ANOMALY DETECTION MODELING

IN

MEDICAL PERVASIVE SYSTEMS

by

Biniyam Asfaw

A thesis submitted to

The school of Graduate Studies of Addis Ababa University

in partial fulfillment of the requirements for the Degree of

Masters of Science in Computer Science

July, 2007

Acknowledgement

I would like to pass my many thanks to my advisor, Dr. Dawit Bekele, for assisting me by providing priceless comments on the documents and by controlling the progress throughout the time of the research. I would also like to thank ato Asfaw Iddosa for making final readings on most chapters.

Finally, I would like to thank Addis Ababa University, especially the Informatics Faculty, for allotting research budget to cover my expenses in the course of the research.

Table of Contents

Table of Figures

Abstract

Pervasive computing is being applied to different areas of specialization. This is basically because of the features of pervasive computing like context-awareness, invisibility, non-intrusiveness, and mobility. The medical area is one where such devices are hugely deployed. In this case, the pervasive devices; PDA (Personal Digital Assistant), mobile phones and the like, are used for manipulating medical records on the move.

The use of pervasive devices also comes with new challenges that did not exist with traditional computing systems. Among these challenges, security is probably the major one. In fact, insuring security with pervasive systems is difficult due to the use of wireless communication, the physical nature and the low processing and low power nature of the devices.

In this research, we deal with intrusion detection, ID, to secure such systems. ID Systems, IDSs, are used to monitor a resource and notify someone in the event of a specific occurrence for an appropriate response. Based on attack identification, they can be those which implement misuse detection, matching against known attack patterns, and those which implement anomaly detection, deviation from normal patterns. Misuse detection is used for matching only known patterns of attacks while anomaly detection is capable of identifying new attacks by matching with an already established normal profile. Based on source of information for the IDS, it may be host-based, network-based or application-based. For our case, we deal with application based anomaly detection modeling issues through building normal users' application usage profiles.

1. Introduction

1.1. Background

The field of intrusion detection has been in the computer security since the early 1980s. Some of the earliest work in intrusion detection was performed by Jim Anderson in the early 1980s [1]. Anderson defines an intrusion as any unauthorized attempt to access, manipulate, modify, or destroy information, or to render a system unreliable or unusable. Intrusion detectors are meant to warn of attempted intrusions. He also suggested ways to detect the different intruders, except for intruders who tamper security mechanisms and authorized users misusing their privileges, for which no solution was offered. Based on Anderson's study, early work focused on developing algorithms and procedures for automated audit trail analysis in batch mode. Minimizing the amount of data to examine would ease the security officer's work.

The first model for an intrusion detection system was proposed by Denning [2]. The model is based on the hypothesis that intrusions can be detected by searching for abnormal patterns of system usage. Rules are used to compare current usage with statistical profiles of normal behavior. Many of the early systems were based on this model.

Most initial attempts to intrusion detection were made on deploying them on networks, network-based IDSs. The network-based IDSs look for known attack patterns on a network, like receiving more than a specified amount of packets from a single host is possibly a denial of service attack, or deviations from an already established normal operation of the network. But there can also be attacks targeted to a specific host, not possible to be detected by the network based IDS, by legitimate users who operate beyond the network protections.

As a result, the need for integrating host-based and application-based IDSs became apparent. Application-based IDSs collect information and detect intrusion at the application level, for example, application-based IDSs deployed at e-commerce server, medical server and web server. The host-based IDSs monitor event logs and critical system files. Figure 1 shows a sample IDS architecture indicating the possible deployment points of the different IDSs types.

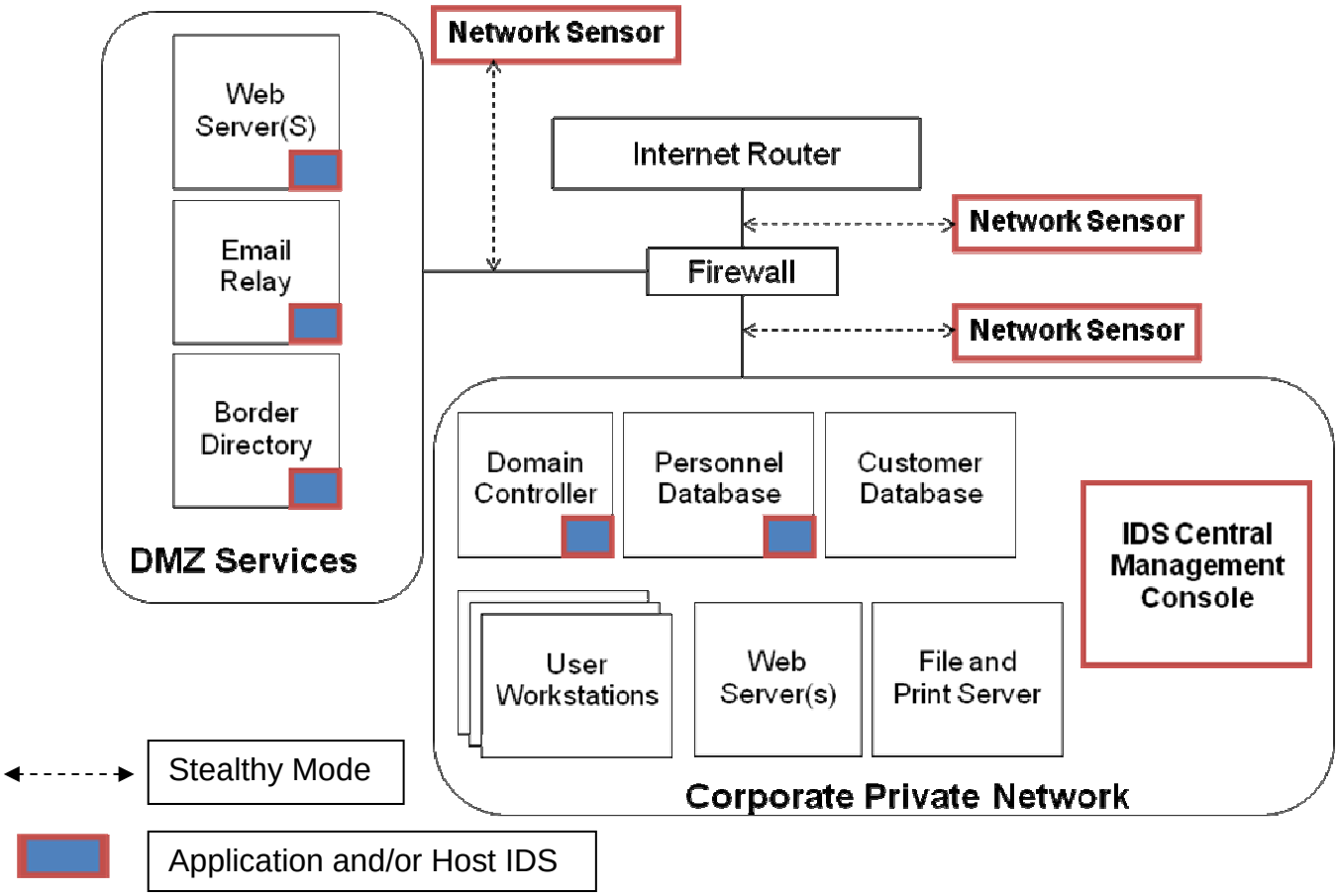

Figure 1: Sample IDS Architecture

1.2. Motivation

Computer security has become an area of utmost importance as the number of people who use computers continues to increase. As more and more computer systems and networks are setup, there are inevitably bugs in these systems which attackers attempt to exploit. Even though the

fields of security and cryptography have successfully produced a wide range of technologies that are now broadly used to secure computer systems, news media, almost on a weekly basis, report that malicious users still succeed in attacking systems with sometimes devastating losses, in vital data or services. Many fear that such losses often go undetected; attackers often announce their successes before anyone has noticed their deeds. The reasons such attacks remain commonplace are due to lenient security policies or procedures, error in configuring security systems, and, perhaps worse, insider attacks.

Considering specific cases where medical records are stored to be accessed by multiple users, it is crystal clear that it attracts attackers to get access to this medical record. Medical records may be required by corporations in deciding who should be promoted, by insurance companies in refusing coverage for people with poor health, and by spouses and their attorneys in divorce cases etc. Currently, a number of mobile devices are being integrated into the healthcare environment due to the mobility of healthcare professionals. Detail on the application of pervasive devices in healthcare environments is provided in section 1.5. Figure 2 shows one possible scenario for a pervasive healthcare system that integrates the unique capabilities of current and emerging mobile devices and wireless networks.

This research work is specifically motivated by the following factors:

- Most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse; finding and fixing all these deficiencies is not feasible for technical and economic reasons;
- Existing systems with known flaws are not easily replaced by systems that are more secure-mainly because the systems have attractive features that are missing in the more-secure systems, or else they cannot be replaced for economic reasons;

- Developing systems with absolute security is extremely difficult, if not generally impossible;
- Medical systems have far more security requirements since they deal with sensitive medical records that need confidentiality, integrity and availability. Also, different governmental laws and ethical issues enforce the security of such systems.
- Application of pervasive devices into medical systems comes with its own challenges, security being the most challenging as will be discussed in section 1.5;
- Even the most secure systems are vulnerable to abuses by insiders who misuse their privileges.

1.3. Objective

This research is aimed at conducting a study on modeling Application-based IDS for medical pervasive systems. Such Application-based IDSs are expected to withstand and identify attacks with good precision since they are close to the application and can integrate application knowledge.

The pervasive nature of the application aids us to integrate location information which can be one good resource for constructing normal profile for lrgitimate users.

In addition to identifying a model, we also use sample dataset, which is shown in Appendix C, to show the rules generated to be used for identifying anomalous activities.

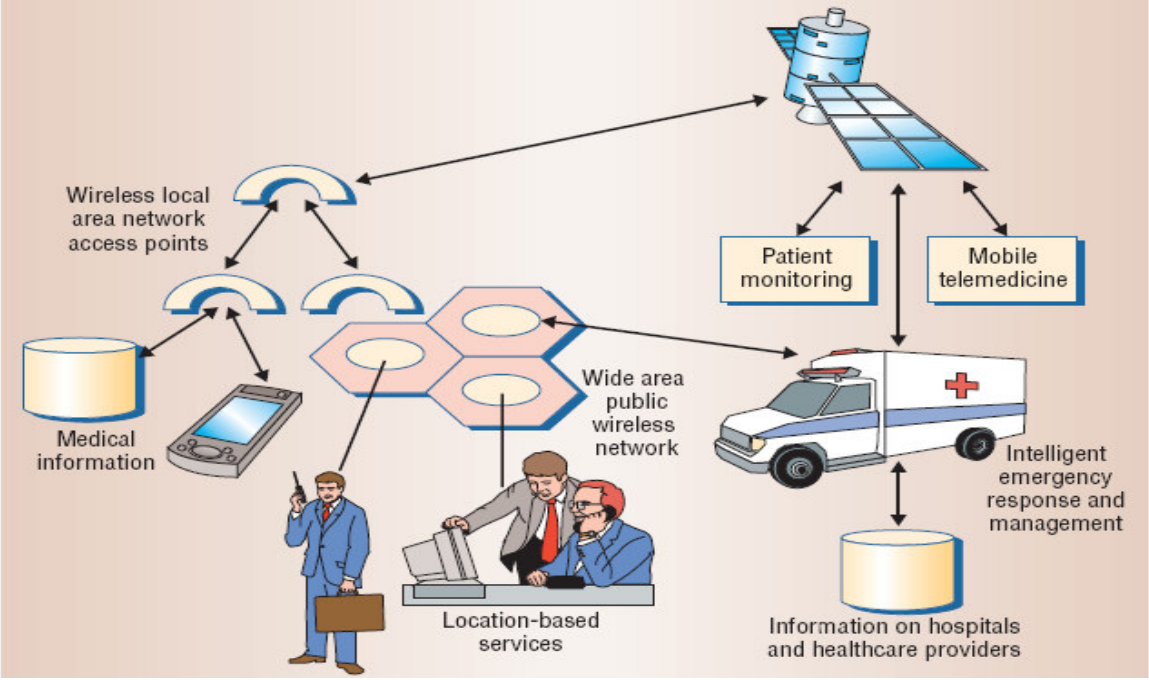

Figure 2: Pervasive healthcare scenario

1.4. Methodology

There are anumber of methods for constructing IDS models. As indicated on figure 1, it is possible to have IDSs deployed at different points in a working environment; like firewalls and application servers.

It is also possible to have different ways of detecting intrusions; anomaly and misuse detections. Besides these, we can also use different methds of modeling a specific IDS primarily based on the data source used in normal or attack pattern construction. Examples can be user commands/request, packets exchanged, and system calls raised by applications

Our method for constructing the ID model is based on anomaly detection using user requests to construct normal profile. We have implemented data mining techniques, associative classification which is integration of association and classification. More on this are included in section 5.

1.5. Application and Challenges of Pervasive Computing in Healthcare Environments

Traditional computing provides functionalities which may not be comfortable to use for some specific areas of specialization. This can be attributed to lack of allowing mobility, inability to integrate context information and their intrusive nature. Context awareness refers to actively exploring physical location, available computing infrastructure and offered services running on such infrastructures to fulfill and enhance their ability to perform their designated tasks. The intrusive nature, requiring inputs from the user now and then, arises mostly due to lack of exploiting context information.

This is particularly true with healthcare systems. Medical work is usually referred as nomadic, where the healthcare works exhibit mobility and interrupted operations. Figure 3 shows the promises of pervasive computing in healthcare environment. Location based computing is required in healthcare environments to enable delivery of accurate medical information anywhere and anytime, thereby reducing errors and improving access. In the login process, for example, there are interrupted operations where the healthcare workers usually tend to log in but forget to log out. This is basically because medical work is nomadic while login is fixed to one computer in the traditional computing [3].

In such cases, implementing pervasive devices is ideal. Pervasive computing provides context-awareness, invisibility and non-intrusiveness, and mobility. For this reason, pervasive devices are widely deployed in healthcare environments to exploit its features.

The application of pervasive computing in hospitals, or healthcare centers, has a number of advantages. A large amount of information about patients' health status, like blood pressure and heart beat, is collected from different devices monitoring the patients' health. In this case,

without necessarily requiring intervention of the medical staff, pervasive devices can monitor the information sent from these devices and alarm the medical staff only whenever like, for example, blood pressure becomes out of the expected range for some specific patient.

Figure 3: The promise of pervasive computing in healthcare environment

Ability of exploiting location information can also be an additional advantage through procedural improvement in hospitals [3]. Location is also basic feature in detecting context features like indicating a doctor in an operating room that he/she should not be contacted in matters of low urgency.

The use of pervasive devices also comes with challenges that are more difficult to solve compared with traditional computing systems. Among these challenges, security is probably the major one. In fact insuring security with pervasive systems is difficult due to the use of

wireless communication, ease of physical attack, and the low processing and low power nature of the computing devices.

The communication means used with pervasive devices is wireless which makes eavesdropping, listening to the data transmitted, simple. This imposes great privacy challenges in the case of exposing medical data being transmitted. Even worse can be in the case where active interception is done, that is, transmitted data is modified. In addition, firewalls, which are effectively implemented in traditional computing environments, can not be used with pervasive devices since anyone in range can communicate with wireless devices and lack of natural place to put firewalls.

Heterogeneous devices with limited computational power best describe pervasive devices. This would require coming up with cryptographic primitives that are efficient on a variety of platforms which puts great challenge on ensuring security on such devices.

The low power nature of these devices also poses additional challenge which is not the case with traditional computing. Special denial of service attacks, like battery exhaustion attacks, can be launched on pervasive devices.

1.6. Why Intrusion Detection?

As computer systems play increasingly vital roles in modern society, they have become the targets of enemies and criminals. When an attack occurs which can be any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource, prevention techniques such as encryption and authentication, using passwords or biometrics, are usually the first line of defense. However, attack prevention alone is not sufficient because as systems become ever more complex, while security is still often the after-thought, there are always

exploitable weaknesses in the systems due to design and programming errors.

Attacks on computer systems may originate from external or internal sources. Figure 4 shows deployment of IDSs for both sources of threats. External attackers do not have any authorized access to the system they intend to attack. Internal attackers have at least some authorized access to the system. Once external attackers manage to gain some access to a system, using exploitable holes in the system, they appear as internal attackers. A good example can be in the case where in a well guarded building, an intruder gets in with a forged id card. Once the intruder is beyond the first line of defense, he/she is classified as an internal attacker, who still can be caught by tracing his/her activities in the building with a security camera, which can be considered as a second line of protection.

