AN ABSTRACT OF THE THESIS OF

Bertrand Boichon for the degree of Master of Science in Electrical & Computer Engineering presented on March 28, 2003. Title: CytoSensor: An Application for Distributed Bio-sensor Networks.

Abstract approved: Redacted for Privacy

Wojtek J. Kolodziej

The purpose of the thesis is to design and develop a network of automated, distributed, living cell-based sensors, called *CytoSensors*. Their main role is to detect a variety of biological and chemical toxins. The system is designed to help researchers to carry out multitude of experiments, in order to build a practical knowledge base in toxin detection. The network is developed in accordance with *industry standards*, to be used and deployed for *prevention* in inhospitable environments such as battlefields, toxic urban locations or polluted agricultural regions.

The sensor is composed of a processing unit (processor and memory), an archiving unit (permanent data storage), a communication unit, input devices attached to a data acquisition unit, and control devices. The CytoSensor is specifically designed to acquire and analyze visual information about the living cells: hence cameras are used as input devices and frame grabbers are used as the digitizers. The control devices are additional external devices developed to help control and automate the process of data acquisition: they comprise light intensity control USB boards to provide the correct amount of light to view the cells, touch panels for user-instrument interaction, and bar code readers to identify vials and experiments. The software, on the other hand, is a complex mosaic of different elements, each of which has a specific task to accomplish. These building blocks include the real-time acquisition, archiving, networking, processing, modelling, sensor output presentation and user interfaces. Our goal is to develop, integrate and

optimize all these components to produce a viable and working device. The prototypes evolved from an offline, portable sensor equipped with a single high-resolution CCD camera and high-quality optics, to distributed online sensors with multiplexed CCD cameras and affordable optics.

The acquisition board digitizes in real time the images from one to twelve multiplexed high resolution cameras. Several operational requirements must be met. First, a fault-tolerant and stable control over the input devices and control devices must be provided. Secondly, acquisition timing errors should be minimized as a trade-off between performance and the use of a low-cost, general-purpose, industry-standard operating system such as Microsoft Windows NT. Finally, in order to reduce development time and increase code reusability, a common abstraction layer is designed to provide for flexible use with various types of digitizers and cameras.

As part of a distributed detection network, each sensor is able to exchange data with other "trusted" sensors and users, and to allow remote control of certain tasks. The sensor may be seen as a node capable of transmitting and receiving acquired or processed data to a distant device (another sensor, a workstation or a PDA) for visualization, inspection and decision-making by a front-end user. Each node on the network provides a set of complementary *services* including data acquisition, data processing, communication and system. The mandatory *system* service monitors the local system performance and manages data archiving. The communication service connects the various services on the network by enabling message-passing, file transfer and caching. The sensor network integrates a lightweight, interoperable and flexible RPC (Remote Procedure Call) protocol to achieve real-time control and monitoring of these distributed resources. A reliable embedded database system is used to store *metadata* bound to acquired and processed images. This database is also used to maintain information on neighbor nodes, and to check access credentials of available local services. Finally, by adding *store-and-forward* messaging capabilities, the application can be extended to work in wireless and mobile networks.

CytoSensor: An Application for Distributed Bio-sensor Networks

by

Bertrand Boichon

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 28, 2003
Commencement June 2003

Master of Science thesis of Bertrand Boichon presented on March 28, 2003

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical & Computer Engineering

Redacted for Privacy

Chair of Department of Electrical & Computer Engineering

Redacted for Privacy

Dean of 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

Bertrand Boichon, Author

ACKNOWLEDGMENT

First I would like to acknowledge Prof. Wojtek Kolodziej (Department of Electrical and Computer Engineering), Prof. Byoung-Chul Ahn (Yeungnam University), Prof. Frank Chaplen (Department of Bioresource Engineering), Prof. Philip McFadden (Department of Microbology). I also would like to acknowledge DARPA (Defense Advanced Research Project Agency), NSF (National Science Foundation) and the Catalyst Foundation that help us make this project possible.

Finally I would like to thank the entire Cytosensor Research Team including the graduate students in Electrical and Computer Engineering Voranon Kiettrisalpipop, Jiseok Liew, Nicolas Roussel and Angela Teng.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

CYTOSENSOR: AN APPLICATION FOR DISTRIBUTED BIO-SENSOR NETWORKS

1. INTRODUCTION

Environmental protection is a major concern for both developed and developing countries. Among the high priority tasks is the protection against various hazardous toxins of biological or chemical origin. Thus the availability of a reliable, accurate and versatile sensor which can detect various toxins is of big interest. The goal could be, for instance, to detect poisoning in food borne diseases brought by bacteria. Indeed some bacteria such as staphylococcus or salmonella can release dangerous toxins. The Departments of Microbiology, Biochemistry and BioPhysics at Oregon State University have discovered that living cells are able to signal the presence of a broad variety of toxins. An interdisciplinary project named *CytoSensor* was established to create a portable device to detect such toxins using living cells. I have been involved in this project since January 2000 as a graduate student in Electrical and Computer Engineering Department. My main interest is to participate in the development of a distributed and collaborative network of CytoSensors. My work is technically very challenging and consists of designing software architecture for the CytoSensor, then implementing and integrating the various operations. The design part defines the general guidelines used for the development of the CytoSensor functionalities. The implementation describes the software development of optimized image acquisition schemes integrated with highly specialized hardware. It also describes the research and development of techniques enabling distributed collaboration among the sensors.

2. THE CYTOSENSOR PROJECT

First the motivations and the goals of the project are introduced. After defining what sensors are and what expert domains it involves, the objectives are exposed, along with the computer engineering ones. The specifications pertaining to the project are defined and some related works are discussed.

2.1. Motivation

Today most of the systems for detecting biological threats or dangerous chemical products are relatively specialized systems focused on a specific toxin or chemical molecule, or the more versatile ones are neither portable nor standalone. Therefore the need for an autonomous, mobile device for the detection of a wide variety of biological or chemical threats is legitimate. Such detector — or suitably referred to as *computational sensory device* — should encompass the acquisition of information, the detection itself followed next by the threat assessment. Ultimately this kind of device could help save both the environment and human lives. Currently no compact computational systems can meet this challenge. At Oregon State University, we are developing a hybrid detection device which combines biological reaction and digital acquisition and processing technology. With the growth of knowledge in microbiology, new sciences are emerging such as biotechnology and nanotechnology. These can provide useful and practical solutions to today's problems concerning interaction with organisms and the environment. The use of biotechnology in a computational device implies that a biological system is used in the data processing operation and interacts with it. Some biological or chemical threats (e.g. toxins, some bacteria) interact naturally with specific biological recognition elements, such as enzymes, antibodies, microorganisms or cells, microorganisms — or even organisms (e.g. a canary). These biological elements may naturally respond

and consequently provide data to the processing device which can filter and present the information in a meaningful and useful manner. The Department of Microbiology of Oregon State University has thus discovered idiosyncratic biological cells, namely chromatophores, part of the scales of exotic fishes such as the *Betta Splendens* shown in Figure 2.1, capable of detecting a broad variety of toxins. These cells containing pigments visually respond to physically close threats. The goal is to exploit these results in order to create a biological *sensor* device.

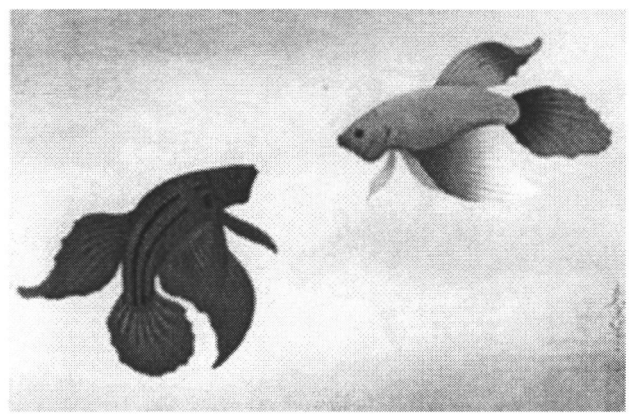

FIGURE 2.1: Exemplary variety of *Betta Splendens* also known as the Siamese fighting fish

2.2. Sensor, biosensor and CytoSensor

A *sensor* is generally defined as a device that can detect a change in a physical quantity such as light, pressure and produce a signal. A *biosensor* is a subset of sensor and is generally defined as follows:

> *"A biosensor can be defined as a compact analytical device incorporating a biological or biologically-derived sensing element either integrated within or*

intimately associated with a physicochemical transducer. The usual aim of a biosensor is to produce either discrete or continuous digital electronic signals which are proportional to a single analyte or a related group of analytes."

We gave the name of *CytoSensor* to the biosensor device in order to emphasize its specialization, i. e. detecting visual changes using living cells when exposed to toxic agents (from the Latin prefix "cyto" meaning cell).

2.3. A multi-disciplinary project

Designing such a sensor requires various scientific and engineering expertise including biology, optics, computer science, fluidics, chemistry, micromachining, etc. The team of professors, students and technicians from various departments are involved in the CytoSensor project. These include the departments of:

- Microbiology
- Pathology and Biology
- Biochemistry
- Chemical Engineering
- Manufacturing Engineering
- Electrical and Computer Engineering
- Mechanical Engineering

The first four of the above areas are concerned with toxins, cells, their interaction and their survival while the last three are involved in the acquisition, storage, sharing and processing of information in a network of *CytoSensors*. The effort to share data in a network of sensors is the object of this work. This project is funded by the Defense

Advanced Research Projects Agency (DARPA) and the National Science Foundation (NSF). DARPA is the central research and development organization for the U.S. Department of Defense (DoD), and its financial participation in the project is 1.9 million dollars. NSF is an independent U.S. government agency responsible for promoting science and engineering projects, and it is financially participating in the project with 1.2 million dollars.

2.4. Computer engineering objectives

The objective of this work is to design an architecture and develop practical tools to acquire, exchange and process data between the sensors. We have to emphasize the fact that the device should be an autonomous system, i.e. capable of processing by itself acquired data. As cells are changing shape and colors based on the presence of toxic agents, the detection is of a visual nature, and an autonomous imaging system is needed. The CytoSensor must acquire a sequence of images, process them in the real-time and finally assess the presence and type of toxins. My work concentrates on the design of the software framework, a suitable network topology of such sensors, the implementation of a robust data acquisition and the integration of diverse processing tasks. The thesis will follow a simple approach, starting with the specifications, a design and an implementation of several prototypes.

2.5. Specifications

First of all, project requirements need to be assessed, as a starting point of any design. Originally, DARPA and NSF are interested in building a more general-purpose biosensor for uses ranging from civilian such as urban pollution (air), environmental threat assessments, to military such as a detection kit for soldiers on a battlefield (bi-

ological and chemical weapons). Certain criteria must be met to create a CytoSensor device:

- It must be compact and lightweight, i.e. portable,
- It integrates the data acquisition and video image processing,
- It yields good performance of numerical calculations,
- It uses a wide variety of data acquisition hardware (frame grabbers and cameras) in order to adapt the hardware to the experimental requirements,
- It has standard I/O connectivity to control additional electrical and mechanical devices,
- It communicates with other sensors or computers,
- It accommodates complex software tools in an easy and flexible way.
- Search for a system with good performance while satisfying the industry standards is a significant task towards ensuring a reliable, modifiable and reusable design.

These conditions are demanding and imply use of multipurpose yet powerful and advanced hardware. Finally, since the image processing requires large memory and processing resources, technological choices (both hardware and software) must be judicious in order to give the optimal results.

2.6. Related work

Advances in processor, memory and communications technology enable the creation of cheaper, smaller sensors capable of communication and significant computation. Many projects are being developed in the areas of Distributed Sensor Networks (DSNs) and Sensor Networks (SNs). The literature on distributed detection is quite extensive, including the topic of multi-sensor data fusion as explained in [1] and [2].

Decentralized network development becomes increasingly popular and many research groups and companies work on generic peer-to-peer frameworks such as the .NET from Microsoft, the Intel Peer-to-Peer Accelerator Kit, Adaptinet or the JXTA virtual network from Sun Microsystems. Collaboration networks also exist: for instance, the SETI@home [3] project are examples of distributed applications on the Internet.

3. DESIGN OF THE CYTOSENSOR

Based on the specifications and needs discussed previously, a conceptual description and architecture of the different system components is explained. In the first part, we focus on describing the different modes and required operations. Next a software architecture is described thoroughly.

3.1. Sensor configurations

3.1.1. Offline vs online usage

The CytoSensor network is to be self-configuring, scalable, and robust in order to adjust to changing topologies. A sensor is not necessarily always connected to its network. If the sensor intends to perform its tasks disconnected from the rest of the network, it must provide suitable resources. For instance, in the case of data acquisition, the offline sensor will manage local data storages; for data processing, enough memory and processor power are required. The online usage will take advantage of the distributed network resources, but at the same time it must cope with possible and unpredictable network failures.

3.1.2. Field vs laboratory usage

Although the goal of the CytoSensor network is to provide various kinds of environmental monitoring, the sensor is presently only used in various laboratories across the campus. This serves the dual purpose of gathering crucial experimental data along with for various toxic agents name and concentration, and testing the sensor in a real-world environment. The continuing experiments allow to build a knowledge base for vari-

ous chemical and biological agents, and enhance the software capabilities for acquiring, processing and presenting the results.

3.1.3. Ad hoc vs infrastructure usage

The third type of sensor configuration describes how the online sensors are interconnected. Based on the existing Internet routing substrate, the network topology is reduced to directly interconnect all the sensors at the application layer of the OSI (Open System Interconnection) protocol stack model. The CytoSensor network therefore acts as an *overlay* network where any sensor can directly connect to any other one using the underlying routers and switches. On the other hand, an *ad hoc* network would interconnect the sensors using other sensors as the routing nodes. It is usually used in an environment where a network infrastructure is not available, although both configurations can operate jointly.

3.2. Assumptions

For the CytoSensor network, the infrastructure configuration will be used because the design and the implementation are simplified. The subsequent design will therefore assume the use of a routing substrate such as the *Internet Protocol* (IP), and the physical interconnection of all the sensors. In the CytoSensor overlay network, a neighbor sensor is defined as a node separated by exactly one *logical* network hop. In this case, any sensor can therefore be seen as a potential neighbor sensor.

3.3. Use cases

Use cases are very useful tools when it comes to develop and plan a complex software system involving many different scenarios. According to the Unified Modeling

Language (UML) standard [4], a use case is defined as a set of scenarios tied together by a common user goal. The following use cases are the most important ones.

3.3.1. All-in-one sensor

All the basic operations are embedded in one sensor. The primary role of the CytoSensor is to acquire and digitize raw data such as 2-dimensional still images. The data are then stored on a permanent storage device to provide the operator with the options of visualization (replay) or processing. The information has to be processed at several stages before comparison to known cases in the CytoSensor knowledge base is made. The decision-making based on the toxin detection outcome is the last stage of CytoSensor operation.

1. As part of the first scenario, the operator starts a data acquisition session, for instance during a laboratory experiment. The images are treated as the frames, part of a sequence or "movie". The image sequence and its metadata, such as a time stamp, the operator name, a hash key uniquely identifying the sequence, are stored on a local mass storage device, e.g. a hard disk. The operator can ask either to process and detect in the real time, or just to monitor the acquired sequence. The operator ends the session manually by using a command or automatically by setting a duration or size limit on the acquired sequence.

