Image processing program. The future work is to implement MATLAB code in JAVA to improve the processing speed. Continuation of this effort can lead to the development of new JAVA based advanced image processing algorithms.
3.4.3. **P2P (Peer to Peer) – Data sharing and search**

The processing results are compressed in JPEG format and saved on the HTTP server. Also, the property files of every single image written in XML are saved in the same location too. There are easily thousands of images and documentation files on the server since each image has over 100 cells, and Cytosensor acquires on the average 200 images per experiment. A high performance search model is required, and a P2P based model is a good solution to this problem. A new search engine is proposed here based on the P2P search model as shown in Figure 3.7. In this case, the search model is not pure ‘P2P’ because there is only one peer (the user). However, the basic concepts and the technology are the same because the server and user are using Arbiter [Chapter 2].

Thus, the procedure of data search, sharing and retrieval of data can be made transparent from the user’s point of view. The user types the search key words in the Arbiter’s interface, and the list written in XML is sent to the processing unit. The processing unit searches all storage and finds out where the data is. The processing unit conveys the search results and shares the proper directory with the user. The user accesses the directory and downloads the image files or information documents according to the server’s guidelines.

![Figure 3.7: Scenario of P2P based search model](image-url)
4. IMAGE PROCESSING AND MACHINE VISION IN JAVA

4.1. Remote image processing in JAVA applet

4.1.1. JAVA Image processing

There are many programming languages and software tools suitable for image processing. One of the popular computer languages is C++, but to build the Web-Based Distributed Application, the JAVA applet is preferred because an applet can be embedded directly in the web browser. When the user wants to process an image using the remote processing unit [Chapter 3], the data has to be sent to a server and the processing results are sent back to the user. This procedure is inconvenient and inefficient because it often depends on the network load condition.

The Web Based Application should be a stand-alone, thus overcoming the above drawbacks. Fortunately, JAVA has a powerful tool to interpret, analyze and process the images. JAVA supports JAVA 2D API (Application Programming Interface) [3], which provides enhanced two-dimensional graphics and imaging capabilities for Java programs through extensions to the Abstract Windowing Toolkit (AWT) [15]. The AWT provides the java.awt package for creating and managing windows. Because most users are familiar with handling Windows based applications, this GUI helps them to use JAVA program. JAVA 2D supports persistent image objects, advanced image filters, and various color models and image formats.

Figure 4.1: Overview of the JAVA structure for the Image Processing
While we have used the JAVA 2D, the Java Advanced Imaging API (JAI) [15] is more powerful, is compatible with the Java 2D API and has many routines for image processing. With the aim of retaining the flexibility of Java applets, the JAI API enables developers to easily incorporate high-performance, network-enabled, scalable, platform-independent image processing into Java technology-based applications and applets. It also provides image processing techniques ranging from simple operations such as contrast enhancement, cropping, and scaling to much more advanced operations such as geometric warping and frequency domain processing.

### 4.1.2. Image processing - Implemented by Java Applet

In the case of JAVA, it is straightforward to read image information such as ‘width’, ‘height’, ‘pixel data’, ‘RGB values’ and so on. For example, to read the pixel value, one uses a single call: ‘grabPixels()’. This is the first step in translating from the Machine Vision Algorithm to a JAVA program for the image processing. The pixel information can be stored in the form of an array defining the RGB color model. Once the object is converted to the array form, the program is able to analyze the pixel information corresponding to the visual image and perform complicated calculations. With only one line, the program can create arrays in the RGB color model from image objects in JAI. Figure 4.2 shows how to use this method [12].

```java
helimage = getToolkit().createImage(new MemoryImageSource(width, height, pixels, 0, width));
```

**Figure 4.2: Integer arrays in the RGB color model [12]**

To perform fast image processing, the 3x3 pixel information can be used. The matrix shows below how this is implemented. $N(x,y)$ is that the set of pixels arranged around a pixel $(x,y)$. 

```
The JAVA code shown below uses a ‘for’ loop to build a 3x3 matrix. The order of elements in this matrix follows alphabetical order [Figure 4.3].

