A Novel Exception Handling Scheme for Out Patient Workflow in a Wireless Handheld Hospital Environment

Suneetha Uppu, Doan B.Hoang, Tom Hintz
Advanced Research in Networking Laboratory, Faculty of Information Technology, University of Technology, Sydney
{suppu, dhoang, hintz}@it.uts.edu.au

Abstract
The healthcare services are among the fastest growing service markets deploying wireless technologies. As healthcare organizations continue to become more distributed, mobile handheld devices and their associated support has evolved into a viable platform for these organizations. However, one factor that hinders the deployment of the handheld technology is the lack of support for exception handling over mobile handheld devices in current workflow management systems. In this paper, we propose an extended architecture and scheme for handling the exceptions. The proposed scheme scans through earlier exception records from a dynamic knowledge base, evaluates and suggests an appropriate solution for the current exception on the mobile handheld device. The scheme is evaluated in terms of time and accuracy compared with previously known schemes [6].

Keywords: *Exception Handling, Workflow Management, Handheld System, Healthcare Services*

1. Introduction

Over the last five years, the healthcare industry has projected to be the largest vertical market to adopt wireless technology. As healthcare organizations continue to become more distributed, they are increasing their reliance on mobile links by improving the convenience to the caregivers. The entire healthcare industry is centered on the patient's clinical records and involves activities such as retrieving information about the patients and exchanging information among healthcare professionals. The above activities are heavily paper based; they are characterized by duplications of information, possible errors, omissions and delays. This situation was dramatically improved by automating workflow in the wards by incorporating Workflow management systems.

Workflow management systems (WFMSs) are software systems that support the specification and execution of business processes. WFMSs allow the description of business processes in terms of sub processes, activities, and execution dependencies among activities, and handle the control of process execution by scheduling activities and by assigning them to the proper executing agent. WFMSs can effectively support processes that feature a predictable and repeatable behavior [1]. During the enactment of a business process many exceptions, that is, deviations from the ideal sequence of events occur. It has been observed that exceptions happen rather frequently in real working environments [2, 3]. Exceptions may affect the execution of every process. These exceptions should be handled successfully to resolve as they occur for the efficient process execution.

Recently, many new types of exceptions in the workflow and the need for their handling have been identified. Many techniques have been proposed to handle the exceptional situations in workflows. Usually the performance of an exception-handling scheme is strongly influenced by the way the exceptions are classified. In the current literature, exceptions are classified into either expected or unexpected exceptions. Expected exceptions are known in advance to the workflow designer. A patient cancels his appointment with the doctor or the violation of constraints over data or temporal conditions are few examples for expected exceptions [4]. These types of exceptions have been studied

intensively within the context of extended transaction models[1] [5].

Unexpected exceptions occur as the consequences of unanticipated events or something that just cannot be predicted at the design time. Traditionally, these can only be handled with human involvement. Currently, there is growing interest in handling these exceptions approaches, which include ignore, retry, compensation or alternative task. Some of the recent research proposals include exception handling methods and tools within their frameworks over WFMSs. However, none of the realized research prototypes incorporate exception-handling mechanism of WFMSs over mobile hand held devices in health care.

In this paper, we propose an extended novel architecture approach to handle the unexpected exceptions, and we integrate the exception handling workflow architecture with the mobile handheld device. We also describe the attributes of exceptions, the degree of similarity between two exceptions and propose an algorithm for suggesting a solution to the current exception. The proposed algorithm, scans through the previous exception records from the knowledge base, and suggests the appropriate solution for the current exception on the mobile handheld device. The algorithm is evaluated in terms of time and accuracy compared with previously known algorithms [6].

The rest of the paper is structured as follows. Section 2 focuses on related work. Section 3 describes the extended novel architecture for handling the exceptions. Section 4 presents the algorithm with definitions. Section 5 presents preliminary results of the prototyped system. And section 6 concludes the paper.

2. Related Work

Health care sector is a vital area where exceptions should be handled efficiently. Exception handling in healthcare sectors is a multi-disciplinary area. It involves almost all aspects of process definition, enactment, monitoring, and administration. METEOR [3], WAMO [4], ADOME [7], ADEPT [2], EXOTICA [8] are few research projects which identified the importance of incorporating the exception handling mechanism into WFMSs. [9] presented a theoretical basis based on Petri-Net, for handling exceptions. This work is purely driven by organizational semantics rather than by a workflow process model. [10] Described Event Condition Action (ECA) rules based on object-oriented databases to handle the exceptions. In [11], an exception handling mechanism employing a combination of programming language concepts and transaction processing techniques was proposed. However, aborting or canceling a workflow task would not always be appropriate or necessary in a workflow environment. Therefore, the error handling semantics of traditional transactional processing systems are too rigid for handling exceptions in WFMSs.