2. For the second scenario, the operator wants to view a previously acquired data sequence. The latter can be processed at the same time to yield the final results of detection. Intermediate data are computed at different stages, and they can be stored on the media storage device along with their metadata (i.e. the corresponding sequence ID, time stamp, algorithms used) for a post-analysis.

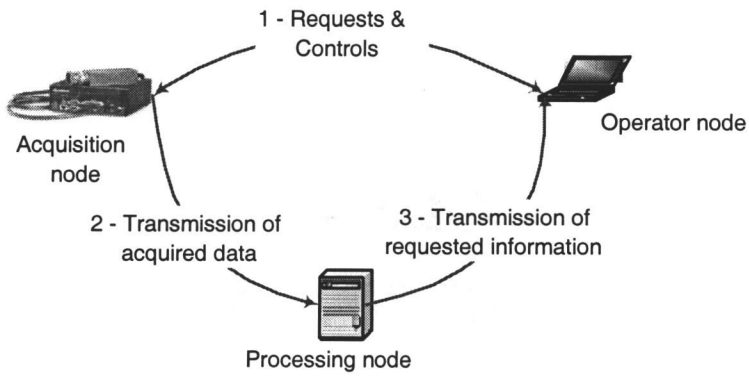

FIGURE 3.1: Collaboration scenario involving three interconnected nodes

3.3.2. Three interconnected nodes

Three nodes are interconnected on a wired or wireless network and each node performs a specific task. The input node acquires data samples in real time. In the case of the CytoSensor the samples are two-dimensional color images. The raw data are then compressed in a lossy or lossless way depending on the next stage. The processing node gets the raw data, transforms it and extracts information by using image segmentations and data modeling algorithms. The control or operator node overlooks the global operation by suitably requesting and synchronizing functions between the data acquisition and processing nodes.

Figure 3.1 shows a simple scenario describing the collaboration among the three nodes for the detection of an agent. First the control node sends a request to the data acquisition node to find out which input channels are available. The control node then asks to grab data samples and ship them to the *best available* processing node. A default sampling rate is applied although the operator may provide custom settings for each. During the acquisition, the data is transferred to the processing node for analysis using computer-intensive resources. The results are finally sent back to the control node for visualization, interpretation and/or decision-making. The operator ends, pauses and

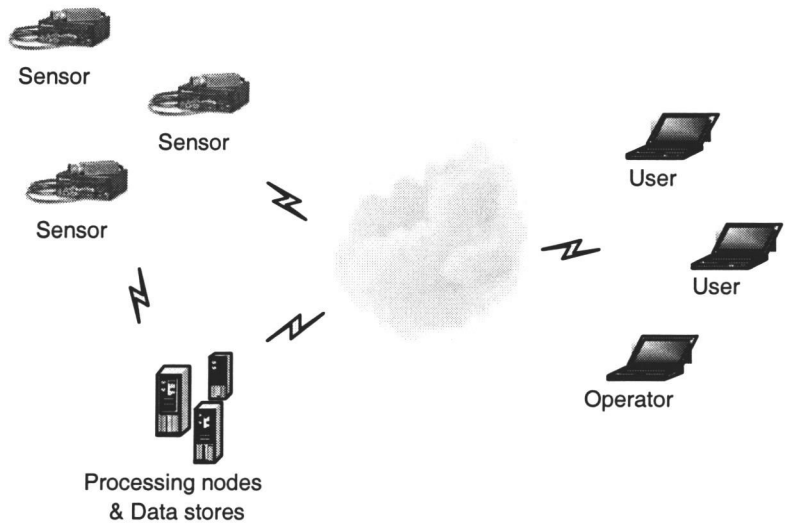

FIGURE 3.2: Distributed resources and collaboration

resumes the acquisition and/or the processing by sending proper requests. This scenario is typical in the usage of the CytoSensor network and it provides an insight of the corresponding simple and streamlined architecture.

3.3.3. A network of collaborating sensors

Operators and passive viewers remotely access resources such as sensing input devices, data processing workstations and data repository systems. As shown in Figure 3.2, these resources are combined to acquire, store and process information. A user can request to acquire live data samples using deployed portable sensors, transfer them to a nearby processing facility where data is both organized and processed. Results can then be provided on demand or broadcasted to a set of users.

For the scientific usage, sensors can be deployed in various restricted laboratories. Using the existing network framework such as a local area network (LAN), the portable sensors are used simply as data collectors or *sources*. Intermediate workstations for

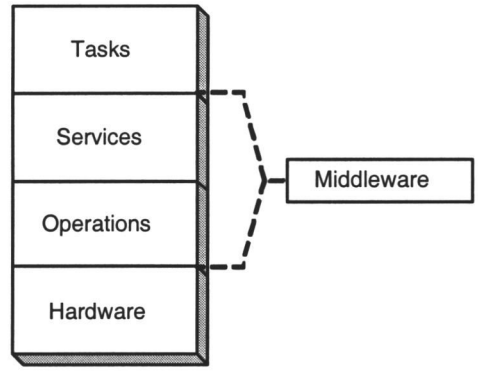

FIGURE 3.3: General four-layer stack model for the CytoSensor

data archiving and processing act both as *data sinks* for the input sensors and *data sources* for the users and operators. For the field usage, the network does not rely on an existing framework, therefore the input sensors are conFigured in ad hoc mode where they operate both as inputs and routers for the transmission of messages and data.

3.4. A top-down layered design

In order to overcome the relative complexity of implementing the CytoSensor network, a simple *top-down* design approach is used. The design is defined and described by a four-layer stack model shown in Figure 3.3.

The task layer: The layer offers the functionalities to control, process and display information requested by the user. It provides network-transparent tasks including data acquisition, data processing, archiving and viewing. Each of these tasks uses several services from the lower layer.

The service layer: A service is defined as a set of operations provided by a local sensor node that are offered to other local or remote services. Services are offered by a sensor to perform requested tasks. Not all the services are present in a sensor

node. Instead, a node can specialize into data acquisition, processing or archiving and therefore the node needs only the corresponding service. However, because the sensor network needs to store, monitor and transfer local or remote data, a mandatory *system* service and a *communication* service implement core functionalities necessary to perform distributed tasks.

The operation layer: Operations are the most basic functions used to build services. Like the services, operations are part of the *middleware* which provides a common set of APIs (Application Programming Interfaces) to access local or remote resources. For instance, the data acquisition service is composed of the digitizer operation and the device control operation. Several operations are provided for the data acquisition, data processing, archiving and networking. In order to provide platform-independent service and task layers, this layer is defined as a hardware abstraction layer. As such, the layer is often an extension to the existing abstraction layer of the operating system (e.g. Microsoft Windows NT).

The hardware layer: The hardware or physical layer defines functions and libraries directly controling various hardware devices. Such devices include digitizers, cameras, USB control devices, mass storage devices, network interface cards (NIC). This layer will be described in detail in section 4.2.1..

3.5. Sensor tasks

A *task* defines an orderly arrangement of services in order to respond to a request. It corresponds to a high-level functionality offered to the users and the operators of the sensor network. The tasks are using services including data acquisition, data processing, archiving, data transfers and presentation. As shown in Figure 3.4, these services are logically connected in the network and they are the basic elements used to perform the intended *distributed* task.

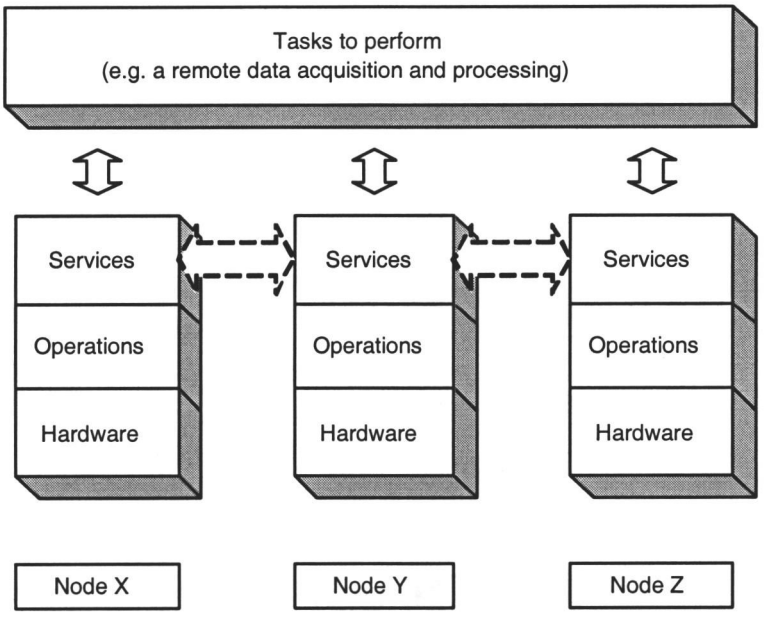

FIGURE 3.4: Logical connectivity across the CytoSensor network

The tasks can be classified into four categories:

Data acquisition task defines the capability of acquiring, digitizing and archiving raw input data samples. For the CytoSensor system, two-dimensional still color images serve as the input data samples acquired at a constant rate. The mode also includes the control of additional peripheral devices used to help and enhance the process of acquisition: their usage is tightly synchronized with the processes of digitizing and archiving. For instance, the CytoSensor must control the light intensity level in order to acquire "good" quality color images. The image quality depends directly on the processing task requirements.

Data processing task consists of several successive processing stages. In the case of CytoSensor, it includes image segmentations, data modeling and decision-making. Intermediate and final results can be stored and archived to improve the access of useful information to the users and operators.

Archiving task ensures that acquired data samples, processed data and information related to these data, also called *metadata*, are physically saved on a media storage such as the hard disks. Archiving enables the access to the gathered information, even in the event of certain faults.

Viewing task: User interfaces and presentation defines this task. User interfaces correspond to the last layer in the system and they are used to present to the user the requested information, either in the form of directly acquired data or processed data. These interfaces provide a user-level access to the available operations. Therefore this task translates difficult to interpret data and commands into comprehensible information for the user.

The tasks can run concurrently on either local or remote nodes. They hide the complexity of the underlying network of sensors. For instance, the data acquisition can potentially perform the acquisition on one node and transfer the acquired data to another node for processing. Both core local tasks of acquisition and processing then have to agree on whether data are pulled from the *source* node or pushed onto the *sink* node. The mechanism of local or remote task synchronization implies the use of *interprocess communication* (IPC) such as messages or events. The choice of the IPC for the implementation is discussed in section 4.2.3.4..

3.6. Sensor services and operations

3.6.1. A modular design

Decoupling the functions of data acquisition, data processing and archiving into different physical locations of the network has significant advantages. Among these are meeting the need for remote sensing capabilities in dangerous environments, portability, mobility and fault-resilience. Also the interdependencies across operations are minimized and *loosely-coupled* nodes are created. Grouping the functions into *services* helps to

define the sensor network as a *network of services*. It also helps to increase the flexibility of software development and deployment. Modular distributed services prevent certain faults from occuring and allow the creation of custom-made sensors.

A service can produce information for another local or remote service in the network. It can give access or control to a local resource such as a digitizer. A service can therefore be defined as a *data source* or *resource provider*. On the other hand, a service can "consume" information from another service and can control a resource through its corresponding service provider. A service can also be defined as a *data sink* or *resource requester*.

Consequently, services are the active elements of a producer-consumer model where data are created, exchanged and transformed by nodes and users. As the CytoSensor network is *data-centric*, each service has a role in producing, sharing, archiving and transforming the data. Four services are designed to fulfill these tasks: the system service, the data acquisition service, the data processing service and the networking service.

3.6.2. Fault-tolerance

The CytoSensor network is designed to provide few fault-tolerant characteristics, meaning that the network and the sensor nodes can still provide their services even in the presence of certain failures. Applied to distributed systems, fault-tolerance can be classified into several types [5], described in Table 3.1. The crash, omission, incorrect computation and Byzantine types of failure can also apply to the data acquisition or processing systems.

According to [5], the fault-tolerance of a distributed system is related to the concept of *dependability* which includes availability, reliability, safety and maintainability. For the CytoSensor project, the two most important criteria we consider are the *availability* and the *reliability*. Since the nodes in the CytoSensor network are *loosely coupled* due to its

Type of failure	*Description*
Crash failure	A node halts, but is working correctly until it halts
Omission failure	A node fails to respond to incoming requests
- *Receive omission*	A node fails to receive incoming messages
- *Send omission*	A node fails to send messages
Timing failure	A node's response lies outside the specified time interval (timeout)
Incorrect computation failure	A node's response is incorrect
- *Value failure*	The value of the response is wrong
- *State transition failure*	A node deviates from the correct flow of control
Arbitrary or Byzantine failure	A node may produce arbitrary responses at arbitrary times

TABLE 3.1: Types of failure in distributed systems

modular design, sensors can be unavailable because of a disconnection from the network for instance. In a data-centric network such as the CytoSensor network, acquired data and processed data should remain available to other nodes. The generic technique for handling such failures is *redundancy*, and in the case of data availability in a distributed system, the technique uses *data replication.*

3.6.3. The system service

The system service is the *core* service that provides basic operations necessary for other local or remote services such as the data acquisition or processing. A remote service can take control of the local acquisition and/or local processing if the latter services are present. A remote service can also query a node's system in order to know what data is present locally. This service is a complementary and necessary element linking the more

specialized services of data acquisition, processing and communication. Figure 3.5 shows the system service can monitor its local resources such as CPU, memory, storage space and can also archive information for the CytoSensor system.

3.6.3.1. The system monitoring operation

In order to manage a node's resources such as the CPU usage, the memory usage and the mass storage usage, the real-time monitoring of these resources is critical. Generally, the underlying operating system on which the CytoSensor software runs, provides needed information. The system chooses the "best" available nodes based on these performance criteria, to distribute the workload in the case of a decentralized collaboration. Moving or replicating data from one node to another node implies the knowledge of the amount of available storage space. In the case of the data processing, the CPU and memory usage are also checked to enable the quickest processing among the available nodes.

3.6.3.2. The archiving operation

The data management is the most fundamental and shared element of a sensor's system since every service manipulates data or metadata. Four types of data are stored in one or several data stores:

- The first type is the acquired data samples or any intermediate processed data,
- The second type is the metadata directly related to the first type,
- The third type is the detection knowledge base used by the processing service,
- The fourth type is the sensor network metrics (latency, throughput) provided by the communication service.

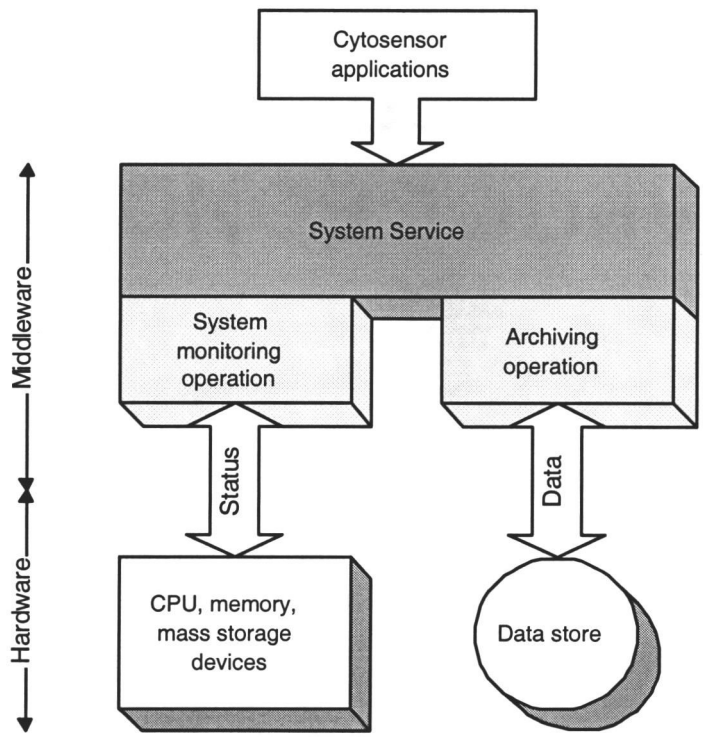

FIGURE 3.5: The system service

These data stores physically correspond to local mass storage devices such as hard disks or tapes. A data store can be built on a logical layer such such as a database system. The implementation of the data management is discussed in section 4.4.1..