```java
For(int y = 0; y < height; y++)
    For (int x=0; x<width; x++)
    {
        a = input.getxy(x-1,y-1); b = input.getxy(x-1,y-1); c = input.getxy(x-1,y-1);
        d = input.getxy(x-1,y-1); e = input.getxy(x-1,y-1); f = input.getxy(x-1,y-1);
        g = input.getxy(x-1,y-1); h = input.getxy(x-1,y-1); i = input.getxy(x-1,y-1);
    }
```

Figure 4.3: To read pixel information in JAVA [12]

The next step is the combination of the filters. In order to test the image filtering with Web-Based Distributed Application, the Sobel Edge Detector is selected [12]. The Sobel Edge Detector uses a 3x3 mask to determine the edge gradient with the Roberts operator. The new components of the matrix are derivatives calculated using the two convolution masks. E1 can be approximated by E2 using a similar scheme of the Robert operation which is the simplest gradient edge detector and is calculated from cross differences in a 2x2 matrix.

\[
e = \frac{\sqrt{[(a+2b+c)-(g+2h+i)]^2 + [(a+2d+g)-(c+2f+i)]^2}}{6}
\]

This equation can be implemented as shown by a short JAVA program [Figure 4.4].
For(int y = 0; y < height; y++)
    For(int x=0; x<width; x++)
    {
        z1 = Math.abs(a + (b << 1) + c - (h << 1) - i);
        z2 = Math.abs(a + (d << 1) + g - c - (f << 1) - i);
        z3 = flow((z1 + z2) >> 2);
        output.setxy(x,y,z3);
    }

Figure 4.4: Sobel Edge Detector using a 3x3 mask[12]

The final goal of the Web-Based Distributed Application is a stand-alone real
time processor using a JAVA applet running under a web browser. Figure 4.5 shows
the example of the Sobel Edge detector in the Web-Distributed Application.
Unfortunately, the JAVA applet cannot get source data from user directly. User needs
to upload a source image file to a server using Perl program or FTP server. When the
user clicks the ‘Browse’ button, the popup window will appear. The user can select his
source data and upload it to a server. Then, JAVA applet retrieves it from this server.
After retrieving the data, the applet program reads the pixel information of the source
and re-draws the processing result on the display panel in the same web browser using
its own processing program, which is Sobel Edge detector [13].

Figure 4.5: Real time processing using stand-alone JAVA applet
Visualizing the outcome of the image processing with a web browser is one of the main tasks of the Web-Based Application in the Cytosensor project. Therefore, 'display' is an important consideration for a user-friendly interface. When the program draws the new image, a 'flickering phenomenon' [15] occurs. In order to display image sequences without flickering, 'double buffering' is required. After display of the tinted image frame, the subsequent frame is created off screen and drawn line-by-line instead of replacing the entire image at once.

4.2. Machine Vision in Java

The formal definition of machine vision is "The use of devices for optical, non-contact sensing to automatically receive and interpret an image of a real scene in order to obtain information and control machines or processes" [1985’ AVA - Automated Vision Association]. This sentence describes well the difference between Computer Vision and Machine Vision. The goal of Machine Vision Engineering is to solve specific engineering problems, not to develop ‘Human-like’ visual systems. The machine vision techniques in JAVA contribute considerably to the image processing. They are capable of making precise measurements at high speed, but lack the key elements of the human vision system [12].

4.2.1. Machine Vision technique for image segmentation

In the image segmentation, one of the popular methods is 'Watershed Segmentation' which separates neighboring objects. For the Cytosensor project, this technique is important because the biosensor needs to detect a specific object such as a cell or tissue. Watersheds are zones dividing adjacent Catchment Basins (CB), which is a local minimum representing a low gradient image intensity region interior [11]. The immersion approach illustrates this procedure [Figure 4.6].

Using a negative of the second picture's distance function, we can draw its profile. A black hole indicates each local minimum along the horizontal line. Suppose that we could pour water into it after inverting the profile. If two CBs are likely to
merge due to further immersion, a dam is built on the boundary. This dam represents the watershed line. An efficient implementation of this approach involves sorting the pixels in increasing order of gray scale values. The last picture is the result of the watershed segmentation process. If we want to segment a complex image that has several black holes, the immersion continues until all the watershed points are found and flagged. As a result, the landscape is partitioned into regions separated by dams.

4.2.2. **Machine Vision technique for object detection**

Isolated cells in an image file can be detected by an association watershed line with a binary map derived from the original image in the gray scale space. To generate this map, we use a relatively straightforward method based on automatic thresholding. Once the binary map is generated, standard morphological cleaning methods such as ‘Erosion’ and ‘Dilatation’ are applied [12]. These allow connected components to be separated clearly. Dilation is filling and growing and results in the expansion of an image A by Structuring Element B (A⊕B). Erosion is equivalent to a shrinking of the image A by a Structuring Element B (A⊖B). Figure 4.7 illustrates these specifications.
