Design of a rule based system using Structured Query Language

Umair Abdullah
FUIEMS, Foundation University
Islamabad, Pakistan
umair_pitafi@yahoo.com

Mohammad Jamil Sawar
Barani Institute of Information Technology,
Rawalpindi, Pakistan
sawar@biit.edu.pk

Aftab Ahmed
FUIEMS, Foundation University
Islamabad, Pakistan
aftab_ff@hotmail.com

Abstract—This paper describes successful implementation of a Rule Based System at MTBC, for applying billing compliance rules on medical claims. Rule engine has been developed in Structured Query Language as stored procedures, which is one of the unique features of this rule based system. Implementing rule engine in SQL has provided two major benefits. Firstly, as operational data of the organization is in relational form, stored in Microsoft SQL Server database, therefore rule engine, using the native language, works at real time without any need of data transformation for working memory. Secondly due to SQL server, rule engine is using interpreted approach instead of compiled approach, which helps dynamic updating, editing and execution of rules. A rule is represented as a query stored in database, along with associated attributes like rule name, rule description and rule priority. ‘where’ clause of query contains condition and then part of the rule. A rule editor has been developed to facilitate domain users to edit rules in English like format, which is then translated to SQL statements. Editing of business logic has become very easy in MTBC billing software by using MTBC-RBS.

Keywords-component: Computing Methodologies, Artificial Intelligence, Knowledge Representation Formalisms and Methods, Representations (procedural and rule-based)

I. INTRODUCTION

Rule based systems evolved from preliminary work of Artificial Intelligence (AI) researchers, one of the oldest computer science technologies being used today. Similarly SQL a set based declarative language used in specialized domain of data base management systems, evolved from the business need of storing huge volumes of data and querying information from it. Combining these two domains of software technology has resulted in exciting, useful and effective outcome. Although rule based system being presented here has been developed specifically for medical billing domain, it can still be applied virtually to any real life domain.

Conventional rule based systems use human expert knowledge to solve real world problems. Expert knowledge is often represented in the form of rules or as data within the computer [7]. A typical rule base system consists of three parts, an interpreter (engine), a rule base, and a working memory [2]. Code of rule engine remains unchanged during operational use, while rule base expands. Working memory is used by the rules for interaction with each other and for validating their conditions. Code of rule engine is normally developed either by using specialized AI tools like LISP, Prolog, Official Production System (OPS-5), Knowledge Engineering Environment (KEE) or by using high level languages like C++, Java etc. During this research we have not found any rule engine developed using SQL except a simple example developed by Greenberg [8].

For almost four decades computer-based information systems have been developed and implemented in health care domain. Hospital information systems were designed primarily for administrative purposes to ensure that all charges were billed and collected correctly [9]. Today application of information technology in Healthcare domain has emerged as a new multi-million dollar industry. Huge investments are being made by many government and private organizations to produce health related tools. Similarly a lot of revenue is being generated by those IT organizations, which are related to healthcare. Top company of 2003 “GE Healthcare” founded in 2000 has annual revenue of $2.7 billion. And the 100th company “MED 3000 Inc.” (Founded in 1985) has $7.3 million annual revenue [3].

Medical billing is a complex, dynamic, knowledge oriented domain. As generally defined medical billing is the process of submitting and following up on claims to insurance companies in order to receive payment for services rendered by a healthcare provider. According to MDSERV handout about 30% of all medical claims are denied initially [16]. 35% of initially rejected claims are rejected again at least once. Up to 10% of rejected claims are never collected.

Many companies and organization like MediSoft [4], AlphaII [14] with their businesses related to medical claim processing have developed customized software for claim ‘scrubbing’ i.e. to verify charge information before a claim is sent to the payer and checking for any potential errors [5]. AthenaHealth [1] has developed a rule based engine for medical billing compliance of claims, where their experts are transforming medical billing knowledge in the form of rules. Another such organization is MTBC, a unique healthcare IT company, which is benefiting from data mining, machine learning and other AI techniques, in order to incorporate decision support technology in its business cycle to avoid claim rejections and thus increase service quality and customer satisfaction.

