CS2310 – Software Engineering, Fall 2014,

Professor: Dr. Shi Kuo Chang

Name: Nikolaos Romanos Katsipoulakis

Title: The design of an SIS Recommender System Component

Abstract

This report presents a detailed overview of a Recommender Engine component for the SIS Server. This component is capable of taking elaborate decisions and pro actively generate alert messages in the SIS component network. This way, unwanted situations can be avoided in which patients may be in a state of imminent health deterioration.

Introduction

Slow Intelligence systems have been proven to be an alternative, more elegant tool for problem solving. Through a discrete set of binary operators, a slow intelligent system can effectively reach a solution for a real-life problem.

In the course of Software Engineering we have been presented with the SIS Server workbench. This system consists of some individual components which can exchange information in a distributed manner. In the heart of the SIS system lies the server. Its main responsibility is to keep track of component-members, each one of them having a different functionality. All of the components communicate through a well-defined message set, which are exchanged in XML form.

Components model different stand-alone systems in the SIS network. Each of them can be a sensor for capturing temperature, a motion detector, or even a remote supercomputer. One can easily understand that the SIS test bed can be easily extended and scale indefinitely.

In this report, I am going to present (i) the design of the Recommender Engine component, (ii) its implementation details, and (iii) three usage scenarios in a real-life situation.

System Architecture

Illustration 0 depicts a full network of SIS components. One can see that many components gather data from different sensors. A sensor can be gathering Temperature (Temp_Component), Blood Pressure of a human, ambient humidity etc. In addition, a component can be a processing entity (like a supercomputer for processing large data files gathered by sensors), or even a communication interface to pass on messages from the SIS system to a remote network (i.e. an Up-loader can make data available to other users by uploading them to a website - illustration 0).

