CSAS: *Course Schedule Advisory Expert System*

By

Samer Hamam

A thesis submitted to the
Faculty of Natural and Applied Sciences (FNAS)
In partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

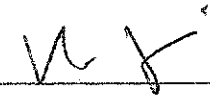
NOTRE DAME UNIVERSITY
FACULTY OF NATURAL AND APPLIED SCIENCES
DEPARMENT OF COMPUTER SCIENCE
Feb, 2015

CSAS: *Course Schedule Advisory Expert System*

By

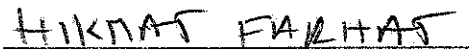
Samer Hamam

Approved:



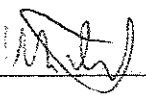
Dr. Khaldoun El-Khaldi: Associate Professor of Computer Science

Advisor



Dr. Hikmat Farhat: Associate Professor of Computer Science

Member of Committee



Dr. Khalil Challita: Assistant Professor of Computer Science

Member of Committee

Date of Thesis Defense: February 2015

ACKNOWLEDGEMENTS

My thesis owes its existence to the help, support and inspiration of several people. Firstly, I would like to gratefully and sincerely thank Dr. Khaldoun El-Khaldi for his guidance, understanding, patience, valuable and thorough supervision, and his friendship during all phases of this work. His support and inspiring suggestions have been precious for the development of the content of this thesis. This accomplishment would not have been possible without him.

I would also like to thank my committee members, Dr. Khalil Challita, Dr. Hikmat Farhat, and Dr. Khaldoun El-Khaldi for serving as my committee members. I also want to thank you for letting my defense be an enjoyable moment, and for your valuable comments and suggestions.

I would like to thank my wife Georgette for her support, encouragement, and quiet patience during my years of study for Master of Science degree in Computer Science.

Finally, I thank my parents, Fayez and Mary, for their faith in me and allowing me to be as ambitious as I wanted.

ABSTRACT

Keywords: *expert systems, knowledge acquisition and management, scheduling, web service, decision support system, academic advising*

Student academic advising is an educational decision support process required nowadays to be automated for both the student and advisor to speed up the planning process of course registration by finding out the optimal sequencing or prioritization of students degree requirements to enrol in for next semesters based on their academic standing in a way that suits students' interests and meets overall graduation requirements within a time frame. This thesis integrated all the rules and constraints from all precedent references relevant to this subject that impacts the scheduling process and proposed a hybrid of model-driven and knowledge-driven decision support system (DSS). Model-driven DSS is based on interoperability between JESS inference engine and Java engine whereas knowledge-driven DSS is based on discovery of new knowledge or patterns using data mining techniques. The solution is named ***Course Schedule Advisory Expert System*** (CSAS) implemented as a Web-based application as well as mobile application deployed on android smart device. CSAS analyses data in knowledge systems and allows students to seek quick responses to their queries regarding their plan of study and progress in the program. Technically, inference engine is a hybrid of rule-based engine using JESS (Java Expert System Shell) and knowledge-driven (data mining) based engine using RapidMiner-java libraries. JESS uses Rete algorithm which processes rules and facts in its working memory to generate feasible course registration plan for next semesters of the uncompleted requirements according to contract sheets of each major. JESS version 8 is a Java–based rule engine and scripting environment that supports android platform unlike precedent versions that supported only Java platform (NetBeans). JESS allows dynamic knowledge management in real time. Application retrieves student information from SQL Server knowledgebase database via web service application deployed on a communication server for further processing by computational model before rendering results in the GUI component of implemented application whether it's Web-based or mobile-app. The results of the developed prototype revealed that the model generated accurate results according to system specifications and implemented rules.

TABLE OF CONTENTS:

Chapter 1

Introduction

1.1 Introduction to the general problem

Academic institutions offer their students a variety of programs from which a student may choose his domain, register and satisfy requirements of degree in order to eventually obtain a degree [10]. The student has to satisfy all required courses which are categorized into four types: Core Requirements, General Requirements, Major Requirements and electives. Course registration is an integral aspect of student academic advising where students make decisions on the choice of courses to take in specific semesters based on their current academic standing. It's a course schedule planning process of assigning students to courses that satisfy their respective curricula [1]. Automating this process in a form of DSS tool is a need to both students and their advisors.

1.2 Problem definition

The complexity in course schedule planning process is due to the increased number of sequencing rules (e.g., prerequisites) that need to be satisfied by the student. Finding an automated solution is a need for both students and advisors because such complexity may guide the advisor while planning student's requirements (throughout his graduation period) to fall in mistakes prohibiting the student from timely graduation [1].

Such automated process is divided into two tasks, the first task is structured in retrieving the non-satisfied requirements and the available class schedule for a student and less structured when sequencing these requirements over all semesters of graduation period according to predefined rules and constraints. The sequencing of courses is a less structured and mechanical than the extraction of non-satisfied requirements [7].

1.2.1 Problem of student advising and course planning

Some of the shortcomings in the process of student advising and course planning:

- Student advising becomes an iterative task and a heavy burden on the academic faculty [10]
- Instability in requirements that change often which requires advisors interference to update them [10]

- The existing set of requirements is often inconsistent, which make its interpretation ambiguous and highly subjective especially when planning courses for next semesters
- Variation in degree requirements and types of course offering vary from one department to another (variation in facts by department)
- Rules are subjected to be changed rapidly by the faculty (particularly "last minute" changes) [10]
- Variety of exemptions (exempted courses),
- Substitution of courses: students who have taken certain sophisticated mathematics courses, for example, are sometimes exempted from college algebra and/or business calculus. Such decisions can only be made by university personnel [7]
- Equivalences: differences in transferable units of different universities,

1.2.1 Course planning challenges

Although, this process may encounter some challenges due to many factors that may cause mistakes during scheduling process such as the:

- Availability of classes: system cannot take into account when required courses will be offered. Certain courses are offered only during certain terms, such as only during the fall semester. When the DSS produces its prioritized eligible course listing, it assumes that all courses are immediately available [7].
- Course content issues such as case of the two courses having similar priority and similar role as prerequisites for higher courses [7]. A student may dislike taking related courses together in the same semester. For example, DSS may suggest to schedule both macroeconomics and microeconomics courses in same semester [7].
- Individual attributes of students such as case of a student having some weakness in a recommended course for registration by system and prefers postponing it to next semester

1.2.2 Need for interactive course planning DSS

Here comes the need for a system that should overcome the above mentioned problems and plans the student's remaining degree requirements and meet his preferences. System should store information about domain problem in a repository or knowledge base, processes data and generates results. For example, system has to find the remaining unsatisfied requirements, filter for eligible courses, apply the business rules and interactively suggests

alternatives, verify and accept the student's requests and schedule the desired courses [10]. System should have an interactive graphical user interface to display degree requirements of a student, recommended plans of study, generated schedules and allows planner to input his preferences or wishes as parameters which impact the output of system. For example, a student may plan to graduate within 3 or 4 years or inputs the number of credits per semester or postpone some courses for next year, etc... Such choices are allowed as long as his/her plans doesn't conflict with the following specifications:

- University and departmental regulations must be satisfied, for example: prerequisite must be taken before its consequent, number of courses taken must be within the bounds
- Individual preferences of student must be considered—e.g., total number of credits taken must be within the specified limits, preferences, etc…
- The recommended set of courses must be scheduled without any type of conflict

Such as system has the characteristics of a decision support system. Turban (1995) defines DSS as "an interactive, flexible, and adaptable computer-based information system, especially developed for supporting the solution of a non-structured management problem for improved decision making [3].

1.2.3 Need the best fit programming paradigm for problem solving

In general, the choice of a convenient programming paradigm is an important matter for the modelling of any problem in real world. If the course schedule problem is classified as a non-structured problem, then the developer may encounter some challenges during the implementation phase in case the adopted programming paradigm doesn't support the optimal approach to problem solving. The developer tries to compose the problem by writing code that describes in exacting detail the steps that the computer must take to accomplish the goal, but managing the problem of the increased number of rules and managing the priority of these rules impacts course sequencing process and may make it difficult for developer to model the problem by an explicit detailed algorithm or explicit sequence of code statements.

1.2.4 Need for automation

The automation of this complex process is a need for both students and their advisors in reducing the frustration on the part of students and providing timely and reliable course

registration schedule for each student to register at the beginning of a new semester. Also, it eases the burden on advisor instead of iterating again and again the same task which becomes a time consuming and monotonous process [6]. It removes the time consuming tasks associated with course registration and allows advisors to concentrate on more complex advising functions. Moreover, automating the ordering of courses minimizes the amount of time required to complete the degree by ensuring that prerequisites are completed prior to the time the student needs to enrol in a particular required course [7].

1.2.5 Need for dynamicity and extensibility

Due to the aspect of variation in changing requirements and the rules set by advisors, the proposed system should take into consideration the need to dynamically manage knowledge and rules and stay away from any form of static knowledge management because this costs continuous modification to the code of the program. So, the system should read the rules as inputs and all data relevant to the domain problem (knowledge base) from external resource and process them in real time. Our system adopts the methodology of dynamic knowledge management mentioned in thesis [2] where rules and facts are dynamically formed prior to being further processed by model base component. For example, the degree requirement rule is a form of dynamic rules that changes overtime. Courses may be substituted or their prerequisites may change with time. Also, system can be designed extensible by saving the rules modelled in the programming paradigm in knowledge base database.

1.2.6 Need for accuracy

The automation of course schedule planning process of the proposed system must guarantee to generate a feasible schedule plan for the problem of course planning by generating a proper sequencing of courses based on predefined rules. System should suggest timely course schedule planning within graduation period of a student, adhere to the rules that sequence the courses and schedule courses, overcome class time conflict, propose more than one schedule plan and extends the insight of advisor by analysing the impact of affinity between courses scheduled within same semester on performance of students based on historical data using data mining association algorithm.

1.3 Thesis organization

The next part of the thesis is described as follows: In section 2 we define expert system and its benefits in academic advising. Section 3 introduces knowledge discovery process and data mining techniques. Section 4 elaborates on related work and all the exerted effort in CSP problem. Section 5 details personnel work and contribution of my thesis. Section 6 reports a trial evaluation of the developed prototype. Section 7 combines conclusions and future work.

Chapter 2

Expert System

2.1 Definition of artificial intelligence

Artificial Intelligence is the field that seeks to "build systems that exhibit intelligent behaviour and perform complex tasks with a level of competence that is equivalent or superior to the level currently exhibited by human experts" [6].

2.2 Definition of an expert system

Expert system is one of the components of artificial intelligence developed with the objective of solving complex tasks by processing data stored in the knowledge base database thus improving efficiency and speed. Expert system captures the technical know-how of an expert into a knowledge-base which can then be analysed by computer systems to arrive at solutions thus freeing the expert to devote his/her time to more pertinent problems [6].

Edward Figenbaum [11], the father of expert systems, defines expert system as "an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution."

In academic advising, the faculty advisor suffers from doing the same repetitive tasks in course registration and student advising. It’s needed to free up his time to focus on more important matters such as choice of electives, career options, life career goals, etc…

The purpose of using an expert system is to provide decision support for advisor and not to replace the advisor. Applying expert system in academic advising domain requires capturing the knowledge of the advisor in the knowledge base component of an expert system. Expert systems are built for the purpose of handling complex problems such as course schedule planning problem.

2.2.1 Expert system components

An expert system typically includes two main components knowledge base and inference engine (model base). The knowledge base is the repository of pertinent knowledge related to the problem that the expert system purports to solve. The inference engine interprets the knowledge using algorithms and rules to provide solutions [6]. The third component of an expert system is its interactive graphical user interface (GUI).

The knowledge engineering and software development tasks involved in constructing software for an expert system are very different from the task of designing an imperative program. Some of the problems in the construction of a rule based expert system include deducing or inferring the heuristics of the expert, converting these heuristics into a working taxonomy and rule base, and ordering the rule base so that the system performs efficiently and correctly [12].

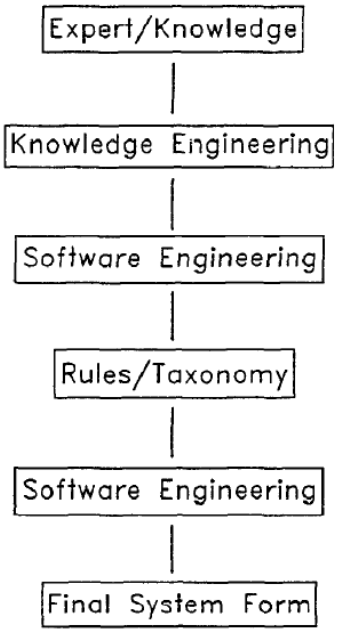


Figure1: Software Engineering Flowchart

2.2.1.1 Knowledge engineering phase

In general, knowledge engineering is the process of organizing the knowledge gathered by knowledge engineer based on a set of interviews done with domain experts. In academic advising domain, experts are the advisors of faculties. Knowledge engineer has to deduce from the expert work the heuristics and the data base which are transformed later into a rule base. The objective of knowledge engineer is to bridge the gap between interviews of domain expert and rule base which is not an easy task [12].

The problem of representing the gathered knowledge has been resolved by representing knowledge in a form of a pictorial tree called “k-tree” (knowledge tree). The form of “k-tree” carry all the information of the if-then rules and group rules that have common antecedents. Each branch of a “k-tree” corresponds to a rule, the leaf represents the consequent and parent node represents the antecedent. This “k-tree” notation is also used as a mean to document

gathered knowledge of the expert. The “k-tree” notation, helps also to lead the knowledge engineer, in a systematic way, to a complete exploration of the heuristics.

In our advising system, the antecedents include the student’s class, the semester, courses the student has taken previously, and so on. The consequents are often, but not always, course recommendations [12].

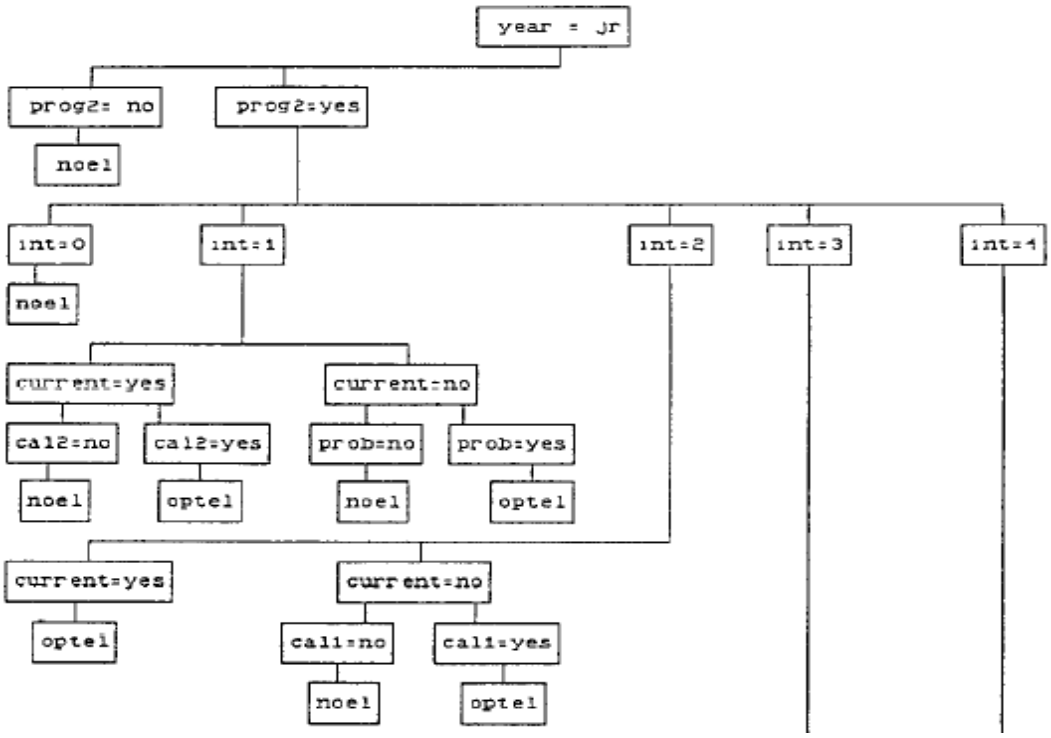


Figure 2: “k-tree” notation

2.2.1.2 Software engineering phase

After finalizing the process of representing the grouped heuristics and data base of an expert using “k-tree” notation, next objective of the knowledge engineer is to embody this domain knowledge in a software program. Before starting in programming phase of a rule based system, it’s required to construct the final taxonomy and construct the rule base. All antecedents of rules must appear in taxonomy. Construction of the rule base comes after completing the final taxonomy and this is done by grouping the “k-trees” by advice type which leads to logical grouping of rules. The “k-tree” notation helps in spotting repetitive rules and by creating new properties for similar property values of same nature or type of the existing ones in sub-“k-trees” to consolidate or merge sub-“k-trees”; consequently, this eases the contraction process of the k-tree and generates a more optimized non duplicated rule-based tree [12]. For example, the offered course in each sub “k-tree” is different; so, we can create a new property named “current” and then merge the sub “k-trees”. Similar to concept of inheritance in Object Oriented Programming where generalization requires maximizing the

commonalities and specialization requires maximizing differences. The commonalities are the similar property values in different sub “k-trees” are consolidated into a new property name. By accomplishing task of grouping the “k-trees”, the construction of taxonomy (classification) is finalized and the rules can be written down. Next step is to implement the rules using a declarative style programming language that support inference type such as LISP, PROLOG or JESS.