Figure 4.8 illustrates cell detection algorithm using a combination of watershed lines and thresholding to detect the cells and clear their boundaries. In many machine vision applications, thresholding is useful to separate the regions of the image corresponding to objects of interest from the background. Thresholding is based on the different intensities of color. The input to a thresholding operation is typically a grayscale image. The segmentation is defined by a single parameter known as the intensity threshold. If the pixel's intensity is higher than the threshold, the pixel is set to "white". If it is less than the threshold, it is set to "black". If the outcome is not satisfactory, we can adjust the threshold value.

Figure 4.8: Overall view of cell detection [12]
5. UNDERSTANDING IMAGE ANALYSES IN MATLAB

The results of the image analysis provide the data for users who study in the Bio-Engineering field. Analyses of image data are achieved by the remote processing unit controlled by the Web-Based Distributed Application (WBDA) [Chapter 2,3].

The remote processing unit has a built-in server and sends the analysis results in XML format to the user’s web browser. Currently, MATLAB [16] is used for image processing and analysis, but the future work could use a standalone JAVA applet processor instead of a MATLAB-based processor. MATLAB programs are easy to write, but they are relatively slow and include unnecessary steps such as file transmission between the user and the processing unit. Fortunately, the MATLAB image analysis and processing algorithms can be translated into The JAVA language. This allows us to prototype and debug a MATLAB program before porting it into WBDA.

The image analysis, which is discussed in this chapter, can be briefly characterized as follows: Assume that the cells exposed to toxins react in a variety of ways. Basically two kinds of reactions can be expected: structural change and color change [11]. For the structural change, geometrical analysis leads to the characterization of the cell shape, as defined by the binary mask computed during the cell extraction process. In the case of color analysis, the objective is to estimate the statistical model of the cell color content in a color space.

5.1. Image Analyses in MATLAB - Geometrical analyses

This process is dedicated to computing geometrical features, such as area, major axis length, minor axis length, and eccentricity. MATLAB Image Toolbox has a powerful function ‘imfeature (L, measurements)’ that computes a set of measurements for each labeled region in the image L. Figure 5.1 is an actual display of the Web-Based application. The list below explains the definitions [16].
Figure 5.1: Geometrical Features using JAVA applet and XML

(1) Cell Area: The actual number of pixels in the region
(2) Centroid: x- and y-coordinates of the center of mass of the region. (1-by-2)
(3) BoundingBox: The smallest rectangle that can contain the region. (1-by-4)
(4) MajorAxisLength: The length of the major axis of the ellipse that has the same second-moments as the region. The format is [x; y; width; height;]
(5) MinorAxisLength: The length of the minor axis of the ellipse that has the same second-moments as the region.
(6) Eccentricity: The eccentricity is the ratio of the distance between the focus of the ellipse and its major axis length. A circle has an eccentricity of zero.
(7) Orientation: The angle between the x-axis and the major axis of the ellipse.
(8) ConvexArea: The number of pixels in a convex image.
(9) FilledArea: The number of 'on' pixels in a binary image of the same size as the bounding box of the region.
(10) EulerNumber: Equal to the number of objects in the region minus the number of holes in those objects.
(11) EquivDiameter: The diameter of a circle with the same area as the region.
(12) Solidity: The proportion of the pixels in the convex hull that are also in the region.

Figure 5.2: MajorAxisLength, MinorAxisLength, and Orientation

5.2. Image Analyses in MATLAB - Color analysis

We assume that we are working in the RGB (Red, Green and Blue) domain and that we can reproduce color with additive and subtractive technique. Additive color involves the use of colored lights, where mixing red, green and blue light produces other colors. The final objective is to analyze the color of each cell. In statistical analysis, the notion of a statistic is intimately connected to the concept of a probability distribution. We can define the probability distribution function of the color space in a region and the probability density function of the color space in that region. The different methods for PDF estimation can be classified as parametric, non-parametric and semi parametric. In the parametric method, the PDF is assumed to be of a standard form, for example, ‘Gaussian’ [11]. The standard Gaussian probability density function has the following form.

\[
p(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-1/2(x-\mu)^T \Sigma^{-1}(x-\mu)}
\]  

(5.1)

Where \( x \) is RGB 2D vector representing a pixel of the image \( x \), \( \mu \) is the mean vector, \( \Sigma \) is the covariance matrix, and \( d \) is the number of color bands. This Gaussian density function can be easily obtained by computing the mean color \( \mu \) and the covariance matrix \( \Sigma \) that is of particular interest to us because it allows us to obtain the axes direction of the ellipsoids of constant probability density. These axes can be computed directly by performing the eigenvalue decomposition of \( \Sigma \).
Figure 5.3: Statistical color analysis using JAVA applet and XML

Figure 5.3 is an actual display of the WBDA and the list below explains the definitions [16].

1. Circumference: The number of boundary pixels.
2. NbConnectedComponents: The number of the connected components, which around detected cell.
3. meanIntensity: The mean intensity of the detected figure.
4. meanColor: The mean color of the detected figure.
