WEATHER FORECASTING SYSTEM
(WFS)

PRITAM KUMAR CHOUDHARY- 021207
AMIT KUMAR DWIVEDI- 021217
GAURAV PRAKASH- 021407

Submitted in partial fulfillment for requirement of the degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT
MAY-2006

CERTIFICATE

This is to certify that the work entitled, "Weather Forecasting System" submitted by Pritam Kumar Choudhary, Amit Kumar Dwivedi and Gaurav Prakash in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering / Information Technology of Jaypee University of Information Technology has been carried out under my supervision .This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Satish Chandra

Mr. Satish Chandra
(Senior Lecturer)
Department of Computer Science
Jaypee University of Information Technology
Waknaghat, Solan

Acknowledgement

Many people have contributed to this project in a variety of ways over the past few months. We thank our project instructor Mr. Satish Chandra, without his guidance, this project would have not been successful. We specially thank Dr. Naveen Prakash for his valuable suggestions. We also acknowledge the helpful comments received from various teachers of different departments. We are indebted to all those who provided reviews and suggestions for improving the software, and we extend our apologies to anyone we may have failed to mention.

Thank You

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

WFS	Weather Forecasting System
SWI	Social Science Informatics
SWIP	Swi-Prolog
AI	Artificial Intelligence
HTML	Hyper Text Markup Language
XML	Extended Markup Language
HTTP	Hyper Text Transfer Protocol
UI	User Interface
GUI	Graphical User Interface
API	Application Protocol Interface

LIST OF FIGURES

ABSTRACT

WEATHER FORECASTING SYSTEM (WFS) forecasts the weather of a particular place on the basis of historical data about its climatic conditions viz. maximum temperature, minimum temperature, sunrise time, sunset time, maximum humidity, minimum humidity, rainfall and snowfall. The cities selected are Shimla, Chandigarh, New Delhi, Kolkata.

Forecasting services can be used in agricultural research, economy performance, risk management and sporting event.

This forecasting system is based on the following field of study:-

I. ARTIFICIAL INTELLIGENCE:

Artificial Intelligence is concerned with the study and creation of systems that exhibit some form of intelligence system that learn new concepts and tasks, systems that can reason and draw useful conclusions about the world around us, systems that can understand a natural language or perceive and comprehend a visual scene, and systems that perform other types of feats that require human types of intelligence.

II. MACHINE LEARNING:

Machine Learning is an area of Artificial Intelligence concerned with the development of techniques which allow computers to learn. More specifically, Machine Learning is a method for creating computer programs by the analysis of data sets. Machine learning overlaps heavily with statistics, since both fields study the analysis of data, but unlike statistics, Machine Learning is concerned with the algorithmic complexity of computational implementations. There are various types of learning mechanism such as decision trees, rule-based systems, neural networks, nearest-neighbor, Bayesian methods etc.

The learning mechanism for WFS is rule-based (Figure 1). The rule-based system starts with a rule-base, which contains all of the appropriate knowledge encoded into If-Then

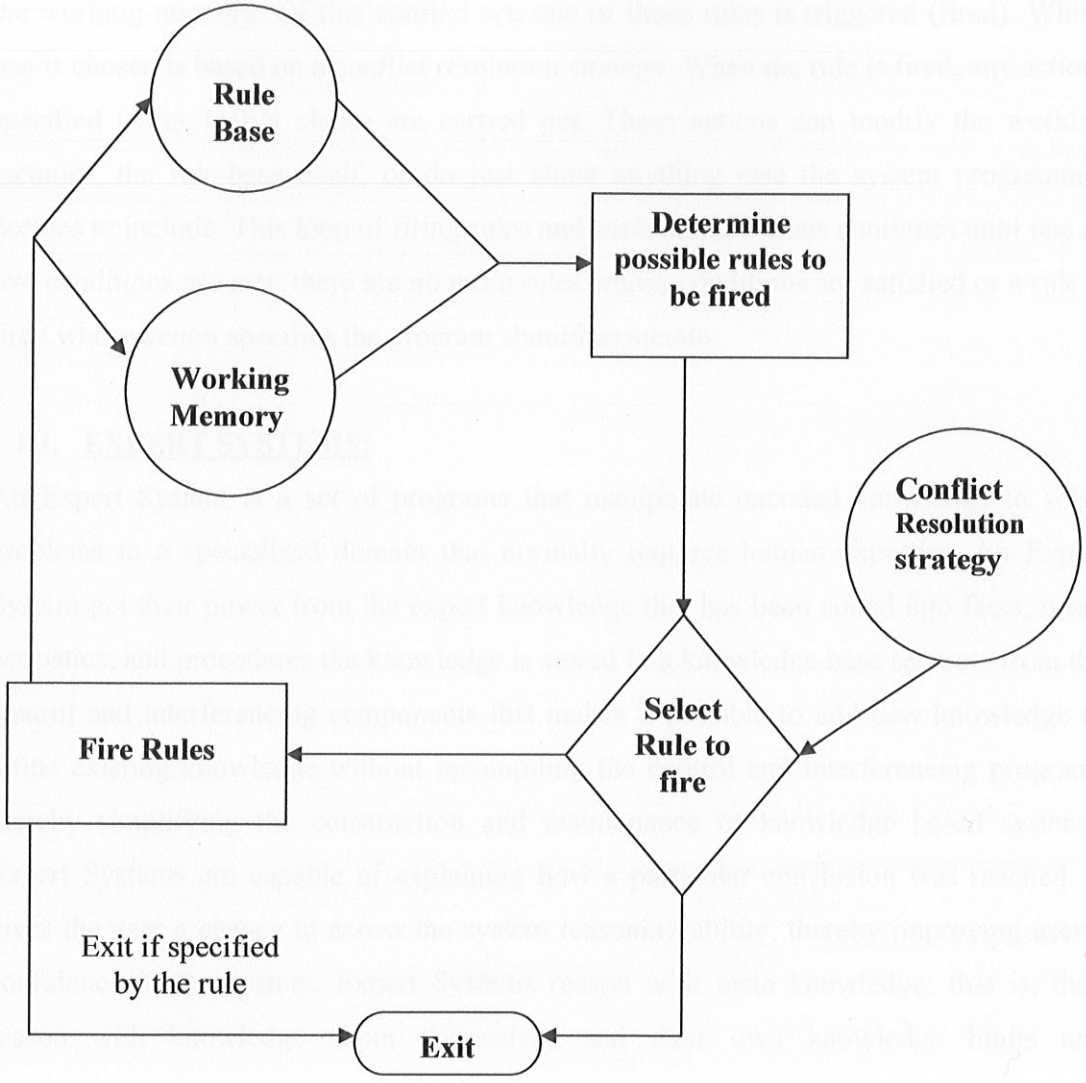

Figure 1 Flow diagram of rule based system

rules, and a working memory, which may or may not initially contain any data, assertions or initially known information. The system examines all the rule conditions (IF) and determines a subset, the conflict set, of the rules whose conditions are satisfied based on the working memory. Of this conflict set, one of those rules is triggered (fired). Which one is chosen is based on a conflict resolution strategy. When the rule is fired, any actions specified in its THEN clause are carried out. These actions can modify the working memory, the rule-base itself, or do just about anything else the system programmer decides to include. This loop of firing rules and performing actions continues until one of two conditions are met: there are no more rules whose conditions are satisfied or a rule is fired whose action specifies the program should terminate.

