An Object-Oriented Framework for Predictive Models in Intensive Care Units

Moyano M., Camaña R., Cechich A.

Department of Informatics and Statistics - University of Comahue
Buenos Aires 1400, 8300 Neuquén, Argentina
E-mail: acechich@uncoma.edu.ar

Abstract

When used in conjunction with patterns, class libraries, and components, object-oriented application frameworks can significantly increase software quality and reduce development effort. Frameworks are a kind of domain-specific model whose structure can reuse existing patterns. In the field of medical applications, one of the important trends is the move towards frameworks describing different situations. Frameworks in medicine entails capturing, storing, retrieving, transmitting and manipulating patient-specific health care related data, including clinical, administrative, and biographical data. Using predictive methods in Intensive Care Units is a standard procedure to determine a measure of disease severity, based on current physiologic measurements, age and previous health condition. These situations can be described by reusing existing models and patterns, and building new structures based on flexible issues.

In this paper, we present a Java object-oriented framework developed for modelling predictive methods in Intensive Care Units. We also briefly discuss future work, which will include a formal specification as part of the framework's documentation.

Key words: Object-Oriented Design Patterns – Analysis Patterns – Object-Oriented Frameworks – Health Care Information Systems

1. Introduction

Software component technology has emerged as a key element for developing complex systems, where maintainability, integrability, and other quality characteristics are increasingly important. Software reuse has been one of the main goals for decades. However, reusing software is not an easy task and most successful efforts are reduced to reuse small software components. Object-oriented paradigm has provided with a way of reusing more complex and larger components. In this context, frameworks [1] attract researchers and developers since among its advantages reuse improving and less time to market are found.

A framework can be defined as “a set of cooperating classes making a reusable design for a specific kind of software”. Frameworks supply an architectural guide by partitioning the system into abstract classes, and its responsibilities and collaborations. A developer customises a framework by using subclassing and composition.

A framework has a physical representation in terms of classes, methods, and objects, which is adapted by customising and extending the provided structure. The parts of a framework that are suitable to be extended and customised are called “hot spots” [2]. These spots represent domain features that cannot be completely anticipated. The task of discovering hot spots involves domain analysis and an active collaboration between developers and domain experts.

Patterns are deeply related with frameworks. However, patterns are more abstract in essence and it document ideas that can be applied on different contexts. In this sense, patterns are smaller and less complex than frameworks [3][4].

As a first work towards building a framework for predictive models in Intensive Care Units (ICU’s), we have analysed different models – APACHE II, SAPS II [5][6] – and we have produced a model based on patterns. We have also identified hot spots building a structure based on abstract classes [7][8]. However, a complete framework is more than a model: it also includes guidance and implementation.

In this paper we present an extension of our previous model of predictive methods in ICU’s to implement it as a Java framework. We have focused on preserving flexibility by using abstract classes, subclassing and composition. Analysis and design patterns are used, as well as other useful structures in ICU’s domain. In Section 2 of the paper we introduce some predictive methods used in ICU’s and we describe one particular method in some detail. Section 3 overviews our model focusing on reused patterns and created structures. Section 4 presents the Java framework. Future work will extend the documentation of the framework to include a formal specification of some structures based on a formal model of patterns [9][10][11]. This is discussed briefly in the final section of this paper.

2. Predictive Methods in Intensive Care Units

The information recorded by doctors is the main element used to assure a continue attendance to the patients in an ICU. An efficient record must include multiple types of data capable of grouping a variety of information, which is used by physicians to perform their evaluations. Supplying the necessary elements to facilitate the measurement of every relevant variable is the most important issue. But sometimes, the meaning of a “relevant variable” depends on the interpretation of the doctor who is performing the analysis. As a result, different evaluations of the same patient can be done. In order to standardise the procedure for evaluating the level of risk associated to a patient, who is arriving to an ICU, several predictive models have been developed. For example, APACHE II – a predictive model – has been used to evaluate the quality of the attention in an ICU, or to estimate the mortality of a group of patients.

Our analysis of the Intensive Care Unit domain started with the analysis of different mechanisms used to establish some state variables (predictors) which indicate the state of critical incoming patients. The variables, also called “score”, are measured when a patient arrives to an ICU in order to quantify his risk level, and to make medical attentions more efficient. Of course, there are also periodically calculated variables that are used for analysing the evolution of a patient, but these variables aren’t taken into account in our present work. A process based on the parameterisation of different state variables – temperature, blood pressure, etc. – is used to calculate the predictor. An algorithm takes these variables as input and calculates the value of the predictor. This value must be compared with patterns whose meaning should be known.