[12] Reported exception taxonomy combined with the exception design pattern [13]. This intelligent exception handling mechanism ensures the similarities among cases during case retrieval and analysis. As exceptional situations are complicated, the researchers moved their way on knowledge based approaches, for handling exceptions in an intelligent way. The Process Handbook project [12] at the MIT center for Coordination Science, involved a decade of development and evaluation of systems for handling multi-agent conflicts. The exception handling mechanism discussed in this work was very high level and its applicability was not convincing, as their discussion is conducted without a workflow prototype system. In [14], the researcher activated Case-based reasoning (CBR) approaches to handle exceptions. The system collects the previous exception cases and reuses the experiences in the future. All these previous works focuses on the runtime support. Only authorized person can decide the appropriate solution for the current exception.

[6] Proposed a novel exception handling architecture and presented algorithms to scan previous records that are similar to the current exception. They also defined the exception and evaluated the algorithms on synthetic datasets to show their relative performance. The research addressed the problem of supporting unexpected exceptional behaviors and suggests the closest solution for the current exception. Although this research seems a step in the right direction, the accuracy of the solution for the current exception is reduced as the control flow between two similar exceptions are not considered. In this paper, we propose to further extend this exception handling architecture and an algorithm to provide an appropriate solution to the healthcare professionals. The degree of exception flow between similar exceptions is considered such that, the accuracy of the solution for the current exception is improved. Further, we incorporated the exception handling architecture, and the graph algorithm will be implemented over the

[1] Transactional workflows are workflows, which are able to recover and proceed in case of failures. The important techniques to support are backward recovery, forward recovery, and compensation [5].

mobile hand held prototyped system, for the healthcare professional to demonstrate the results.

The Exception Handling Model

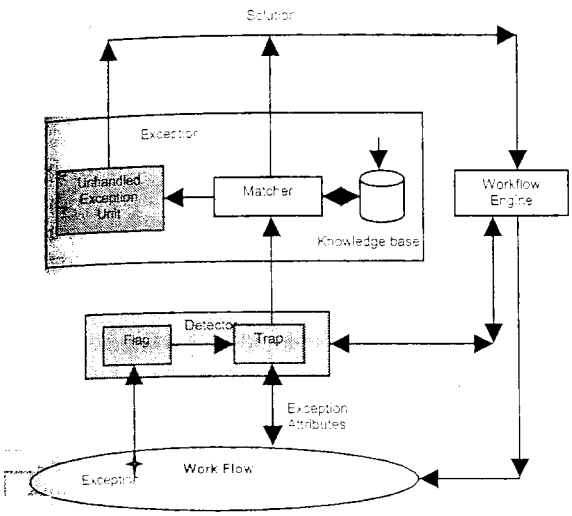

Fig 1: The exception handling architecture from [6] showing added detector

Figure 1 depicts the architecture proposed in this paper. The workflow comprises a number of logical steps, each of which is known as an activity and these are stored in a workflow database. An activity can involve manual interaction with a user or a workflow participant, or the activity might be executed using machine resources. A workflow engine is a multithreaded engine that can handle a number of workflow instances simultaneously in an outpatient workflow. It coordinates the workflow execution such that, all interactivity dependencies are enforced as specified. It also monitors the execution of activities and keeps track of their status.

Exceptions are deviations from the normal execution of the program. Exceptions are caused by inconsistent or corrupted data, resource conflict, concurrency violation, unexpected contingencies, communication errors, changing environment, etc. An exception can occur at any point of the workflow execution. When an exception occurs in the workflow, status flag is set. The flag is sticky, that is, it remains set until explicitly cleared by applying an appropriate solution to the exception or by human involvement. It signals the trap for occurrence of the exception. Trap traces the attributes of the exception and stores the information in the workflow engine. It further sends the exception to the exception handler, in which appropriate handling approach is determined.

