DATA PEAR SYSTEMS

Danielle Lottridge

Simon Hatch

Joshua Collings

Detailed Design: The HMV Information System Interface

CSC 340S

Monday, April 16th, 2001

TA: Afsaneh Fazli

Table of Contents

Introduction:

The objective of this report is to provide a detailed design for the HMV Kiosk Search system as proposed by the Requirements Analysis Phase (see Appendix D). This detailed design includes our specification of our selection of hardware, networking, and software, for our system. The program design consists of: a three-tiered architecture composed of an Interface, an Application, and a Database component. These components are described in class diagrams. The functionality of our system is described with sequence diagrams and statechart diagrams. Analysis was done on the database component to create the most efficient and cost effective solution. We emphasized the creation of the interface as described in our class and statechart diagrams, so that it is user-friendly and easy to use.

Part A: Global Architecture

Network

Our system will interface with the current system using a network card and twisted-pair cabling. The current system is a Local Area Network running off an IBM AS/400 server using the IPX/SPX standard protocol, with a portal to the Wide Area Network administered by the HMV head office (which uses TCP/IP protocol for its inter-network communication). There is an additional network in the store, the transaction handling system, which runs from an IBM AS/400 "Cash-Controller series" server, but the new system does not interact with this network.

Hardware

The key areas of concern for the hardware of the new system are speed, durability, economy and storage.

The most important component of the hardware is the touch-screen. All other aspects of the system will be dependent upon the requirements of the screen chosen. The top three considerations for the screen vendor are listed in the table below.

Vendor	Price (adj. Price)*	Durability	Aesthetics	Complete System	Customisable
EZScreen	N/A ($1800)	Excellent	Good	Yes	Yes
Acma	$575 ($1375)	Poor	Good	No	Yes
Mass Multimedia	$879 ($1679)	Good	Poor	No	Yes

**Price adjustment reflecting the inclusion of all additional components required for the system, i.e. the CPU, memory, etc.*

It is the opinion of this report, after considering many options, that purchasing complete RxKiosk systems from Ezscreen is the best alternative.
The RxKiosk system addresses the criteria in the following areas:

- Speed and storage requirements – The system has a 533 Celeron processor, 64 Mb of RAM and a 10.4 Gb Hard-drive, which exceeds all the speed and storage requirements.
- Durability requirements - The RxKiosk system's physical design is ideally suited to the demanding environment of a retail store.
- Economy requirements - Ezscreen's willingness to tailor the systems to HMV and give quantity discount pricing makes them a very economical choice.

Software

The two software components that will be used in our design and are commercially available are the operating system, and the web browser. Various options were considered for both with the results summarized below.

Operating System

The two main possibilities for consideration as operating systems are tabulated below.

Operating System	**GUI**	**Stability**	**Price**	**Maturity**	**Driver Avail.**
Windows 2000	Very Good	Good	No cost**	7.5 years (NT)	Excellent
Linux	Average	Excellent	Free	9.5 years	Good *

** Depends on distribution.*
*** HMV has an existing corporate license for Windows 2000*

Based on these criteria, Windows is the best operating system for use on the kiosks. Windows 2000 is fast, reliable (99.999% advertised "up time"), and extremely easy to maintain. Additionally, many users would be more comfortable working in a Windows environment, since the majority of home users run some version of Windows on their computer(s).

Web Browser

There are several different browsers available for Windows and Linux platforms. Among the best known are Internet Explorer, Netscape, and Mozilla. Each browser has its strengths and weaknesses, including ease of use, familiarity, and stability. The analysis of the browsers does not lend itself to a tabular comparison.

Internet Explorer 5.5

Internet Explorer is produced by Microsoft Inc. It is packaged with the Windows operating system.

Advantages: Internet Explorer is reliable, fast, easy to use and cost-effective.

IE has been proven to be an exceedingly reliable software package. (see Appendix A)

Speed is another advantage with Internet Explorer. It loads web pages very quickly, and runs Java faster than other browsers.

Customers will easily adopt IE because it is the most commonly used browser by home users. Its familiarity allows the users to feel comfortable and empowered. Many features are included to make the program more user-friendly, including "Autocomplete" which is a feature that remembers urls, passwords and form text. Also, autocomplete automatically adds or completes entries for you.

Lastly, Internet Explorer is freeware.