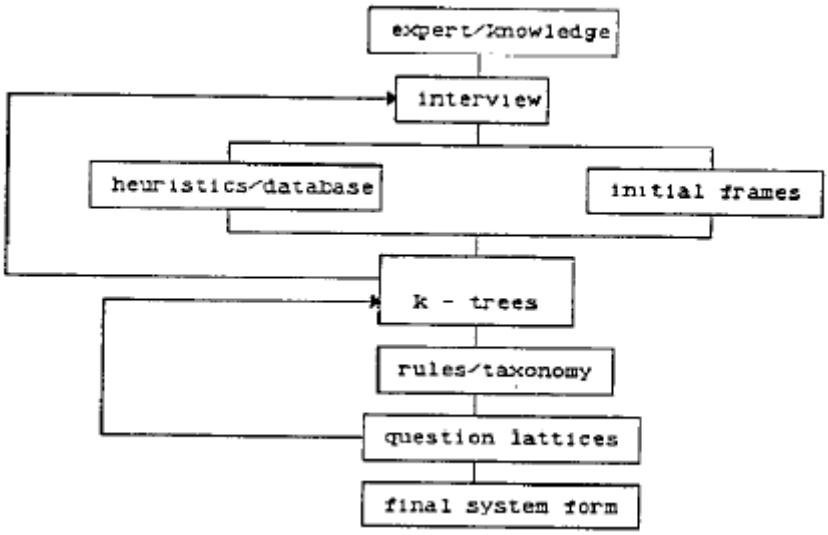


Figure 3: Knowledge Engineering Flowchart

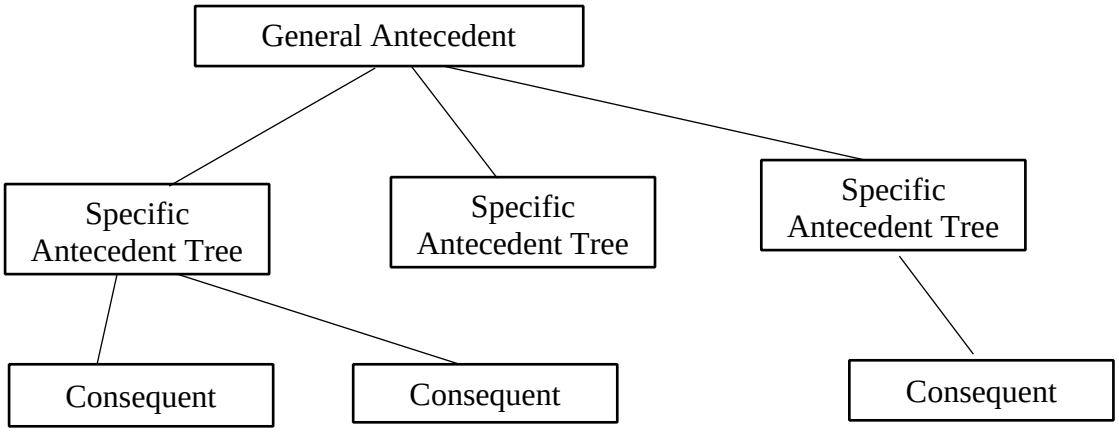
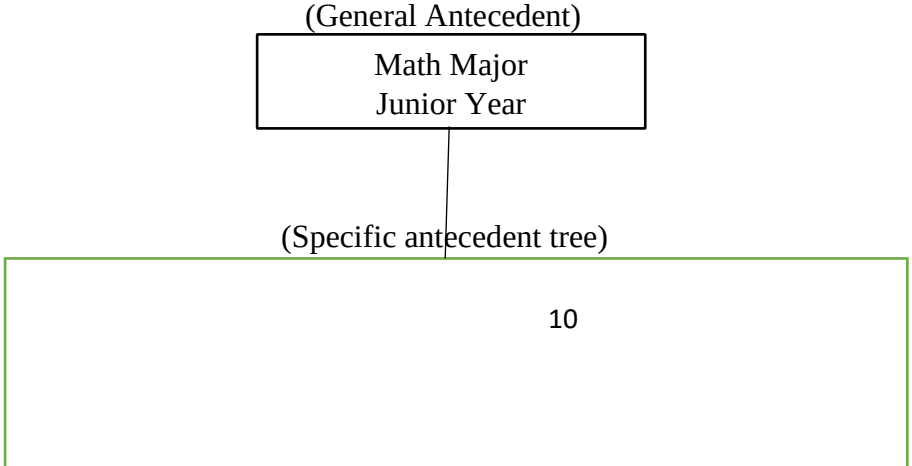


Figure 4: Antecedent/Consequent Relation



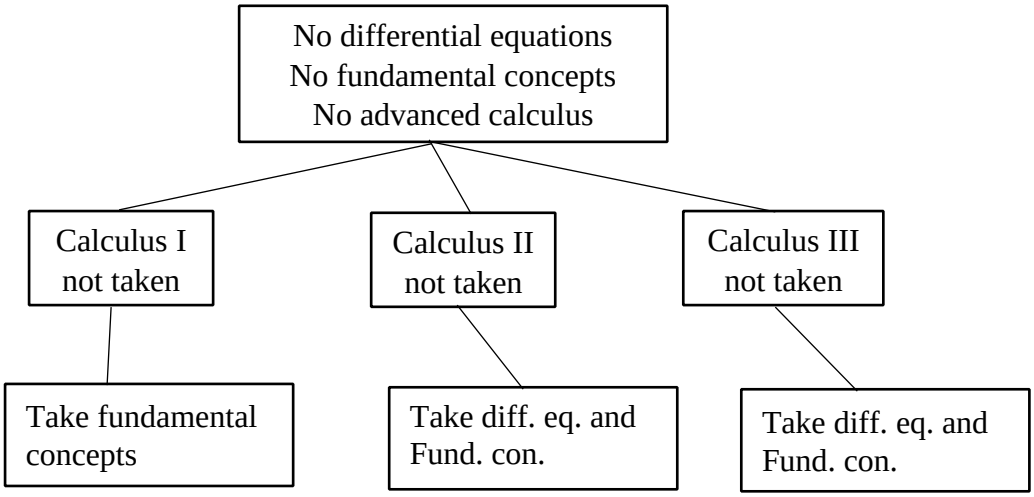


Figure 5: An Instance of Antecedent/Consequent Relation

2.3 Need for inference engine in academic advising

In academic advising, there exists complex problems that require inference aspect in order to be solved. For instance, the deepest layer rule in academic course schedule planning is a good example where we try to find the layer position at which a prerequisite stands in a chain of prerequisite courses. For any given course, inference engine checks if a course is a prerequisite to another course (of another layer). If yes, the process proceeds and again the deduced core-course becomes the prerequisite of another core-course in other layer and so forth. So, inference is a need to deduce at each level the course(s) of each prerequisite at its current layer with the minimum use of control flow statements. For structured problems in academic advising such as computing the remaining non-satisfied courses, imperative style is enough to state explicitly the sequence of steps of in a simple algorithm. For these reasons, we found that the domain of student advising and course schedule planning is amenable to benefit from features provided by both styles of programming paradigms: declarative style supported by inference type and imperative style. Inference feature exists in many declarative programming paradigms as mentioned before but not all (SQL is a declarative programming paradigm but doesn't have inference feature); so, our inference engine for academic advising could be a hybrid of imperative and declarative (with inference type feature) to tackle structured and less structured problems.

2.4 Programming Paradigms (algorithmic vs. declarative style)

Mainly, there are two styles of programming paradigms or approaches for problem solving: imperative **(**algorithmic programming) and declarative (functional programming). In algorithmic paradigm, the execution of statements or commands is ordered, detailed and

explicit using control flow statements (If, While, etc…) which control the flow of execution. The focus in imperative approach is in ***how*** to perform tasks (algorithms) and how to track changes in state. In declarative paradigm, the sequence of computation or the control flow is implicit and the focus is on ***what*** information is desired and what transformations are required. If the problem is structured then it's recommended to use the imperative style. If the problem is less or unstructured and it's difficult or ambiguous to specify a detailed and controlled sequence of execution, it's recommended to use declarative style where the developer states what the results should look like and not how to obtain it. In academic advising, there are lot of difficult and unstructured problems as well as structured problems. For example, paper [7] classifies extraction of eligible courses from knowledge base database as a structured problem by just retrieving from database by a simple query the courses marked as eligible whereas sequencing of courses is considered as a less structured problem. Paper [10] gives more examples about difficult problems which requires declarative style to solve them such as: the form of degree requirements (could be a course, training, etc…) and types of course offerings vary considerably from one department to another, last minute changes to rules occurs rapidly, variety of exemptions, substitutions, equivalences, rules of thumb, etc..

Certain tasks or functions can be solved using imperative style such as matching the requirements with courses taken so far. Scheduling of courses comes after sequencing them; scheduling a set of sequenced courses requires implementation in an efficient algorithmic style. Also, the deepest level rule, which we will elaborate more in a later chapter, is one of the rules that impacts the sequencing of courses and considered as a complex task due to challenges a developer may encounter during implementation in case of adopting the imperative style. Such task may require using too many nested control flow of execution statements (if/then or for/loop or while) because the knowledge structure of such task is like a tree view with too many branches to find the longest path. Modelling such task in imperative approach is a difficult for developer whereas applying the declarative approach such as the rule-based inference engine is much feasible to find out by deduction the deepest level of a course in a chain of prerequisites. In fact, implementation of some rules in course scheduling problem require imperative approach (rules pertinent to semester-scheduling of courses such as requirement type pattern rule) whereas other rules require declarative style (rules pertinent to sequencing of courses such as deepest level rule). So, it's important to select appropriate programming paradigm to implementation of scheduling and sequencing rules.

Example of imperative languages: C#, Visual Basic, C++ and Java whereas SQL, PROLOG, JESS are examples of declarative languages but SQL doesn't have rule-based engine Instead of telling explicitly the computer what steps to executed to achieve the goal, the computer (compiler) is working harder doing this by itself. Declarative style is an advantage because it abstracts the use of control flow statements.
In conclusion, we conclude that both styles of programming paradigms are necessary in order to model both structured and unstructured problems in academic advising; so, the model component of our DSS will be a hybrid of both approaches the declarative (rule-based engine) and imperative programming paradigms.

2.5 Expert system benefits

A rule based expert system supports extensibility and dynamic management of knowledge features in which rules and facts can be modified and extended in an external resource file before being loaded and processed by the rule-based inference engine of an expert system. This feature is an essential need in academic advising domain as we mentioned before the last minutes changes to rules by faculty. In imperative programming paradigm, rules are specified in terms of an explicit hard coded algorithm and any new changes to rules cost changes to source code of the program (intervention of programmer, testing changes, compilation, etc…). In addition, benefits of an expert system include increased efficiency, reduced costs, improved decision-making, and preservation of expert knowledge. Expert systems have been developed using LISt Processing (LISP) and PROgramming in LOGic (PROLOG) [6]. These languages are fairly simple and highly flexible. How-ever, the necessity of shortened processing times and the advent of increased computing power at lower costs have engendered a move to more recent programming languages such as Java. Java-based Expert System Shell (JESS) is a rule engine written in Java that facilitates easy inter-operability and data exchange between the rule engine and Java-based applications [6].

Chapter 3
Contribution of KDD in CSP Problem

This chapter presents the contribution of knowledge discovery in databases (KDD) process in course schedule planning problem.

3.1 Necessity of KDD

In dynamic databases, new transactions are appended as time advances. The data volume growth is in continuous increase in educational databases. Knowledge discovery can help by applying new techniques and tools that can intelligently and automatically transform the processed data into useful information knowledge.

3.2 KDD or data mining?

Many publications use the term Knowledge Discovery in Databases as a synonym for Data Mining. In fact, they are not the same. Paper [15] presents a formal definition of knowledge discovery process (KDP): “It is defined as the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.”
Among the definitions of Data Mining: “Data mining is one of the processes of Knowledge Discovery in Database (KDD) that is used for extracting information or pattern from large database.” [13] and “The non-trivial extraction of implicit, previously unknown and potentially useful information (such as rules, constraints and regularities) from data in databases.” [14].

Data mining is a particular step in KDP as shown in below figure. The previous steps in KDD such as understanding of problem domain, understanding of data and data preparation (selection and cleaning) are essential to ensure that knowledge is derived from the data.

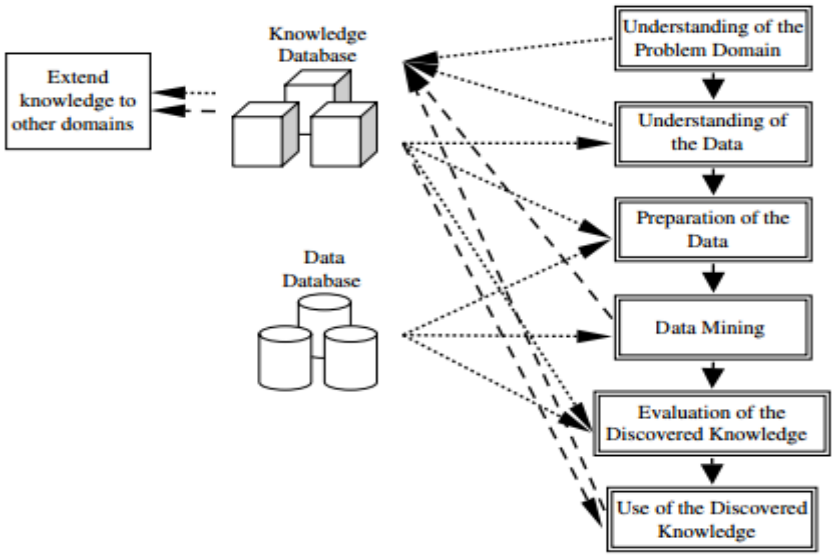


Figure 6: An Overview of the steps comprising KDD process

3.3 Data Mining Techniques

As of paper [14], several typical kinds of knowledge can be discovered by data miners, including association rules, characteristic rules, classification rules, discriminant rules, clustering, evolution, and deviation analysis. Among all these data mining techniques, we will focus on identifying association rule mining that discovers hidden knowledge in database.

3.3.1 Association rule mining

In fact many papers addressed in detail the algorithm of association rule such as [13] and [14]; so, we will focus on its specification and how we can employ such concept in course schedule planning problem.

In short, the association rule mining problem is to find out all the rules in the form of X => Y (antecedent => consequent) in a given number of transactions in a dynamic database, where X and Y $\subset$ I are sets of items, called itemsets. The association rule discovery algorithm is usually decomposed into 2 major steps. The first step is to find out all large itemsets that have support value exceed a minimum support threshold which are called also frequent itemsets and the second steps is find out all the association rules from the itemsets generated in first step that have value exceed a minimum confidence threshold [13]. For example, consider the occurrence of items A, B, C, D and E in four transactions in a database table:

Database D

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Scan D →

C_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

Scan D →

C_2

Itemset	Sup.
{A B}	1
{A C}	2
{A E}	1
{B C}	2
{B E}	3
{C E}	2

L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

C_3

Itemset
{B C E}

Scan D →

C_3

Itemset	Sup.
{B C E}	2

L_3

Itemset	Sup.
{B C E}	2

Figure 7: Support Count of All Elements of Itemset

All subsets of itemset ***I*** must exceed the minimum support threshold (usually 22% of total count of transactions). This means each subset set of ***I*** must have a minimum frequency of repetition 22% (for example).

Procedure of generating association rules from frequent itemsets is the following:

For every nonempty subset ***s*** of ***I***, the output rule **"s → (I - s)"**

if support_count(I)/support_count(s) >= min_conf

Suppose I = {B, C, E} and the minimum confidence threshold is **70%** and minimum support **40%**:

Set ***I*** satisfies the minimum support condition because it has support of 2 in 4 rows dataset: 2/4 = 0.5 => 50% (>= 40%).

Rule	**"s → (I - s)"**	**support_count(I)/support_count(s)**	**Confidence**	**Decision**
R1	B ^ C → E	2/2	100%	Accepted
R2	B ^ E → C	2/3	67%	Rejected
R3	E ^ C → B	2/2	100%	Accepted
R4	B → E ^ C	2/3	67%	Rejected
R5	E → B ^ C	2/3	67%	Rejected
R6	C → B ^ E	2/3	67%	Rejected

Only 2 rules (R1 and R3) out of 6 association rules satisfy minimum confidence condition. According to this dataset example:

- R1 accepted: when items B and C (antecedent) exist together then there's 100% (exceeds 70%) confidence that item E (consequent) will be also present
- R3 accepted: when items E and C exist together then there's 100% confidence (exceeds 70%) that item B will be also present

3.3.2 Benefits and application of association rule

Paper [17] provides an example of applying association rules in domain of market basket analysis in supermarkets. Consider a supermarket setting where the database records items purchased by a customer at a single time as a transaction. The planning department may be interested in finding "associations" between sets of items with some minimum specified confidence. Such association might be helpful in designing promotions and discounts or shelf organization and store layout. For example, a planner may be interested to:

- Know the confidence factor of transactions that purchased bread and butter also purchased milk.
- Find all rules that have "Diet Coke" as consequent. These rules may help plan what the store should do to boost the sale of Diet Coke.

- Find all rules that have “bagels” in the antecedent. These rules may help determine what products may be impacted if the store discontinues selling bagels.
- Find all the rules relating items located on shelves A and B in the store. These rules may help shelf planning by determining if the sale of items on shelf A is related to the sale of items on shelf B.

3.4 Advantage of employing association rules in course schedule planning

Each student is supposed to obtain a scheduled academic plan on his remaining unsatisfied courses. Advisor, as well as students, need to run the DSS to review the recommended courses by the system and advise the advisee accordingly. A further analysis can be performed by advisor on recommended set of courses for next semester by applying the association rule-mining technique on historical data of students who have taken within same semester all or a subset of these recommended courses. The objective of applying the association rules is to elicit which sub list of courses have a high affinity or high confidence ratio of being present together within same semester. This preliminary result will be employed in analysing the impact of high affinity between courses on the performance of students.

Chapter 4
Related Work

Traditional information systems that provide simple services like access to students' current term enrolment, the GPA, units earned, and some mentoring resources lack major important features to facilitate effective academic advising. Typically, such systems do not support the student academic decisions in the development of her total academic potential. Many decisions that must be taken by both the student and the advisors are not supported by these systems [5].

In this section, we present an overview of the related works pertinent to advising system precisely the educational decision support systems which are based on inference engine expert systems and data mining techniques that address the course scheduling problem. Such systems were designed to provide intelligent advice and a true decision support for students and advisors by processing specific student information and recommending for next semesters a course schedule plan for that particular student.

4.1 Introduction

A brief description on each of the most relevant papers in domain problem:

*Web-based Expert System for Class Schedule Planning (**CSP**) Using JESS* was reported in [2]. This is a web-based DSS based on JESS inference engine aims to help advisor and student to develop a course schedule plan while fitting individual's interests such as courses students may try to avoid or postpone their registration, time availability, concentration, and number of units per semester. CSP automatically checks the degree requirements and available class schedule dynamically to produce a set of schedules that fits the students' needs. Each schedule includes a detail class schedule for next semester and a multiple semester plan of classes that students should take to meet graduation requirements. The unique and new technical contribution of CSP system is that unlike most other expert systems that require static expert knowledge this expert system allows dynamic management of knowledge by administrator in real time using web interface. CSP supports two sets of features for its users: students who seek class scheduling advice, and administrators who perform knowledge management tasks.

*A hybrid model of an RBR-CBR Course Advisory Expert System (**CAES**)* was reported in [1]. This paper is another attempt that tries to provide an automated solution for the complex

problem of course schedule planning in academic advising. It introduces the case-based reasoning model (CBR) which is a concept of AI-problem solving that relies on knowledge gained from previous problem-solving episodes to resolve new problems once sufficient similarity between the current case (problem) and previously stored cases have been established. The inference engine of CAES is a hybrid engine of a rule-based reasoning (RBR implemented using JESS) and case-based reasoning (CBR).

A Decision Support System for Academic Advising was reported in [7]. This paper focuses on functional requirements of a DSS by proposing three hierarchical sequencing rules for courses at the cost of its implemented software architecture which can be a better one in terms of database and user interface. It presents design of the database table that stores student's degree requirements and status of each course and linked to its prerequisites. Also, DSS generates four types of reports.

A Simple Decision Support Tool for university academic advising was reported in [5]. This simple DSS is a spreadsheet-based decision support tool using VBA scripts for Microsoft Excel. It describes the advising process in Effat University as an example of undergraduate pre-registration advising process.

Expert System for Student Advising using JESS was reposted in [6]. A JESS expert system that separates the rules stored in XML file from the execution. Users are able to customize or extend the system by updating the XML file that stores the rules.