5. Cov (Covariance): The covariance matrix $\Sigma$ (See 5.1) allows the acquisition of the axes direction of the ellipsoids of constant density of probability.
6. eig (Eigen Value): The Eigen Value of $\Sigma$ is used for axes direction of the ellipsoids of constant density of probability.
7. ProximityCount: The number of other detected components' parts in the rectangle.
8. Periphery_mean_intensity: The average of the gray scale of boundary.
6. CONCLUSION

During work on this project, I became familiar with the Web-Based Distributed Application (WBDA) in JAVA, and began to implement, optimize and integrate programs in order to perform remote access (control and observe the local sensor) and remote processing based on a web browser. I also worked on the translation of MATLAB image processing functions into JAVA code to build a stand-alone WBDA using the machine vision technique in JAVA.

A prototype of WBDA was developed according to the strict specifications proposed to the Cytosensor project. We plan to continue optimizing the performances of the WBDA until it becomes a stable, accurate, and safe application for a widespread use.

In order to improve the WBDA, we have tried the ‘Peer-to-Peer’ architecture for search and retrieval of data between user and sensors. This concept may need additional study and testing to improve its performance. A sensor network using P2P communication will allow users to share information about their respective results.

To control the local sensor and processing unit, we proposed a solution based on the XML-RPC. Using this approach, greater flexibility can be achieved compared to a simple transfer of command with basic socket programming. The interoperability of XML-RPC is a great functionality for WBDA, since the local Cytosensor is programmed in C++, but WBDA is programmed in JAVA in order to be embedded on the web browser. When the local Cytosensor is controlled or managed, HTTP is used for the transmission mechanism, and XML is the container of the information. HTTP is apt for Internet and standardized, and XML is flexible and extendable. XML is the common language that allows communication between different platforms. XML-RPC also has a standardized cross-platform approach.
BIBLIOGRAPHY


APPENDIX (SCREEN SHOTS)

- Main page of the Web-Based Distributed Application -

Greetings! My name is Liew who studies in Oregon State University as a master student with Prof. Wojtek J. Kolodziej in Cytom Team since Fall 2000.

To use this site, please read this statement carefully. This site is protected to keep the several security issues and entire project. Also, Website information can be changed and/or updated. Anyone relying on such information must bear the risk that the information provided may be changed and/or updated without notice.

The purpose of all over project is to develop a variety of biological and chemically identification of a living cell, which change embedded Windows 2000 for image acquisition and processing. Matrosk (2003) works as three parts: real-time image acquisition, image processing and statistical data analysis.

For various reason, if a research worker, who uses only web browser, can control the specific application in the acquiring unit (named Cytosensor) and processing unit (Matlab installed), it can be a great deal of assistance as far as a complex investigation is concerned. Also, handling plenty of acquired images and processed results can be an easy job if he can search and obtain them seating in front of his personal computer.
- Remote control in JAVA applet (Controller & Observer) -
There are two methods to get the grabbed images by a scanning unit. First one is direct
connection method, the other one is download images before processing.
To use the first method, just skip this step and click the "FTP" button on the Matrox Applet
which you will see next website. Besides, if you see the popup window which is saying,

This means I did NOT run the server. Email me if you need to any assistance.
To use the second method, click one of below buttons which you'd like to connect to get the
images from the specific Matrox. The popup window will appear. Just 'drag and drop' or
'copy and paste' the folder you want to process to your computer. Using any image viewer,
you can check and choose one of them.

Matrox#1 - 128.193.48.108
Matrox#2 - 128.193.??,???
Matrox#3 - 128.193.??,???
Matrox#4 - 128.193.??,???
Matrox#5 - 128.193.??,???
Matrox#6 - 128.193.??,???
Matrox#7 - 128.193.77.77

When you find and want to run specific image
server directly
with this site. Keep in mind that you have to upload the images. To
upload your folder, 'drag and drop' or, 'copy and paste' the folder into the below box. If there
are another folders before you upload, just delete them all. Otherwise, the remote access would
not be processed correctly.

Go on next step!!
Remote processing page and Indicator -

Uploading is completely done, click blow button to process. It will take pretty long time usually. Below JAVA Applet can show what the status is during the process. XML file allows you to check it out. To avoid errors, please do not close this browser and keep watching the indicator. Please make sure that you just click one time. If you won't do so, other users will get problem and waste long time to wait the result.

Please, click only one time. Below indicator shows processing is started.

If below indicator doesn't show or any changes for 3min, please email me immediatly. That means the processing unit gets over tasking and it will have malfunction not only now but also future.

After all processing is done, the Indicator will say 'Thank you for your patience' You can go next step to see and check the result. Thank you!!
- Observer for the remote processing in JAVA applet -