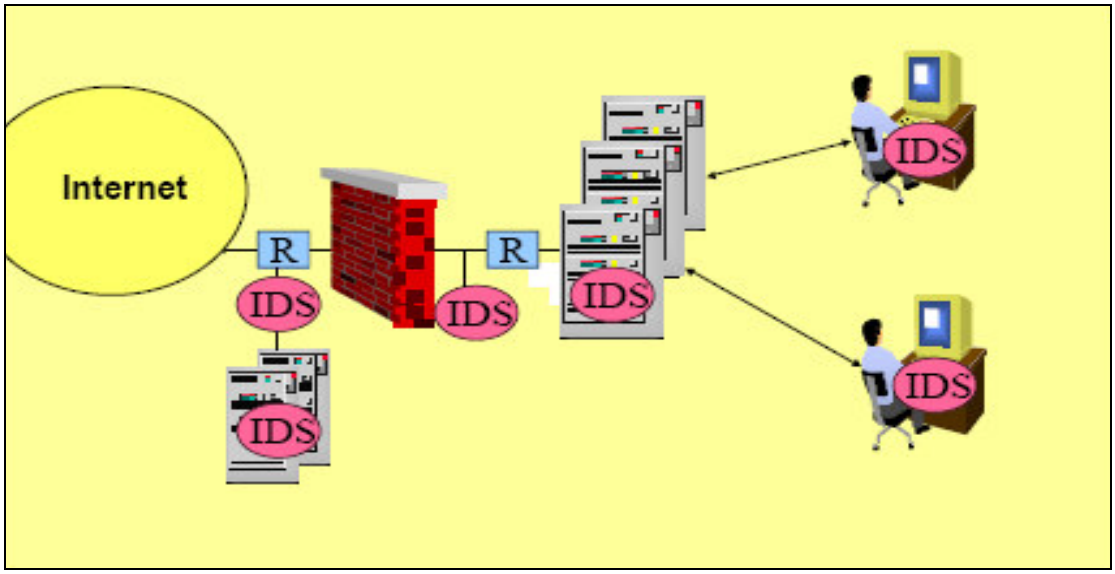

Figure 4: IDSs for internal and external threats[17]

In addition to the illegitimate attackers, legitimate users, who have access to the system, may try, consciously or unconsciously, to misuse their privileges. Intrusion detection can be used as a second wall to protect systems because once an intrusion is detected, response can be put into

place to minimize damages, gather evidence for prosecution, and even launch counter-attacks.

Intrusion Detection Systems (IDSs) are used to monitor a resource and notify someone in the event of a specific occurrence for an appropriate response [4]. IDSs are the `burglar alarms' of the computer security field. From a high-level view, the goal is to find out whether or not a system is operating normally. Abnormality or anomaly in the system behavior may indicate the occurrence of system intrusions that are the consequences of successful exploitation of system vulnerabilities.

IDSs can be categorized based on their detection methods. When the IDS uses information about the normal behavior of the system it monitors, it can be categorized as anomaly-based IDS. When the IDS uses information about the attacks, attack signatures, it qualifies as a misuse-based IDS. There are a number of research efforts exerted towards both anomaly-based and misuse-based IDSs. Anomaly-based IDSs are capable of identifying new attacks, but they may suffer from high false positive rate, identifying an activity as anomalous while it is not indeed.

The behavior on detection describes the response of the IDS to attacks. When it actively reacts to the attack by taking either corrective (closing holes) or proactive (logging out possible attackers, closing down services) actions, then the IDS is said to be active. If the IDS merely generates alarms (such as paging), it is said to be passive.

The audit source location discriminates IDSs based on the kind of input information they analyze. This input information can be audit trails (system logs) on the host, network packets, application logs, user commands (operations), or even intrusion alerts generated by other IDSs.

Even if the researches in Intrusion Detection are very young, there are already efforts done

especially in the case of fixed networks. However, techniques implemented for security and privacy issues in the fixed networks can not be directly applied to pervasive environment. This is because there are additional security issues introduced due to implementation of pervasive devices as explained in section 1.5.

Location-based services allow applications to customize their services to users based on location information. This can have privacy problems since users may not want service providers to know about their whereabouts [5]. A good example in this case can be the mobile phones which periodically inform base stations about their location.

In addition to this, the wireless links used in pervasive environments make passive eavesdropping, listening to transmitted data, and active interference, involving alteration of data, possible.

The cooperation environment in pervasive systems also contributes to the aggravated security problems. With cooperative environment, devices report their state and sensor information to other objects [6].

The devices used are also less physically secured, compared to nodes used in fixed networks, which contributes to the ease of compromising the devices [5]. Resource limits also contribute to pausing problems in implementing heavy cryptographic algorithms.

Specifically considering the case of Intrusion Detection, the network-based ID Systems for fixed networks rely on traffic information collected at switches, routers and gateways which can not be the case in pervasive environment which do not have such traffic concentration points where the IDS can collect audit data for the entire network. Furthermore, it is very difficult to make clear distinctions between normal and anomalous operations. A node that

sends out false routing information can be the one that has been compromised, or temporarily out of sync due to volatile physical movement [5]. Intrusion detection may find it increasingly difficult to distinguish false alarms from real intrusions.

Yet, considering a specific case where this pervasive computing technology is deployed in a healthcare system, it is likely to create concerns about security. In a medical environment, patients give their medical information. As a result there are a number of attempts to have unauthorized access to medical records. Deployment of pervasive devices in pervasive environment may aggravate the inherent security problem. Reasons can be; increased data aggregation, ubiquitous access, and increasing dependency on technical solutions [4]. As indicated in [4], pervasive computing technology in a healthcare environment raises privacy and security concern due to collecting, exchanging and processing of data reflecting current user situation is done in the background.

Even though it has these concerns, application of pervasive computing in healthcare has a number of advantages which outweigh the concerns. Some of the advantages are those that are discussed in the previous sub-section.

Communication reliability, usability (especially for intermediate devices handled by patients), and auditing capabilities (for sensitive changes which may endanger patients' well-being) are the basic requirements expected from pervasive healthcare systems as described in [4]. In such cases of remote health monitoring integrity and authenticity of data are of importance. Failure to handle these can cause attacks leading to battery exhaustion attacks in sensors and intermediary devices, jamming of transmission link, and also injection of computationally heavy processes which may cause denial of services which in turn may endanger patient life. Most serious and hard to detect attacks can also be impersonation and insider attacks, in which

case intrusion detection can play great role towards detecting such attacks and notify before they cause more serious problems.

The aim of this research is to develop a model for host-based anomaly detection system for pervasive medical systems. The model to be developed will be tailored specifically for medical applications that tries to model normal activities of the medical staff to evaluate every activity against these normal behaviors.

The remaining of this document is organized as follows. Section 2 describes the main works done in the area of ID. The modeling architecture that we propose will be described in section 3. The integrated framework, associative classification, used in this paper is described in section 4. This section also covers the data structures worth implementing for efficiency reasons. A prototypical implementation where rules are generated using sample data is described in section 5. Section 6 and 7 include the conclusive statements and directions for future works, respectively.

2. Literature Review

IDSs have gone through two decades of development. It is a relatively younger research area. But it is not only the “age” of the research area that contributes to its sluggish development. There are a number of challenges with ID Systems:

1. The IDS itself may be compromised.
2. Cooperation between different commercially available IDSs is not going to be an easy task since they implement their own proprietary protocol for communication between sensors and analyzers.
3. No industry standards against which to compare such systems.

Intrusion Detection Systems are categorized by source of information they use and how they identify attacks. Based on information source, as described in section 1.6, there are three kinds of ID systems; Host-Based, Network-Based, and Application-Based.

There have been many proposals for how to do host-based IDSs, but a paradigmatic example is the general approach of Forrest et al [7]. Their scheme is motivated by using the human immune system as a biological analogy. If the system call traces of normal applications are self-similar, then we can attempt to build an IDS that learns the normal behavior of applications and recognizes possible attacks by looking for abnormalities. In the learning phase of this sort of scheme, the IDS gathers system call traces from times when the system is not under attack, extracts all sub-traces containing six consecutive system calls, and creates a database of these observed sub-traces. A sub-trace is deemed anomalous if it does not appear in this database. Then, in the monitoring phase, the abnormality of a new system call trace is measured by counting how many anomalous sub-traces it contains. The work done by Forrest et al. [7] is yet another good example for anomaly detection based ID systems.

Bro is a research tool being developed by the Lawrence Livermore National Laboratory [8]. It is a network-based ID system. In fact, there are additional design goals for Bro; high load monitoring not to drop packets in the case of intentional flooding by the attacker, real-time notification, system extensibility, and an ability to repel attacks. Bro has three level hierarchy of functions; libpcap to extract packets from the network, event layer to check packet headers for integrity, and policy script interpreter to interrogate events generated by the event layer.

There are also a number of hybrid ID systems which integrate both attack identification techniques, anomaly and misuse. EMERALD, Event Monitoring Enabling Responses to Anomalous Live Disturbances, is a research ID tool developed by SRI [9]. It is capable of both

misuse and anomaly detection. The researchers initially used the multivariate statistical algorithm for analyzing user behaviors as usual or unusual, anomaly detection. Afterwards, they integrated the expert system P-BEST, Production-Based Expert System Toolset, for signature analysis-analyzing against known patterns of attack, misuse detection.

In the pace to reach at EMERALD, researchers at SRI initially generated the IDES, Intrusion Detection Expert System, which monitors activities on multiple host in real-time. Based on IDES, NIDES, Next generation IDES was developed. Initially there were little changes integrated into NIDES; it was a host based tool implementing P-BEST. However, it was progressed further by incorporating a RESOLVER for fusing results of anomaly and misuse detection components. A lot of improvements also made on the user interface component of the ID System. EMERALD was then introduced based on the efforts on IDES and NIDES. But EMERALD is a Network ID System. It is built to function in loosely coupled enterprise network. Due to the autonomy of these networks, challenges were there in the development. Inability to centrally analyze such autonomous network systems led EMERALD to exercise "divide and conquer" technique. As a result, a three-tiered system of monitors is implemented; Service Monitors, Domain Monitors, and Enterprise Monitors. Similar architecture for all monitors: a set of Profile Engines, for anomaly detection, a set of Signature Engines, for misuse detection, and a Resolver Component for integrating results of the anomaly and misuse detection engines.

Prior implementation of NIDES demonstrated that the statistical profiling techniques are effective. Concerning signature analysis for known attacks, the service monitor signature engines monitor domain components for abnormal activities. Based on this information, enterprise monitor signature engines try to detect if there are any broader attacks. The resolver

component provides further functionalities, in addition to fusing the outputs of misuse and anomaly detection engines, like providing a subscription service to integrate a third-party tool to be integrated, providing interface component and initiating attack defensive reaction like process termination. EMERALD attracts future researches basically due to possibilities for integrating additional tools especially for incorporating tools which detect distributed attacks in real-time.

NetSTAT is the latest in a line of “STAT” research tools produced by the University of California at Santa Barbara [10]. The STAT research tools implement state transition analysis supporting real-time Intrusion Detection. That is, there are sequences of actions which initiate transmission from authorized state to compromised state. In the case of STAT the audit trail information is transformed through an “audit trail analyzer” that filters and abstracts the information gathered at the audit trail level. These abstractions are termed as signatures which effect portability, analysis and human understanding. The signature actions are attributed to state transitions. Transitions captured in production system rule-sets are set apart as intrusion.

The initial implementation of STAT was USTAT; a UNIX-based host-based system. It was composed of a preprocessor which is used to filter and manipulate data into a form independent of audit-file. The knowledgebase has two components; a fact base which stores the dynamically changing state of the system and a rule base component responsible for storing state transition rules that indicate predefined intrusion sequences. The inference engine, which is other part of USTAT, is responsible for identifying any significant state changes and updates the fact base component of the knowledgebase. The inference engine is also responsible for notifying the decision engine, other part of USTAT, of possible security violations. The decision engine in turn notifies the administrator or triggers itself a countermeasure. One big advantage of such

system, implementing state transitions, is that intrusions can be detected prior to compromising a system. Yet another advantage can be since it implements inference engine table to track each potential intrusion it can detect attacks originating from multiple sources, distributed attack. NetSTAT is the current system under construction which addresses network intrusions.

Other research described in [17] attempted to develop a new and efficient technique for the detection and alleviation of Denial-of-Service, DoS attacks. They present a new block based technique that may provide significant reduction of overhead in processing and analyzing traffic data. They proposed a distributed system that collects diagnostic data from sensors across the mirror servers and computers on the condition of the network. This data is mined in real time to generate traffic models. Incoming traffic is matched against traffic models to check for anomalies and corrective measurements are taken as needed.

The researchersclaim that in the most DoS attacks, an attacker floods the victim's network by transmitting millions of garbage packets or by issuing illegitimate requests to consume the web server's resources. This method of issuing fake requests can be classified into two types; SYN attacks and HTTP request attacks.

They suggested a novel approach for detection and response towards a DoS attack. Their network topology consists of sensor agents called DCA (Data Collection Agents) located at the edges and managing agents called DFA (Data Fusion Agents) located inside the network. The DFA's generate traffic models based on current network traffic and the DCA's use these models to detect anomalous traffic. If traffic is deemed anomalous, then the DCA can either drop it or load balance it to another low priority server.

The algorithm they used is a rule-based inductive learner that builds a set of rules over the features they prepared. They associated threshold values over all the features obtained during

training, and if a certain threshold was breached then an anomaly was recorded.

As an experiment analysis, they used the DARPA's 1999 Lincoln Lab dataset. The dataset consisted of tcpdump data from a simulated Air Force network. Within the dataset, there appear a number of DoS attacks. They evaluated their work with respect to; True Positive, the number of malicious blocks classified as malicious blocks, True Negatives, the number of benign blocks classified as benign blocks, False Positive, the number of benign blocks classified as malicious blocks, and False Negative, the number of malicious blocks classified as benign blocks. They found an overall accuracy of 68.38%.

Another similar paper [18] that implements data mining technique is that proposes a methodology towards developing a more-robust Intrusion Detection System through the use of data-mining techniques and anomaly detection. These data mining techniques will dynamically model what a normal network should look like and reduce the false positive and false negative alarm rates in the process. They used classification-tree techniques to accurately predict probable attack sessions. Overall, their goal is to model network traffic into network sessions and identify those network sessions that have a high-probability of being an attack and can be labeled as a "suspect session."

They started their research towards developing a robust network modeler through the use of data-mining techniques using the MIT Lincoln Lab data sets from their work in 1999. The MIT researchers created a synthetic network environment test bed to create their data sets. These data sets – which contained both attack-filled and attack-free data – allowed them to look at network traffic stored in tcpdump format where the attack types, duration, and times were known and published. Looking at the data sets, the individual rows represent network packets while the columns represent the associated variables found in that particular network packet.

They used a predictive, classification tree method for their model. This technique allowed them to determine which categorical variables were pertinent to their ability to accurately predict the attack sessions in their network.

Further, they intend to conduct monthly “manual remodeling” of their training and validation data set to ensure that any installations of new systems and protocols will be captured in their model and thereby reduce the number of false positives.

Somehow related to our work is the research that describes a novel framework, MADAM ID, for Mining Audit Data for Automated Models for Intrusion Detection. This framework, as described in [12], uses data mining algorithms to compute activity patterns from system audit data and extracts predictive features from the patterns. It then applies machine learning algorithms to the audit records that are processed according to the feature definitions to generate intrusion detection rules. Results from the 1998 DARPA Intrusion Detection Evaluation showed that their ID model was one of the best performing of all the participating systems. They also briefly discussed their experience in converting the detection models produced by off-line data mining programs to real-time modules of existing IDSs.

They have developed a framework, MADAM ID (for Mining Audit Data for Automated Models for Intrusion Detection). The main idea is to apply data mining techniques to build intrusion detection models. The main components of the framework include programs for learning classifiers and meta-classifiers, association rules for link analysis, and frequent episodes for sequence analysis. It also contains a support environment that enables system builders to interactively and iteratively drive the process of constructing and evaluating detection models. The end products are concise and intuitive rules that can detect intrusions, and can be easily inspected and edited by security experts when needed. Note that currently

MADAM ID produces misuse detection models for network and host systems as well as anomaly detection models for users. We are extending MADAM ID to build network and host anomaly detection models. Also note that the detection models produced by MADAM ID are intended for off-line analysis.

They have described their experiments in building intrusion detection models on the audit data from the 1998 DARPA Intrusion Detection Evaluation Program. In these experiments, they applied the algorithms and tools of MADAM ID to process audit data, mine patterns, construct features, and build RIPPER classifiers.

In addition, they have also discussed that "Insiders" misusing their privileges can also seriously compromise security. They claim that these insider attacks are hard to detect because the insiders don't need to break in. The goal of user anomaly detection is to determine whether the behavior of a user is normal (i.e., legitimate). They propose that a user's actions, during a login session, need to be studied as a whole to determine whether he or she is behaving normally. They used Bro event handlers to examine the telnet sessions, and extract the shell commands of the users.