III. EXPERT SYSTEMS:

An Expert System is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An Expert System get their power from the expert knowledge that has been coded into facts, rules, heuristics, and procedures the knowledge is stored in a knowledge base separate from the control and interferencing components this makes it possible to add new knowledge or refine existing knowledge without recompiling the control and interferencing programs thereby simplifying the construction and maintenance of knowledge based systems. Expert Systems are capable of explaining how a particular conclusion was reached. It gives the user a chance to assess the system reasoning ability, thereby improving user's confidence in the system. Expert Systems reason with meta knowledge; that is, they reason with knowledge about themselves and their own knowledge limits and capabilities.

1. INTRODUCTION

1.1 HISTORY OF WEATHER FORECASTING

During the early 20th century, meteorological science was not much understood. Basic physical concepts were understood and the physics of fluid flow had been applied to the atmosphere. The basic equations of atmospheric flow had already been formulated. It was also clear that solutions of these equations were not going to be easy to come by. Primarily linear equations were used. Hence, pre-20th century meteorology was a sort of patchwork quilt of isolated theoretical problems that could be solved analytically and a collection of vague empirical results that provided little insight and no systematic basis for forecasting.

The development of telegraphy brought about a revolution, because weather observations at more or less the same time could be collected and turned into weather. Weather systems could be identified and tracked as they moved and evolved, so weather forecasting was becoming a real possibility.

The next big advance was associated with Lewis Fry Richardson, who envisioned a way to solve the equations numerically. His forecasting system was essentially what we now call a "primitive equation" model, which even attempted to include the effect of decaying vegetation! Richardson actually solved the system of finite difference equations *by hand* for a real case, not realizing there were some serious problems with his approach. Anyway, Richardson's efforts represented a watershed in thinking about the problem of weather forecasting.

The next big breakthrough came with the development of digital computers and the entrance of John von Neumann into the field, along with a number of collaborators, including Jule Charney, Ragnar Fjortoft, John Freeman, and others. This team chose to work with a filtered system of equations rather than the primitive equations, and they also were much more knowledgeable than Richardson when it came to numerical methods. Hence, they were able to develop some reasonable forecasting results. The model they

developed was essentially a barotropic model and so was pretty simple in its concepts. However, the ability to solve nonlinear forecast problems through the process of using digital computers to solve the finite approximations to a continuum mathematical model of an atmospheric flow was the key demonstration. If forms the backbone of all modern numerical weather prediction models and inspired a whole new subfield of meteorology: numerical simulations of atmospheric processes that would involve the "solution" of mathematical models.

Since then, we have seen an explosion in the field. In fact, the very term "dynamic" meteorology has come to be virtually synonymous with numerical solution of mathematical atmospheric models of all sorts. As the operational NWP models have improved, they have come to be the de facto standard by which operational forecasts are judged. In combination with statistical post-processing, the output from numerical models can be used to develop weather forecasts that can be used to create forecasts that are indistinguishable in form (if not in content) from those produced by human weather forecasters.

1.2 WEATHER FORECASTING SYSTEM (WFS)

This system forecasts the weather of a particular place on the basis of previous data about its climatic conditions viz. maximum temperature, minimum temperature, sunrise time, sunset time, maximum humidity, minimum humidity, rainfall and snowfall. The cities selected are Shimla, Chandigarh, New Delhi, Kolkata.

Our system forecasts the weather by deducting data and information on basis of training the system with examples. This is done by using "learning by examples" with if-then approach.

2. DESIGN AND IMPLEMENTATION

Development of WFS is divided into three main sub systems as illustrated in Figure 2:

1) Knowledge Base,
2) Inference Engine,
3) User Interface.

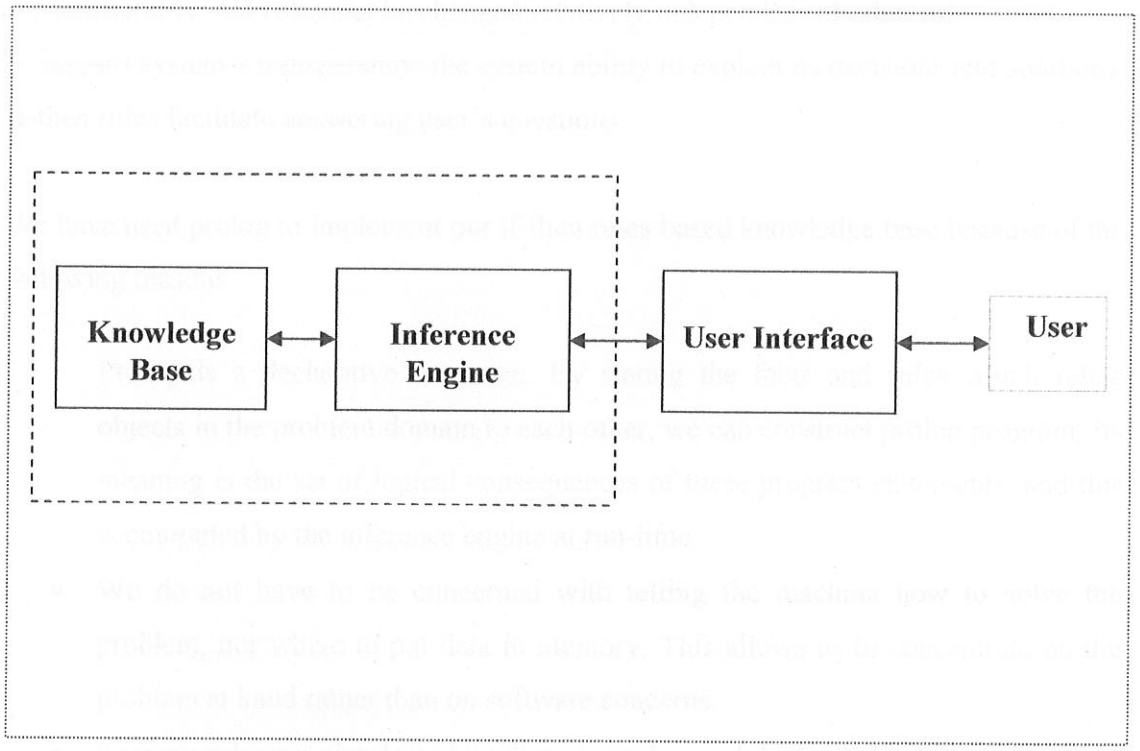

Figure 2 The structure of WFS

2.1 KNOWLEDGE BASE

The knowledge base comprises the knowledge which consists of historical meteorological data of four years, from April 2002 to April 2006. It also consists or rule and constraint that describe phenomena and parameters, method, heuristic and ideas for forecasting. There is separate knowledge base for different cities.

We have used **if-then rules** for expressing knowledge:

If condition A

Then conclusion B

We have used if-then rules for expressing knowledge because it has the following desirable feature:-

- Modularity: each rule defines a small, relatively independent piece of knowledge.
- Incrementability: new rules can be added to the knowledge base relatively independently of other rules.
- Modifiability: old rules can be changed relatively independent of other rule
- Support system's transparency: the system ability to explain its decisions and solutions. If-then rules facilitate answering user's questions.

We have used prolog to implement our if-then rules based knowledge base because of the following reasons:

- Prolog is a declarative language. By stating the facts and rules which relate objects in the problem domain to each other, we can construct prolog program. Its meaning is the set of logical consequences of these program statements, and this is computed by the inference engine at run-time.
- We do not have to be concerned with telling the machine how to solve the problem, nor where to put data in memory. This allows us to concentrate on the problem at hand rather than on software concerns.
- Scoping rules are simple and uniform in prolog and declaration of variable names is not required. This reduces code size and opportunities for error.
- Prolog programs tend to be from five to ten times smaller than the equivalent procedural programs. This reduces the opportunity for human error and reduces maintenance cost.
- Prolog is widely used in Artificial Intelligence. It is a powerful general-purpose programming language with efficient implementations available on most computing platforms today.