Knowledge base contains the information about the previous experience in handling exceptions. Matcher, searches the appropriate solution for the current exception from the previous exceptions. The solution for the corresponding current exception is forwarded to the workflow engine for implementation. If none of the solution for the current exception in the knowledge base matches, then the current exception is forwarded to the Unhandled Exception Unit, where exceptions are handled manually. This exception and its handling approach are inserted into the knowledge base for future use.

3.1 Exception Attributes

The exception in this approach is defined by the following attributes [6]:

1. *Workflow ID:* It represents the ID of workflow instance.
2. *Status:* It represents the status of the flag. Flag, sets true when exception raises, sets false when exception is handled.
3. *Type:* The exception type, which describes by the set of user defined semantics.
4. *Time:* It represents the time when the exception occurs.
5. *Activity:* It represents the activity to which the exception occurred.
6. *Vertex:* It represents the vertex of the exception occurred.
7. *Edge:* It represents the direction of control flow of task where exception occurs.

It may not be possible to find the similar exception record from the knowledge base, as the time attribute cannot be same as that of current exception. i.e., two similar exceptions may not occur at the same time. [6], considers the *who* and the *activity* attributes of an exception for identifying the closest match for the current exception. The paper, however, concentrated only on the performer and the activity performed by the performer. In this case, the control flow of the task is the same irrespective of the activity performed by the professional. In practice, an activity has number of tasks irrespective of the control flows. For example, a doctor may have number of activities like, sending patient to lab tests, specialist, financial department and/or even sending the patient back to the nurse for further clarifications. However, in few cases, data may flow in both directions. Direction of data flow should be considered when exception occurs, as healthcare workflows are very complex. It's also necessary to keep track of the path of workflow instance, vertex and its neighboring vertices where the exception

occurs, when suggesting a solution for the current exception. Therefore, it is necessary to consider the control flow and data flow of each task of the performer and receptor, as proposed in our paper. The accuracy of the current exception match is increased by adding attributes *vertex* and *edge* for the current exception.

3.2 An example

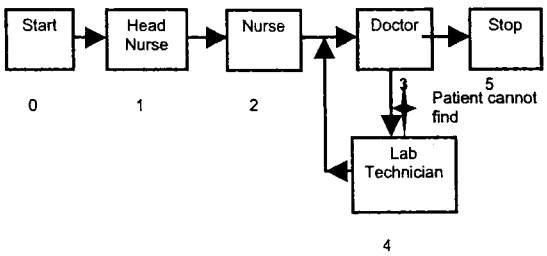

Fig 2: Out patient Workflow [15]

Figure 2 depicts the workflow of an outpatient [15]. When a patient checks into the hospital, his/her details will be entered at the reception where the workflow instance starts. The workflow engine monitors and coordinates the instance to follow the set of logical steps. The patient is forwarded to the Head Nurse, where s/he allocates the work to a Nurse. e.g., to measure weight, height, blood pressure etc. Once the Nurse has performed the scheduled activity, the patient is forwarded to the Doctor. The Doctor may forward the patient to undergo some lab tests (e.g., Blood test, Urine test, etc.) if required. Finally, the patient will be prescribed and he/she will be sent to the Finance Department, where the bills will be created. Finally, the workflow engine stops the instance, once patient checks out from the hospital.

Consider the situation where the patient cannot be found, when the Lab technician performs his/her activity. After a time threshold, an exception is raised by setting the status Flag to true. It signals the handling Trap. The exception is sent to the exception handler to handle the current exception. The knowledge base stores the records of previous exceptions and its handling mechanisms. Table 1, shows the sample records in the knowledge base for the current exception type.

In the Knowledge base, there are five records that match the exception type of "cannot find the patient". In the first record, the exception was raised by the Head nurse when data flow from the reception to the Head nurse. It was resolved by skipping the activity. In the second record, the exception was raised by Nurse when data flow from the Head Nurse to the Nurse. It was resolved by skipping the activity and filling the data before forwarding the patient to the Doctor. In the third record, the exception was raised by the Doctor when data flow from the Nurse to the Doctor. It was resolved by aborting the entire treating process. In the fourth record, the exception was raised by the Lab technician when data flow from the Doctor to the Lab technician. It was resolved by pending the examination until the patient showed up. In the fifth record, the exception was raised by the Doctor when data flow from the Lab Technician to the Doctor and is resolved by aborting the activity. The current exception focuses on the vertex and the edge of the exception records in order to find the closest degree of match. In table 1, the numbers in the parenthesis represents the vertices of the workflow and weights of the workflow instance path. The algorithm proposed in the following section, identifies the closest match of the current exception from the knowledge base. The solution of the algorithm for the current exception, cannot find the patient, is resolved by pending the activity until the patient turns up.

