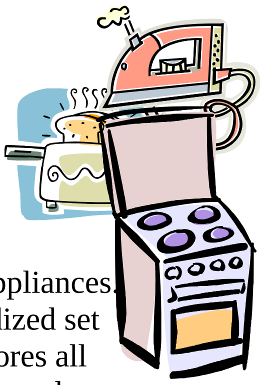
CMPUT-410 Term Project Description: The “Appliance On-line Store”



Release: Part I – January 21th, 2013;
Submission due: week of April 8th, 2013 at demo time
Demo times: during lab time week of April 8th. A schedule will be set.

Brief Introduction: The project is to build an e-commerce site to sell small and major appliances. This is a hypothetical situation where all products sold in all stores intersect in a standardized set of appliances. Each group will deal with one e-store. There will be ten (10) different e-stores all of them selling the same standardized appliances in addition to their own unique products and accessories. The description of the standardized products will be given but the students can create their additional products and their descriptions. The description of appliances should be consistent among teams. The product characteristics are described latter in this document.

Project Description: You are assigned the task of building a web-based application to serve as an appliance e-commerce. The service will offer options to on-line users and will provide activity summaries to management as decision support for a given business. In other words, the web-based application has two parts: one for the buyers to purchase appliances seen by the customers inquiring about products (purchase module); and one for the manager to track the business activity (management module).

1- Purchase Module

This module provides a user with tools to purchase and order appliances and accessories from the on-line store. While each store keeps information in a local database only about this customers, product stock and suppliers, the tools should transparently access the other available stores if requested products are not available locally. How this information about products from other stores will be accessed is explained later in this document. Briefly, if a customer C accesses a store S and requests product P but P is not available in S, the software in S should access other stores automatically and try to get P at the best price and best delivery timeframe for its own customer C without having to inform C.

The tools that this module should provide are:

a- A list of product categories
b- A list of products (that can be navigated)
c- A search capability to search for products.
d- A recommendation capability to recommend products
e- A possibility to order products
f- A possibility to view ones own cart and outstanding orders

2- Management Module

The management module is solely for administration purposes. It should allow a decision maker see some summaries about the purchasing activities. The module should provide simple reporting tools such as:

a- A list of customers for a given time period with aggregated purchase counts and amounts.
b- A list of sold products for a given time period with aggregated purchase counts and amounts.
c- A list of stores for a given time period from which indirect purchases were made with products ordered aggregated purchase counts and amounts.

d- Other decision support reports that you see fit: example, top n selling products for a given time period, etc.

Notice that reports a, b, and c can be consolidated into one interactive report that allows drilling down and rolling up.

Customer and product information should be stored in a database on ORACLE (or mySQL if available). The database should be designed and an E-R model is required. The database should be populated with simulated data. Each group is responsible to add data in their database. An initial list of products will be provided by the TAs. This list is standardized among all groups to allow the purchase of products from one store to the other. Other products and description specific to a unique store are allowed. A list of the content of the database should be provided to the TAs just before the demo of the project.

Information about products (appliances and accessories):
Each product is described using the following features. ***These do not have to be real and are not exclusive.***

- Product code
- Product category
- Product name
- Product brief description
- Product picture
- Product price
- Product weight
- Product dimensions
- Availability in stock
- Other information as you see fit.

You can either have a unique price for a product or adopt a policy that reduces the price based on quantity. These policies should be stored outside the code.

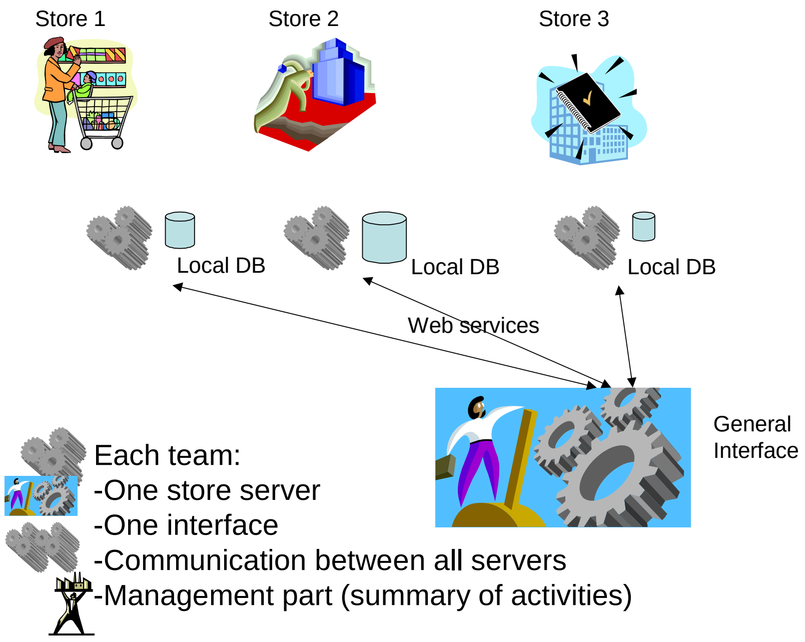
Search capability:
A user should be able to search for a product by code, or by name, by category etc. and specifying constraints such as price, availability, or even weight.