MTBC processes thousands of medical claims daily. Conventional software, built in-house, using Microsoft .Net technology, is being used for data entry and billing purpose. Many checks have been implemented in the software for data validation, in order to avoid possible rejection of claims by medical insurances. But with the passage of time new checks...
and conditions are required to be applied. Adding new billing compliance checks in data entry software introduces new bugs, thus consuming a number of man hours of development team. This is a less efficient and resource consuming way of satisfying the business needs. These hard coded billing compliance checks are not adequate to meet the data validation requirements, being refined and updated daily by medical insurances.

As needs drive actions, an efficient, flexible, and robust system has been developed using SQL, based on rule based technology. This system is now fully operational at MTBC.

II. ARCHITECTURE OF MTBC-RBS

Medical claims data from heterogeneous resources is imported to MTBC database. These data sources include MTBC website, company’s billing software, data files, direct import from clients’ database server, and sync server, shown in Fig 1. MTBC website is mostly used by the providers for making patient appointments and inserting patients’ demographic information. The term medical service provider or simply ‘provider’ refers to doctor, physician, and surgeon. Website is directly linked to the main database. Billing software, which is also linked directly to the database server of the company, is used by Billing Executives (BEs) to manually enter claims from scanned images into the database.

Data files are mostly in Health Level 7 (HL7) format. These files are sent by some doctors’ practices, especially those using third party Electronic Medical Record (EMR) software. In case of practice, newly enrolled with MTBC, data import is done from practice’s previous database server to MTBC’s database. This newly imported data mostly lacks data consistency. Custom built synchronization server transfers data from MTBC-EMR, being used by doctors, to MTBC database server via internet.

From MTBC database these medical claims are sent using Electronic Document Interchange (EDI) technology via internet, to insurances for reimbursement. All insurance payers have defined rules and guidelines for submitting clean claims. Only those claims are paid by the insurances which satisfy all the required conditions, rest are rejected by the insurance. All these billing compliance checks are missing when importing data from data files, sync server and other database server, so erroneous data get imported into the database which later result in claim rejections. Further, insurance payers frequently update their existing business rules and add new ones. So even in case of billing software and website it is difficult to keep these compliance checks up-to-date. By introducing MTBC-RBS we do not need to apply billing compliance checks separately on each data source. Rather, all these checks are applied after data is imported and before data is sent to insurances through EDI.

III. RULE BASE

Rule base comprise of three entities, meta-rules, rules and logical variables stored in Meta_Rules_Table, Rules_Table, and Logical_Variables_Table respectively.

A. Logical Variables

In rule queries, symbols within angled brackets <> are considered as logical variables. Rules are executed/ applied after replacing logical variables with their corresponding values. Value of a logical variable is obtained by executing its respective query. Some logical variables along with their queries are given below;

- PAT_GENDER:= select gender from patient where patient_account = @patient_account
- PAT_ACCOUNT:= select patient_account from claims where claim_no = @claim_no
- DOS (i.e. date of service) := select DOS from dbo.claims where claim_no = @claim_no
- DOB (i.e. date of birth of the patient) := select Date_Of_Birth from dbo.patient where patient_account = @patient_account
- ACCIDENT_DATE:= select accident_date from claims where claim_no = @claim_no
- DX_CODE (i.e. diagnosis code) := select dx_code from claims where claim_no = @claim_no
- CLAIM_NO:= select @claim_no as claim_no
- AGING := select datediff(day,DOS,Bill_date) as aging from claims where claim_no = @claim_no
- INS_ID (i.e. insurance payer id) := select i.InsPayer_Id from patient_insurance i where i.insurance_id = ins.insurance_id and i.patient_account = @patient_account

In above queries @claim_no is SQL variable, which holds id of the claim on which RBS is being applied. Similarly @patient_account is SQL variable, which holds id of the patient to whom current claim belongs.