An *Academic DSS for Student, Course and Program Assessment* was reported in [8]. The paper describes capabilities of a performance based academic decision support system (PADSS) with some built-in data mining capabilities like analyses of students in 'graduated position' with respect to CGPA and ACT/TIU, analysis of the state of a course based on grade distribution, etc…. It allows the filtering and presentation of data in suitable forms and graphical representations (visualizations) more easily understandable by concerned advisors.

A *Knowledge-Driven Educational Decision Support System* (***EDSS***) was reported in [9]. This paper presents the challenging issues in the educational knowledge discovery process emerging from the flexibility of a semester credit system. It mentions five types of decision support systems such as data-driven, model-driven, document-driven, knowledge-driven and

communication-driven. This DSS is about providing support to decision of education managers about students who have a poor study performance should they stop or should they be given the chance to extend their studies one more semester. System architecture is a web-based knowledge-driven DSS with DBMS relational databases. System is a knowledge-driven DSS; it employs the various data mining techniques such as classification, clustering and association rules.

4.2 What is JESS?

JESS is a rule engine and scripting environment written entirely in Oracle's® Java™ language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. ***JESS*** is available at no cost for academic use and can be licensed for commercial use [4]. JESS is a tool for building expert system shell [2]. The expert knowledge and user inputs are stored as rules and facts in the JESS knowledge engine. JESS uses the Rete algorithm to match the rules and facts and generate new results. Java has been chosen to implement CSP mainly because JESS is implemented by Java and Java can interact directly with JESS which eases the interoperability process between JESS and Java. Also, Java can be easily ported to run on various operating systems although CSP was developed in a Windows environment.

4.2.1 JESS Facts and Rules

An expert system relies on a set of facts and rules to make decisions. In CSP, facts and rules are formed dynamically according to user request and current program information. Facts in CSP include the detail schedule data for the next semester, the course information, and the user-defined parameters. In CSP, All the facts are dynamically generated from the XML fact data either stored in the data files, loaded in real-time from school website, or entered by users. For example, CSC 201 starts from 5:20 to 8:20 pm on Monday only. CSP only requires the day, time, and the course name to generate feasible schedule. Other information, such as location and instructor, is for display purpose only. The JESS fact for this class would be:

```
(assert (opencourse (name "CSC 201") (section "1") (call_num "41453") (seat 0) (day "M")
(start 1730) (end 2020) (location "RVR 1008") (instructor "Zhang C"))
```

Rules in JESS are a list of actions waiting to be triggered when the required condition(s) are satisfied. Because CSP allows an administrator to change the degree requirements from an

interface, the degree requirement rules must be generated dynamically to reflect the update. For example, students must take all the Core area courses. The following is an example of the Core requirement rule:

```
(defrule satisfy_core
        (declare (salience 950))
        (course_taken (name "CSC 201"))
        (course_taken (name "CSC 204"))
        (course_taken (name "CSC 205"))
        (course_taken (name "CSC 206"))
        (course_taken (name "CSC 209"))
        (not (core_passed yes))
=>
        (assert (core_passed yes))
```

Dynamic generation of above rule means that the XML translator translates above rule (satisfy_core) to a JESS rule by constructing dynamically the conditional part of the rule by appending fact condition “course_taken” to each course read from XML facts source file. “course_taken” fact verifies if the slot value "CSC 201" exists in working memory of JESS or in fact “course_taken”. Although, I may suggest an alternative way to write this rule in a static form using JESS function *accumulate* function which counts the number of non-completed courses in a fact named **Course** containing two slot values (*CourseNo* and *CourseStatus*).

Note: CourseType = 2 (core course), CourseStatus = 1 (course completed)

```
(defrule satisfy_core
  ?CountUnSatisfied <- (accumulate (bind ?count 0) (bind ?count (+ ?count 1)) ?count
                          (Course (CourseType 2)
                                  (CourseStatus ?CourseStatus &:(neq ?CourseStatus 1)) )
                       )
=>
        (if  (= ?CountUnSatisfied 0) then
                    (assert (core_passed yes))
        )
```

4.2.2 Prototyping with JESS

JESS provides a command line interface (CLI) and application programming Interface (API). The CLI is an interactive command console program and it actually calls the CLI batch function for every interaction. The API typically requires a lot of hard coding and is not so suitable to create dynamic facts and rules. However, the batch function of the JESS API is most efficient for loading and defining the rules and facts in a program. The batch function of the JESS API allows the facts and rules to be saved in a file which can be updated dynamically. As a simple prototype, we save nine facts and one rule in a fact-rule data file in plain text format.

The following is a sample output of the simple JESS prototype. The first three lines state the results of the rule indicating the three classes that has passed the prerequisite rule. The rest of the output is the verbose JESS output specifying the nine facts from the data file [2].

```
You can take csc275
You can take csc258
You can take csc255
f-0 (MAIN::course (name csc255) (prereq csc175))
f-1 (MAIN::course (name csc258) (prereq csc175))
f-2 (MAIN::course (name csc275) (prereq csc175))
f-3 (MAIN::course (name csc175) (prereq none))
f-4 (MAIN::taken_course (name csc175))
f-5 (MAIN::taken_course (name csc205))
f-6 (MAIN::taking_course (name csc255))
f-7 (MAIN::taking_course (name csc258))
f-8 (MAIN::taking_course
```

4.3 Architecture

In general, CSP is composed of two components: (1) framework implemented with a subset of Java technologies and (2) JESS engine. The framework provides JESS facts and rules for the JESS engine which acts as the core intelligence to generate the schedule results.

CSP architecture includes four main server components:

• **Apache Web Server**: listens for web page requests.

• **Tomcat Servlet Engine**: serves dynamically generated web page using JSP and Servlet technology. The java server page (JSP) that contains HTML design code and business logic in java is converted to a servlet class in real time. An instance of servlet class is executed by Tomcat Servlet engine.

• **CSP Business logic**: processes the schedule request and translates the request into JESS language.

• **JESS Engine**: processes the incoming JESS language and return the expert advising result. Tomcat Servlet Engine, CSP Business Logic, and JESS Engine are running under the same JVM (Java Virtual Machine).

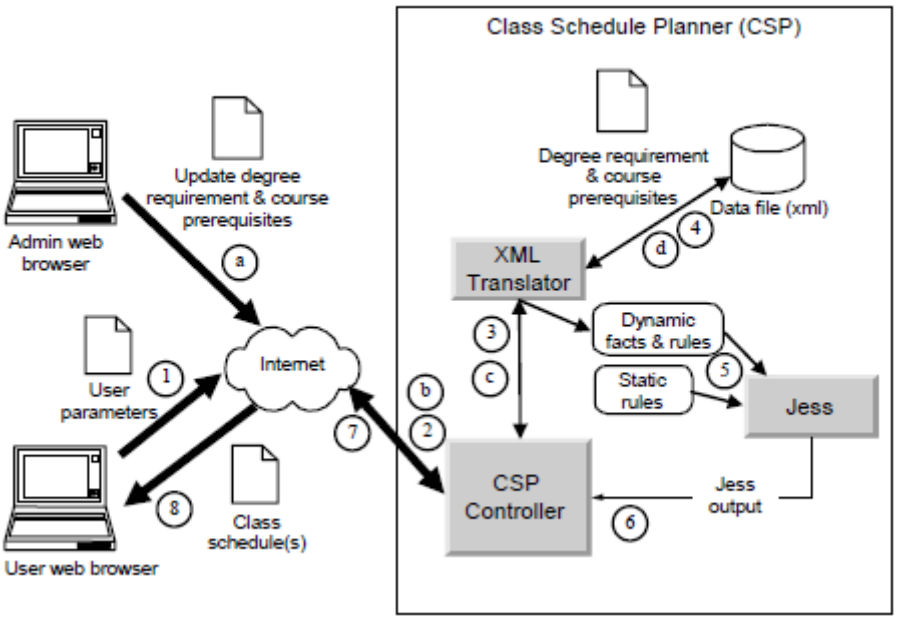


Figure 8: CSP system data flow

CAES is a 3-tier web-based architecture:

1. **Presentation layer**: web browser in presentation layer which communicates with middle later through HTTP protocol.
2. **Middle layer**: Webserver, CBR and RBR engines in middle layer. Middle layer communicates with data layer through JDBC (Java Data Base Connectivity) protocol.
    a. RBR engine: is a JESS rule-based engine
    b. CBR engine: case-based reasoning engine that contains stored cases of pervious advice
    c. Webserver: handles communication with the external environment and routes external calls to appropriate components (GUI or web browser application and database components). Implemented webserver is Apache Tomcat. Tomcat implements Java Servlet and the JSP specifications, providing an environment for Java code to run in cooperation with a web server. Tomcat includes its own

internal HTTP server. Communication with the data tier is through the JDBC (Java Data Base Connectivity) protocol.

3. **Data layer**: There are two databases in data layer, the educational relational database which is maintained by the University's database administrator stores information for each student and details of course registration information of the university, the knowledge base database which stores the facts (prepared data) and rules transferred to inference engines for further processing to provide advice in later stages. The knowledge base which is a component of the expert system is maintained by the knowledge engineer who models the rules as used by the human course adviser in advising student.

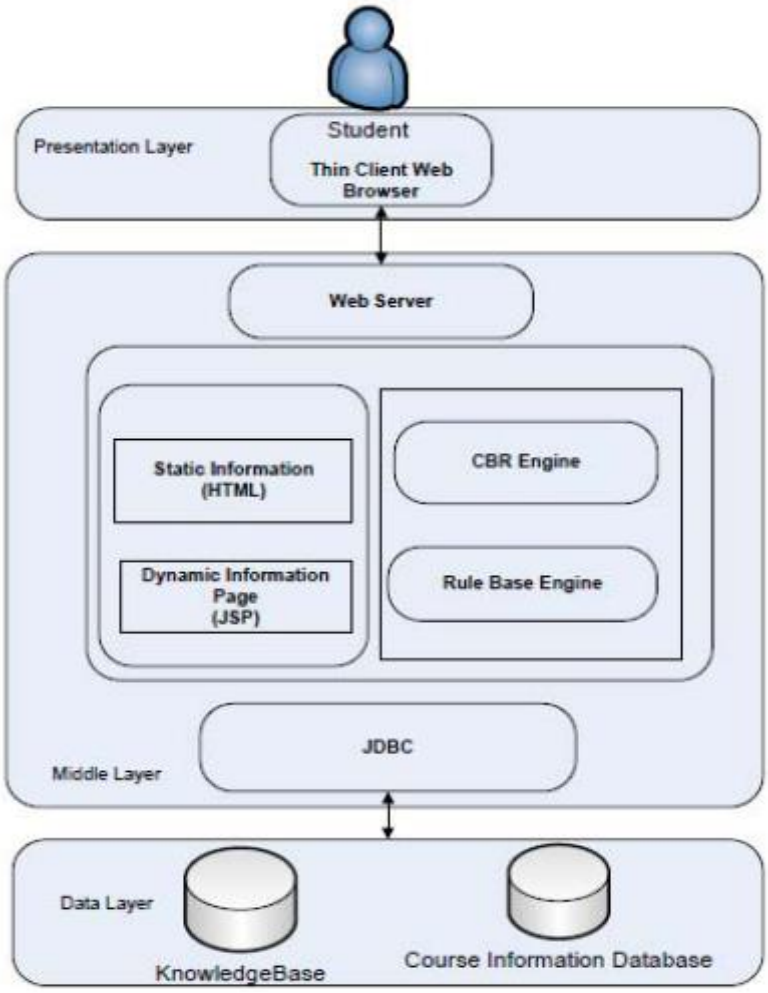


Figure 9: The Three-tire Architecture of EDSS

Architecture of the proposed knowledge-driven EDSS in paper [9] is based on a typical three-tier component-based architecture:

- Presentation tier: The presentation tier is a web interface. It is developed with Java technologies.
- Business logic tier:
- Storage tier:

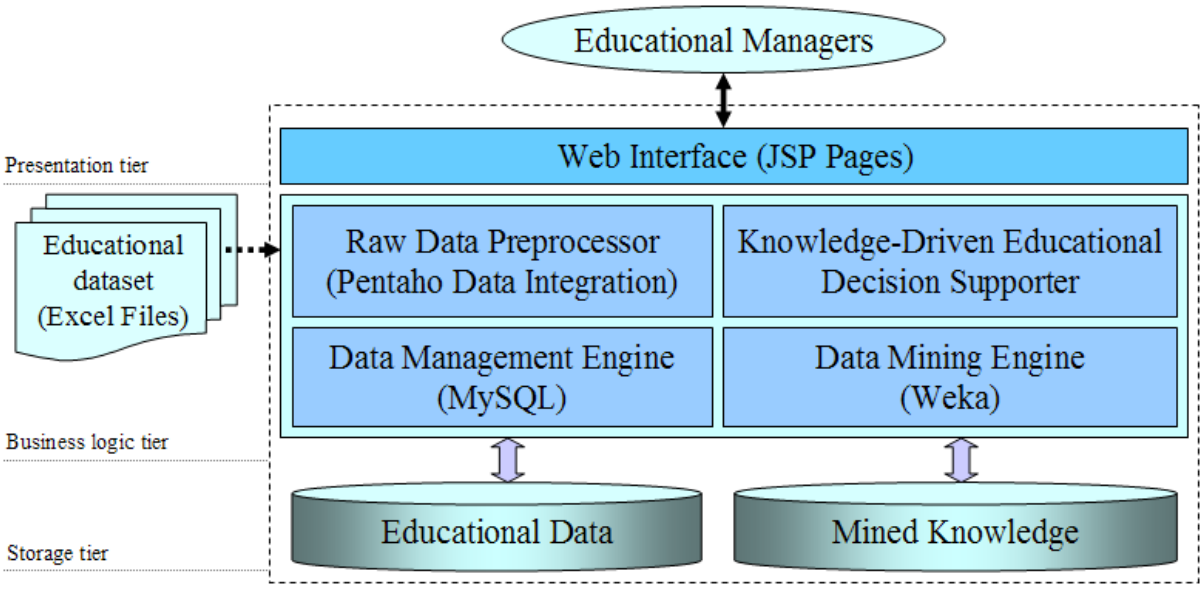


Figure 10: Architecture of the proposed knowledge-driven educational DSS

4.4 Functional Requirements

CSP in paper [2] implements three rules:

1. **Degree Requirement Rule**: enforce the requirements that the students must fulfil to graduate.
2. **Class Time Conflict Rule**: ensures course schedule does not overlap
3. **Enrolment Unit Rule**: ensures the enrolment unit does not exceed the user-defined input. In other words. It specifies the maximum number of credits allowed to schedule per semester.

The last two rules are static but the degree requirement rules can change overtime.

Paper [7] describes important functions which are possible to automate:

1. Identify unsatisfied courses required for a certain business degree by reconciling student's records (transcript) with university bulletin

2. Discard from the unsatisfied list of courses generated in previous function the courses that didn't meet their prerequisites (after evaluating the prerequisite rules). The student is not yet eligible to register courses that have unsatisfied prerequisites.
3. Prioritize or sequence list of eligible courses based on their importance in meeting prerequisites for remaining courses. For example, the highest priority would go to courses which serve as prerequisites for the greatest number of subsequent courses. Three hierarchical rules has been applied and fired in order to determine optimized and effective course sequence (although not always optimal):
    a. **Deepest Layer Rule**: has the highest priority and is used to determine the position of a course in a chain of prerequisites. Courses on the deepest level of prerequisites should be taken first. For example, if the accounting major has a maximum of five layers of courses, the first course(s) that should be taken is/are positioned on the fifth level. Each layer stands for one semester. For example, for a student majoring in computer science at NDU, course CSC212 (Program Design & Data Abstraction I) must be taken before CSC213 (Program Design & Data Abstraction II), which must be taken before CSC311 (Theory of Computation), which must be taken before CSC431 (Compiler Design). If a student must graduate within a period of 3 years (6 semesters), the student must begin taking courses in this chain starting third semester (CSC212 is the first course of deepest level 4).
    b. **Maximum Dependency Rule**: this rule is used for sorting eligible courses within a given layer i.e. eligible courses on the same layer should be taken in order of the number of prerequisites served. This requires that each course be analysed to see how many other higher-layer courses depend upon it as a prerequisite.
    c. **Lower Course Number Rule**: this rule prioritizes courses on the same layer and with the same number of dependencies in ascending order of course number but within same requirement type (Major or Core or General Requirement). The logic is that lower numbered courses are intended for less advanced students. For example, MAT215 is given before MAT224 and both of them have the same deepest layer and maximum dependency level.

Paper [5] describes rules that impacts sequence of scheduled process of courses such as:

1. Maximum credits per semester,

2. Course offering in two scenarios (considering instructor is available), either the number of pre-registered students exceeds 10 or the course is compulsory for graduation or a 'critical core course' (course that is a prerequisite for other important course(s)).
3. Failed or dropped course is an incomplete course included in the registration plan for next semester

4.5 Description of the Course Advisory inference mechanism

The implementation of CSP consists of the following steps:

1. Gathers data from various data sources: user parameter data, degree requirement update data, and real-time school web site data.
2. Translates the gathered data into one format (XML) for easy processing.
3. Translates the facts from XML format to JESS facts.
4. Interacts with JESS to process the facts and rules to produce the proper results.
5. Converts the JESS results into a class schedule and present the result.

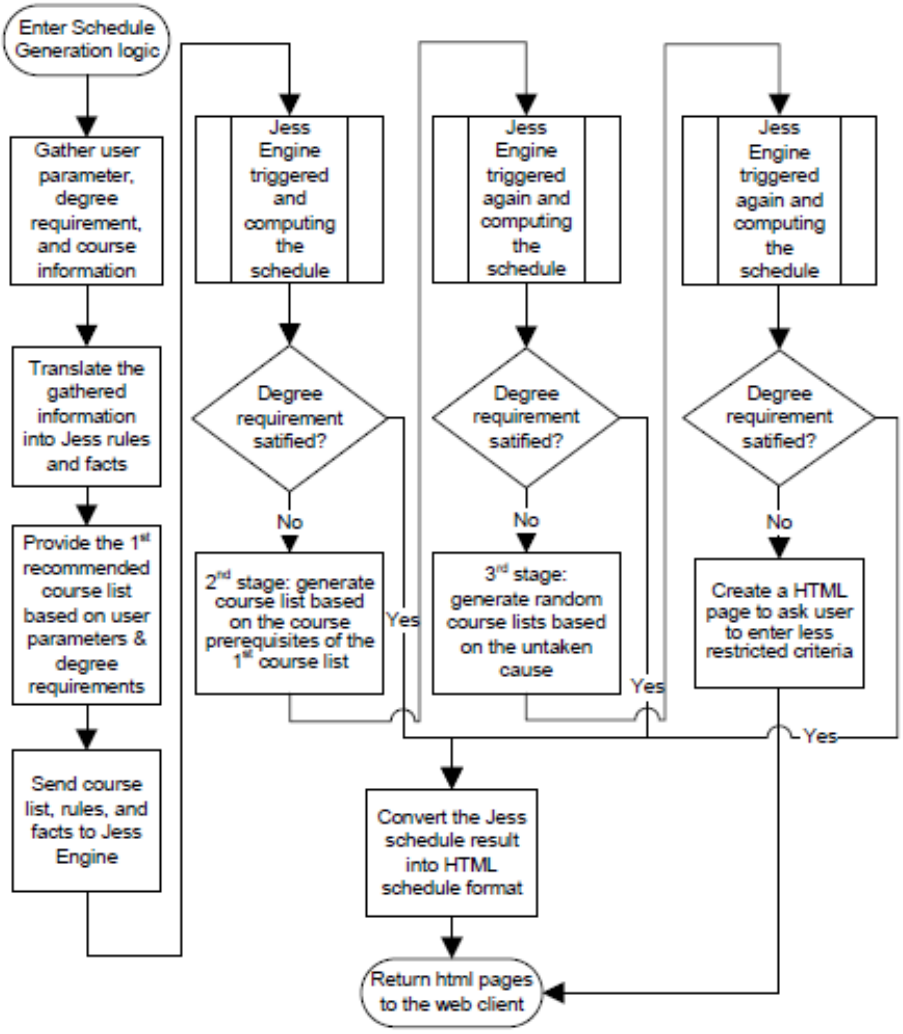


Figure 11: Multiple Stages Activity Flowchart

In paper [1] CAES Inference mechanism performs a similar case matching and if found, it retrieves the matched case; otherwise, the rule base inference engine will handle the computation process and stores the recommended solution as a new case in the case base reasoning engine.