To analyze a user login session, they mined the frequent patterns from the sequence of commands during this session. This new pattern set was compared with the profile pattern set and a similarity score is assigned. Assume that the new set has n patterns and among them, there are m patterns that have "matches" (i.e., rules that they can be merged with) in the profile pattern set; then the similarity score is simply m/n. Obviously, a higher similarity score means a higher likelihood that the user's behavior agrees with his or her historical profile.

3. Overview of the Proposed System

In this research, we propose a technique for identifying intrusion, from legitimate users, in a medical pervasive system. As described in the introduction section, there are two ways of detecting intrusions, through maintaining known attack patterns to match with, misuse detection, and through maintaining expected normal behaviors and identifying deviations as intrusion, anomaly detection. In our case, we have used the anomaly detection technique of identifying intrusions. This, obviously, requires defining expected normal behaviors of the medical staff that use the medical records through the application. This section deals with exposing the architecture of the proposed system and to detail every component in terms of its inputs, outputs and functionalities.

In the pervasive medical system, there are two components which communicate over a wireless link, the pervasive device component and the server component. Requests for services are sent from the pervasive device to the server. In our specific case, these are requests for manipulating the Medical Record Store MRS) database. The server, in this case, responds to the pervasive device component. The response can be a record fetched from the MRS or even it can be an acknowledgment message. Figure 5 shows this request/response between the pervasive device and server components through a web server.

There are a number of components, inside the server, which are responsible to respond to the requests appearing from the pervasive device. From these components, user commands, requests, can be collected since they are responsible to receive requests from the pervasive devices, belonging to a specific user, and send responses.

The User Activity Dump, UAD, is a database used for storing the users' requests collected from the Server Side Application Components, SSAC. The data in the UAD is going to be used as an input for building the users' normal profile, with an assumption that the UAD contains noise-free data, free from anomalous activities.

One reason why we are considering medical systems accessed through pervasive end points is that, as described in the introduction section, such devices are frequently used in healthcare environments because of the mobility of the medical staff. As a result, we exploit location information to maintain expected behaviors of the medical staff.

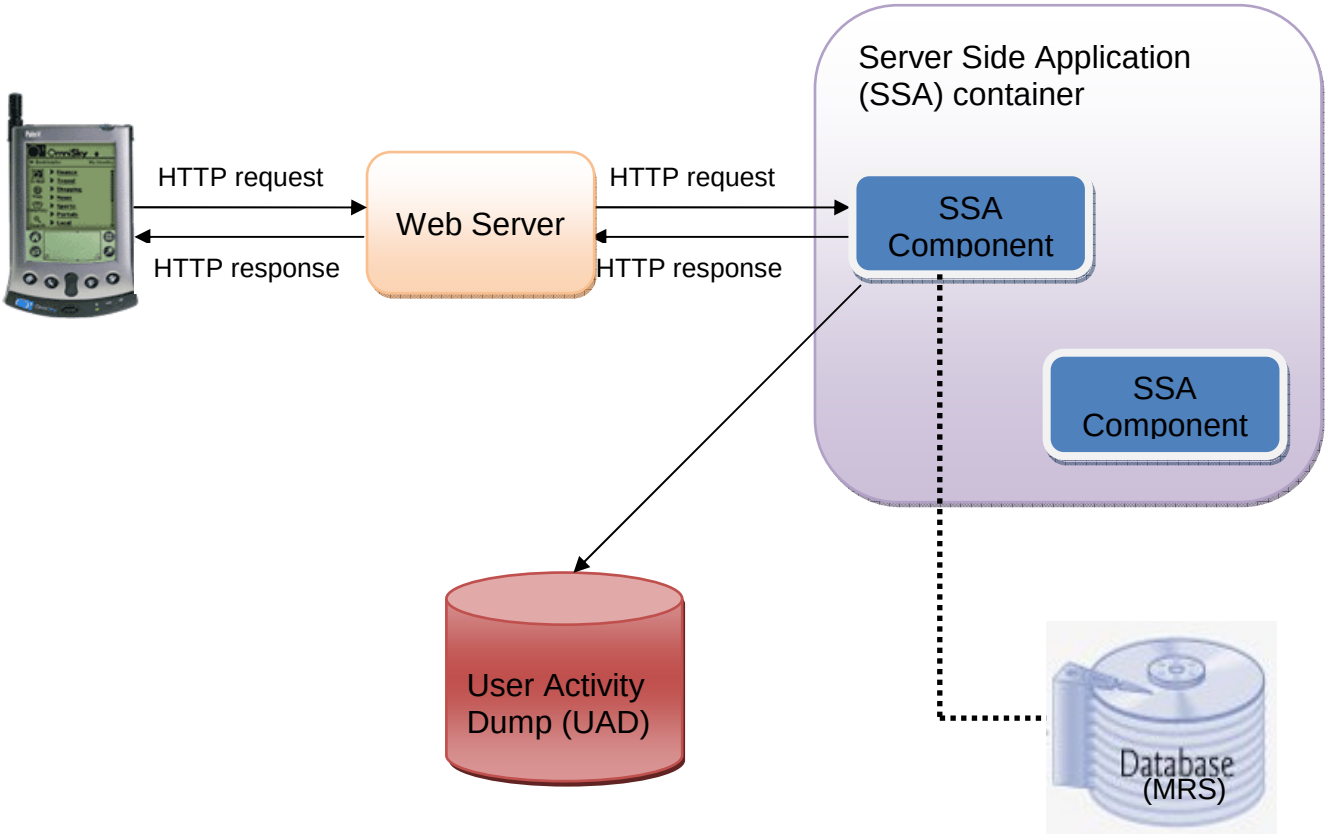

Figure 5: Communication between client and server using a pervasive end point and collection of dataset for normal profile construction

3.1. The Pervasive Side Application Component (PSAC)

The PSAC contains the Pervasive Side Application (PSA), the User Category Record (UCR) and Location Record (LR) databases as shown in figure 6.

The PSA is, basically, a user interface for the pervasive device, that is, the user interacts with the PSA to send requests to the server and view responses sent by the server.

The UCR database is used to maintain current users of the system. 'Current users' refers to the category of the users of the pervasive medical system; like doctor1, doctor2, doctor3, nurse, lab technician, and pharmacist. We have maintained different doctor category type to represent doctors working in different time categories, am, pm, and night. The reason behind maintaining this database on the PSAC is for randomly selecting a user category that currently interacts with the server, since we are using a single pervasive device end for our case.

The need to have LR database on the PSAC is for similar reason. One advantage we can collect for our anomaly detection model is the availability of location information of users. For this reason, we have integrated an LR database that contains location information; like room12, room22, room32, office, laboratory, pharmacy, and other. When a user sends a request, location information is selected randomly from this database and is sent along with the request. This is done for simulating GPS, Global Positioning System, enabled pervasive devices.

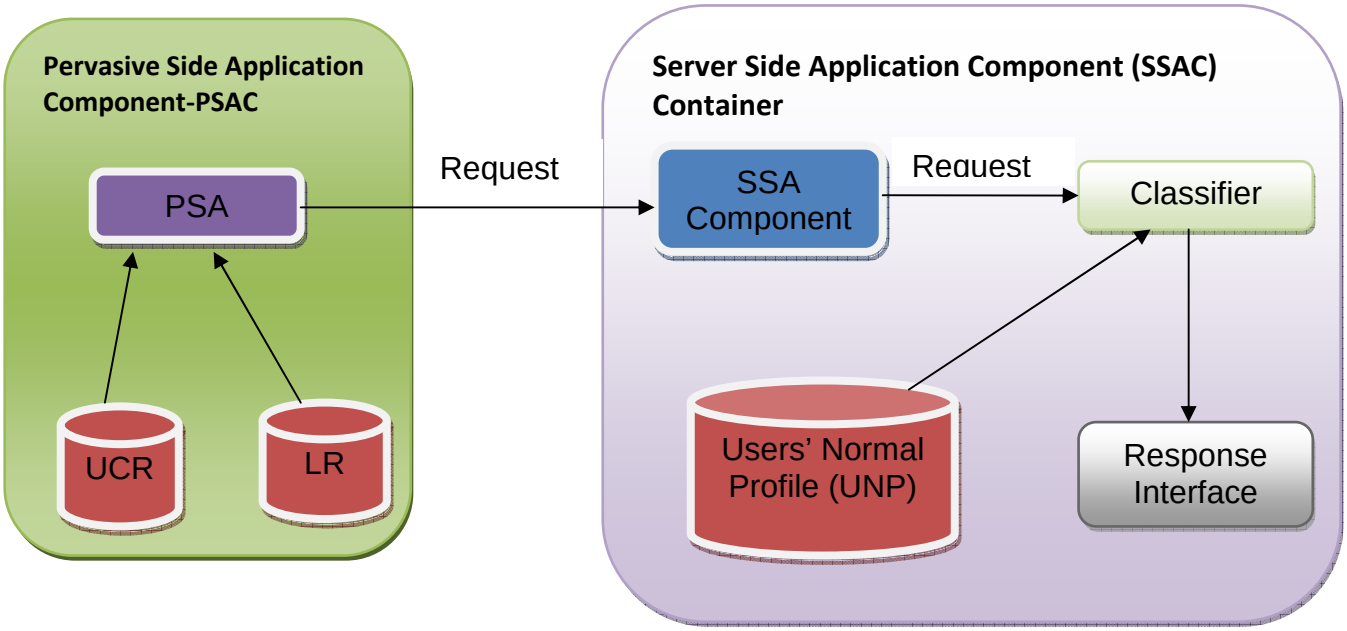

Figure 6: Architecture of the anomaly detector

3.2. The Server Side Application Components (SSAC) Container

This component is the major one with respect to functionalities. Before going to the detection phase, classifying each request as normal or anomalous, we have to construct the basis for our classification. The phase for constructing this basis for classification is the learning phase. In this phase, we depend on the data contained in the UAD. During learning, the UAD is consumed as an input and the Users Normal profile (UNP) is generated, which in turn is going to be used as an input for the classifier in the detection phase.

The SSACs, as depicted in figure 6 above, receive request from the PSAC. From the requests, the SSACs extract the required information; like the specific remote host address from which the request originated, the request type, read or write, data source, record identifier, user category, and location. All these information will be stored in the UAD database for the learning phase or passed directly to the classifier in the detection phase.

The classifier component classifies a specific request, passed to it from the SSACs, as normal or anomalous based on an already established collection of normal rules. If the classifier identifies a specific request as anomalous, it sends an alarm to the Response Interface Component (RIC) describing the remote host address and all the necessary information required for understanding the intrusion well.

4. Data mining for anomaly detection

A basic premise for intrusion detection is that when audit mechanisms are enabled to record system events, distinct evidence of legitimate activities and intrusions will be manifested in the audit data. Because of the sheer volume of audit data, both in the amount of audit records and in the number of system features (i.e., the fields describing the audit records), efficient and intelligent data analysis tools are required to discover the behavior of system activities.

Data mining generally refers to the process of extracting useful models from large stores of data [12]. The recent rapid development in data mining has made available a wide variety of algorithms, drawn from the fields of statistics, pattern recognition, machine learning, and databases. As described in [12], several types of algorithms are particularly useful for mining audit data:

Classification: maps a data item into one of several predefined categories. These algorithms normally output “classifiers,” for example, in the form of decision trees or rules. An ideal application in intrusion detection would be to gather sufficient “normal” and “abnormal” audit data for a user or a program, then apply a classification algorithm to learn a classifier that can label or predict new unseen audit data as belonging to the normal class or the abnormal class;

Link analysis /association rule/: determines relations between fields in the database records.

Correlations of system features in audit data, for example, the correlation between command and argument in the shell command history data of a user, can serve as the basis for constructing normal usage profiles.

Sequence analysis: models sequential patterns. These algorithms can discover what time-based sequence of audit events frequently occur together. These frequent event patterns provide guidelines for incorporating temporal statistical measures into intrusion detection models. For example, patterns from audit data containing network-based denial-of-service (DoS) attacks suggest that several per-host and per-service measures should be included.

We have used two of the algorithms for our anomaly detector, classification rule mining and association rule mining. The association rule mining is implemented to identify association rules between the database attributes. It finds all rules in the database that satisfy some minimum support and minimum confidence constraints. The classifier deals with constructing accurate rules for classification.

As described in [13], we have used the integration of the association and classification rules. The integration is done by focusing on mining a special subset of association rules, called Class Association Rules, CARs. The special subset is determined by selecting rules whose right-hand-side are restricted to the classification class attribute. As a consequence of this merging, we have used the integrated framework, called associative classification.

Adaptation of the existing association rule mining algorithm to mine only the CARs is needed so as to reduce the number of rules generated, thus avoiding combinatorial explosion.

4.1. Association Rules

The goal of mining association rules is to derive multi-feature, attribute correlations from a

database table [12]. A simple yet interesting commercial application of the association rules algorithm is to determine what items are often purchased together by customers, and use that information to arrange store layout. Formally, given a set of records, where each record is a set of items, an association rule is an expression

$$X \rightarrow Y, \text{confidence, support}$$

X and Y are subsets of the items in a record, support is the percentage of records that contain X+Y, whereas confidence is support(X+Y)/support(X). For example, an association rule from the shell command history file, which is a stream of commands and their arguments, of a user is

$$trn \rightarrow rec.humor; [0.3, 0.1]$$

This indicates that 30% of the time when the user invokes trn, he or she is reading the news in rec.humor, and reading this newsgroup accounts for 10% of the activities recorded in his or her command history file. Here 0.3 is the confidence and 0.1 is the support.

The motivations for applying the association rules algorithm to audit data are:

- Audit data can be formatted into a database table where each row is an audit record and each column is a field, system feature, of the audit records;
- There is evidence that program executions and user activities exhibit frequent correlations among system features. For example, one of the reasons that ``program policies", which codify the access rights of privileged programs, are concise and capable to detect known attacks [12] is that the intended behavior of a program, e.g., read and write files from certain directories with specific permissions, is very consistent. These consistent behaviors can be captured in association rules;

- We can continuously merge the rules from a new run to the aggregate rule set, of all previous runs.

A formal definition for association rules is provided in [15] as let D = {T_1, T_2... T_n} be a set of n transactions and let I be a set of items, I = {i_1, i_2 . . . i_m}. Each transaction is a set of items, i.e. T_i is subset of I. An association rule is an implication of the form X →Y, where X and Y are proper subsets of I, and X and Y should be disjoint. In this case, X is called the antecedent and Y is called the consequent of the rule. In general, a set of items, such as X or Y, is called an itemset.

In this work, a transaction is activity/command of a medical staff in the pervasive medical system. Let P(X) be the probability of appearance of itemset X in D and let P(Y |X) be the conditional probability of appearance of itemset Y given itemset X appears. For an itemset X, subset of I, support(X) is defined as the fraction of transactions T_i, element of D, such that X is subset of T_i. That is, P(X) = support(X). The support of a rule X → Y is defined as support(X → Y) = P(X U Y).

An association rule X → Y has a measure of reliability called confidence(X → Y) defined as P(Y |X) = P(X U Y)/P (X) = support(X → Y)/support(X).

The standard problem of mining association rules is to find all rules whose metrics are equal to or greater than some specified minimum support and minimum confidence thresholds. A k-itemset with support above the minimum threshold is called frequent.

4.2. The Integrated Framework, associative classification

The associative classification is based on the assumptions, as indicated in [13], that the dataset is a normal relational table, which consists of N cases described by l distinct attributes. These N

cases have been classified into q known classes.

Let D be the dataset. Let I be the set of all items in D, and Y be the set of class labels. We say that the data case d, element of D, contains X, subset of I, if X is subset of d. A class association rule, CAR, is an implication of the form X → y, where X is subset of I and y is element of Y. A rule X → y holds in D with confidence c if c% of cases in D that contain X are labeled with class y.

The basic objectives of the associative classification are to:

- generate the complete set of CARs that satisfy the specified minimum support, called minsup, and minimum confidence, called minconf, constraints.
- build a classifier from the CARs

In our case, the dataset is collected from user requests. Rather than exploiting these requests directly, they are preprocessed, which primarily involves categorizing. For example, instead of using time-stamp of requests directly, they are categorized into am, pm, and nt (night).For the location information as well, instead of using exact location information, which may be collected from a GPS enabled PDA, they are categorized into areas like room12, room22, room32, office, laboratory, pharmacy, and other.

With regards to the classes which are used as the right-hand-side restrictions for the association rules, we have used two classes, normal and abnormal. The type of Intrusion detection used, in our case, is anomaly detection that requires maintaining normal operations to detect deviations from. As a result, we have used the normal class as our only right-hand-side restriction.

