The Annotation Station: An open source technology for data visualization and annotation of large biomedical databases

by

Omar T. Abdala

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical [Computer] Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 4, 2005

Author_____

Department of Electrical Engineering and Computer Science

February 4, 2005

Certified by_____

Prof. Roger Mark

Thesis Supervisor

Accepted by_____

Prof. Arthur C. Smith

Chairman, Department Committee on Graduate

1 Introduction

Over the past 20 years, there has been a large ongoing effort in the Laboratory for Computational Physiology (LCP) to collect and annotate large databases of physiologic signals in order to facilitate an open initiative to develop algorithms to automate a variety of clinical tasks [1]. The success of this approach motivated the collection of a temporal database of signals typically collected in the intensive care unit (ICU) from the Beth Israel Deaconess Medical Center (BIDMC), known as the Multi-parameter Intelligent Monitoring for Intensive Care (MIMIC) database [11]. Although certain basic clinical information was recorded about the patients in this database, much of the information available from the ICU, such as nursing notes, were not recorded. Approximately 5 years ago, the LCP began an initiative to collect a massive temporal database consisting of almost all the information available in the BIDMC ICU [3] known as MIMIC II. This database not only consists of the beside monitor waveforms collected in the original MIMIC database, but also most of the sources of information available to the clinical staff, ranging from medication drip rates and fluid balances, to subjective evaluation scales and discharge and clinical progress notes detailing patient interventions, reactions, clinical visits and past medical history.

The complexity of this database presents many challenges, but also many opportunities for the application of technology to improve patient care. In particular, previous review and annotation schemes are inappropriate for this type of data. A system for efficiently presenting this information in a coherent manner to an expert reviewer is required. Furthermore, an updated annotation structure for providing machine-readable descriptors of the clinical evaluation through time must be developed. This thesis details a system that attempts to provide a framework to facilitate the annotation process. The implemented solution is an Annotation Station used for clinical data review and annotation recording.

1.1 Challenges and opportunities in the ICU

Current trends in ICU demographics include, aging populations [13], a reduction in the clinical staff to patient ratio [19], increased reliance on technology [14] and an increase in the volume of data recorded for each patient [15]. All of these factors lead to an increasing stress on modern ICUs.

Within the ICU, a tremendous amount of data is collected, displayed and stored [14]. The variety and volume of this data can present a burden to clinicians in the form of information overload. Clinicians are expected to be aware of and react to all the available

information. Unfortunately, it is difficult for a person to maintain the necessary vigilance and concentration on this continuous data stream and to recognize and react to important changes in patient state. Furthermore, the high number of multiple simultaneous signals that must be assimilated and correlated in order to provide a correct assessment of the patient's state are presented in a variety of formats, including written notes, machine drip rates or threshold settings, audible clues (such as alarms or verbal exchanges) and graphical formats (such as charts or waveform morphologies). To aid this situation, many alarm algorithms have been developed to indicate critical events in each of the streams of data. However, these algorithms generally produce a tremendous number of false alarms [16] (in order to reduce the probability of missing a real alarm) and little inter-signal correlation is performed to alleviate these problems. Not only do these false alarms cause resources to be inappropriately directed, and the clinical staff to become desensitized to the importance of the alarm, but also the unnecessary increase in noise and activity can lead to deterioration in the efficacy of the care administered to the patient [12]. In order to alleviate this situation a new approach to presenting and synthesizing this data is required.

While the tremendous amount of data collected in the ICU presents many challenges, it also presents many unique opportunities. Collecting and annotating a rich database of ICU data, including alarms will provide an infrastructure on which more intelligent algorithms can be tested. The current feasibility of collecting and analyzing such data can be attributed to the advent of low cost computing power, massive data storage capabilities and high-speed networks. The cost of data storage has dropped drastically over the last five years (and continues to do so) and this makes it possible to maintain a huge database of patient records containing tens of thousands of patient stays. Moving this data between systems for analysis and review is far less cumbersome than it would have been several years ago with recent advances of terabyte Firewire drives and gigabit Ethernet. Large increases in disk access speed, RAM size and CPU speeds means that large sections and multiple channels of data can be rapidly processed and allow efficient algorithm development. For example, a typical desktop computer with 1Gb of RAM provides the facility to load an entire patient record into memory (for display or processing) and therefore rapidly review or operate on the entire stay at once. With the current rate of increase in memory capacities, the simultaneous processing of multiple patient records is now a possibility.

The MIMIC II database currently contains over 3000 complete patient records and 1 terabyte of data, and will continue to grow to a projected 10000 records [4]. In order to develop a set of robust algorithms that assist clinical decision making in the ICU, expert annotations detailing the clinical evaluation and decision process must be made. However, neither a platform to efficiently review the sum of all ICU data, nor a current standard for this type of annotation exists.

1.2 The Envisioned Solution – An Advanced Monitoring System

In order to alleviate the problems detailed above, an Advanced Monitoring System (AMS) has been proposed [4]. The AMS is a framework within which all data available in the ICU is considered jointly and intelligent decisions are made about how to alarm and whether or not to alarm. The main goal of the AMS is to use the multiple streams of data available in the ICU to assess the current state and predict the future state of the patient. The way the AMS can do this is to perform multi-dimensional analysis, which will result in improved signal quality metrics, hypothesis generation, physiological state tracking, and alarms.

One of the most important issues connected to Intelligent Patient Monitoring (IPM) is that of hypothesis generation, where an algorithm automatically determines the state of the patient and reports it to the clinicians. As a result, this automatic determination may reduce the time necessary for a clinician to perform an assessment and come to a conclusion concerning the state of a patient. Not only would this reduction in time result in a condition being treated earlier and more effectively, but also it would free resources and increase the level of care and attention to other patients. Algorithms for hypothesis generation may also detect complex and subtle combinatorial trends that are either impossible for a human to detect or are too taxing for them to invest the effort to do so. For example, it could be found that a weighted combination of changes in heart rate, blood pressure, oxygen saturation, respiration, hematocrit, and urine output is significant for some condition although subsets of these signals exhibiting the same changes may be normal. It is unreasonable to expect a clinician to be able to assess all these parameters simultaneously. Such a multidimensional space of parameters would most probably have a nonlinear and complex decision boundary between healthy and unhealthy patients which itself may be best described in terms of probabilities. Determining these boundaries requires the analysis of a large database and implementation of an alarm system may not be amenable to human interpretable thresholds. However, the more computationally intensive an alarm trigger is and the less intuitive the reason for an indication of abnormality, the less a direct hypothesis for the problem can be derived. A unintuitive, yet effective alarm is nonetheless useful.

Another simpler possible modality of IPM is that of acuity scoring, where the time evolution of how acute or critical a patient's state has become is recorded. This is important in a clinical setting because it allows the identification of high-risk patients to be identified to improve efficiency and save time. The prioritization of patients needs to be automated so as not to occupy valuable ICU clinician time. An acuity score should take into account the health of several systems in the body and combine them so that overall risk can be quickly assessed. If this risk can be assessed, clinicians will be able to focus their attention on more critically ill patients and make the necessary interventions at the appropriate times thereby improving patient care.

One seemingly simple modality of IPM is the task of determining which of the patient data is abnormal and of note in determining the state of the patient. This would be extremely useful to clinicians since this type of algorithm could automatically bring relevant data to their attention in an effective and accurate manner.

If the concept of an acuity score is extended to include prediction, then the process can be automated to produce early warning scores (EWS). [17] An EWS score attempts not only to answer the question of what state a patient is in at a given moment in time, but also to predict the state of the patient at a later time from the data at previous times. This prediction can provide clinicians with early alerts to deteriorations in patient state, which will allow them to intervene sooner and, ideally, to avoid the predicted deterioration.

1.3 Tools Toward The Solution

In order to develop a functional AMS, a framework must be constructed that allows the clinician to answer the following questions:

1) What is the current state of the patient given the current and past data?

2) What is the prospective diagnosis at a future time given the current and past data?

3) What are potential treatment options and which of these do the AMS recommend given the current and past data?

Different branches of Computer Science and Electrical Engineering attempt to answer these questions in different manners. Three general approaches have been considered when developing the plan for the AMS.

One way of answering the above questions is to apply a statistical model to a large corpus of patients. The answers to the posed questions are then found through looking at past patients where the answers to these questions are known. Firstly, a conditional distribution is derived over the possible answers to the questions given the available data. Then an appropriate estimate is selected depending on the desired properties of the estimate. There are many forms that this approach can take which are collectively referred to as Machine Learning. One important example of a machine learning problem relevant to patient state determination is that of classification problems.

A second way of answering these questions is to mathematically model the underlying processes that cause the observable data. For example, a cardiovascular model has been developed at the Laboratory for Computational Physiology [18]. Using such a model, parameters can be estimated with observable data to provide clues as to the underlying physiology that produce the observed patterns in these data sections.

The third method of answering these questions involves text-based knowledge extraction. Instead of limiting oneself to the numerical data available to the AMS, text knowledge extraction can be performed on the corpus of notes that are routinely written as part of a patient's documentation. These resources are rich in contextual information concerning the patient's history and previous events. The results of this type of knowledge gathering can aid any other technique by making the context it discovers available.

1.4 The Need for a New Annotated Database

In the past, in a limited context, there has been success in arrhythmia analysis and the use of artificial intelligence (AI) techniques in analyzing the ECG waveform [1]. Mark et. al. [20] demonstrated this approach by developing appropriate graphical user interfaces (GUI) and algorithms and having clinically trained experts annotate each beat on a set of ECGs of patients suffering from a variety of conditions. By open-sourcing these databases [2] the biomedical signal processing community has developed and improved algorithms that perform arrhythmia analysis. It is hoped that the community that has developed from the Physionet resource can be leveraged in order to collectively solve some of the problems of IPM. There are several reasons why IPM is much more difficult than arrhythmia analysis. Firstly, the arrhythmia problem is naturally synchronized by the onset and shape of beats. In the case of the ICU problem, it is unclear how to divide the patient stay into time units, or even what time scale over which it is appropriate to view the data. In the arrhythmia problem, there are a limited number of distinguishable complications and effects. For example, the ECG can be tachycardic, there could be fibrillation, or there could be ST changes [21]. The number of potential complications for an ICU patient is enormous in part because of the complex interaction of the multiple organ systems in the human body. In fact, the entire NIH Universal Medical Language System (UMLS) [7] database structure is an effort to be able in part to simply express some of these ICU complications in a regular and formulaic way. The massive number of codes in the UMLS is [10] a result of the fact that complications can involve several organ systems, each of which in itself has a tremendous number of complications that can occur. Aside from the annotation process, the data itself is far more tractable in the ECG arrhythmia problem where one signal source is present, which can be parameterized by a finite and small number of features related to intra-beat times and amplitudes. In the ICU problem, many different types of data must be dealt with, which are not all as easily parameterized and identified as in the case of the ECG.

Despite the challenges involved, a couple of attempts have been made at the LCP to collect the type of database necessary to support multi-dimensional IPM. Initially, between 1994 and 1997, Mark et. al. [11] collected data from the BIDMC ICU and carefully selected 100 patients who were deemed to be representative of the ICU population. They then publicly posted these records on physionet [2] for dissemination in the research community. This database proved to be highly important in developing and validating algorithms related to heart arrhythmias and hemodynamic alarms based on ECG and ABP [22]. Following the success of this database, Mark et. al. [4] began the ongoing effort named MIMIC II which took advantage of new technological capabilities in high speed networking and massive data storage to record a richer set of data for a dramatically larger set of patients over their entire ICU stays. MIMIC II now includes data for over 3000 patient records. The limits of MIMIC I, which consists of only 100 patient records each of length 24 or 48 hours, are striking. Each of these records consists of at most a set of waveforms and measurements as well as some clinical information in an Access database. The most important and critical difference between MIMIC I and MIMIC II is that there are no annotations concerning a patient's state on MIMIC I data. Therefore, it cannot be used as is for a testbed to validate algorithms without significant extra work invested in it. The goal of MIMIC II is to produce a database with a set of data almost as rich as is available in the ICU (with notes, medications, lab values, trend plots, waveforms) that is annotated to the extent that representative subpopulations can be selected out of the larger database and separated into categories that lend themselves to be used as a testbed to validate multi-parameter algorithms.

There are several similar data collection efforts also ongoing in the medical research community. For example, the SIMON database [5] at the Vanderbilt University Medical Center (VUMC) collects almost as much information as is in the MIMIC II database except waveforms. In fact, the monitors in use for the collection of the SIMON database are the same Philips CareVue monitors that are used at the BIDMC for the collection of MIMIC II. The IMPROVE [6] data collection project assembled a significant amount of waveform and trend data from the Kuopio University Hospital in Finland. However, MIMIC II provides significant additional information beyond these data collection systems, which facilitates the clinician to assess the state of a patient. In particular waveforms, parameter data, clinical progress notes, lab tests, medications, and fluids data are available in MIMIC II as will be described in Section 2.

Of the previous attempts at creating annotated databases, SIMON comes closest to, but falls short of, the type of database that would support this form of annotation. At the VUMC, clinicians respond to alarms and indicate whether they are true or false and physiologically relevant or not. These clinicians also have the option of including free text notes that describe the information they have entered. SIMON would be sufficient for work in reducing the plethora of false alarms existent in the ICU, but this type of annotation scheme will not support the hypothesis generation and clinical reasoning to be gained from the AMS. Firstly, without coded identifiers, it will be impossible to construct any type of labeled test and training set for this purpose. Secondly, without a good approximation of the onset and offset of a condition, there is no basis against which to

evaluate the performance of a prediction algorithm. If an intelligent algorithm is to predict the onset of some condition, that onset must first be clearly known. Thirdly, without the links to the data streams, relevant portions of data cannot be identified. Fourthly, causal links between patient states are critical in determining the evolution of the disease in a patient. Fifthly, quantitative evaluations of the patient state give us a target against which to evaluate our algorithms for estimating patient state. Therefore, the MIMIC II effort and its annotation scheme are completely indispensable in facilitating the advancement of Intelligent Patient Monitoring.

As the MIMIC II database stands now, it is a large and growing set of patient records with a rich array of data recorded. This will prove useful in clinical studies where a researcher can retrospectively examine the effects of various treatments and medications on patients with different histories and conditions and can hypothesize on how to create better treatments, how to better identify which patients are good candidates for particular treatments, or how to better prepare patients for these treatments.

As the listing of tools that are leveraged for development of the AMS demonstrates, a test bed is needed to guide development and validation of the results of probabilistic models of disease progression and deterministic models of physiological systems. It is important to note that framing the problem properly is of critical importance in clinical decision making. Once a set of features to use for prediction and physiological state evaluation are clearly identified in quantifiable terms and records have been annotated with a clear label for presence or absence of a condition or event to predict, many tools from artificial intelligence can be used to analyze this data. Such annotations can be used to break down the patients into subpopulations by disease category, and well-known classification techniques can then distinguish patients of one type from patients of another type. It is clear, though, that in order to do this, we need to construct a massive, annotated ICU patient database. A standard result from learning theory, dubbed the “curse of dimensionality” states that the amount of training data required by any algorithm grows exponentially as one increases the number of parameters to consider. Since the AMS must produce multi parameter alarms, we know that the amount of labeled data we need is immense. This is further emphasized by the fact that studies will typically be done on subpopulations within the ICU. For example, if one wants to find and model the effect of levophed on a patient who was previously septic, there are probably only a limited number of patients who fulfill these criteria. Therefore, we want an annotated database large enough that at least for some subpopulation groups, there is enough data to provide the statistical power necessary to pursue the work of the AMS. The key to IPM and the successful development of an AMS is therefore its formulation as a standard machine learning or modeling problem and an annotated testbed is therefore critical.

1.5 The Annotation Challenge

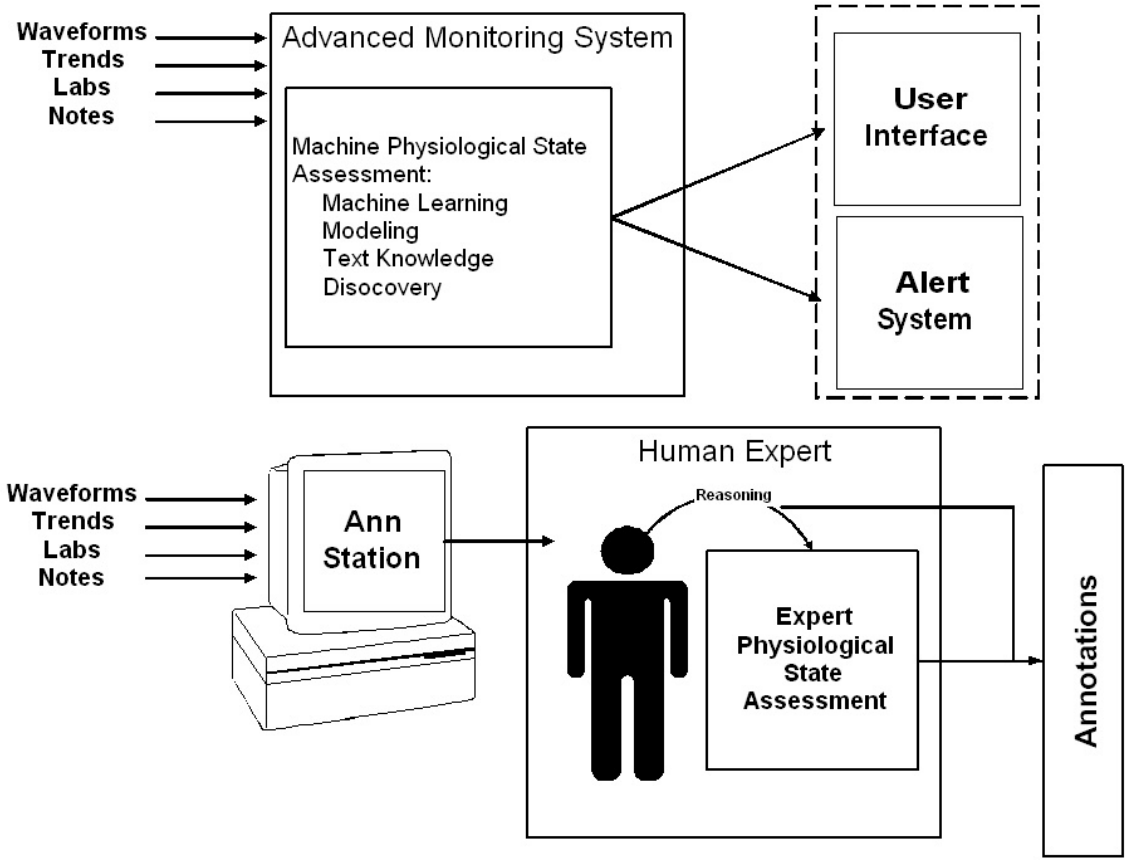

Figure 1. The annotator must imitate the AMS in analyzing the data and developing a hypothesis. The AMS displays this data to the clinicians. The annotator records his analysis in annotations.