These queries are executed dynamically, after passing values of SQL variables @claim_no and @patient_account, to get the value(s) of logical variables.

B. Rules

In a typical rule based system each rule consists of a conjunction of condition elements corresponding to the if-part of the rule, and a set of actions corresponding to then-part of the rule. The rule-actions can add, remove, or modify
working memory elements, or perform input-output [6].

In this engine, a rule is a condition portion of an SQL query statement. Rules are selected and applied in order of their priority. Priority of rules has been declared as positive integer value ranging from 1 to 100. Two priorities 25 and 75, (low 'must execute' priority and high 'must execute' priority respectively), have been defined. A rule having 'must execute' priority (25 or 75) is always tried/applied on the claim. Otherwise for a rule to apply on a claim its Meta-rule should be true.

Five different rules are given below;

**Rule Name:** R1 **Priority:** 75
**Description:** Patient Gender Missing
**Query:** where isnull (<PAT_GENDER>, '') = ''

Note that rule R1 and R2 does not have explicit then-part. So displaying their description as a message to user will serve as default then-part of these rules.

**Rule Name:** R3 **Priority:** 75
**Description:** Date of birth follows the date of service
**Query:** where convert(datetime,'<DOB>') > convert(datetime,'<DOS>') ; if @Recordcnt = 1 begin SELECT 'Date of birth follows the date of service <DOS>' where <DOB> follows the date of service <DOS> end

In above rule query, begin and end block after 'if @Recordcnt = 1' form then-part or action-part of the rule. Note that then-part of R3 is just a select statement containing logical variables of date of birth (i.e. <DOB>) and date of service (i.e. <DOS>). So after replacing these logical variables with their corresponding values, resultant text will be displayed as message to the user. This displaying of message is the then-part or action-part of the rule.

**Rule Name:** R4 **Priority:** 25
**Description:** Accident date (indicator) missing for ICD range 800 to 999
**Query:** where isnull (<ACCIDENT_DATE> , '1900-01-01') = '1900-01-01' and left(<DX_CODE>,3) between '800' and '999' ; if @Recordcnt = 1 begin update claims set Accident_Date = <DOS> end

Then part of R4 is an update statement which is setting <DOS> i.e. date of service as accident date for the given claim, as for the given range of ICDs (i.e. International Classification of Disease) accident date is required.

**Rule Name:** R5 **Priority:** 25
**Description:** Claim aging greater than 95 days limit of insurance payer
**Query:** where <AGING> > 95 and <INS_ID> in (5002471)

Similarly in rule R5, default action will be taken i.e. description will be displayed as message to user, if its condition is true (i.e. query returns true). Two things are done with every rule before dynamically executing it. Firstly, with every rule query a fixed string i.e. 'select @Recordcnt = count(*)' is attached at the beginning of rule query. Secondly, logical variables (symbols with angled brackets) are replaced with their respective values. For example rule query of R1 will become 'select @Recordcnt = count(*) where isnull ('Male', '') =''' for the patient having gender 'Male'.

Besides attaching the select portion to rule query, variable <PAT_GENDER> is replaced with its value. So in this case where part will be false (as 'Male' is not equal to blank) thus saving 0 in @Recordcnt variable. When a rule query returns true, variable @Recordcnt gets value 1 and the claim is flagged as faulty claim in rule application log. Descriptions of some of the rules currently active in MTBC-RBS. Action part of most of the rules is to display the given description as error message to user. Some critical rules like 1, 3, 6, and 9 block the claims i.e. do not allow submission to insurance.