The following formula computes the similarity score which compares the similarity of new case with each old case i.e. it counts for each case the common courses taken so far and the different and generate similarity ratio. The highest similarity score is chosen if it exceeds a certain threshold which means the candidate of highest score is selected for adaptation in recommending courses for next semester for the student of the new case.

$$\text{Similarity (NC, OC)} = \frac{\text{common}}{\text{common} + \text{different}}$$

NC = new case, OC = old case, common = matching common courses, different = courses not found in old case

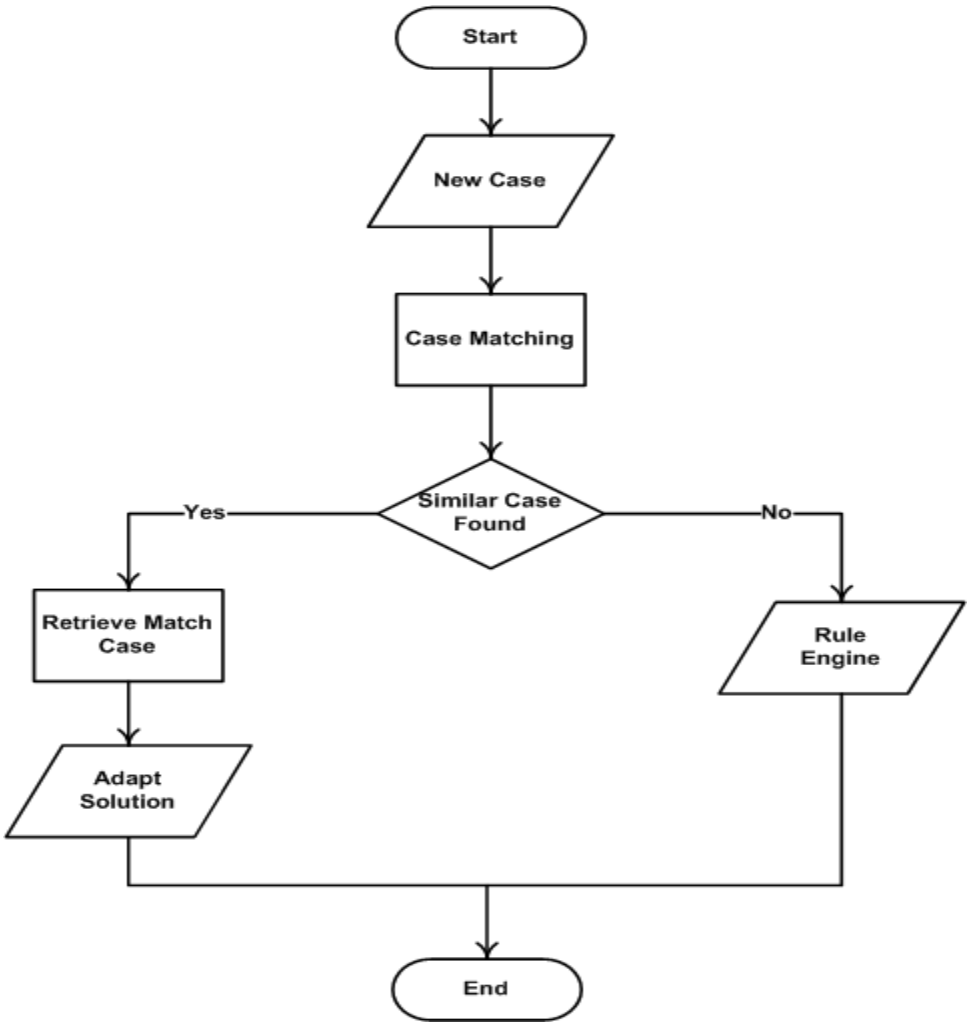


Figure 12: Schematic representation of CAES recommendation process

EDSS in paper [9] has four data mining functionalities to discover actionable knowledge from educational data imported from Excel files. They are: final status-based student classification using J48 algorithm; study trend prediction with clustering using k-means algorithm; status prediction with clustering in terms of probability using Expectation-Maximization algorithm; and course association analysis using Apriori algorithm. It is worth noting that these functionalities are mainly devoted to decision making support for problems about students with poor study performance and doesn't target our domain problem (course schedule planning). A short description on kind of discovered actionable knowledge by the employed data mining algorithms:

J48 algorithm is used for classifying students based on status (building a classifier to classify all students into the seven groups such as “complete”, “incomplete”, “drop-out”, “first warning”, “second warning”, “extended”).

K-means algorithm is a clustering algorithm used for study trend prediction to decide on whether to extend or not a student with poor performance. Applying a clustering algorithm to derive the study result of a student in the past who is the most similar to the student being considered in terms of accumulated knowledge and credits from courses they took.

Course Association Analysis using Apriori algorithm: in course registration, course association analysis has been applied on course registrations of all students who have the most similar characteristics to the student being considered to find out which groups of courses the students never pass when studying them together. From the resulting groups of courses, the course registration of the student being considered should not consist of any group of those courses in its entirety; however, maybe part of some group.

4.6 Database component

Paper [7] describes by field name the design of course table in a database as follows:

1. Course-no (department name + course number like Mat1302)
2. Course-description: description of course
3. Status (indicates course status: complete, failed, dropped, incomplete, to-be-repeated course of grade D)
4. Multiple prerequisite fields (of domain course-no) for each record or course
    a. PreReq1, PreReq2, etc…
5. Eligible: this field is used by the DSS in extracting courses to recommend to the student (if eligible = 1 => all the prerequisites are taken, if eligible = 0 then there are still incomplete prerequisites).

CSP in paper [2] uses XML template files are used as a knowledge base data sources which stores fact data. Also, user parameters are considered as fact data. The role of XML translator in CSP is to translate XML data into JESS facts which are then inserted into JESS engine (working memory) by CSP controller.

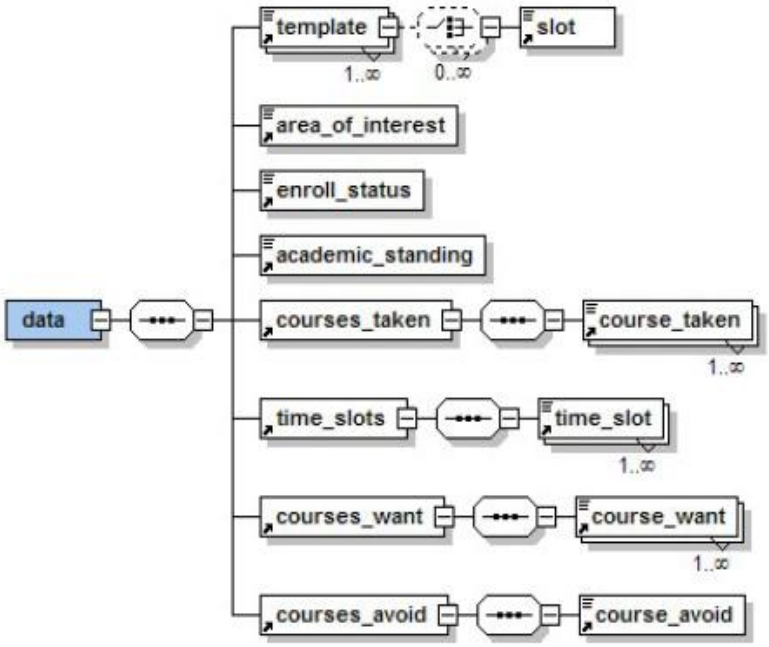


Figure 13: User Parameter XML Template in W3C Schema Format

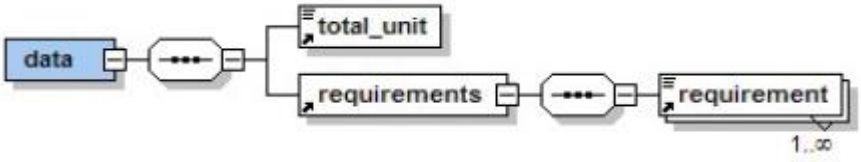


Figure 14: Degree Parameter XML Template in W3C Schema Format

4.7 Modelling component

In Paper [7] the modelling component of the DSS selects by retrieving from knowledge base database the eligible courses for the student. Then, it suggests an effective and efficient sequence for completing these courses. Modelling component is supposed to handle the computation of eligible courses first as the case in traditional DSSs, but this DSS does only the sequencing of courses. This makes its modelling function of the modelling component in DSS strongly tied to knowledge base. The computation of eligible field of a course is handled at database level which makes the processing function built into structure of data itself. In traditional decision support systems, the modelling component handles both tasks that is the computation and sequencing of eligible courses.

4.8 Conclusion

I recapped my observations on the exerted effort pertinent to our domain problem as the following:

- CSP in paper [2] is the only system that provides interfaces for administrative knowledge management by changing the rules and facts without requiring source code changes and for Student users to input their personalized parameters from a web interface. Also, the design and implementation of CSP is a successful integration of the following key elements:
  1. Thorough understanding of needs for student academic planning and knowledge of academic advising
  2. Tools: JESS, Java, JSP, Servlet, XML and web technology
  3. Unique methodology that supports dynamic knowledge management

  The implemented technology in tools CSP using same web tools, Java and Jess with dynamic knowledge management are considered in my personal work.

- In CAES [1], the change in requirements in a credit system with time may be considered as a drawback in CBR although we can step over this issue partially by mapping previous courses with new only in case they are related. Also, it suffers from cold start up problem for new students who don't have yet any historical data to be used for matching in CBR.

- The presented functional requirements in DSS [7] are worthy to consider in my personal work. The software architecture of this DSS is not much appropriate and a better system architecture can be made using relational DBMS database, inference engine, better user interface, integration of knowledge base with student information system (SIS) to facilitate getting current status of student courses.

- We can benefit from the application of course association analysis on course registrations in EDSS of paper [9] in our domain problem (course schedule planning) by applying the same data mining technique on the scheduled courses for next semesters but for different purpose than the one mentioned in EDSS. The course association analysis in our proposed system is meant to provide improved decision support for faculty advisors by finding the percentile of any indicated average grade value (by advisor) of the courses that have high affinity or confidence ratio of being mostly present together within same semester (based on historical data).

Chapter 5
Course Schedule Advisory Expert System (CSAS)

5.1 Introduction

Based on the current position of the problem elicited from the previous work, we propose a DSS named Course Schedule Advisory Expert System (CSAS) that adopts many of the previous presented functional aspects and implemented technologies in problem domain and suggests a new algorithm for course schedule planning problem with an implemented prototype on web server.

CSAS consists of the following components with description on specification of each:

- Knowledge base database: the back end component that stores the rules and prepared data extracted originally from university educational database (student information system). Knowledge base include current information about the academic status of a student (dropped or failed courses, completed courses, untaken courses). Establishing a synchronous connection between the educational database (SIS) and the mined knowledge base is needed to ensure data consistency property. Any change in degree requirements by university knowledge administrator or in student status has to be synchronized with knowledge base database.
- Rule-based inference engine processes business rules and facts (extracted from knowledge base) and outputs a non-sequenced list of eligible courses for next semester that becomes an input to Java engine. Each eligible course in the course list is concatenated with additional information like the course status, deepest level, maximum dependency, course number, requirement type, etc… to form a unique composed key used when performing a bubble sort operation on the list in order to sequence and prioritize the courses before starting the selection or scheduling process from the course list.
- Java engine (algorithm) sorts the course list and selects and schedule courses for next semester based on scheduling rules such as enrolment unit rule or predefined pattern by semester rule for allowed type of courses (for example: 2 core courses, 2 general courses, 1 major course). After scheduling next semester courses, call again inference engine to regenerate the course list to add new eligible non-scheduled courses to the list. The process of interaction between Java and JESS engine iterates until scheduling or allocating all courses to all semesters throughout graduation period.

- Knowledge-driven inference engine which is considered as a heuristic approach that processes historical data of students who had taken some or all of the courses recommended by CSAS within one semester and tries to finds new patterns or relationships between scheduled courses within same semester by applying data mining techniques. Then the students who had taken courses of high affinity to one another (minimum confidence percentage) are categorized based on a certain threshold which is the average of courses for each student. This kind of classification of students constitutes an indicator for advisor to infer the performance of students in case of scheduling these courses within same semester whether it may lead to cause a low grade average.
- An interactive graphical user interface (GUI) is needed to allow users to perform interactive tasks.

The following figure illustrates the steps of automatic course scheduling process:

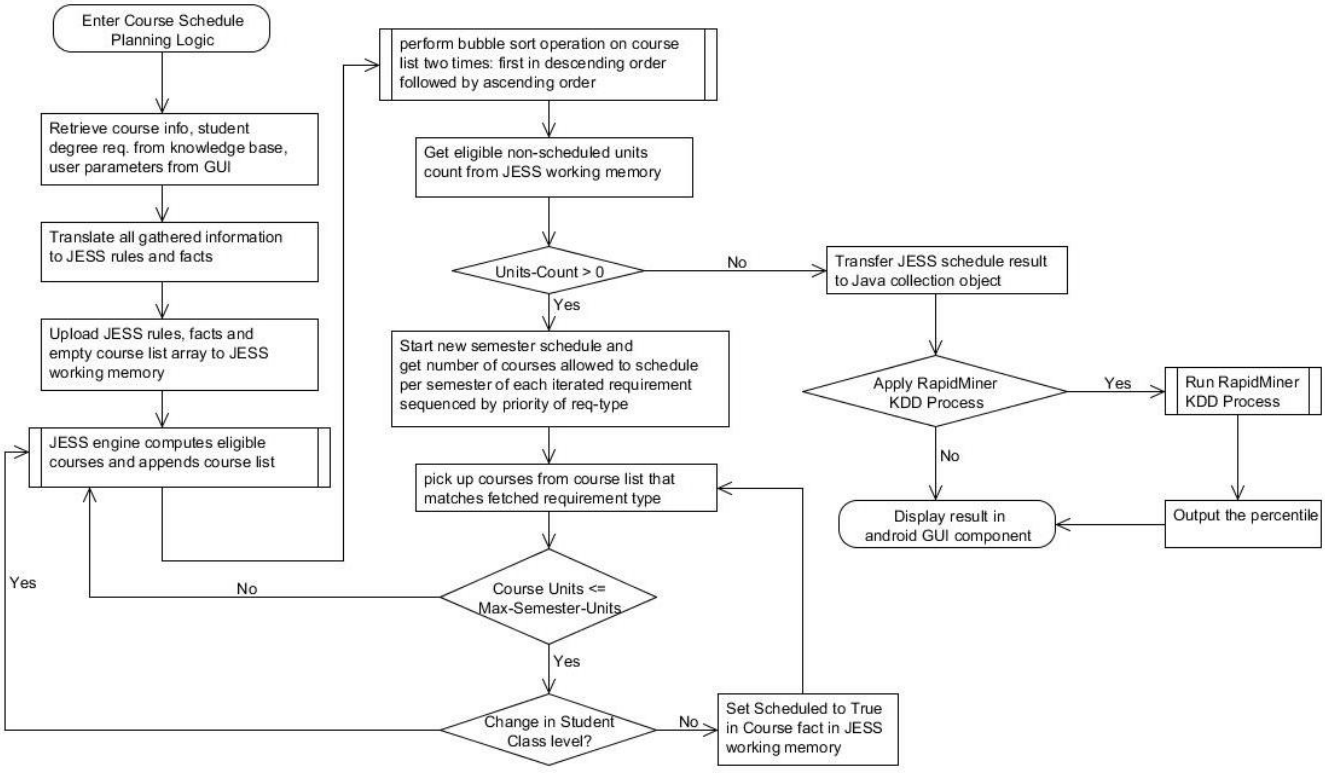


Figure 15: Automatic Course Scheduling Process Flowchart

5.2 Detailed description of CSAS implementation

5.2.1 Classification of courses

The process starts by retrieving all the degree requirements and rules from knowledge base and uploading them to working memory of JESS. The status of each degree requirement or course may be either completed, unfinished, dropped, failed. The unfinished courses (not taken yet) are divided into two sets: eligible and ineligible courses. Eligible courses are the courses eligible to be scheduled next semester whereas ineligible courses are the courses that didn't satisfy yet their preconditions like for example an unsatisfied prerequisite.

5.2.2 Type of rules

Static and dynamic rules are applied on facts when running JESS inference engine. The three hierarchical rules of sequencing courses mentioned in paper [7]: Deepest Layer Rule, Maximum Dependency Rule, Lower Course Number Rule and enrolment unit rule are adopted in CSAS. These rules are considered as static rules because they will not be subjected to any modifications later on as other dynamic rules such as the pattern of scheduling courses by requirement type (requirement type pattern rule). This rule, for example, is stored in form of a data-table in knowledge base database. It specifies the preferences of requirement types on one another and determines how many courses of each requirement type to schedule each semester. This rule is subjected to be modified by system administrator through a GUI. For example, the priority goes first for remedial requirements (RR), followed by core requirements (CR), etc…

Another example of dynamic rule is the class level of the student (sophomore, junior or senior) which determines the ineligible courses of the current class level. Each class level has a set of courses a student cannot enrol in until he/she completes a minimum number of credits or change his class level. For example, a computer science sophomore student is not eligible to take CSC480 and CSC490 at NDU (eligible only at senior level). So, inference engine sorts out eligible courses according to student current class level.