In order to produce an annotated testbed, expert human annotators must hand review the cases in the database, reason about the patients' states and record that reasoning in a human and machine readable format. Essentially, the humans are imitating the intelligent alarms we hope to produce. By presenting the annotators with the same data at the disposal of the AMS, and recording their assessments of the patient's state, we can evaluate our AMS against the thought process of these annotators. The tool developed to support human annotation is termed the "Annotation Station". See Figure 1 for a schematic representation of the role of the Annotation Station in the annotation process. The Annotation Station must perform several functions effectively and efficiently to enhance the annotators' understanding of each medical case and to allow them to efficiently record that understanding thus maximizing the quality of the produced annotations. The goals of this M. Eng. Project are therefore to design an Annotation Station that:

1) provides efficient access to the huge amount of Mimic II data that is distributed over vastly different storage formats.

2) fuses and enforce time synchronization over data that is collected asynchronously, suffers from missing data points and missing data channels.
3) provides a flexible GUI for reviewing and analyzing patients. It must enable customized views for reviewing cases where different disease processes are active.
4) provides annotators the ability to organize and their reasoning and logical flow and record it in a human and machine-readable format.

The following 6 chapters detail the implementation of a software system to fulfill these goals. In chapter 2, a description of the format of the MIMIC II data is presented. In chapter 3, the method used to load the data into the Annotation Station is described. In chapter 4, an overview of the design considerations made when constructing the Annotation Station are discussed. In chapter 5, the annotation structure and methodology for annotating is presented. In chapter 6, a guide to using the Annotation Station interface is given. In chapter 7, documentation of the Annotation Station software classes is given. Finally, in chapter 8, a summary of the utility of this framework and future development strategies are presented.

2 Data Description

2.1 Summary of Collection

The transfer of the MIMIC II data from the BIDMC to MIT involves acessing two different data bases which use two different formats. One type of data is that stored in CareVue, an Oracel-based Philips system for recording clinical measurements (such as medications) [8], is archived within the hospital's Information Support Mart (ISM) [8]. This data is imported using SQL queries over a Virtual Private Network (VPN) connection between the Laboratory for Computational Physiology (LCP) at MIT and BIDMC. The trasferred data is stored at the LCP in a Pstgres databse. The other class of data are the waveforms recorded from the bedside monitors (such as ECG). These waveform files, which are not stored in the ISM, but are streamed to Philips server running windows, are periodically copied by hand onto firewire drives over a local network connection, and physically transported to the LCP. The files are subsequently copied off the files via a windows machine onto an NFS mounted server running Linux. The firewire disks are then archived off-site as a static backup.

2.2 Data Types

There is a wide variety of types, sources, and formats of data in MIMIC II. Below is a listing of the types of data that are relevant to the Annotation Station with an explanation of the source and format of each. The tables described in this section are required for use of the Annotation Station. To simplify reading this section all tablenames and fields will be displayed in bold.

2.2.1 Text Data

2.2.1.1 Clinical progress notes

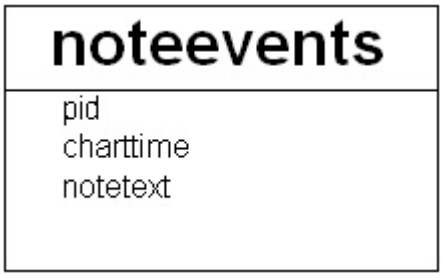

Figure 2. The noteevents table schema.

Clinical progress notes are hand typed by a clinician and recorded into CareVue in the **noteevents** table. The **noteevents** schema is shown in Figure 2. Each note is associated with a unique **pid**, a timestamp indicating the time (**charttime**) at which the note is recorded and an amount of free text (**notetext**). The clinician typically gives a summary of important events in the patient's stay for the previous 8 hours, the length of a nurse's shift. Although there is some limited regularity to the frequency of the clinical progress notes, they are still far from regularly sampled. Nurses can change shifts more or less frequently, remarkably important and notable events can occur, or nurses can simply fail to record a note at the appropriate time. All these factors can cause either an extra note to be written, a note to be missed, or a note to be recorded at an unexpected time resulting in irregularly sampled notes. The important elements of the **noteevents** table used by the Annotation Station are the patient id (**pid**) which identifies the patient to which a note refers, a **charttime** which indicates when a note was written, and **notetext** which is up to 4000 characters of free text documenting significant events of the shift.

2.2.1.2 Discharge Summaries and Other Text Based Laboratory Reports

Figure 3. The discharge table schema

In addition to clinical progress notes, there are several other types of text data. Each patient has a discharge summary written by the attending physician. There are also results to diagnostic tests such as X-rays, CT scans, and some text format lab tests. At the hospital, these are stored in a database other than CareVue. Automatic download of these records is pending for administrative reasons, but a subset of these text records is available to us through manual querying. The discharge summaries are stored in a database table called **discharge** with the relevant portions of the schema shown in Figure 3. The important features of the discharge table are the patient identifier (**pid**) and the text of the doctor's discharge note (**dischargetext**). There is no need for a timestamp on a discharge summary because only one is created per patient stay. The **dischargetext** field can contain up to 100000 characters of free text. The schema to be used for the text based diagnostic test results such as X-rays and CT scans will resemble the **noteevents** table described in Figure 2 with the addition of an **itemid** indicating the type of result (X-Ray, CT). Therefore, the schema will be as is shown in Figure 4.

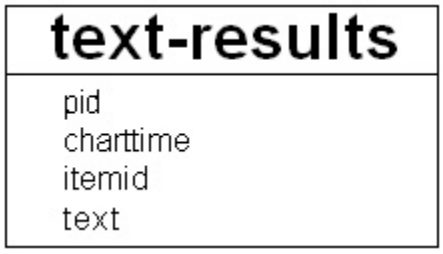

Figure 4. The text-results table schema.

2.2.1.3 Alarms

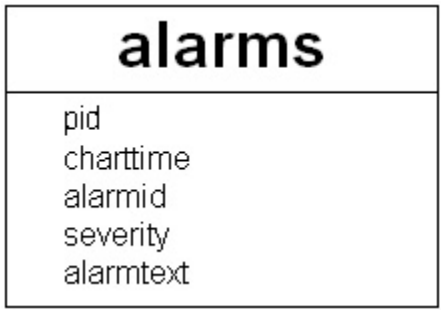

Figure 5. The alarms table schema.

The Philips monitors also record alarms fired during the patient stay. Each alarm fired creates a new row in the **alarm** table described in Figure 5. Each alarm row consists of a **pid** to identify the patient, a timestamp called **charttime** to localize when the alarm was fired, **severity** to indicate how important the alarm is, an **alarmid** to indicate the type of alarm, and **alarmtext** for additional human readable information. The **severity** is coded as an integer ranging from 0 to 3 with 3 indicating the most severe and 0 indicating the least severe alarm. The color codes red, yellow, green and blue are associated with the alarm severity classes 3, 2, 1 and 0 respectively. The **alarmid** is an identifier that references a Philips structure representing all possible alarms. This is not stored in the database, but is available. The **alarmtext** gives information about which data value crossed which threshold to produce this alarm. For example, if the alarm fired on an episode of tachycardia, the **alarmtext** might read "HR 115>100" to indicate that the measured heart rate was 115 which is above the threshold of 100 set to be indicative of tachycardia. The alarm rows are unevenly sampled as they are tied to clinical events in the patient stay.

2.2.2 Unevenly Sampled Numeric Data

2.2.2.1 Chartevents (Nurse Verified Values)

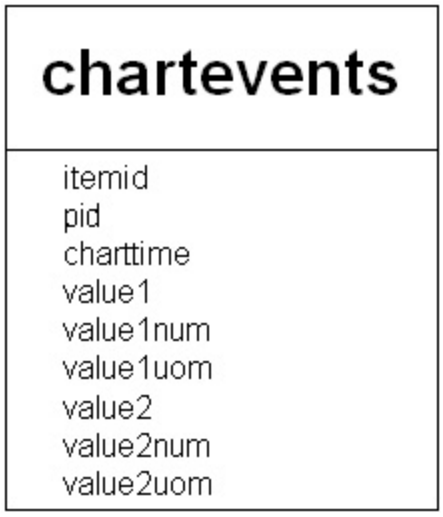

Figure 6. The chartevents table schema.

Figure 7. The d_chartitems table schema.

CareVue contains many types of information from the ICU stay of a patient. **Chartevents** is a monolithic table that includes verified measurements of many trended values including heart rate, arterial blood pressures, oxygen saturation, ventilator settings, lab results for blood chemistries, cardiac output, and many other measurements. Several thousand of these types are stored in a database table called **chartevents**. Each row in the table represents one data point of one type. Within **chartevents**, an **itemid** identifies the type of data in a row. The human readable label of the types demarcated by the **itemids** is given in the **d_chartitems** table where each row ties an **itemid** with a textual **label** naming that signal. For example, the itemid 51 is tied to the label "Arterial BP". Other important fields in **chartevents** are an identifier for the patient (**pid**), the time of the recorded value (**charttime**), the value(s) themselves (**value1, value2**), and the unit(s) of measurement (**value1uom**, **value2uom**). **Chartevents** can include up to two textual or numeric data values for each sample point (in the **value1** and **value2** fields). An example where both values are used is when blood pressure is reported. **Value1** is used for the systolic value and **value2** is used for the diastolic value. Sampling all values of one type for one patient results in construction of a "signal". For example, we can select all of the systolic blood pressures for a particular patient from the **chartevents** table and construct a systolic blood pressure time series, which is a signal. Depending on the type of data, the frequency of recording varies quite significantly. Some measurements such as heart rate

are recorded up to every 5 minutes over some periods, and some blood chemistry results are recorded only once or a couple of times for each patient stay. In addition to the difference between signals, within each type of signal, the rate at which values are recorded changes. For example, it is typical for heart rate and blood pressures to be recorded every hour and inserted into CareVue. But, at times where there are significant events occurring in the patient stay, these values might be recorded much more often, up to every 5 minutes. Therefore, these signals are quite irregularly sampled. For all values in **chartevents**, nurses should verify the values before they are recorded into CareVue by calibrating measurements such as heart rate or blood pressure, ensuring that the ventilator values match the set ventilator settings, or checking that blood chemistries that come back from the lab match the values inserted into CareVue.

2.2.2.2 Medication drip rates

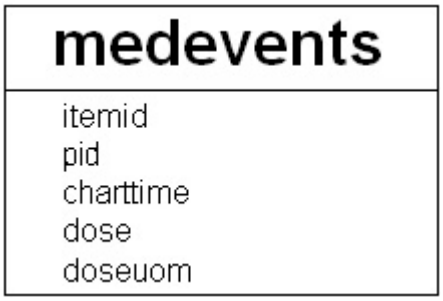

Figure 8. The medevents table schema.

Figure 9. The d_meditems table schema.

In addition to the **chartevents**, there are other types of irregularly sampled data in MIMIC II. One is the medications data, which is recorded in the **medevents** table in CareVue. This records IV medication rate changes. The structure is similar to **chartevents** except that the value stored is called a **dose**. The **d_meditems** table indicates which **itemids** correspond to which medications and give a textual **label** for each. The **medevents** table has one row for each drip rate change. The rate that is recorded in **medevents** is valid until the next time the medication has a rate recorded. These changes occur at irregular times resulting in unevenly sampled signals.

2.2.2.3 I/O Fluids

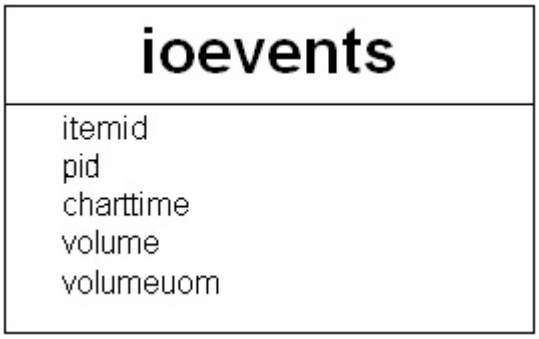

Figure 10. The ioevents table schema.

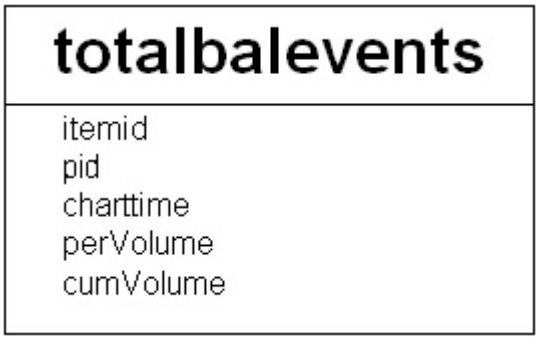

Figure 11. The totalbalevents table schema.

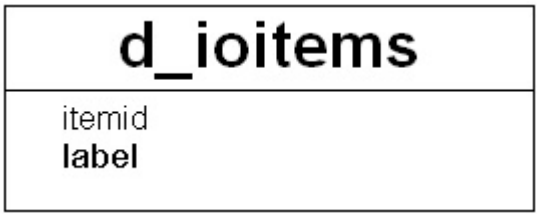

Figure 12. The d_ioitems table schema.

Similarly, the **ioevents** table in CareVue records various IV fluids and how much **volume** of each is infused into the patient. This table also resembles **chartevents** with the exception of its use of fields called **volume** and **volumeuom**, which represent how much volume of a particular fluid was given via IV since the previous row with the same itemid. Many of these values are recorded hourly, by standard, but there are skipped hours, where the total intake or discharge is reported at the end of some number of hours. Also, these data do not arrive on a clock. So, while mostly they are spaced at an hour or some multiple of an hour, this is not necessarily the case. The time spacing can differ from an hour by several minutes. Therefore, these signals must also be represented with a data format accomodating unevenly sampled signals.

The **totalbalevents** table stores information on the total amount of IV fluid taken in by a patient and the total amount of urine discharged. It is very similar to the **ioevents** table except that it has a **perVolume** field indicating the volume that has gone in or out since the last measurement and a **cumVolume** indicating the running total in or out for the entire day.

For both the **ioevents** table and the **totalbalevents** table, the **itemid** demarcates the data type of each row in the respective table. The **d_ioitems** table indicates the text **label** for each of these **itemids**.

2.2.2.4 Census data

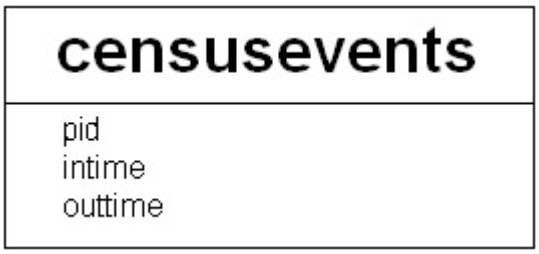

Figure 13. The censusevents table schema.

The **censusevents** table is used by the Annotation Station to determine the length of the stay of a patient. As indicated by the names, the **intime** corresponds to patient admit time and **outtime** corresponds to patient discharge time from the ICU. In the database, there can be more than one row for each patient in censusevents. For example, if the patient were transferred between departments during the stay at the hospital, there will be several intimes and outtimes for the same patient. Therefore, for the purposes of annotating an entire ICU stay, the earliest of all the intimes is taken as the marker of the beginning of the patient stay and the latest of the outtimes is taken as the end.

2.2.3 Evenly Sampled Numeric Data

2.2.3.1 Parameter Data

MIMIC II also includes higher resolution unverified data that is not recorded in CareVue, but is collected alongside it. There are two formats for this higher resolution data.

The first type of data is called “parameter” data, which are unverified parameters sampled at a period of 1 minute such as heart rate, blood pressures, oxygen saturations, and

cardiac output. Each patient has one parameter file which includes 30 such parameters. The parameter files include a variable number of data chunks. Each chunk includes a timestamp, and 30 values, one for each of the 30 parameters at that time point. These values can be either an available value or a value representing missing data. This missing data value, "-888" by convention, is reserved to indicate that the value of that parameter is missing for that time point. Parameter data is mostly evenly sampled at one data chunk per minute, but there can be segments where one chunk follows the previous chunk by more or less than one minute.

2.2.3.2 Waveforms

The second type of data is waveform data, which is recorded at 125Hz. The total possible waveform data that can be recorded is: 3 leads of ECG, respiration, pulmonary artery pressure, central venous pressure, arterial blood pressure, pulse oximetry, and intra-cranial pressure. There is a maximum of 4 of these data streams recorded at once. This data is recorded in binary data files and text header files. Each line of the header file specifies a one minute chunk (7500 samples) written in the data file. These one-minute segments need not be consecutive, although they are continuous within themselves.

The waveform data is currently restricted to hold data values in one byte. This restriction means that a waveform can only take on 256 distinct values. The effect on the signal is a highly quantized look and easy signal saturation. Threfore, because of the data format, the waveforms become artifactual and saturated.

2.3 Data Format

All of the data in its original format suffers from lack of time synchronization and various errors including byte-skipping misalignments. Primarily for adherence to our open source data standard, this data was all converted to WFDB. During the conversion, many of these errors were easily fixed. Therefore, the resulting parameters and waveforms in MIMIC II are better organized and easier to read than the original files that represent the same data in the hospital information system.

2.3.1 Breakup of Data

Querying the Postgres portion of the MIMIC II structure in the ISM format is cumbersome. There are a tremendous number of signals stored in each database table. Consequently, retrieval of one signal for one patient is extremely time consuming. To further complicate this problem, tables such as chartevents and totalbalevents include

many signals that are not relevant or useful to the Annotation Station. Figure 14 shows the size of chartevents and totalbalevens when the database contained 1800 patients. Chartevents alone contains 32 million rows, while the two tables combined contain 38 million rows. In the face of these huge monolithic tables, the queries for individual signals take on the order of minutes to execute. The Annotation Station would not be able to operate with this level of performance. More efficient access had to be made available. The solution to this problem was to copy relevant rows from the monolithic tables to smaller tables that could be more easily queried. Under Postgres, such a system is clearly a major improvement. The average time to extract a signal dropped to several hundred milliseconds.