Sample representation of knowledge base for the city Chandigarh:-

Format of facts

weather(date, month, year, max temp, min temp, max humid, min humid, rainfall, snowfall, sunrise, sunset).

weather(1,1,2005,11,23,45,67,4,1,5,18).

weather(2,1,2005,25.8,13.0,87,30,0,0,6.40,6.21).

weather(3,1,2005,22.4,11.2,88,22,0,0,6.47,6.22).

weather(4,1,2005,28.2,11.0,88,34,0,0,6.45,6.22).

weather(5,12005,28.6,12.0,82,26,0,0,6.44,6.23).

weather(6,6,2005,43.6,26.4,42,12,0,0,5.19,7.23).

weather(7,6,2005,34.4,23.1,40,10,0,0,5.19,7.23).

weather(8,6,2005,35.2,24.6,44,16,0,0,5.19,7.24).

weather(9,6,2005,36.6,28.0,49,24,0,0,5.19,7.24).

weather(10,6,2005,39.8,24.0,61,23,0,0,5.19,7.25).

weather(11,6,2005,39.0,23.4,75,20,0,0,5.19,7.25).

weather(12,6,2005,41.8,27.0,51,12,0,0,5.19,7.26).

weather(13,6,2005,39.2,28.4,45,12,0,0,5.19,7.26).

weather(14,6,2005,40.2,29.2,49,19,0,0,5.19,7.27).

weather(15,6,2005,39.0,28.4,48,19,0,0,5.20,7.27).

weather(16,6,2005,40.2,27.0,57,20,0,0,5.20,7.27).

weather(17,6,2005,41.4,26.4,55,15,0,0,5.20,7.27).

weather(18,6,2005,42.4,26.8,53,15,0,0,5.20,7.27).

weather(19,6,2005,42.4,28.0,46,16,0,0,5.20,7.28).

weather(20,6,2005,42.6,27.2,58,19,0,0,5.20,7.28).

Similarly for all date, month, year and different cities.

Rules:-

maxt(X,Y,Z,T):-weather(X,Y,Z,_,T,_,_,_,_,_,_).

mint(X,Y,Z,M):-weather(X,Y,Z,M,_,_,_,_,_,_,_).

maxh(X,Y,Z,P):-weather(X,Y,Z,_,_,P,_,_,_,_,_).

```
minh(X,Y,Z,H):-weather(X,Y,Z,_,_,_,H,_,_,_,_).
rain(X,Y,Z,N):-weather(X,Y,Z,_,_,_,_,N,_,_,_).
snow(X,Y,Z,W):-weather(X,Y,Z,_,_,_,_,_,_,W,_,_).
rise(X,Y,Z,R):-weather(X,Y,Z,_,_,_,_,_,_,R,_).
set(X,Y,Z,S):-weather(X,Y,Z,_,_,_,_,_,_,_,S).
forecast_weather(X,Y,Z):-
        maxt(X,Y,Z,T),
        mint(X,Y,Z,M),
        maxh(X,Y,Z,P),
        minh(X,Y,Z,H),
        rain(X,Y,Z,N),
        snow(X,Y,Z,W),
        rise(X,Y,Z,R),
        set(X,Y,Z,S).

forecast:-
        writeln('Enter date, month & year'),
        getdata(_,_,_).


getdata(D,M,Y):-  write('Enter Date:'),
                  read(D),
                  nl,
                  write('Enter Month:'),
                  read(M),
                  nl,
                  write('Enter Year:'),
                  read(Y),
                  nl,
                  weather(D,M,Y,_,_,_,_,_,_,_,_).
```

2.2 INFERENCE ENGINE

An inference engine knows how to actively use the knowledge from the knowledge base. The design of inference engine involves the following steps as shown in Figure 3:

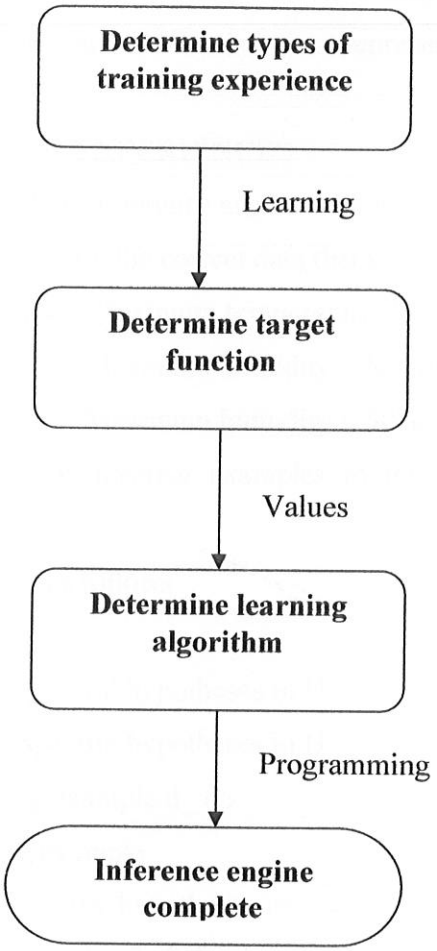

Figure 3 Steps involved in the design of inference engine

The inference engine addresses the following issues:-

- Which algorithms can approximate functions well and when?
- Influence of number of training examples on accuracy.
- Influence of noisy data on accuracy.
- Theoretical limits of learnability.
- How can prior knowledge of learner help?
- How can systems alter their own representations?

2.2.1 TRAINING EXPERIENCES

We have trained our inference engine with help of positive and negative examples.

Positive examples are the correct data that we feed to our machine.

Negative examples: Maximum temperature < Minimum temperature

Maximum humidity < Minimum humidity

Maximum humidity < 50% and it rains

The machine adds positive examples to the knowledge base and removes negative examples.

The algorithm is as follows:

G _ maximally general hypotheses in H

S _ maximally specific hypotheses in H

For each training example d_ do

If d is a positive example

_ Remove from G any hypothesis inconsistent with d

_ For each hypothesis s in S that is not consistent with d

Remove s from S

Add to S all minimal generalizations h of s such that

__ h is consistent with d_ and

__ some member of G is more general than h

Remove from S any hypothesis that is more general than another hypothesis in S

If d is a negative example

_ Remove from S any hypothesis inconsistent with d

_ For each hypothesis g in G that is not consistent with d

Remove g from G

2.2.2 TARGET FUNCTION

((.4*x1) + (.6*x2) + (.8*x3) + (x4)) / (2.8)