<table>
<thead>
<tr>
<th>SR</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A claim paid by Medicare should not be billed to secondary Medicare as paid amount is greater</td>
</tr>
<tr>
<td>2</td>
<td>Accident date missig for ICD range 800 to 999</td>
</tr>
<tr>
<td>3</td>
<td>Add-on code used without its primary code</td>
</tr>
<tr>
<td>4</td>
<td>Claim aging exceeding next year limit of Medicare</td>
</tr>
<tr>
<td>5</td>
<td>Claim aging greater than 365 days limit of insurance paeyer</td>
</tr>
<tr>
<td>6</td>
<td>Date of birth follows the date of service</td>
</tr>
<tr>
<td>7</td>
<td>Diagnosis code inconsistent with patient age</td>
</tr>
<tr>
<td>8</td>
<td>Diagnosis code inconsistent with patient gender</td>
</tr>
<tr>
<td>9</td>
<td>Group name missing which is required by the payer</td>
</tr>
<tr>
<td>10</td>
<td>Medicare policy number suffix used for age under 65, not matching with patient age</td>
</tr>
</tbody>
</table>

Descriptions of some of the rules currently active in MTBC-RBS. Action part of most of the rules is to display the given description as error message to user. Some critical rules like 1, 3, 6, and 9 block the claims i.e. do not allow submission to insurance.

C. Meta Rules

Concept of meta rules has been used by many researchers, such as Bottioni et al. [1] used it in an APL rule based system for image interpretation. Uri J. Schild and Shai Herzog [15] used meta rules for adding meta knowledge in their rule based system of legal domain.

MTBC-RBS uses divide and conquer strategy, somewhat like indexing. Common conditions of rules can be identified and saved as a separate meta rule query. It first evaluates meta rule query, if it returns true then all the rules related to that meta rule are selected and applied in order of their priority, otherwise no rules are executed in that group reducing the search space of the rule base. For example rule R4 can be divided into a meta rule and a related simple rule. Checking of accident date missing can be defined as meta rule.

**Meta Rule Name:** MR1
**Description:** Accident date (indicator) missing
**Query:** where isnull (<ACCIDENT_DATE> , '1900-01-01') = '1900-01-01'
Rule Name: R4  
Meta Rule Name: MR1  
Priority: 35  
Description: Accident date (indicator) missing for ICD range 800 to 999  
Query:  
where left('<DX_CODE>',3) between '800' and '999'; if @Recordcnt = 1 begin update claims set Accident_Date = '<DOS>' where claim_no = <CLAIM_NO> end  

Modified R4 is given above. Condition of accident date missing can be utilized again in other rule by referring MR1 as meta rule in that rule. Similarly R5 can be divided to form meta rule as shown below;  
Meta Rule Name: MR2  
Description: Claim belongs to payer 5002471  
Query:  
where <INS_ID> in (5002471)  

Rule Name: R5  
Meta Rule Name: MR2  
Priority: 50  
Description: Claim aging greater than 95 days limit of insurance payer  
Query:  
where <AGING> > 95  

Now by utilizing MR2 we can define more rules for claim belonging to insurance payer 5002471.  

In rule 1 of TABLE 1 'claim is being billed to Medicaid' has been implemented as meta-rule and 'claim paid by Medicare and paid amount is greater than Medicaid limit' is implemented as rule. Designing of rules and meta-rules depend upon the expertise of the knowledge engineer (i.e. the person who is building rules from domain knowledge).

IV. RULE ENGINE DESIGN  
Input of MTBC-RBS is a claim no, which it needs to check for validation with respect to billing compliance rules and do scrubbing (small modifications, corrections and enhancements). The actual data of the medical claim consists of information about diagnosis also known as International Classification of Disease (ICD) or diagnosis code (DxCode), procedures / treatments also known as Current Procedure Terminology (CPT), patient demographic data like its first name, last name, date of birth, SSN etc. and some other information which is required by the insurances for making payment of the claim. This data is stored in relational database connected with the billing software of the company. Rule engine will pick this data from the concerned tables.  

Initial logic of rule engine is very simple i.e. take one claim at a time, keeping in view the rule priority and meta rules, apply all rules on it one by one, and execute then part of those rules for which rule condition is true. The above simple logic has been transformed into rule engine design shown in Fig. 2.  