5.2.3 JESS inference engine facts

Every fact has a *template*. A fact gets its *name* and its list of *slots* from its template. Therefore a template is something like a Java class. It's a class of JESS facts. The deftemplate construct is the most general and most powerful way to create a template. This is its syntax in JESS language:

```
(deftemplate template-name
   ["Documentation comment"]
   [(declare  (slot-specific TRUE | FALSE)
            (backchain-reactive TRUE | FALSE)
            (from-class class name)
            (include-variables TRUE | FALSE)
            (ordered TRUE | FALSE))]
   [extends template-name]
    (slot | multislot slot-name
       [(type ANY | INTEGER | FLOAT |
             NUMBER | SYMBOL | STRING |
             LEXEME | OBJECT | LONG)]
       [(default default value)]
       [(default-dynamic expression)])*)
```

The following templates are pertinent to CSP problem and created in JESS working memory from java engine by creating new java objects of type Deftemplate. The light green background color in below tables means that the slots are initially populated with data from knowledge base database and other slots are updated later upon running by JESS engine:

Template structure of ***StudentClass*** Fact:

Slot	Multi-Slot	Slot Name	Description	Data Type	Domain	Example
Y	N	Class	Class code	Integer	1-5	1 = Sophomore
N	Y	ExcludedCourses	List of ineligible courses for current class level	String	A-Z	CSC480, CSC490
Y	N	FromUnits	Class level from units limit	Integer	1-500	60
Y	N	ToUnits	Class level to units limit	Integer	1-500	90

Template structure of ***StudentInfo*** Fact:

Slot	Multi-Slot	Slot Name	Description	Data Type	Domain	Example
Y	N	CurrentClass	active Class code	Number	1-5	1 = Sophomore
Y	N	StudentStatus	Status of student whether regular or on probation	Number	0-1	0 = regular 1= on-probation
N	Y	CompletedCourses	List of completed course with grade different than D	String	A-Z0-9, A-Z0-9, etc..	

Template structure of ***Course*** Fact:

Slot	Multi-Slot	Slot Name	Description	Data Type	Domain	Example
Y	N	StudentId	Id of student	Integer	1..∞	10
Y	N	TypeReqId	Requirement type id	Integer	1-5	1 = major req.

Y	N	TypeReq	Requirement type	String	A-Z	MR
Y	N	CoreReq	Course number	Integer	A-Z0-9	CSC311
Y	N	CourseDescrip	Course Description	String	A-Z	THEORY OF COMPUTATION
Y	N	Preference	A preference to course(s) to be scheduled before other courses within a requirement type of same layer	Integer	0-100	1
Y	N	CreditCarrying	Course is credit carrying or not	Integer	0-1	1
N	Y	Temp_PreReqList	List of temporary usage of prerequisites	String	A-Z1-9, A-Z1-9, etc,,	
Y	N	CourseCount_In_PreReqList	Count of courses in PreReqList	Integer	1-100	
N	Y	PreReqList	List of prerequisites	String	A-Z1-9, A-Z1-9, etc…	CSC213, MAT211
Y	N	Status	Course status	Integer	0-6	2 = not-taken
Y	N	Grade	Course grade	String	A-F	A
Y	N	TakeAllPreReq	Take all prerequisites or any	Integer	0-1	1 = take all 0 = take any
Y	N	EligibleClassCourse	Eligible courses after discarding ineligible courses as of student class level	Integer	0-1	1 = eligible 0 = ineligible
Y	N	Eligible	Course is eligible for scheduling	Integer	0-1	1
Y	N	Units	Course credits no	Integer	0-15	3
Y	N	Available	Course is available or not by faculty	Integer	0-1	1
Y	N	CanEnroll	Student can enrol in eligible course	Integer	0-1	1
Y	N	DeepestLevel	Deepest layer level of a course	Integer	0-20	2
Y	N	MaxDependency	Number of courses that depends on current course	Integer	0-20	3
Y	N	LowerCourseNo	Course number	Integer	0-9	311
Y	N	Scheduled	Course is scheduled or not	Integer	0-1	1
Y	N	SchedSemester	In which semester number course scheduled	Integer	0-20	3
Y	N	Class	Indicates student class level	Integer	1-3	1 = sophomore

Y	N	PreSchedCourses	Indicates how many prerequisites are scheduled so far for current course in a chain of prerequisites	Integer	1-8	3
N	Y	PrevSchedCoursesList1	Lists prerequisites of CoreReq	String	A-Z0-9, A-Z0-9, etc…	Prereq of CSC325 ***is*** CSC313
N	Y	PrevSchedCoursesList2	Lists prerequisites of courses in **PrevSchedCourses List1**	String	A-Z0-9, A-Z0-9, etc…	Prereq of CSC313 ***is*** CSC213
N	Y	PrevSchedCoursesList3	Lists prerequisites of courses in **PrevSchedCourses List2**	String	A-Z0-9, A-Z0-9, etc…	Prereq of CSC213 ***is*** CSC212
N	Y	PrevSchedCoursesList4	Lists prerequisites of courses in **PrevSchedCourses List3**	String	A-Z0-9, A-Z0-9, etc…	
N	Y	PrevSchedCoursesList5	Lists prerequisites of courses in **PrevSchedCourses List4**	String	A-Z0-9, A-Z0-9, etc…	
N	Y	PrevSchedCoursesList6	Lists prerequisites of courses in **PrevSchedCourses List5**	String	A-Z0-9, A-Z0-9, etc…	
N	Y	PrevSchedCoursesList7	Lists prerequisites of courses in **PrevSchedCourses List6**	String	A-Z0-9, A-Z0-9, etc…	
N	Y	PrevSchedCoursesList8	Lists prerequisites of courses in **PrevSchedCourses List7**	String	A-Z0-9, A-Z0-9, etc…	
N	Y	DeepestLayerList1	Same as **CoreReq**	String	A-Z0-9, A-Z0-9, etc…	CSC212
N	Y	DeepestLayerList2	List courses that have their prerequisites in **DeepestLayerList1**	String	A-Z0-9, A-Z0-9, etc…	CSC213, CSC218
N	Y	DeepestLayerList3	List courses that have their prerequisites in **DeepestLayerList2**	String	A-Z0-9, A-Z0-9, A-Z0-9	CSC311, CSC313, CSC316, CSC387, CSC423, CSC426, CSC425, CSC432

N	Y	DeepestLayerList4	List courses that have their prerequisites in **DeepestLayerList3**	String	A-Z0-9, A-Z0-9, A-Z0-9	
N	Y	DeepestLayerList5	List courses that have their prerequisites in **DeepestLayerList4**	String	A-Z0-9, A-Z0-9, A-Z0-9	
N	Y	DeepestLayerList6	List courses that have their prerequisites in **DeepestLayerList5**	String	A-Z0-9, A-Z0-9, A-Z0-9	
N	Y	DeepestLayerList7	List courses that have their prerequisites in **DeepestLayerList6**	String	A-Z0-9, A-Z0-9, A-Z0-9	
N	Y	DeepestLayerList8	List courses that have their prerequisites in **DeepestLayerList7**	String	A-Z0-9, A-Z0-9, A-Z0-9	

5.2.4 JESS inference engine rules

JESS inference engine processes rules according to their salience or priority. The computation process in inference engine is confined in determining the eligible courses for next semester during the scheduling process of all unsatisfied courses throughout the graduation period. Also, Jess inference engine computes the following attributes for each eligible course, concatenates them into one key value, and adds it to the course list array.

- Status
- Preference
- PreSchedCourses
- DeepestLevel
- MaxDependency
- LowerCourseNo

The JESS inference engine doesn't perform any course scheduling. The process starts by firing the following set of rules in the following sequence:

- **Rule I (**"EligibleCoursesAsOfClass"): the class level of the student rule refines the unsatisfied list of courses by discarding tentatively ineligible courses according to his current class level. In fact, there are two kinds of ineligible courses, the ones that are associated to student class level and the ones that are ineligible to be taken or

scheduled next semester because their prerequisites are not satisfied yet. Perquisites become satisfied either when they are taken or scheduled during the automatic scheduling process of all unsatisfied courses throughout the graduation period.

- **Rule II** ("EligibleCourse"): The consequent of rule-1 fires rule-2 which determines the eligible courses for scheduling that have no prerequisite courses at all.
- **Rule III** ("CanTakeNextCourse_After_Taking_All_Prerequisites"): The consequent of rule-1 fires rule-3. If all the prerequisites of a course are completed or scheduled, then the course becomes eligible for scheduling next semesters. To recall, a course is eligible if it satisfies any of the following conditions:
    - has no prerequisite
    - has a completed prerequisite
    - has a scheduled prerequisite in previous semester
- **Rule IV** ("CanTakeNextCourse_After_Taking_Any_Prerequisite"): The consequent of rule-1 fires rule-4. If any of the prerequisites of a course has been completed or scheduled, then the course becomes eligible for scheduling next semesters.
- **Rule V (**"UpdateStatusOfCourseGradeD"): this rules changes the status of a completed course from completed to untaken only for students having on probation status and had taken grade D on completed courses in order to schedule them again.

- **Rule VI** ("DeepestLevelRule1"): This rule determines the deepest level layer of a course in a chain of prerequisites. There are eight multislots or lists (DeepestLayerList1, DeepestLayerList2… DeepestLayerList8) created for this purpose. The first list (DeepestLayerList1) contains only the active course code in course fact. DeepestLevelRule1 updates *DeepestLayerList2* with all courses that have their prerequisites in *DeepestLayerList1*.
- **Rule VII** ("DeepestLevelRule2"): updates *DeepestLayerList3* with all courses that have their prerequisites in *DeepestLayerList2* (the consequent of Rule V fires antecedent of Rule VI).
- **Rule VIII** ("DeepestLevelRule3"): updates *DeepestLayerList4* with all courses that have their prerequisites in *DeepestLayerList3* (the consequent of Rule VI fires antecedent of Rule VII).

- **Rule IX** (“DeepestLevelRule4”): updates *DeepestLayerList5* with all courses that have their prerequisites in *DeepestLayerList4* (the consequent of Rule VII fires antecedent of Rule VIII).
- **Rule X** (“DeepestLevelRule5”): updates *DeepestLayerList6* with all courses that have their prerequisites in *DeepestLayerList5* (the consequent of Rule VIII fires antecedent of Rule IX).
- **Rule XI** (“DeepestLevelRule6”): updates *DeepestLayerList7* with all courses that have their prerequisites in *DeepestLayerList6* (the consequent of Rule IX fires antecedent of Rule X).
- **Rule XII** (“DeepestLevelRule7”): updates *DeepestLayerList8* with all courses that have their prerequisites in *DeepestLayerList7* (the consequent of Rule X fires antecedent of Rule XI).
- **Rule XIII** (“ComputeDeepestLevel”): determines the deepest level of a course based on the last non-empty DeepestLayerList (1→8) list.
- **Rule XIV** (“PrevSchedCoursesRule1”): Updates *PrevSchedCoursesList1* with prerequisites of *CoreReq* field.
- **Rule XV** (“PrevSchedCoursesRule2”): Updates *PrevSchedCoursesList2* with prerequisites of courses in *PrevSchedCoursesList1*.
- **Rule XVI** (“PrevSchedCoursesRule3”): Updates *PrevSchedCoursesList3* with prerequisites of courses in *PrevSchedCoursesList2*.
- **Rule XVII** (“PrevSchedCoursesRule4”): Updates *PrevSchedCoursesList4* with prerequisites of courses in *PrevSchedCoursesList3*.
- **Rule XVIII** (“PrevSchedCoursesRule5”): Updates *PrevSchedCoursesList5* with prerequisites of courses in *PrevSchedCoursesList4*.
- **Rule XIX** (“PrevSchedCoursesRule6”): Updates *PrevSchedCoursesList6* with prerequisites of courses in *PrevSchedCoursesList5*.
- **Rule XX** (“PrevSchedCoursesRule7”): Updates *PrevSchedCoursesList7* with prerequisites of courses in *PrevSchedCoursesList6*.
- **Rule XXI** (“PrevSchedCoursesRule8”): Updates *PrevSchedCoursesList8* with prerequisites of courses in *PrevSchedCoursesList7*.
- **Rule XXII** (“MaxDependencyRule”): determines for each eligible course the maximum dependency

- **Rule XXIII** ("LowerCourseNumberRule"): determines for each eligible course its course number, concatenates the following arguments which constitute a unique key combination and adds the composed key to the course list array:
    - Fact Id
    - Course Number
    - Status
    - Preference
    - PreSchedCourses
    - DeepestLevel
    - MaxDependency
    - LowerCourseNo
    - TypeReqId

5.2.5 Java engine algorithm

The output of JESS engine (CourseList array) is the input in java engine. Java main algorithm starts by performing in sequence the following explicit computational tasks:

1. Bubble sort on course list array in descending order based on sub key value: Status + PreSchedCourses + DeepestLevel + MaxDependency + LowerCourseNo + TypeReqId.

2. A second bubble sort operation is done on course list array in ascending order based on sub key value (DeepestLevel + MaxDependency) to sort courses at same deepest layer level and maximum dependency level in ascending order by course number.

3. Based on enrolment unit rule and the requirement type pattern rule, the program loops over the elements of the sorted CourseList array and selects the non-scheduled courses. The process iterates as long as it doesn't violate the two previously mentioned rules.

4. Java algorithm validates the student class constraint. It's a pre-condition or a predicate that has to be true before scheduling course by updating the "Scheduled" slot to true in the JESS course fact in working memory.

5. In case of violation occurs in student class constraint, then a sequence of actions must be performed:
    a. Loop over all facts in JESS working memory and reset "EligibelClassCourse" slot in course fact
    b. Increment "Class" slot in "StudentInfo" fact

c. Refresh “ActiveStudentClassFact” (Java object of type Fact)

6. Run JESS engine

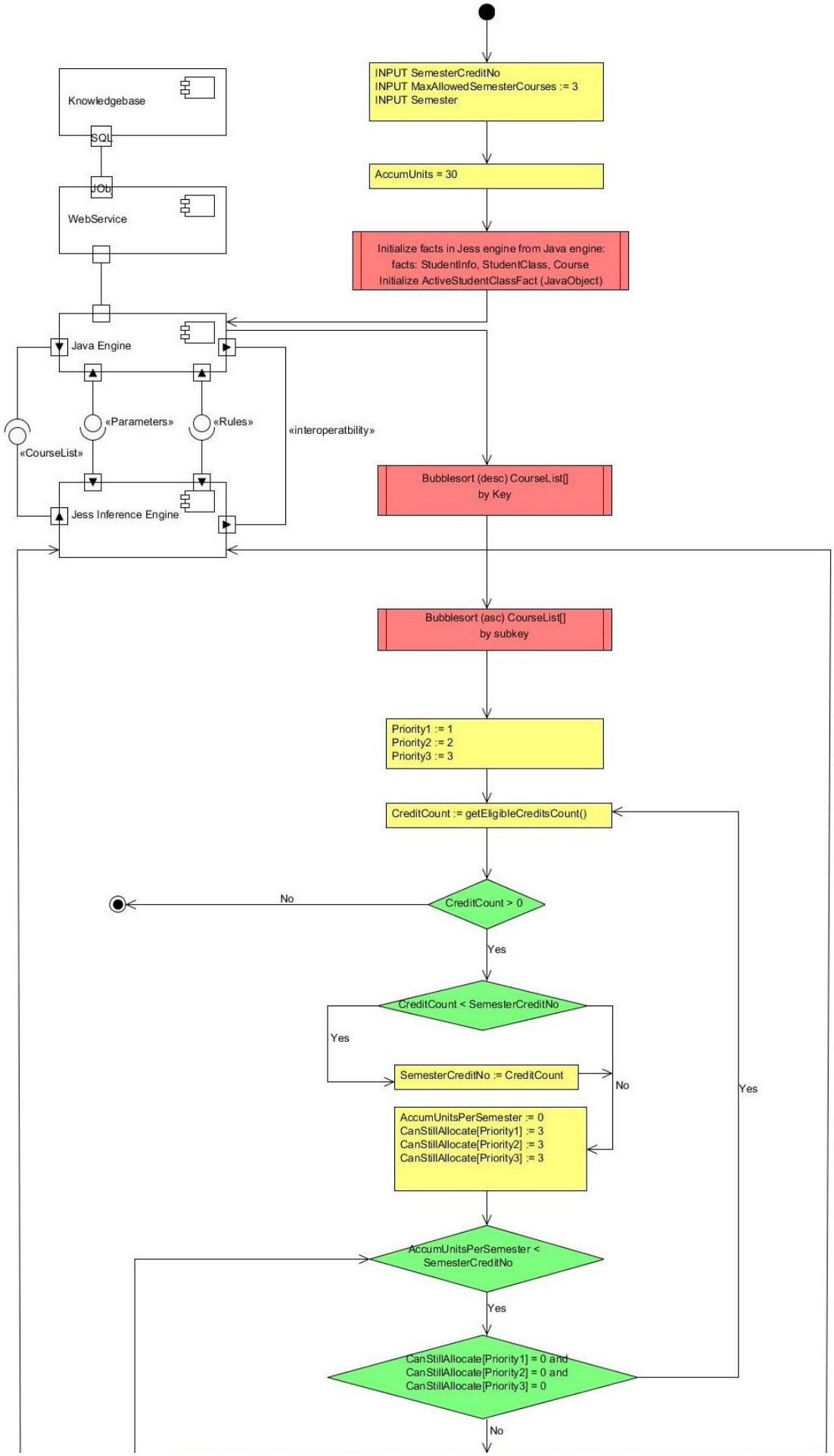
7. Iterate over the above 6 steps until scheduling all unfinished courses over all semesters throughout graduation period.

The following is a description on each dependant variable used in below flow chart:

- Status: is the slot value in a working memory fact that holds the status of the course (1: taken/completed, 2: not taken, 3: dropped, 4: assigned to a completed course with grade D for rescheduling if student is on probation, 5: failed, 10: scheduled). The domain values of status field must be in this listed sequence because failed and dropped courses have the highest priority to be scheduled first.
- PreSchedCourses: is the slot that holds the count of previous scheduled courses in a chain of prerequisites
- DeepestLevel: is the slot that holds deepest layer level value
- MaxDependency: is the slot that holds maximum dependency value
- LowerCourseNo: : is the slot that holds lower course number value
- TypeReqId: is the slot that holds the Id value of the requirement type (remedial, core, general, major, elective)
- *Priority1*, *Priority2* and *Priority3* variables are indexes of *CanStillAllocate* array. The values of these variables stand for priority of each requirement type defined in database table TypeReqPattern
- *CanStillAllocate* array of index priority stands for the maximum limit of courses of a certain requirement type a student allowed to register in one semester
- *CreditCount* variable holds the remaining count of eligible and non-scheduled units
- *SemesterCreditNo* is the maximum number of units allowed per semester
- *AccumUnitsPerSemester* is for accumulating the number of credits or units scheduled so far in current semester
- Get the pattern of requirement type for current semester from knowledge base. This is for indicating the type and number of courses to register in current semester in sequence. For example, if the current semester is Fall/Winter then the pattern is 2 core, 2 major, 1 general, the current semester is Spring/Summer then the pattern is 1

core, 2 major, 2 general. The pattern is sorted by Priority to indicate the importance of which courses to schedule first. The program must iterate

- *CoursesOfReqTypeExist* is a Boolean type variable that indicates whether course list array still contains non-scheduled courses of the current fetched requirement type
- *CoursesCountOfReqType* is the number of courses allowed to schedule in current semester of the fetched requirement type
- *FactId* is the fact id value generated by JESS of the fact in working memory that has the current fetched course from course list
- *FactObject* is the reference from Java engine to the fact in JESS working memory. This allows modifying JESS facts from Java engine. Algorithm checks through *FactObject* the course type and current status (scheduled or not) of the fetched course from course list array.
- *ActiveStudentClassFact* is a reference from Java Engine to the fact in JESS that holds the student class information about his current class level, excluded courses, FromUnits, ToUnits.