x1 = data of first year

x2 = data of second year

x3 = data of third year

x4 = data of fourth year

2.2.3 LEARNING ALGORITHM

Learning of simple if-then rules

```
learn( Class) :-
  bagof( example( ClassX, Obj), example( ClassX, Obj), Examples),
  learn( Examples, Class, Description),
  nl, write( Class), write(' <== '), nl,
  writelist( Description),
  assert( Class <== Description).
learn( Examples, Class, []) :-
 not member( example( Class, _ ), Examples).
learn( Examples, Class, [Conj | Conjs]) :-
learn_conj( Examples, Class, Conj),
  remove( Examples, Conj, RestExamples),
  learn( RestExamples, Class, Conjs).
learn_conj( Examples, Class, []) :-
  not ( member( example( ClassX, _ ), Examples).
learn_conj( Examples, Class, [Cond | Conds]) :-
  choose_cond( Examples, Class, Cond),
  filter( Examples, [ Cond], Examples1),
```

```
  learn_conj( Examples1, Class, Conds).
choose_cond( Examples, Class, AttVal) :-
  findall( AV/Score, score( Examples, Class, AV, Score), AVs),
  best( AVs, AttVal).
best( [ AttVal/_], AttVal).
best( [ AV0/S0, AV1/S1 | AVSlist], AttVal) :-
  S1 > S0, !,
  best( [AV1/S1 | AVSlist], AttVal);
  best( [AV0/S0 | AVSlist], AttVal).
filter( Examples, Cond, Examples1) :-
  findall( example( Class, Obj),
  ( member( example( Class, Obj), Examples), satisfy( Obj, Cond)),Examples1).
remove( [], _, []).
remove( [example( Class, Obj) | Es], Conj, Es1) :-
  satisfy( Obj, Conj), !,
  remove( Es, Conj, Es1).
remove( [E | Es], Conj, [E | Es1]) :-
  remove( Es, Conj, Es1).
satisfy( Object, Conj) :-
  not ( member( Att = Val, Conj),
      member( Att = ValX, Object),
      ValX \== Val).
score( Examples, Class, AttVal, Score) :-
  candidate( Examples, Class, AttVal),
  filter( Examples, [ AttVal], Examples1),
  length( Examples1, N1),
  count_pos( Examples1, Class, NPos1),
  NPos1 > 0,
  Score is 2 * NPos1 - N1.
candidate( Examples, Class, Att = Val) :-
  attribute( Att, Values),
  member( Val, Values),
```

```
  suitable( Att = Val, Examples, Class).
suitable( AttVal, Examples, Class)  :-
  member( example( ClassX, ObjX), Examples),
  ClassX \== Class,
  not satisfy( ObjX, [ AttVal]), !
count_pos( [], _, 0).
count_pos( [example( ClassX,_ ) | Examples], Class, N)  :-
  count_pos( Examples, Class, N1),
  ( ClassX = Class, !, N is N1 + 1; N = N1).
writelist( []).
writelist( [X | L])  :-
  tab( 2), write( X), nl,
  writelist( L).
```

2.3 USER INTERFACE

The user interface caters for smooth communication between the user and the system, also providing the user with an insight into the problem solving process, carried out by the inference engine.

We have used XPCE to design our user interface.

XPCE offers the following advantage:-

- An object oriented language for building GUI.
- It provides high level GUI specification primitives and dynamic modification of the program .
- Provides a hybrid environment for Graphical User Interfaces.
- Available free under GNU GPL

Snapshot of user's interface is shown in Figure. 4

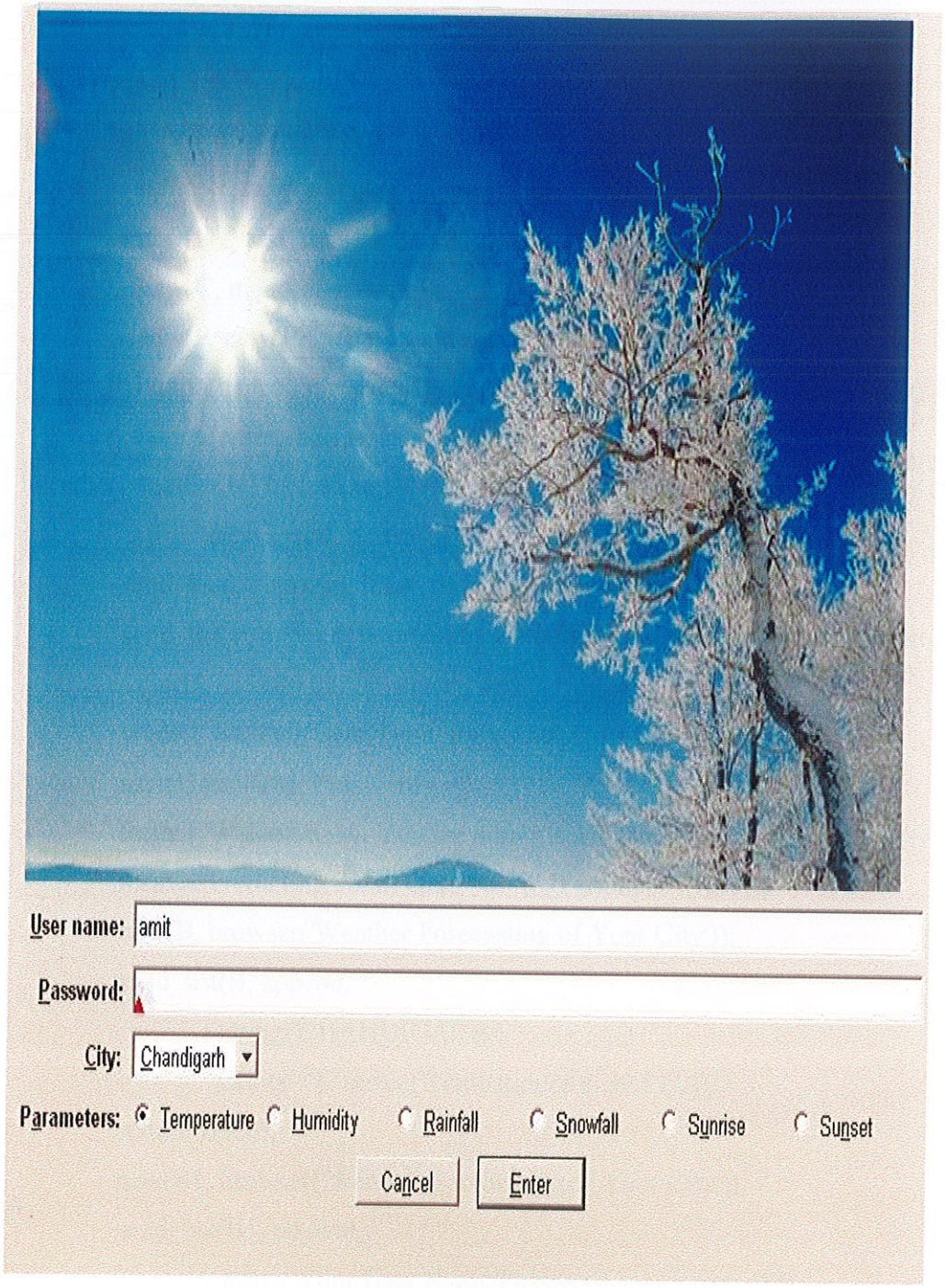

Figure 4 Snapshot of user's interface

2.3.1 CODE FOR THE USER'S INTERFACE

```
forecast_weather :-
        new(D, dialog('Weather Forecasting System')),
        send_list(D, append,
        [ label(identifier, image('na.jpg')),
          new(N1, text_item(user_name)),
          new(N2, text_item(password)),
          new(C, menu(city, cycle)),
          new(P, new(P, menu(parameters))),
          button(cancel, message(D, destroy)),
          button(enter,and(message(@prolog,
predict_weather,N1?selection,N2?selection,C?selection,P?selection ),
                message(D, destroy))) ] ),
        send_list(C, append, [chandigarh, delhi, kolkata, shimla]),
        send_list(P, append, [temperature, humidity, rainfall, snowfall, sunrise, sunset]),
        send(D, default_button, enter), send(D, open).
        predict_weather(UserName, Password, City, Parameters) :-
        user(UserName, Password),
        format('Welcome ~w, You are authorised to view this page ~n',[UserName]),
        format('Predicting ~w of city  ~w ~n',[Parameters,City]),
        new(B, browser('Weather Forecasting of Your City')),
        send_list(B, append,
                [ string('CHANDIGARH'),
                  string('  '),  string('Temperature is 39.5 c')]),
        send(B,open);
        new(B1, browser('Weather Forecasting of Your City')),
        send_list(B1, append,
                [ string('Your User Name/Password is incorrect !!!'),
                  string('You are not authorised to view this page.')]),
        send(B1, open).  user(amit,star).
:- forecast_weather.
```

2.4 CONNECTING THE KNOWLEDGE BASE, INFERENCE ENGINE AND THE USER INTERFACE

The XPCE message passing system is guarded with a single mutex, which synchronizes both access from Prolog and activation through the GUI.