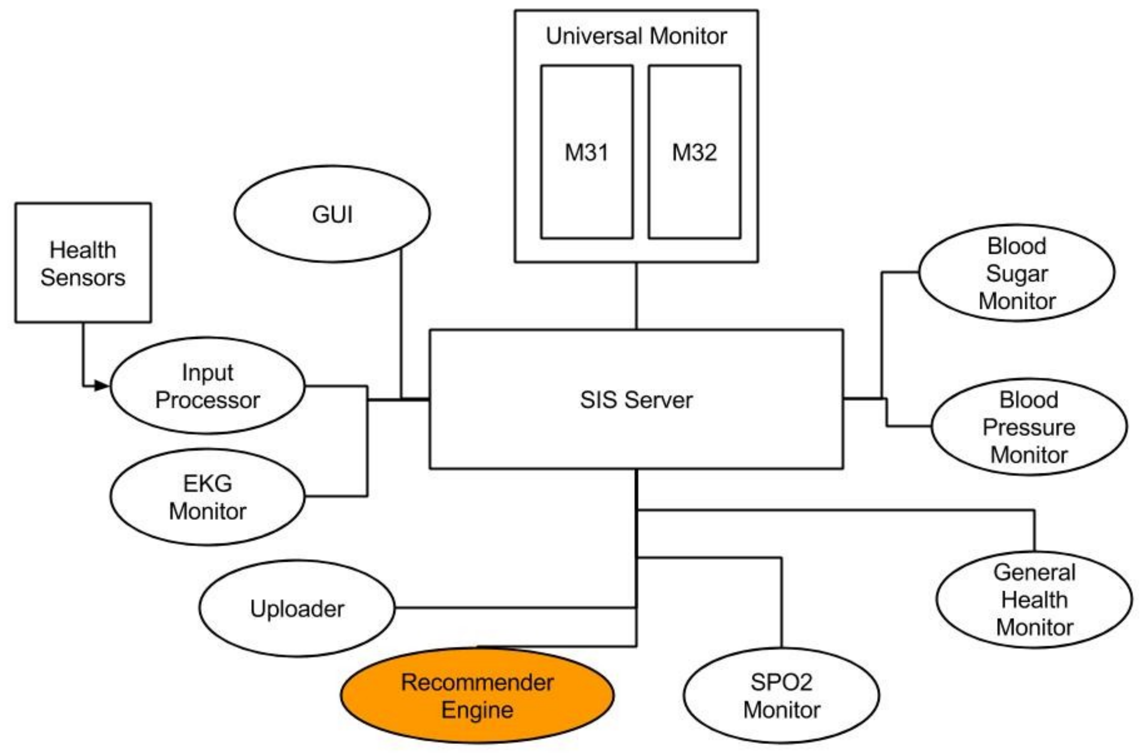


Illustration 0: SIS component network with Recommender Engine

The Recommender Engine is itself a component in this network of SIS components and waits for messages produced by sensors. Specifically, in its current state the Recommender Engine receives messages from: (a) blood pressure, (b) blood sugar, (c) EKG, and (d) SPO2 sensors. In the event that the Recommender Engine detects an imminent dangerous state, it produces alert messages and disseminates them in the SIS network.

In illustration 1 one can see the different components inside the Recommender Engine. First, the message parsing interface is responsible for handling input messages and produce alerts in the SIS network. The data transformation logic receives sensor data and turns them into a binary form that is readable by the recommender logic. The latter is responsible for building prediction models in order to implement the prediction logic needed for identifying dangerous situations in a proactive manner. Finally, the Recommender Engine keeps an internal storage module for storing user-defined Rules (conditions under which an alert should be generated) and precomputed Prediction Models.

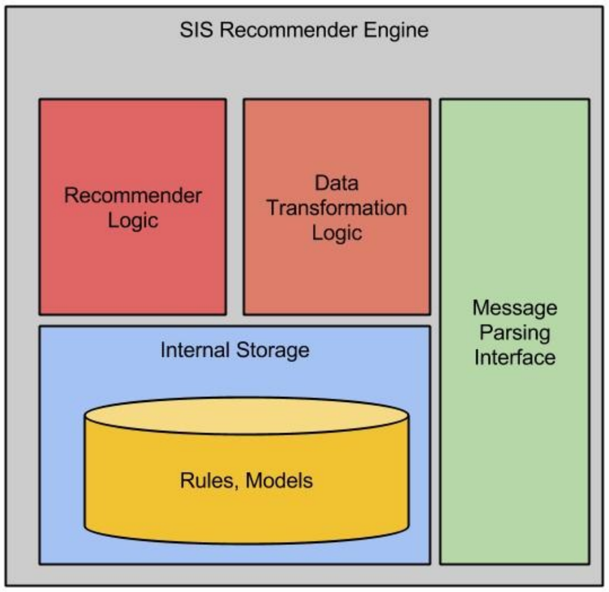
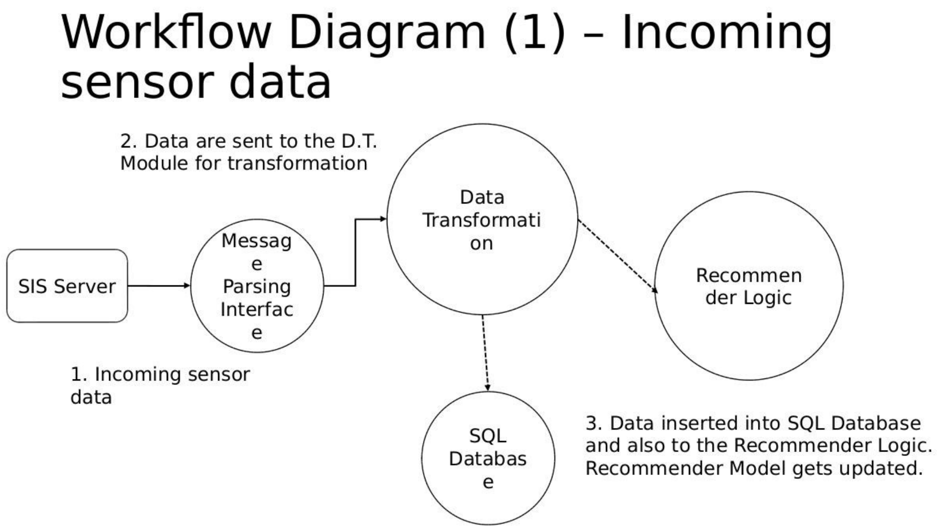
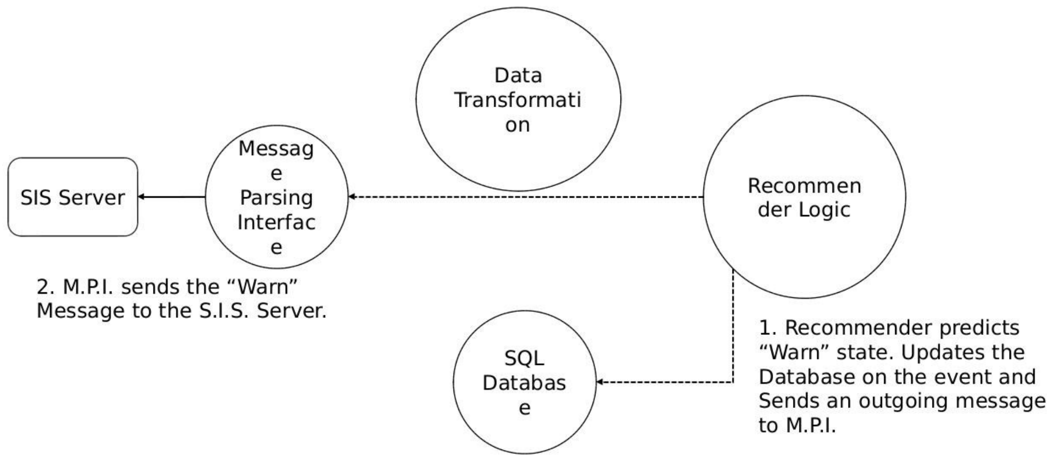