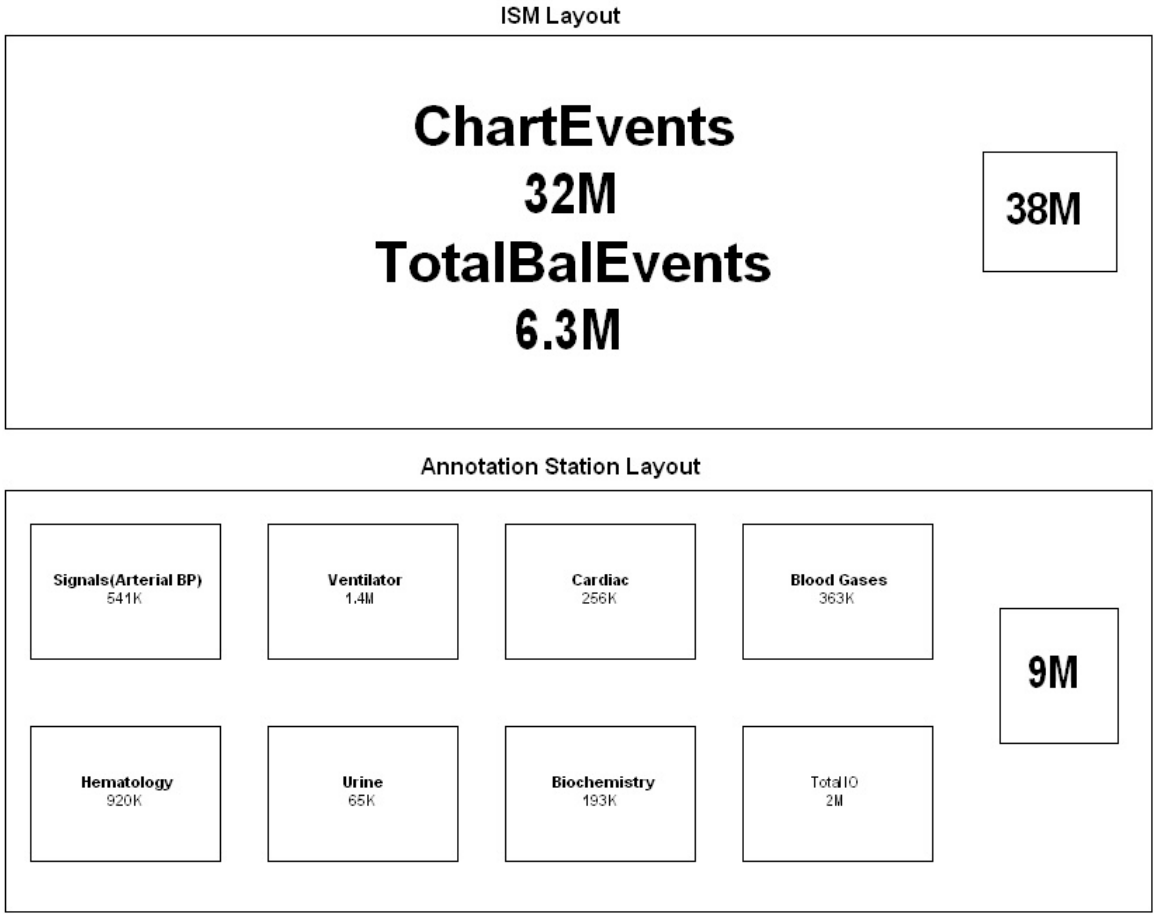

Figure 14. The breakup of chartevents and totalbalevents into more reasonably sized manageable tables. Chartevents contains 32M rows while Totalbalevents contains 6.3M rows in the ISM structure. In the Annotation Station Layout, the largest table to query has 2M rows and the total number of rows in all the relevant tables is 9M.

Some measurements are high rate and individually have a great number of rows in the database. For these signals, there is one separate table for each in the breakup. These are signals such as respiration, CVP, heart rate and blood pressures. These constitute one type of table in the breakup and are identified by the following:

- High-Rate Measurements (SIGNALS)

For signals that are less dense, some number of signals are grouped together and stored in one table. Below are 8 additional tables each corresponding to a category of measurements.

- Cardiac (CARDIAC)
- Hematology (HEMATOLOGY)
- Renal (RENAL)
- Biochemistry (BIOCHEM)
- Ventilator (VENTILATOR)
- Blood Gases (BLOODGASES)
- Urine (URINE)
- Total I/O (TOTALIO)

The first seven of these select various signals of the specified types from the chartevents table. (ventilator settings, various cardiac measurements, blood gases, blood counts[RBC, WBC], urine tests, blood chemistries that indicate renal function, and other blood items categorized as biochemistry) respectively. The Total I/O table contains information from the totalbalevents table having to do with volume measurements of fluid in or out of the patient's body.

A program (ReadCEFile.java) was created to break down the database from the ISM schema into the more compact Annotation Station Layout. The specific types of data included in each sub-table are dictated to this program through a second "breakup" file called itemids. The file has a regular grammar as described in the following:

BREAKUP-FILE = {SECTION}*

SECTION = "SECTION:"$section_name<CR>{LINE<CR>}+"END"<CR>

LINE ={$itemid" "}+{|$signal_name}!{|$field_name}?

An example of a valid breakup file is given in Appendix B. The breakup file is composed of zero to 9 possible sections as listed above. For each section, there are one or more lines terminated by a carriage return. Each of these lines defines one signal, which can be

extracted, queried, plotted, and annotated on the Annotation Station. Unfortunately, in the ISM structure, one type of data does not necessarily have one unique itemid. For example, there are 8 different itemids that map to Sodium. This undesirable property is caused by a nurse changing the name of the signal in CareVue. The system consequently creates a new itemid for that new name. But for our purposes, this is in fact the same signal as the unchanged version, regardless of slight variations in the text label nurses apply. Therefore, when defining a signal in a line of the breakup file, the option must be given to map multiple itemids to one signal. One or more itemids are specified by placing a whitespace-separated list of them at the beginning of the line. This section is terminated by a pipe character, "|". The text after this pipe is the name of the signal. Since, in the SIGNALS section, each of these signals will have a table created in the database, the name of the signal must be a valid database name (i.e. no spaces or unusual characters). Since a signal will be identified by its name in the breakup file, this name has to be unique within the breakup file. In other words, there may not be more than one signal in the same breakup file with the same name. After the signal name, the constructed line can optionally contain a second pipe and an identifier for the field in the database where the values of the signal reside. The default, if this final option is not specified, is the value1 field. So, for example, to specify the diastolic blood pressure signal, we must specifically put an extra pipe and the name "value2" at the end of the line. For the TOTALIO signals, the cumvolume or pervolume also must be specified, since the default value1 field does not exist in totalbalevents.

All of these created tables will be repeatedly queried based on pid and itemid. The common query will be to select all of the points that correspond to one patient and one signal (one or more itemids). Therefore, the program that breaks up the data creates indices in the database based on these two fields individually and jointly. This ensures that all queries that select rows based on the two fields are as fast as possible.

The program that is used to do the breakup of the monolithic ISM tables is called ReadCEfile.java To use it, one must have the java class ReadCEFile.class and a valid breakup file of the documented form.

```
java ReadCEFile $breakup_file $pg_server
```

The above command can be issued from the main AnnotationStation directory, the $breakup_file argument points the program to the breakup file on disk and the $pg_server argument points the program to the address of the postgres database server.

The breakup procedure is quite time consuming, taking on average 10 hours to complete. It only needs to be performed once at the time of database setup as long as the underlying data in chartevents and totalbalevents does not change. If data is added in these tables, the procedure must be repeated to make the new data available to the Annotation Station.

2.3.2 Chartevents Locations

As a side effect of the breakup of the chartevents and totalbalevents tables into smaller tables, a metadata table is created to keep track of which signals have been copied to which tables. The schema of the table is as follows:

Figure 15. The chartevents-locations schema.

The chartevents-locations table ties each **itemid** with two things. Firstly, the table it was copied to (**table_name**). Secondly, the source table it was copied from (**source_table**).

3 Data Loading

3.1 ReadCEFile

The ReadCEFile interface, which is used to breakup MIMIC II, as discussed in Section 2.3.1, can also be used to read the broken down signals from the database. There are several things that need to be done in order for this to happen. The same breakup file that was used to break down the database must be specified to the Annotation Station for correct data loading. Once the file is made available, the ReadCEFile class can be instantiated to form an object containing all of the needed data. The constructor for this class has the following form:

```
public ReadCEFile(String file)
```

The file argument points the ReadCEFile class to the breakup file on disk. Once the ReadCEFile object is created, it can be used to populate a HashMap. The keys of this map are the signal names, exactly as indicated in the breakup file, and the values of the map are SignalData objects, which will be described at length in Section 7.9. Since SignalData is an abstract class and hence cannot be instantiated, the runtime type of these SignalData objects is TrendData, a subtype of SignalData. There are also other subtypes of SignalData as described in Section 7.9. The Java method used to populate this HashMap has the following signature:

```
public void populateSignalsMap(HashMap map, int pid)
```

This method takes an instantiated map to populate and the pid to identify the patient for which the user wants the signals extracted. There is no direct way, using the ReadCEFile interface to extract only one signal from the database. This is due to the fact that the Annotation Station loads all of the needed signals from the database upon loading a patient record. Therefore, the Annotation Station never needs to load signals on an individual basis.

3.2 Loading Procedure

There are several points in the operation of the Annotation Station where data loading occurs. The action that the Annotation Station takes in three of these circumstances will be described.

3.2.1 Patient Load

The patient load procedure is in the main Annotation Station class which drives the system, called AW.java, and has the signature:

public void loadPatient(String pid)

The first type of data that is loaded in the patient load procedure is the parameter data. The ReadParam class is used to do this. ReadParam reads the parameters file, of the format specified in Section 2.2.3.1, into a subclass of the SignalData class called ParamData. Since the parameter data is not very dense (1 sample per minute), loading all the parameter data for one patient is not extremely costly in terms of space considerations. On an average patient stay of 2 days, there are around 3000 minutes of data. So, to store a 4 byte floating-point value for each of these time points only corresponds to about 12 kilobytes of data for each channel of the parameter data. This is easy to fit in memory, fast to search through, and most importantly fast to display. It is for this reason we decided to load all of the patient's parameter data at the time of loading the patient's record.

The second operation that the loading procedure performs is that it copies the notes and alarms for the given patient from the voluminous tables where they reside to a temporary notes table and a temporary alarms table. This is so that queries on notes and alarms can be done more quickly while the annotation station is running.

The third operation is to prepare the medications and IV fluids to be extracted as signals with the rest of the numeric data from the database specified in the breakup file. The medevents table and the ioevents table are sparsely populated enough that it is reasonably efficient to extract the relevant rows at the time of patient loading. The Annotation Station queries the database for a list of the available signals from the medevents and ioevents tables. The itemids as well as the labels of the available signals are then added to the ReadCEFile object representing which itemids will be extracted out of the database as signals to be plotted on the Annotation Station. Once these are added, the list of signals in the ReadCEFile object is complete and the populateSignalsMap method can be run on the ReadCEFile object. The object then has a complete HashMap of all the numeric signals the Annotation Station will need from CareVue.

3.2.2 Update Time

As an annotator is reviewing a case, there is a time point that indicates where the annotator's attention is focused. There are several pieces of data that need to be updated when this operation occurs. The note being viewed gets updated to the nearest note to the

updated time. All the available numeric signals are updated to have the nearest previous value displayed. The signals that are plotted get zoomed around the new time. Also, the timelines all get zoomed to the same region that the signal plots get zoomed to. All of these data display objects will be described in detail in Section 6.

3.2.3 Waveform Load

Although waveform data and parameter data are both evenly sampled data, waveform loading differs from parameter loading. At 125Hz, the average amount of data over a patient stay for a single channel of waveform data would be around 80 Mb. While such an amount of data could be stored in memory, it would be unwieldy to try to fit all of it into the data structure that the Annotation Station's plotting module uses WaveData (described in Section 7.9.3). The strain on the processor would cause the screen refresh on the Annotation Station to grind to a halt. Therefore, there are two approaches that can be taken. The Annotation Station can leave waveforms on disk and retrieve pieces of them as needed to populate a WaveData object with the needed subsection. A second plausible approach would be to load all of the waveform data into memory at the time of loading the patient's record, but not to attempt to plot this entire data signal. When the data needs to be displayed, it can be copied into the WaveData structure used for plotting. The advantage to this approach would be that the data would be loaded from memory rather than from disk. While this would be a significant advantage for speed of waveform viewing in the Annotation Station, such an approach is unnecessary because waveform viewing is not the primary modality of the Annotation Station. Annotators will not perform detailed waveform review with the Annotation Station. The annotators will use the waveforms in quite a limited setting; the waveforms will be used to validate a condition noticed in the trend values, lab values, fluids, etc. Therefore, for simplicity, the first approach was selected. The annotator may zoom in on any point in time, and if he would like to view waveforms, he can load a window of 20 minutes of waveforms around the time of interest. This is around 100,000 points, which is a reasonable amount of data to plot without putting excessive stress on the Annotation Station screen refresh.

A summary of the types of data loaded into the Annotation Station, and where they reside on the server is presented in Figure 16.

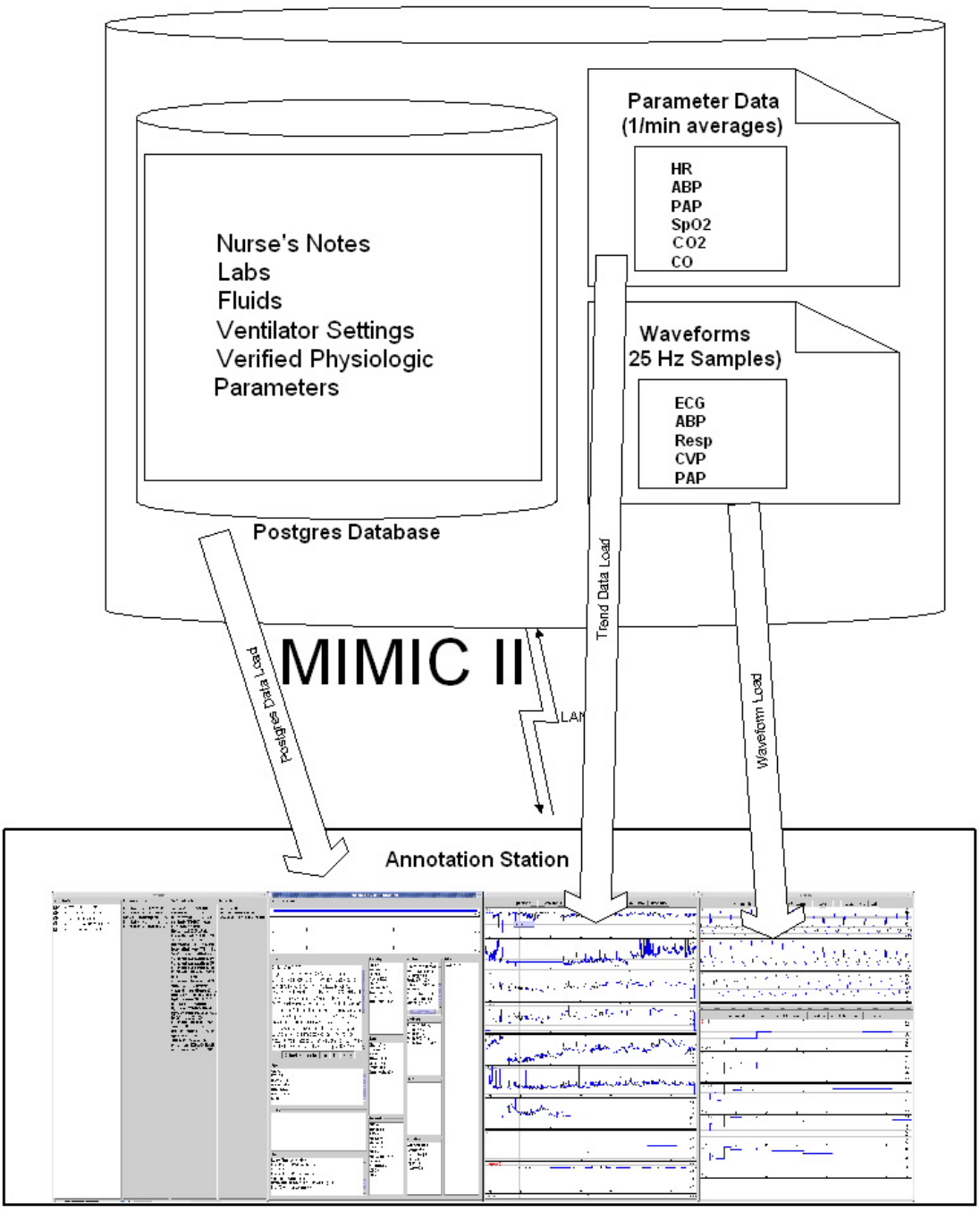

Figure 16. The locations on the Annotation Station where various pieces of the MIMIC II database are loaded. The CareVue data resides in a Postgres database, while the waveforms and parameter data reside in flat files.

3.3 Future Data Format

WFDB is an open data format that includes a header file and a data file as well as an optional annotations file. The header file describes offsets, gains, number of channels,

and interleaving as well as other properties of the data file. In the future, the evenly sampled numeric data will all be stored in WFDB data records. The unevenly sampled data such as notes and lab values will be recorded in WFDB annotations. This is advantageous for several reasons. Firstly, there are many tools available from physionet [2] for use on WFDB ECG and ABP records. Secondly, there are lightweight viewing tools such as WAVE [23] for WFDB which can be used if a user does not want to take on the involved task of setting up a four screen annotation station and database server. So, if a user wishes to download the data for one patient or a small group of patients and review it, this will be possible. In MIMIMC II's current format, loading of subsets of patients is not possible. Once all the data has been converted to WFDB, the Annotation Station classes which read data can easily be ported to be compatible with this format. This will allow us to achieve the gains of viewing patient records on the Annotation Station without an entire installation of the MIMIC II database.

4 Design Considerations

There are many design options to be considered whenever there is a human machine interface. Specifically in this case, the interaction between an annotator and the Annotation Station is of concern. There are two main tasks that need to be accomplished in the process of annotation. Firstly, a clinician needs to review the data of a patient's case, understand the main events of the case and their evolution, and formulate an opinion on the causes for these events. Secondly, a clinician needs to convey these opinions to the Annotation Station. The Annotation Station must then record these opinions in a way that allows humans and machines to later understand the annotator's thoughts. Careful consideration of these two parts of the annotation process results in design principles for the GUI and requirements on the annotation structure in the Annotation Station. These principles guided the Annotation Station development process from the conception stage to the final decisions on graphical user interface and annotation structure. This section will present some of these principles and will provide the motivation for the solutions selected.