Using XPCE in the foreground simplifies debugging of the UI and generally provides the most comfortable development environment. The GUI creates new threads using thread create/3 and, after work in the thread is completed, the sub-thread signals the main thread of the the completion using in pce thread/1.

in _pce _thread(:Goal)

Assuming XPCE is running in the foreground thread, this call gives background threads the opportunity to make calls to the XPCE thread. A call to in pce thread/1 succeeds immediately, copying Goal to the XPCE thread. Goal is added to the XPCE event-queue and executed synchronous to normal user events like typing and clicking.

The prolog/XPCE interface is shown in Figure 5

The behavioral model of various message passing is shown in Figure 7

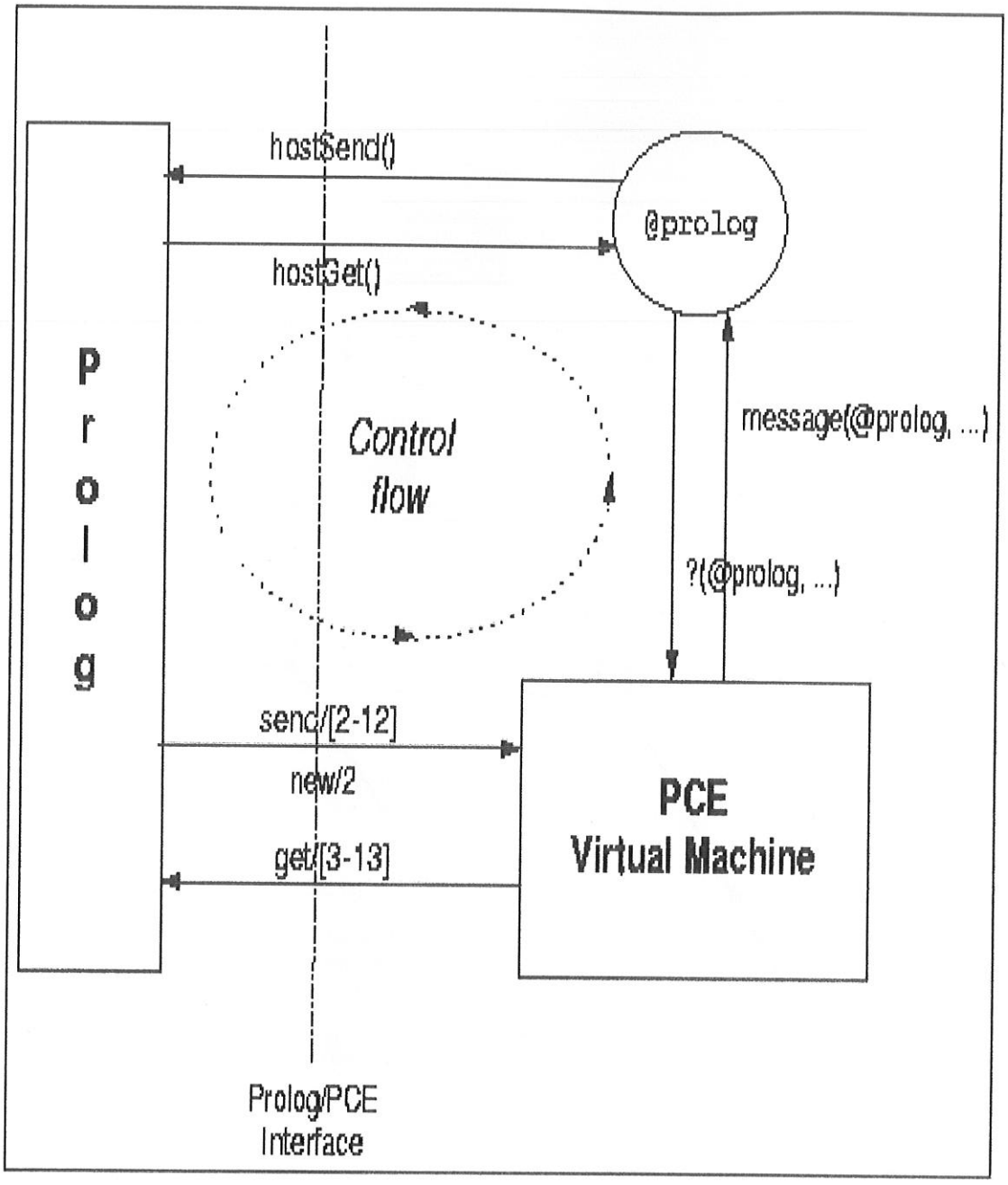

Figure 5 Data and control flow in XPCE

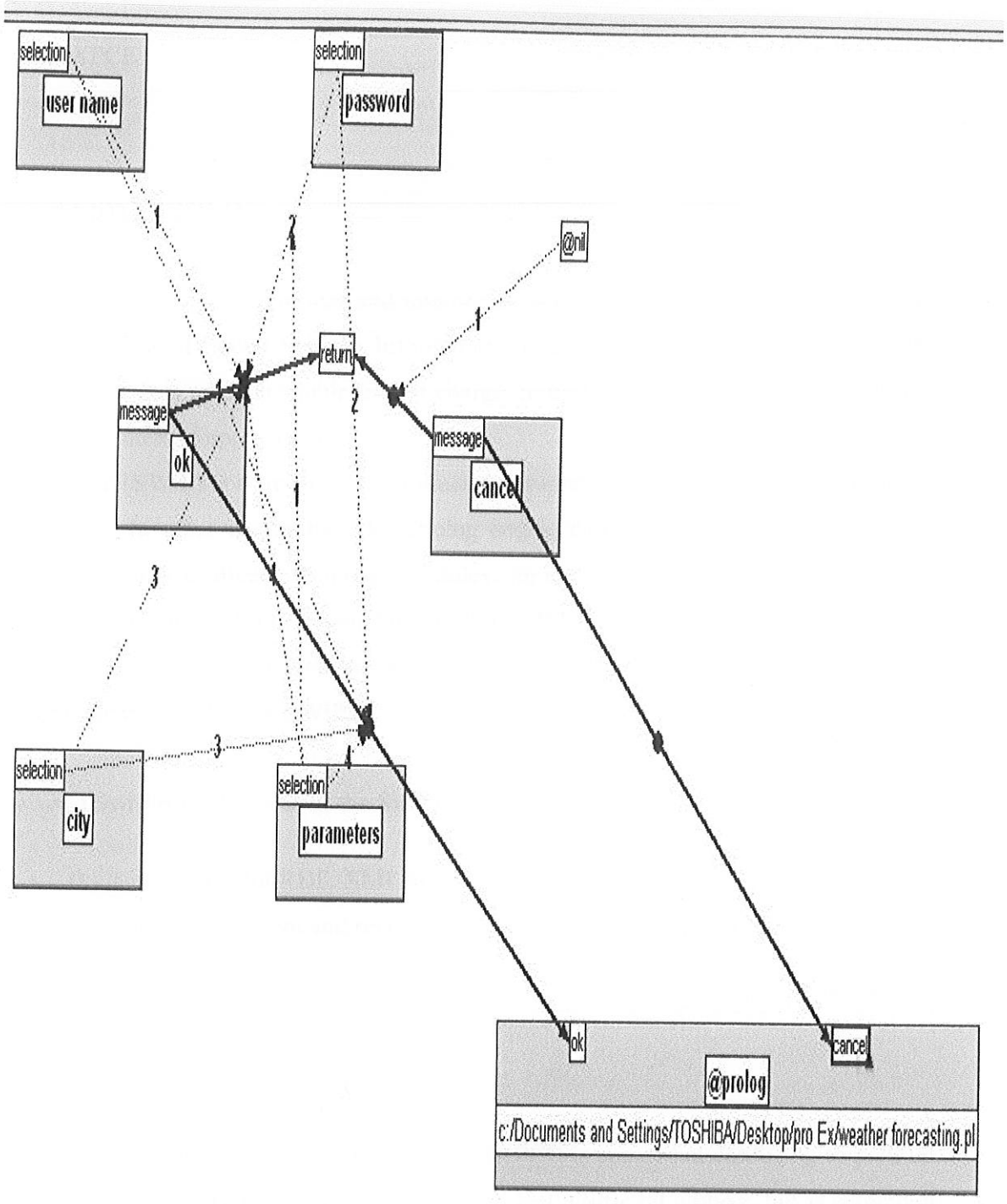

Figure 6 The behavioral model

3. TOOLS USED

1) Swi-Prolog
2) XPCE

3.1 SWI-PROLOG

SWI Prolog is a prolog compiler and interpreter suite developed by Jan Wielemaker of the Department of Social Science Informatics of the University of Amsterdam in the Netherlands. It is made available free of charge, in both source and binary distributions, for non-commercial applications.

SWI Prolog (SWIP) is a well-crafted product that supports embedding and programmatic extensions. In other words, the SWI Prolog engine can be used as a component in a larger framework of diverse language modules. In addition, SWIP is easily extended to support additional capabilities available on Microsoft Windows platforms.

This extensibility is greatly enhanced by the fact that complete Windows-compatible source code is available for SWIP.

We used Swi-Prolog because of the following features:-

- Built-in support for RDF, XML and HTML parsing.
- Network, HTTP client and server, database and file I/O connectivity.
- Multithread capable.
- Comprehensive library of Prolog functions.
- Support for constraint solving.
- ODBC database connectivity.
- XPCE graphical user interface builder (XPCE is somewhat confusing to pick up at first, and results in GUIs that look and feel a little bit clumsy, but is actually a really simple and powerful user interface API. XPCE allows you produce front-

ends to your Prolog applications in a really short time – the interface won't be a slick work of art but it will certainly be very functional and quick to develop).

3.2 XPCE

XPCE is an object-oriented system. This implies that the basic entity in XPCE's world is an object, an entity with state capable of performing actions. Such an action is activated by sending the object a *message*.
So far, most object oriented systems agree. Starting from these notions however one can find object oriented environments that take widely different approaches for representing objects, actions on objects and sending messages.
Rather than specifying operations on each individual object most OO environments define some way of sharing the operation definitions (called *methods*). There are two ways to share methods. One is to create objects as a copy of other objects and then modify them (by attaching and deleting slots and methods) to fit the particular need. If a series of similar objects is needed, one first creates an object that satisfies the common functionality and then creates multiple copies of this object. This approach is followed by SELF. The other —more traditional— approach is to define a *class*. A class is an entity in the object oriented environment that defines the constituents of the persistent state and the methods for each of its *instantiations*.