The methods APACHE II and SAPS II have been chosen in our analysis because of the possibility of applying them to many situations. Other methods, such as TISS (Therapeutic Intervention Scoring System), MPM (Mortality Probability Models), or The SUPPORT Prognostic Model, could be used in a similar analysis and modelled in the same way.

To clarify the notion of predictive models used in an ICU, a summary of the method APACHE II is included. APACHE II was defined based on 13 variables and the valuation of the previous state of the patient. The APACHE II score is a general measure of disease severity, based on current physiologic measurements, age and previous health condition. Scores range from 0-71, with an increasing score associated with an increasing risk of death. The score can help in the assessment of patients to determine the level and degree of diagnostic and therapeutic intervention. The algorithm used to calculate the score is based on the following expression:

APACHE II score = (acute physiology score) + (age points) + (chronic health points)

	+4	+3	+2	+1	0	+1	+2	+3	+4
Rectal temp in C°	>= 41	39-40.9		38.5-38.9	36-38.4	34-35.9	32-33.9	30-31.9	<= 29.9
Mean arterial pressure mm Hg	>= 160	130-159	110-129		70-109		50-69		<= 49
Heart rate in beats/min.	>= 180	140-179	110-139		70-109		55-69	40-54	<= 39
Respiratory rate in breaths/min	>=50	35-49		25-34	12-24	10-11	6-9		<= 5
Oxygen: A-aDO2 (FIO2 >= 0.5)	>= 500	350-499	200-349		< 200				
Oxygen: PO2 (FIO2 < 0.5)					> 70	61-70		55-60	< 55
Arterial pH	>= 7.7	7.6-7.69		7.5-7.59	7.33-7.49		7.25-7.32	7.15-7.24	< 7.15
Serum sodium	>= 180	160-179	155-159	150-154	130-149		120-129	111-119	<= 110
Serum potassium	>= 7	6-6.9		5.5-5.9	3.5-5.4	3-3.4	2.5-2.9		<2.5
Serum creatinine in mg/dL	>= 3.5	2-3.4	1.5-1.9		0.6-1.4		< 0.6		
Hematocrit in percent	>= 60		50-59.9	46-49.9	30-45.9		20-29.9		< 20
WBC in thousands	>= 40		20-39.9	15-19.9	3-14.9		1-2.9		< 1

Table 1. Acute Physiology Score.

Age	**Points**
<= 44	0
45-54	2
55-64	3
65-74	5
>= 75	6

Table 2. Age points.

History of severe organ insufficiency or immunocompromised	**Points**
Nonoperative patients	5
Emergency postoperative patients	5
Elective postoperative patients	2

Table 3. Chronic Health Points

Table 1, Table 2, and Table 3 are used to obtain the acute physiology score, the age points, and the chronic health points respectively. In Table 1, the score for serum creatinine is doubled if the patient has acute renal failure and the mean arterial pressure is calculated as ((systolic blood pressure) + (2 * (diastolic pressure))) / 2. In Table 3, the organ insufficiency or immunocompromised state must have preceded the current admission. Immunocompromised means that: (1) the patient is receiving therapy reducing host defenses (immunosuppression, chemotherapy, radiation therapy, long term steroid use, high dose steroid therapy), or (2) the patient has a disease severe enough to interfere with immune function such as malignant lymphoma, leukemia or AIDS. Liver insufficiency means that one or more of the following situations arises: (1) biopsy proven cirrhosis, (2) portal hypertension, (3) episodes of upper GI bleeding due to portal hypertension, (4) prior episodes of hepatic failure, coma or encephalopathy. A cardiovascular insufficiency means New York Heart Association Class IV. A respiratory insufficiency means (1) severe exercise restriction due to chronic restrictive, obstructive or vascular disease, (2) documented chronic hypoxia, hypercapnia, secondary polycythemia, severe pulmonary hypertension, or (3) respirator dependency. Finally, a renal insufficiency is produced when the patient is on chronic dialysis.

Several software tools implementing predictive models can be found. The software UTICalculos [12] and the expert system prototype for evaluation and advising of critical patients [13], are some examples.