Knowledgebase
SQL
JOb
WebService
Java Engine
«Parameters»
«Rules»
«CourseList»
«interoperatbility»
Jess Inference Engine
INPUT SemesterCreditNo
INPUT MaxAllowedSemesterCourses := 3
INPUT Semester
AccumUnits = 30
Initialize facts in Jess engine from Java engine:
facts: StudentInfo, StudentClass, Course
Initialize ActiveStudentClassFact (JavaObject)
Bubblesort (desc) CourseList[]
by Key
Bubblesort (asc) CourseList[]
by subkey
Priority1 := 1
Priority2 := 2
Priority3 := 3
CreditCount := getEligibleCreditsCount()
CreditCount > 0
No
Yes
CreditCount < SemesterCreditNo
Yes
SemesterCreditNo := CreditCount
No
AccumUnitsPerSemester := 0
CanStillAllocate[Priority1] := 3
CanStillAllocate[Priority2] := 3
CanStillAllocate[Priority3] := 3
AccumUnitsPerSemester <
SemesterCreditNo
Yes
CanStillAllocate[Priority1] = 0 and
CanStillAllocate[Priority2] = 0 and
CanStillAllocate[Priority3] = 0
Yes
No

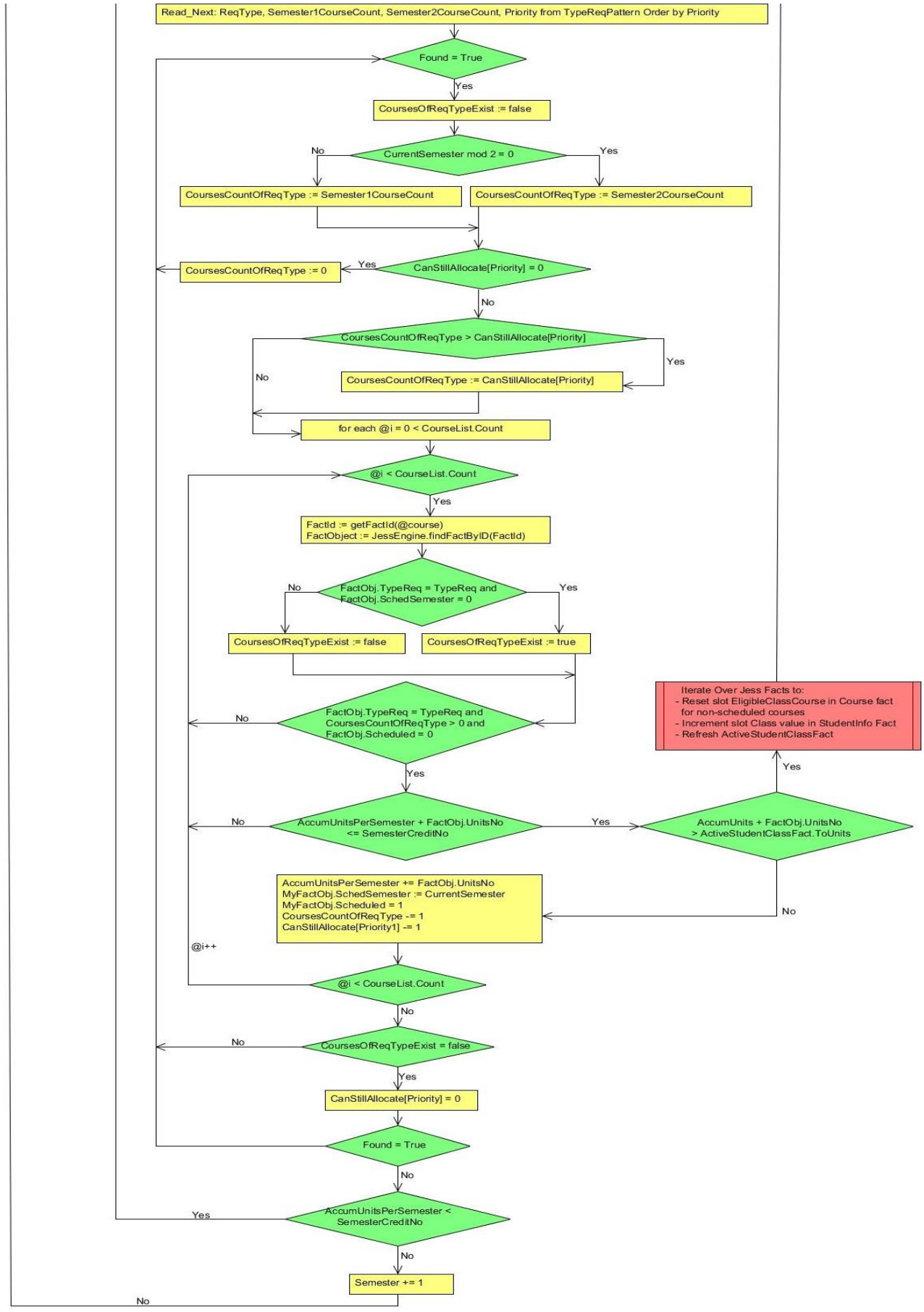


Figure 16: Core Algorithm of CSAS Hybrid Engine Flowchart

5.2.6 Physical database design of Knowledge base

The physical database design is the process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures. Knowledge base repository is implemented in SQL Server 2008 relational DBMS and the below figures show a detailed explanation of the fields of each SQL table: Course, StudentClass and TypeReqPattern.

Course Table Design:

Field	Description	Data Type	Domain	Example
StudentId	The student identification no.	Number	1-*	10
TypeReqId	Type Requirement Id	Number	1-5	1 = Major Req. 2 = General Req. 3 = Core Req. 4 = Elective Req. 5 = remedial Req.
TypeReq	Type Requirement Code	String	A-Z	CR = core req. MR = major req. GR = general req. ER = elective req. RR = remedial req.
CourseNo	Course Code	String	A{A*}d{d*}	CSC213
Description	Course Title	String	A-Z	Program Design & Data Abstraction II
PreReqList	Prerequisite list of course (if exists)	String	A{A*}d{d*}	CSC212, CSC201, etc…
Units	Number of credit hours	Integer	1-12	3
Available	Course is available or will be given by faculty	Integer	0-1	0 = not available 1 = available
Status	Status of course	Integer	1-4	1 = taken , 2 = not taken 3 = failed, 4 = dropped
CreditCarrying	Credit carrying courses (countable in degree requirements)	Integer	0-1	0 = non-credit carrying 1 = credit carrying
TakeAllPreReq	Indicates whether a student has to take all courses in PreReqList domain or take any	Integer	0-1	0 = take any 1 = take all

StudentClass Table Design:

Field	Description	Data Type	Domain	Example
Class	Student class level	Number	1-3	1
Description	Description of class level	String	A-Z	Sophomore

ExcludedCourses	List of non-eligible courses	String	A{A*}d{d*}	CSC480, CSC490
FromUnits	From range of class units	Integer	1-100	31
ToUnits	To range of class units	Integer	1-100	60

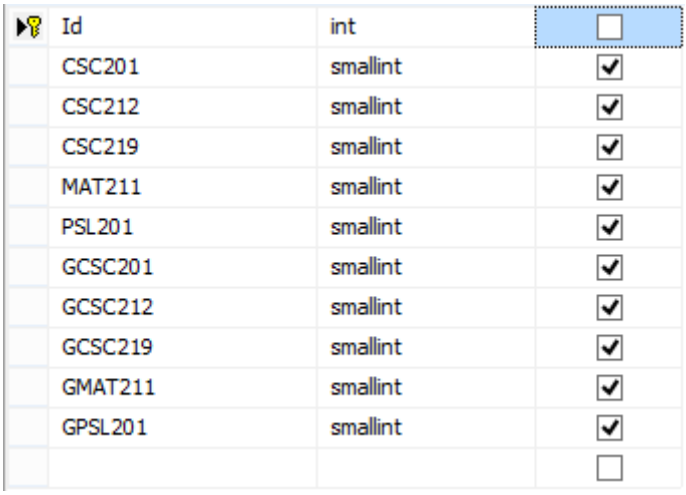
TypeReqPattern Table Design:

Field	Description	Data Type	Domain	Example
TypeReqId	Type Requirement Id	Number	1-5	3 (core req.)
TypeReq	Type Requirement Code	String	A-Z	RR or CR or MR or GR or ER
Semester1	Number of planned courses in Fall Semester	Integer	1-3	2
Semester2	Number of planned courses in Spring Semester	Integer	1-3	1
Semester3	Number of planned courses in Summer Semester	Integer	1-3	0
Priority	Importance of requirement	Integer	1-3	1

StudentHistory SQL table

Field	Description	Data Type	Domain	Example
Id	The student id	Number	1-*	3 (core req.)
Course1	1st scheduled course	Integer	0-1	1
Course2	2st scheduled course	Integer	0-1	0
Course3	3st scheduled course	Integer	0-1	1
Course4	4st scheduled course	Integer	0-1	0
Course5	5st scheduled course	Integer	0-1	1
Course1Grade	Grade of Course1	Integer	0-100	80
Course2Grade	Grade of Course2	Integer	0-100	79
Course3Grade	Grade of Course3	Integer	0-100	75
Course4Grade	Grade of Course4	Integer	0-100	0
Course5Grade	Grade of Course5	Integer	0-100	90
AvgGrade	Used for temporary average computation of the courses of itemsets which exceeded minimum confidence specified by advisor	Float	0-100	88

StudentHistory is a temporary table created in real time by the system during course association analysis. The first five fields' names stand for scheduled courses in any selected semester from the recommended plan by the prototype. The field name is the name of the scheduled course (e.g. Course1 replaced by CSC201). The other remaining five fields stand for the grade of each scheduled course field (e.g. Course1Grade replaced by GCSC201: G for Grade, CSC201 for course name). The following figures show the design form and output result of *StudentHistory*:



Id	int	☐
CSC201	smallint	☑
CSC212	smallint	☑
CSC219	smallint	☑
MAT211	smallint	☑
PSL201	smallint	☑
GCSC201	smallint	☑
GCSC212	smallint	☑
GCSC219	smallint	☑
GMAT211	smallint	☑
GPSL201	smallint	☑
		☐

The following figure stands for the historical data of 11 anonymous graduated students who had taken a subset of these courses but within same semester. Information of one student may be found in one or more records in below table and each record stands for the semester he/she took a subset of these five courses. Of course, the data preparation of below table require implementation of a subroutine that fetches from historical data for students who had accomplished the recommended set of courses shown as column names in below figure:

	Id	CSC201	CSC212	CSC219	MAT211	PSL201	GCSC201	GCSC212	GCSC219	GMAT211	GPSL201
1	1	1	1	1	0	1	80	75	78	0	90
2	2	0	1	1	0	0	79	85	0	0	0
3	3	1	0	1	0	1	81	0	73	0	89
4	4	1	1	0	1	1	83	86	0	86	91
5	5	1	1	0	0	1	82	81	0	0	0
6	6	0	1	1	0	1	0	87	82	0	88
7	7	1	1	1	1	1	81	77	74	91	92
8	8	0	1	0	1	1	0	90	0	85	86
9	9	1	1	0	1	0	88	90	0	78	0
10	10	0	1	1	0	1	0	91	86	0	83
11	11	1	1	0	0	0	88	89	0	0	0

Figure 17: *StudentHistory* Output

5.2.7 Prototyping with RapidMiner

RapidMiner is a software platform developed by the company of the same name that provides an integrated environment for machine learning, data mining, text mining, predictive analytics and business analytics. RapidMiner is developed on a business source model which means the core and earlier versions of the software are available under an OSI-certified open source license on Sourceforge. RapidMiner offers Starter Edition for free download whereas Personal Edition and Professional Edition require licensing. Not all software components in Starter Edition are accessible for free, for example, the operator that reads data from databases is locked. So, I had to extract data table StudentHistory to excel file and utilize the "Read Excel" operator instead of "Read Database" operator.

In RapidMiner, we create a process that utilizes operators and data repositories.
The implementation of knowledge discovery process (KDP) in RapidMiner tool starts by creating a new process which is based on a chain of interconnected operators of different functionalities. Each operator has input and output ports. Finally, the RapidMiner process can be saved as a file (e.g. StudentHistory.rmp) on hard disk storage.

	A	B	C	D	E	F	G	H	I	J	K
1	Id	CSC201	CSC212	CSC219	MAT211	PSL201	GCSC201	GCSC212	GCSC219	GMAT211	GPSL201
2	1	1	1	1	0	1	80	75	78	0	90
3	2	0	1	1	0	0	79	85	0	0	0
4	3	1	0	1	0	1	81	0	73	0	89
5	4	1	1	0	1	1	83	86	0	86	91
6	5	1	1	0	0	1	82	81	0	0	0
7	6	0	1	1	0	1	0	87	82	0	88
8	7	1	1	1	1	1	81	77	74	91	92
9	8	0	1	0	1	1	0	90	0	85	86
10	9	1	1	0	1	0	88	90	0	78	0
11	10	0	1	1	0	1	0	91	86	0	83
12	11	1	1	0	0	0	88	89	0	0	0
13	12	1	1	0	1	0	91	93	0	88	0
14	13	0	1	1	0	1	0	87	85	0	84

Figure 18: StudentHistory table in excel format - StudentHistory.xls

After finalizing preparation process of the input file, the next step is to import StudentHistory.xls into a new process created in RapidMiner application. The following snapshot shows five operators followed by an explanation of the specification of each:

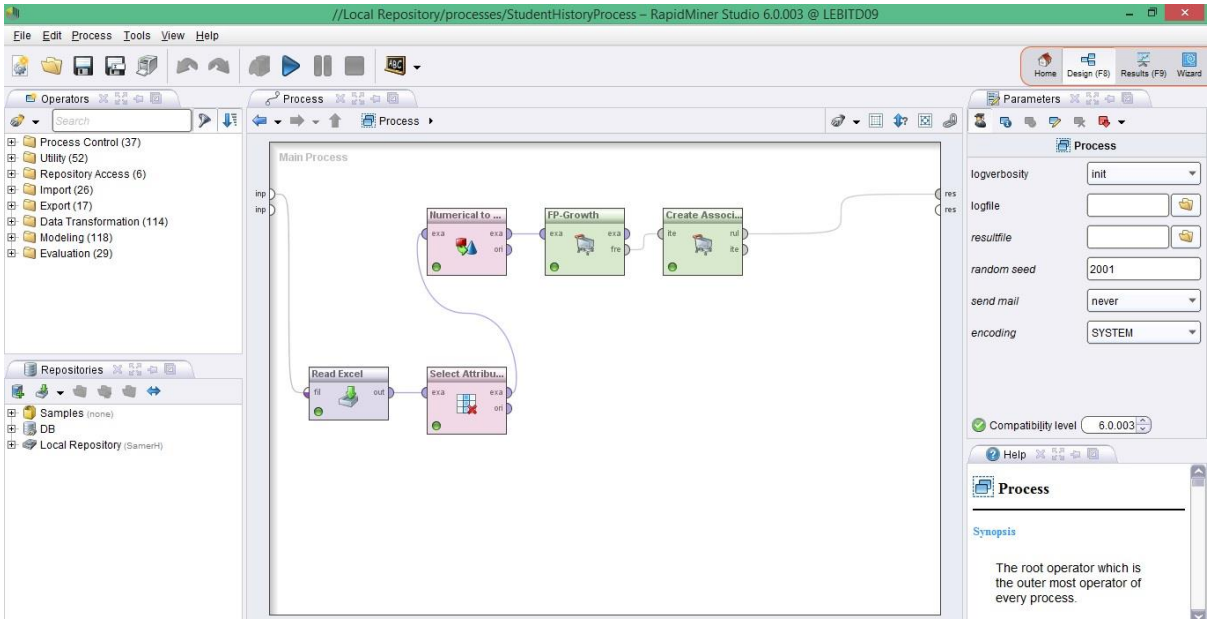


Figure 19: RapidMiner Application

"Read Excel" operator: This operator reads an ExampleSet by loading data from Microsoft Excel spreadsheets via its input port and delivers the Excel file in tabular form along with the meta-data via its output port.

Row No.	Id	CSC201	CSC212	CSC219	MAT211	PSL201	GCSC201	GCSC212	GCSC219	GMAT211	GPSL201
1	1	1	1	1	0	1	80	75	78	0	90
2	2	0	1	1	0	0	79	85	0	0	0
3	3	1	0	1	0	1	81	0	73	0	89
4	4	1	1	0	1	1	83	86	0	86	91
5	5	1	1	0	0	1	82	81	0	0	0
6	6	0	1	1	0	1	0	87	82	0	88
7	7	1	1	1	1	1	81	77	74	91	92
8	8	0	1	0	1	1	0	90	0	85	86

Figure 20: Output of Database Table

"Select Attributes" operator: This operator selects which attributes of an ExampleSet should be kept and which attributes should be removed. This is used in cases when not all attributes of an ExampleSet are required; it helps in selection of required attributes. The output of this operator is an ExampleSet with selected attributes.

Row No.	CSC201	CSC212	CSC219	MAT211	PSL201
1	1	1	1	0	1
2	0	1	1	0	0
3	1	0	1	0	1
4	1	1	0	1	1
5	1	1	0	0	1
6	0	1	1	0	1
7	1	1	1	1	1
8	0	1	0	1	1

"Numerical to Binominal" operator: This operator changes the type of the selected numeric attributes to a binominal type (also called binary). It also maps all values of these attributes to corresponding binominal values. This operator not only changes the type of selected attributes but it also maps all values of these attributes to corresponding binominal values. Binominal attributes can have only two possible values i.e. 'true' or 'false'.

Row No.	CSC201	CSC212	CSC219	MAT211	PSL201
1	true	true	true	false	true
2	false	true	true	false	false
3	true	false	true	false	true
4	true	true	false	true	true
5	true	true	false	false	true
6	false	true	true	false	true
7	true	true	true	true	true
8	false	true	false	true	true

"FP-Growth" operator: This operator efficiently calculates all frequent itemsets from the given ExampleSet using the FP-tree data structure. It is compulsory that all attributes of the input ExampleSet should be binominal.

"Create Association Rule" operator: This operator generates a set of association rules from the given set of frequent itemsets which contain a subset of the recommended courses. Each itemset has a confidence ratio.