Data samples or intermediate processed data can be produced locally by the corresponding services. They can also be received from a distant node and stored locally as replicated data. Whenever acquired or processed data moves from one node to another, their metadata move accordingly. The metadata provides additional information used by operators to query and retrieve the corresponding data. Metadata may consist of timestamps, the sensor identification, the operator identification, the type of cells, etc.

Built-in operations such as incremental, partial and total data backup can be scheduled or manually triggered by a user. The backup consistently replicates data to either a separate local mass storage device or a remote node if the source node knows any online neighboring nodes (see section 3.6.6.4. for the resource discovery operation). Distributed data backup can be seen as data sharing and it can not only improve data availability in the network but it can also leverage performance. Data can be acquired on one node, then backed up onto a second node providing both archiving and data processing capabilities. Data can then be processed later and faster.

3.6.4. The data acquisition service

The data acquisition manages and synchronizes the various input and control devices connected locally. As shown in Figure 3.6, two operations compose the acquisition:

- The control of the digitizers and input channel devices, such as the frame grabbers and the video cameras,
- The control of optional devices related to the acquisition such as the light intensity control boards and the bar code readers to help track the experiments.

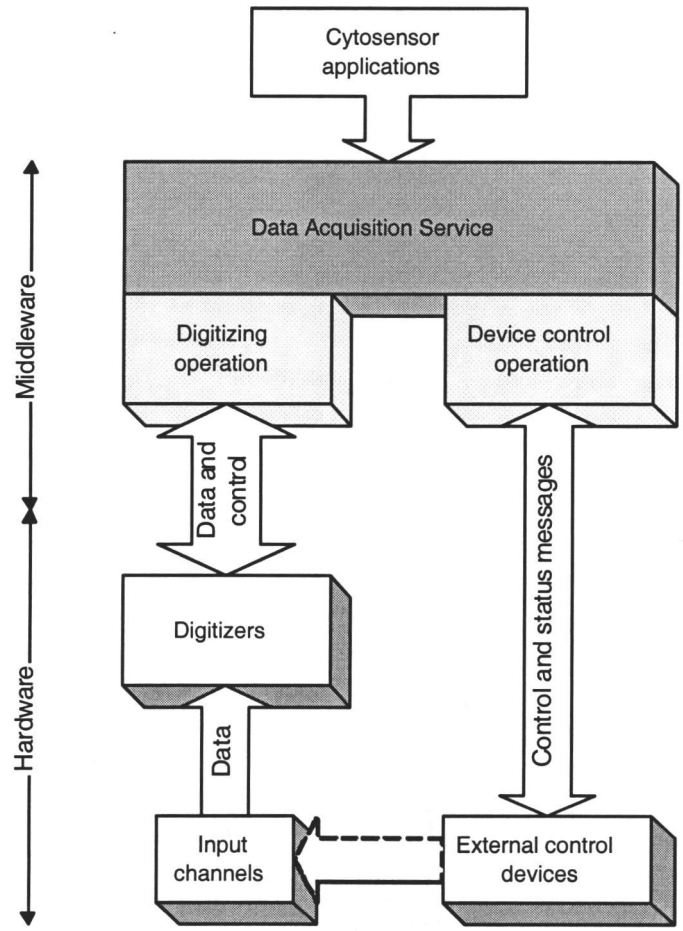

FIGURE 3.6: The data acquisition service

The data acquisition combines the aforementioned operations to provide a complex set of functionalities including:

- The detection and the initialization of the existing digitizers and input channel devices,
- The initialization of the optional control devices: these devices provide an additional degree of control of the quality of the acquired data samples,
- The synchronization of the data capture process using the active input channels, with the control devices (e.g. light on/off control),
- The archiving of the acquired data and metadata on permanent data storage devices: it may use also compression schemes to minimize the impact on data storage and on data transfers between the nodes,
- The detection of failures when an input device is damaged or disconnected.

3.6.5. The data processing service

The data processing service is defined by several computation stages or operations shown in Figure 3.7. The three successive stages of data pre-processing, data modeling and decision-making [6] are generic stages but their implementation is data and application dependent.

For the CytoSensor system where acquired data samples are two-dimensional images, complex image segmentation algorithms are applied before the data modeling. Data modeling uses previous experiments to "recognize" known situations of detection in order to determine the nature and concentration of an agent. The data modeling uses the information stored in a local knowledge base to enable the final stage of decision-making. Various output data at the various stages can be stored and transferred to a node if necessary.

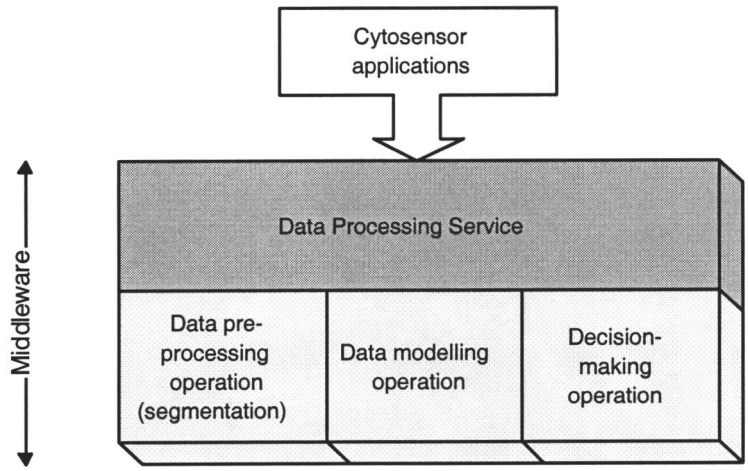

FIGURE 3.7: The data processing service

Finally, the data processing can compute additional corrective parameters to enhance the quality of the acquired data samples. In the case of the CytoSensor project, a feedback control of the light intensity is developed to improve the quality of the color image samples. It in turn enhances the data processing and it may improve the results of the decision-making.

3.6.6. The communication service

The *communication* or *networking* service provides core functionalities for the other services to interact through communication channels. If the sensor node is online, this service is mandatory. The service lays ground to two communication schemes to enable data sharing and collaboration among nodes in the network. The two communication schemes define the operations of this service i.e. the messaging operation and the data transfer operation shown in Figure 3.8.

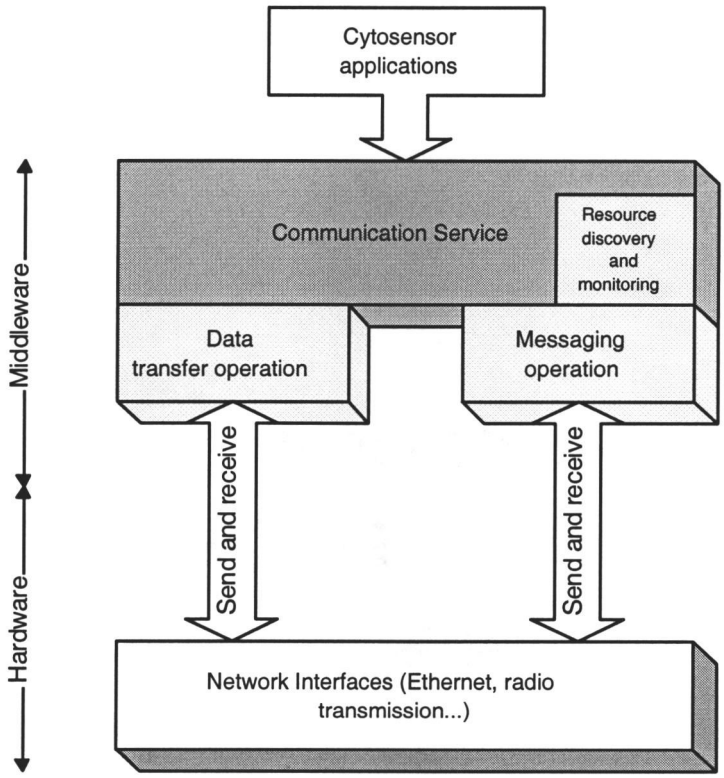

FIGURE 3.8: The communication service

3.6.6.1. The messaging operation

Using the message passing scheme, a service can be accessed or controlled from a distant node. The message passing communication model provides a highly flexible communication feature. Unfortunately, the handling of details such as data representation, interoperability and failures makes the development of distributed applications difficult.

A standard mechanism called *Remote Procedure Call* (RPC) hides this complexity [7]. Originally used as a mechanism for control and data transfers among processes in a local computer, procedure calls are extended for communication across a network. The RPC mechanism is based on the well understood mechanism of local procedure calls. When a procedure is invoked remotely, the calling process (client) is suspended and parameters are passed to the remote process (server) where the procedure is executed. On completion of the procedure execution, the results are sent back to the client which then resumes execution. Several implementations of RPC are discussed in section 4.2.3.4.. RPC is often based on a reliable transport protocol such as the connection-oriented *Transmission Control Protocol* (TCP), part of the standard TCP/IP suite.

3.6.6.2. The data transfer operation

The second communication scheme used in our distributed collaborative application is the transfer of important collections of acquired or processed data. While messages enable the control of operations and tasks, an efficient and reliable protocol such as TCP is used to transfer large amount of data necessary for these tasks. More powerful and dedicated protocols exist but TCP is the most widely accepted protocol and its behavior is well-known. For the CytoSensor project, each experiment run typically generates 300 to 1000 color image samples or *frames*. Each frame has a size of 900kB, providing that a frame is an uncompressed RGB color image of 640 pixels by 480 pixels. Therefore, an experiment may collect 0.5GB of uncompressed data on the average. If an experiment is to

be processed in a remote node, the sequence of frames must be transferred to this node. Considering the large amount of data an experiment produces, efficient data transfers may require on high-bandwidth networks and use lossy image compression schemes such as JPEG or lossless image compression schemes such as PNG, applied to the frames.

3.6.6.3. Peer-to-peer models

In the CytoSensor network, sensor nodes are usually geographically scattered to collect data before processing them. The sensor network is therefore built around a distributed data collection architecture. Collected data could then be stored and processed by a central server. This scheme implies the use of a standard *client/server* architecture where the clients are the acquisition nodes and the server stores and processes the data. This scheme though does not allow remote users to access and operate directly the acquisition node. Moreover, the centralization and the dependency defined in the client/server model causes the server to become potentially a single point of failure. This property is a major drawback for the development of a fault-tolerant distributed application since critical data must remain available as much as possible. The solution to these problems is the use of a second model called *peer-to-peer* (P2P). Peer-to-peer is not a novel concept as it has existed since the Internet was taking form in the 1970s. The greatest strengths of P2P-based models are their decreased dependency on the server, their decentralization of control from servers and an increased network scalability. The sensor nodes can directly establish connections to other nodes and the role of the server is restricted to a bare minimum. The relegation of the server to the background could prevent the situations of bottlenecks, low performance and even *denial of service* (DoS) if the server crashes.

A *pure* peer-to-peer model can be used without relying on any central server. However this model introduces two shortcomings: first, the resource and node discovery tends to be long and difficult, and secondly, the network *authentication* is more complex

to implement. Two hybrid P2P models involve the use of a server that addresses the latter problems. The first hybrid model uses a simple *discovery server* providing the names and network addresses of the online sensor nodes. A node notifies the discovery server by logging in and therefore it enables an easy authentication scheme for the sensor network. In the second hybrid model, a *discovery and lookup server* is used to provide the list of connected peers along with the resources and services available with each of them. The major advantage of the second model over the first model is the reduction of burden on peers, since there is no longer a need to visit each peer personally to look for the required resource or service. Even though such a model depends on a server, it improves considerably the performance of the resource discovery process.

3.6.6.4. Resource discovery and monitoring

Although the RPC mechanism can be used in the discovery and monitoring of network resources, lightweight, unreliable, connectionless protocols such as the *User Datagram Protocol* (UDP) can also be used. Datagrams are messages that are sent to a single destination node (unicast), without obtaining acknowledgement of receipt of the transmission. Therefore UDP packets could be discarded or delayed as explained in [8]. Since monitoring resources is an operation performed on a regular basis, UDP is well-suited for this task.

As discussed in section 3.6.6.3., resource discovery can be greatly improved by the use of a simple lookup server. Yet another resource discovery improvement in a decentralized system is the use of *multicast* protocols. In a packet switch network, multicast is a technique that allows data to be simultaneously transmitted to a selected set of nodes, usually without acknowledgement of reception. IP-based networks support multicast by allowing a network interface to belong to one or more multicast groups: for IPv4, multicast IP addresses range from 224.0.0.0 to 239.255.255.255 (Class D IP addresses); for IPv6, the 128-bit multicast addresses consist of a byte of ones (255) followed by a

4-bit flag (permanent or not), a 4-bit scope (subnet to worldwide) and a 112-bit multicast group identifier. In a multicast-enabled network, *multicast* resource discovery and monitoring can save bandwidth and it can simplify the search of "nearby" resources. Although this mechanism is more fault-resilient than the use of a central lookup server, the lookup system can outperform multicast in certain networks. Therefore a hybrid discovery strategy is designed and represents a trade-off between performance and fault-tolerance. The strategy is defined by three steps used successively if a preceding step fails:

1. A *resource locator table* is locally used to locate a remote resource. This table stores the location of remote services and data in a local cache managed by the system service of a sensor node. The local table keeps track of resources located at the neighboring sensors.

2. If the first step fails, a *discovery and lookup server* is queried by the sensor to find the resource and download the corresponding information into the local service locator table.

3. If the second step fails, a mechanism called *expanding ring* is started to discover and contact unknown sensor nodes. The expanding ring algorithm is carried out using multicast. It increments the time-to-live (TTL) value of the multicast packets at each search iteration until a CytoSensor node is found. The latter node transmits its resource locator table back to the requester node. If no node is discovered after a maximum number of iterations, the expanding ring is stopped and the sensor failed to find the resource.

The expanding ring technique does not announce a resource but rather finds a resource, either the requested remote service or the data. In contrast, the *user directory* algorithm uses expanding-ring search to announce resources; however, it will not be considered for the CytoSensor for simplicity purposes. Moreover, in order to help monitor resources of

the sensors, message updates can be sent to the discovery and lookup server and neighbor nodes using *unicast* messages.

3.6.6.5. Identification

Each sensor node is required to have a permanent and unique ID used to track the node and its data. The metadata, the acquired data and the processed data use the ID to keep track of their origin. If the node changes location in the infrastructure, its network address changes but the ID stays identical.