4.2.1. Generating the CARs

As indicated in [13], the Classification Based on Association (CBA) algorithm constitutes two

parts, the Rule Generator (CBA-RG) which is based on algorithm Apriori [14] for finding association rules, and a Classifier Builder (CBA-CB).

The key operation of CBA-RG, figure 7, is to find all ruleitems that have support above minsup. A ruleitem is of the form: <condset, y>, where condset is a set of items, y is element of Y is a class label. The support count of the condset, condsupCount, is the number of cases in D that contain the condset. The support count of the ruleitem, rulesupCount, is the number of cases in D that contain the condset and are labeled with class y. Each ruleitem basically represents a rule: condset → y, whose support is (rulesupCount / |D|) *100%, where |D| is the size of the dataset, and whose confidence is (rulesupCount / condsupCount)*100%.

F_1 = {large 1-ruleitems};
CAR_1 = genRules(F_1);
$prCAR_1$ = pruneRules(CAR_1);
for (k = 2; $F_{k-1} \neq \varnothing$; k++) **do**
    C_k = candidateGen(F_{k-1});
    for each data case d $\in$D **do**
        C_d = ruleSubset(C_k, d);
        for each candidate c $\in C_d$ **do**
            c.condsupCount++;
            if d.class = c.class **then** c.rulesupCount++;
        end
    end
    F_k = {c $\in C_k$ | c.rulesupCount $\geq$ minsup};
    CAR_k = genRules(F_k);
    $prCAR_k$ = pruneRules(CAR_k);
end
CARs = U_k CAR_k; prCARs = U_k prCARk;

Figure 7: The CBA-RG Algorithm

Ruleitems that satisfy minsup are called frequent ruleitems, while the rest are called infrequent ruleitems. For example, the following is a ruleitem:

$$< \{(A, 1), (B, 1)\}, (class, 1)>,$$

where A and B are attributes. If the support count of the condset {(A, 1), (B, 1)} is 3, the support count of the ruleitem is 2, and the total number of cases in D is 10, then the support of the ruleitem is 20%, and the confidence is 66.7%. If minsup is 10%, then the ruleitem satisfies the minsup criterion. We say it is frequent.

For all the ruleitems that have the same condset, the ruleitem with the highest confidence is chosen as the possible rule (PR) representing this set of ruleitems. If there are more than one ruleitem with the same highest confidence, we randomly select one ruleitem. For example, we have two ruleitems that have the same condset:

1. < {(A, 1), (B, 1)}, (class: 1)>.
2. < {(A, 1), (B, 1)}, (class: 2)>.

Assume the support count of the condset is 3. The support count of the first ruleitem is 2, and the second ruleitem is 1. Then, the confidence of ruleitem 1 is 66.7%, while the confidence of ruleitem 2 is 33.3% With these two ruleitems, we only produce one PR (assume |D| = 10):

$$(A, 1), (B, 1) \rightarrow (class, 1) \text{ [supt = 20\%, confd= 66.7\%]}$$

But, in our case, such situations can not happen. This is because, as we have described in the beginning of this section, we intend to generate rules belonging to normal class.

If the confidence is greater than minconf, we say the rule is accurate. The set of class association rules (CARs) thus consists of all the PRs that are both frequent and accurate.

4.2.2. The CBA-RG Algorithm

The CBA-RG algorithm generates all the frequent ruleitems by making multiple passes over the data. In the first pass, it counts the support of individual ruleitem and determines whether it is frequent. In each subsequent pass, it starts with the seed set of ruleitems found to be frequent in the previous pass. It uses this seed set to generate new possibly frequent ruleitems, called candidate ruleitems. The actual supports for these candidate ruleitems are calculated during the pass over the data. At the end of the pass, it determines which of the candidate ruleitems are actually frequent. From this set of frequent ruleitems, it produces the rules (CARs).

Let k-ruleitem denote a ruleitem whose condset has k items. Let F_k denote the set of frequent k-ruleitems. Each element of this set is of the following form:

<(condset, condsupCount), (y, rulesupCount)>.

C_k represents the set of candidate k-ruleitems.

The first three lines of the algorithm deal with describing what happens on the first pass. The genRules functionality generates rules based on the 1-frequent ruleitems.

The previous section has already indicated how the rule generation should proceed. $prCAR_1$ contains the pruned CAR_1. Pruning is an optional step in building rules. The function pruneRules uses the pessimistic error rate based pruning method [13]. It prunes a rule as follows: If rule r's pessimistic error rate is higher than the pessimistic error rate of rule r' (obtained by deleting one condition from the conditions of r), then rule r is pruned. This pruning can cut down the number of rules generated substantially.

For each subsequent pass, say pass k, the algorithm performs four major operations. First, the frequent ruleitems F_{k-1}, found in the $(k-1)^{th}$ pass, are used to generate the candidate ruleitems

C_k using the condidateGen function. It then scans the database and updates various support counts of the candidates in C_k. After those new frequent ruleitems have been identified to form F_k, the algorithm then produces the rules CAR_k using the genRules function. Finally, rule pruning is performed on these rules.

The candidateGen function is similar to the function Apriori-gen in algorithm Apriori [14]. It involves the below algorithm.

```
forall itemsets c ∈ C_k do
    forall (k-1)-subsets s of c do
        if (s ∉ F_{k-1}) then
            delete c from C_k
```

Figure 8: Generating candidate set

If, for example, the below set contains the set of itemsets in a 3-frequent ruleitems and it is required to generate C_4,

$$F_3 = \{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\},$$

After candidate generation: $\{\{1\ 2\ 3\ 4\}, \{1\ 3\ 4\ 5\}\}$.

After pruning: $\{1\ 2\ 3\ 4\}$.

The set containing $\{1\ 3\ 4\ 5\}$ is pruned from the candidate set because $\{1\ 4\ 5\}$ and $\{3\ 4\ 5\}$ are not found in F_3.

The ruleSubset function takes a set of candidate ruleitems C_k and a data case d to find all the ruleitems in C_k whose condsets are supported by d. This and the operations in the next three lines are also similar to those in algorithm Apriori [14]. The difference is that we need to increment the support counts of the condset and the ruleitem separately whereas in algorithm

Apriori only one count is updated. This allows us to compute the confidence of the ruleitem. The final set of class association rules is in CARs. Those remaining rules after pruning are in prCARs.

4.2.3. The CBA-CB Algorithm

This section presents the CBA-CB algorithm for building a classifier using CARs. To produce the best classifier out of the whole set of rules would involve evaluating all the possible subsets of it on the training data and selecting the subset with the right rule sequence that gives the least number of errors. There are 2^m such subsets, where m is the number of rules, which can be more than 10,000, not to mention different rule sequences. This is clearly infeasible. The algorithm we have used is a heuristic one, proposed in [13].Before presenting the algorithm, let us define a total order on the generated rules. This is used in selecting the rules for our classifier.

Definition: Given two rules, r_i and r_j, $r_i \succ r_j$, also called r_i precedes r_j or r_i has a higher precedence than r_j, if

1. the confidence of r_i is greater than that of r_j, or
2. their confidences are the same, but the support of r_i is greater than that of r_j, or
3. both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j.

Let R be the set of generated rules, CARs, and D the training data. The basic idea of the algorithm is to choose a set of high precedence rules in R to cover D.

Our classifier is of the following format: $<r_1, r_2 \ldots r_n$, default_class>, where $r_i \in R$, $r_a \succ r_b$ if b > a. default_class is the default class. In classifying an unseen case, the first rule that satisfies the

case will classify it. If there is no rule that applies to the case, it takes on the default class.

A naive version of the algorithm we have used for building such a classifier is shown in [13]. It has three steps:

Step 1: Sort the set of generated rules R according to the relation "$\succ$". This is to ensure that we will choose the highest precedence rules for our classifier.

Step 2: Select rules for the classifier from R following the sorted sequence. For each rule r, we go through D to find those cases covered by r, they satisfy the conditions of r. We mark r if it correctly classifies a case d. d.id is the unique identification number of d. If r can correctly classify at least one case, that is, if r is marked, it will be a potential rule in our classifier. Those cases it covers are then removed from D. A default class is also selected, the majority class in the remaining data, which means that if we stop selecting more rules for our classifier C this class will be the default class of C. We then compute and record the total number of errors that are made by the current C and the default class. This is the sum of the number of errors that have been made by all the selected rules in C and the number of errors to be made by the default class in the training data. When there is no rule or no training case left, the rule selection process is completed.

Step 3: Discard those rules in C that do not improve the accuracy of the classifier. The first rule at which there is the least number of errors recorded on D is the cutoff rule. All the rules after this rule can be discarded because they only produce more errors. The undiscarded rules and the default class of the last rule in C form our classifier.

This algorithm is simple but appears to be inefficient when the database is not resident in the

main memory because it needs o make many passes over the database.

4.3. The Data Structure description

The efficiency of frequent itemset mining algorithms, as indicated in [16] is determined mainly by three factors:

- the way candidates are generated,
- the data structure that is used
- the implementation details.

In this section, we will be discussing only about the data structure we have implemented. Central data structures of our algorithm are Set, List, and Map, which are collections in the Java Language [19].

A collection, sometimes called a container, is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. A collection framework is a unified architecture for representing and manipulating collections [19]. All collection frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms

are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

4.3.1. Set Collection

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Two Set instances are equal if they contain the same elements.

4.3.2. List Collection

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations like; positional access, manipulates elements based on their numerical position in the list, search, looks for a specified object in the list and returns its numerical position, iteration, extends Iterator semantics to take advantage of the list's sequential nature, and range-view, performs arbitrary range operations on the list.

4.3.3. Map Collection

A Map is one interface object that maps keys to values. It cannot contain duplicate keys, each key can map to at most one value. HashMap is a Map implementation that is the best performing out of the other Map collection implementations. Map objects store key/value pairs. You can find a value if you provide the key. Keys must be unique; you cannot store two values with the same key.

The Collection view methods allow a Map to be viewed as a Collection in these three ways;

keyset, the Set of keys contained in the Map, values, the Collection of values contained in the Map, and entrySet, the Set of key-value pairs contained in the Map. The values Collection is not a Set, because multiple keys can map to the same value. The Collection views provide the only means to iterate over a Map.

We have used the Set collection for maintaining collections of the datasets from the database. Each entry of our Set contains a List object. Each List object represents collection of cases for each distinct attributes in the database. This has great performance advantage since we can find all the items on the Set object rather than making multiple scans of the database. For our case, fetching the database onto the main memory by using a Set object is required because as pointed out in the CBA-CB algorithm section, the naïve algorithm becomes inefficient when the database is not resident in the main memory.

Map is the other important collection we have implemented. It is used to maintain our frequent ruleitems, F_k and rules, CAR_k. The key of the Map objects for F_k and CAR_k are Set objects representing our condset. Since it is not possible to have multiple keys, the keys can be maintained in a Set object.

The condsupCount and the rulesupCount for the ruleitems are maintained in a List object representing the values in the Map object. Since different keys can have similar condsupCount and rulesupCount combinations, we can not use Set object to represent the values. For a similar reason, the support and confidence for the rules are maintained in a List object for the CAR_k.

From the ruleitem representation described in the CBA-RG section, the class representation y is not included in our ruleitem since we are dealing with anomaly detection modeling. The rules generated are also considered to represent rules for normalcy since, as described in the introduction section; anomaly detection involves modeling normal activities only.

5. Prototype

We have developed a prototypical implementation to demonstrate that the architecture proposed is a feasible solution. We have already pointed out a number of implementation issues we have considered in the previous sections, concerning the PSAC and SSACs. In this section we will be a bit specific to our implementation.

5.1. PSAC

PSAC, Pervasive Side Application Component, resides on the pervasive device. In response to this, the application development for the pervasive device end involves languages which are specifically designed to such devices.

In our case, we have used J2ME, Java Mobile Edition, for developing the pervasive end application. This is primarily because J2ME provides a modular, scalable architecture to support a flexible deployment of Java technology to devices with diverse features and functions [20]. A J2ME “configuration” targets devices with a specific range of capabilities. A “profile” selects a configuration and a set of APIs (Application Programming Interfaces) to target a specific domain of applications. By selecting the best configuration and profile, application developers can produce a wide range of flexible applications. Since lightweight appliances do not need to support the entire Java 2 platform, their resource requirements (and therefore cost) will be reduced [20].

The PSAC is comprised of user interface components specifically adapted to the device to read inputs and display formatted outputs to the users. As figure 9 shows, we have used the sun Java Wireless Toolkit emulator to show what the user interfaces look like.

The PSA, in our case, is collection of MIDlets, Java application that uses the MIDP profile and

the CLDC configuration. CLDC is Connected, Limited Device Configuration and MIDP is Mobile Information Device Profile. Sample MIDlets are attached in Appendix A.

As described in the architecture section, the pervasive side of the architecture contains storage units for location, LR, and user category, UCR. J2ME supports persistence of application data through its Record Management System, RMS [20]. We have used the RMS to store data for location and user category. This is basically to be able to simulate multiple users on the move by selecting, randomly, user category and location when a request is sent to the server. The code for constructing and manipulating RMS can be found in the sample MIDlets included in Appendix A.

5.2. SSAC

It is in these components that most of our work is done. As pointed out in the architecture section, there are a number of components at the server side which are responsible to receive requests from the PSAC and respond accordingly. We have implemented the server components using Servlets and contained them in Apache Tomcat, a Servlet container. Sample Servlet codes are included in Appendix B. Since these components are responsible for receiving requests from the users and respond accordingly, they play a central role in constructing the UAD as well as for passing individual user requests for analysis, to be classified.

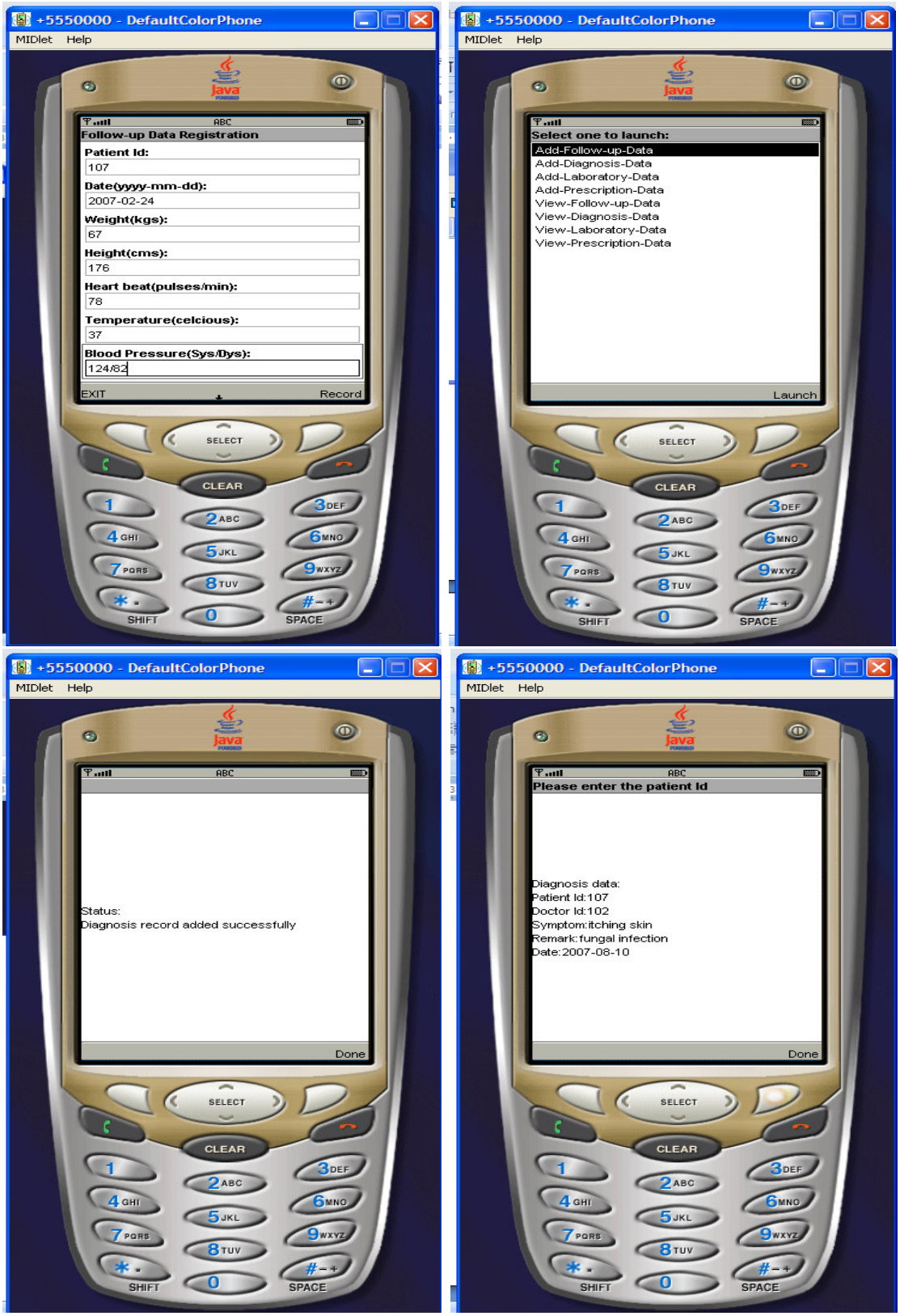

Figure 9: Sample screen shots from PSA

The UAD, for prototypical consumption, is constructed manually as shown in Appendix C. The dataset is constructed for doctor1 user category. It contains both normal and abnormal class categories. Based on this dataset, rules are generated for the normal class category cases in the dataset since our concern is building an anomaly model.