5.2.8 Computational steps of student's percentile

In order to find out the percentile of the threshold grade specified by advisor:

1. Consider from all the generated frequent itemsets by RapidMiner process only the courses that have their minimum confidence exceeds the specified confidence parameter by the advisor.
2. Compute the average of courses in these itemsets for each student.
3. Find the percentile of any chosen threshold grade by the advisor. Advisor may specify for each itemset a different threshold grade; consequently, a new percentile will be generated for each itemset and its threshold grade. By knowing such percentile, we can deduce the count the students that fall below the percentile. For example, suppose the threshold grade is 80 and the count of students that have average below 80 is 70 students out of a total of 100 students. This means the students that have 80 average grade are at the 70^{th} percentile. To recall, a percentile is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall). The below figure is an example on percentile:

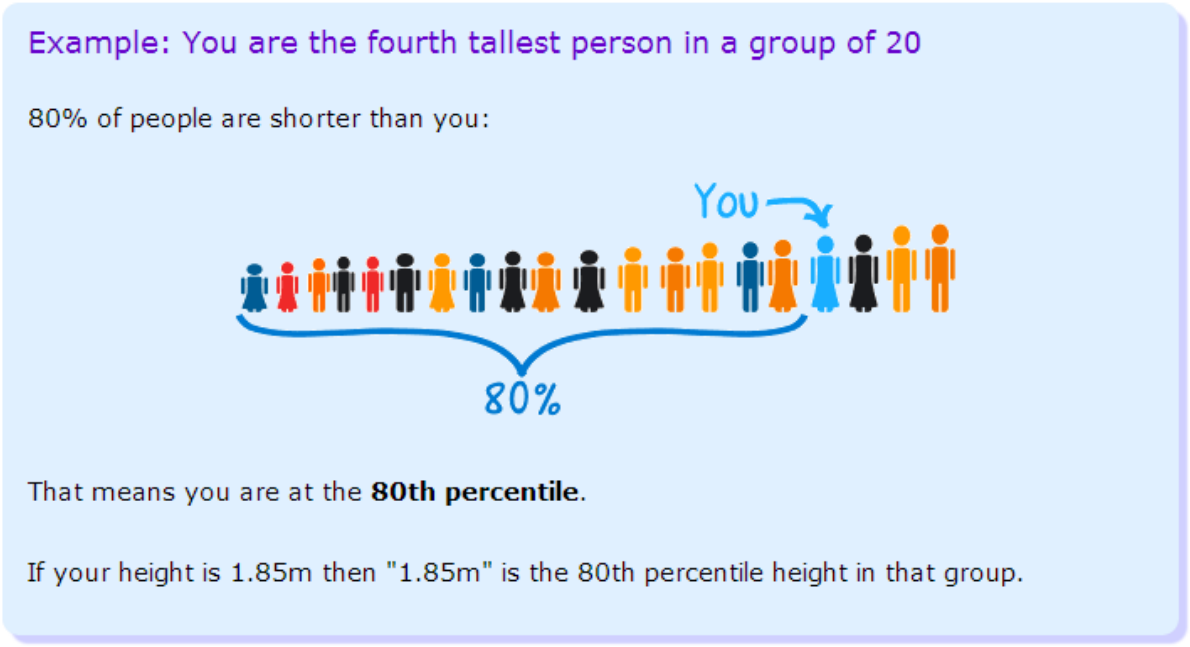


Figure 21: Percentile Illustration

http://www.mathsisfun.com/data/percentiles.html

The computation of the percentile takes place at the back end SQL Server database (knowledge base). The output itemset (courses) of RapidMiner subroutine and the threshold grade in java engine become input arguments of the stored procedure parameters that

computes the percentile and returns results to java engine based on data of StudentHistory table.

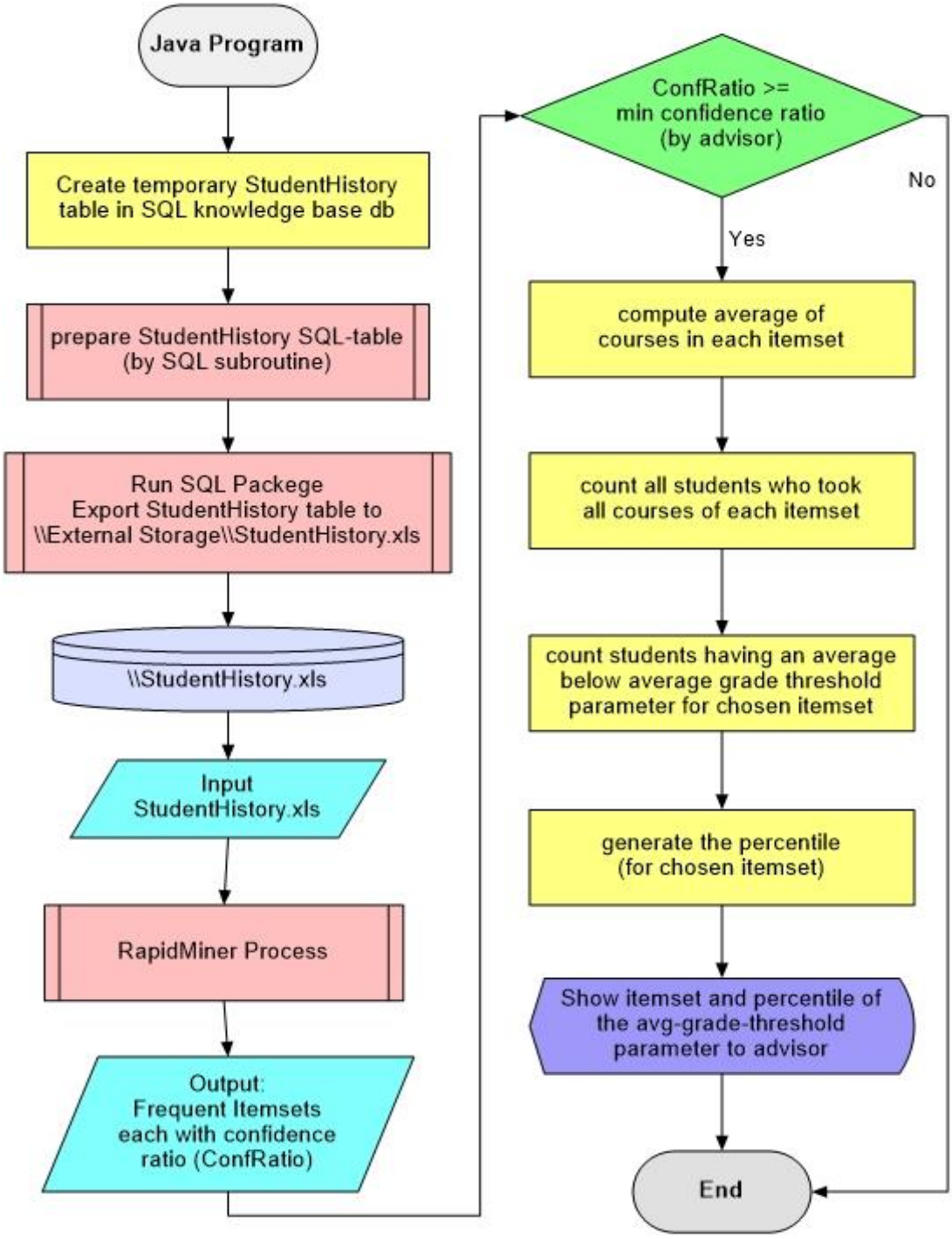


Figure 22: Percentile Computational Process Flowchart

5.2.9 Dynamic management of RapidMiner process in Java

We can benefit from the created process file by RapidMiner (StudentHistory.rmp) with RapidMiner.jar library in programming a subroutine in Java using NetBeans IDE 8.0

framework. The generated process file (StudentHistory.rmp) is based on the input excel file StudentHistory.xls which consists of meta-data (columns names) and students' data. The excel file is dynamically generated each time the content of SQL-table StudentHistory gets exported to StudentHistory.xls file. When running the process file from NetBeans, the "Read Excel" operator loads the dynamically generated data in excel, but the meta-data of the operator didn't change (metadata of previous courses itemset). The following snapshot shows a subroutine that can manipulate the meta-data of the "Read Operator" by renaming the column names to the names of a new set of scheduled courses.

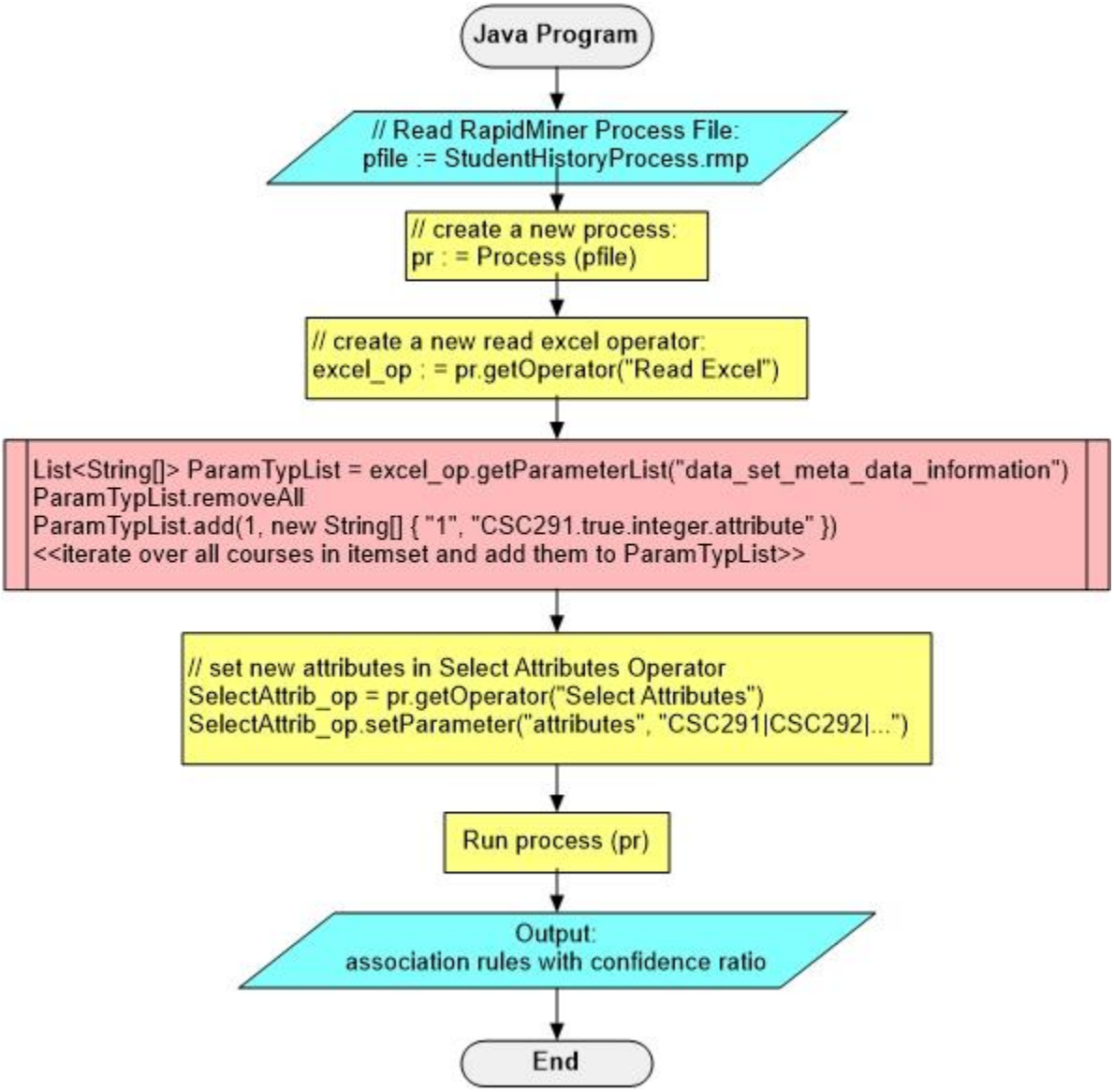


Figure 23: Integration of RapidMiner Process File into Java Application

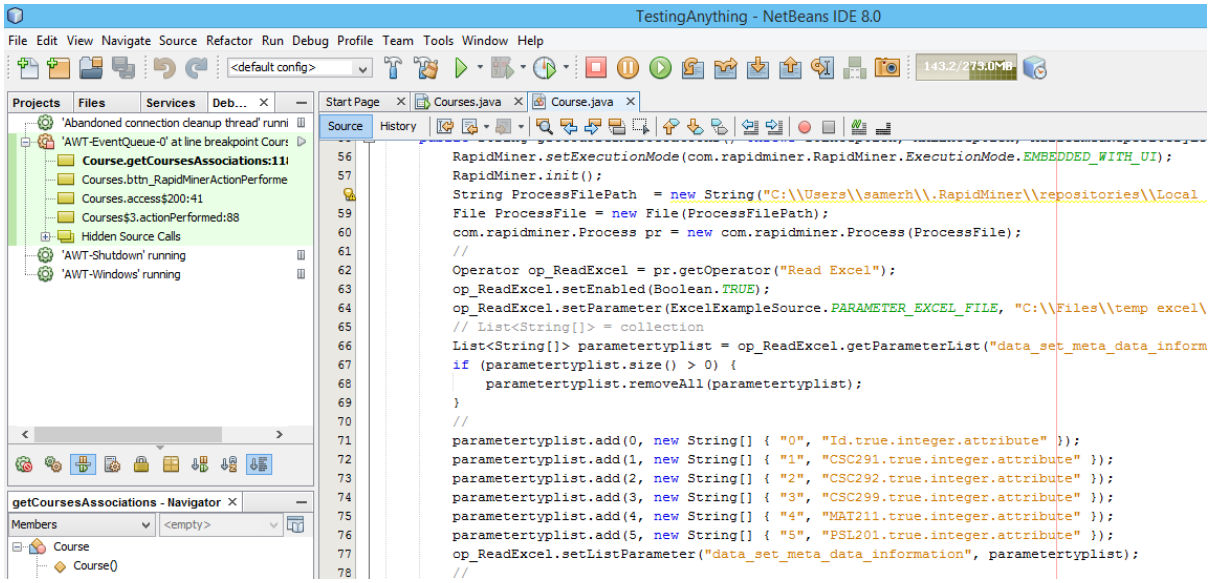


Figure 24: Example of Java Application Code

5.3 CSAS Architecture

CSAS architecture includes four main server components:

5.3.1 JESS Engine

Its role is to process the facts and rules and generates a customized advice to the student. JESS 7 operates fine on NetBeans java platform but when testing it on android platform, I encountered some technical problems which I shared with Mr. Friedman-Hill who confirmed to me by email that JESS 8.0 supports android platform and will be released in the coming months. The coming up version 8.0 is going to be used in CSAS system. The expert knowledge and user inputs are stored as rules and facts in the JESS knowledge engine to be processed by JESS-Rete algorithm to match the rules and facts to generate new results. Android platform uses Java language; so, we chose JESS which is written in Java and allows interoperability and data exchange with Java in both directions. Many expert systems had been developed using PROLOG and LISP, but with the advent of more powerful languages such as Java it's better to use a more compatible inference engine with Java such as JESS.

5.3.2 Internet Information Services (IIS, formerly Internet Information Server):

IIS is Microsoft Web Server on which a web service application is deployed on a communication server that acts as an interface between mobile applications and knowledge

base (SQL Server database). Web service responds to http-read-write requests issued by mobile application. For read-requests, it accesses back-end database and transforms retrieved dataset result into JSON string format (serialization) prior to sending data via internet connection to mobile device. For write-requests, it transforms JSON string sent from mobile device and transforms it to objects (deserialization) prior to writing to back-end database.

5.3.3 Android Mobile Application

Is a java application developed on eclipse that is responsible for preparing facts to be processed by JESS engine and transforms user requests to http-requests sent to web service application for storing or retrieving data from knowledge base database. Gson is a java library used by mobile application to convert JSON to Java objects and vice-versa.

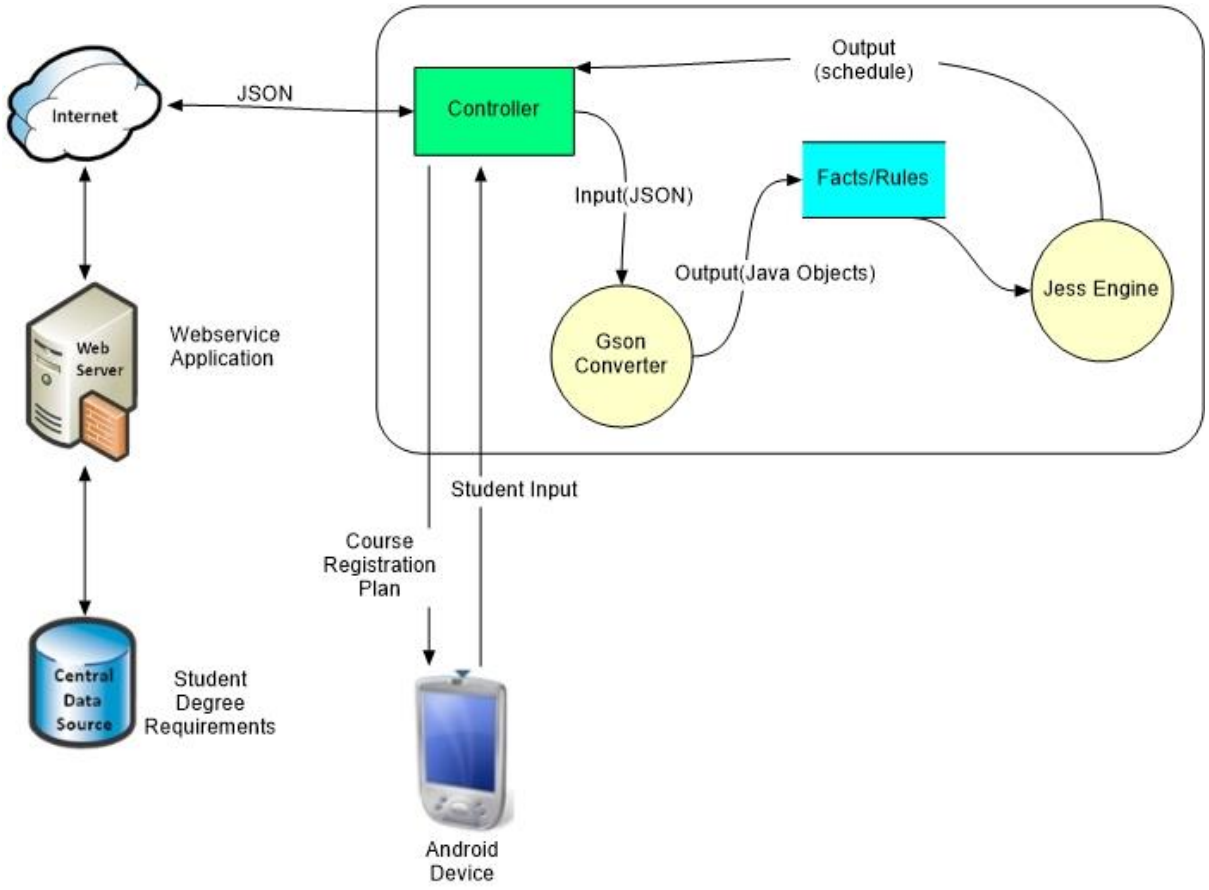


Figure 25: CSAS System Data Flow Diagram

1) For students:

1. Students request for their course registration plan with personal inputs through mobile application.
2. The CSAS controller sends http-request to web server to retrieve student information degree.
3. The web server which acts as an interface between database data-source and mobile application retrieves data of student request from data-source database, translates it to JSON format and send it via internet to mobile application for further processing.
4. The CSAS controller receives JSON data and translates it to Java objects using GSON library (deserialization) and populates the slots of facts of the JESS engine
5. JESS engine processes the dynamic facts and rules and generates the schedule output to controller which displays it finally in the android-GUI application (XML layout) to be viewed by the student.