A data sequence is identified using its source sensor ID and the timestamp (date and time) of the first frame. Each data sample in a sequence is identified by its time offset or its index in the sequence. In order to ensure *data consistency* across the CytoSensor network, a *hash key* could also be computed for each sequence based on a *message digest* (MD) algorithm. Message digests are used to calculate a unique checksum from any kind of data. The calculation is one way i.e. that it can not be reversed. It is theoretically impossible to compute the same fixed-length binary message (or *key*) from two sets of data differing by one single bit. The two most popular message digests are the Secure Hash Algorithm (e.g. SHA-1), Message Digest (e.g. MD5, RIPEMD-160). They not only provide unique identification but also a *validity* check of the data across the network. Message digests are also easy and fast to compute, and they are implemented in most major programming languages and environments. The main weakness is the risk of data and digest forgery, but this problem can be alleviated by applying an encryption scheme.

3.6.6.6. Security

The CytoSensor network not only interconnects sensor nodes but also human operators from the Internet or from a large intranet. In order to ensure privacy and counteract possible forgeries, the nodes have to enable the exchange of encrypted and authenticated

messages as well as encrypted data. Among cryptosystems, there exist two types: the *secret-key* and the *public-key.*

In secret-key cryptography, also referred to as *symmetric* cryptography, the same key is used for both encryption and decryption. The most popular secret-key cryptosystem in use today is known as DES, the Data Encryption Standard.

In public-key or *asymmetric* cryptography, each CytoSensor node has a public key and a private key. The public key is made known while the private key remains secret. Encryption is performed with the public key while decryption is done with the private key. The RSA (Rivest, Shamir, and Adleman) public-key cryptosystem is the most popular form of public-key cryptography.

In a secret-key system, the secret keys must be transmitted either manually or through a communication channel, since the same key is used for encryption at the sender and for decryption at the receiver. A serious concern is that an the secret key may be discovered during transmission. By contrast, the advantage of public-key cryptography is increased security and convenience: private keys never need to be transmitted or revealed to anyone. Another major advantage of public-key systems is they can provide digital signatures that cannot be *repudiated.* Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming the shared secret was compromised by one of the parties sharing the secret. A disadvantage of using public-key cryptography for encryption is speed. There are many secret-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used in combination with secret-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public and secret-key systems in order to get both the security advantages of public-key systems and the speed advantages of secret-key systems. Such a protocol is called a *digital envelope.*

3.7. User interfaces and presentation

The control of the various operations and the presentation of the results [9] are part of the task layer. A task coordinates the distributed services by triggering events. The services fire events to control the flow of actions of a task. These events include manual events triggered by operators of the CytoSensor network. An operator fires events through *user interfaces* to control a task. Several user interfaces are developed to control the data acquisition and processing services as well as the data transfers.

3.8. Component diagram

The services and operations compose the middleware of the CytoSensor infrastructure. In Figure 3.9, the component diagram shows nine major operations divided into four modular services that can be manipulated across the network and through user interfaces.

The operations are the middleware interfaces to the hardware layer. They provide the basic features to the aforementioned services. A short review of the nine main operations is given below.

System monitoring provides metrics on the local system performance such as the CPU, memory and mass storage usage.

Archiving manages data on permanent mass storage devices. It controls the file systems and databases through the underlying operating system.

Data transfers enable the movement of data from a remote sensor to the local system. They download sets of data; if data need to be uploaded, for a backup for instance, a message is sent to the archiving node telling it to download the data. The transmitted data are typically image files for the CytoSensor network.

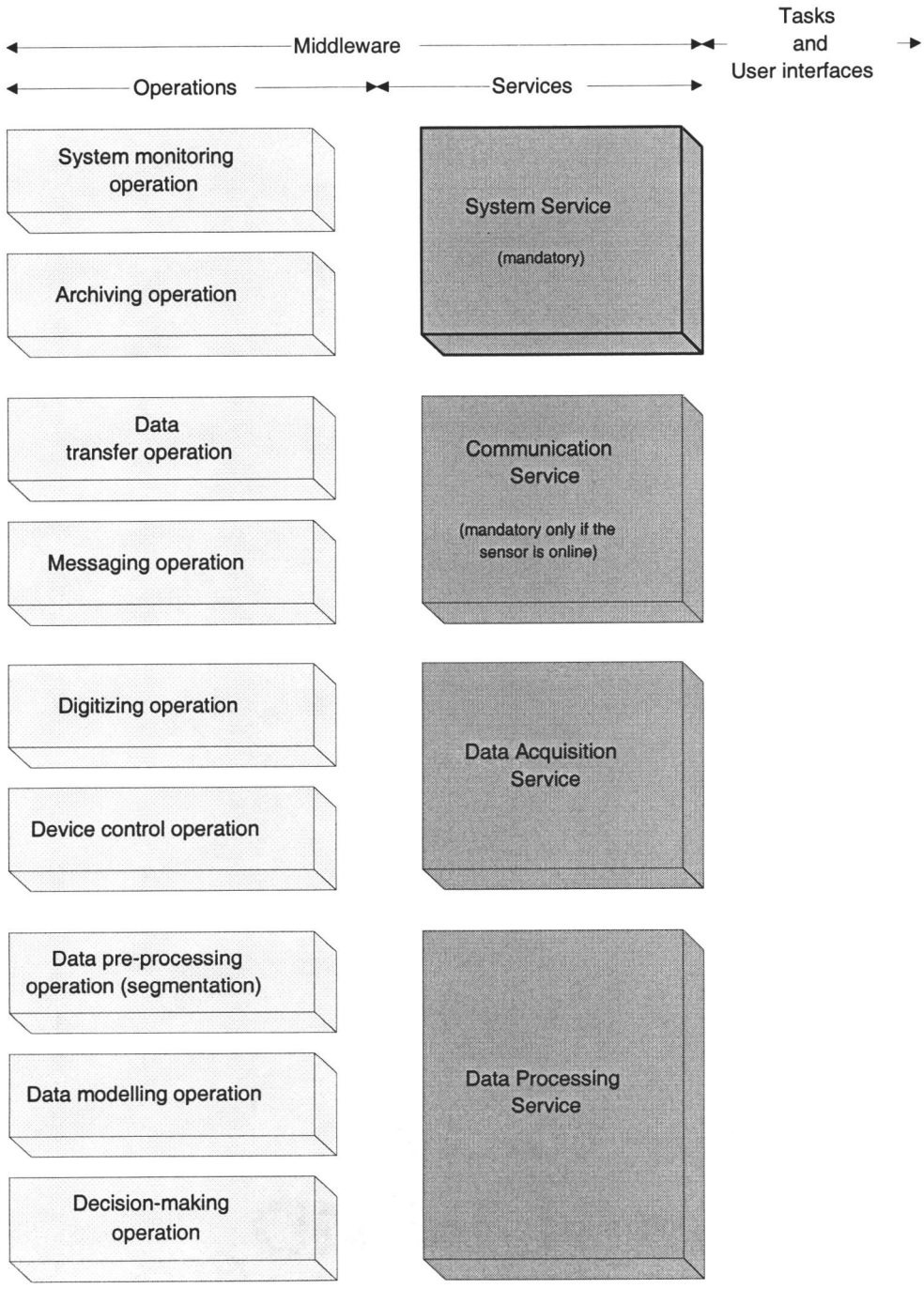

FIGURE 3.9: Component diagram of the CytoSensor services and operations

Messaging is a core operation allowing sensor nodes to communicate. The protocol used needs to be interoperable in order to interconnect sensor services developed in various operating systems and languages.

Digitizing controls the acquisition of data samples at regular intervals. For the CytoSensor project, it initializes and accesses CCD cameras and frame grabbers that digitize two-dimensional color images.

Device control is used and synchronized by the digitizing operation to control and enhance the data acquisition process. For this project, various serial bus devices are developed to control the light intensity necessary for the optical image capture.

Data pre-processing is the first operation of the data processing service. It processes the acquired image samples of a sequence using application-specific segmentation algorithms.

Data modeling is the second operation of the data processing service. It computes a statistical models using time series techniques.

Decision-making is the third and last operation of the data processing service. It matches the current model to existing scenarios stored in a knowledge base, and produces the final detection result.

3.9. Interaction diagrams

The previous design elements of the CytoSensor network have been explained and described as the modules. The following diagrams show the interaction of the services and operations inside a specific sensor node. Diagram 3.10 shows a data acquisition sensor node, whereas diagram 3.11 describes a data processing node and diagram 3.12 shows a data archiving node.

In Figure 3.10, a message is received by the data acquisition node from another node on the network. This message requests the node to start acquiring at a certain sampling rate. The data acquisition service is then notified and starts the task accordingly. The data and metadata are next archived by the system service, and shared at the same time to let the remote sensor download it through the communication service. In Figure 3.11, a data sequence is downloaded from a remote sensor node. While the download operation is in progress, the data processing service starts computations. Intermediary processed data are stored by the local system service. The results are shared with the rest of the network and can be safely downloaded by other nodes. In Figure 3.12, the archiving node is used to store and view the data. A local operator can send a message to a data acquisition node to start data collection. Each data sample is then available for download. The operator can choose to process the data using a third node with a data processing service. The acquired and processed data are then available for download from the third node. The operator can choose the type of information (acquired, processed) to be displayed on the local archiving node.