Disadvantages: IE has poor portability and has various bugs.

IE is restricted to the Windows platform.

IE’s bugs include but aren’t restricted to the following: fonts are often oversized, bold is overused, font colours are inconsistent, and table formatting is poorly implemented.

Netscape 6.01

Netscape Navigator is produced by Netscape Communications Corporation. It is distributed over the internet.

Advantages: Netscape is familiar to users, stable and portable.
Netscape is widely used and thus well-known to users.

Netscape is reliable when compared with other browsers, although not as reliable as IE.

It is also available for many different platforms, including Windows 9x/NT, many versions of UNIX, and Linux.

Disadvantages: Netscape is relatively slow, has a substantial cost, does not conform to protocol standards and is not memory efficient.

When compared with Internet Explorer, Netscape does not load graphics and large pages as quickly.

Netscape is free for personal use only. Commercial use of the software requires an expensive licensing fee.

In Netscape, Javascript and DHTML implementations do not conform to established standard; as a result, some pages do not load correctly.

Netscape is a memory intensive application. It requires more memory to run than IE.

Mozilla 0.8.1
Mozilla is an open-source web browser, designed for standards compliance, performance and portability.

Advantages: The advantages of Mozilla are that it is flexible, portable and aesthetically pleasing.

Mozilla is flexible because is that it is open-source and easily modified. The browser is constantly under development, and problems can be fixed without having to wait for the next patch.

Mozilla is portable, and like Netscape, it is available on Linux or Windows.

Mozilla's user interface is streamlined and easy to use, and the browser is "skinnable", allowing for a customised look.

Disadvantages: The disadvantages of Mozilla are its immaturity, and its speed.

Mozilla is a relatively new project. The latest version released is 0.8.1. It has not had much time to develop and thus it can be very unreliable.

Mozilla loads web pages slower than Internet Explorer or Netscape.

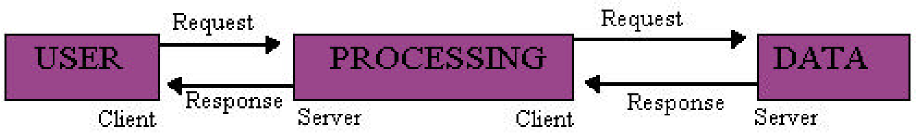
Conclusion of Web Browser Comparison

In the previous section, we had chosen Windows 2000 to be our Operating system. Based on this choice of OS, and on the abundant advantages of Internet Explorer over other browsers, the best choice of browser would be Internet Explorer.

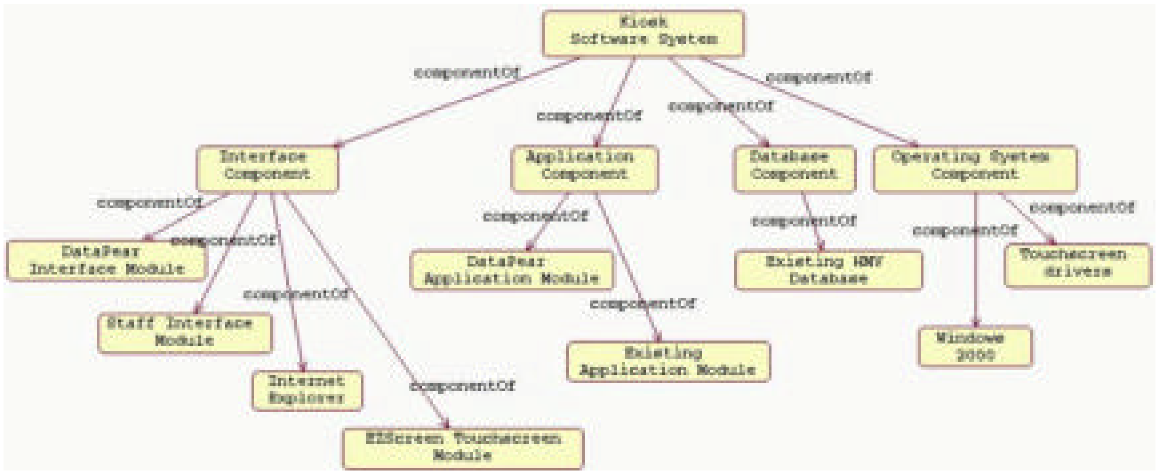
Software Architecture

The kiosk system will be integrated with the existing HMV system, which uses Three-Tier Client/Server Architecture design. The kiosks will be an addition to the interface (or user) tier, acting as clients to the stores local server (processing or application tier).