Rules generated based on the dataset constructed are included in Appendix E. For example, on rule id 76 we have the rule condition,

timeStamp:nt requestType:read location:room12

with support 15.00% and confidence 50.00%. This indicates the rule,

timeStamp:nt,requestType:read,location:room12→normal[supt=15.00%,confd=50.00%].

This rule means that 15.00% of cases in the dataset contain timeStamp:nt, requestType:read, location:room12 and are labeled normal and 50.00% of the cases in the dataset that contain timeStamp:nt, requestType:read, location:room12 are labeled with normal class category.

All the rules included in the appendix are not selected based on minimum confidence and support. For the precision of the classifier which is based on the rules for normalcy, confidence percentages could be fixed with values more than 90%.

Based on the rules generated, our classifier will try to discriminate, based on the profile maintained for the specific user category, each request received from the different users as normal or abnormal. In fact, the classifier itself needs training phase to improve precision of classification.

In its training phase, the classifier, shown in Appendix F, should first order the rules generated based on the conditions provided in section 4.2.3. To reduce potential error of the rules, the classifier should be trained on a well developed training dataset. For our case, we have applied

the same dataset we used to generate the rules and got the rules with id 35 and 8 selected as classifiers with error counts 0 and 5 respectively.

6. Conclusion

This paper has tried to show the possibility of constructing anomaly detection in a medical pervasive environment for legitimate users. Due to the sensitiveness of medical records, it is worthy not only to protect medical records from ouside attacks, but also from insiders who try to abuse thie privileges.

As can be seen from the rules generated, for representing normalcy, are very clear and concise for classification purpose, Appendix E. The rules generated are also ample and we can select any minimum confidence and support values to set the precision for our classifier.

What we can conclude from this work is that maintaining user activity information for intrusion detection can play a great role in protecting our system from insider attack. And one way to accomplish this is through maintaining associations between selected attributes to represent normalcy, which is required for classification purpose.

7. Future Work

In this research, our aim is to study the feasibility of modeling normalcy in application usage of medical staff. As can be seen from the content of the research, we have not used dataset collected from real working places. To identify the efficiency of the model it requires that we do so, collect a real working environment dataset, which we have left as future work.

In addition, in most cases, application specific datasets, especially representing legitimate users activity, is difficult to find. In our case, the assumption is to collect this dataset from the real working environment. A problem we can face in such case can be the data that we collect may

not be free from anomalous activities which in effect lead us generating anomalous rules as normal. This can be avoided by integrating noise resistant rule generation which is also left out as a future work.

8. References

[1] Computer Security Threat Monitoring and Surveillance

J. P. Anderson, James P. Anderson Co., Fort Washington, Pa., April 1980.

[2] An Intrusion-Detection Model

D. E. Denning, IEEE Transactions on Software Engineering, February 1987.

[3] The Trouble with Login – User Authentication and Medical Cooperation

Jakob E. Bardram, Centre for Pervasive Computing, Department of Computer Science, University of Aarhus, Denmark

[4] Nethost-sensor: A Novel Concept in Intrusion Detection Systems.

Abimbola, A., Merabti, M., Qi, S. (2003). 8th IEEE International Symposium on Computers and Communications, June 2003. (Pages 232-237).

[5] Security in Pervasive Computing.

Frank Stajano. In Proceedings of the 1st International Conference on Security in Pervasive Computing, Germany 2003, Pages 285-289, Springer-Verlag Berlin Heidelberg 2004.

[6] An Intrusion Detection Model

Denning D IEEE Transaction on Software Engineering, 1987. Pages 222-232.

[7] S. Forrest, A. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for UNIX processes.

In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages120-128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[8] "Bro: A System for Detecting Network Intruders in Real-Time.

Paxson, Vern. (Lawrence Berkeley National Laboratory). Proceedings of 7th USENIX Security Symposium. San Antonio, TX, January 1998.

[9] EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances.

In Proceedings of the 20th National Information Systems Security Conference, pages 353-365, Baltimore, Maryland, USA, 7-10 October 1997. NIST, National Institute of Standards and Technology/National Computer Security Center.

[10] NetSTAT: A Network-Based Intrusion Detection Approach.

Vigna, Giovanni & Kemmerer, Richard A. (University of California, Santa Barbara). Proceedings of the 14th Annual Computer Security Applications Conference. Scottsdale, AZ, Dec. 1998

[11] Security in Pervasive Computing.

Frank Stajano (University of Cambridge). Security in Pervasive Computing-First International Conference, Boppard, Germany, March 2003.

[12] A Framework for Constructing Features and Models for Intrusion Detection Systems.

Wenke Lee, Georgia Institute of Technology, and Salvatore J. Stolfo, Columbia University. In Proceedings of the 1999 IEEE Symposium on Security and Privacy and the Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2001.

[13] Integrating Classification and Association Rule Mining

Bing Liu, Wynne Hsu, and Yiming Ma. Department of Information Systems and Computer Science, National University of Singapore. American Association for Artificial Intelligence, 1998

[14] Mining association rules between sets of items in large databases.

Agrawal, R., Imielinski, T., AND Swami, A. 1993. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, P. Buneman and S. Jajodia, Eds. ACM Press, New York, NY, 207–216.

[15] Comparing Association Rules and Decision Trees for Disease Prediction.

Carlos Ordonez, University of Houston, Houston, TX, USA. ACM Publication, 2006.

[16] Fast Algorithm for Mining Association Rules

M.H.Margahny and A.A.Mitwaly, Dept. of Computer Science, Faculty of Computers and Information, Assuit University, Egypt. AIML 05 Conference, 19-21 December 2005, CICC, Cairo, Egypt

[17] Defending Against a large Scale Denial-of-Service Attack

Suhail Mohiuddin, Shlomo Hershkop, Rahul Bhan, and Sal Stolfo, Department of Computer Science, Columbia University, New York. In the proceedings of the IEEE Workshop on Information Assurance and Security., United States Military Academy, West Point, NY, 17-19 June 2002.

[18] A Dynamic Data Mining Technique for Intrusion Detection Systems

LTC Bruce D. Caulkins USA, Department of Modeling and Simulation, University of Central Florida. Joohan Lee, Computer Science Department, University of Central Florida. Morgan Wang, Department of Statistics and Actuarial Science, University of Central Florida. In the Proceedings of the 43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA.

[19] A Java Tutorial, a practical guide for programmers

Sun Microsystems, Inc., 1995-2005

[20] Wireless J2ME™ Platform Programming

Vartan Piroumian. Publisher: Prentice Hall PTR, Pub Date: March 25, 2002

ISBN: 0-13-044914-8, Pages: 400

Appendix A: Sample MIDlets from the PSA

A.1: MIDlet used to view diagnosis information

```
package MobileMedicalSystem;
import java.io.*;
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.microedition.rms.*;
//import java.util.Random;
public class MobileMedicalSystem extends MIDlet implements CommandListener {
    private String url ="http://localhost:8080/diagnosis/sqltest1";
  private Display display;
    private Command exit = new Command("EXIT", Command.EXIT, 1);;
    private Command connect = new Command("Connect", Command.SCREEN, 1);
    private TextField id;
    private Form menu;
    DB db;
    private RecordStore recordStore;
    private Random r=new Random();
    public MobileMedicalSystem() throws Exception {
            display = Display.getDisplay(this);
 }
 public void startApp() {
       displayMenu();
    }
 public void displayMenu() {
       menu = new Form("Pateint Diagnosis Information");
       id = new TextField("Enter patient ID: ","",30,TextField.ANY );
       menu.append(id);
       menu.addCommand(exit);
       menu.addCommand(connect);
       menu.setCommandListener(this);
       display.setCurrent(menu);    }
```

```
public void pauseApp() {}
public void destroyApp(boolean unconditional) {}
public void commandAction(Command command, Displayable screen) {
   if (command == exit) {
      destroyApp(false);
      notifyDestroyed();
   } else if (command == connect) {
      db  = new DB(this);
      db.start();
      db.connectDb(id.getString());
   }
}
public class DB implements Runnable {
   MobileMedicalSystem midlet;
   private Display display;
   String idn;
   public DB(MobileMedicalSystem midlet) {
      this.midlet = midlet;
      display = Display.getDisplay(midlet);
   }
   public void start() {
      Thread t = new Thread(this);
      t.start();
   }
   public void run() {
      StringBuffer sb = new StringBuffer();
      String randomStaff=new String();
      String randomLocation=new String();
      //record_add();
      //deleteAllRecords();
      try {
         HttpConnection c = (HttpConnection) Connector.open(url);
         c.setRequestProperty(
           "User-Agent","Profile/MIDP-1.0, Configuration/CLDC-1.0");
```

```
        c.setRequestProperty("Content-Language","en-US");
        c.setRequestMethod(HttpConnection.POST);
        DataOutputStream os =
              (DataOutputStream)c.openDataOutputStream();
        os.writeUTF(idn.trim());
        randomStaff=recordReadStaff();
        randomLocation=recordReadLocation();
        os.writeUTF(randomLocation);
        os.writeUTF(randomStaff);
        os.flush();
        os.close();
        / Get the response from the servlet page.
        DataInputStream is =(DataInputStream)c.openDataInputStream();
        //is = c.openInputStream();
        int ch;
        sb = new StringBuffer();
        while ((ch = is.read()) != -1) {
           sb.append((char)ch);
        }
        showAlert(sb.toString());
        is.close();
        c.close();
    } catch (Exception e) {
        showAlert(e.getMessage());
    }
}
    /* This method takes the patient id from the user and passes to servlet */
public void connectDb(String idn) {
    this.idn = idn;
}
/* Display Error On screen*/
private void showAlert(String err) {
    Alert a = new Alert("Please enter the patient Id");
    a.setString(err);
    a.setTimeout(Alert.FOREVER);
```

```
    display.setCurrent(a);
  }
 } ;
void record_add(){
            String locations[] =
            {"room12","room22","room32","office","laboratory","pharmacy","other"};
            //String medicalStaff[]=
            {"doctor1","doctor2","doctor3","pharmacist","nurse","labtechnician"};
        try{
    recordStore=RecordStore.openRecordStore("locations",true);
    //recordStore=RecordStore.openRecordStore("medicalStaff",true);
  }catch(RecordStoreException rse){
      rse.printStackTrace();
  }
       for(int i=0;i<7;i++){
                    ByteArrayOutputStream baos = new ByteArrayOutputStream();
                    DataOutputStream dos = new DataOutputStream(baos);
                    try{
                        dos.writeUTF(locations[i]);
                        //dos.writeUTF(medicalStaff[i]);
                        //dos.writeUTF(phone);
                    }catch (IOException ioe){
                        ioe.printStackTrace();
                    }
                    try{
                        int id = recordStore.addRecord(baos.toByteArray(), 0,
                                baos.toByteArray().length);
                    }catch(RecordStoreException rse){
                        rse.printStackTrace();
                    }
            }
            try{
                    recordStore.closeRecordStore();
            }catch (RecordStoreNotOpenException rsno){
            }catch(RecordStoreException rse){
```

```
                }
        }
String recordReadStaff(){
    int numRecords=0;
    String name[] = {"empty","empty","empty","empty","empty","empty"};
    int i=0;
    try{
        recordStore=RecordStore.openRecordStore("medicalStaff",false);
       //num=recordStore.getNumRecords();
    }catch(RecordStoreNotOpenException rsnfe){
          rsnfe.printStackTrace();
    }catch(RecordStoreException rse){
         rse.printStackTrace();
    }
        try{
        RecordEnumeration re = recordStore.enumerateRecords(null,null,false);
        if (re.numRecords() > 0)
        {
                ByteArrayInputStream bais = null;
                DataInputStream dis = null;
                while (re.hasNextElement())
                {
                byte[] record = re.nextRecord();
                bais = new ByteArrayInputStream(record);
                dis = new DataInputStream(bais);
                String strRec = new String(record);
                name[i] = dis.readUTF();
                //append(name, null);
                i++;
                numRecords++;
                }
                i=r.nextInt(numRecords);
        }
        }catch (RecordStoreException re){
                re.printStackTrace();
```

```
        }catch (IOException ioe){
                ioe.printStackTrace();
        }
        try{
                recordStore.closeRecordStore();
        }catch (RecordStoreNotOpenException rsno){
        }catch(RecordStoreException rse){
        }
        return name[i];
}
  String recordReadLocation(){
     int numRecords=0;
     String name[] = {"empty","empty","empty","empty","empty","empty","empty"};
     int i=0;
     try{
        recordStore=RecordStore.openRecordStore("locations",false);
        //recordStore=RecordStore.openRecordStore("medicalStaff",false);
       //num=recordStore.getNumRecords();
     }catch(RecordStoreNotOpenException rsnfe){
          rsnfe.printStackTrace();
     }catch(RecordStoreException rse){
          rse.printStackTrace();
     }
        try
        {
        RecordEnumeration re = recordStore.enumerateRecords(null,null,false);
        if (re.numRecords() > 0)
        {
                ByteArrayInputStream bais = null;
                DataInputStream dis = null;
                while (re.hasNextElement())
                {
                byte[] record = re.nextRecord();
                bais = new ByteArrayInputStream(record);
                dis = new DataInputStream(bais);
```

```
                String strRec = new String(record);
                name[i] = dis.readUTF();
                //append(name, null);
                i++;
                numRecords++;
                }
                i=r.nextInt(numRecords);
        }
        }catch (RecordStoreException re){
                re.printStackTrace();
        }catch (IOException ioe){
                ioe.printStackTrace();
        }
        try{
                        recordStore.closeRecordStore();
        }catch (RecordStoreNotOpenException rsno){
        }catch(RecordStoreException rse){
        }
        return name[i];
}
  void deleteAllRecords()
        {
                try{
                        recordStore=RecordStore.openRecordStore("locations",false);
                        //recordStore=RecordStore.openRecordStore("medicalStaff",false);
       //num=recordStore.getNumRecords();
        RecordEnumeration re =recordStore.enumerateRecords(null, null, false);
                        while (re.hasNextElement())
                        {
                                int id = re.nextRecordId();
                                recordStore.deleteRecord(id);
                        }
                }catch (RecordStoreException rse){
                        rse.printStackTrace();
                }/*catch(RecordStoreNotOpenException rsnfe){
```

```
            rsnfe.printStackTrace();
        }*/
            }
  ////////////////////////RMS Methods/////////////////////////////////////////////////////////
  }
   ////////////////////////RMS Methods/////////////////////////////////////////////////////////
  }
```