4.1 User Interface Design Principles

The annotator who sits to work at the Annotation Station has a very complex task ahead of him. He needs to maintain a tremendous amount of information in his mind for an average of four hours as he reviews the case. All this time, he needs to be accurate and insightful in drawing conclusions from this data. He needs to concentrate fully on the task at hand and cannot be hindered by clumsy, non-intuitive interfaces and unnecessary fiddling with the software. The software needs to be a pleasure for the annotator to use so that he will produce the best possible annotations. The annotator must feel completely at home using the software. In order for this to occur, all Annotation Station features must become habit and second nature for experienced Annotators. There are several ways to facilitate this habit forming.

An important design principle is drawing on a user's previous experiences as metaphors to the task he needs to learn. In this case, the fact that annotators are experienced clinicians can be leveraged. They have been using patient monitors for the duration of their experience and keeping the Annotation Station's features as similar to patient monitors as possible will speed the ramp up time in teaching any clinician how to use the Annotation Station. The task of annotating or even of simply reviewing a patient case is significantly more complicated than the task of viewing a patient's current physiologic metrics on a monitor. This additional complication makes it impossible to copy the paradigms used in the patient monitors directly, but the paradigms can form an important base on which to build. For example, clinicians are used to viewing waveforms as strip charts. ECG channels, ABP waveforms, and respiration signals are typically plotted one on top of the other on patient monitors. Therefore, the annotator is given this

functionality in the format he's accustomed to. A second new problem unique to annotation is the need to display lab values, medications and IV fluids to the annotator. In order to annotate effectively, he needs to be able to quickly scan and pick out significant changes in all of these types of data. This is a task that is not performed with current patient monitors, and as a consequence, only the current values of these data are displayed. The waveform display that the clinician is used to is used as a metaphor for the new display of all the additional type of data, plotted as a time series. The plot strips are then stacked so that they resemble waveform strip charts. This approach gives the annotator completely new functionality, but puts it in a familiar and comfortable format. In this way, the annotator's familiarity and previous experience with old systems is leveraged in order to familiarize him with the new features in the Annotation Station.

A principle that was meticulously followed in constructing the Annotation Station was that of pattern uniformity. This is the idea that once an Annotator has learned to perform any action or become accustomed to a display, this action or display should be consistent throughout the annotating experience. As has been described in depth, the data that comprises MIMIC II is highly heterogeneous. It has varying sources, forms, densities, availability, and most importantly varying data structures within the Annotation Station. Despite these huge disparities, the Annotation Station still manages to display all information in only two interfaces. In addition, the distinction between these interfaces is a necessary property of the data display, not a consequence of the disparity in data type. The two interfaces are: the Information Viewer and the Signal Panel, which will be discussed in detail in Section 6. In addition, annotation adheres to this principle of pattern uniformity. Because all data is displayed in one of two formats, it is easy to maintain the property that all data is annotated in the same way. If an annotator sees a notable piece of data and wants to annotate it, he simply right clicks it and selects "Annotate" in the menu that pops up. This is uniform throughout the Annotation Station.

An important principle in the design of the Annotation Station's graphical user interface was the idea that the annotator must continuously have quick access to features that orient him in time. When reviewing a patient's ICU stay, normally of several days length, and viewing different data streams that must be viewed at highly variable resolutions, it can become overwhelming for an annotator to keep track of what time point, what zoom scale, and what data streams he has focused the station on. There are many features in the Annotation Station that deal with this issue. Whatever region of data is currently displayed on the screen, the feature called the Orienter will always show which subregion of the entire patient stay is displayed. For this region, timelines will show the endpoints and clearly identify their times. These timelines also display important information such as clinical progress notes, created annotations, or alarms. The display of created annotations on the timeline is an important feature that allows annotators to be comfortable with time localizing the statements that they've made about patient state. The Annotation Station's current time is prominently displayed on all timelines as well as on Signal Panels. This is important because the annotator doesn't need to do any searching to align different pieces of data. He simply needs to focus the Annotation Station on a time point. All data will be immediately aligned and the time point will be

clearly identified. As a final aid in time localization, the lists of created annotations are listed chronologically, so that reading the lists from top to bottom form a story of the major events over the course of the patient's stay from beginning to end.

A design principle that affects both the way annotators view MIIMC II data as well as the way they view the annotations they've created is that the annotator must have the ability to view data at multiple levels of granularity. A summary of the data must be available for quick review, but the annotator must also have the ability to dig deeper when he recognizes a point of interest. Functionality exists in the Annotation Station that facilitates the idea that an annotator should have an "at a glance" view as well as functionality that allows in depth analysis of all available data. With respect to the MIMIC II data, this dual functionality is achieved through the side-by-side display of the Information Viewer with the Signal Panel. For all available signals, the Information Viewer gives the value of a signal at the current time. If this value seems amiss, the annotator can dig deeper and plot the entire length of the signal on the Signal Panel. With respect to the annotations, the annotations tree gives a quick summary of all the created annotations and how they link together. If the annotator wants a more in depth look at a particular annotation, he can open up a popup box displaying the full state of that annotation.

Data fusion for simultaneous review is a design principle that results from the type of research the Annotation Station is envisaged to support. Multi-parameter trend analysis is a major area to benefit from the produced annotations. Consequently, annotators need the ability to view different streams of data simultaneously and discern patterns indicative of various conditions. The Annotation Station's ability to view arbitrary streams of data side by side and compare them allows annotators to couple data for review as necessary. This approach of allowing display of arbitrary subsets of parameters not only helps the annotator couple the data to interpret one event at one point in time, but also allows him to better correlate events at different points in time within the stay. When the data representing two events is viewed side by side, the annotator has better ability to determine relationships between the two.

Annotating patient records is time consuming work. On average, it takes an annotator 4 hours to digest a case and annotate the main events. Not only is it frustrating for the users of the Annotation Station to lose their valuable work, but even from the economic perspective, considering the dollar value of an hour of a clinician's time, it is entirely unacceptable to lose any amount of data. This leads to the design principle of maximum safety possible for the annotator's work. The Annotation Station implements automatic saving of the annotations at regular time periods during the annotator's work as well as on exit to minimize the maximum possible data loss resulting from any system crash, any user mistake or any user misunderstanding of the interface.

A crucial design principle employed throughout the development of the Annotation Station was that of user involvement. Clinicians were involved in every part of the product lifecycle from conceptualization through development to testing. This ensures maximum usability of the final product.

4.2 Annotation Requirements

The preceding section focuses on the needs of the annotator on the software. But as previously mentioned, there is a second main component to the annotating process, the encoding of the annotator's thoughts into a machine and human useable format. There are several annotation requirements that were developed to serve these goals.

The main uses envisioned for the annotations that will be produced with the Annotation Station are as a research database amenable to human review and also as a labeled test set for algorithm development and verification. The two uses lead to two goals. Firstly, that the annotations are labeled with regular, easily searchable codes that uniquely represent the state that is being described. Without coding the annotator's label, we would simply create a second problem of automatically interpreting free text entered by a clinician. There is little hope of being able to effectively do so in the near future. Therefore, it is a requirement that these annotations must be populated with coded medical terminology. Secondly, that the annotator is able to accurately express himself with regard to the state of the patient. These dual goals conflict when one considers how the annotator should describe his findings in the annotation label. As an annotator reviews a case, he develops quite detailed opinions about what specific combinations of conditions exist and why. After an extensive search of available medical lexicons, the UMLS was deemed to be the best available. Although the UMLS structure for coding physiological state is powerful, it is still often insufficient to fully describe the thoughts of the annotator. In addition, the UMLS also often provides multiple ways of coding the same piece of information without any standard way of mapping the codes to each other. Coding medical concepts specifically, reliably, accurately, and uniquely is a challenge that is as yet unsolved. Nevertheless, it is clear that coding of annotation labels is a requirement for the annotations to be useful in future work on the AMS.

The annotator needs to track the evolution of a particular condition throughout its duration from its onset to restoration of the patient to health, death, or discharge from the unit. Specific markers of the onset and offset of a condition are critical to useful annotations. Without a good approximation of the onset and offset of a condition, there is no basis against which to evaluate the performance of a detection or prediction algorithm. If the AMS is to predict the onset of some condition, it must be known at what point the alarm will effectively be saying, "the patient is in such a condition" rather than saying "the patient will deteriorate into such a condition". This is a very critical difference, which is easy to overlook in testing predictive algorithms. Lack of precise onset times

can invalidate the results of a predictive intelligent alarm. Also, from the data modeling perspective, it is important to know the offset time. If it is unclear where the offset is, derived models can be inaccurate due to training on regions of data where the modeled condition is in fact not present. This makes precise onset and offset times vital to producing useful annotations.

As a way of ensuring that the annotators formulate well-reasoned opinions rather than an expression of biases, it is necessary to require that annotators document their evidence from the available data to support a particular medical hypothesis. When this evidence is required, an added benefit is brought of allowing subsequent annotators to review the produced annotations and quickly determining whether or not they agree with the reasoning behind it. Therefore, data evidence links are useful from the administrative point of view in ensuring annotation quality. Furthermore, these links are useful from the machine learning point of view. Given the tremendous variety of data in MIMIC II, it would be challenging to determine what streams of data to use to predict or detect any condition. Without the links to the data streams, there is no way of immediately selecting an appropriate subset of data to include in a feature vector. Algorithm designers would be forced to use an imperfect subset selection algorithm to try to identify the most predictive streams. It would be clearly better to leverage the annotator's knowledge of physiology to document which streams are the most predictive. Therefore, links to data are required supporting each of the claims that an annotator will make in the annotations about the progression of disease.

It is common for ICU patients to have several concurrent complications that can affect each other. For example, a patient can be in hemorrhagic shock and because of the loss of blood perfusion this can lead to kidney failure. The shock and the kidney failure are in fact two distinct effects and thus two distinct disease processes. Each should be individually evaluated over time so as not to ask the annotator to make an overly complex statement about patient state. But while each disease process is to be evaluated individually, there also needs to be an indication that the shock disease process caused the renal failure disease process. So, in addition to links to evidence in the data, different disease processes should be linked together where one causes a change in the other. This type of annotation is useful for many purposes. Firstly, we may want to isolate classical cases of a particular type of disease process. Too many additional disease processes in the same ICU patient may make the case too complicated to analyze. Secondly, we may want to explore the onset of one disease process as the result of another. If we look at this type of deterioration, there is potential that, in our AMS, we could model the disease progression from one condition to another or one organ system to another. These possibilities necessitate the inclusion of causal links in the annotations.

While labels applied to annotations in a patient record are useful for identification of the set of disease processes active within a case, they do not give a good idea of how well or badly a patient is doing at a particular point in time. In order to remedy this problem, we require that annotators give quantitative responses to several questions including the

overall current state of the annotated disease process as well as the trajectory the annotator believes the patient is taking. Requiring the annotator to respond in this way gives a significant advantage to the AMS algorithm designers. They can immediately make use of the quantitative measure of the state and trajectory of the patient in order to train algorithms to match the results that the annotator gives. This is a powerful feature the annotations offer algorithm developers. Therefore, the quantitative assessments are a requirement on the annotator.

5 Annotation Structure and Methodology

5.1 Annotation Elements

The state of a patient over the duration of their stay in the ICU is a function of a highly complex web of interrelated episodes. Each of these episodes may potentially be relevant to various organ systems and disease processes. This variety of possible effects and the variety of potential meaning for each make it very difficult to annotate these events and apply causal links when appropriate. Many simplifications are necessary in order allow the annotator to focus on a limited task at one given time. To demonstrate how these simplifications are used, the components available for use in creating annotating will be presented, with explanations of how the annotator can focus his annotating task when using each of these.

To begin, the annotator must decide how many distinct complications (e.g hemorrhagic shock, renal failure) he wants to track in the record. These will be the disease processes he is concerned with and must construct annotations for. For each of these disease processes, the annotator must determine an onset time, an offset time, all times he feels the state of the disease process changed positively or negatively, and any factors the annotator feels might have brought about this disease process. All of these determinations must be coded into the annotation components presented in this chapter, which are at the annotator's disposal in the Annotation Station.

5.1.1 The State Annotation

The first important annotation component the annotator will need to instantiate in order to record the progression of a disease process is a State Annotation. A State Annotation must be created at the onset time of a disease process, at each point the state changes during the disease process, and at the offset of the disease process. Within each of these State Annotations, the annotator must indicate several things. Firstly, the name of the disease process that is to be annotated must be explicitly entered and coded. Secondly, the annotator must enter the qualitative assessments of the patient state. Thirdly, the annotator must enter the data-backed evidence for his assessments as links to Flag Annotations, which are described in Section 5.1.2. Fourthly, the annotator must enter links to any previously created State Annotations from other disease processes or from the Problem Annotations that caused the onset or change of state that is currently being annotated. The target of such a link is termed a precipitating factor. The visualization of a State Annotation from the Annotation Station is provided in Figure 17.

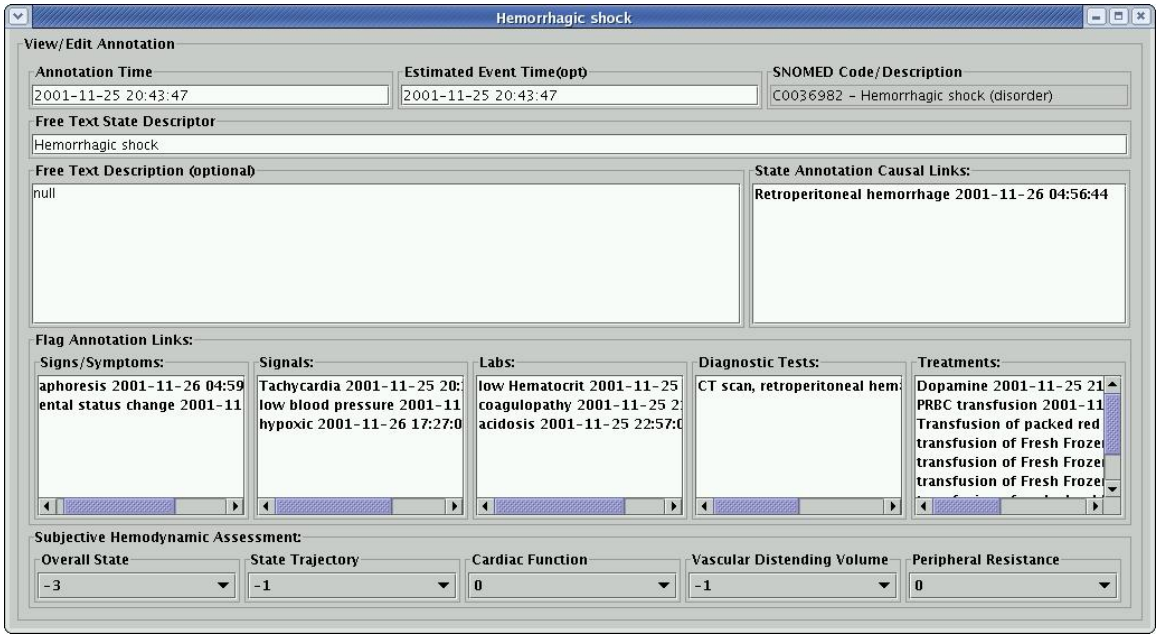

Figure 17. The visualization of a State Annotation

5.1.2 The Flag Annotation

A flag annotation is the compartment that is used as an intermediary between a State Annotation and data that can be used to support the hypothesis presented by the annotator in that State Annotation. The reason for the flag annotation's existence is so that an annotator can add additional qualifying information to the data that he's pointing to. For example, a high heart rate could be observed at some point in the patient record. The annotator would place a flag annotation at that point on the stream of data in the patient record. He would then code the label for "tachycardia" into the flag annotation. As a second example, if an annotator observes an abnormally low pH value, he can create a flag annotation on that value and add the label and code for "acidosis". Additionally, an annotator could use a flag annotation to highlight a region of data as artifactual or as otherwise inaccurate. Flag annotations must be tied to a piece of MIMIC II data, since the annotations' function is to flag, bring attention to, and qualify the data to which they point. As a convention that simplifies the annotation structure, flag annotations may not link to any type of annotation.

5.1.3 The Problem Annotation

The third type of annotation that can be created is called a problem annotation. This type of annotation indicates a pre-existing complication or an important active problem in the ICU. If the annotator notices such an issue as he is reviewing the case, he can label and code it into a problem annotation. A problem annotation is very similar to a Flag annotation, except that because an annotator can discover these problems without using the data streams from the patient's stay in the ICU, the problem annotation need not have

a link to data. Therefore, the only information in a problem annotation is the coded medical term. The problem annotation cannot link to any other annotation. For the purposes of the Annotation Station's annotation structure, the problem annotations form a list of the patient's medical diagnoses.

5.1.4 Putting the Annotations Together

The individual objects described above are used to construct annotations in a specific fashion. The annotation structure dictates a three-tiered web of causal reasoning as shown in Figure 18. At the bottom level, on the first tier, we have the data streams themselves. The information present at this level is the data included in the stream (such as: Arterial BP, Hematocrit, Cardiac Output, or ECG). As a doctor reviews a case on the Annotation Station, he may note a particular piece of data and want to flag it as interesting and relevant to a change in patient state. In our annotation scheme, flagging can be done by creating a flag annotation, a component of the second tier in the three-tiered structure. The flag annotation automatically includes the time and data stream that was flagged and also gives the annotator the opportunity to qualify the tagged piece of data in the previously described fashion.