XPCE takes the latter approach, but adds some notions of the object-copying approach because GUI's often contain unique objects and because object modification is more dynamic and therefore more suitable for rapid prototyping.

Using XPCE offers the following advantages

- XPCE is a Graphical User Interface system (GUI). XPCE is not a programming language itself. Instead it may be connected to a programming language to form a hybrid development environment.

- XPCE may be connected to any programming language, but it fits best with languages that are dynamically typed or have strong static typing and allow for programmable type conversion . A dynamically typed language is a language with untyped variables for which the type of the current value of a variable can be deduced at runtime. Currently XPCE defines interfaces to Prolog, Lisp and C++.
- XPCE is an object management system (storage and message-passing) which can have its methods defined in various languages. It offers a large number of built-in classes that concentrate on graphical user interfaces.
- XPCE method resolution is done at runtime. Classes and methods may be inspected and modified at runtime. This makes XPCE especially useful in combination with interpreted languages for GUI prototyping. Proper method and graphical caching generally provides good performance for interactive applications.
- XPCE graphical capabilities

3.2.1 XPCE GRAPHICAL CAPABILITIES

3.2.1 a) DIALOGUE DESIGN

XPCE contains all the standard *controls*: buttons, various styles of menus, sliders, text entry fields, etc. Layout of controls in a window is normally specified in terms of `above', `below', `left' and `right'. Layout may be refined by changing the alignment details. For emergencies, it is possible to specify the layout in coordinates.

Finally, XPCE/Prolog provides a direct-manipulation interface for the definition of dialogue windows.

3.2.1 b) INTERACTIVE DIAGRAM EDITORS

Implementation of graphical editors for diagramming languages is a very common application area for XPCE. XPCE offers full object-oriented graphicals, opaque

and transparent graphicals, composition of primitive graphicals into compound graphical objects (recursive), automatically maintained graphical relations and a comprehensive and extensible library of `gestures' (objects that allow the user to manipulate graphical objects with the mouse).

3.2.1 c) TEXT MANIPULATION

XPCE offers a programmable text-editor similar to **GNU-Emacs**. The editor offers about 150 predefined methods. It can handle multiple fonts, embedded graphics (prototype version), mouse-sensitive areas, etc.
These features make the editor well suited for the implementation of WYSIWYG **hypertext** editor. The XPCE/Prolog library offers such an editor to implement **help** for XPCE applications. These files are written using this editor and then converted to **HTML**.

3.2.1 d) INTERPROCESS COMMUNICATION

To facilate graphical user interfaces for traditional stream-based Unix applications as well as to realise client-server applications, XPCE offers interprocess and networking communication primitives.

3.3 DIFFERENCE BETWEEN XPCE AND PROLOG:

XPCE and Prolog are very different systems based on a very different programming paradigm. XPCE objects have global state and use destructive assignment. XPCE programming constructs use both procedures (code objects and send-methods) and functions (function objects and get-methods). XPCE has no notion of non-determinism unlike prolog. The hybrid XPCE/Prolog environment allows the user to express functionality both in Prolog and in XPCE. This chapter discusses representation of data and dealing with object-references in XPCE/Prolog.