3. A Flexible Model for Predictive Methods

Information, which represents a patient's record, is summarised by the following components [8], as shown in Figure 1:

- Observation component: represent the set of data obtained from test, signs, background, etc. Some data are fixed but others vary depending on therapeutics a patient is receiving. In general, a patient must go into an intensive care unit because some of these values risk its life. The main goal of an ICU is to monitor the evolution of these values until a stable and normal condition is reached. Predictive models are used to quantify risks when a patient arrives to an ICU.
- Medical conclusion: according to observations, doctors can determine a diagnosis when information is secure enough or they can deduce a prognosis by analysing a patient's evolution.
- Therapeutics: doctors define actions and therapeutics to normalise values measured for a patient.

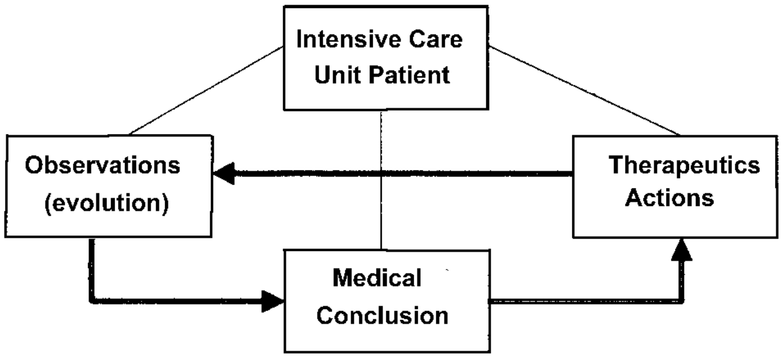

Figure 1: Model for Recording Patients.

3.1 Patterns for Medical Domains

The following description concentrates on the static elements of an observation as is presented in [14]: what an observation or measurement is and how we can record it in a generic way to support the analysis that clinicians need to perform on it.

Modelling quantities as attributes may be useful for a single hospital department that collects a couple of dozen measurements for each in-patient visit. However, thousands of potential measurements can be made on one person. One solution is to consider all the various things that can be measured (height, weight, blood glucose level, and so on) as objects and to introduce the object type phenomenon type. A person would then have many measurements, each assigning a quantity to a specific phenomenon type. The person would have only one attribute for all measurements, and the complexity of dealing with the measurements would be shifted to querying thousands of instances of measurement and phenomenon type.

Just as there are many quantitative statements made about a patient, there are also many important qualitative statements, such as gender, blood group, etc. To record a person's gender, which has two possible values – male and female – a new type, category observation, can be devised. It is similar to a measurement but has a category instead of quantity, so a new type of observation acts as a supertype to a measurement and a qualitative observation. In this model, gender is an instance of phenomenon type, and male and female are instances of category.

But certain categories can be used only for certain phenomenon types. For example, "tall", "short", or "average" might be categories for the phenomenon type height, while "A", "B", "A/B", and "O" might be categories for the phenomenon type blood group. So, a phenomena class defines the possible values for some phenomenon type (see Figure 1). For example, the fact that a person is blood group A is indicated by a category observation of a person whose phenomenon is blood group A. The blood group A phenomenon is linked to the phenomenon type of blood group.

Many observations involve merely a statement of absence or presence rather than a range of values. The model shown in Figure 2 allows any category observation to have presence or absence. Observation Concept is a supertype of phenomenon to allow observation concepts without attaching them to some phenomenon type. For example, the fact that a person has diabetes is recorded by a presence observation of the person linked to the observation diabetes.