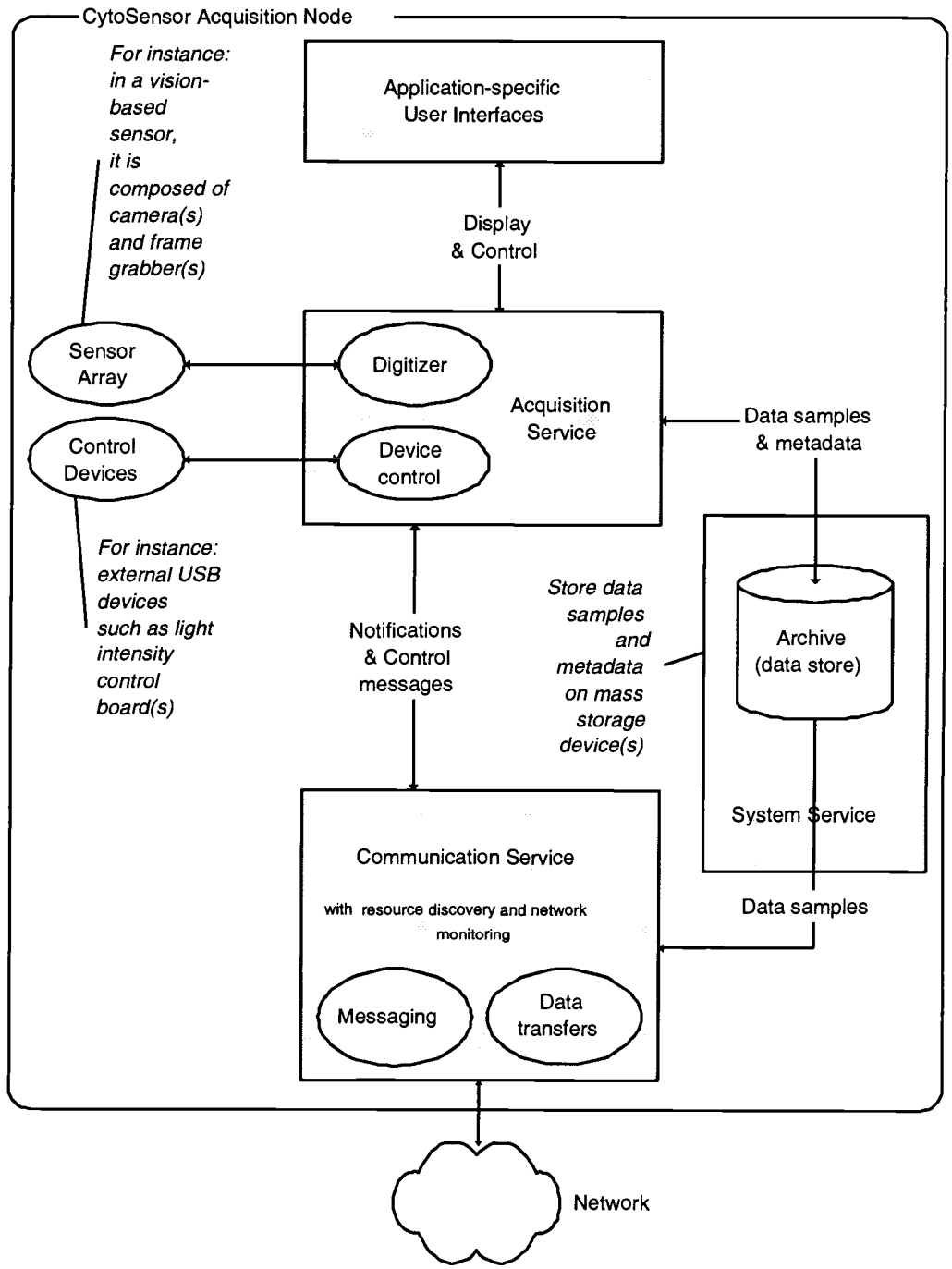

FIGURE 3.10: The acquisition sensor node

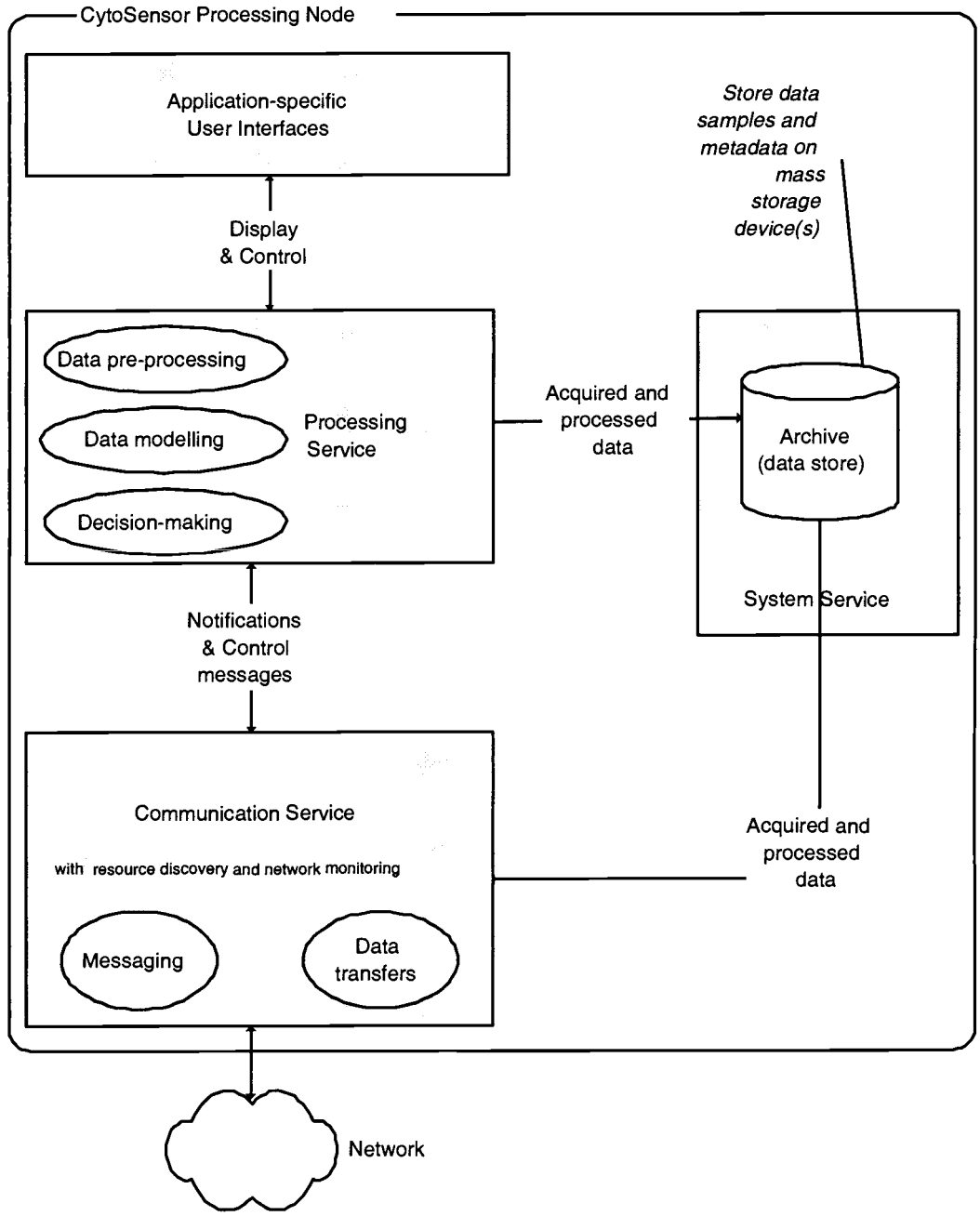

FIGURE 3.11: The processing sensor node

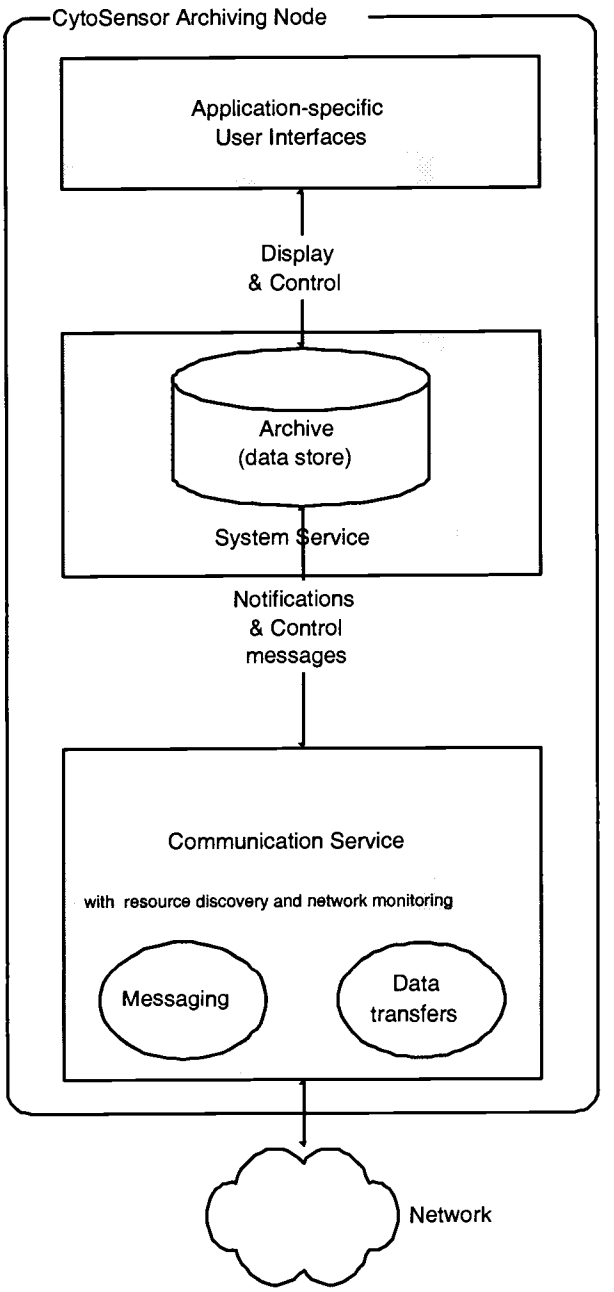

FIGURE 3.12: The archiving sensor node

4. IMPLEMENTATION

Following the specifications and the design guidelines described in the previous sections, an implementation of the CytoSensor is discussed. The implementation starts with the selection of the hardware platform used as a base for the software development. The software development is therefore limited to the hardware capabilities and the goal here is to create a set of *reusable* libraries and tools that can be deployed across platforms and programming languages.

4.1. Hardware integration

In order to stay flexible during the development cycle of the CytoSensor, a general-purpose platform is selected which meets the hardware requirements necessary to implement the services described in section 3.6.. As listed in Table 4.1, these requirements can be met by a modern PC workstation. They include a powerful CPU, large RAM memory, a hard disk, and additional connectivity for communications.

Services	*Hardware requirements*
System	Large hard disks
Communication	Network interfaces including LAN Ethernet and wireless options
Data Processing	Powerful and industry-standard CPU, memory (extendable)
Data Acquisition	- Digitizers: image frame grabbers for image acquisition - Input devices: CCD color cameras - Control devices: external programmable boards

TABLE 4.1: Hardware requirements by service

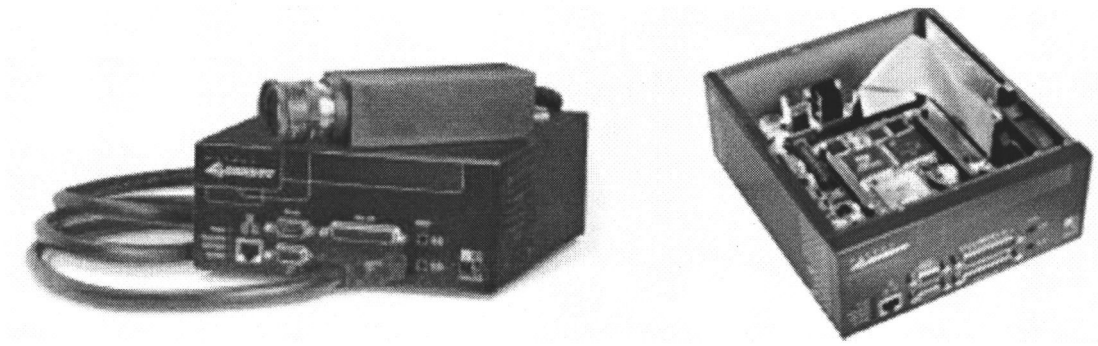

FIGURE 4.1: An embedded stand-alone vision platform: the Matrox 4Sight

4.1.1. The Matrox 4Sight platform

One key element that characterizes the sensor is its *portability.* Therefore the PC machine should have a small footprint and high connectivity to enable its mobility. After careful search and tests, an industrial vision system was selected. The *Matrox 4Sight* combines embedded PC technology and compactness in a rugged enclosure as shown in Figure 4.1. This stand-alone platform offers the same performance as a desktop PC, and integrates image capture, processing, display, networking and general purpose I/Os. In order to easily communicate with the hardware elements, a robust, low-cost and industry-standard operating system called MS Windows NT (version 4 and 5) is used. This operating system provides a common architecture to attach various types of hardware through the Hardware Abstraction Layer (HAL).

The platform provides various powerful frame grabbers to enable standard analog color video acquisition. The Matrox 4Sight integrates various compatible Matrox frame grabber products such as the Meteor, Meteor-II and Orion cards using the small footprint, industry-standard PC/104 form factor. Several different frame grabbers are used for different CytoSensor prototypes to adapt to the changing needs of the data acquisition.

The Matrox computer contains an Intel Pentium class microprocessor, 256MB of RAM, a 6GB IDE hard disk which allows to archive locally up to 15 sequences (or experiment runs) of 400 uncompressed image samples each. Additional connectors such as USB ports, serial ports (RS232, RS485), a parallel port, IEEE 1394 ports (FireWire), display (VGA/TV) and Ethernet (RJ-45) ports are also present on this platform. They bring flexibility and allow evolution in the development process of the software components.

4.1.2. The Euresys frame grabbers

One of the goals of the project is to create a large knowledge base of agent "signatures" used by the detection algorithms. To this end, multiple experiment runs should be carried out simultaneously using the same sensor. A multiple-camera acquisition system is then integrated into a single sensor machine. As the sensor is to be used mainly in microbiology and bioengineering research labs, the machine does not need to be highly portable, so a regular PC can be used. This choice offers more options in terms of selection of a powerful and low-cost multichannel acquisition devices. The Euresys company allows the use of expandable PCI frame grabbers through its Picolo Pro family of products, shown in Figure 4.2. The various types of acquisition hardware also motivate *code reuse* in order to minimize the impact on software development.

4.1.3. Cameras, optics and chamber holders

In order to acquire image samples, small size CCD color cameras are used. They produce high-quality, high-resolution images of the microscopic living cells, which in turn supply the necessary data for processing and detection. In addition, high quality optics are used to magnify the cell image and focus in random areas of the cell holding vial, with the field of view ranging from $(0.5mm)^2$ to $(1.5mm)^2$ depending on the sensor prototype. The vial is a small sealed chamber containing the living fish cells at the bottom. The

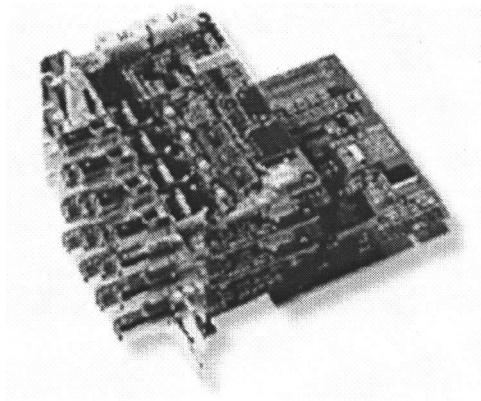

FIGURE 4.2: The expandable multichannel Picolo Pro acquisition card from Euresys

chamber holder in which the vial is placed may be equipped with switches capable of detecting the presence of a vial. This helps the operator and the sensor to minimize experimental errors before or during the process of data acquisition. Figure 4.3 shows the arrangement of the different elements of an input channel unit based on the *Apollo-II* prototype described in section 4.1.5.3..

The cameras used can produce various standard video signals ranging from the NTSC RGB signal for the single channel prototypes to the NTSC Y/C (S-video) for the multichannel prototype.

4.1.4. Control devices

Throughout the CytoSensor prototypes, control devices are developed to carry out specific and important tasks such as light control, pump control, switch change detection and sample ID using a bar code reader. The devices are external boards integrating a programmable microcontroller for which a specific firmware is developed. They are connected to the sensor (Matrox 4Sight platform) using standard USB or serial (RS-232) ports. The devices are controlled by the acquisition service and they enable duplex

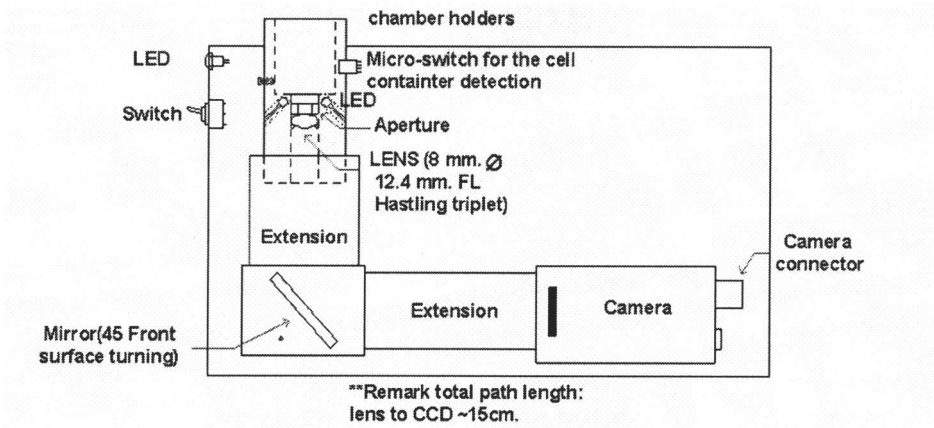

FIGURE 4.3: Typical input channel unit elements based on the Apollo-II prototype

communications in order to access the status of operation. The development involves programming microcontrollers or PIC (Programmable Interrupt Controller) to perform specific tasks such as setting the intensity of light for the vial. The PICDEM USB board from Microchip used in the Apollo-II prototype uses the PIC 16C765 (Figure 4.4).

4.1.5. Prototypes

Since the beginning of the project, several steps in the development of the CytoSensor have been developed and they have resulted in three major successive stages called *Mercury*, *Gemini* and *Apollo*.

4.1.5.1. The Mercury prototypes

The Mercury family of prototypes shown in Figure 4.5 is the early stage of development where various cell chambers are tested and a high magnification is used. An external light is used but it tends to change the image quality from one experiment run to another. Using the Matrox 4Sight platform, the image acquisition library is based

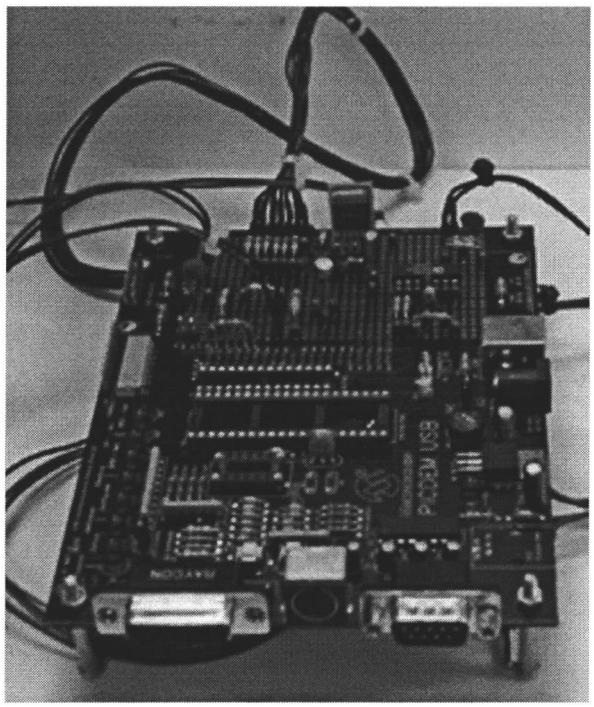

FIGURE 4.4: The PIC 16C765 from Microchip is used on a USB board to control the light of an input channel unit for the Apollo-II prototype

only on the Matrox Imaging Library (MIL). Together with the data processing library, the libraries are integrated into a MS Windows application called CytoSoft. The acquisition library is at this point hardware-dependent, data and metadata are stored locally as uncompressed image files.

4.1.5.2. The Gemini prototypes

The Gemini prototypes (Figure 4.6) capture images from an increased cell area (lower magnification) with a different type of cell chamber. Several configurations of light are tested such as the use of indirect or direct light through the chamber.