Illustration 1: Recommender Engine architecture

Special attention needs to be paid in the storage module of the Recommender Engine. In it, information in the form of tuples are stored, so that the system is able to predict dangerous situations. In the next section I explain (i) the data model of the Recommender Engine, (ii) the recommender logic, and (ii) the form of Rules provided along with a step-by-step execution of the recommender logic. Below we can see the work flow diagrams of the Recommender Engine:


Workflow Diagram (1) – Incoming sensor data
2. Data are sent to the D.T. Module for transformation
SIS Server
Messag e Parsing Interfac e
Data Transformati on
Recommen der Logic
SQL Databas e
1. Incoming sensor data
3. Data inserted into SQL Database and also to the Recommender Logic. Recommender Model gets updated.

Workflow Diagram (2) – Recommender identifies Warn State


Data Transformation
SIS Server
Message Parsing Interface
Recommender Logic
SQL Database
2. M.P.I. sends the "Warn" Message to the S.I.S. Server.
1. Recommender predicts "Warn" state. Updates the Database on the event and Sends an outgoing message to M.P.I.

Data Model

The Recommender Engine works by using Collaborative-Filtering Algorithms to predict users' preferences. This approach is more generic compared to the Context-based approach of other recommendation systems. Hence, it can be easily modified to work with different scenarios. The only information that need to be stored in the data model are tuples of the form:

user-id, item-id, preference

The user-id refers to a user showing interest (or a general connection) for a specific object (item-id). The interest can be represent any kind of connection among the two entities. For instance, it can represent how much a user likes a product, or it can represent that a user has had a characteristic represented by a specific id (object). Preference models the intensity of the connection of a user with an object. A preference can take values from 1 to 5, but it can also have a binary interpretation if a binary recommendation system is needed (i.e. yes or no answer).

The reason I used this data model in the Recommender Engine is its simplicity. By forming the data accordingly, one can approximate any kind of situation and have the Recommender Engine produce successful predictions.

Recommender Logic

In the Recommender Logic, the following components had to be implemented:

- a similarity metric
- a user-neighborhood implementation

- a user-based recommender

The similarity metric is needed to calculate the similarity among two users. Users need to be placed in a two dimensional space, based on a metric. Many types of similarities can be used in order to simulate different scenarios. In my case I implemented an Euclidean distance metric for two reasons: (a) implementation efficiency and (b) it achieved the best results in my use-case.