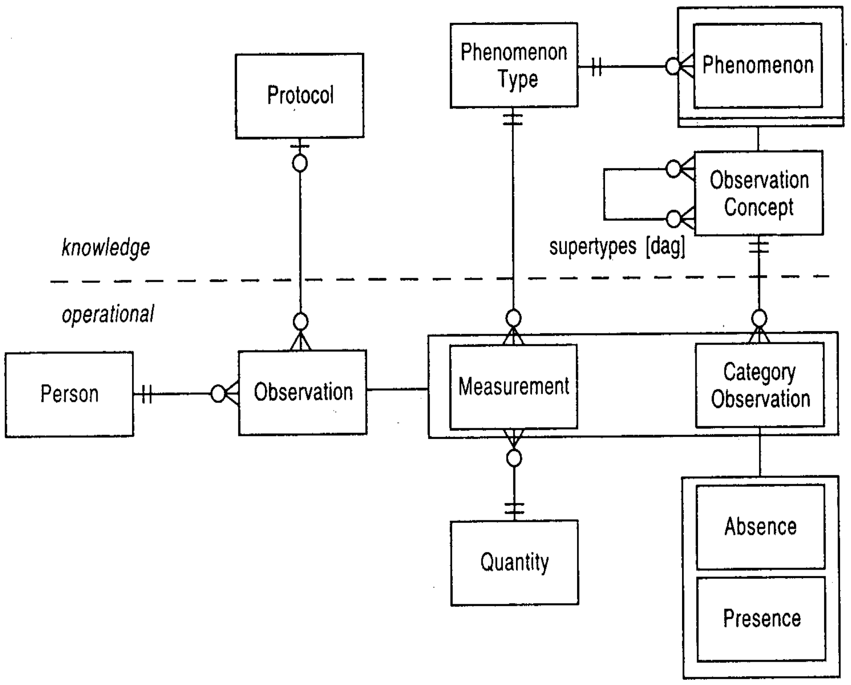

Figure 2: Analysis Pattern for Observations

3.2 Flexible Structures

The following description introduces some modelling artefacts we have built by reusing existing patterns and structures or by creating new elements based on the notion of "hot spots" given by Pree [15].

Our abstraction of observations is quite different from the concepts introduced in Section 3.1. The state of a patient in an ICU is monitored by measuring different variables – temperature, blood pressure, cardiac frequency, etc. – which conforms an observation. So, an observation represents the variables measured for a patient instead of a general measure. However, our observations are also classified building a hierarchy. This hierarchy represents that an observation is composed of the more recently observation and a set of all previous observations of a patient. A temporal classification is used to recursively specify the complete record of a patient in which two adapted Composite [16] patterns are used [7].

Predicting the evolution of a patient based on the variables measured during the observations, is one of the more common applications. The methods mentioned in Section 2 – APACHE II and SAPS II – are used to predict the evolution of a patient and by analysing them, many common features can be detected. A general model of predictive methods can be abstracted by reusing an Observer [16]: every time the status of a patient – modelled as a set of measured variables – changes, observer objects modify their states depending on the changed variables. In that way, the prediction given by a particular test remains consistent with the status of a patient [7], as Figure 3 shows.

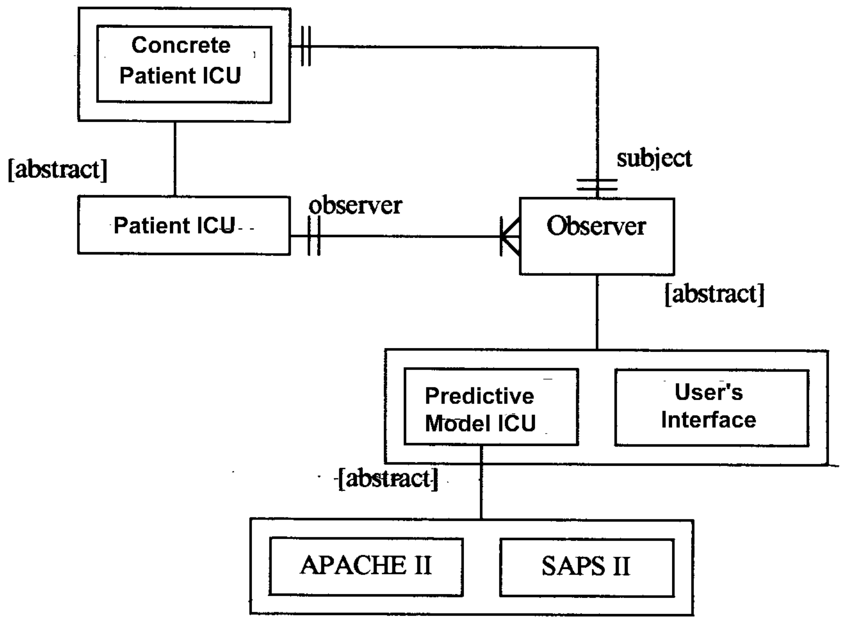

Figure 3: Structure for Predictive Models in ICU's

In similar way, other structures have been developed [7][8]. Some of the classes in the model depends on the implementation context and should be flexible enough for allowing customisation and adaptation. For example, subclassing "Predictive Model ICU" in Figure 3, makes extension and modification easier. Several other flexible characteristics, not discussed here for brevity, have been underlined in our model

4. Implementation: A Java Framework

In this Section, we present an example of some behavioural features in our Java [17] implementation. A sequence of actions and messages performed as a consequence of recording a new observation is first introduced. In this case, the last temperature measured for a patient is recorded, and a calculation of the predictor APACHE II is performed. Figure 4 shows the corresponding sequence diagram.