This architecture was chosen because, in general, client-server is simpler to implement and maintain, as the sub-systems are less tightly coupled then they are when peer-to-peer or other similar architectures are used.



Sub-Systems and Major Components



Description of Sub-Systems and Major Components

The HMV Kiosk Software System has four main components: the Interface Component, the Application Component, the Database Component, and the Operating System Component.

The Interface Component is composed of four modules: the DataPear Interface Module, the Staff Interface Module, the Internet Explorer module and the EZScreen Touchscreen Module. The DataPear Interface Module consists of the interface classes that DataPear will create for HMV. The Staff Interface Module exists already, and provides assistance in interfacing with the existing HMV Application and Database modules. The Internet Explorer module allows the interface to exhibit browser functionality. The EZScreen Touchscreen Module is a software module included in the purchase of the EZScreen Touchscreen.

The Application Component has two sub-systems: the DataPear Application Module and the existing Application Module. The DataPear Application Module comprises the application classes that DataPear will write for HMV. These classes will work with the existing Application Module to provide quick and efficient searches for users.

The Database Component of the Kiosk Software System is simply the existing HMV Database.

The Operating System Component is composed of the Windows 2000 software package, in addition to the EZScreen Touchscreen drivers that are included in the purchase of the EZScreen Touchscreen.

Justification That the Design Meets the Relevant Requirements

The relevant design requirements for the Global Architecture of the system would be:

1. **Interface Requirements**
   The system must be user friendly with an intuitive and unambiguous interaction mechanism. The system must have a familiar design so customers will feel comfortable using the system.
2. **Efficiency Requirements**
   The system must be fast because customers can be very impatient.
3. **Durability Requirements**
   The system must be able to survive heavy daily use, and be resistant to petty vandalism.
4. **Lifecycle Requirements**
   There is no foreseeable obsolescence of the service provided by the system so a long life span is required. The system must be able to have both the hardware and the software upgraded.
5. **Economic Requirements**
   The system must be implemented in over one hundred stores, which means any savings that do not detract from the quality of the system are of great value. Additionally the system should have a low maintenance and replacement cost for the same reason.

The Interface requirements are addressed in a few ways: by having the touch screen input device users have a simple and unambiguous input device and by using a browser style program users have a software interface they are familiar and comfortable with. Additionally, as Windows is the dominant OS of home users, customers will feel more comfortable using a Windows based system.

The Efficiency Requirements are met because the system is actually overpowered for the operations it will be performing; there should be no lag in system performance.

The Durability Requirements are met because the RxKiosk system uses bulletproof plastic in its construction, it should be able to withstand daily use and any minor acts of vandalism.

The Lifecycle Requirements of the system are met because all the components of the RxKiosk can be upgraded, and our software design is focused on ease of development and maintenance with the eventual replacement of the local stores system in mind.

The Economic Requirements are met because although the RxKiosk is slightly more expensive than the other systems, the increased durability it has compared to the other systems makes it a more cost-effective system in the long run.

Part B: Program Design

The following section outlines the design of the system using Class diagrams, Sequence diagrams and State diagrams. All diagrams are preceded by a general description and followed by the relevant definitions from the Data Dictionary for this system.