Table 1: A Sample Exception Knowledge Base

Workflow ID	Status	Time	Activity	Type	Vertex	Edge	Solution
----------	----------	-------	-------	Cannot find the patient	HN(1)	R(0)->HN(1) (1)	Skip
----------	----------	-------	-------	Cannot find the patient	N(1)	HN(1)->N(2) (2)	Skip and fill in the data before forwarding to the doctor
----------	----------	-------	-------	Cannot find the patient	D(3)	N(2)->D(3) (3)	Abort
----------	----------	-------	-------	Cannot find the patient	LT(4)	D(3)->LT(4) (4)	Pending
----------	----------	-------	-------	Cannot find the patient	D(3)	LT(4)->D(3) (5)	Abort

4. Algorithm

In this section, an algorithm is developed to suggest the most accurate solution for the current exception from the knowledge base. The attributes of the current exception are translated into numeric measures by using graph theory concepts.

Definition 1

The directed workflow graph G= (V, E) has *n* vertices and set of ordered pairs of vertices, E ∈ V×V. Let G' is the exception sub graph where G' ∈ G. Let, *vj* is the vertex where exception is identified whose in degree is I|(vj)|. *vi* is the corresponding neighbor vertex where exception would have been originated whose out degree is A|(vi)|.

Example: In fig 2, V={0,1,2,3,4,5} with edges {(0,1),(1,2),(2,3),(3,4),(4,3),(3,5)}. vj = 4 where exception is identified and whose edge is 3 -> 4. The in degree of 4 is 1 and out degree of 3 is 1.

Definition 2

Let W(v(i),v(j)) be the weight on the edge connecting v(i) to v(j). An exception path is a non-empty sequence of vertices P={v0,v1,...vj}. The weighted exception path length P is given by ∑ i=0 to j-1 W(vi,v(i+1)). Sum of the weight length of the exception path is the match of the exception record.

Once vertex and edge of the exception is been identified, path weight of the current exception is calculated by adding the weights followed by the path of the current workflow instance. The current workflow exception edge weight is matched with the edge weights of exception records found in knowledgebase. Thus, the algorithm picks up the closest match from the existing records.

Example: The sum of edge weighted path of the current exception is calculated as P = {w(0,1) + w(1,2) + w(2,3) + w(3,4)})}. At the end of the traversal of exception vertex and its neighboring vertices, the exception record with equal weight is identified from the knowledgebase, where Match record M is <= P. In this case, solution of record 4, "Pending the process until patient turns up", is suggested whose edge weight is 4 when I assumed weight of each edge flow is equal to one.

The steps of the algorithm are listed below:

1. Find edge and vertex of the current exception
2. The weighted path of the current exception is calculated, starting from the vertex whose in-degree is zero.
3. Check for the current exception weighted path length, which matches to weighted edges of the exception vertex and its neighboring vertices in the knowledgebase.

Graph G=(V,E) can be represented either by Adjacency matrices or by Adjacency Lists.

Adjacency matrix representation uses an n × n matrix A of 0's and 1's given by

A(i,j) = { 1 (vi, vj) ∈ E
{ 0 otherwise

Where, (i,j)th element of the matrix is a 1 only if vi → vj is an edge in G

Pseudo code representation of Adjacency matrix appears below:

```
Public class GraphAsMatrix extends AbstractGraph
{       protected Edge[][] matrix;
        public GraphAsMatrix (int size)
        {
                super(size);
                matrix = new Edge [size][size];
        }       // ...
}
```

Adjacency Lists representation uses |V| linked lists one for each vertex.
The linked list for vertex vi ∈ V contains the elements of {w: (vi,w) ∈ A(vi) }, the set of nodes adjacent to vi. As a result the lists are called adjacency lists.