As a way of dealing with complexity, the framework was divided into clearly identifiable components. Functionality was split and a more understandable model was created. For example, there is a component for dealing with monitoring of observations, another component for monitoring treatments, another for monitoring medical conclusions, and another for dealing with predictors. The following example is part of the component for predictors.

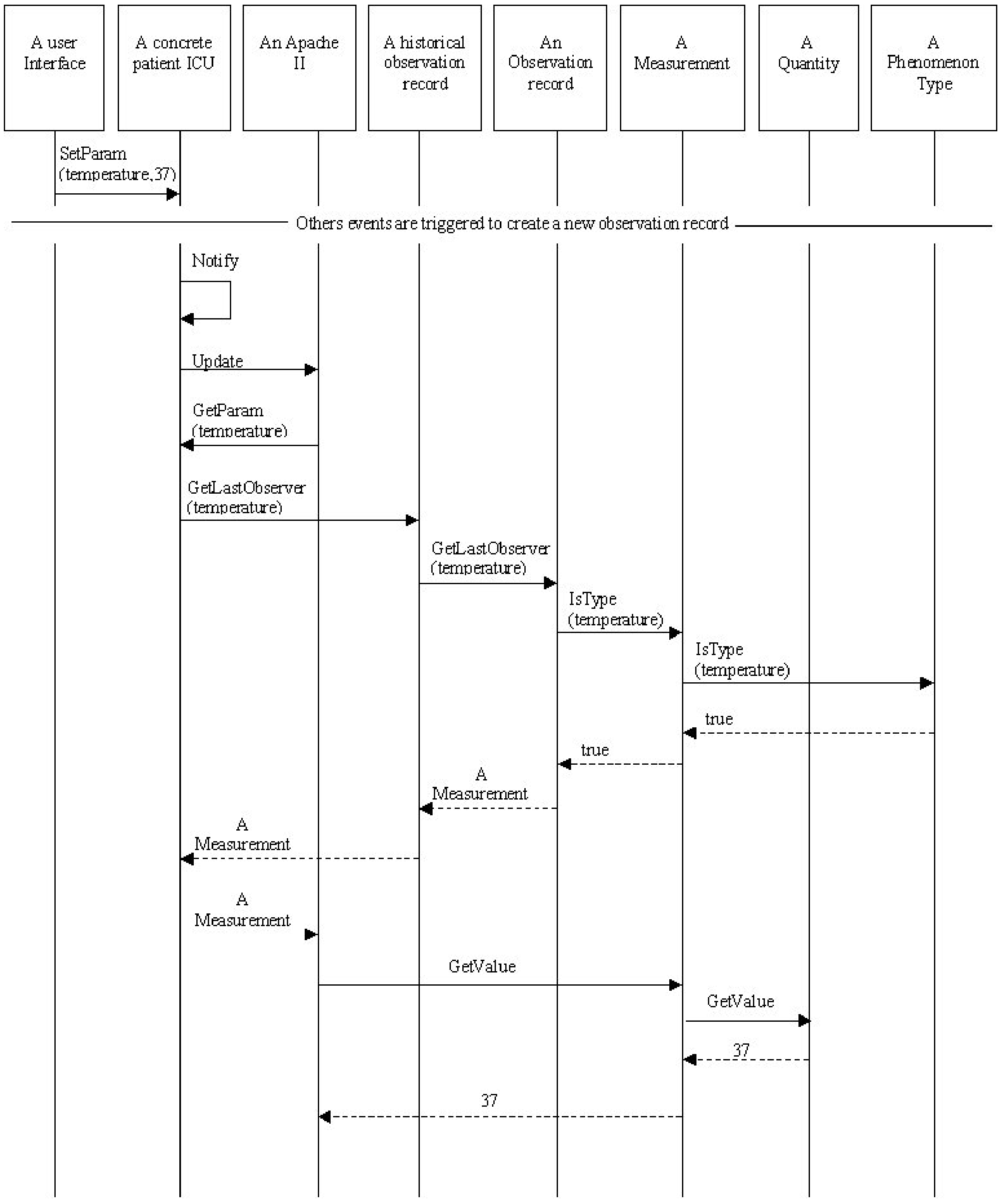

Figure 4: Sequence diagram for updating a predictor.