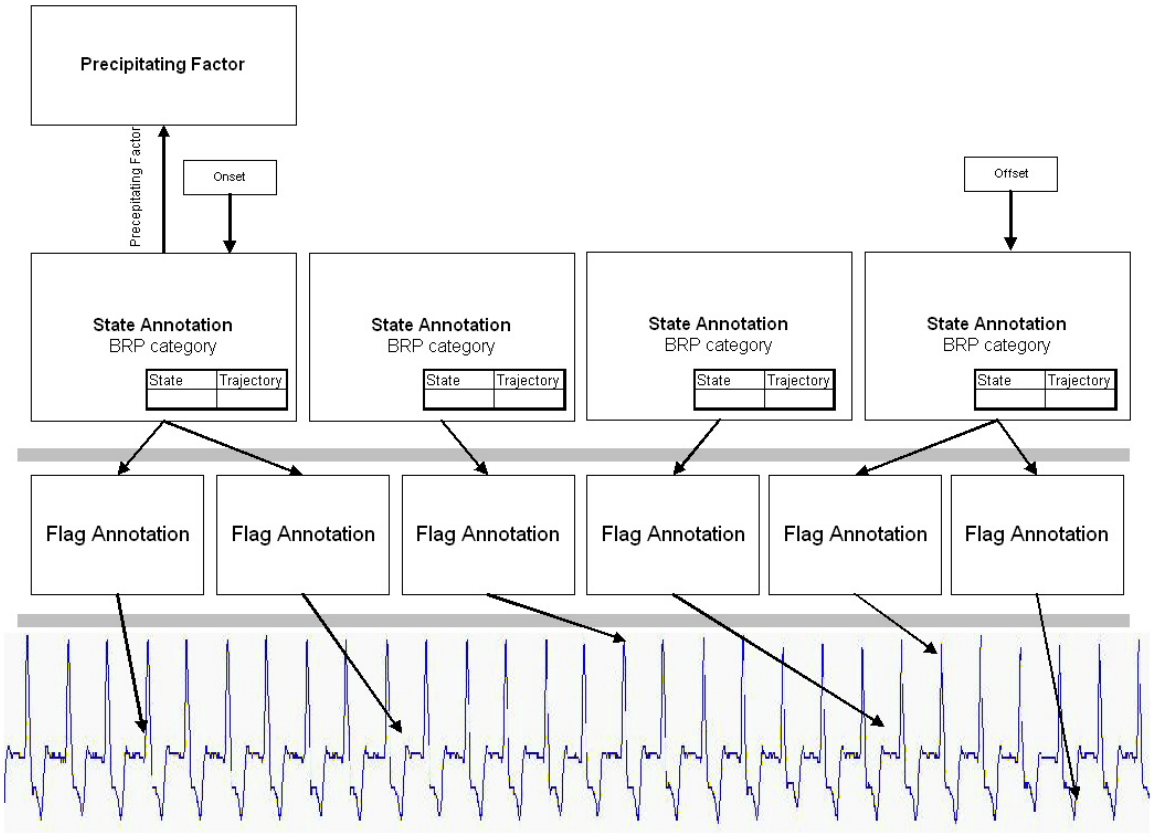

Figure 18. The three-tiered approach to annotating MIMIC II data.

Once all the relevant data evidence is flagged in the second tier, the annotator makes judgments about where there are significant changes in patient state. He then creates the state annotations and links to all of the flag annotations that he considers relevant to this state annotation as well as all previous state annotations and problem annotations that play a role in bringing about the state currently being annotated. For each disease process, it is assumed that all state annotations that are part of the same disease process affect one another. For example, the fact that a patient had hemorrhagic shock of level –2 an hour previous to a state annotation is obviously relevant to the fact that the hemorrhagic shock worsened to level –3 at the time of the new annotation. The proper use of the state annotation links is to identify precipitating factors that are not a part of the same disease process.

For the case where more than one disease process per patient is being annotated, we need to be able to quickly and reliably identify which annotations are relevant to which disease processes. This involves the creation of a list of preferred labels for disease processes to be annotated, and the assignment of one of these disease process labels to each state annotation. This will result in a number of disease processes for which we have trains of state annotations in patient records. The progression of a disease process can then be tracked by sampling the reported state in the train of state annotations. A "disease process signal" representing the progression of the disease process over time can then be quickly and correctly extracted from these state annotations.

5.2 Recording Annotations

eXtensible Markup Language (XML) [24] was used in order to record annotations. There are several properties to XML that make its use attractive for application in the Annotation Station. Firstly, storing annotations in XML means that the information will reside on disk as normal text files, which are highly portable. It is likely that annotations will need to be transferred to many different users using many different operating systems. Using a portable data format like XML is essential in such an environment. Secondly, XML is easily reviewable and modifiable by text editing. Therefore, the annotations can be looked over in a text editor and the contents are not too cryptic for an annotator to be able to spot obvious errors, correct them, and still have a valid annotation file. A third advantage of XML is the ability to apply style sheets. This allows users to specify the appearance of annotations in standard forms, and then use a web browser to view them.

In order to put annotations in XML, an open source publicly available XML class generator, which is part of the Oracle XML Developer's Kit [25] for Java, was used. The generator takes as input what is called a DTD (Document Type Definition). The DTD defines what elements a valid XML document can contain and how they can be

composed together in the document. The DTD used for the annotations is shown in Appendix A. From this DTD, the Oracle tool automatically generates Java classes to represent the elements in the XML file. The Java classes can then be linked to each other to indicate how the XML file should be written. After this structure of Java classes is built, an Oracle routine called print() automatically generates XML files. To be able to load the XML files from disk into the Annotations Station, the XML parser from the Oracle toolkit is used and the output structure is queried to populate the Annotation Station's internal data structures.

5.3 Document Type Definition description

The DTD and each of the elements specified in the annotation will be discussed in this section. At the highest level, each XML document contains one element called AnnotationsForPatient. The AnnotationsForPatient element contains exactly one element called PatientID and a variable number of elements called Annotation. PatientID is indicated as an element containing only PCDATA, which means that it contains only text characters with no additional XML elements. The variable number of Annotation elements each has several components, which will be described sequentially below.

1) id – Each annotation has an id which is identified in the XML document as an "ID" XML element. By XML standard, each ID in the document must be unique. The way we ensure that each annotation's id is unique is to use the following text for the id of each annotation type:
    1. Problem Annotations – "ID-Time_ms-ShortDesc"
    2. Flag Annotations – "ID-Time_ms-source"
    3. State Annotations – "ID-Time_ms"

    Problem annotations, flag annotations, and state annotations all have a different and unique id format. Within each of these annotation categories, their formats are unique to one annotation within a patient record. Specifically for problem annotations, it is redundant to have more than one problem annotation with the same description. It does not make sense to declare the same condition twice in the same patient. As for the flag annotations, if a qualification is put on a specific data stream at a specific time, it must be unique. If more than one qualification is made on one stream of data at one point in time, they must all be in one flag annotation. For state annotations, it should never occur that an annotator attaches two interpretations of the state of the patient at one point in time. If the annotator needs to express more than one idea, he should do so in one annotation.

2) SnomedCode – Text field holding a dash separated list of SNOMED codes applied to this annotation by the annotator.

3) SnomedDesc – A text field containing the descriptions from the UMLS database corresponding to the codes in the SnomedCode field.
4) ShortDesc – A text field containing the text provided by the annotator to try to code his medical interpretation.
5) Time – A text field containing the time of this annotation in the format yyyy-MM-dd HH:mm:ss
6) EstimatedTime – Occasionally the time of the data does not exactly match the time of the event the annotator is trying to annotate. For example, if a nursing note at the end of a shift discusses an event at the beginning of a shift, the Time field must be populated with the time of the note at the end of the shift. However, the optional EstimatedTime field allows the annotator to enter a time to indicate the real time of the event. The text must be in the same format as the Time field.
7) Dt – is an optional field that indicates a time range. For flag annotations, a region of data may be annotated rather than a point. To indicate a region, this field will be present and will contain the length of the region centered on the time in the Time field that this annotation represents.
8) Type – is a required field that indicates what type of annotation this is (0 for Flag, 1 for State, 2 for Problem)
9) Source – indicates the unique source in the MIMIC II data that a flag annotation points to. For state and problem annotations it is null.

   The unique source-name of each of the types of data in MIMIC II is as follows:

   1. CareVue numerical data: tablename-itemid

      The table where the numerical data was taken from (chartevents, totalbalevents, ioevents, medevents) is indicated in addition to the itemid of the value in the database.

   2. CareVue notes: noteevents-from-to

      For notes, the notevents table is indicated. Also, the region of text that was annotated is specified by the offsets within the note as from and to.

   3. Parameter Data: param-signalname

      For parameter data, the fact that this data comes from the parameter file is indicated (with the word “param”), and the specific stream (HR, ABPSys, ABPDias) is added.

   4. Waveform Data: wave-channel

      For waveform data, the fact that it comes from a waveform file (with the word “wave”) is indicated. The channel number is indicated as well.

10) Annotator – indicates the annotator who created these annotations.

11) FreeText – stores the free text data that the annotator entered into the annotations without any attempt at coding. This portion of the justification will only be useful for humans who review the annotations later.

All fields below exist only for state annotations. For flag and problem annotations they are null.

12) OverallState – is a field which stores the annotator's quantitative evaluation of the state of the patient
13) StateTrajectory – stores the trajectory.
14) CardiacFunction – stores the annotator's evaluation of the cardiac function.
15) VDVolume – evaluation of vascular distending volume
16) PeripheralResistance – evaluation of peripheral resistance
17) StateLinks – is an ATTLIST element, which is a built in feature in XML. This represents links in the XML document to other annotations using their unique ids. The annotation ids referenced in this field must be present in the same XML document. The StateLinks link to either state annotations or problem annotations.
18) SymptomsLinks – represents evidences that are symptoms using the same ATTLIST feature. SymptomsLinks link to flag annotations.
19) LabsLinks – represents evidence from Flag Annotations that have lab values.
20) DiagnosesLinks – represents evidence from Flag Annotations that have diagnoses.
21) TreatmentsLinks – represents evidence from Flag Annotations that have treatments.

6 Use of the Annotation Station

Annotators need an organized yet flexible way to view a patient record. As discussed in Section 4.1, there are particular pieces of information that should be available to the annotator by default at a glance. Additionally, the view must be flexible enough so that it can be customized in order to be able to track the progression of the various disease processes that the annotators annotate with the Annotation Station. In order to annotate the MIMIC II database, not only do we need to give doctors the ability to efficiently review patient records, but then we also need to give them the ability to quickly note their beliefs about the progression of the patient state. In the following sections, a description of how the Annotation Station gives the doctor the ability to perform these two tasks is given.

6.1 Viewing - Data Presentation in the Annotation Station

The choice of view in the Annotation Station is driven by the idea that annotators need the ability to view a patient record at many different levels of granularity. They also need a flexible, yet organized way to view the patient data. In order to accomplish these goals, the Annotation Station gives the annotator four screens. One is dedicated to a high level view of all available data at the current time of the system. This is called the Information Viewer. A second panel gives the annotator the ability to view the evolution of pieces of data over time through plotting. A third panel to view waveform plots is also given. Finally, a fourth panel is given to organize annotator's annotations. On this panel, there are lists of previously created annotations as well as a tree to give a quick view of the annotation links in the case.

6.2 Information Viewer

The first screen that the annotator views is the information viewer shown in Figure 19. At the top of this screen are a series of timelines. The first one is called an Orienter, because it orients the annotator to which region of the patient record he is viewing. The Orienter is labeled as element A in Figure 19. The second timeline displays notes and annotations, as is shown in label B. The third is an events timeline, which displays the color-coded alarms and is shown in Figure 19 as element C. There is a well-defined Timeline Java interface, which will be described in detail later, such that timelines with new information can easily be added if needed. The function of the orienter is to show the region of the patient stay that we are zoomed on at the moment with respect to the entire length of the patient stay. So the beginning of the orienter shows the admission date to the ICU, the end of the orienter shows the release date, and there are hashes at every midnight in

between these two times. The orienter highlights the region of interest the annotator is currently investigating as a blue bar across the currently zoomed time.

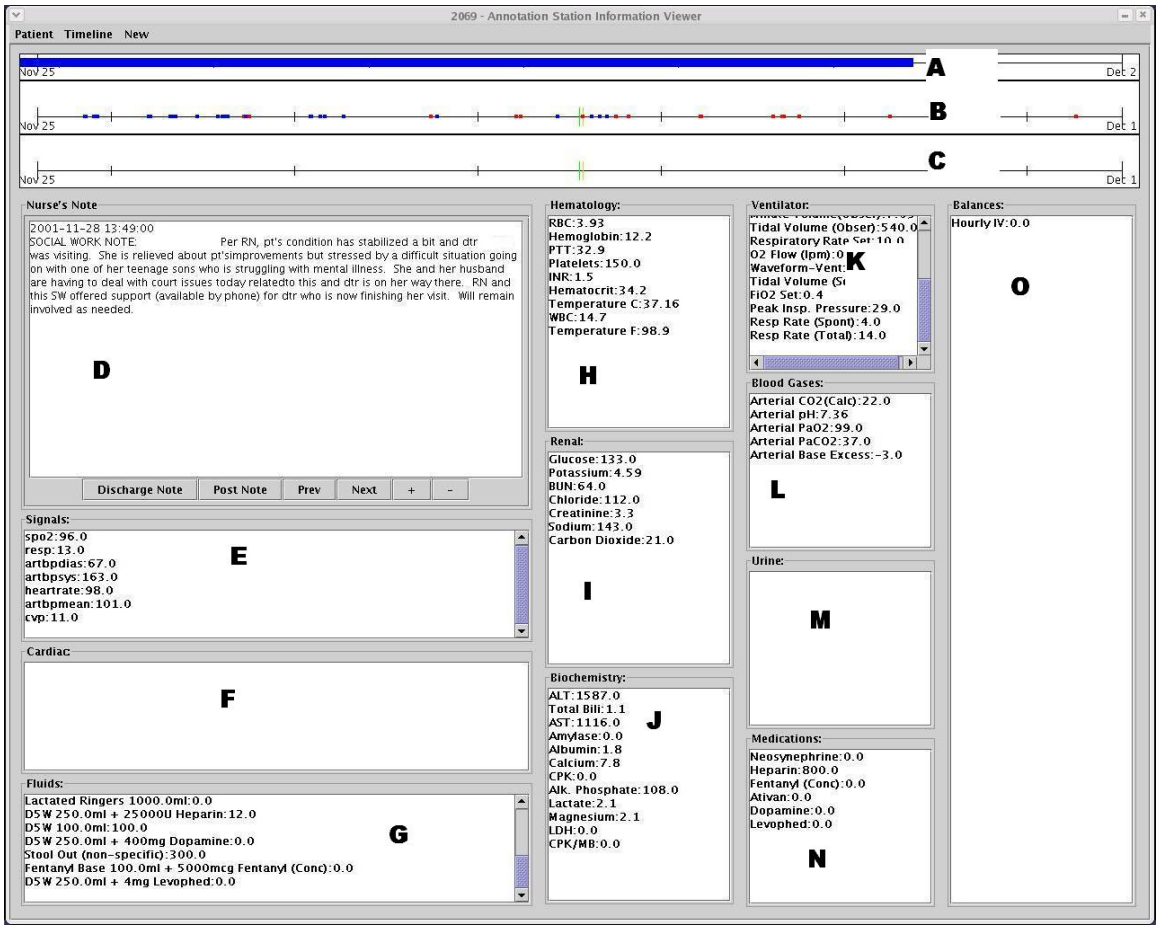

Figure 19. Annotation Station Information Viewer

There are two ways to control the orienter:

(1) The blue bar on the orienter is draggable. This dragging shifts the region being currently viewed.

(2) The orienter's zoomed region can always be reset to the entire patient stay from the pull down menus by selecting the "Timeline" menu and selecting the "Zoom Original" option.

The main timeline (element B) below the orienter shows all annotations created by the annotator as blue dots on the timeline at the time point of each annotation. Clinical progress notes are also plotted as red dots, so the annotator knows where they were recorded during the stay. The endpoints of the main timeline are the endpoints of the blue bar within the orienter. The hash marks indicate the midnight lines between these two times. The orange hash mark indicates the specific note being displayed in the nursing-

note-viewer. The green hash mark indicates the "current time" for all the numerical data displayed below (lab tests, physiologic parameters, etc.). Specifically, for each parameter, the most recent value at or before the "current time" is displayed.

The main timeline is the driver for the whole system. Actions, such as zooming and scrolling on the timeline, can cause all display elements (trend plots, lists of lab results, waveforms) on the annotation station to be updated based on the timeline's new view. Also, performing these zoom and scroll actions on the display elements in the annotation station can (again optionally) cause the timeline to be updated based on the new view selected by the annotator. If the annotator zooms or scrolls to a different temporal section of the medical record in some other part of the Annotation Station (for instance, the notes viewer, or the trend plots), the "current time" will change as well. For instance, if, in the nursing-note-viewer, the annotator moves forward or backward through the notes, the "current time" will automatically move with the time of the currently viewed nursing note. If the annotator scrolls through a different portion of the trend data, the "current time" will also change (assuming the trend plots and the timelines have not been "desynchronized"). Synchronization will be described in Section 6.3.

Below this main timeline, there is an events timeline (element C), which plots the alarms that the CareVue monitor produces. The use of the alarm timeline is similar to that of the main timeline in all respects except that it contains the color-coded alarm information instead of created annotation and note times.

There are several actions that can be performed on these timelines

(1) Left click to zoom in to 50% of the current zoom

(2) Right click to zoom out to 150% of the current zoom

(3) Left click and drag right to zoom to a region

(4) Middle click to update the current time described above

Further down on the Information viewer are several types of information. There is a clinical progress notes pad where the annotator can sequentially review the notes (element D in Figure 19). The note viewer is rather simple: The discharge summary can be viewed by clicking on the "discharge summary" button on the viewer. Clinical progress notes can also be viewed in this box.

There are two ways to control the notes viewer.

(1) From the note viewer itself, next and previous notes can be displayed by clicking the next and previous buttons.

(2) If the annotator would like to view a note at a specific time as shown on the timeline, he can set the current time on the timeline near the time of the note (by middle clicking near the red dot). The note corresponding to that dot will then appear on the nurse's note viewer, as it will be the nearest one.

All data described as Chartevent data, drip medication rates, as well as fluid balance information is displayed in text boxes (elements E, F, H, I, J, K, L, M, and O in Figure 19) and placed on the information viewer. These boxes represent the subclasses of data, which were broken down by the ReadCEFile program as described in Section 2.3.1. The elements E, F, H, I, J, K, L, M, and O correspond to the following list of sections from the breakup file respectively:

- High-Rate Measurements (SIGNALS)
- Cardiac (CARDIAC)
- Hematology (HEMATOLOGY)
- Renal (RENAL)
- Biochemistry (BIOCHEM)
- Ventilator (VENTILATOR)
- Blood Gases (BLOODGASES)
- Urine (URINE)
- Total I/O (TOTALIO)

Each signal specified in the breakup file will appear in the appropriate box, if available. Boxes G and N correspond to the I/O fluids data and the medications data respectively. The information in each comes directly from the appropriate tables in the database as described in Sections 2.2.2.3 (fluids) and 2.2.2.2 (medications). The nearest previous value to the current time is indicated for each signal within the boxes. The annotator also has the option of plotting a time series of any of these signals on the signal panel. The parameters displayed represent a subset of the available MIMIC II parameters. They are comprised of only those signals indicated in the signal breakup file that was used to sub select parts of the database for use by the Annotation Station.

There are several functions that can be performed with the data in the text boxes on the information viewer.

(1) To create a time series plot of any parameter on the data display page, right click the parameter of interest and select "chart as time-series".

(2) To view the time that this value originated from, left click the parameter of interest and look at the timeline. A large green dot will appear at the real time of this measurement.