Pseudo code representation of Adjacency Lists appears below:

```
Public class GraphAsLists extends AbstractGraph
{       protected LinkedList[] adjacencyList;
        public GraphAsLists (int size)
        {
                super(size);
                adjacencyList=new LinkedList[size];
                for (int i=0;i<size;++i)
                adjacencyList [i] = new LinkedList();
        }       // ....
}
```

Pseudo code of the Algorithm is shown below:
//vj is the vertex where exception is identified
//Ej[] is the edges to the vertex vj starting from the vertex whose in-degree is zero
//W is the weighed edge
//n is the number of records in knowledge base for the current exception type
//We is the weighted path of the exception.

```
public abstract class AbstractGraph extends
AbstractContainer implements Graph
{       //...
        public void exception()
        {
        Vertex vj=v.getVertex();
        Enumeration Ej[]=e.getIncidentEdges(Vertex
vj)
        Int j=Ej.length();
        for (i=0;i<j;++i)
        {       P[j]=∑W(vi,v(i+1));     }
        for(n=1;n<n.length();++n)
        {       if(We[j]==P[j])
                { accuratematch(); }
                else
                {closestMatch(); }
        }//...
}
```

5. Preliminary Results

We have developed a prototype of the system based on the proposed architecture and exception handling scheme discussed in previous sections. The system is to be deployed by the healthcare professionals in their outpatient workflow. The healthcare professionals are provided with mobile devices to have real time access to existing clinical records and to interact with the other professionals. The workflow agent, which administrates the workflow of the patient, was developed in C# on an IPAQ 3800 series Pocket PC with 802.11b wireless network capabilities. The Server software was deployed on the Pentium based desktop running Windows XP and was developed in Microsoft Asp.net, Microsoft SQL Server 2000 and Microsoft SQL Notification Server. The Dispatcher was developed in XML Web Service. It listens to the client requests and invokes the agent to perform the client request. The Pocket PC software was developed with Microsoft Visual studio.NET. To optimize the user interface, user input was restricted to numeric data and the selection of text strings from previously-defined lists. The thin client framework was implemented and was demonstrated on a Pocket PC Emulator. The system has 6 modules, Reception, Administrator, Head Nurse, Nurse, Doctor, and Lab Technician Modules. The user interface was designed with reference to the demo presented in [19].

Figure 3 represents the user interface used by the Head Nurse to view the patient records, task assigned to the Nurse, task performed by the Nurse such as appointments etc. Figure 4 represents the user interface used by the Nurse to perform the tasks assigned by the Head Nurse and the Doctor for the patient. Figure 5 represents the user interface used by the Doctor to prescribe the drugs to the patient. Figure6 represents the user interface used by the Lab Technician to view the tests to be performed on the patient and provides the links to the tests (such as CBC, Chemistry, LTF etc) to be performed.

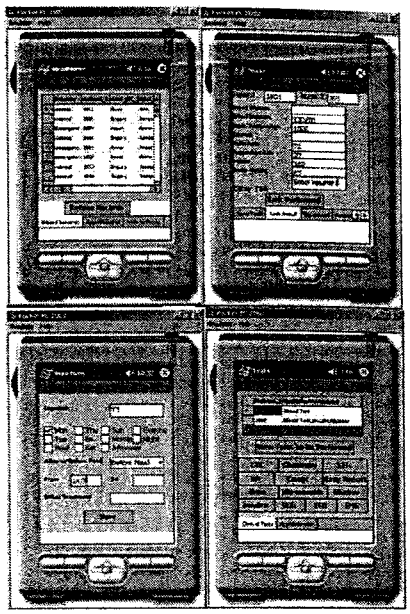

Fig 3, 4, 5 and 6: Some of the User Interface of the prototyped system

The system was then extended by incorporating the exception handling architecture and algorithms on top of the workflow. The exception handling architecture from [6] was incorporated over the proposed simulated system by extending its architecture to the proposed architecture of this paper. The proposed graph algorithm, which was developed in Java, scans the previous records and suggests the appropriate solution for the current exception. The degree of exception flow and dataflow between the similar exceptions that were lacking in the algorithms presented in [6] were considered.

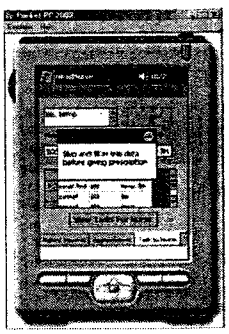

Fig 7: Solution for the current exception

Figure 7 demonstrates a typical exception handling solution. In this case, the graph algorithm suggests, "Skip and fill the data before giving prescription", when the exception "Patient cannot be found" occurs at the Head Nurse activity.