Implementation is presented with hot classes, that are classes customised or extended in a particular application, written in Italics. The class *Enumeric_Type* was created for classifying all sort of observations (temperature, blood pressure, etc.). Every subclass represents a possible kind of observation.

The following pieces of code show a partial Java implementation:

```
abstract classs Patient_icu
{   ...
   private void notify ( )
  {    int i = 0;
    while ( i < Observers .count)
      {    (Observers .getObject (i )).update ;
         inc ( i );       }   }
  ...
  private list_of_observers Observers;
}
```

```
class Concrete_Patient_icu extends
Patient_icu
{   ...
   public boolean getparam (Enumeric_type
et;  Observation obs)
  {
    return O_monitor.getLastObservation
(et,Obs);
   }
  private Observation_monitor O_monitor;
 ... }
  abstract class Observer
{  ...
  private Conrete_Patient_icu Patient
  ... }
```

```
abstract class Predictive_Model_icu extends
Observer
{  ...
  abstract public  void update ( );
  private int test_value;
  ...}
abstract class Observation_Monitor
{  ...
  abstract public boolean
getLastObstervation (EnumericTyoe et;
Observation obs)
 ...}
```

```
class apacheII extends Predictive_Model_icu
{   ...
  public void update ( )
  {     test_value=0;
    float temperature_value ;
    Observation temperature_object;
    Teperature_enumeric_Type ett= new
Temperature_enumeric_Type;

  if (patient.getParam
(ett,temperature_object) )
    {       temperature_value =
temperature_object.getValue;

      if  (temperature_value >=41)
         test_value = test_value + 4
     else
          if  (temperature_value >=39 )
             test_value = test_value + 3
         else
           if  ((temperature_value >=38.5 ) ||
((temperature_value <36 ) &&
(temperature_value<=34)))
               test_value = test_value + 1
           else
               if  (temperature_value >= 32)
                  test_value = test_value + 2
               else
                    if (temperature_value >=30)
                      test_value = test_ value + 3
                   else
                    if (temperature_value < 30)
                      test_ value = test_value + 4 ;
     }   ...
    return test_value  } ...
}
```

```
class Temperature_Enumeric_Type extends
Enumeric_Type
{   boolean Compare (Enumeric_Type et)
  {      return  (et instanceof
Temperature_Enumeric_type)   } }
```

```
class Observation_record extends
Observation_Monitor
{  ...
  public boolean getLAstObservation
(EnumericType et; Observation obs)
 {   int i =0;
      while ( i < obslist.Count ) && ( ! (obslist
.GetObject (i ) ).IsType (et) ))
       inc ( i );

 if   (i == obslist.Count)
     { Obs=obslist .GetObject (i )
       return True
 else
      return False ;  }
  ...

 private List_of_observations  obslist; }
```

```
class Historical_Observation_Record extends
Observation_Monitor
{   ...
   public boolean getLastObservation
(EnumericType et; observation Obs )
   {   if ( Actual_Record.getLastObservation
(et,Obs))
         return True
   else
         return
Historic_Record.getLastObservation (et,Obs);
   }
   private Observation_Record
Actual_Record;
   private Observation_Monitor
Historic_Record;
   ...}
```

```
class Phenomenon_Type
{  public boolean isType (Enumeric_Type et)
  {   return description.Compare(et);
   }
  private enumeric_Type description;
  ... }
```

```
abstract class Observation
{  ...  abstract public boolean isType
(Enumeric_Type et);
  abstract  public float getValue ( ) ;
  ... }
class Measurement extends Observation
{  ...
  public boolean isType (Enumeric_Type et)
  {   return ph_Type.isType (et );  }

  public float getValue ( )
  {   return  q.getValue ( );  }

  private Phenomenon_Type ph_Type;
  private Quantity q;
   ... }
```

```
class Quantity
{ ...
   public float getValue ( );
    {      return value;     }
   private float value;
  ... }

class list_of_observations
{   ...
   public Observation getObject (int index)
    {       if (index < upperBound)
           if (index < lastAssigned)
               return data[index]
           else
                return null
      else
           return null   }

public int Count ( )
  {     return lastAssigned;   }

 private int upperBound=50;
 private int lastAssigned;
 private Observation [ ] data = new
observation [upperBound];
... }
```

```
abstract class Enumeric_Type
{   abstract boolean Compare
(Enumeric_Type et); }
```

```
class list_of_observers
{   ...
  public Observer getObject (int index)
   {      if (index < upperBound)
        if (index < lastAssigned)
            return data[index]
        else
             return null
    else
         return null   }

public int Count ( )
  {     return lastAssigned;   }

 private int upperBound=50;
 private int lastAssigned;
 private Observer[ ] data = new observation
[upperBound];
... }
```

Conclusions and Future Work

In this paper, we have presented an object-oriented framework for predictive models used in Intensive Care Units. The framework was developed focusing on flexible characteristics to be extended or adapted, and a partial Java implementation was presented.