Rule base consists of four permanent tables in the relational database, three for storing rules, and one log table for storing the results of rules when applied on claims. Main loop of rule engine is implemented as stored procedure named sp_ApplyRule. It applies all the rules on given claim one by one.  

There are three main steps when rules are applied on the claim. First step involves finding all suitable rules which should be applied. This step is implemented as a stored procedure in SQL named sp_find_applicable_rules. Second step implemented as sp_replace_LV_values, is to replace all logical variables present in rule with their values. Third step is the execution of rule i.e. checking the condition of rule engine, storing its result in rule application log table, if result is true then executing the query present in then part of the rule which will do the scrubbing function.

As stored procedures in SQL cannot input or output tables as argument so temporary tables #Queries and #AppRules are used for sharing of data between above stated stored procedures. These three steps are repeated when we want to execute a meta-rule, as shown in stored procedure sp_find_applicable_rules of Fig 2.  

At first sp_findApplicable_rules checks status of rules, only those rules are considered which have active status. Then rules with priority 25 and 75 (low must and high must) are selected for application. Finally those rules are selected for which meta-rules return true. All these selected rules are saved in temporary table of #AppRules (which will be washed automatically by SQL server when RBS is applied on next claim).  

Second phase starts after the selection of rules, which applies all rules on the given claim one by one. For each rule all the logical variables in rule query are replaced with their values specific to the given claim. If a logical variable has more than one value then copies of the rule query are generated. Thus number of copies of rule query equals to number of values of that logical variable, each copy having one value of that variable.  

Thus at the end multiple queries are produced from single rule query. Suppose a claim having id 1001 has date of service ‘02/18/2009’ has two diagnosis, 872.061 (i.e. ‘open wound ear drum uncomplicated’) and 998.30 (i.e. ‘disruption of wound unspecified’). So rule query of R4 after replacing logical variables, will produce two copies shown below;  

R4 Query Replica 1:  
select @Recordcnt = count(*) where left('872.061',3) between '800' and '999'; if @Recordcnt = 1 begin update claims set Accident_Date = '02/18/2009' where claim_no = 1001 end

R4 Query Replica 2:  
select @Recordcnt = count(*) where left('998.30',3) between '800' and '999'; if @Recordcnt = 1 begin update claims set Accident_Date = '02/18/2009' where claim_no = 1001 end

Both of the above queries will be stored in temporary table of #Queries, being ready for execution. Note that if every logical variable of a rule query has one value then result of sp_replace_LVs_values stored procedure will be one rule query.  

After the replacement of logical variables resultant rule
queries are executed independently. For each query if \( @\text{Recordcnt} \) get value 1 then claim is faulty and then-part (action part) of the rule query is executed. Else if \( @\text{Recordcnt} \) get value 0, it means given condition is clear. In both cases result is stored in rule_application_log table. Data of this log table is later used for reporting purpose and for measuring the performance of the rule engine.

V. RULE EDITOR

Rule editor is an interface for domain users to insert their knowledge into RBS in the form of rules. Window shown in Fig. 3 is the main window for construction of rules and meta-rules. It consists of three panes under a search box.

First panel is a list box containing atomic information (piece of data which cannot be further divided). Search text box on top is to facilitate searching of these atomic information pieces from the list box. Second panel is the main area where domain user will construct complex billing compliance logic by joining the atomic information segments selected from upper box, with the help of logical operators ‘and’, ‘or’, ‘not’. User can use parenthesis to set the precedence of conditions.

User can press ‘Generate Query’ button to view the resultant rule query generated by the rule editor on the basis of construct created by the user. User can edit this query shown in third panel, if he has knowledge of SQL programming.

User clicks ‘Next’ button for going to next window of setting rule attributes (name, description, priority, meta-rule etc.) and then ‘Finish’ button to finally save the rule.