6. Conclusion

We have presented the architecture and an exception handling scheme over mobile handheld devices in the current workflow management systems. We have implemented the system with servers and thin clients in the form of handheld devices. Working prototyped has been demonstrated and preliminary results were presented. Our next step is to provide a comprehensive evaluation of the proposed algorithm in terms of accuracy and running time by comparing it with that of algorithms presented in [6]. The improved accuracy of our approach will be demonstrated by using computer-simulated system.

References

[1] Fabio Casati, and Giuseppe Pozzi, "Modeling exceptional behaviors in commercial workflow management systems," IFCIS International conference on cooperative information systems COOPIS'99, September 2-4, 1999.

[2] M.Reichert and P.Dadam, "ADEPT-Supporting Dynamic changes of workflows without Losing Control," Journal of Intelligent Information Systems, 10(2), 1998.

[3] A. Sheth, K.J.Kochut, "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems," Proc. Of NATO Advanced study Institute on Workflow Management Systems and Interoperability, Istanbul, Turkey, 1997.

[4] J.Eder and W.Liebhart, "The Workflow Activity Model WAMO", 3rd International Conference on Cooperative Information Systems (CoopIS'95), Wien, Austria, May 1995.

[5] Miller, J.A., Palaniswami, D., Sheth, A.P., Kochut, K.J., and Singh, H., "Webwork: METEOR2's Web-Based Workflow Management System", Journal of Intelligent Information Systems, 10(2), 1998.

[6] San-Yih Hwang, Sun-Fa Ho, and Jian Tang, "Mining Exception Instances to Facilitate Workflow Exception Handling", Database Systems for Advanced Applications, Proceedings of the Sixth International Conference on Database Systems for Advanced Applications (DASFAA), IEEE Computer Society, 1999, Page no: 45-52.

[7] D.K.W.Chiu, K.Karlapalem, Q.Li, "Exception Handling in ADOME Workflow Management System," Proc. Of 2'nd Int'l. Conf.on Co-operative information Systems, 1997.

[8] Alonso,G.,etal., "Failure Handling in Large Scale Workflow Management Systems,", Technical Report RJ9913, IBM Almaden Research Center, San Jose, CA, November 1994.

[9] H.Saastamoinen, "On the Handling of Exceptions in Information Systems". PhD thesis, University of Jyvaskyla, 1995.

[10] M. Klein et al., Proceedings of CSCW-98 Workshop towards Adaptive Workflow Systems, Seattle, WA, 1998.

[11] C.Hagen, G.Alonso, "Flexible Exception Handling in the OPERA Process Support System", 18th International Conference on Distributed Computing Systems (ICDCS), Amsterdam, Netherlands, May 1998.

[12] M. Klein and C. Dellarocas, "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems, Proceedings of CSCW-98 Workshop towards Adaptive Workflow Systems, Seattle, WA, 1998.

[13] Fabio Casati, MAriagrazia Fugini and Isabelle Mirbel, "An Environment for Designing Exceptions in Workflows", Information Systems, Vol.24, No.3, pp255-273 1999.

[14] Zongwei Luo, Amit Sheth, Krys Kochut, and John Miller, "Exception handling in workflow systems", Applied Intelligence: the International Journal of AI, Neural Networks and Complex Problem-Solving Technologies, Volume 13, Number 2, 2000, pp 125-147.

[15] Christodulos Pappas, Eva Coscia, Gabriella Dodero, Vittoria Gianuzzi and Mark Earney, "A Mobile E-health system based on Workflow Automation Tools", Proceedings of the 15th IEEE Symposium on Computer-Based Medical Systems(CBMS) , 2002

[16] Bruno R.Preiss,"Data Structures and Algorithms with object oriented design patterns in Java",John Wiley & Sons, Inc., 2000, Page no:538 to 597.

[17] Gabriel Valiente, "Algorithms on Trees and Graphs",Springer-Verlag Berlin Heidelberg New York, 2002, Page no: 3 to 34 and 255 to 288.

[18] DAOU Systems White Paper, "Going Mobile: From eHealth to mHealth", Page: 2,April 2001,http://whitepapers.zdnet.co.uk/0,39025945,60029049p-39000514q,00.htm

[19] http://www.wardinhand.org/demo/demo.htm