A.2: MIDlet used to request add diagnosis information

```
package MobileMedicalSystem;
import java.io.*;
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.microedition.rms.*;

public class testMySQL5 extends MIDlet implements CommandListener {
    private String username;
    private String url =  "http://localhost:8080/adddiagnosis/sqltest1";
    private Display display;
    private Command exit = new Command("EXIT", Command.EXIT, 1);
    private Command Record = new Command("Record", Command.SCREEN, 1);
    private Form menu;
    private TextField tb1;
    private TextField tb2;
    private TextField tb3;
    private TextField tb4;
    private TextField tb5;
    DB db;
    private RecordStore recordStore;
    private Random r=new Random();
    public testMySQL5() throws Exception {
        display = Display.getDisplay(this);
    }
```

```
 public void startApp() {
      displayMenu();
  }
  public void displayMenu() {
      menu = new Form("Diagnosis Data Registration");
      tb1 = new TextField("Patient Id:","",30,TextField.ANY );
      tb2 = new TextField("Doctor Id:","",30,TextField.ANY);
     tb3 = new TextField("Symptom(s):","",30,TextField.ANY);
      tb4 = new TextField("Remark:","",30,TextField.ANY);
      tb5 = new TextField("Date(yyyy-mm-dd):","",30,TextField.ANY);
      menu.append(tb1);
      menu.append(tb2);
      menu.append(tb3);
      menu.append(tb4);
      menu.append(tb5);
      menu.addCommand(exit);
      menu.addCommand(Record);
      menu.setCommandListener(this);
      display.setCurrent(menu);
  }
  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}
  public void commandAction(Command command, Displayable screen) {
     if (command == exit) {
         destroyApp(false);
         notifyDestroyed();
      } else if (command == Record) {
         db  = new DB(this);
         db.start();
  db.connectDb(tb1.getString(),tb2.getString(),tb4.getString(),tb5.getString(),tb3.getString());
  }
}
   public class DB implements Runnable {
      testMySQL5 midlet;
      private Display display;
```

```
String PID;
String DrID;
String Symptom;
    String Remark;
    String DiagDate;
public DB(testMySQL5 midlet) {
   this.midlet = midlet;
   display = Display.getDisplay(midlet);
}
public void start() {
   Thread t = new Thread(this);
   t.start();
}
public void run() {
   StringBuffer sb = new StringBuffer();
   String randomStaff=new String();
   String randomLocation=new String();
   //record_add();
   //deleteAllRecords();
   try {
      HttpConnection c = (HttpConnection) Connector.open(url);
      c.setRequestProperty(
        "User-Agent","Profile/MIDP-1.0, Configuration/CLDC-1.0");
      c.setRequestProperty("Content-Language","en-US");
      c.setRequestMethod(HttpConnection.POST);
      DataOutputStream os = (DataOutputStream)c.openDataOutputStream();
      os.writeUTF(PID.trim());
      os.writeUTF(DrID.trim());
     os.writeUTF(Remark.trim());
     os.writeUTF(DiagDate.trim());
     os.writeUTF(Symptom.trim());
     randomStaff=recordReadStaff();
      randomLocation=recordReadLocation();
      os.writeUTF(randomLocation);
      os.writeUTF(randomStaff);
```

```
            os.flush();
              os.close();
              DataInputStream is =(DataInputStream)c.openDataInputStream();
              int ch;
              sb = new StringBuffer();
              while ((ch = is.read()) != -1) {
                 sb.append((char)ch);
              }
              showAlert(sb.toString());
              is.close();
              c.close();
            } catch (Exception e) {
              showAlert(e.getMessage());
            }
        }
        public void connectDb(String PID,String DrID,String Remark,String DiagDate,String Symptom) {
            this.PID = PID;
            this.DrID = DrID;
            this.Symptom = Symptom;
                this.Remark = Remark;
                this.DiagDate = DiagDate;
        }
        private void showAlert(String err) {
            Alert a = new Alert("");
            a.setString(err);
            a.setTimeout(Alert.FOREVER);
            display.setCurrent(a);
        }
    };
    void record_add(){
                        String locations[] =
                        {"room12","room22","room32","office","laboratory","pharmacy","other"};
                        //String medicalStaff[]=
                        {"doctor1","doctor2","doctor3","pharmacist","nurse","labtechnician"};
                  try{
```

```
        recordStore=RecordStore.openRecordStore("locations",true);
        //recordStore=RecordStore.openRecordStore("medicalStaff",true);
    }catch(RecordStoreException rse){
          rse.printStackTrace();
    }
           for(int i=0;i<7;i++){
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        DataOutputStream dos = new DataOutputStream(baos);
                        try{
                                dos.writeUTF(locations[i]);
                                //dos.writeUTF(medicalStaff[i]);
                                //dos.writeUTF(phone);
                        }catch (IOException ioe){
                                ioe.printStackTrace();
                        }
                        try{
                                int id = recordStore.addRecord(baos.toByteArray(), 0,
                                        baos.toByteArray().length);
                        }catch(RecordStoreException rse){
                                rse.printStackTrace();
                        }
                }
                try{
                        recordStore.closeRecordStore();
                }catch (RecordStoreNotOpenException rsno){
                }catch(RecordStoreException rse){
                }
        }

String recordReadStaff(){
    int numRecords=0;
    String name[] = {"empty","empty","empty","empty","empty","empty"};
    int i=0;
    try{
        recordStore=RecordStore.openRecordStore("medicalStaff",false);
```

```
   //num=recordStore.getNumRecords();
}catch(RecordStoreNotOpenException rsnfe){
       rsnfe.printStackTrace();
}catch(RecordStoreException rse){
      rse.printStackTrace();
}
    try{
    RecordEnumeration re = recordStore.enumerateRecords(null,null,false);
    if (re.numRecords() > 0)
    {
                ByteArrayInputStream bais = null;
                DataInputStream dis = null;
                while (re.hasNextElement())
                {
                byte[] record = re.nextRecord();
                bais = new ByteArrayInputStream(record);
                dis = new DataInputStream(bais);
                String strRec = new String(record);
                name[i] = dis.readUTF();
                //append(name, null);
                i++;
                numRecords++;
                }
                i=r.nextInt(numRecords);
    }
    }catch (RecordStoreException re){
                re.printStackTrace();
    }catch (IOException ioe){
                ioe.printStackTrace();
    }
    try{
                recordStore.closeRecordStore();
    }catch (RecordStoreNotOpenException rsno){
    }catch(RecordStoreException rse){
    }
```

```
        return name[i];
}
 String recordReadLocation(){
      int numRecords=0;
      String name[] = {"empty","empty","empty","empty","empty","empty","empty"};
      int i=0;
      try{
          recordStore=RecordStore.openRecordStore("locations",false);
          //recordStore=RecordStore.openRecordStore("medicalStaff",false);
         //num=recordStore.getNumRecords();
      }catch(RecordStoreNotOpenException rsnfe){
            rsnfe.printStackTrace();
      }catch(RecordStoreException rse){
           rse.printStackTrace();
      }
          try
          {
          RecordEnumeration re = recordStore.enumerateRecords(null,null,false);
          if (re.numRecords() > 0)
          {
                    ByteArrayInputStream bais = null;
                    DataInputStream dis = null;
                    while (re.hasNextElement())
                    {
                    byte[] record = re.nextRecord();
                    bais = new ByteArrayInputStream(record);
                    dis = new DataInputStream(bais);
                    String strRec = new String(record);
                    name[i] = dis.readUTF();
                    //append(name, null);
                    i++;
                    numRecords++;
                    }
                    i=r.nextInt(numRecords);
          }
```

```
        }catch (RecordStoreException re){
                re.printStackTrace();
        }catch (IOException ioe){
                ioe.printStackTrace();
        }
        try{
                        recordStore.closeRecordStore();
        }catch (RecordStoreNotOpenException rsno){
        }catch(RecordStoreException rse){
        }
        return name[i];
}
 void deleteAllRecords()
        {
                try{
                        recordStore=RecordStore.openRecordStore("locations",false);
                        //recordStore=RecordStore.openRecordStore("medicalStaff",false);
       //num=recordStore.getNumRecords();
        RecordEnumeration re =recordStore.enumerateRecords(null, null, false);
                        while (re.hasNextElement())
                        {
                                int id = re.nextRecordId();
                                recordStore.deleteRecord(id);
                        }
                }catch (RecordStoreException rse){
                        rse.printStackTrace();
                }
        }
 }
}
```

Appendix B: Sample Servlets from the SSAC Container

B.1: Servlet for accepting diagnosis view request

```
import java.io.*;
import java.text.*;
import java.lang.*;
import java.util.*;
import javax.servlet.*;
//import javax.ServletRequest.*;
import javax.servlet.http.*;
//import javax.servlet.http.ServletRequest;
import java.sql.*;

public class getConnection extends HttpServlet {
  private String clientAddr;
  private String clientName;
  private String requestType;
  private String dataSource;
  private String recordId;
  //private String timeStamp;
  private String location;
  private String message;
   public void init() {
   }
  public void doPost(HttpServletRequest request ,
        HttpServletResponse response)  throws ServletException,
        IOException {
      message ="Diagnosis data:\n";
      DataInputStream in = new DataInputStream(
           (InputStream)request.getInputStream());
      String id = in.readUTF();
      location=in.readUTF();
       clientName=in.readUTF();
      clientAddr=(request.getRemoteAddr()).toString();
      /clientName=(request.getRemoteHost()).toString();
```

```
     requestType="read";
     dataSource="diagnosis";
     recordId=id;
    java.sql.Date dt= new java.sql.Date(time);
    java.sql.Time tm= new java.sql.Time(time);
    timeStamp=dt.toString()+" "+tm.toString();*/
   String values="";
    try {
     values=connect(id.toLowerCase().trim());
    message += values;
 } catch (Throwable t) {
        message += "200 " + t.toString();
    }
    response.setContentType("text/plain");
    response.setContentLength(message.length());
    PrintWriter out = response.getWriter();
    out.println(message);
    in.close();
    out.close();
    out.flush();
}
public void doGet(HttpServletRequest request,
        HttpServletResponse response, FilterChain chain) throws ServletException,IOException {

    doPost(request,response);
}
/* This method connects to MYSQL database*/
private String connect(String id)
throws Exception {

      String values="";
          int idnum=Integer.parseInt(id);
    // Establish a JDBC connection to the MYSQL database server.
    //Class.forName("org.gjt.mm.mysql.Driver");
    Class.forName("com.mysql.jdbc.Driver").newInstance();
```

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/patient","root","passme");
      Statement st= conn.createStatement();
      String sql = "";
     sql = "SELECT PID, DrID, Symptom,Remark,DiagDate FROM diagnosis WHERE PID="+idnum;
     ResultSet rsList = st.executeQuery( sql );
    if( rsList != null )
    {
            while( rsList.next() )
                  {
                        //depends on your table you are accessing
                  values=values+"Patient Id:"+rsList.getString( "PID" )+"\n"+"Doctor
Id:"+rsList.getString("DrID")+"\n"+"Symptom:"+rsList.getString(
"Symptom")+"\n"+"Remark:"+rsList.getString("Remark")+"\n"+"Date:"+rsList.getString("DiagDate");
            }
                  rsList.close();
          }
          conn.close();
          conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/UserRequestDump","root","passme");
          sql="INSERT INTO tblUserRequest
(clientAddr,clientName,requestType,dataSource,recordId,location) VALUES('" + clientAddr + "','"
+clientName +
                  "','" + requestType + "','" +dataSource + "','" + recordId + "','" + location + "')";
          st= conn.createStatement();
          st.executeUpdate(sql);
          conn.close();
          return values;
          }

}
```

B.2: Servlet for accepting diagnosis add request

```
import java.io.*;
import java.text.*;
```

```
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class getConnection extends HttpServlet {
   private String clientAddr;
  private String clientName;
  private String requestType;
  private String dataSource;
  private String recordId;
  private String location;
  public void init() { }
  public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
      DataInputStream in = new DataInputStream(
            (InputStream)request.getInputStream());
      String id = in.readUTF();
       location=in.readUTF();
       clientName=in.readUTF();
       clientAddr=(request.getRemoteAddr()).toString();
       //clientName=(request.getRemoteHost()).toString();
       requestType="read";
       dataSource="vital-sign";
       recordId=id;
       String values="";
       String message ="Patient's vital signs:\n";
      try {
            values=connect(id.toLowerCase().trim());
            message += values;
      } catch (Throwable t) {
         message += "200 " + t.toString();
      }
```

```
        response.setContentType("text/plain");
        response.setContentLength(message.length());
        PrintWriter out = response.getWriter();
        out.println(message);
        in.close();
        out.close();
        out.flush();
    }
        public void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,IOException {

        doPost(request,response);
    }
    /* This method connects to MYSQL database*/
    private String connect(String id)
    throws Exception {
         String values="";
             int idnum=Integer.parseInt(id);
        // Establish a JDBC connection to the MYSQL database server.
        //Class.forName("org.gjt.mm.mysql.Driver");
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost:3306/patient","root","passme");
        Statement st= conn.createStatement();
        String sql = "";
sql = "SELECT id, date, weight,height,hbeat,temperature,BP FROM vitalsigns WHERE id="+idnum;
       ResultSet rsList = st.executeQuery( sql );
      if( rsList != null )
          {
      while( rsList.next() )
{
     values=values+"Patient Id:"+rsList.getString( "id" )+"\n"+"Examined on:"+ rsList.getString("date")+
"\n"+"Wieght(kgs):"+rsList.getString( "weight")+
"\n"+"Height(cms):"+rsList.getString("height")+"\n"+"Heart Beat(pulses/min):" +
rsList.getString("hbeat")+ "\n"+ "Temperature(celcious):" + rsList.getString("temperature")
+"\n"+"Pressure(Dis/Sys):"+rsList.getString("BP");
```

```
    }
rsList.close();
}
conn.close();
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/UserRequestDump","root","passme");
sql="INSERT INTO tblUserRequest (clientAddr,clientName,requestType, dataSource ,recordId,location) VALUES('" + clientAddr + "','" +clientName + "','" + requestType + "','" +dataSource + "','" + recordId + "','" + location + "')";
st= conn.createStatement();
st.executeUpdate(sql);
conn.close();

return values;
}
}
```

Appendix C: Dataset used for rule generation

id	clientAddr	clientName	requestType	dataSource	recordId	timeStamp	location	classCat
146	127.0.0.1	doctor1	read	vital-signs	101	2007/06/12 21:10:59	room12	normal
147	127.0.0.1	doctor1	read	vital-signs	102	2007/06/12 21:11:59	room32	normal
148	127.0.0.1	doctor1	read	vital-signs	103	2007/06/12 21:12:59	room12	normal
149	127.0.0.1	doctor1	read	vital-signs	104	2007/06/12 21:13:59	room12	normal
150	127.0.0.1	doctor1	read	vital-signs	105	2007/06/12 21:14:59	room32	normal
156	127.0.0.1	doctor1	read	vital-signs	101	2007/06/12 9:12:20	room12	normal
157	127.0.0.1	doctor1	read	vital-signs	102	2007/06/12 9:13:20	room32	normal
158	127.0.0.1	doctor1	read	vital-signs	103	2007/06/12 9:14:20	room12	normal
159	127.0.0.1	doctor1	read	vital-signs	104	2007/06/12 9:15:20	room12	normal
160	127.0.0.1	doctor1	read	vital-signs	105	2007/06/12 9:16:20	room32	normal
161	127.0.0.1	doctor1	write	vital-signs	101	2007/06/14 9:47:38	room12	abnormal
162	127.0.0.1	doctor1	write	vital-signs	102	2007/06/14 9:48:38	room32	abnormal
163	127.0.0.1	doctor1	write	vital-signs	103	2007/06/14 9:49:38	room12	abnormal
164	127.0.0.1	doctor1	write	vital-signs	104	2007/06/14 9:50:38	room12	abnormal

165	127.0.0.1	doctor1	write	vital-signs	105	2007/06/14 9:50:38	room32	abnormal
166	127.0.0.1	doctor1	read	vital-signs	101	2007/06/15 19:55:58	room12	abnormal
167	127.0.0.1	doctor1	read	vital-signs	102	2007/06/15 19:56:58	room32	abnormal
168	127.0.0.1	doctor1	read	vital-signs	103	2007/06/15 19:57:58	room12	abnormal
169	127.0.0.1	doctor1	read	vital-signs	104	2007/06/15 19:58:58	room12	abnormal
170	127.0.0.1	doctor1	read	vital-signs	105	2007/06/15 19:59:58	room32	abnormal

Appendix D: The rule generating class

```
package anomalyrulegenerator;

import java.util.*;
import java.sql.*;

public class RuleGenerator {

   private DatabaseToSet dbts;
   private Set<List<String>> setOfList;
   private String val;
   private String timeRegion;
   public final double MINSUPPORT=20.0;
   public final double MINCONFIDENCE=90.0;
   public final int MINSUP=2;
   private char spcChar=' ';
   /** Creates a new instance of RuleGenerator */
   public RuleGenerator() {

      dbts=new DatabaseToSet();
      setOfList=dbts.generateSet();
   }
   public Map<String,List<Integer>> getCondRuleSupCount(){
      Map<String, List<Integer>> mRuleItem1 = new HashMap<String, List<Integer>>();
         for(List<String> lst : setOfList){
            for(int i=0;i<7;i++){
               val= lst.get(i);
              List<Integer> lFreq= mRuleItem1.get(val);
              Integer freq;
              if(lFreq==null){
                 lFreq=new ArrayList<Integer>();
                 freq=1;
                 lFreq.add(0,freq);
                 if(lst.get(7).equals("classCat:normal"))
                    lFreq.add(1,freq);
                 else lFreq.add(1,0);
```