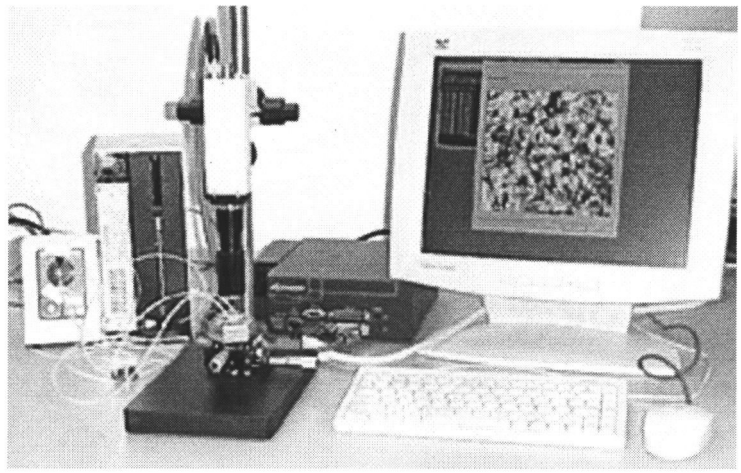

FIGURE 4.5: A Mercury prototype

4.1.5.3. The Apollo prototypes

The Apollo prototypes define the most recent stage in the development of the CytoSensor. The *Apollo-I* prototype uses a single channel to capture high resolution RGB images of living cells located in a sealed vial. As shown in Figure 4.7, the input

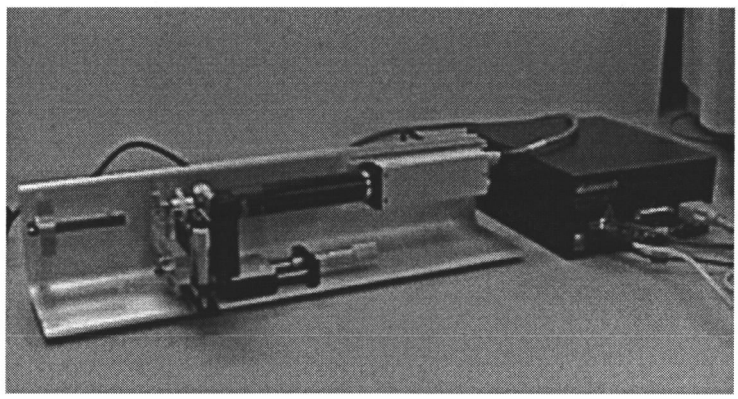

FIGURE 4.6: A Gemini prototype

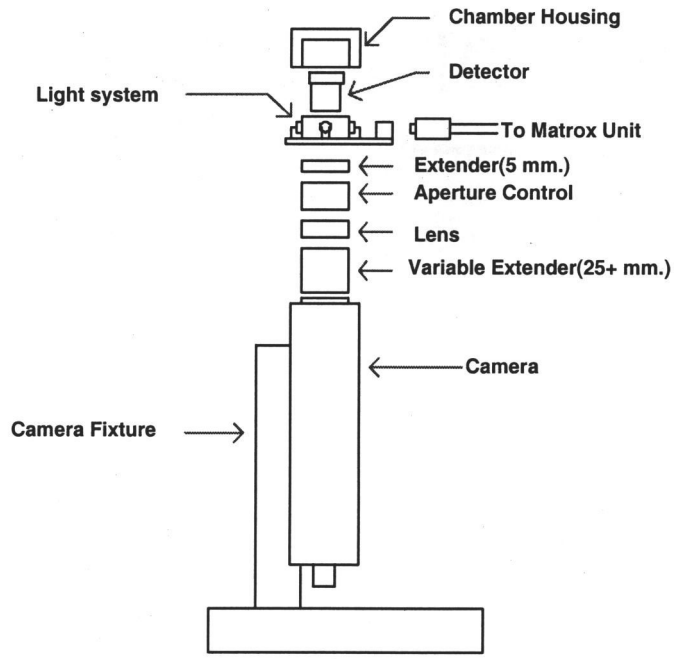

FIGURE 4.7: The input channel unit of the Apollo-I prototype

channel unit is composed of a vertical stand, a Pulnix CCD camera, a set of extenders, lenses and an aperture to control the depth of field.

The Apollo-I prototype also uses a light ring of 6 LEDs controlled by a microcontroller. The PIC allows the operator to switch on and off the lights, change the intensity level and check the status of the LEDs and the current they draw. The complete Apollo-I sensor is shown in Figure 4.8.

The *Apollo-II* prototype is an enhancement of the Apollo-I prototype. It allows the acquisition of up to 12 simultaneous experiment runs and the aim is to help build a knowledge base for the detection of broad variety of agents and concentrations. Figure 4.9 shows the sensor, its touch panel and a USB hub connecting two input channel units.

The Apollo-II sensor uses a Matrox 4Sight platform equipped with a Matrox Orion/Dual frame grabber capable of sequentially acquiring from up to 12 NTSC Y/C (S-video) video sources. As shown in Figure 4.10, each input channel unit is composed

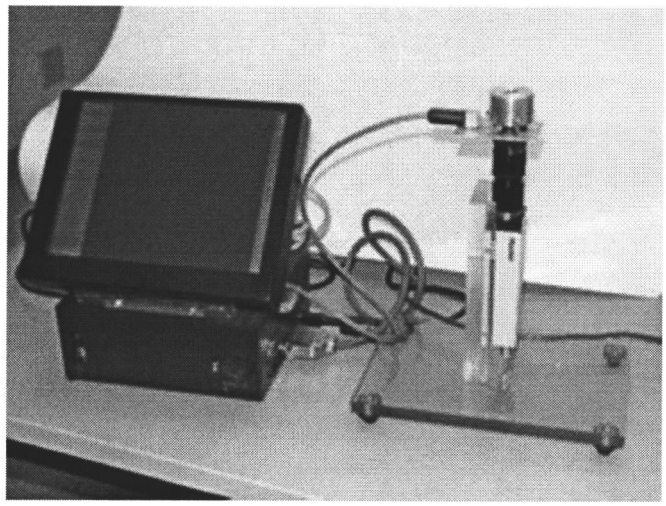

FIGURE 4.8: The complete Apollo-I prototype

of a CCD color camera (640 by 480 pixels), optics, a chamber holder, an LED light ring, switches, an LED status and USB board integrating the Microchip PIC 16C765. The latter controls through an I2C bus the LED intensity of the light ring. It also controls the LED status, it checks various switches such as vial detection switch and it communicates with the Matrox 4Sight platform through its USB interface.

4.2. Software development and integration

4.2.1. Development tools

After the selection of the hardware platform for the development of the sensor prototypes, tools for the software development have to be chosen. As the Matrox 4Sight platform integrates the widely-used and general-purpose operating system MS Windows NT, the development uses the language C/C++ and the Visual Studio (6.0) tools. Both of them allow us to easily produce effective applications and libraries.

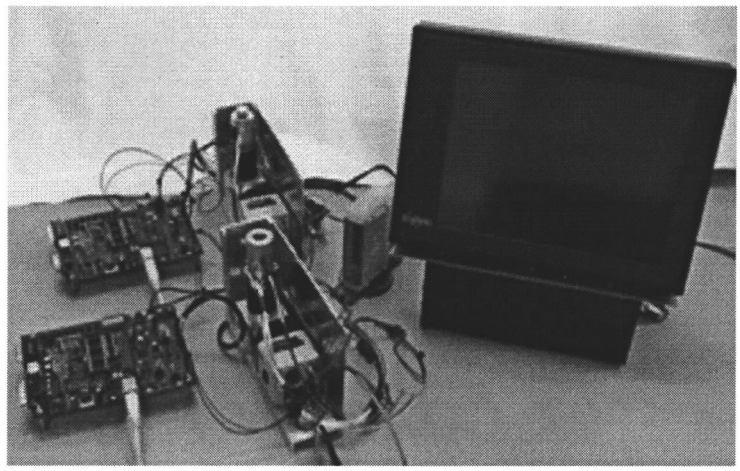

FIGURE 4.9: The Apollo-II prototype with two input channel units

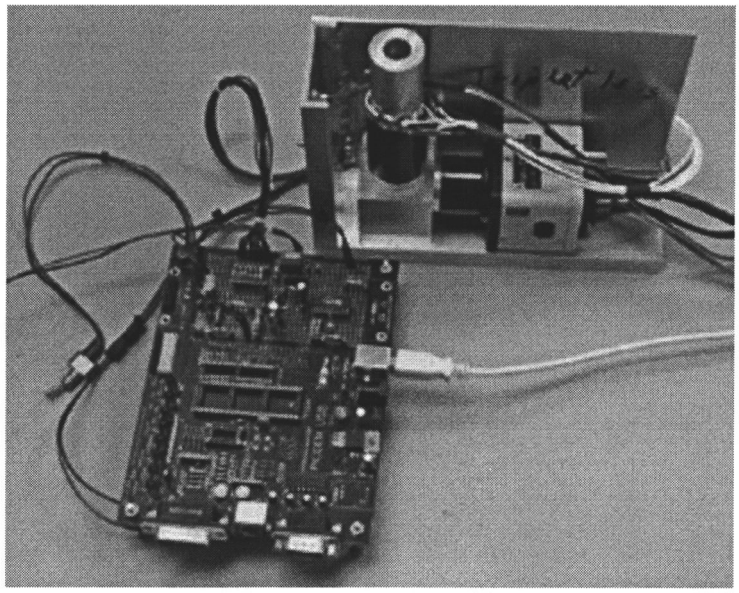

FIGURE 4.10: An Apollo-II input channel consisting of a CCD camera, optics, a chamber holder and a USB board

In order to control the hardware such as the USB control devices, the hard disks or the network card, low-level libraries are provided by the operating system in the form of Application Programming Interfaces (API). In the case of non-generic tasks such as the acquisition from a Matrox or Euresys frame grabber, the manufacturer of the device provides the API to control it. The Matrox frame grabbers use a common library for acquisition, display and buffer management called the Matrox Imaging Library (MIL); and the Euresys products integrate a native library called the MultiCam API. The CytoSensor middleware libraries then use these APIs to develop the functionalities specific to the operation and service. The middleware libraries can therefore be described as the *link* between the hardware and the applications. The goal is to produce a set of reusable code that can be used across sensor platforms and with various hardware. At the same time, since different types of sensors tend to expose the same basic operations and services such as digitizing, archiving and transferring data, the middleware libraries attempt to create a set of reusable code for a *broader* class of sensors. To achieve hardware independence, the middleware libraries have to create a common abstraction layer. While Windows NT provides such a layer for operations such as networking, mass storage access, serial and USB communications, no abstraction exists for the data acquisition from industrial frame grabbers. Some efforts by Microsoft have designed and produced an unfinished Software Development Kit (SDK) called the Vision SDK. It enables the selection, initialization and use of various image sources, and its greatest strength is its extensibility to unknown hardware. The Vision SDK is extended to control the Matrox Meteor, Meteor-II and Orion/Dual cards as well as the Euresys Picolo Pro cards. Finally, the Driver Development Kit (DDK) is used to develop the control device library for the management of USB transactions with the external PIC devices.

4.2.2. Techniques

As described in section 3.6.1., the operations and services have a *modular* structure. The process of the CytoSensor software development is an iterative and incremental development process. It consists of an *elaboration* or design phase and a *construction* or implementation phase. The implementation itself consists of many iterations in which each iteration builds tested and integrated software that satisfies a subset of the requirements of the project. In order to help the maintainability of the CytoSensor project, a set of rules should be followed from the design to the implementation. Therefore a standard object modeling language called the *Unified Modeling Language* (UML) [4] is used. Together with the *Object-Oriented Design* (OOD), it eases the team development of the complex distributed CytoSensor applications.

Finally, a *Concurrent Versions System* (CVS) is deployed to let the developers safely share, update and correct different pieces of the libraries. Integrated with the MS Visual Studio development tool, the MS Visual SourceSafe is a project-oriented version control system that allows to keep track of the various versions of the CytoSensor applications and libraries.

4.2.3. Core technologies

Many different software and hardware technologies are used in the CytoSensor at different levels of the development. These technologies are in general recognized as industry-standard tools to create programs, control hardware devices, exchange information between application processes.

4.2.3.1. Standard Template Library

Operations and services that do not provide User Interfaces (UI) use the Standard Template Library (STL). This library is compliant with the standard C++ and it provides a standard I/O library and a set of generic data structures called containers (vectors, linked lists, queues, stacks...). STL-enabled code can then be reused on other platforms such as Unix and Windows.

4.2.3.2. User Interfaces

The presentation of coherent information is made possible through the use of User Interfaces (UI) specific to the platform of development. For instance, MFC (Microsoft Foundation Classes) are used to build standard Windows interfaces for the operators of the CytoSensor. Platform and language dependent libraries such as MFC should be used only for the development of user interfaces in order to reduce the dependability on specific systems.

4.2.3.3. Information encoding scheme

A common data encoding scheme must be used to consistently encapsulate the data and messages in order to enable interoperability across heterogenous sensors nodes. A widespread information encoding scheme on the Internet is the *eXtensible Markup Language* (XML) which is well understood and implemented across many platforms. XML is a simpler version of the SGML (Standard Generalized Markup Language) and the two basic operations are *parsing* and *building* XML documents. XML can be used to encode messages for a message-passing scheme such as the XML-RPC described in the subsequent section.

4.2.3.4. Interprocess Communications

The services are part of the CytoSensor middleware libraries that are integrated into local processes or running tasks. Since the services are designed to interact between each other and with the hardware, these processes must *communicate* either locally or remotely through communication channels. Three classes of Interprocess Communication (IPC) are used with the CytoSensor.

Communications between the local processes: The first class involves the use of local *synchronization* objects. These objects are usually fast *kernel* objects part of the multitask-enabled operations systems. On Windows NT and Unix, such core objects are mutexes, semaphores, critical sections (very fast) and events. These objects are used for the synchronization between the operations of a service such as the digitizing operation and the control device operation. Each operation and service can use one or several *threads* of execution that can be controlled using the kernel objects. Besides the critical sections, these objects can be used to synchronize processes on a local sensor. Providing separate processes for the user interfaces and the core services can prevent faults from occuring if a local operator misuses an application. Other forms of interprocess communication are the shared memory and the pipes that provide a mechanism to exchange large amount of data between processes. Pipes can be used across local and remote processes, but network sockets are preferred because they can work on various types of network and machine. Shared memory is integrated into the Vision SDK to share acquired images in memory. This allows other local processes such as the data processing service to quickly access the images without using the slower hard disks.

Serial communications with the control devices: Communications are also developed between the control device operation running on the Matrox 4Sight and the control device firmware running on the PIC. The communication channel is the

RS-232 serial bus for the Apollo-I prototype and the USB 1.1 bus for the Apollo-II prototype. Windows NT defines HID (Human Interface Device) classes to access and control USB devices. The firmware communicates with the HID USB driver which in turn communicates with the CytoSensor middleware libraries.

Communications between the remote processes: Two tasks on two distant sensors can exchange information through a wired or wireless communication channel, using *sockets*. The standard Berkeley sockets use the TCP/IP protocols as they have better overall performance than *named pipes*, especially on slower networks. Built on top of the TCP/IP protocols, additional protocols implements the Remote Procedure Call (RPC) scheme discussed in section 3.6.6.1.. The protocols are in general platform-dependent, such as CORBA for Unix, COM+ for Microsoft and Java RMI for Java. Used as a message passing scheme between network sensors on the Internet, two web-based protocols called SOAP (Simple Object Access Protocol) and XML-RPC add more flexibility of use across platforms and languages. The XML-RPC protocol is selected for the development of the messaging operation because the protocol is lightweight and easy to use in applications but it allows the exchange of complex data structures. XML-RPC uses the universal HTTP as the transport protocol and XML as the data encoding scheme.