VI. RESULTS AND DISCUSSION

Rule engine performance is real time i.e. it is being used in operation department of the company, where billing executives are entering medical claims into the database with the help of data entry software. It takes approximately 2.986 (± 0.8) second for a claim to apply 5320 rules on it.
Performing checks on database with the help of normal queries involve joining of tables, which reduce efficiency. A normal query for performing timely filing check which is being performed by R5 is given below;

```sql
SELECT @Recordcnt = COUNT(*)
FROM claims c, patient_insurance pis, insurances i
WHERE i.Inspayer_id = 5002471
AND c.Claim_No = @claim_no
AND c.Patient_Accout = pis.Patient_Accout
AND pis.Insurance_Id = i.Insurance_Id
AND datediff(day,DOS,Bill_date) > 365
AND isnull(c.Deleted,0) <> 1
```

This query involves joining of two tables while rule query i.e. `SELECT @Recordcnt = COUNT(*) WHERE <AGING> > 95 and <INS_ID> in (5002471)` performs same check without any table join.

Table II, shows number of claims and the number of errors in the claims billed by MTBC during 15 working days. Per day average count of billed claims is 4182 and per day average error count is 1289, which is not minimized. One reason for non minimized error count is the manual data entry process. Other reasons include absence of billing compliance checks on data import from HL7 files and synchronization process. MTBC-RBS is capturing these errors effectively thus helping to avoid claim rejections from insurances, and to achieve early reimbursement of claims.

**VII. CONCLUSION**

Merger of rule base system’s technique with database management systems technology have produced efficient, flexible, and powerful result. Efficiency gain is due to splitting of single query into pieces and thus avoiding table joins.

This type of rule engine is useful for domains involving frequent updating of knowledge. As shown by initial results, manual data entry process induces a lot of mistakes, and with huge volumes of data being processed on daily basis such a system is useful to minimize data consistence, knowledge oriented errors.

A new component ‘Auto Rule Generator’ is being added to MTBC-RBS, which is based on machine learning techniques [10]. This will help to identify new billing compliance rules by automatically analyzing the response patterns of images against the billed claims. Work done by Stefano Ceri et al. [13] can help in this direction. They had developed a ‘Rule Analyzer and Selector’ for producing a collection of production rules.

Policy-based management for federated healthcare systems has recently gained increasing attention due to strict privacy and disclosure rules [12]. MTBC-RBS can be improved by adding rules for policy enforcement.

**ACKNOWLEDGMENT**

Thanks to MTBC for providing excellent research environment and extensive data bank for rule development and testing, keeping in line with HIPAA compliance.

**TABLE II. DAILY ERROR COUNTS**

<table>
<thead>
<tr>
<th>Date</th>
<th>Claim Count</th>
<th>Error Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/31/2009</td>
<td>64551</td>
<td>4759</td>
</tr>
<tr>
<td>2/02/2009</td>
<td>62842</td>
<td>4268</td>
</tr>
<tr>
<td>2/03/2009</td>
<td>64198</td>
<td>2974</td>
</tr>
<tr>
<td>2/05/2009</td>
<td>64104</td>
<td>8673</td>
</tr>
<tr>
<td>2/06/2009</td>
<td>54697</td>
<td>4119</td>
</tr>
<tr>
<td>2/07/2009</td>
<td>64302</td>
<td>7724</td>
</tr>
<tr>
<td>2/09/2009</td>
<td>63178</td>
<td>5099</td>
</tr>
<tr>
<td>2/10/2009</td>
<td>54014</td>
<td>4114</td>
</tr>
<tr>
<td>2/11/2009</td>
<td>54993</td>
<td>2522</td>
</tr>
<tr>
<td>2/12/2009</td>
<td>65003</td>
<td>3680</td>
</tr>
</tbody>
</table>

Claim Count represents the modified number of claims billed on given date and Error Count represents the modified number of errors identified by RBS keeping in line with the confidentiality policy of the company. Note that one claim may have more than one error, all have been counted.

**REFERENCES**