However, a complete framework also includes guidance for using and adapting it. Framework documentation is considered as relevant as the process used for building it. In the next stage of our work, we will add several helpful features to our actual description of the framework, such as application examples instantiated on particular domains. A more unambiguous description could also be of interest considering previous works on this area [9][10][11], so formalisation of some parts of the framework will be considered.

Acknowledgements

The authors wish to thank personnel of the *Castro Rendom* Hospital, and personnel of the *Pasteur* Clinic (Neuquén City), especially Dr. Urdapilleta, Dr. Calvo, Dr. Shutto, Dra. Morán, and Dr. Moyano for their invaluable collaboration.

References

[1] Fayad M., Schmidt D., *Object-Oriented Application Frameworks,* Communications of the ACM, Vol. 40, No. 10, October 1997

[2] Fayad M., Schmidt D., Johnson R., *Building Application Frameworks*, John Wiley & Sons, 1998.

[3] Johnson R., *Documenting Frameworks using Patterns*. Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings, 1992 (63-76)

[4] Johnson R., *Frameworks = (Components + Patterns)*. Communications of the ACM, Vol. 40, No. 10., October 1997 (39-42)

[5] Pusajó J.F. , Doglio G.R. , Hernandez M.S., Salvador M.C. , Bonfigli G.C. *Valoración Del Paciente en Estado Crítico,* Hernandez Editoriales, 1989

[6] Svirbely John and M.G. Sriram, *The Medical Algorithms Project* – Chapter 30, 1999 http://www.medal.org/ch30html

[7] Luzuriaga J., Martínez R., Moyano M., Braicovich G., Camaña R., Cechich A., *Object-Oriented Patterns Applied to Predictive Models in Intensive Care Units*, 3rd Argentine Symposium on Health Care Informatics, 29 JAIIO, 2000 (116-125)

[8] Luzuriaga J., Martínez R., Moyano M., Camaña R., Cechich A., *Un Modelo basado en Reuso para Sistemas de Pronóstico en Unidades de Terapia Intensiva*, 4th Argentine Symposium on Health Care Informatics, 30 JAIIO, September 10-14, 2001(to be published)

[9] Cechich A., *A Formal Model for Semantic Statements of Analysis Patterns*, IV Workshop Iberoamericano de Ingeniería DE Requisitos y Ambientes de Software, San José, Costa Rica, 3-6 April 2001 (22-30)

[10] Buccella A. and Cechich A., *A Formal Model for Some Behavioural Features of Analysis Patterns*, 6° Congreso Argentino en Ciencias de la Computación, CACIC'2000, Usuhaia, 2-7 October 2000 (123-132)

[11] Cechich A. and Moore R., *A Formal Basis for Object-Oriented Patterns*, APSEC'99 – 6th Asia-Pacific Software Engineering Conference, Takamatsu, Japan, Dec. 1999 (284-291)

[12] Zarazaga A. , Prast A. , Castell J.T. , Valderrábano S. , Rodríguez Montes J.A., *Metodología, desarrollo y validación de un Sistema Asesor Informatizado para la toma de decisiones clínicas. Predicción del Riesgo y de la Supervivencia, Calidad de vida. Análisis comparativo de la utilidad y costes de los procedimientos terapéuticos en la clínica*, Hosptital Universitario La Paz - Universidad Autónoma Madrid, 1997, http://www.servitel.es/infosalud97/10/10.htm

[13] Campos Eneida Rached, *Un Sistema Interactivo de Cálculo para Terapia Intensiva*, INFORMÉDICA, 1993, http://www.epub.org.br/informe/uticalc.html

[14] Martin-Fowler, *Analysis Patterns - Reusable Object Models*. Addison-Wesley, 1997

[15] Pree W., *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995

[16] Gamma E., Helm R., Johnson R., and Vlissides J., *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley 1995

[17] http://java.sun.com/products