4.2.4. Architecture

The development of the Mercury and Gemini prototypes lead to the architecture for a single channel data acquisition, archiving and data processing system called Apollo-I (Figure 4.11). The CytoSoft application integrates all the services into a single user interface accessible with a touch panel. The CytoGrab application only acquires and stores images but it is based on a common middleware libraries called CytoCore. The libraries comprise the image digitizer and display, the device control, the archiving man-

agement (data store) and the data processing. Two separate Windows NT services called CytoMonitor and CytoScheduler provide networking functionalities. The former monitors the system resources (e.g. CPU and memory usage) using the performance counters provided by the Windows NT kernel; and it broadcasts this information to a multicast group for the other CytoSensor nodes. This mechanism serves both to announce the resource information on the overlay network and to discover new sensor nodes. But several multicast-enabled routers must be available in order to take advantage of this mechanism, therefore the latter is not used in the subsequent Apollo prototypes. Also a Windows NT service, the CytoScheduler application performs scheduled tasks of transferring the data from the local sensor to a backup server or to any other node. The data can be automatically transferred daily, weekly or monthly. Both CytoMonitor and CytoScheduler can be controlled using a control panel applet.

The Apollo-II architecture in Figure 4.12 shows the latest work where the multichannel data acquisition service libraries, the communication service libraries and the system service libraries are separated from one another. These modular middleware libraries are built on hardware and operating system dependent APIs such as MIL, VFW (Video For Windows) or the file system. The major difference with the previous architecture is the use of the MS Vision SDK for the data acquisition to control multiplexed input cameras, the use of control devices on a single USB bus, the use of XML-RPC and a HTTP server for the communication service, and the use of a MySQL database for the archiving operation.

4.3. Image acquisition and device control libraries

The development of the image acquisition libraries is divided into two categories of prototypes: the single channel image acquisition prototypes and the multichannel image acquisition prototype.

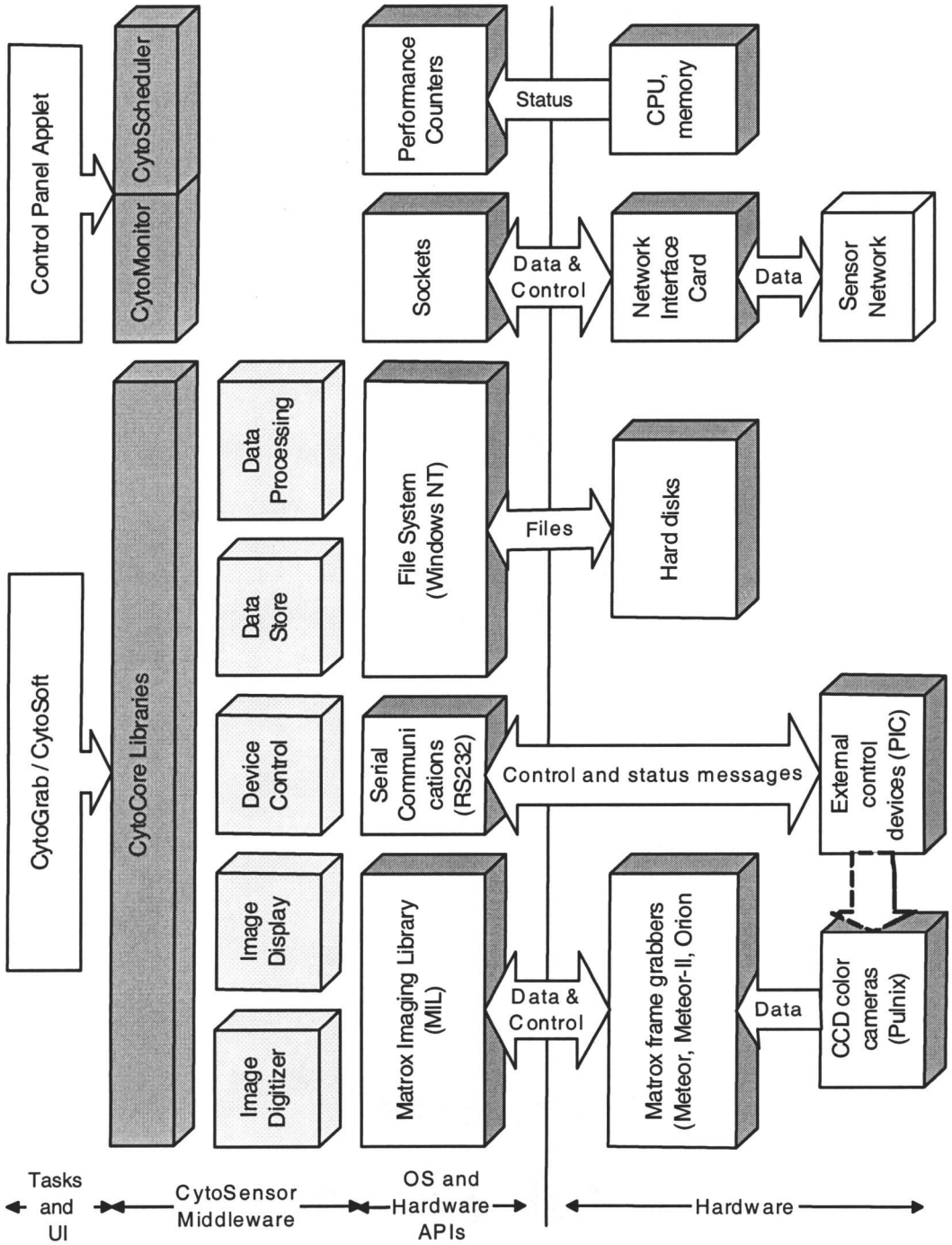

FIGURE 4.11: Software implementation architecture for the Apollo-I sensor prototype. It inherits from the previous Mercury/Gemini prototypes

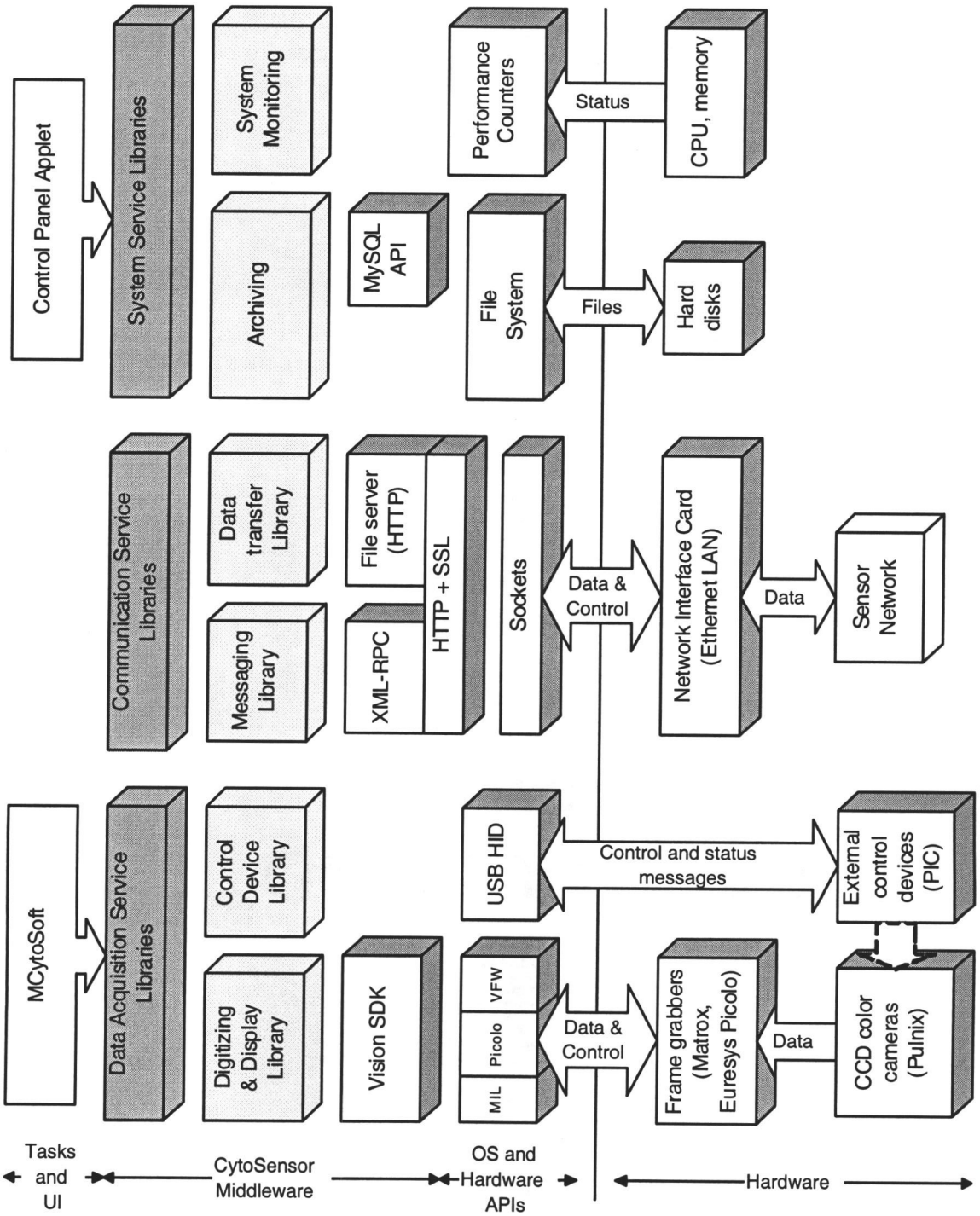

FIGURE 4.12: Software implementation architecture for the Apollo-II sensor prototype (without the data processing service)

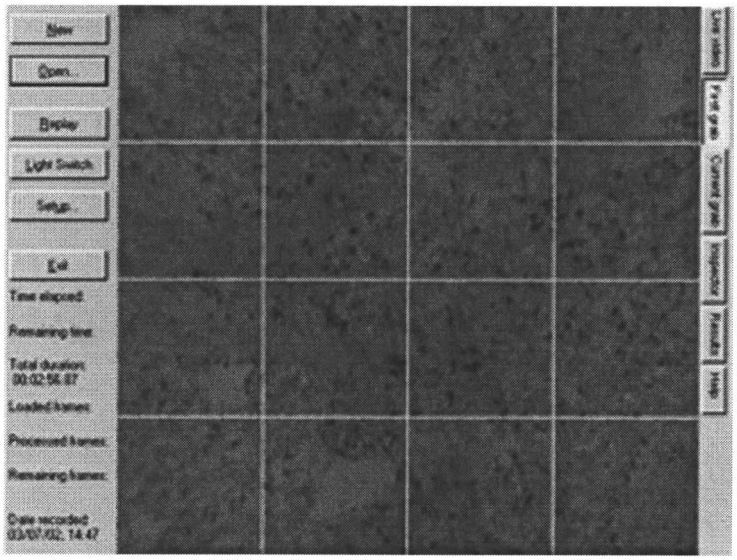

FIGURE 4.13: CytoSoft displays real-time acquired images of microscopic cells

4.3.1. The single channel image acquisition

The development of this important part of the software starts with the Mercury prototype, then the image acquisition is adapted for the Gemini prototype and a mature and stable architecture is used for the Apollo-I prototype. The latter controls a single color high-quality CCD camera plugged to the Matrox 4Sight. A unique control device composed of a PIC controling a LED light ring around the chamber communicates with the image acquisition library through the RS-232 serial bus of the Matrox 4Sight. The intensity level of the LED light is controlled by the software. Flashing low-power LEDs are used in lieu of regular flash lights to reduce the effects of heat on the living cells in the chamber. The LED flashes are synchronized with the image captures taking place at a constant sampling rate chosen by the operator before an experiment starts. The operator controls the processes of data acquisition and processing using CytoSoft, a simple user interface shown in Figure 4.13 and Figure 4.14.

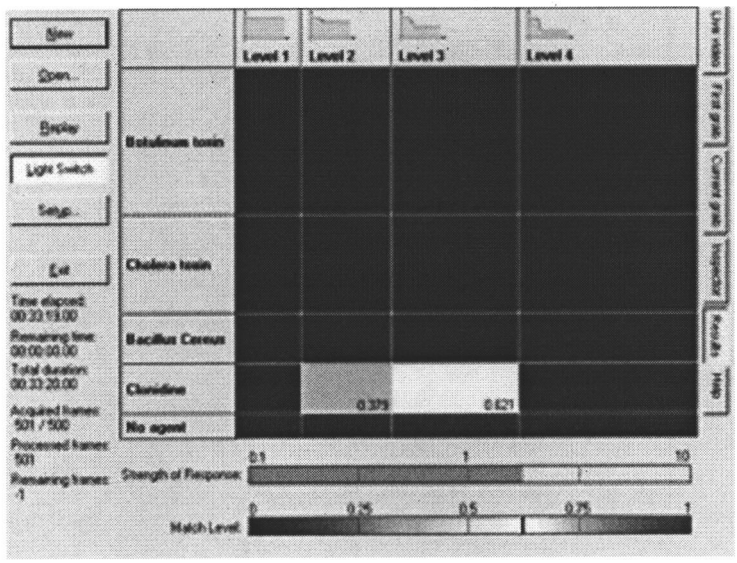

FIGURE 4.14: CytoSoft processes acquired images and visually presents detection results (colorful grid)

The CytoGrab and CytoSoft applications use a separate thread to perform the responsive image capture on a preemptive scheduling operating system such as Windows NT. On Windows NT, the time quantum, defined as an assigned CPU time unit, cannot be less than 10ms for high priority threads. In reality, when the applications capture images every 2 seconds, a timing error not greater than 50ms is introduced. This timing error is tolerable since the cell changes occur gradually and span over several minutes. But a major issue is the time drift that can potentially be introduced at each capture of image samples. To avoid the problem and acquire in the real time, the delay between two image samples is not calculated using the prior image sample's absolute timestamp but rather by using the *first* image sample's absolute timestamp. Therefore the timing error is *bounded* and its impact is minimal.

Finally, the Apollo-I prototype implements the data acquisition operations designed earlier but it lacks a solid device-independent data acquisition in order to use

the applications on different data acquisition platforms. The device independence is attempted with the latest prototype, the Apollo-II.

4.3.2. The multichannel image acquisition

The Apollo-II prototype introduces a major change for the image acquisition platform: multiple cameras acquire images from several simultaneous experiment runs. The platform must operate 12 color video channels. To realize this with a relatively low-cost and compact solution, the video quality is lowered by choosing cameras delivering a NTSC Y/C video signal and a resolution of 640 by 480 pixels. Two frame grabbers are tested before one of them is selected: the Matrox Orion/Dual frame grabber and the Euresys Picolo Pro frame grabber. In order to find the most suitable hardware platform in a short time, the image acquisition operation relies on the MS Vision SDK libraries that allow to enable various image sources such as the Matrox frame grabbers and the Euresys frame grabbers. The MS Vision SDK offers a common and fast substrate to the CytoSensor middleware image acquisition libraries. However the actual version of the Vision SDK is limited to the VFW (Video For Windows) compliant cameras and the Matrox Meteor frame grabber. The Image Source library is therefore *extended* to accept the Matrox Orion/Dual frame grabber and Euresys Picolo Pro frame grabber.