Class diagrams

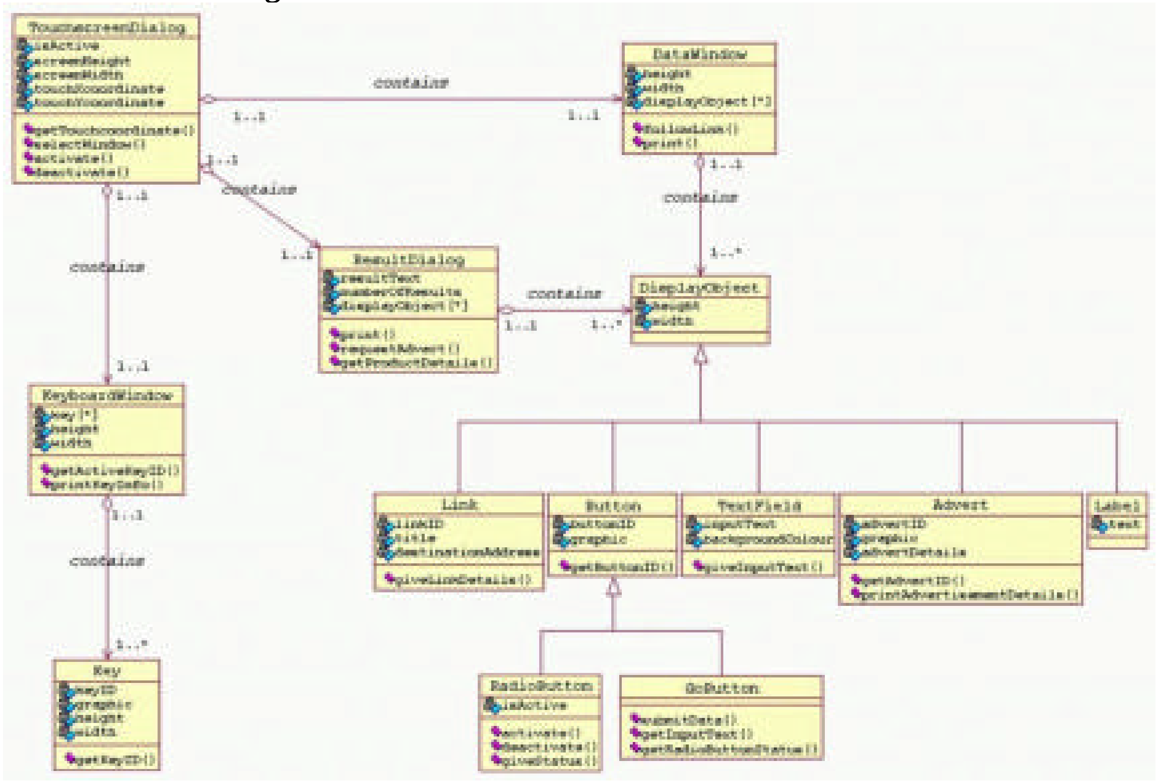
There are three relevant Class diagrams, the Interface class diagram, the Application class diagram and the Database class diagram.

Description for the Interface Classes

The initial screen that the user sees is a welcome screen displayed from the DataWindow class. The user can touch the screen, and the TouchscreenDialog class detects this. Then, a search screen is shown with a keyboard displayed from the KeyboardWindow class. The user can touch the keyboard, which is broken up into Key: LetterKey and FunctionKey. Selecting these keys will input text into the Textfield, saved in its InputText attribute. To complete the search, the user must select one of the RadioButtons (specifying the type of search), and select the GoButton. The top half of the screen that comprises these elements is the DataWindow. There is an Advert displayed in this portion of the screen.

Upon selecting the GoButton, the user will see the results of their query (generated from the ResultDialog class). The results display also shows a new Advert based on the results of the search.

Interface Class Diagram



Glossary of Interface Classes

TouchScreenDialog - *represents a generalization of the physical touchscreen with abilities to analyze user input.*

Public Attributes:

isActive : boolean
Stores whether or not the touchscreen is active.
screenHeight : Integer
The vertical resolution of the screen.
screenWidth : Integer
The horizontal resolution of the screen.
touchXcoordinate : Integer
The x-coordinate of a user touch.
touchYcoordinate : Integer
The y-coordinate of a user touch.

Public Operations

getTouchcoordinate()
Operation to determine the screen coordinates of a touch, when the screen is touched.
selectWindow(Integer x, Integer y)
Operation to determine which window is being accessed.
activate()
Operation to activate the touchscreen.
deactivate()
Operation to deactivate the touchscreen.

KeyboardWindow - *represents the software application emulating an on-screen keyboard for the touch screen.*

Public Attributes:

key[*] : Key[]
An array of all the keys of the keyboard.
height : Integer
Vertical height of the KeyboardWindow, in pixels.
width : Integer
Horizontal width of the KeyboardWindow, in pixels.

Public Operations

getActiveKeyID(Integer x, Integer y) : Integer
Returns the ID of the key at coordinates (x,y).
printKeyInfo(Integer id)
Prints the key's info.