6.3 Signal Panel

The Signal Panel, which can be seen in Figure 20 with the panel labeled P, can plot the annotator's choice of an arbitrary number of signals simultaneously. These plots are zoomable and scrollable. There are also specific types of data that cause incorporation of several data streams into one plot. For example, the unverified systolic, diastolic, and mean arterial blood pressure parameter data are merged with the verified CareVue data and displayed simultaneously. The parameters are displayed at the resolution of one minute, while the asynchronous nurse verified values are overlaid to give the annotator the maximum amount of relevant, related information in the minimum space possible. All of these plots can optionally control the active time range that the annotation station is focused on if the annotator wishes to keep the signal panel synchronized. As a default, the orienter's blue bar is synchronized with the time-series window used on the trend-plot page, and the large vertical green line corresponds to the "current time" in the annotation timeline.

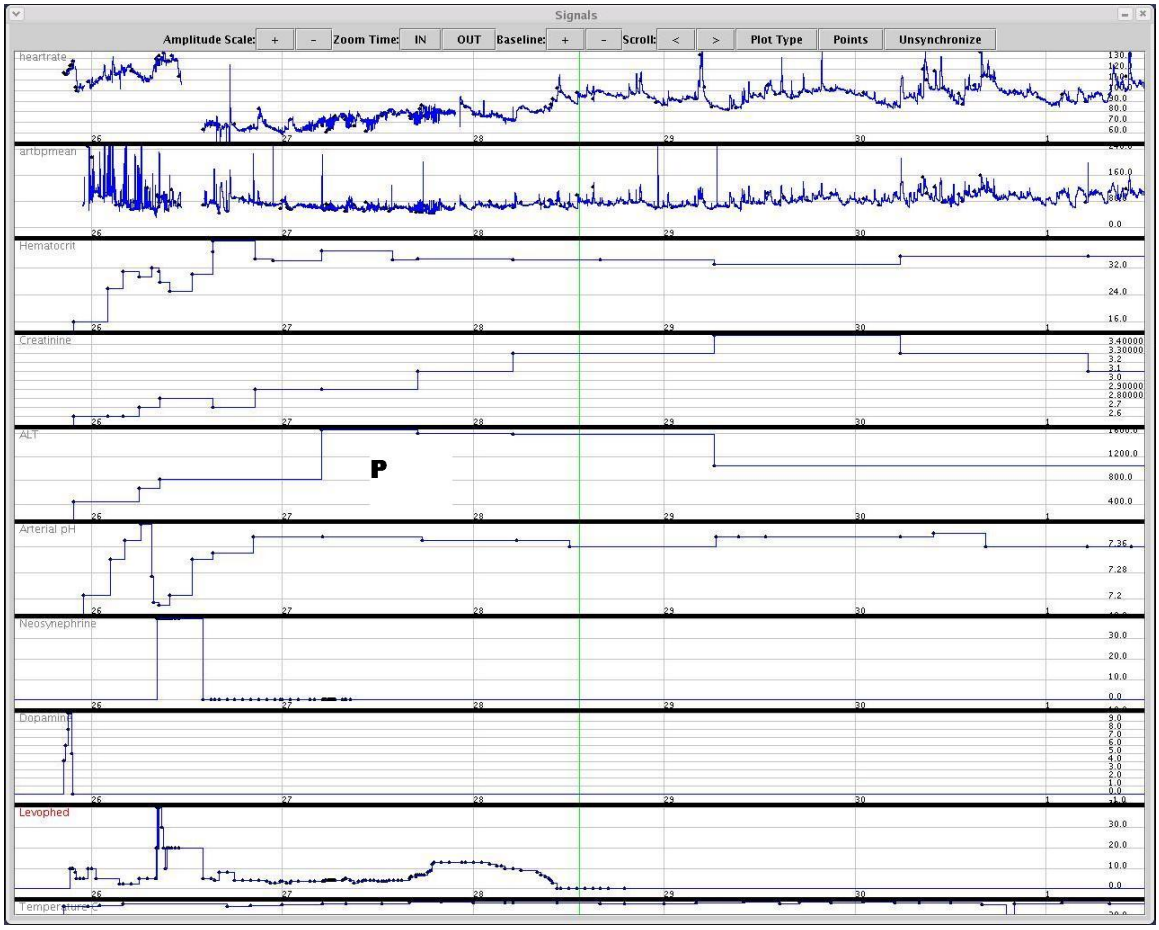

Figure 20. Annotation Station Signal Panel

There are many functions that can be performed with the signals viewer. Some important ones are listed below.

(1) The annotator can make one chart in the series of strip charts “active” by clicking that time-series plot. When he does this, the label will turn red indicating it is now the active time-series plot.

(2) The individual parameter values of any chart can be determined by placing the cursor over the data point. A tooltip box will appear and display the parameter value as well as the time of that value.

(3) The annotator can zoom in on a subsection of a time-series chart by clicking-and-dragging. This will also change the orienter’s endpoints, and the “current time” will become the middle of the new time window (if the signal panel is synchronized).

(4) If the annotator toggles off the desynchronize button, the orienter and current time will no longer synchronize with the time-series displays.

(5) There are buttons at the top of the trend plot page to zoom in or out.

(6) There are buttons at the top of the trend plot page to scroll through the time-series data forwards and backwards.

(7) For individual time-series charts, the annotator can customize the amplitude or the baseline by using the buttons at the top of the trend plot page.

(8) For the active time-series chart, the annotators can change the plot type. The Annotation Station supports stem plots, sample-and-hold, and connected points. This is done by clicking the "plot type" button at the top of the trend plot page and selecting the appropriate type. The Annotation Station attempts to automatically select the correct type of plot based on the type of data. For example, the medications data are rates that should be correct until the next recorded value. Thus, sample and hold is the correct plot type for medications. For the fluids, stem plots are the correct plot type because volumes are reported for a particular time. The annotator is free to choose whichever plot type suits his needs, but should remember that this is simply a representation of underlying data. He should be aware not to come to false conclusions based on the plot type chosen.

(9) If the annotator right clicks the active time-series, a pull up menu with several options appears, including annotating and deleting the chart.

6.4 Waveforms

The waveforms page is functionally identical to the trend plot page except for some minor differences. The waveform pane is shown as label Q in Figure 21. Since the waveforms are such dense information and are viewed at such a minute scale with respect to the rest of the data on the annotation station, the pane requires special treatment. Firstly, the waveforms are not automatically loaded at any time. If the annotator wishes to see the waveforms available at the time of a particular event in order to verify his assessment of the patient's condition, he must click the load button at the top of the waveform page. This will cause 20 minutes of data around the current time to be loaded. The interaction between the waveforms and the rest of the data displayed within the Annotation Station is also different. Zooming and scrolling the waveforms will not affect the rest of the data. The region of interest does not change and the current time of the system does not change. However, when data in the Annotation Station updates the system's current time or zoom region, the currently viewed time of the waveforms changes. The zoom of the waveforms is unaffected by the rest of the Annotation Station because the time scales on which this beat by beat data is viewed are so much smaller than the rest of MIMIC II data.

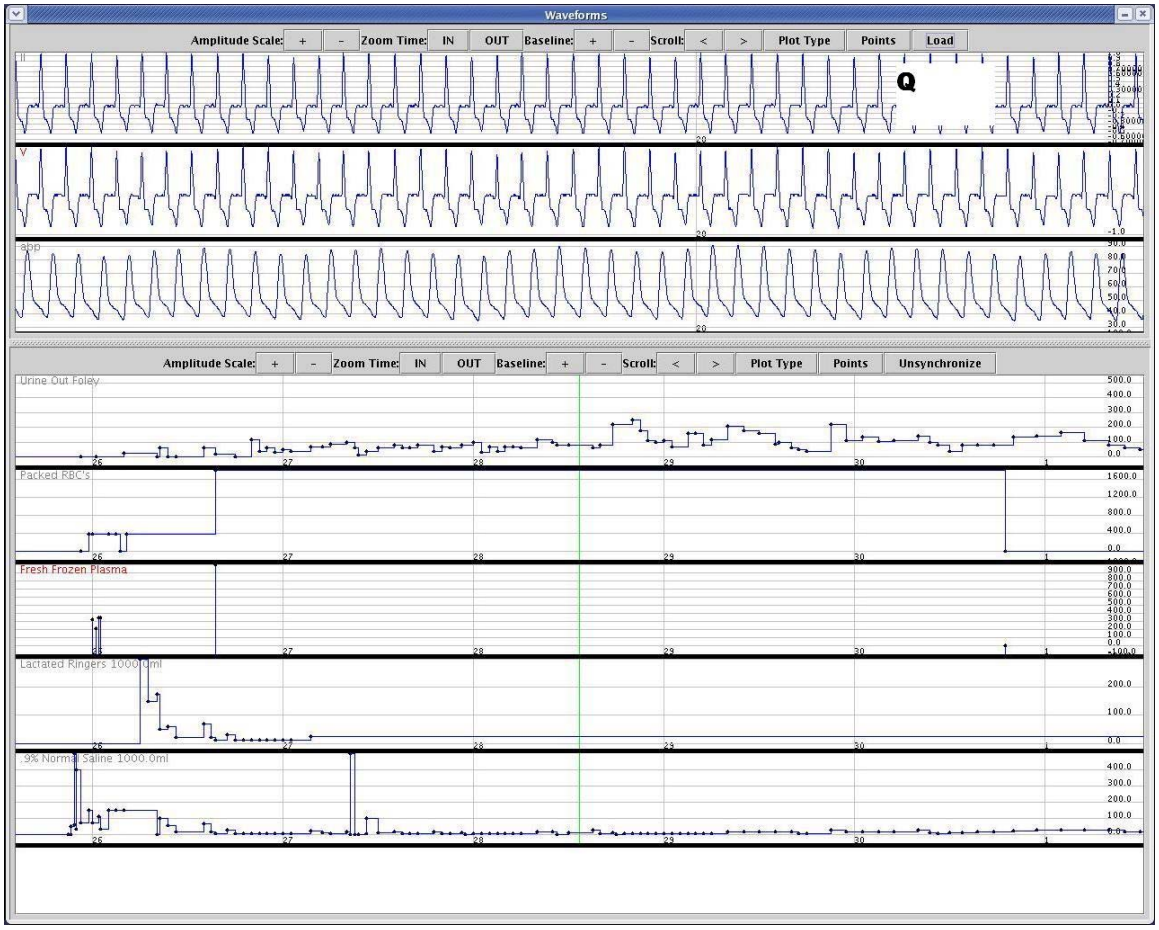

Figure 21. Annotation Station Waveform Panel

6.5 Annotations

Every piece of data in the annotation station can be used as the source for the creation of a flag annotation. The flag annotation will automatically take the time and data source of the selected stream. State annotations can be created with time anchors from data streams, but state annotations do not have a data source. Therefore, state annotations will only take the time from the selected data source. All created annotations are displayed on the Annotations Pane as seen in Figure 22.

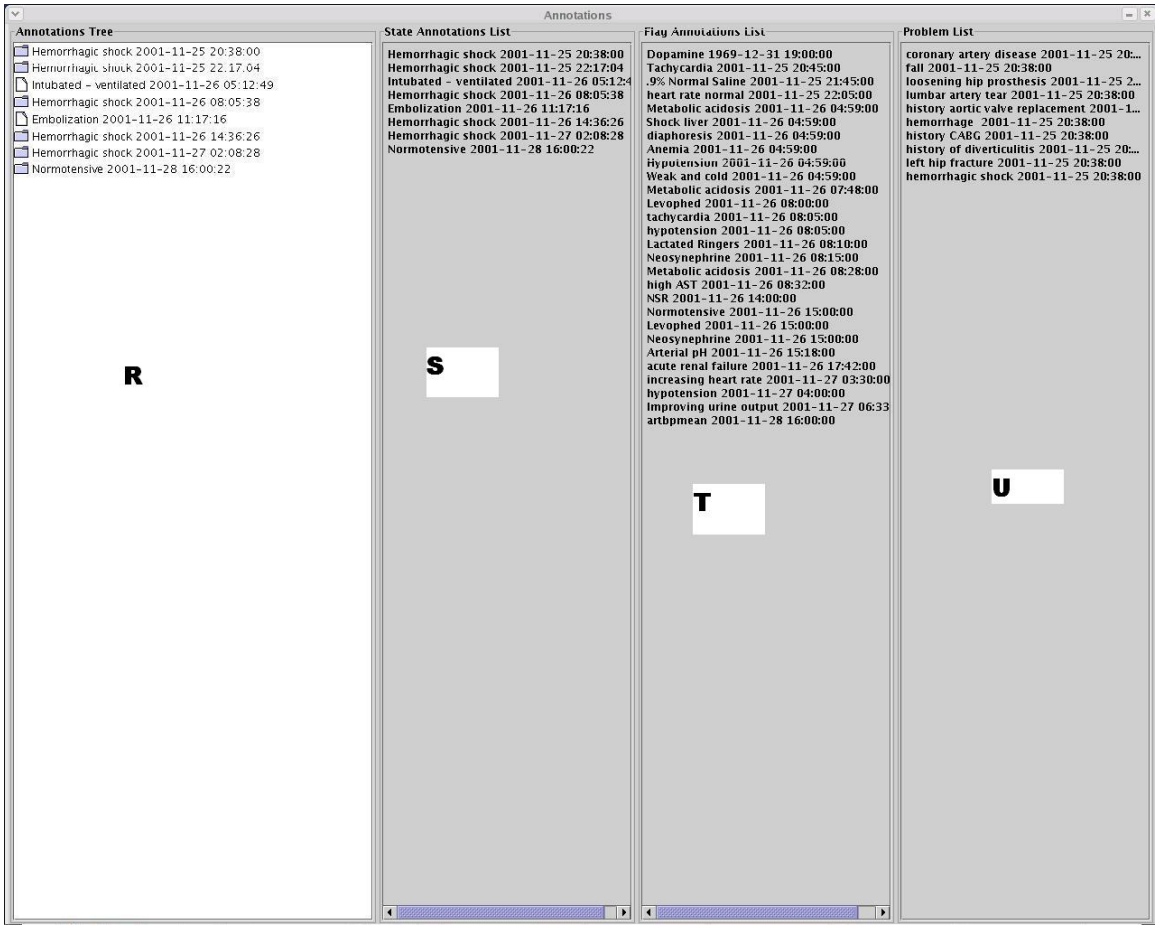

Figure 22. Annotation Station Annotations Pane

6.5.1 Flag Annotations

Flag annotations are qualifications that attach to individual data. For instance, a laboratory value for hematocrit equal to 23 might be flagged and given the label "anemia". Or, if the nursing note suggests that that hematocrit might have been artifactual, it might be flagged "artifact". A section of a nursing note can be flagged and given a label. A portion of the ECG might be flagged and labeled "ST segment elevation".

There are several ways to create flag annotations in the Annotation Station

(1) The annotator can annotate a portion of a time-series chart by clicking that chart so it becomes "active" (its label turns red). Then right-click that active chart and select the annotation option. This will annotate the available data point closest to the point on the plot that was clicked.

(2) The annotator can also annotate a measurement displayed on the information viewer by right clicking the value and selecting the annotation option.

(3) The annotator can also annotate any section of free text by highlighting the section (click-and-drag) then right-clicking the selection and selecting the annotation option.

When created, flag annotations appear in the Flag Annotation List of the Annotations screen, which can be seen as element T in Figure 22. The created flag annotations are listed chronologically from the beginning of the stay with the earliest flag at the top and the latest at the bottom of the list.

6.5.2 Problem Annotations

In any ICU case, there are preexisting problems and complications that the patient has upon admission or develops during their ICU stay. The annotator needs a way to record, code, and reference these problems inasmuch as they are precipitating factors to the disease processes the annotator wishes to annotate. Problem annotations are extremely similar to flag annotations except that they are not based in the data (the annotations do not have a source in the data). They do not represent a point in time during the patient stay and they cannot themselves have causal links to any other annotations. Problem annotations should be created while the annotator is initially reviewing a case and developing an overall understanding for the patient's ailments, before trying to record the details of the disease process's progression over the stay. In order to create a problem annotation, the annotator should:

(1) Click on the menu titled "New" then select the option "Problem" This will popup a box asking for the name of the problem. After giving the name, the problem annotation will appear on the problem annotations list (element U in Figure 22) the same way state and flag annotations appear on their respective lists.

(2) Once on the list, problem annotations can be edited, deleted, and linked in the same way flag annotations are.

6.5.3 State Annotations

State Annotations codify a medical state of interest that relates multiple pieces of information in the patient record. For instance, when the data suggest the patient had chest pain, ECG changes, and rising cardiac enzymes, the underlying state could be myocardial infarction. This state annotation, by definition, involves the relationship of

multiple pieces of information in the MIMIC II data. Moreover, state annotations can exhibit causal relationships between themselves (such as hemorrhage causing hypotension causing ischemia).

There are two ways to create state annotations:

(1) State annotations can be created from the main timeline. The annotator can hold ctrl while he left clicks the point at which he wants a new annotation. A new state annotation will be created.

(2) State annotations can be created from the signal plots. The annotator can hold ctrl while he left clicks at any point on a trend. This will create a new state annotation at that time point.

When a new annotation is created, it appears in the State Annotation List (element S in Figure 22) on the Annotations Screen. It also is displayed as a blue dot on the main timeline. The State Annotation List has the same functionality as the Flag Annotation List.

By double-clicking any annotation in any of the Annotation Lists, the Annotations Popup Box opens up, on top of the information viewer as shown in Figure 23. The Annotation Popup Box allows the Annotator to edit many of the properties of an annotation. Some of the main editable fields in the Annotation are:

(1) Times: The time of Annotation, or estimated time of an event can be updated in the Annotation Popup Box if a correctly formatted time string is inserted in the appropriate field.

(2) State descriptor: The field where a name is assigned to the state, such as cardiogenic shock, hemorrhagic shock, etc. The Annotator can hit return. This state can then be SNOMED coded using Shu's SNOMED coding scheme [10].

(3) Free text description: is where the annotator can make notes. These notes will not be coded and cannot be immediately used by a machine in an automatic way. Nevertheless, these notes will be useful to algorithm designers who wish to examine what the annotator had said about the annotation.

(4) Causal links: State annotations can be click-and-dragged from the state annotations list into the State Annotations Causal Link box, to show they have a causal relationship with the active state annotation. In a similar way, all other evidentiary annotations (flag annotations from the flag annotations list or problem annotations from the problem annotations list) can be click-and-dragged to the appropriate evidence boxes, which are described below.