Turning to the user-neighborhood, a mechanism of that kind is needed in order to group similar users together (based on the similarity metric). This way, the Recommender Engine can effectively correlate a user's behavior with its closest neighbors, and successfully predict the former user's future actions. In my implementation I used a threshold-based user neighborhood. This type of neighborhood works according to a threshold value given as a parameter (i.e. *t*). Hence, a user's neighborhood is defined by all other users that are not further than *t* from the user's position. This kind of neighborhood proved to achieve the best results in the use-case of the Health-Care system. In the future, more kinds of user-neighborhoods can be tested on the same scenario.

Finally, the user-based recommender is responsible for tying all of the above components together. It makes use of the data model, the similarity metric, and the user neighborhood. Predictions are produced based on the user-based recommendation algorithm, which iterates all objects not connected with a user and calculates a prediction possibility for each one. The top-k objects are returned as possible future connections for the user.

Rules

In order to produce predictions with the Recommender Engine, one should populate the storage with the required rules for predictions. There is no specific algorithm for creating rules. Rather, one has to take history of users and turn it in the data model form. For instance, if we have a rule like the following:

“User X had a connection with objects Y and Z before he connects with object K.”

The following tuples of the data model need to be produced:

X,Y,5
X,Z,5
X,K,5

The above tuples indicate the sequence of connection events happened with user *X* and objects *Y*, *Z* and *K*. The 5 as a preference denotes the strength of the connection and can be completely omitted in a binary user-based approach. Let us assume that a user *U* appears in the system and has a connection only with object *Y*. Then, the data model will have the following additional tuples:

X,Y,5

X,Z,5

X,K,5

U,Y,5

In its current state, the Recommender Engine will not predict an object for *U*, since he is not in *X*'*s* neighborhood yet (the Euclidean distance is still above a reasonable threshold between them). However, if *U* adds a connection for object *Z*, then the data model will become:

X,Y,5

X,Z,5

X,K,5

U,Y,5

U,Z,5

and the Recommender Engine will produce a prediction for a connection between user U and object K. If someone follows the same approach when transforming his current database accordingly, the Recommender Engine will successfully use prediction rules.

Real-life Application: Health-Care SIS System

In the demonstration I presented to the class I have shown the Recommender Engine's use when we need to predict dangerous situations for patients. I have assumed that for a single patient we have a blood pressure, a blood sugar, a SPO2, and an EKG sensor. The Engine's responsibility is to combine readings from the aforementioned sensors, and by consulting user-defined rules, produce alert messages to the system. I have a Youtube video on the link with the interactive steps of the application at: https://www.youtube.com/watch?v=4AomMuTTReo

For the implementation I made use of Apache Mahout library (http://mahout.apache.org) which is a complete framework for recommendation systems. It provides stand-alone and distributed implementations of the basic components needed for a recommendation system. Also, it features implementations of additional Machine-Learning algorithms.

The three scenarios I present on that video differ with each other by using different rules:

1. A patient is in alert if blood pressure and blood sugar are in near-dangerous levels.
2. A patient is in alert if blood pressure and SPO2 levels are in near-dangerous levels.
3. A patient is in alert if blood pressure, blood sugar and SPO2 levels are in near-dangerous levels.

The above scenarios could not be identified by the current SIS components, without communication overhead and the addition of application logic. The Recommender Engine takes care of everything.

The Alert message produced by the Recommender Engine has the following form:

Name	Value
MsgID	64
Descriptiom	Recommender System Alert
AlertType	Recommender Alert
DateTime	Current Date (i.e. “2014-10-30 15:05:10”)

In the Youtube video the user can see the three scenarios live and understand the strength of the Recommender Engine.

Conclusion

In this report I gave a detailed overview of the Recommender Engine as an SIS Component. Also, I provided information about its architecture, its required modules and an overview of its logic. The reader is encouraged to watch the provided video for a live demonstration-explanation of the Recommender Engine and understand its potential. In the future, one can extend the Recommendation Engine to work in other applications also and improve its success rate further.