```
            }
            else{
               freq=lFreq.get(0)+1;
               lFreq.set(0,freq);
               if(lst.get(7).equals("classCat:normal"))
                  if(lFreq.get(1)==0)
                     lFreq.set(1,1);
                  else{
                     freq=lFreq.get(1)+1;
                     lFreq.set(1,freq);
                  }
            }
            mRuleItem1.put(val, lFreq);
          }
       }
       mRuleItem1=extractFrequent(mRuleItem1);
    return mRuleItem1;
 }
 public Map<String,List<Integer>> extractFrequent(Map<String,List<Integer>> m){
    for(Iterator<List<Integer>> val=m.values().iterator();val.hasNext();){
       List<Integer> list=val.next();
       if(list.get(1)<MINSUP) val.remove();
    }
    return m;
 }
public Map<String,List<Integer>> candidateGen(Map<String,List<Integer>> m) {
   Map<String,List<Integer>> mRuleItemi=new HashMap<String,List<Integer>>();
   boolean f2=false;
   for (Iterator<String> i1=m.keySet().iterator();i1.hasNext();){
      String key1=i1.next();
      int lastSpcChar1=key1.lastIndexOf(spcChar);
      String partKey1;
      if(lastSpcChar1!=-1) partKey1=key1.substring(0,lastSpcChar1);
      else{
         partKey1=key1;
         f2=true;
      }
```

```
   String key2=new String();
   boolean done=false;
        for (Iterator<String> i2=m.keySet().iterator();i2.hasNext();){
          key2=i2.next();
          if(done){
          int lastSpcChar2=key2.lastIndexOf(spcChar);
          String partKey2;
           if(lastSpcChar2!=-1)partKey2=key2.substring(0,lastSpcChar2);
          else partKey2=key2;
          if(partKey1.equals(partKey2)){
             List<Integer> lst=new ArrayList<Integer>();
             lst.add(0,0);
             lst.add(1,0);
             mRuleItemi.put(key1+key2.substring(lastSpcChar2),lst);
          }
          else if(f2){
             //System.out.println(key1+" "+key2);
             List<Integer> lst=new ArrayList<Integer>();
             lst.add(0,0);
             lst.add(1,0);
             //System.out.println(key1+"--"+key2);
             mRuleItemi.put(key1+String.valueOf(spcChar)+key2,lst);
          }
          }
          else
             if(key1.equals(key2))done=true;
        }
}
////////////////pruning candidate keySet/////////////////
if(!f2){
for(Iterator<String> i=mRuleItemi.keySet().iterator();i.hasNext();){
   //String ruleItemMember[]=new String[10];
   String key=i.next();
   SortedSet<String> test=new TreeSet<String>();
   String array[]=key.split(String.valueOf(spcChar));
   for(int start=0;start<array.length;start++){
      test.add(array[start]);
```

```
        }
        //ruleItemMember[i]=key.substring(start);
        boolean frequent=true;
        for(int j=0;j<test.size()&&frequent;j++){
           frequent=false;
           SortedSet<String> subset1=new TreeSet<String>(test);
           //SortedSet<String> subset2=new TreeSet<String>();
           Iterator<String> iSet=test.iterator();
           String element=new String();
           for(int k=0;k<=j;k++)
              element=iSet.next();
           subset1.remove(element);
           boolean frequentSubset=false;
           Iterator<String> i1=m.keySet().iterator();
           while(!frequentSubset&&i1.hasNext()){
              key=i1.next();
              String keyArray[]=key.split(String.valueOf(spcChar));
              SortedSet<String> setKey=new TreeSet<String>();
              for(int l=0;l<keyArray.length;l++)
                 setKey.add(keyArray[l]);
              if(setKey.equals(subset1)) frequentSubset=true;
           }
           if(frequentSubset)frequent=true;
        }
        if(!frequent)i.remove();
     }
  }
     ////////////////attaching condsupCount and rulesupCount////////////////
     for(Map.Entry<String,List<Integer>> e : mRuleItemi.entrySet()){
        for(List<String> cases : setOfList){
           String item=e.getKey();
           String stringItem[]=item.split(String.valueOf(spcChar));
           boolean found=true;
           int i=0;
           while(i<stringItem.length&&found){
              found=false;
              if(cases.contains(stringItem[i])) found=true;
```

```
                i++;
            }
            if(found){
                int freq=e.getValue().get(0);
                e.getValue().set(0,freq+1);
                if(cases.contains("classCat:normal")){
                    freq=e.getValue().get(1);
                    e.getValue().set(1,freq+1);
                }
            }
        }
    }
    mRuleItemi=extractFrequent(mRuleItemi);
     return mRuleItemi;
}
  public Map<String,List<Double>> genRules(Map<String,List<Integer>> m){
     Map<String,List<Double>> mCAR=new HashMap<String,List<Double>>();
     for(Map.Entry<String,List<Integer>> e : m.entrySet()){
        List<Double> lstDouble=new ArrayList<Double>();
        List<Integer> lstInteger=e.getValue();
        lstDouble.add(round(((double)lstInteger.get(1)/dbts.datasetSize)*100));
        lstDouble.add(round(((double)lstInteger.get(1)/lstInteger.get(0))*100));
        mCAR.put(e.getKey(),lstDouble);
     }
     return mCAR;
  }
  public double round(double d){
      d*=100;
      d+=0.5;
      long l=(long)Math.floor(d);
      return ((double)l/100);
   }
}
```

Appendix E: Rules for classifying normalcy, generated based on the dataset on Appendix C

id	ruleCondition	support	confidence
1	clientName:doctor1	50	50
2	recordId:102	10	50
3	clientAddr:127.0.0.1	50	50
4	recordId:103	10	50
5	timeStamp:nt	25	50
6	recordId:104	10	50
7	location:room32	20	50
8	requestType:read	50	66.67
9	recordId:101	10	50
10	recordId:105	10	50
11	dataSource:vital-signs	50	50
12	timeStamp:am	25	50
13	location:room12	30	50
14	location:room32 dataSource:vital-signs	20	50

15	location:room32 requestType:read	20	66.67
16	requestType:read recordId:101	10	66.67
17	clientName:doctor1 recordId:105	10	50
18	recordId:101 location:room12	10	50
19	clientAddr:127.0.0.1 timeStamp:nt	25	50
20	clientName:doctor1 requestType:read	50	66.67
21	clientName:doctor1 timeStamp:nt	25	50
22	clientAddr:127.0.0.1 location:room12	30	50
23	recordId:103 dataSource:vital-signs	10	50
24	clientName:doctor1 recordId:104	10	50
25	clientAddr:127.0.0.1 recordId:103	10	50
26	recordId:104 requestType:read	10	66.67
27	recordId:105 dataSource:vital-signs	10	50
28	requestType:read location:room12	30	66.67
29	requestType:read dataSource:vital-signs	50	66.67
30	location:room32 recordId:105	10	50
31	timeStamp:nt requestType:read	25	50
32	clientName:doctor1 timeStamp:am	25	50

33	dataSource:vital-signs timeStamp:am	25	50
34	clientName:doctor1 location:room12	30	50
35	requestType:read timeStamp:am	25	100
36	recordId:104 location:room12	10	50
37	recordId:103 requestType:read	10	66.67
38	recordId:102 clientAddr:127.0.0.1	10	50
39	location:room32 timeStamp:am	10	50
40	dataSource:vital-signs location:room12	30	50
41	recordId:103 location:room12	10	50
42	clientAddr:127.0.0.1 location:room32	20	50
43	clientAddr:127.0.0.1 timeStamp:am	25	50
44	timeStamp:nt dataSource:vital-signs	25	50
45	clientAddr:127.0.0.1 recordId:101	10	50
46	clientAddr:127.0.0.1 recordId:104	10	50
47	timeStamp:am location:room12	15	50
48	clientAddr:127.0.0.1 requestType:read	50	66.67
49	clientName:doctor1 location:room32	20	50

50	clientAddr:127.0.0.1 dataSource:vital-signs	50	50
51	timeStamp:nt location:room32	10	50
52	recordId:102 requestType:read	10	66.67
53	clientName:doctor1 recordId:102	10	50
54	clientName:doctor1 clientAddr:127.0.0.1	50	50
55	clientName:doctor1 recordId:101	10	50
56	clientAddr:127.0.0.1 recordId:105	10	50
57	requestType:read recordId:105	10	66.67
58	recordId:102 location:room32	10	50
59	clientName:doctor1 dataSource:vital-signs	50	50
60	recordId:101 dataSource:vital-signs	10	50
61	recordId:102 dataSource:vital-signs	10	50
62	timeStamp:nt location:room12	15	50
63	recordId:104 dataSource:vital-signs	10	50
64	clientName:doctor1 recordId:103	10	50
65	recordId:103 dataSource:vital-signs location:room12	10	50
66	requestType:read dataSource:vital-signs recordId:105	10	66.67
67	clientAddr:127.0.0.1 requestType:read recordId:105	10	66.67

68	clientName:doctor1 requestType:read location:room32	20	66.67
69	clientAddr:127.0.0.1 recordId:101 requestType:read	10	66.67
70	dataSource:vital-signs timeStamp:am location:room12	15	50
71	clientName:doctor1 location:room32 dataSource:vital-signs	20	50
72	requestType:read location:room12 timeStamp:am	15	100
73	clientAddr:127.0.0.1 location:room12 requestType:read	30	66.67
74	clientAddr:127.0.0.1 timeStamp:nt location:room12	15	50
75	clientName:doctor1 location:room32 clientAddr:127.0.0.1	20	50
76	timeStamp:nt requestType:read location:room12	15	50
77	recordId:101 location:room12 dataSource:vital-signs	10	50
78	requestType:read recordId:101 location:room12	10	66.67
79	recordId:102 clientAddr:127.0.0.1 dataSource:vital-signs	10	50
80	clientName:doctor1 location:room12 recordId:101	10	50
81	clientName:doctor1 recordId:105 clientAddr:127.0.0.1	10	50
82	clientAddr:127.0.0.1 location:room12 recordId:101	10	50
83	timeStamp:nt dataSource:vital-signs location:room12	15	50
84	clientAddr:127.0.0.1 timeStamp:am dataSource:vital-signs	25	50

85	clientAddr:127.0.0.1 location:room32 recordId:105	10	50
86	clientAddr:127.0.0.1 timeStamp:nt requestType:read	25	50
87	recordId:102 requestType:read location:room32	10	66.67
88	clientName:doctor1 location:room12 clientAddr:127.0.0.1	30	50
89	clientAddr:127.0.0.1 timeStamp:nt location:room32	10	50
90	requestType:read dataSource:vital-signs timeStamp:am	25	100
91	clientName:doctor1 recordId:105 requestType:read	10	66.67
92	recordId:102 clientAddr:127.0.0.1 location:room32	10	50
93	clientName:doctor1 requestType:read location:room12	30	66.67
94	recordId:104 requestType:read dataSource:vital-signs	10	66.67
95	recordId:104 location:room12 dataSource:vital-signs	10	50
96	clientName:doctor1 requestType:read recordId:104	10	66.67
97	clientName:doctor1 timeStamp:am dataSource:vital-signs	25	50
98	location:room32 requestType:read recordId:105	10	66.67
99	clientAddr:127.0.0.1 recordId:104 requestType:read	10	66.67
100	clientName:doctor1 recordId:102 dataSource:vital-signs	10	50
101	clientName:doctor1 recordId:105 dataSource:vital-signs	10	50
102	clientAddr:127.0.0.1 location:room12 dataSource:vital-signs	30	50

103	clientAddr:127.0.0.1 recordId:103 dataSource:vital-signs	10	50
104	recordId:102 requestType:read dataSource:vital-signs	10	66.67
105	recordId:103 requestType:read location:room12	10	66.67
106	location:room32 dataSource:vital-signs timeStamp:am	10	50
107	clientAddr:127.0.0.1 location:room12 timeStamp:am	15	50
108	requestType:read recordId:101 dataSource:vital-signs	10	66.67
109	clientName:doctor1 dataSource:vital-signs recordId:103	10	50
110	location:room32 requestType:read timeStamp:am	10	100
111	clientName:doctor1 recordId:105 location:room32	10	50
112	clientName:doctor1 requestType:read clientAddr:127.0.0.1	50	66.67
113	clientAddr:127.0.0.1 timeStamp:nt dataSource:vital-signs	25	50
114	clientName:doctor1 clientAddr:127.0.0.1 recordId:103	10	50
115	clientName:doctor1 requestType:read dataSource:vital-signs	50	66.67
116	clientName:doctor1 location:room12 recordId:103	10	50
117	clientAddr:127.0.0.1 location:room32 requestType:read	20	66.67
118	clientAddr:127.0.0.1 location:room12 recordId:103	10	50
119	clientName:doctor1 timeStamp:nt location:room32	10	50

120	recordId:103 dataSource:vital-signs requestType:read	10	66.67
121	clientAddr:127.0.0.1 location:room12 recordId:104	10	50
122	clientName:doctor1 location:room12 dataSource:vital-signs	30	50
123	clientName:doctor1 recordId:104 location:room12	10	50
124	location:room32 dataSource:vital-signs requestType:read	20	66.67
125	clientName:doctor1 requestType:read recordId:103	10	66.67
126	clientName:doctor1 timeStamp:nt dataSource:vital-signs	25	50
127	clientAddr:127.0.0.1 location:room32 dataSource:vital-signs	20	50
128	recordId:104 requestType:read location:room12	10	66.67
129	recordId:102 clientAddr:127.0.0.1 requestType:read	10	66.67
130	clientName:doctor1 requestType:read recordId:101	10	66.67
131	location:room32 dataSource:vital-signs recordId:105	10	50
132	clientName:doctor1 recordId:102 clientAddr:127.0.0.1	10	50
133	clientName:doctor1 requestType:read recordId:102	10	66.67
134	clientName:doctor1 timeStamp:nt clientAddr:127.0.0.1	25	50
135	clientName:doctor1 location:room32 recordId:102	10	50
136	timeStamp:nt requestType:read location:room32	10	50
137	clientAddr:127.0.0.1 requestType:read dataSource:vital-signs	50	66.67

138	clientName:doctor1 timeStamp:am clientAddr:127.0.0.1	25	50
139	clientName:doctor1 timeStamp:am location:room32	10	50
140	timeStamp:nt requestType:read dataSource:vital-signs	25	50
141	clientAddr:127.0.0.1 location:room32 timeStamp:am	10	50
142	clientAddr:127.0.0.1 timeStamp:am requestType:read	25	100
143	recordId:102 location:room32 dataSource:vital-signs	10	50
144	clientName:doctor1 clientAddr:127.0.0.1 dataSource:vital-signs	50	50
145	timeStamp:nt dataSource:vital-signs location:room32	10	50
146	clientName:doctor1 requestType:read timeStamp:nt	25	50
147	clientName:doctor1 requestType:read timeStamp:am	25	100
148	clientName:doctor1 recordId:104 dataSource:vital-signs	10	50
149	clientAddr:127.0.0.1 recordId:101 dataSource:vital-signs	10	50
150	clientName:doctor1 timeStamp:nt location:room12	15	50
151	clientAddr:127.0.0.1 recordId:104 dataSource:vital-signs	10	50
152	clientName:doctor1 clientAddr:127.0.0.1 recordId:101	10	50
153	clientAddr:127.0.0.1 recordId:103 requestType:read	10	66.67
154	clientName:doctor1 recordId:104 clientAddr:127.0.0.1	10	50
155	clientAddr:127.0.0.1 dataSource:vital-signs recordId:105	10	50

156	requestType:read location:room12 dataSource:vital-signs	30	66.67
157	clientName:doctor1 timeStamp:am location:room12	15	50
158	clientName:doctor1 recordId:101 dataSource:vital-signs	10	50
159	clientAddr:127.0.0.1 location:room12 recordId:101 dataSource:vital-signs	10	50
160	clientName:doctor1 requestType:read dataSource:vital-signs recordId:102	10	66.67
161	clientAddr:127.0.0.1 location:room32 requestType:read timeStamp:am	10	100
162	clientAddr:127.0.0.1 location:room12 requestType:read recordId:101	10	66.67
163	location:room32 dataSource:vital-signs timeStamp:am requestType:read	10	100
164	clientName:doctor1 requestType:read dataSource:vital-signs recordId:103	10	66.67
165	clientAddr:127.0.0.1 location:room32 recordId:105 dataSource:vital-signs	10	50
166	clientName:doctor1 clientAddr:127.0.0.1 dataSource:vital-signs recordId:101	10	50
167	clientName:doctor1 recordId:105 requestType:read location:room32	10	66.67
168	clientAddr:127.0.0.1 location:room12 requestType:read dataSource:vital-signs	30	66.67
169	clientName:doctor1 location:room12 recordId:101 dataSource:vital-signs	10	50
170	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1	30	66.67
171	clientAddr:127.0.0.1 location:room12 requestType:read recordId:103	10	66.67
172	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs	30	66.67

173	clientName:doctor1 requestType:read location:room12 recordId:104	10	66.67
174	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1	20	66.67
175	clientName:doctor1 requestType:read clientAddr:127.0.0.1 timeStamp:nt	25	50
176	clientName:doctor1 requestType:read location:room32 recordId:102	10	66.67
177	clientName:doctor1 recordId:104 location:room12 clientAddr:127.0.0.1	10	50
178	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 location:room32	10	50
179	clientName:doctor1 recordId:105 requestType:read dataSource:vital-signs	10	66.67
180	clientName:doctor1 location:room12 clientAddr:127.0.0.1 dataSource:vital-signs	30	50
181	clientAddr:127.0.0.1 location:room32 requestType:read dataSource:vital-signs	20	66.67
182	location:room32 dataSource:vital-signs requestType:read recordId:105	10	66.67
183	clientName:doctor1 requestType:read dataSource:vital-signs recordId:101	10	66.67
184	clientAddr:127.0.0.1 location:room32 dataSource:vital-signs timeStamp:am	10	50
185	clientName:doctor1 requestType:read location:room12 timeStamp:am	15	100
186	clientName:doctor1 timeStamp:am clientAddr:127.0.0.1 location:room12	15	50
187	clientName:doctor1 recordId:102 dataSource:vital-signs clientAddr:127.0.0.1	10	50
188	clientName:doctor1 timeStamp:am clientAddr:127.0.0.1 location:room32	10	50
189	timeStamp:nt requestType:read location:room12 dataSource:vital-signs	15	50

190	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 requestType:read	10	66.67
191	requestType:read recordId:101 location:room12 dataSource:vital-signs	10	66.67
192	clientName:doctor1 recordId:104 location:room12 dataSource:vital-signs	10	50
193	clientName:doctor1 location:room12 clientAddr:127.0.0.1 recordId:103	10	50
194	clientName:doctor1 recordId:105 dataSource:vital-signs location:room32	10	50
195	clientName:doctor1 timeStamp:am dataSource:vital-signs location:room32	10	50
196	recordId:102 clientAddr:127.0.0.1 dataSource:vital-signs location:room32	10	50
197	recordId:102 requestType:read location:room32 dataSource:vital-signs	10	66.67
198	clientAddr:127.0.0.1 location:room12 requestType:read recordId:104	10	66.67
199	clientAddr:127.0.0.1 timeStamp:nt requestType:read dataSource:vital-signs	25	50
200	clientName:doctor1 requestType:read recordId:104 dataSource:vital-signs	10	66.67
201	clientName:doctor1 requestType:read location:room12 recordId:101	10	66.67
202	clientAddr:127.0.0.1 recordId:103 dataSource:vital-signs requestType:read	10	66.67
203	clientName:doctor1 clientAddr:127.0.0.1 recordId:103 dataSource:vital-signs	10	50
204	clientAddr:127.0.0.1 requestType:read recordId:105 dataSource:vital-signs	10	66.67
205	clientName:doctor1 requestType:read location:room12 recordId:103	10	66.67
206	clientName:doctor1 timeStamp:am dataSource:vital-signs clientAddr:127.0.0.1	25	50
207	clientAddr:127.0.0.1 location:room12 dataSource:vital-signs recordId:104	10	50

208	clientName:doctor1 timeStamp:nt location:room32 clientAddr:127.0.0.1	10	50
209	recordId:103 dataSource:vital-signs location:room12 requestType:read	10	66.67
210	clientAddr:127.0.0.1 timeStamp:am dataSource:vital-signs requestType:read	25	100
211	clientName:doctor1 requestType:read recordId:104 clientAddr:127.0.0.1	10	66.67
212	clientName:doctor1 recordId:104 dataSource:vital-signs clientAddr:127.0.0.1	10	50
213	recordId:104 requestType:read dataSource:vital-signs location:room12	10	66.67
214	clientName:doctor1 timeStamp:am dataSource:vital-signs location:room12	15	50
215	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 dataSource:vital-signs	10	50
216	clientName:doctor1 requestType:read location:room32 dataSource:vital-signs	20	66.67
217	recordId:102 clientAddr:127.0.0.1 dataSource:vital-signs requestType:read	10	66.67
218	clientAddr:127.0.0.1 location:room32 recordId:105 requestType:read	10	66.67
219	clientAddr:127.0.0.1 timeStamp:nt location:room12 dataSource:vital-signs	15	50
220	requestType:read location:room12 timeStamp:am dataSource:vital-signs	15	100
221	clientAddr:127.0.0.1 timeStamp:nt location:room32 dataSource:vital-signs	10	50
222	clientName:doctor1 timeStamp:nt dataSource:vital-signs clientAddr:127.0.0.1	25	50
223	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:102	10	66.67
224	clientName:doctor1 location:room32 clientAddr:127.0.0.1 recordId:102	10	50

225	clientName:doctor1 location:room32 dataSource:vital-signs recordId:102	10	50
226	clientName:doctor1 location:room12 recordId:103 dataSource:vital-signs	10	50
227	clientAddr:127.0.0.1 location:room12 dataSource:vital-signs recordId:103	10	50
228	clientName:doctor1 requestType:read location:room32 timeStamp:nt	10	50
229	recordId:102 clientAddr:127.0.0.1 location:room32 requestType:read	10	66.67
230	timeStamp:nt requestType:read location:room32 dataSource:vital-signs	10	50
231	clientName:doctor1 requestType:read dataSource:vital-signs timeStamp:am	25	100
232	clientAddr:127.0.0.1 location:room12 requestType:read timeStamp:am	15	100
233	clientName:doctor1 timeStamp:nt dataSource:vital-signs location:room12	15	50
234	clientAddr:127.0.0.1 recordId:101 requestType:read dataSource:vital-signs	10	66.67
235	clientAddr:127.0.0.1 timeStamp:nt requestType:read location:room32	10	50
236	clientName:doctor1 timeStamp:nt location:room32 dataSource:vital-signs	10	50
237	clientName:doctor1 requestType:read clientAddr:127.0.0.1 timeStamp:am	25	100
238	clientName:doctor1 requestType:read location:room32 timeStamp:am	10	100
239	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:101	10	66.67
240	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:103	10	66.67
241	clientAddr:127.0.0.1 location:room12 dataSource:vital-signs timeStamp:am	15	50
242	clientAddr:127.0.0.1 recordId:104 requestType:read dataSource:vital-signs	10	66.67

243	clientName:doctor1 location:room32 dataSource:vital-signs clientAddr:127.0.0.1	20	50
244	clientName:doctor1 timeStamp:nt clientAddr:127.0.0.1 location:room12	15	50
245	clientName:doctor1 requestType:read dataSource:vital-signs timeStamp:nt	25	50
246	clientName:doctor1 requestType:read location:room12 timeStamp:nt	15	50
247	clientAddr:127.0.0.1 timeStamp:nt location:room12 requestType:read	15	50
248	clientName:doctor1 requestType:read clientAddr:127.0.0.1 dataSource:vital-signs	50	66.67
249	clientName:doctor1 location:room12 recordId:101 clientAddr:127.0.0.1	10	50
250	clientAddr:127.0.0.1 timeStamp:nt location:room12 dataSource:vital-signs requestType:read	15	50
251	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:103 dataSource:vital-signs	10	66.67
252	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:102 dataSource:vital-signs	10	66.67
253	clientName:doctor1 timeStamp:am dataSource:vital-signs clientAddr:127.0.0.1 location:room12	15	50
254	clientAddr:127.0.0.1 location:room32 recordId:105 dataSource:vital-signs requestType:read	10	66.67
255	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 location:room32 requestType:read	10	66.67
256	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs timeStamp:am	15	100
257	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 recordId:103	10	66.67
258	clientAddr:127.0.0.1 location:room12 requestType:read recordId:101 dataSource:vital-signs	10	66.67
259	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 recordId:102	10	66.67

260	recordId:102 clientAddr:127.0.0.1 dataSource:vital-signs location:room32 requestType:read	10	66.67
261	clientAddr:127.0.0.1 timeStamp:nt requestType:read dataSource:vital-signs location:room32	10	50
262	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 requestType:read dataSource:vital-signs	10	66.67
263	clientName:doctor1 requestType:read recordId:104 dataSource:vital-signs clientAddr:127.0.0.1	10	66.67
264	clientAddr:127.0.0.1 location:room12 requestType:read dataSource:vital-signs recordId:103	10	66.67
265	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 timeStamp:am	10	100
266	clientName:doctor1 timeStamp:nt location:room32 clientAddr:127.0.0.1 dataSource:vital-signs	10	50
267	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 timeStamp:nt	15	50
268	clientName:doctor1 requestType:read clientAddr:127.0.0.1 timeStamp:nt dataSource:vital-signs	25	50
269	clientName:doctor1 requestType:read location:room32 dataSource:vital-signs timeStamp:am	10	100
270	clientName:doctor1 recordId:105 requestType:read location:room32 dataSource:vital-signs	10	66.67
271	clientName:doctor1 requestType:read clientAddr:127.0.0.1 timeStamp:am dataSource:vital-signs	25	100
272	clientName:doctor1 requestType:read clientAddr:127.0.0.1 recordId:101 dataSource:vital-signs	10	66.67
273	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 recordId:104	10	66.67
274	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs recordId:103	10	66.67
275	clientName:doctor1 requestType:read location:room32 recordId:102 dataSource:vital-signs	10	66.67
276	clientAddr:127.0.0.1 location:room32 requestType:read timeStamp:am dataSource:vital-signs	10	100
277	clientName:doctor1 timeStamp:nt dataSource:vital-signs clientAddr:127.0.0.1 location:room12	15	50

278	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs recordId:104	10	66.67
279	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 timeStamp:am	15	100
280	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs recordId:101	10	66.67
281	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 dataSource:vital-signs	30	66.67
282	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 recordId:101	10	66.67
283	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 timeStamp:nt	10	50
284	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 location:room32 dataSource:vital-signs	10	50
285	clientName:doctor1 timeStamp:am dataSource:vital-signs location:room32 clientAddr:127.0.0.1	10	50
286	clientName:doctor1 requestType:read location:room32 dataSource:vital-signs timeStamp:nt	10	50
287	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 dataSource:vital-signs	20	66.67
288	clientName:doctor1 recordId:104 location:room12 clientAddr:127.0.0.1 dataSource:vital-signs	10	50
289	clientName:doctor1 location:room12 recordId:101 dataSource:vital-signs clientAddr:127.0.0.1	10	50
290	clientAddr:127.0.0.1 location:room12 requestType:read dataSource:vital-signs recordId:104	10	66.67
291	clientAddr:127.0.0.1 location:room12 requestType:read dataSource:vital-signs timeStamp:am	15	100
292	clientName:doctor1 location:room12 clientAddr:127.0.0.1 dataSource:vital-signs recordId:103	10	50
293	clientName:doctor1 requestType:read location:room12 dataSource:vital-signs timeStamp:nt	15	50
294	clientName:doctor1 location:room32 dataSource:vital-signs recordId:102 clientAddr:127.0.0.1	10	50

295	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 dataSource:vital-signs recordId:101	10	66.67
296	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 timeStamp:nt dataSource:vital-signs	10	50
297	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 recordId:102 dataSource:vital-signs	10	66.67
298	clientName:doctor1 requestType:read location:room32 clientAddr:127.0.0.1 timeStamp:am dataSource:vital-signs	10	100
299	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 recordId:104 dataSource:vital-signs	10	66.67
300	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 timeStamp:am dataSource:vital-signs	15	100
301	clientName:doctor1 recordId:105 clientAddr:127.0.0.1 location:room32 requestType:read dataSource:vital-signs	10	66.67
302	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 recordId:103 dataSource:vital-signs	10	66.67
303	clientName:doctor1 requestType:read location:room12 clientAddr:127.0.0.1 timeStamp:nt dataSource:vital-signs	15	50

Appendix F: The Classifier Class