Key - *represents a software emulated key utilized by the KeyboardWindow.*

Public Attributes:

keyID : Integer
The key's ID.
graphic : Graphic
The key's graphic.
height : Integer
The vertical height of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.

Public Operations

getKeyID() : Integer
Returns the key's ID.

DataWindow - *represents the browser used to perform searches, and display results.*

height : Integer
The vertical height of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.
displayObject[*] : DisplayObject[]
An array of objects being displayed on the screen.

Public Operations

followLink(String link)
Operation to follow a hyperlink.
print()
Operation to print information to the display area.

Link - *represents a hyperlink.*

Public Attributes:

linkID : Integer
The link's ID.
title : String
The link's title.
destinationAddress : String
The destination of the link.

Public Operations

giveLinkDetails()
Returns the details of the address being linked.

Label - *represents a label, which is printed by DataWindow.*

Public Attributes:

text : String
The label's text.

DisplayObject - *represents an object that is to be displayed in the DataWindow.*

Public Attributes:

height : Integer
The vertical heigh of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.

Advert - *represents an advertisement banner which can be displayed in the DataWindow.*

Public Attributes:

advertID : Integer
The advertisement's ID.
graphic : Graphic
The advertisement's banner graphic.
advertDetails : String
The advertisement's description.

Public Operations

getAdvertID(String genre)
Gets an advertisement for a specific genre.
printAdvertisementDetails()
Prints the advertisement's details to DataWindow.

TextField - *represents text-field which the user types into.*

Public Attributes:

inputText : String
The text in the text-field.
backgroundColour : Integer
The background colour of the text-field.
Button - *represents a generalization of a button to be displayed on the DataWindow.*
buttonID : Integer
The button's ID.
graphic : Graphic
The button's graphic.

Public Operations

getButtonID() : Integer
Returns the button's ID.

RadioButton - *a specialization of the Button superclass, used to present choices for different searches*.

Public Attributes:

isActive : Boolean
Holds whether or not this radiobutton has been selected.

Public Operations

activate()
Activates the button.
deactivate()
Deactivates the button.
giveStatus() : Boolean
Returns the status of the button.

GoButton - *a specialization of the Button superclass, used to initiate searches.*