5.3.4 Relational Database Management System (RDBMS)

Is the repository or the knowledge base database that is synchronized with SIS database to retrieve and store all current information needed about student academic status. The synchronized connection is needed to ensure instant refreshing of knowledge base.

5.4 Main Results

After integrating all rules into model base of CSAS such as degree requirement rule, traditional hierarchal (deepest level, mac-dependency, lower course number), suppose running a test case of CSAS with the following inputs:

1. The starting semester of scheduling process is semester 1 i.e. Fall Semester
2. Credits Given on admission is 30
3. The maximum limit of units/semester is 15 (enrolment unit rule)
4. Eligible courses as of student class level (Senior class courses: CSC480, CSC490)
5. An instance of requirement type pattern rule:

Requirement Type Pattern Table indicates the number of courses by semester and req-type:

TypeReq	Semester1 (Fall)	Semester2 (Spring)	Semester3 (Summer)	Priority
RR	3	3	0	1
CR	2	1	0	2
MR	2	2	0	3
GR	1	2	0	4
ER	2	1	0	5

After running CSAS, the following course schedule plan has been generated automatically. The results are valid and meet the functional requirements. For example, the eligible courses that have the highest preference by requirement type, highest deepest level, maximum dependency, lowest course number (at same DeepestLevel and Max-Dependency) and requirement type RR (four dimensions of courses' prioritization) are scheduled first (ENL002). The requirement type pattern is a fourth dimension proposed in this thesis for selection pattern of courses in a semester by course type (doesn't not impact course sequencing). For example, the system can still allocate 3 credits in semester 1 of type RR, but this is not possible for semester-1 because ENL002 is the only eligible of type RR; consequently, the system proceeds in selecting eligible courses of type CR (Priority=2) and again based on the four dimensions of courses' prioritization. Semester-1 has reached its maximum allocation limit as indicated by the enrolment unit rule. It's important to recall that there may be prerequisite courses among the scheduled courses in semester-1; consequently, their courses at next lower deepest layer become eligible for scheduling in semester-2. For example, CSC213 is scheduled at semester-2 directly after scheduling its prerequisite CSC212. Some courses like CSC311 require all its prerequisites CSC213 and MAT211 to be satisfied or scheduled (CSC311 scheduled at Semester-6 whereas CSC213 at semester-2 and MAT211 at semester-5). Some courses require any of their prerequisites to be satisfied or scheduled such as CSC312 (perquisites: CSC219, CSC222, EEN220). You can observe that CSC219 scheduled at semester-3 whereas CSC312 scheduled at semester-4. U can notice that none of the below 8 semesters violated the enrolment unit rule (total number of credits didn't exceed 15). Also, eligible courses by student class level CSC480, CSC490 (senior level) are scheduled at semesters-7 and semester-8 respectively. Moreover, the sequence of scheduling (by semester) for a chain of prerequisites is guaranteed i.e. system must give a scheduling priority for a course in an already started scheduled chain over other recommended courses according to the count of previous scheduled courses in its chain so far. For example,

CSC212 is the first prerequisite scheduled in semester 1 in a chain of four courses ends with CSC325 scheduled in semester 4.

Semester 1						
Req. Type	Course	Description	Previous Scheduled Courses	Deepest Level	Maximum Dependency	Units
CR	CSC212	PRO DESIGN DATA ABSTRACTION I	0	4	2	3
RR	ENL002	INTENSIVE ENGLISH II	0	1	0	12
Semester 2						
RR	ENL105	COLLEGE ENGLISH I	1	4	2	5
CR	CSC213	PRO DESIGN DATA ABSTRACTION II	1	3	8	3
MR	CSC218	PRINC.OF COMMUNICATION SYSTEMS	1	2	1	3
GR	CSC201	COMPUTERS AND THEIR USE	0	1	0	3
Semester 3						
CR	CSC219	DIGITAL COMPUTER FUNDAMENTALS	0	3	1	3
CR	CSC313	DATA STRUCTURES USING C++	2	2	1	3
MR	CSC316	COMPUTER SECURITY & THEIR DATA	2	1	1	3
MR	CSC387	ADV PROGRAMMING USING JAVA	2	1	0	3
RR	ENL110	COLLEGE ENGLISH II	2	3	1	3
Semester 4						
CR	MAT211	DISCRETE MATH	0	2	1	3
MR	CSC325	ANALYSIS OF ALGORITHMS	3	1	0	3
MR	CSC423	SOFTWARE ENGINEERING	2	1	0	3
GR	BAD201	FUNDAMENTALS OF MANAGEMENT	0	1	0	3
GR	ENL213	SOPHOMORE ENGLISH RHETORIC	3	2	1	3
Semester 5						
CR	MAT213	CALCULUS III	0	2	1	3
CR	MAT215	LINEAR ALGEBRA I	0	1	0	3
MR	CSC425	DATA COMMU & COMPUT NETWORKS	2	1	0	3
MR	CSC426	PRINCIPLES OF DATABASE SYSTEMS	2	1	0	3
GR	ENL230	ENGLISH IN THE WORKPLACE	4	1	0	3
Semester 6						
CR	MAT224	CALCULUS IV	2	1	0	3
MR	CSC432	INTRO TO ARTIFI INTELLIGENCE	2	1	0	3
MR	CSC463	ADVANCED SOFTWARE	2	1	0	3
GR	NTR201	BASIC HUMAN NUTRITION	2	1	0	3
GR	CSC202	COMPUTERS FOR VISUAL ARTS	0	1	0	3
Semester 7						
MR	CSC311	THEORY OF COMPUTATION	1	1	0	3
MR	CSC312	COMPUTER ARCHITECTURE	1	2	2	3
MR	CSC480	INTERNSHIP	0	1	0	1
GR	HIT211	HIST OF LEB & THE ME	0	1	0	3
ER	PDP201	BASIC PHOTOGRAPHY	0	1	0	3
ER	PES314	HANDBALL	0	1	0	1

Semester 8						
MR	CSC414	APPLIED OPERATING SYSTEMS	2	1	0	3
MR	CSC490	SENIOR STUDY	0	1	0	3
GR	ARB212	ADV ARABIC GRAMMAR	0	1	0	3
ER	PES322	DANCING	0	1	0	2
Semester 9						
GR	REG314	MARRIAGE & FAMILY-CATH. CHURCH	0	1	0	3

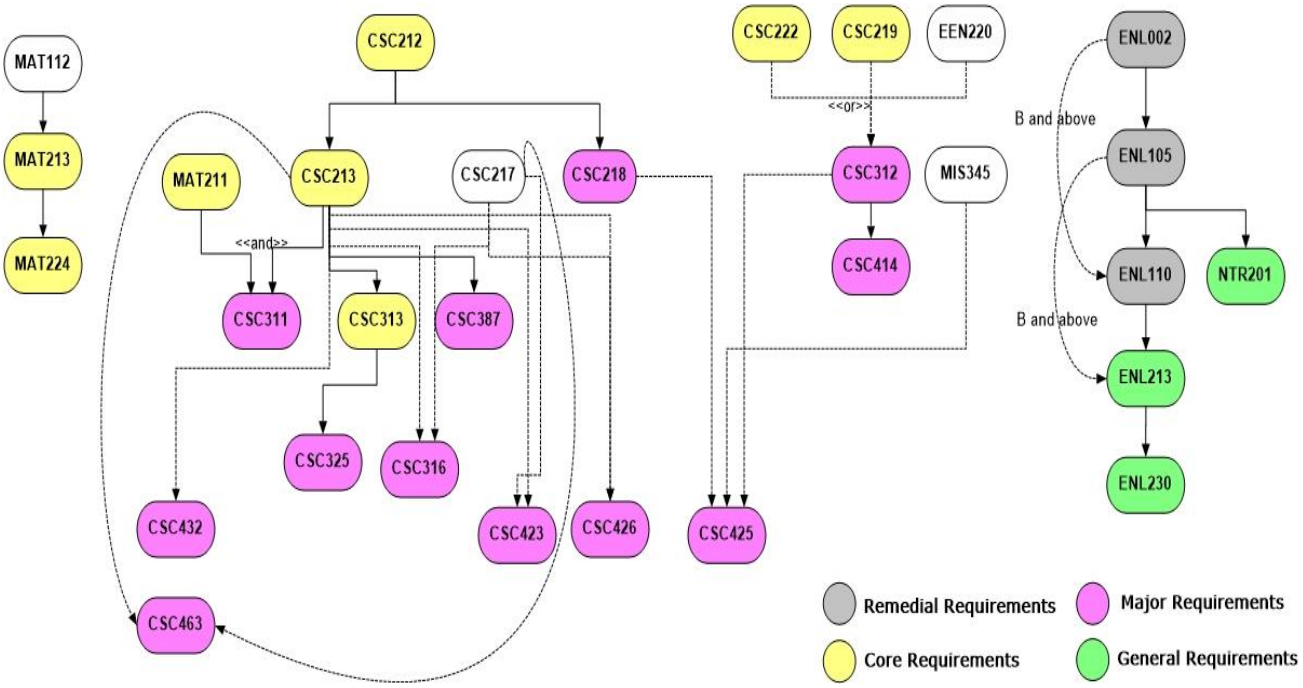


Figure 26: Relation between All Contract Sheet Requirements for CS Student

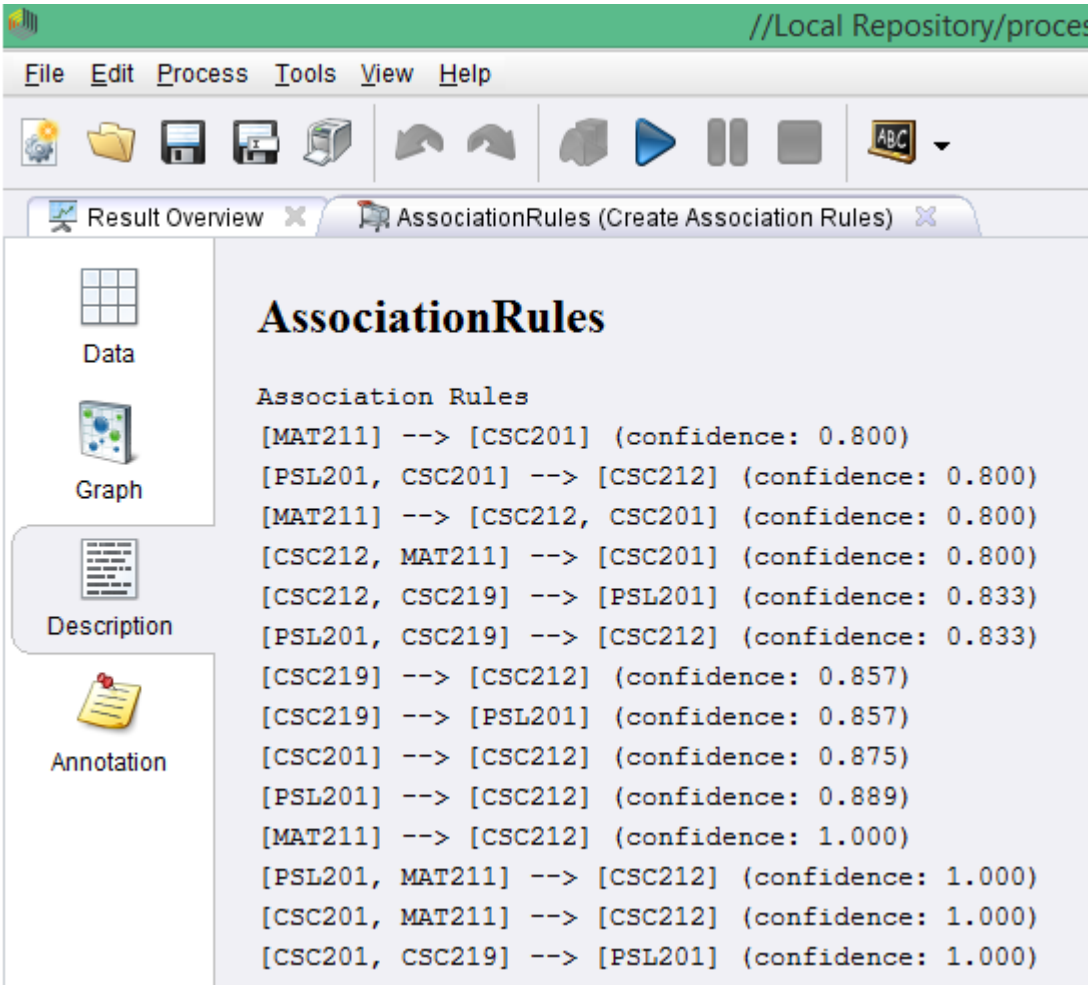


Figure 27: RapidMiner Association Rules Process Results

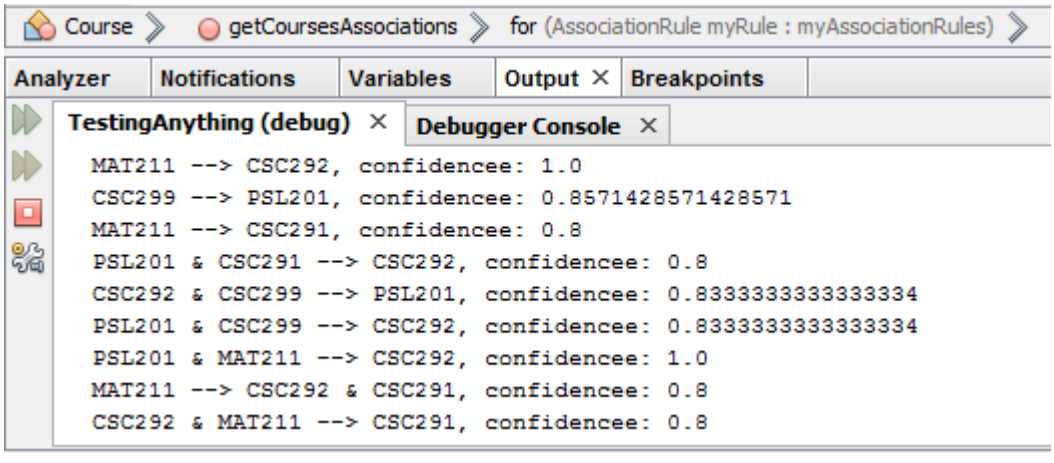


Figure 28: RapidMiner Association Rules Process Console Output Results

Chapter 6
Conclusion and future work

6.1 Conclusion

CSAS has been tested for an anonymous B.S degree student at NDU who has completed his bachelor degree in Computer Science in Spring-2014. The sequence of scheduled courses generated by system were not the same as the sequence presented in student's transcript, but the results show that courses are optimally allocated according to the implemented functional requirements.

The functional requirements are expressed in the form what the system must do according to the predefined rules and facts in Jess engine and the explicit detailed algorithm implemented in Java. To recall, functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish.

6.2 Future work

The proposed CSAS system is intended for use in mid-range universities. As a first step, it's needed to develop an experimental version to be launched at the Faculty of Natural and Applied Sciences at Notre Dame University. The modular structure, web based and mobile application design makes it possible to be launched and used elsewhere. In our future work we hope to elaborate more considering more test cases from other departments and focus on issues relating to data security and database mapping, in order to prevent unauthorized access to data.

CSAS will not replace the need for wise and sympathetic counsel from human advisors, CSAS focuses students more clearly on issues to consider and let them have instant access to the expert system before contacting their advisors, thus alleviating academic staff of part of their burden.

For the future, there is lots of work still to be done to realize a full-fledged automated decision support system such as CSAS especially at the implementation level:

1. Establish a synchronized connection between knowledge database and student information system:

    a. Synchronized connection facilitates connection to the student information database directly so that part of student user input like courses taken can be automated
2. Implementation of Web Server application to receive and respond to http requests, Web page application using Java Server Pages as well as mobile android-application.
3. Testing CSAS model on student requirements of other majors
4. Making the degree requirement specification more scaleable:
    a. Adding more sequencing and scheduling rules to the model

Bibliography

[1] Onyeka, E., Olawande, D., and Charles, A. CAES: A model of an RBR-CBR course advisory expert system. In Information Society (i-Society), 2010 International Conference on (June 2010), pp. 37–42.

[2] Ho, K., and Lu, M. Web-based expert system for class schedule planning using jess. In Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on. (Aug 2005), pp. 166–171.

[3] Lee, C. Importance and current issues of dss. http://www.studymode.com/essays/Article-Importanceandcurrentissuesofdss-66407779.html. Accessed: 6, 2014.

[4] Hill, E. J. F. Jess, the Java Expert System Shell. Tech. rep., SANDIA National Laboratories, 2000.

[5] Al-Nory, M. Simple decision support tool for university academic advising. In Information Technology in Medicine and Education (ITME), 2012 International Symposium on (Aug 2012), vol. 1, pp. 53–57.

[6] Nambiar, A., and Dutta, A. Expert system for student advising using jess. In Educational and Information Technology (ICEIT), 2010 International Conference on (Sept 2010), vol. 1, pp. V1–312–V1–315.

[7] Murray, W. S., and Le Blanc, L. A. A decision support system for academic advising. In Proceedings of the 1995 ACM Symposium on Applied Computing (New York, NY, USA, 1995), SAC ’95, ACM, pp. 22–26.

[8] Deniz, D., and Ersan, I. Using and academic dss for student, course and program assessment. In Int. Conf. on Engineering Education (ICEE’01) (2001), pp. 6B8/12–18.

[9] Chau, V. T. N., and Phung, N. H. A knowledge-driven educational decision support system. In Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on (Feb 2012), pp. 1–6.

[10] Golumbic, M. C., Markovich, M., Tsur, S., and Schild, U. J. A knowledge-based expert system for student advising. IEEE Trans. on Educ. E-29, 2 (May 1986), 120–124.

[11] E. Figenbaum, E. Rich, G. Wiederhold, and M. Harrison. Advanced Software Applications in Japan. Noyes Data Corporation, 1995.

[12] Kiernan, G., Koltun, A., and Schwartz, E. N. Constructing an expert system-software engineering of a different kind. In Proceedings of the 1988 ACM Sixteenth Annual Conference on Computer Science (New York, NY, USA, 1988), CSC ’88, ACM, pp. 223–231.

[13] Amornchewin, R., and Kreesuradej, W. Incremental association rule mining using promising frequent itemset algorithm. In Information, Communications Signal Processing, 2007 6th International Conference on (Dec 2007), pp. 1–5.

[14] Chen, M.-S., Han, J., and Yu, P. S. Data mining: An overview from a database perspective. IEEE Trans. on Knowl. and Data Eng. 8, 6 (Dec. 1996), 866–883.

[15] Cios, K., Pedrycz, W., Swiniarski, R., and Kurgan, L. Data Mining: A Knowledge Discovery Approach. Springer, 2007.

[16] Agrawal, R., Imielinski, T., and Swami, A. Database mining: A performance perspective. IEEE Trans. on Knowl. and Data Eng. 5, 6 (Dec. 1993), 914–925.

……