3.3.1 XPCE IS NOT PROLOG

Data managed by Prolog consists of logical variables, atoms, integers, floats and compound terms (including lists). XPCE has natural counterparts for atoms (a name object), integers (a XPCE int) and floating point numbers (a real object). Prolog logical variables and compound terms however have no direct counterpart in the XPCE environment. XPCE has variables (class var), but these obey totally different scoping and binding rules. Where Prolog uses a compound term to represent data that belongs together (e.g. person (Name, Age, Address)), XPCE uses objects for this purpose:

```
:- pce_begin_class(person(name, age, address), object).
variable(name, name, both, "Name of the person").
variable(age, int, both, "Age in years").
variable(address, string, both, "Full address").
initialise(P, Name:name, Age:int, Address:string) :->
"Create from name, age and address"
send(P, name, Name),
send(P, age, Age),
send(P, address, Address).
:- pce_end_class.
1 ?- new(P, person(fred, 30, 'Long Street 45')).
```

These two representations have very different properties:

• Equality

Prolog cannot distinguish between 'person('Fred', 30, 'Long Street 45')' and a second instance of the same term. In XPCE two instances of the same class having the same state are different entities.

• Attributes

Whereas an attribute (argument) of a Prolog term is either a logical variable or instantiated to a Prolog data object, an attribute of an object may be assigned to. The assignment is destructive.

• Types

XPCE is a dynamically typed language and XPCE object attributes may have types. Prolog is untyped.

3.3.2 DEALING WITH PROLOG DATA

By nature, XPCE data is not Prolog data. This implies that anything passed to a XPCE method must be converted from Prolog to something suitable for XPCE. A natural mapping with fast and automatic translation is defined for atoms, and numbers (both integers and floating point). As we have seen in section 2, compound terms are translated into instancesusing the functor-name as class-name.

In XPCE 5.0 we added the possibility to embed arbitrary Prolog data in an object. There are three cases where Prolog data is passed natively embedded in a instance of the class prolog term.

• Explicit usage of prolog(Data)

By tagging a Prolog term using the functor prolog/1, *Data* is embedded in an in-stance of prolog term. This term is passed unaltered unless it is passed to a method that does not accept the type Any, in which case translation to an object is enforced.

• When passed to a method typed Prolog

Prolog defined methods and instance-variables (see section 7) can define their type as Prolog. In this case the data is packed in a prolog term object.

• When passed to a method typed unchecked

A few methods in the system don't do type-checking themselves.

We will explain the complications using examples. First we create a code object:

```
1 ?- new(@m, and(message(@prolog, write, @arg1),
message(@prolog, nl))).
```

This code object will print the provided argument in the Prolog window followed by a newline:

```
2 ?- send(@m, forward, hello).
hello
```

From this example one might expect that XPCE is transparent to Prolog data. This is true for integers, floats and atoms as these have a natural representation in both languages. However:

```
3 ?- send(@m, forward, chain(hello)).
@774516
4 ?- send(@m, forward, 3 + 4).
```

5 ?- send(@m, forward, [hello, world]).

@608322

In all these examples the argument is a Prolog compound term which —according to the definition of send/3— is translated into a XPCE instance of the class of the principal functor.

In 3) this is an instance of class chain. In 4) this is an instance of class +. Class + however is a subclass of the XPCE class function and function objects are evaluated when given to a method that does not accept a function-type argument. Example 5) illustrates that a list is

converted to a XPCE chain.

We can fix these problems using the prolog/1 functor. Example 7) illustrates that also non-ground terms may be passed.

6 ?- send(@m, forward, prolog(chain(hello))).

chain(hello)

7 ?- send(@m, forward, prolog(X)).

_G335

X = _G335

Below is a another realistic example of this misconception.

1 ?- new(D, dialog('Bug')),

2 send(D, append, button(verbose,

3 message(@prolog, assert,

4 verbose(on)))),

5 send(D, open).

6 [PCE warning: new: Unknown class: verbose

7 in: new(verbose(on))]

One correct solution for this task is below. An alternative is to call a predicate set verbose/0 that realises the assertion.

1 make_verbose_dialog :-2

new(D, dialog('Correct')),

3 send(D, append,

4 button(verbose,

```
5 message(@prolog, assert,
6 prolog(verbose(on)))),
7 send(D, open).
```

3.3.3 LIFE-TIME OF PROLOG TERMS IN XPCE

XPCE is connected to Prolog through the foreign language interface. Its interface predicates are passed Prolog terms by reference. Such a reference however is only valid during the execution of the foreign procedure. So, why does the example above work? As soon as the send/3 in make verbose dialog/0 returns the term-reference holding the term verbose(*o*n) is no longer valid!

To solve this problem, prolog term has two alternative representations. It is created from a term-reference. After the interface call (send/3 in this case) returns, it checks whether it has created Prolog term objects. If it finds such an object that is not referenced, it destroys the object. If it finds an object that is referenced it records Prolog terms into the database and stores a reference to the recorded database record.

Summarizing, Prolog terms are copied as soon as the method to which they are passed returns. Normally this is the case if a Prolog terms is used to fill an instance-variable in XPCE.

XPCE is an object-oriented system. This implies that the basic entity in XPCE's world is an object, an entity with state capable of performing actions. Such an action is activated by sending the object a *messag*e.

So far, most object oriented systems agree. Starting from these notions however one can find object oriented environments that take widely different approaches for representing objects, actions on objects and sending messages.

Rather than specifying operations on each individual object most OO environments define some way of sharing the operation definitions (called *method*s). There are two ways to share methods. One is to create objects as a copy of other objects and then modify them (by attaching and deleting slots and methods) to fit the particular need. If a series

of similar objects is needed, one first creates an object that satisfies the common functionality and then creates multiple copies of this object. This approach is followed by SELF. The other —more traditional— approach is to define a *class*. A class is an entity in the object oriented environment that defines the constituents of the persistent state and the methods for each of its *instantiation*s.

XPCE takes the latter approach, but adds some notions of the object-copying approach because GUI's often contain unique objects and because object modification is more dynamic and therefore more suitable for rapid prototyping.

4. WORKING OF THE SOFTWARE

As we start WFS, we get the following view :