On the Matrox Orion/Dual frame grabber, the input channels are multiplexed. The captures are therefore sequential and the use of multiple threads concurrently accessing their associated channel yield poor performances. Using the thin Vision SDK layer, the CytoSensor image acquisition library uses a single thread to sequentially capture still images from the 12 cameras. Each capture from a single channel takes 40ms and the channel switching adds another 40ms in order to synchronize the digitizer onto the video input signal. Adding the MIL library overhead and the transfer delay of the image into the host (Matrox 4Sight platform) memory, an image capture from one channel takes around 100ms in reality. Therefore with 12 channels, 1.2 seconds is necessary

to grab a round of 12 images. The smallest sampling time interval unit that an operator can choose is 2 seconds for the Apollo-II prototype. For each channel, a custom sampling interval is selected as a multiple of 2 seconds. At each round, the images from the 12 channels are captured and only the images satisfying their sampling rate condition are stored on the hard disk. This process ensures that the time interval between image samples of a channel is respected and the timing errors are minimized even for experiments running for hours.

Each input channel unit is composed of a camera, optics, a chamber holder, an LED light ring around it, a switch to detect the presence of a vial, a manual switch and a USB board. The Microchip PIC 16C765 is integrated onto the USB board to control the light ring and the switches, and it communicates with the Matrox 4Sight platform through the USB bus. The data acquisition library identifies, initializes and accesses each USB device using the standard HID driver provided by the Windows NT. Various commands can be sent from the host to the USB device:

- Switch the light on or off,
- Switch the light on, wait a given amount of time (e.g. 50 ms) then switch off the light,
- Set the global intensity level (between 0 and 100) of the light,
- Get the actual global intensity level,
- Get the status of the light ring (on, off or unknown),
- Get the status of the manual switch (pressed or unpressed),
- Check whether or not a vial is present in the chamber holder,
- Perform a full diagnostic test on the hardware controlled by the board (LED light ring, switches...)

The USB device can report status changes or events:

- The manual switch is pressed or
- A vial is inserted into the holder or
- The vial is removed from the holder.

These commands and events are packed in messages called *report descriptors* transmitting 2 bytes of data each.

4.4. Communication and system libraries

The Apollo-II prototype also defines two libraries used to enable the distributed collaboration between sensors across the network. The first two sections describe the archiving operation and the system monitoring of the system service. Finally the implementation of the three major components of the communication service (messaging, data transfers and resource discovery) is exposed.

4.4.1. Data archiving

The early Mercury, Gemini and Apollo-I prototypes implement the archiving operation using exclusively image files physically stored on a local hard disk of the sensor. An new experiment creates a new disk folder where the image frames are saved uncompressed. The metadata such as the sensor ID, timestamp and time offset used to identify each frame, are written directly in the corresponding file name. The Apollo-II prototype stores and accesses its data using both the file system and a MySQL database. MySQL is an efficient and reliable relational database system that allows to store the metadata and other information such as the network metrics (latency, throughput) and credentials. MySQL can also be used on many different platform and languages and an embedded version allows to integrate a small database into the system service libraries. The image files are still stored in folders on the hard disk. But a feature is added: in

order to identify consistently the sequences of data, a fixed-length hash key is computed for the sequence using the fast MD5 algorithm. It is based on the data contained in the image frames of the sequence. When the sequence is transferred from one node to another, the hash key is used for validity check and it ensures that the data from the source node is identical with the data on the destination node. The MySQL database can be concurrently accessed and updated by the local data acquisition service, the data processing service, the system and communication service. Retrieving data based on its metadata is made possible by using the Structured Query Language (SQL).

4.4.2. System monitoring

The system service monitors the local resources of the sensor using the underlying API of the operating system. Windows NT updates its resource information through its *performance counters.* These counters store dynamic system information such as the current CPU usage workload, the current memory usage the space available on the hard disks. Some general networking metrics are also updated in the Windows NT counters. But statistical information (average throughput, average latency) for each sensor are preferred, therefore the information is collected by the file server (HTTP) under the communication service.

4.4.3. Messaging technology

As described in the design section, the message-passing scheme is an essential element in building a collaborative network of sensors. Several technologies have been previously discussed and XML-RPC is the messaging protocol implemented in the CytoSensor network. XML-RPC is based on the XML encoding scheme and the HTTP protocol. A real advantage of this mechanism is the interoperability between networks, platforms and languages. Using the small XMLRPCPP library implementation for C++,

a Windows application on the Matrox 4Sight can easily integrate a XML-RPC server and client stub to access other XML-RPC servers. An Internet user can easily communicate with the Matrox 4Sight sensor node using a Java applet loaded with the XML-RPC client class package.

First an XML message request is built specifying the remote procedure (or method) name and its parameters using special XML tags defined by the XML-RPC specifications. The major parameter types include the numerical values (integer, double...), the character string, a structure of any complexity (nested or not) and the array. Then the XML message is wrapped up into a HTTP packet. The latter is transmitted to the sensor node identified by its IP address, port number (the default is 80). The message must be posted onto the "RPC2" service of the XML-RPC server. The HTTP packet is unwrapped and the XML message is parsed. The XML-RPC server calls the specified method and a message reply is computed accordingly. The reply is sent back to the requester node that was waiting for the answer. An typical example of a XML-RPC request packet is shown below.

```
POST /RPC2 HTTP/1.0
User-Agent:CytoCom/1.0b(WinNT)
Host:128.193.48.232
Content-Type:text/xml
Content-length:181

<?xml version="1.0"?>
<methodCall>
   <methodName>Acquisition.startNewExperiment</methodName>
   <params>
      <param>
         <value><i4>2</i4></value>
         <value><i4>2000</i4></value>
      </param>
```

```
   </params>
</methodCall>
```

And the corresponding reply message look like this:

```
HTTP/1.1 200 OK
Connection:close
Content-Length:158
Content-Type:text/xml
Date:Fri, 12 Oct 2002 23:55:08 GMT
Server:CytoCom/1.0b

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><boolean>1</boolean></value>
         </param>
      </params>
</methodResponse>
```

Various situations such as the call of an unknown method, too many parameters or a computational fault can produce error replies:

```
HTTP/1.1 200 OK
Connection:close
Content-Length:426
Content-Type:text/xml
Date:Fri, 12 Oct 2002 19:55:02 GMT
Server:CytoCom/1.0b

<?xml version="1.0"?> <methodResponse>
   <fault>
```

```
    <value>
       <struct>
          <member>
             <name>faultCode</name>
             <value><int>4</int></value>
             </member>
          <member>
             <name>faultString</name>
             <value><string>Too many parameters.</string></value>
             </member>
          </struct>
       </value>
    </fault>
</methodResponse>
```

If the authentication and the encryption of messages is necessary, XML-RPC can easily integrate the powerful and widespread Secure Socket Layer (SSL) protocol since the latter is well integrated into the HTTP protocol: XML-RPC then uses HTTPS for Secure HTTP.

4.4.4. Data transfers

Many protocols exist to efficiently transfer data from one node to another. But HTTP is the only universal protocol that can easily interconnect users and machines throughout on the Internet, and also offer a secure transmission (HTTPS). Therefore a lightweight HTTP server is developed in order to be used as a image samples server. Other nodes can download the image samples by connecting to the communication service that integrates the HTTP server. The latter is implemented using Berkeley standard sockets. A listener thread accepts connections from the remote sensors. When a connection arrives, an idle worker thread is picked up from a pool of idle threads to process the

request. This efficient technique of thread pooling can be enhanced by with the Windows I/O completion ports in order to create highly scalable servers. Messages and data transfers using the HTTP protocol can also be moved to and from any firewall-enabled network using the standard port 80. The technique is known as *HTTP tunnelling* or *cloaking*. Nothing in the firewall setup needs to be changed and no involvement is required from the network administrator.

4.4.5. Resource discovery and monitoring

Instead of using the multicast discovery and announcement mechanism used in the Mercury, Gemini and Apollo-I prototypes, the Apollo-II uses unicast XML-RPC messages and a discovery and lookup server. The server uses a MySQL database and a XML-RPC server to inform the users and operators of the available services on the CytoSensor network. An operator can contact such a server to locate a sensor or a service on the network, then the operator connects directly to the sensor to perform a task. The MySQL database on the server stores information about a CytoSensor node such as its sensor ID, its IP address(es) and its service(s). A node can directly query a remote sensor to check the data available for download or to monitor the workload of the sensor in terms of CPU and memory usage. This can help in choosing the best collaborator to perform a distributed task.

5. LIMITATIONS AND FUTURE CONSIDERATIONS

The actual implementation of the Apollo-II prototype needs improvements in programming the libraries, the design and the technological choices.

5.1. Limitations

Most of the actual implementation involves the development of the core middleware libraries. Higher-level services and distributed tasks are yet to be implemented.

The CytoSensor is presently used on a high-speed network with a small set of sensors. Although the peer-to-peer models are more scalable, deployment on a larger scale and tests must be conducted to assess the performance changes. Moreover the actual simplified design is based on the underlying network infrastructure where all the nodes are interconnected.

The data acquisition service of every CytoSensor prototype is specialized in image acquisition since a visual detection of toxins is used. However the actual design is aimed at developing a more general-purpose sensor by creating layered middleware libraries. Therefore there are still flaws in the design and implementation.

5.2. Possible improvements

5.2.1. Network simulations

Programs such as NS2 (Network Simulator 2) can simulate complex network topologies using discrete events. Simulated data can then be compared to improve the design of the CytoSensor network.

5.2.2. *Fault-tolerance*

Software fault-tolerance can be greatly improved in the actual implementation. The data acquisition library service must cope with eventual hardware failures from the frame grabbers and cameras or from the USB control devices. For instance, a camera may be disconnected, but it should not affect the acquisition performance. A transfer of data may be interrupted between 2 sensors, then a recovery procedure must be started.

5.2.3. *Scripting*

In order to create flexible distributed tasks across the network, a scripting layer can be used on top of the service layer. A general-purpose middleware can then be developed to perform basic operations such as acquiring, archiving and transferring the data samples. The application-specific data processing is performed at a higher level above the middleware and a distributed task is defined by a script using the aforementioned basic operations.

5.2.4. *Data fusion*

The present design focuses on the processing of data collected on individual sensors. Although the data acquisition and processing are distributed among nodes, no data fusion takes place at a higher level to assess a threat using a multitude of sensors concentrated in a geographical area. As described in [1], the basic idea is to have a number of independent sensors each making a local decision and then to combine these decisions at a fusion center to generate a global decision. For instance, [10] considers the case where the local decisions made at a number of sensors are communicated to multiple root nodes for data fusion.

5.2.5. Ad hoc mobile sensor networks

Finally, another improvement is the deployment of the CytoSensor on an ad hoc network where the sensors act as nodes and routers. Unlike the actual design, a sensor cannot connect directly to the requested node: instead other nodes route the messages and data to the final destination. This is referred to as the *store-and-forward* property. Complexity in the discovery of resources such as a node, a service or data, increases greatly and performance may be lower.

6. CONCLUSION

The CytoSensor project is an attempt to produce a portable sensor device specialized in the detection of toxic agents. This work involves the development of the real-time image acquisition, the data management and data transfers across a network. The development includes the creation and integration of various libraries and applications that control highly specialized hardware to meet the user needs. Prototypes called Mercury, Gemini and Apollo have been designed and built. To adapt to the hardware changes inherent to the prototypes, a layered and modular design of the software is used. As a result, the CytoSensor middleware libraries allow the applications developed on top of them to gain in flexibility and to reduce the development cycle. Through the course of this project, application of industry standards was of a high priority. The data acquisition uses the Vision SDK to capture images from various frame grabbers and USB as a universal scheme to connect many control devices such as the light control. Moreover, services of a single sensor were extended to a distributed sensor environment. Using XML-RPC messages and HTTP transfers, a sensor is able to acquire, share, process and propagate information among other sensors and Internet users. Assumptions have been made for the design and the implementation of the CytoSensor network. Therefore, they should be addressed in order to develop the project into a wireless and mobile ad hoc sensor network.

BIBLIOGRAPHY

1. R. Viswanathan and P. K. Varshney, "Distributed detection with multiple sensors: Part i - fundamentals," 1997, vol. 85, pp. 54–63, Proceedings of the IEEE.

2. J. N. Tsistsiklis, "Decentralized detection," in *Advances in Statistical Signal Processing, Signal Detection*, 1993, vol. 2.

3. "Seti@home," http://setiathome.ssl.berkeley.edu.

4. Martin Fowler and Kendall Scott, *UML distilled: a brief guide to the standard object modeling language*, Addison-Wesley, 2nd edition, 2000.

5. Andrew S. Tanenbaum and Maarten van Steen, *Distributed systems: principles and paradigms*, Prentice Hall, 1st edition, 2002.

6. Nicolas Roussel, "Advanced image segmentation, and data clustering concepts applied to digital video featuring the response of organic material to toxicological agents," M.S. thesis, Oregon State University, 2003.

7. Mukesh Singhal and Niranjan G. Shivaratri, *Advanced concepts in operating systems: distributed, database and multiprocessor operating systems*, McGraw-Hill, Inc, 1994.

8. James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison-Wesley, 2nd edition, 2002.

9. Voranon Kiettrisalpipop, "Cytosensor: System integration and human interface design," M.S. thesis, Oregon State University, 2003.

10. K. R. Pattipati Z. B. Tang and D. L. Kleinman, "Optimization of distributed detection networks: Part ii generalized tree structures," 1993, vol. 23, pp. 211–221, IEEE Trans. Syst.

11. Athanassios Boulis and Mani B. Srivastava, "A framework for efficient and programmable sensor networks," 2002, OPENARCH.

12. Rajkumar Buyya, *High Performance Cluster Computing: Architecture and Systems (Volume1)*, Prentice Hall, 1999.

13. Andy Oram, *Peer-to-peer: Harnessing the power of disruptive technologies*, O'reilly and Associates, Inc, 1st edition, 2001.

14. Bo Leuf, *Peer-to-peer: Collaboration and Sharing over the Internet*, Addison-Wesley, 2002.

15. Ji seok Liew, "Web-based distributed applications for cytosensor," M.S. thesis, Oregon State University, 2003.

16. "Hypertext transfer protocol – http/1.1," RFC 2616, Internet Engineering Task Force, June 1999, http://www.ietf.org/rfc/rfc2616.txt.

17. "Rpc: Remote procedure call protocol spefication version 2," RFC 1831, Internet Engineering Task Force, Aug. 1995, http://www.ietf.org/rfc/rfc1831.txt.

18. "Extensible markup language (xml)," 1998, http://www.w3.org.

19. Dave Winer, "Xml-rpc specification," 1999, http://www.xmlrpc.com/spec.

20. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple object access protocol (soap) 1.1," 2000, http://www.w3.org.

21. Matrox, *Matrox 4Sight-II: hardware reference*, 2002.

22. Matrox, *Matrox Imaging Library*, 2002.

23. Euresys, *Multicam API for Picolo Pro frame grabbers*, 2002.

24. Microchip, *PICDEM: USB board hardware reference*, 2002.

25. Microchip, *PIC 16C765: USB firmware and application notes*, 2002, http://www.microchip.com.

26. Walter Oney, *Programming the Microsoft Windows Driver Model*, Microsoft, 2002.

27. David A. Solomon and Mark E. Russinovich, *Inside Microsoft Windows 2000*, Microsoft Press, 2000.