(5) Subjective Assessments: These numerical assessments are useful in quantifying how well the annotator feels the patient is doing hemodynamically and what the current trajectory is as well as cardiac function, distending vascular volume, and peripheral resistance.

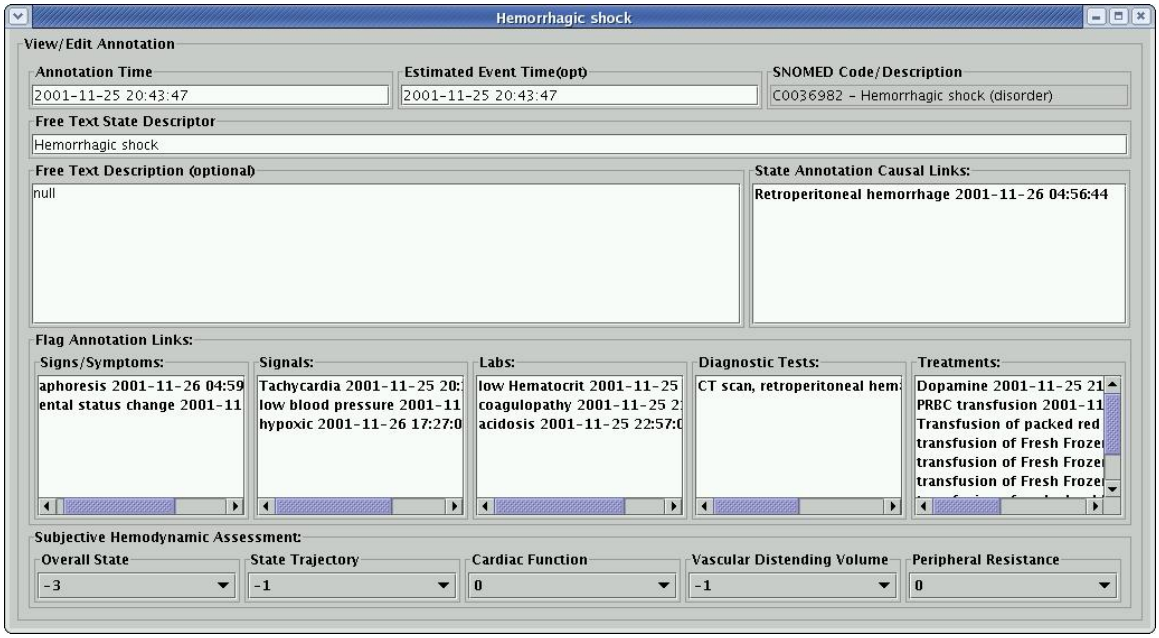

Figure 23. The Annotations Popup Box

The links and assessments will only be available for state annotations. Other types of annotations cannot contain this information.

The fields “signs/symptoms”, “signals”, “labs”, “diagnostic tests”, and “treatments” exist for filing purposes only. They are all ways to represent evidence. “Signs/symptoms” refers to the clinical presentation of the patient, probably noted by the nursing note (e.g. patient is coughing). “Signals” refers to biosignal evidence (e.g. low blood pressure). “Labs” is for laboratory measurements. “Diagnostic tests” refers to other tests that support the state, such as CT scans, X ray results, etc. “Treatments” include pertinent therapies that support the state annotation.

The meanings of the subjective assessments the annotator makes are shown in Appendix C.

6.5.4 Annotations Tree viewer

Once annotations have been created, they can be easily and quickly reviewed using the Annotations Tree (element R in Figure 22). This is a hierarchical view of the created annotations, which has several features:

(1) An annotator can click on any state annotation to view the flag annotations linked as evidence in a hierarchical fashion in the Annotations Tree.

(2) An annotator can double-click on any annotation in the tree to open the associated Annotations Popup Box. This is an alternative to the same action performed on the annotation lists.

(3) By clicking on any annotation in the annotations tree, a large blue dot is displayed on the main timeline indicating the time of this annotation. Also, all other annotations that have been linked to this one flash in red on the Annotation lists.

6.5.5 Deleting an annotation or link

While annotating, an annotator might realize that he's made a mistake in annotating and wish to remove an annotation or a link. The Annotation Station gives the ability to do each of these tasks in the following way:

(1) To delete a flag, state, or problem annotation, the annotator must locate the annotation in the appropriate list on the Annotations Window. He must then hold the ctrl key while left clicking the annotation he wishes to remove.

(2) To delete a link, the annotator must open the Annotation Popup Box of the annotation where the link must be removed. Within the evidence lists, the annotator can hold the ctrl key wile left clicking the link. This will remove the link from the active annotation to the listed annotation.

7 Documentation of Java Interfaces

In this section, documentation is provided for the major classes and interfaces used to produce the GUI of the Annotation Station. Only methods and fields relevant from a functional point of view will be discussed. The purpose of creating each of these important classes and interfaces and their proper usage will be indicated. An important interface that has already been discussed in Section 3.1 is ReadCEFile. This description is tied to Data Loading and is not repeated in this section, but understanding the data loading procedure is critical to understanding how many of the GUI objects operate.

7.1 AW.java

AW (short for Annotation Workstation) is the main driving program. This is the class that is run to launch the Annotation Station. It creates all of the necessary frames and windows and populates them with the Java GUI Components that allow the Annotation Station to function. Some of the main fields and methods within AW.java are:

1) HashMap createdAnnotations

The createdAnnotations field is a HashMap that maps each unique annotation ID to a JLabelWithAnnotation that is displayed on the Problem, State, or Flag lists on the Annotation Window. The createdAnnotations map is used when a handle is needed to an object called a JLabelWithAnnotation, which represents the annotation. This JLabelWithAnnotation is a customized Annotation Station component where the underlying annotation that the visual component represents can be accessed.

2) HashMap annotationsMap

The annotationsMap field is a HashMap that maps each unique annotation ID to an object called an InternalAnnotation that represents it. This map is in fact redundant to the createdAnnotations map, but it is often convenient to be able to immediately access the InternalAnnotation without having to go through the JlabelWithAnnotation. Of course, given the redundancy, it is an invariant that must be maintained in the system is that the InternalAnnotations within the JlabelWithAnnotations that are contained in the createdAnnotations HashMap must be exactly the same set as the InternalAnnotations contained as the values of the annotationsMap HashMap.

3) HashMap problems

Because problem annotations are dealt with in the system separately from flag and state annotations (unique ID construction is different, the actions taken to create them are different, their display is different), they are stored in a separate location. The HashMap where these annotations are stored is called problems. Problems maps the unique IDs of the problem annotations to the JlabelWithAnnotation that are displayed in the problemList. The InternalAnnotation, the underlying data structure, can be provided by the JLabelWithAnnotation.

4) public void zoomAll(long beg, long end, long curTime)

As the AW class is the only one that maintains information about all the JComponents displaying the patient record on the Annotation Station's screens, it is necessary that all synchronized changes to zoom must be reported to the AW class. Therefore, a general zoomAll function exists in the AW class. The arguments are all long values representing times given in milliseconds since epoch. It will set the endpoints of the entire system (timelines, information viewer, signal panel, waveform viewer) to start at the time beg, end at the time end, and to set the current time of the system as well as all of the subcomponents to curTime.

5) public void updateTime(long time)

updateTime performs a similar function to zoomAll, except that it moves the exact size window that is currently being displayed on the Annotation Station so that it is centered on the new time given. The format of the time is the same as that in zoomAll.

6) public long nearestTime(long targetTime)

nearestTime searches for the element on the main timeline which is closest to the targetTime argument and returns it. This method is used when the user clicks on the timeline and wants the system to update it's time to the nearest element on the timeline. This nearest element can be either a nurse's note or a state annotation, both of which are displayed on the timeline.

There are several helper functions that exist to make loading data at the appropriate times for each of the viewable components in the Annotation Station easier. The following are some of those functions:

7) public void loadAllInfo(long time)

loadAllInfo updates all of the parameters displayed on the Information Viewer so that they are the closest previous available value to the given time. It iterates over all text boxes for which the parameter information needs to be updated and uses the loadInfo function on each of them to update the display.

8) public void loadInfo(HashMap map, JTextArea area, long time, Vector elements)

The loadInfo function takes as an argument the signal name to itemids map for the text display areas in the Information Viewer. These maps can be retrieved from the ReadCEFile object, which is initialized on startup. The information relevant to populating the Information Viewer text boxes is the name of each of the signals. The data has already been queried out of the database and put into SignalData objects. The loadInfo function takes the names of the signals for each box, then uses the AW's master map of signals to search for each signal by name and return the SignalData structure. The loadInfo method then finds the closest previous value in that signal to the time argument, adds a row to the appropriate JTextArea, adds a value to the elements vector to keep proper bookkeeping of what value is sitting in which text box, and terminates.

9) public void loadNote(long time)

The loadNote function queries the database for the closest note to the given time, displays it in the nurse's note box and updates the system time to reflect the time argument.

10) public void loadAlarm(long time)

The loadAlarm function finds the alarm time nearest to the time argument and updates the system time to match the time argument.

11) public void updateSignals(long time)

The updateSignals function recenters all of the signals panes in the system and updates the system time to the time argument.

12) public void populateTimeline()

The populateTimeline function repopulates the main timeline by querying the database and reading the annotation's times from the annotationTimes field in AW.

13) public void loadPatient(String pid)

The loadPatient function performs many functions necessary for the Annotation Station to operate. Firstly, it clears all the views and internal data structures of any residual information present from a previously reviewed case. Secondly, it loads data from the data repositories for the given pid as described in the Section 3. This description will not be repeated here.

7.2 JLabelWithAnnotation.java

JLabelWithAnnotation (JLWA) extends the built in Java JLabel class. Because JLWAs extend JLabel, they can be easily added to Java JPanels and used as graphical components. In addition, the JLWA class has a field to include the Annotation Station's internal data structure, InternalAnnotation. The JLWA must be constructed with an InternalAnnotation, and this InternalAnnotation is immutable for the life of the JLWA. The JLWA has one accessor function:

1) public InternalAnnotation getIA()

the getIA function returns the InternalAnnotation associated with the JLWA.

7.3 InternalAnnotation.java

InternalAnnotation is a data structure that represents the state of any annotation created in the Annotation Station, including problems, states, and flags. The InternalAnnotation class has fields for all annotation elements as described in Section 5.3. Internal Annotation has vectors, which hold the annotation links. Some of the important functions within InternalAnnotation will be described:

1) public void populate(AnnotationsPopupBox apb)

The populate function populates the AnnotationsPopupBox (the box that shows the user the state of the annotation and allows him to edit it) with all of the information currently in the InternalAnnotation structure. This function is called before the AnnotationsPopupBox is displayed so that its state is consistent with that of the underlying InternalAnnotaion.

2) public void incorporate(AnnotationsPopupBox apb)

The incorporate function copies the fields of the AnnotationsPopupBox into the InternalAnnotation. This routine is executed before an AnnotationsPopupBox is closed so that the updates the user made to the AnnotationsPopupBox are recorded in the underlying Internal Annotation data structure.

3) public Annotation produceXMLAnnotation()

The produceXMLAnnotation function takes an InternalAnnotation and produces an Annotation, which is the class automatically generated by the XML parser (see Section 5.2) to represent an annotation in XML. The annotations, once in the format required by the XML parser, can then be written to disk using the automatic utilities provided.

7.4 AnnotationsPopupBox.java

The AnnotationsPopupBox is a class within the AnnotationStation that holds a JDialog box, which displays JComponents that represent an Annotation to the user. See Figure 17 for an illustration. The constructor for the AnnotationsPopupBox takes as an argument the InternalAnnotation that this box represents. The InternalAnnotation is immutable for the duration of existence of the AnnotationsPopupBox.

1) public AnnotationsPopupBox(InternalAnnotation ia)

Within the AnnotationsPopupBox, there is a static map to track which AnnotationsPopupBox belongs to which InternalAnnotation.

2) static HashMap annToBox

This map has an InternalAnnotation as a key and the corresponding AnnotationsPopupBox as a value.

7.5 AddEvidenceActionListener.java

The AddEvidenceActionListener class is an ActionListener that is added to several components on the Annotation Station to allow for adding flag and state annotations to the Annotation Station's lists of created annotations. All annotations are created through a unified static method regardless of which component on the Annotation Station initiates annotation creation. This method has the following signature:

1) public static InternalAnnotation displayAnnotation(int type, String time, String estimatedTime, String dt, String source, String shortDesc, String freeText)

This procedure takes the initial information necessary to create an annotation, adds it to the data structures that store created annotations (as described above). It then returns the created InternalAnnotation. The arguments of this method are described in Section 5.3.

7.6 Timeline.java

The Timeline class contains code that controls the Annotation Station's timelines. The interface to the Orienter class is extremely similar. Therefore, the interface will only be described once. The Timeline is a subclass of Graph which is a subclass of JPanel. The Timeline is, consequently, a displayable Java component. The interface to it is as follows:

1) public void setCurTimeToMax()

setCurTimeToMax and the corresponding function setCurTimeToMin set the current time of the timeline to the minimum or maximum of the timeline's domain.

2) public void setEndpoints(long base, long fin)

The setEndpoints function rezooms and recenters the timeline so that the earliest time on the timeline is the base argument and the latest time on the timeline is the fin argument.

3) public void setCurrentTime(long t)

The setCurrentTime function updates the current time which is shown on the Timeline as a green hash mark.

4) public void resetData()

The resetData function clears all information (data points to be displayed) from the timeline.

5) public void addItem(String info, long time, Color c)

The addItem function adds a point to be displayed on the timeline. The text in the info is what will be displayed in the tooltip when the annotator pauses the mouse over the timeline and the added item is the closest one to the location of the mouse. The text is arbitrary and can be used for any additional information that needs to be conveyed about a dot on the timeline. The alarm text is placed as info on the alarm timeline and the note time is placed as info on the main timeline. The time argument is the time on the Timeline at which this data point should be displayed. The color is an optional argument that can be used to indicate the color with which the dot on the timeline should be displayed.

A second feature built into the Timeline class besides displaying a list of points chronologically is flashing colors at points on the timeline to draw attention to them. The interface to do this is:

6) public void flash(String time, Color c)

When the flash function is called, a dot slightly larger than those used for information is displayed on the timeline in the color indicated by c. The flash function is used to draw attention to the point where an annotation lies on the timeline or to indicate what time point a particular data value in the Information Viewer came from.

7) public void unflash()

The unflash function clears the timeline of a previously ordered flash.

7.7 SignalPanel.java

SignalPanel is a class that is used to display trend plots on the Annotation Station. The SignalPanel points to some number of SignalData elements, which it displays sequentially according to the specified SignalProperties. SignalPanel is a custom component, and a subclass of JPanel, meaning that it can be displayed as a GUI object in Java.

1) public SignalPanel(boolean syncTime, SignalPanel overflow)

A SignalPanel is constructed with a syncTime argument to synchronize the zoom of the data on the panel with the rest of the Annotation Station. In addition, if a second SignalPanel is passed in the overflow argument at the time of construction, this SignalPanel can overflow to that SignalPanel. Meaning that when the panel is full, and there is no more visible area on the panel to add signals, the SignalPanel will add the signal to the overflow SignalPanel as well as this one.

2) public void addData(SignalData sd, int type)

The addData function allows the addition of an object that implements the SignalData interface to be plotted by the SignalPanel. The SignalData gives information about the time points and data values of the signal. The optional type argument (with the possible values of SignalProperties.HOLD, SignalProperties.STEM, and SignalProperties.CONNECT) indicates how the SignalPanel should plot the added signal. HOLD will sample and hold a value until the next value is indicated. CONNECT will plot the data with linear interpolation applied. STEM will simply plot an impulse with the height of the value at each point within the signal.

3) public void clear()

The clear function removes all SignalData objects from the SignalPanel so that it displays nothing.

4) public void paintComponent(Graphics g)

The paintComponent function from JPanel is overridden so that the SignalPanel class can customize its look. For each signal the SignalPanel must plot, the paintComponent method draws the borders and for signals where the SignalData structure indicates to do so, automatically determines grid lines for the time and amplitude axes, then calls drawSignal to draw the actual signal on top of the background.

The scheme for grid lines on the time axis is as follows:

- If the total length of time on the plot is more than a day, place grid lines at the beginning of each day.
- If the total length of time on the plot is more than an hour and less than a day, place grid lines at the beginning of each hour.
- If the total length of time on the plot is more than a minute and less than an hour, place grid lines at the beginning of each minute.

The scheme for the auto grid on the amplitude axis attempts to fill the available height in pixels with the amplitude of the signal while still maintaining a multiple of a power of 10 as the spacing between grid lines.

5) public void drawSignal(Graphics g, SignalData data, SignalProperties sp, boolean allowPoints)

The drawSignal function takes the graphics component that Java passes to the paintComponent method, the individual SignalData object to be plotted, a SignalProperties object to indicate how to draw the signal, and an option to disallow the overlaying of points on top of the line drawn to represent the signal. The drawSignal method will then iterate over the points of the SignalData object and draw the signal as specified by the SignalProperties object.

7.8 SignalProperties.java

The SignalProperties class defines how a SignalData structure should be plotted in a SignalPane. In addition, the SignalProperties class has methods to control the zoom and scroll of the signal it is tied to.

1) double amplitude

The amplitude field is a multiplier applied to the data values in SignalData to convert them from the units they are stored in (e.g. volts, milliliters) to number of pixels to offset on the screen. For example, if one percentage point of SpO2 corresponds to 10 pixels, then the amplitude would be 10

2) double plotZoom

The plotZoom field is a second multiplier that is applied to the time values in SignalData to indicate to the SignalPanel how to scale the time axis in order to zoom the signal properly.

3) boolean autogrid

The autogrid option indicates to the SignalPanel whether or not it should attempt to automatically determine the grid size for this signal. If autogrid is false, the panel will attempt to read the grid size from this SignalProperties object instead of using the scheme described in Section 7.7.

4) double start

The start field indicates the value at which the gridding should start. (the lowest signal data value on the plot)

5) double gridHeight

The gridHeight field indicates what total height the strip of signal should occupy in the units of the signal. For Example, if the hematocrit signal ranged from 28 to 35, the gridHeight could be near 7 to allow for maximum usage of the space.

6) public void importGrid(SignalData sd)

The importGrid function allows the SignalProperties object to read the grid preferences indicated in the SignalData structure instead of using the automatic grid technique described above. The importGrid method will populate the start and gridHeight fields in SignalProperties using statistics calculated by the SignalData object.

There are a series of zoom functions available in the SignalProperties class. Each of these has a specific purpose as will be described below.