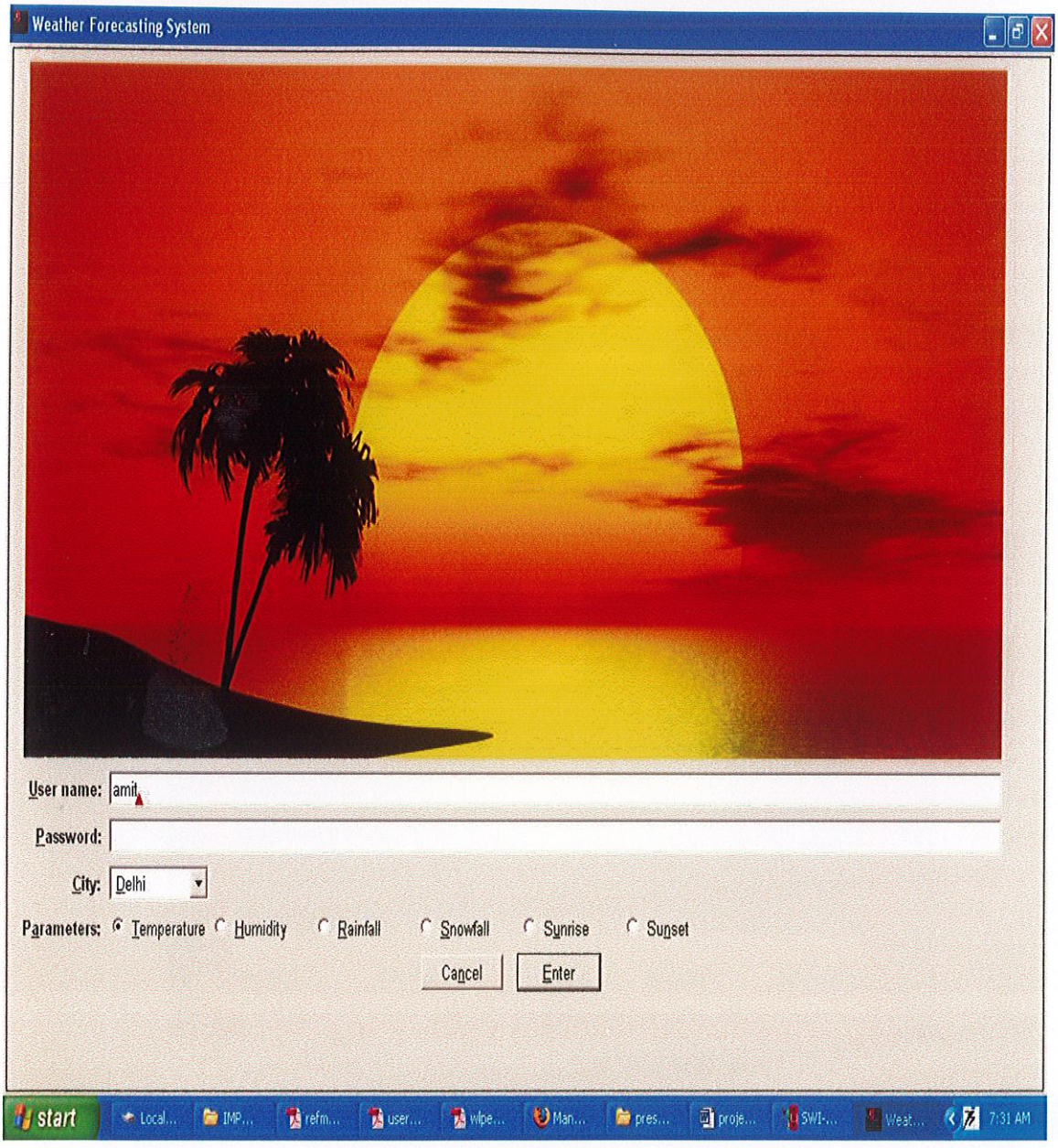

Figure 7 Snapshot of first screen

Enter user name and password and choice of city

And click enter,

And we get the following response ,

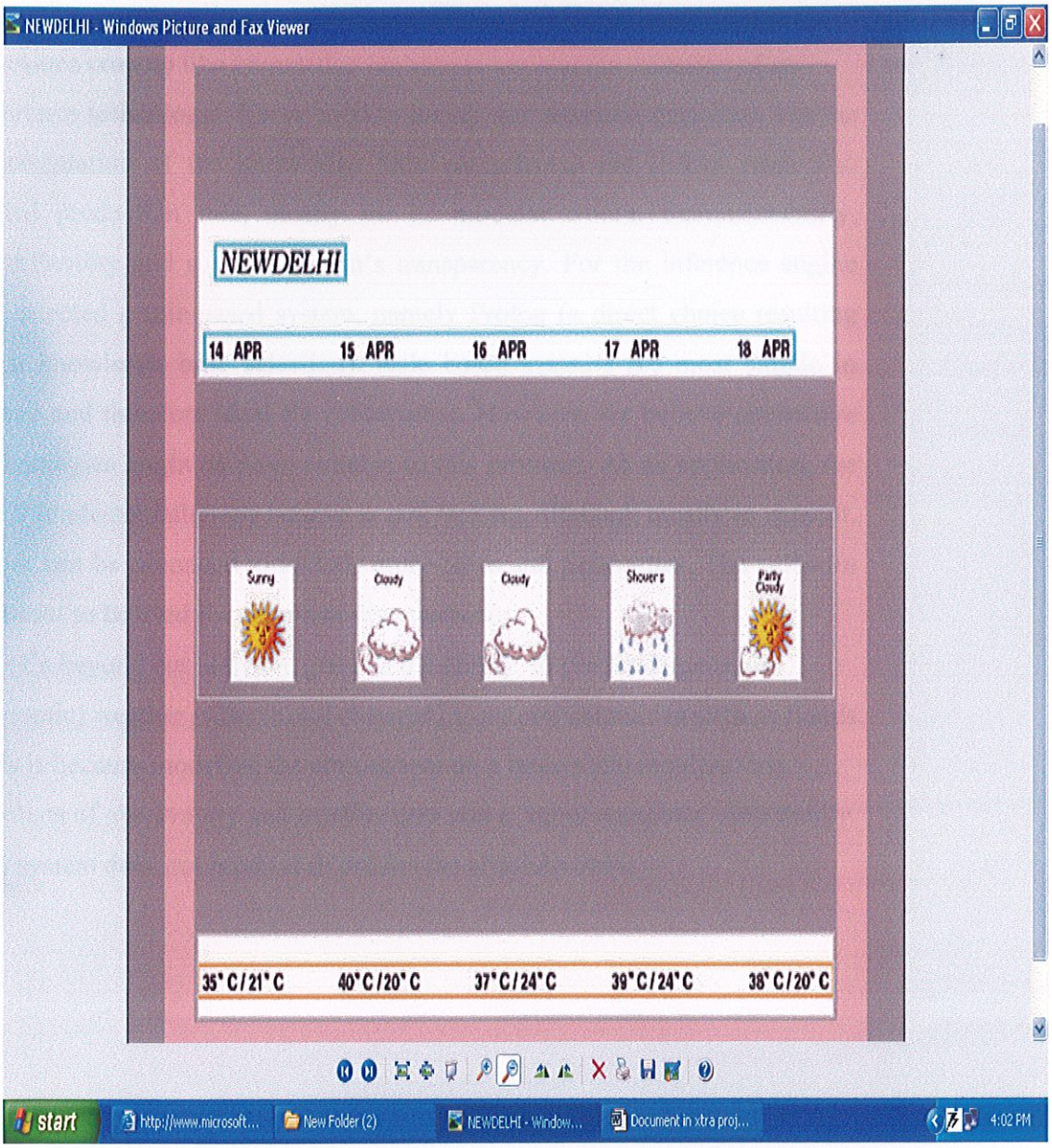

Figure 8 snap shot of second

CONCLUSION

We have introduced the weather forecasting system, a prototype application to predict various climatic conditions. While building such a system many questions crop up like knowledge representation and the selection of an inference technology. We've tried to justify our selection decisions. For the representation of the knowledge base we selected the if-then rules also called production rule, mostly for its modular nature, incrementability, modifiability and support system's transparency. For the inference engine we selected a rule-based system, namely Prolog (a direct choice resulting from knowledge base selection). Rule based systems are most simple in nature and therefore ideal for prototyping. However, we believe alternative technologies might be more suitable to this problem. As an application, the WFS implementation we believe is still lacking, although mostly in content. Work can be expanded to include more cities and parameters. The software is meant to be used for demonstrative purpose.

What's beyond our software grasp is the ability to predict large-scale (synoptic) weather patterns and changes i.e. natural calamities such as floods. This is because modeling the atmosphere on a large scale requires vast amounts of observatory and satellite data and a super computer. And finally this system does not promise to predict the absolute truth.

BIBLIOGRAPHY

Books

1) PROLOG – Programming for artificial intelligence

   Author: - Ivan Bratko

   Publisher: - Pearson Education

2) Introduction to artificial intelligence and expert systems

   Author: - Dan W. Patterson

   Publisher: - Prentice-Hall India

Websites :

http://www.swi-prolog.org

http://www.swi.psy.uva.nl/products/XPCE/.

http://gollem.science.uva.nl/twiki/pl/bin/view/Development/MultiThreadsXPCE