```
package anomalyrulegenerator;
import java.util.*;
import java.sql.*;
public class Classifier {
    private List<Integer> idList;
    private List<Double> supConfList;
    private String ruleCondition;
    private Map<String,List<Double>> mCAR;
    private ConnectionManager cm;
    private Connection conn;
    private Statement st;
    private ResultSet rs;
    private String sql;
    private char idSeparator;
    private SortedSet<Integer> temp;
    private Set<List<String>> databaseSet;
    private Map<String,Integer> classifierRules;
    private DatabaseToSet dbts;
    private char spcChar;
    public Classifier() {
        idList=new LinkedList<Integer>();
        spcChar=' ';
        mCAR=new HashMap<String,List<Double>>();
        classifierRules=new HashMap<String,Integer>();
        sql="SELECT * FROM tblrules";
        idSeparator='-';
        cm=new ConnectionManager();
        try {
            conn=cm.getConnected();
            st=conn.createStatement();
            rs=st.executeQuery(sql);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        try {
            while(rs.next()){
```

```
            supConfList=new ArrayList<Double>();
            supConfList.add(Double.parseDouble(rs.getString("support")));
            supConfList.add(Double.parseDouble(rs.getString("confidence")));
            ruleCondition=rs.getString("id")+String.valueOf(idSeparator)+rs.getString("ruleCondition");
            mCAR.put(ruleCondition,supConfList);
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
public void sort(){
    while(idList.size()<mCAR.size()){
        for(Map.Entry<String,List<Double>> e : mCAR.entrySet()){
            List<Double> lstDoubleMax=e.getValue();
            String ruleConditionMax=e.getKey();
            String idRuleConditionMax[]=ruleConditionMax.split(String.valueOf(idSeparator));
            if(lstDoubleMax.size()<3){
                for(Map.Entry<String,List<Double>> e2 : mCAR.entrySet()){
                    List<Double> lstDouble=e2.getValue();
                    String ruleCondition=e2.getKey();
                    String idRuleCondition[]=ruleCondition.split(String.valueOf(idSeparator));
                    if(lstDouble.size()<3){
                        if(lstDoubleMax.get(1)>lstDouble.get(1)){}
                        else if(lstDoubleMax.get(1)<lstDouble.get(1)){
                            lstDoubleMax=lstDouble;
                            idRuleConditionMax[0]=idRuleCondition[0];
                        }
                        else{
                            if(lstDoubleMax.get(0)>lstDouble.get(0)){}
                            else if(lstDoubleMax.get(0)<lstDouble.get(0)){
                                lstDoubleMax=lstDouble;
                                idRuleConditionMax[0]=idRuleCondition[0];
                            }
                            else{// if(lstDoubleMax.get(0)==lstDouble.get(0)){
                                if(Integer.parseInt(idRuleConditionMax[0])>Integer.parseInt(idRuleCondition[0])){
                                    lstDoubleMax=lstDouble;
                                    idRuleConditionMax[0]=idRuleCondition[0];
```

```
                    }
                  }
                }
              }
            }
            idList.add(Integer.parseInt(idRuleConditionMax[0]));
            for(Map.Entry<String,List<Double>> e3 : mCAR.entrySet()){
              if(e3.getKey().split(String.valueOf(idSeparator))[0].equals(idRuleConditionMax[0])){
                List<Double> lst=e3.getValue();
                lst.add(0.0);
                e3.setValue(lst);
              }
            }
          }
        }
      }
    }
public Map<String,Integer> genClassifier(){
    dbts=new DatabaseToSet();
    databaseSet=dbts.generateSet();
    for(Integer i : idList){
      temp=new TreeSet<Integer>();
      int countError=0;
      boolean found=false;
      boolean omitted=false;
      boolean marked=false;
      String currentCondition=null;
      String splitCurrentCondition[]=null;
      Iterator<String> iString=mCAR.keySet().iterator();
      while(iString.hasNext()&&!found){
        currentCondition=iString.next();
        splitCurrentCondition=currentCondition.split(String.valueOf(idSeparator));
        if(splitCurrentCondition[0].equals(String.valueOf(i))) found=true;
      }
      String eachCurrentCondition[]=splitCurrentCondition[1].split(String.valueOf(spcChar));
      int j=0;
      for(List<String> recordSet : databaseSet){
```

```
        omitted=false;
        int l=0;
        while(!omitted&&l<eachCurrentCondition.length){
            boolean isFound=false;
            int k=0;
            while(!isFound&&k<7)
                if(recordSet.get(k).equals(eachCurrentCondition[l])) isFound=true;
                else k++;
            if(!isFound)
                omitted=true;
            else l++;
        }
        j++;
        if(!omitted){
            temp.add(j);
            if(recordSet.get(7).equals("classCat:normal")){
                  marked=true;
            }
            else countError++;
        }
    }
    if(marked){
        classifierRules.put(currentCondition,countError);
        int m=0;
        for(Integer count : temp){
            Iterator<List<String>> recordSet=databaseSet.iterator();
            int l=0;
            while(l<(count-m)&&recordSet.hasNext()){
                List<String> forNoUse=recordSet.next();
                l++;
            }
            recordSet.remove();
            m++;
        }
    }
}
return classifierRules;    }}
```