7) public void zoom(SignalProperties sp, int width)

One zoom function takes as an argument a SignalProperties structure and forces the endpoints of the receiving SignalProperties object to be the same as the SignalProperties object passed in as an argument. The width argument gives the number of pixels across the SignalPane that this SignalProperties object needs to fill. This is necessary for the proper calculation of the zoom values.

8) private void zoom(long newBase, double plotZoom, int width)

Another zoom function takes as arguments the two fields that represent the state of a zoom in the SignalProperties object. The newBase is a long integer representing the time at the beginning of the plot. The plotZoom argument will set the plotZoom field in the SignalProperties object. This field is described above. The SignalProperties object also needs to know how many pixels wide the plot will be.

9) public void zoom(long newBase, long newEnd, int width)

This zoom function takes the desired start time of the plot, the desired end time of the plot, and the width of the plot in pixels. It then calculates the appropriate zoom constants and stores them in the SignalProperties object.

10) public void recenter(long time, int width)

The recenter function takes a new center time for the plot as the time argument as well as the width of the plot in pixels. The function updates the zoom constants seeking to keep the given time in the middle of the plot.

7.9 SignalData.java

SignalData.java is the Annotation Station's unified interface to all numeric data, including evenly and unevenly sampled data of all sampling frequencies. SignalData is the data structure used when querying data from the postgres database, reading

from parameter files, or reading waveform files. For each of these cases, a different subclass implementing the SignalData interface is used (TrendData for unevenly sampled numeric data, ParamData for parameter data, and WaveData for waveform data), but each of these can be used interchangeably for display and storage. The SignalData interface allows the Annotation Station to hide the detail of how each type of numeric data is stored and retrieved in each of the classes by ensuring commonality among all of the different numeric data structures within the Annotation Station. The common features are outlined and described below:

There are fields of SignalData that identify a time series of numeric data in the Annotation Station.

1) Identifiers

    (1) String name

    Every time series must have a human readable name coming from the MIMIC II database.

    (2) String source

    Every time series must have a source. Sources are unique identifiers of the origin (e.g. table, file) within the MIMIC II database of the values that comprise the time series.

    (3) String uom

    Every time series must have a unit of measurement so that the meaning of the values in the time series is clear.

There are functions of SignalData that allow an accessing class to query the actual numeric values that the SignalData object represents.

2) Value Fields

    (1) int getLength()

    The SignalData object must provide the total number of samples in the time series to the Annotation Station's plotting software.

(2) double getValue(int i)

Value as a function of index - Once the plotting software determines the number of samples in a signal, it may request numeric sample values from the SignalData object using a 0 based index. Valid indices range from 0 to the one less than the length of the signal.

(3) long getTime(int i)

Time as a function of index - Using the same indexing that allows the plotting software to retrieve values, times must be made available by the SignalData object.

(4) int getPointOfTime(long time)

The SignalData object must return the index with a value present nearest to the given time argument.

There are functions in SignalData that answer questions about statistics and properties of the data.

3) Properties of the Data

(1) long getBasetime()

A SignalData object must report the earliest available time point in the signal.

(2) long getTimeWidth()

A SignalData object must report the difference between the latest available time point and the earliest available time point in the signal.

(3) double getMinVal()

A SignalData object must be able to give the minimum available data value.

(4) double getMaxVal()

A SignalData object must be able to give the maximum available data value

(5) double getAmplitudeHeight()

Once a SignalData object implements the methods to give the maximum and minimum, the getAmplitudeHeight method in the abstract class SignalData.java will use those values to give the difference, which is the amplitude height of the data series.

In addition to data access features, there are plotting features required by the SignalData interface that must be offered by all implementing classes.

4) Plotting Features

(1) SignalData[] overlays()

A SignalData object must have the capability to add overlays to any signal data structure. Overlays are SignalData objects to be plotted on top of other SignalData objects. Using the overlays method, the plotting software must be able to extract the SignalData objects to be overlain.

(2) boolean autoGrid()

A SignalData object must return a boolean value to the autoGrid function indicating whether the plotting software should attempt to automatically determine amplitude grid size or to use the SignalData statistics.

7.9.1 TrendData.java

The TrendData class is the Annotation Station's internal data structure for unevenly sampled data queried from tables in the postgres database. Because of the uneven sampling, the time and value must be stored for each measurement of this type of data.

The constructor for TrendData has the following form:

1) public TrendData(String valName, String uomName, int pid, Vector itemid, String source, String tablename, String name)

TrendData is given the name of the database table to select the data from in the tablename argument as well as the column name to query in the valName argument. The tablename and column name will be queried in the MIMIC II database at object instantiation time. Additional necessary information is the name of the column where the unit of measurement is given passed in the uomName argument, the pid of the patient for which the series of measurements is to be extracted, a vector named itemid indicating the itemids that correspond to values to be extracted for this signal. There can be multiple itemids for each signal, as indicated previously. Also, a string uniquely representing the source of this data (source) as well as a string naming the data series (name) are required.

7.9.2 ParamData.java

The ParamData class is the Annotation Station's internal data structure for the 1 minute time averaged data available in the parameter files. A ParamData structure is created with the following constuctor:

1) public ParamData(String name, String source, String uom)

A ParamData structure must be constructed giving the signal a name, a unique source and a unit of measurement (uom). Once constructed, values are added to the signal one at a time using the following method:

2) public void add(long time, float val)

The add method allows addition of values to the ParamData object one data point at a time. For each data point, a time and a floating-point value representing the data or the number –888 representing a missing value are given as arguments. Data points should be supplied to the add method in chronological order, from earliest to latest. The add method will round the time value to the nearest minute and check how many minutes there are between the previous value and the current value. If the difference is one minute, the data point will simply be added. Otherwise, a number of data points

representing missing values will be filled in for the period over which no data is available.

The reason ParamData was constructed in a way that accommodates adding one value at a time is that the structure of the parameter files as discussed in Section 2.2.3.1 stores the parameter data in 30 value chunks at a time. Therefore, it is convenient to iterate over the chunks of the file once and populate all 30 of the ParamData structures at the same time. The file that reads a parameter file, constructs the 20 ParamData objects, and populates them is named ReadParam.java.

7.9.3 WaveData.java

The WaveData class is an internal data structure in the Annotation Station representing the 125Hz sampled Waveform data. A WaveData object is initialized with the following constructor:

1) public WaveData(long basetime, int length, String source, String uom)

A WaveData object is intended to hold information for a region of Waveform data on the order of tens of minutes. The arguments are: a basetime that indicates at what time the WaveData class should start reading the waveform file, a length indicating the number of samples to read from the file, a source indicating the file to read the data from, and a uom giving unit of measurement. The WaveData class then reads the header file to determine the offset to begin reading the data file and what time values to associate with those data values. Since waveform data is stored in contiguous chunks composed of 7500 samples, it is most efficient to store the entire desired length of data to be read in one array, then to build a supporting structure that indicates which values within the array are actually missing data. Therefore, in the internal fields of the WaveData class, there are structures that indicate times between the basetime and the end time of this WaveData object where data is unavailable. For times not included in these, the stored data array contains the values representing the waveform data.

8 Discussions and Conclusions

This thesis has presented a framework for annotating complex biomedical databases with a time ordered set of patient state descriptors causally linked to event or signal annotations to provide evidence to justify the state assessment. A description of the Java suite to facilitate this process has been described. The framework has been devised such that these annotations are machine readable and therefore provide the possibility of developing complex algorithms to replicate many of the steps involved in assessing a patient's welfare and making predictions of significant events, improvement or deterioration in health which may require a change in patient care. To-date, approximately 3500 patient records have been collected and annotation has been at the beta stage for the last six months, with a cycle of improvements that have lead to the current annotation strategy. From this process, a robust and flexible software suite has resulted. Of course, this is the just the first step towards developing the algorithms we require. The following problems and tasks need to be addressed in the immediate future:

- The UMLS is a rich database of medical terms and one may code essentially the same event in a variety of ways. Until the mapping between all such multiplicities is found (and this is probably dependent on how detailed one wishes to be) we must narrow down the range of UMLS codes that can be associated with the subset of events initially being annotated.

- Demonstrate that different parallel annotators can identify significant events in a similar manner, or that if the annotation is done by several experts in series, a significant bias is not introduced into the annotation process.

- A significant number of cases need to be identified by some suitable criteria (for each type of problem we are interested in), and annotated. It is important that these are both representative of a large number of the data patterns that are associated with the particular condition so that new abnormal patters are not missed, or new normal patterns do not trigger false alarms in any algorithm trained on this data.

- Signal quality indices for each data source need to be developed and artifacts need to be identified and removed if possible.

- Automation of the annotation of some of the more obvious (an essential) clinical events (such as large blood pressure drops or heart rate increases) is required in order to remove the tedium from the annotation process and reduce the possibility that an expert may miss a significant event.

- The facility for the annotator to set a specific (machine readable) flag to identify automatically detected events as false or true positives needs to be added to the software.

These developements are outside of the scope of this project, but work is underway within the LCP to address each of these problems.

9 Appendix A: Document Type Definition of Annotation

<!DOCTYPE AnnotationsForPatient [

<!ELEMENT AnnotationsForPatient (PatientID, Annotation*)>

<!ELEMENT PatientID (#PCDATA)>

<!ELEMENT Annotation (SnomedCode, SnomedDesc ,ShortDesc, Time, EstimatedTime?, Dt?, Type, Source, Annotator?, FreeText, OverallState, StateTrajectory, CardiacFunction, VDVolume, PeripheralResistance, StateLinks, SymptomsLinks, LabsLinks, DiagnosesLinks, TreatmentsLinks)>

<!ATTLIST Annotation id ID #REQUIRED>

<!ELEMENT SnomedCode (#PCDATA)>

<!ELEMENT SnomedDesc (#PCDATA)>

<!ELEMENT ShortDesc (#PCDATA)>

<!ELEMENT Time (#PCDATA)>

<!ELEMENT EstimatedTime (#PCDATA)>

<!ELEMENT Dt (#PCDATA)>

<!ELEMENT Type (#PCDATA)>

<!ELEMENT Source (#PCDATA)>

<!ELEMENT Annotator (#PCDATA)>

<!ELEMENT FreeText (#PCDATA)>

<!ELEMENT OverallState (#PCDATA)>

<!ELEMENT StateTrajectory (#PCDATA)>

<!ELEMENT CardiacFunction (#PCDATA)>

<!ELEMENT VDVolume (#PCDATA)>

<!ELEMENT PeripheralResistance (#PCDATA)>

<!ELEMENT StateLinks EMPTY>

<!ATTLIST StateLinks id IDREFS #REQUIRED>

<!ELEMENT SymptomsLinks EMPTY>

<!ATTLIST SymptomsLinks id IDREFS #REQUIRED>

<!ELEMENT SignalsLinks EMPTY>

<!ATTLIST SignalsLinks id IDREFS #REQUIRED>

<!ELEMENT LabsLinks EMPTY>

<!ATTLIST LabsLinks id IDREFS #REQUIRED>
<!ELEMENT DiagnosesLinks EMPTY>
<!ATTLIST DiagnosesLinks id IDREFS #REQUIRED>
<!ELEMENT TreatmentsLinks EMPTY>
<!ATTLIST TreatmentsLinks id IDREFS #REQUIRED>
]>

10 Appendix B: Itemids breakup file

```
SECTION: SIGNALS
618 | resp
113 | cvp
211 | heartrate
51 | artbpsys
51 | artbpdias | value2num
52 | artbpmean
646 | spo2
491 | PAPmean
END

SECTION: VENTILATOR
39 | Airway Size
190 | FiO2 Set
450 | Minute Volume(Obser)
470 | O2 Flow (lpm)
506 | PEEP Set
535 | Peak Insp. Pressure
614 | Resp Rate (Spont)
615 | Resp Rate (Total)
619 | Respiratory Rate Set
682 | Tidal Volume (Obser)
683 | Tidal Volume (Set)
684 | Tidal Volume (Spont)
720 | Ventilator Mode
721 | Ventilator No.
722 | Ventilator Type
834 | SaO2
732 | Waveform-Vent
END

SECTION: CARDIAC
89 | C.O. (fick)
```

116 | Cardiac Index
516 | Pacemaker Type
626 | SVR
662 | Stroke Volume
504 | Wedge Pressure
END

SECTION: BLOODGASES
776 | Arterial Base Excess
777 | Arterial CO2(Calc)
778 | Arterial PaCO2
779 | Arterial PaO2
780 2643 | Arterial pH
18 812 | HCO3
END

SECTION: HEMATOLOGY
1594 | CBC
813 | Hematocrit
814 | Hemoglobin
828 | Platelets
815 | INR
824 825 863 864 2088 | PTT
861 872 1990 2043 2644 2645 2733 | WBC
833 1992 2734 | RBC
676 677 | Temperature C
678 679 | Temperature F
END

SECTION: URINE
2062 2840 4405 | Urine pH
1026 2031 4278 | Urine Spec Gravity
2033 | Urine Nitrite
2034 | Urine Glucose
1095 2064 | Urine Ketones
2035 | Urine Bili
2036 | Urine RBCs

```
END

SECTION: RENAL
20 837 1785 2072 2093 2117 2647 2736 | Sodium
19 829 1740 2007 2118 2648 2673 2737 4593 | Potassium
788 1621 2073 2094 2738 | Chloride
787 2675 | Carbon Dioxide
811 1379 2052 2090 2135 | Glucose
781 2091 2676 | BUN
17 791 2092 2649 4594 | Creatinine
2096 | Anion Gap
END

SECTION: BIOCHEM
772 1444 2104 2346 4873 | Albumin
849 2018 2051 2103 4586 | Total Protein
786 816 2097 4583 | Calcium
821 4342 4585 5101 | Magnesium
769 866 | ALT
770 867 | AST
773 870 | Alk. Phosphate
848 2120 | Total Bili
803 4589 | Direct Bili
2102 | Indirect Bili
775 4293 4523 | Amylase
784 871 | CPK
785 | CPK/MB
851 3341 3923 | Troponin
817 | LDH
820 1504 | Lipase
853 4590 | Uric Acid
789 2099 4584 | Cholesterol
850 2100 4588 | Triglyceride
818 4587 5124 | Lactate
END

SECTION:TOTALIO
```

```
18 | Hourly IV | pervolume
29 | Net Hourly Balance | pervolume
1 | Daily In | cumvolume
2 | Daily Out | cumvolume
1 | Hourly In | pervolume
2 | Hourly Out | pervolume

END
```

11 Appendix C: Meanings of State Assessments

11.1 Overall state

0: Baseline

-1: Early pathophysiology or well-compensated, could very well be tolerated long-term

-2: Probably tolerable for over one hour but very dangerous if on-going for several hours

-3: Imminently fatal without major life-saving interventions

-4: Physiologic collapse (e.g. pulseless VT)

11.2 State trajectory

-1: Getting worse

0: No change

1: Improvement

11.3 Cardiac function / Vascular distending volume / Peripheral resistance

-1: Worse than normal (for the population at large, NOT just for the patient)

0: Normal

1: Superphysiologic

12 Literature Cited

[1] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation 2000 (June 13);101(23):e215--e220. Circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215.

[2] http://www.physionet.org/.

[3] Saeed M, Lieu C, Raber G, Mark R. MIMIC II: A massive temporal ICU patient database to support research in intelligent patient monitoring. Computers in Cardiology 2002; 29:641--644.

[4] Mark RG. Integrating data, models and reasoning in critical care, 2003. National Institute of Biomedical Imaging and Bioengineering Proposal R01 EB001659.

[5] Dawant B, Uckun S, Manders E, Lindstrom D. The simon project: Model-based signal analysis and interpretation in intelligent patient monitoring. IEEE Engineering in Medicine and Biology 1993;12(4):82--91.

[6] Korhonen I, Ojaniemi J, Niieminen K, Van Gils M, Heikela A, Kari A. Building the improve data library. IEEE Engineering in Medicine and Biology Magazine Nov/Dec 1997; 16(6):25--32.

[7] UMLS knowledge sources, 16th edition - July release: 2004ab documentation http://www.nlm.nih.gov/research/umls/umlsdoc.html.

[8] http://www.medical.philips.com/main/products/patient_monitoring/products/carevue/.

[9] Douglass M, Clifford G, Reisner A, Moody G, Mark R. Computer-assisted deidentification of free text in the MIMIC II database. Computers in Cardiology 2004;M6.2.

[10] Shu J, Clifford G, Saeed M, Long W, Moody G, Szolovits P, Mark R. An open-source, interactive java-based system for rapid encoding of significant events in the icu using the unified medical language system. Computers in Cardiology 2004;S41.6.

[11] Moody GB, Mark RG. A Database to Support Development and Evaluation of Intelligent Intensive Care Monitoring. Computers in Cardiology 1996; 657-660.

[12] Balogh D, Kittinger E, Benzer A, Hackl JM. Noise in the ICU. Intensive Care Med. 1993;19(6):343-6.

[13] Kannel WB. The demographics of claudication and the aging of the American population. Vasc Med. 1996;1(1):60-4.

[14] Garder R, Shabot M. Computer Applications in Health Care, Chapter 13, pages 443--484.

[15] Wang K, Kohane I, Bradshaw K, Fackler J. A real time patient monitoring system on the World Wide Web. In Proc. AMIA Annu. Fall Symp., pages 729--732, 1996.

[16] Lawless S. Crying wolf: False alarms in a pediatric ICU. Critical Care Medicine, 22:981-984, 1994.

[17]. Le Gall JR, Lemeshow S, Saulnier F. A new Simplified Acute Physiology Score (SAPS II) based on a European/North American multicenter study. JAMA Vol. 270 No. 24, December 22, 1993.

[18] Heldt T, Shim EB, Kamm RD, Mark RG. Computational modeling of cardiovascular response to orthostatic stress. J. Appl. Physiol., 92:1239-1254, 2002.

[19] Strauss MJ, LoGerfo JP, Yeltatzie JA, Temkin N, Hudson LD. JAMA. Chicago: Mar 7, 1986.Vol.255, Iss. 9; pg. 1143

[20] Moody GB, Mark RG. The MIT--BIH arrhythmia database on CD-ROM and software for use with it. Computers in Cardiology, 17:185-188, 1990.

[21] http://www.physionet.org/physiotools/wfdb/lib/ecgcodes.h .

[22] Zong W, Moody G, Mark R. Reduction of false positive blood pressure alarams by use of electrocardiogram blood pressure relationships. Computers in Cardiology, 26:305--308, 1999.

[23] http://www.physionet.org/physiotools/wug/ .

[24] Bray T, Paoli J, Sperberg-McQueen CM, Maler E. Extensible Markup Language (XML) 1.0. W3C Recommendation World Wide Web Consortium, Feb, 1998.

[25] http://www.xml-schema.com/tools/oracle/xdk/ .