Product recommendation:
The system should be able to recommend products to a user. There are many possibilities for a recommendation agent in this case. You need to implement at least one of them. Here is a list of possible approaches ordered by importance (i.e. mark values).

1- A recommender agent that recommends products based on a failed constraint-based search. Suppose you send a search query specifying constraints such as price range, availability, and weight, and that there is no product that satisfies the set of constraints. The recommender agent could “relax” some constraints such as price or other and provide a list recommending other products that “almost” satisfy the specified constraints. The relaxation of constraints could be done in sequence based on some heuristics or based on preferences (i.e. weights) the user provides on each constraint. Think of a customer profile.

2- A recommender agent that recommends products based on ratings. If users have the possibility to rate products they purchase, the agent could recommend products to a given user A based on the ratings A previously provided and the matched ratings of other users. This recommendation system relies on the collaborative filtering algorithm.
3- A recommender agent that recommends products based on frequency of past purchase. Suppose a user A purchased in the past product1 and product2 and other people who bought product1 and product2 also purchased product3, the agent could recommend product3 to user A. This recommendation system relies on association rule mining.

Communication between businesses:
While each group deals with one store, the interface should allow purchasing an item that is not available in the store but in other stores without hinting the non-availability to the user.
In other words, each group stores information about their own stock in the store but provides the service to access information from other stores and transparently order from other stores as if the product was available locally. To access the information from other stores, you should use web services. A registry will be provided by the TAs (check the moodle announcements for more details and schedule). Basically, we will have 3 available services: *GetProductlList* will query for the product list in a given store with or without some specifications; *Orderproduct* will ask to purchase a product; *CheckOrder* will check the order status of an order previously done. Each group that starts running their server should register their service with the registry. To know the available services (i.e. stores), a simple request to the registry would provide the list of services alive. That same list would provide the address and communication port for each service. To communicate with a given web service, a communication protocol will be specified using a 410 standard WSDL (web service definition language) file. This file will be provided by the TAs and discussed on the moodle forum for final adjustments if need be. More information about the services, the parameters, the data transferred, the format and the WSDL per se will be provided shortly (check course moodle page).



Groups: The project must be solved in groups of 4 students. Exception from this rule (i.e. 1-3 student group) is accepted if the number of students is odd, and only applies to one group.

Note about the management module:
There is no need for web services when dealing with the management module. The management module will only access the local database for its products and its own customers.

Deliverables: Each group must submit the following:

- **project report:** explain how the solutions were implemented, all components (classes/methods) used and relationships between them. Explain your assumptions and simplifications. Present your algorithms for the recommender systems you implemented. List the used queries, explain how they were assembled to get the report for the management module (rough algorithm) and attach a snapshot of the reports and the user interface of the purchasing module. Attach the entity-relationship model of your data in the database to your report. Enumerate specific features added to your implementation (JavaScript enhancements etc.)
- **DO:** use a cover page for the project report with your name(s) and unixID(s), lecture and lab sections.
- **DON'T:** write your studentID(s) in the report or any attached document.

Demonstration: Each group will have a maximum of 20 minutes to demo the project. The TA needs to have access to your database content before the demo. Make sure the project runs well on the machines in the lab, and displays correctly your project web pages using the Firefox browser available in the lab machines. *You will not be allowed to make any modifications to your project at demo time.*

Marking: The marks for this project are divided as follows:

- Demonstration (interface usability, interactivity etc. functionality, etc.): 75 marks
- Project Report: 25 marks

NOTE: As you develop the project with other students, the marks given for the project will be the same for all four students, even if one of the students fails to deliver his/her part. Hence, choose you partners wisely. All partners must be present at demo time otherwise zero marks will be given to the demo component of the project to the absent partner.

Submission: Submission will be done via moodle. Create a tar file including the project report, <u>all</u> source code (no compiled class files) and the Makefile. *You will be allowed to make changes to your code until the demo day but not add new functionalities. You can still attach new snapshots of your user interface at demo day if you decide to improve your CSS for instance.* Test your tar file before submitting it and make sure it contains all required files for your project to run. Additional instructions for the submission process will be added later on on moodle.

Frequent Asked Questions:

1- What technology can I use for implementing my project?
You can use Java servlets, JSP, Perl script CGI, C/C++ CGI, PHP Python. Other technologies available in the lab are also possible.

2- Can I implement my project at home and run it from my server at home?

Yes you can. However, at demo time, you need to execute the demonstration from the lab. You also need to provide your source code with the submission.

3- Can I use another DBMS?

Yes you can use any relational DBMS: ORACLE, DB2, Postgress, msql, SQL-server, etc. Microsoft Access is not considered a RDBMS and cannot be used for this project.

4- Will the look-and-feel of the implementation count or is it only the functionality of the project that counts?

The user interface will count. While it won't count for very much in comparison to other aspects of the project, the look and feel is what the user sees and it is what will distinguish between implementations. In web-based applications, the interface is paramount.

NB: if there are modifications to these descriptions, the modifications will be clearly marked using colours or other means. The last modification date is indicated in the beginning of the document.