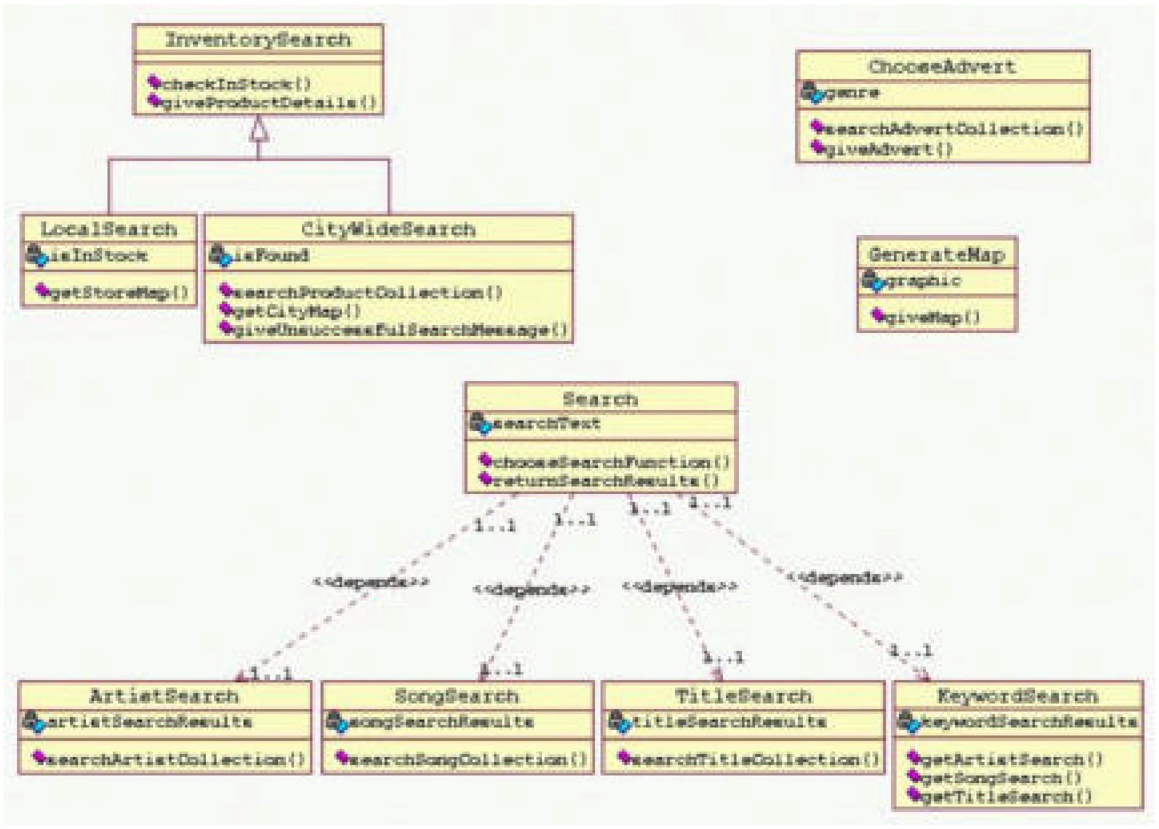
Public Operations

submitData(String data)
Submits data for a search.
getInputText()
Gets the input text from the input-field.
getRadioButtonStatus()
Gets the status of the radio buttons.

Description for the Application Classes

The main process that our HMV kiosk system supports is searching. The user can perform four types of searches: an artist search, a song search, a title search and a keyword search. Each search type is represented by a class. Search is dependent upon these four classes because Search's implementation is dependent on being able to call the operations of the other four search classes.

The search returns some general information about products that are the best matches to the user's query. The user selects one of these products to view more product details. These additional details consist of the products's inventory and a map of the product's location. The inventory of the product is found by the LocalSearch class. If this product is not available in the store then this search returns unsuccessful. Then, a search is initiated for the product's inventory in all HMV stores in the city. The CityWideSearch class performs this search. Once the inventory is found, in the store, or in several other stores, the information is sent to the GenerateMap class. This class generates a map that displays the location of the product.



Glossary of Application Classes

Inventory Search - *represents the software application used by the kiosk system to check for the existence of products.*

Public Operations

checkInStock(String product) : boolean

Return true if the product is in the inventory collection.

giveProductDetails() : Product,Quantity,Map

Returns the details of a product, and a map to where it can be found.

LocalSearch - *a specialisation of the InventorySearch software, used to search the local store's inventory.*

Public Attributes:

isInStock : Boolean

Stores whether or not the product is in stock.

Public Operations

getStoreMap(String store) : Map

Returns a map of the store.

CityWideSearch - *a specialisation of the InventorySearch software, used to search the local area inventory.*

Public Attributes:

isFound : Boolean

Stores whether or not a product is in stock in any store in the local area.

Public Operations

searchProductCollection(String product)

Searches the master database for a product.

getCityMap(String city) : Map

Returns a map of the city.

ChooseAdvert - *represents the software application used by the kiosk system to retrieve advertisements.*

Public Attributes:

Genre : String

The advertisement's target genre.

Public Operations

searchAdvertCollection(String genre)

Search the database for an advertisement geared towards genre.

giveAdvert() : Advert

Returns an advertisement.

GenerateMap - *represents the software application used by the kiosk system to create maps.*

Public Attributes:
graphic : Map
The map to be returned.

Public Operations
giveMap() : Map
Returns a map.

Search - *represents the software application used by the kiosk system to perform searches. Search's implementation depends on being able to call on the* ArtistSearch, SongSearch, TitleSearch, and KeywordSearch applications.

Public Attributes:
searchText : String
The text which is being searched for.

Public Operations
ChooseSearchFunction(int choice)
Calls the appropriate search function, based on "choice".
returnSearchResults() : String
Returns the results of a search.

ArtistSearch - *represents the software application used to perform searches on the artist database.*

Public Attributes:
artistSearchResults : String
Holds the results of an artist search.

Public Operations
searchArtistCollection(String artist)
Operation to search for an artist in the database.

SongSearch - *represents the software application used to perform searches on the song database.*

Public Attributes:
songSearchResults : String
A string holding the results of an song search.

Public Operations
searchSongCollection(String song)
Operation to search for an song in the database.

TitleSearch - *represents the software application used to perform searches on the title database.*

Public Attributes:

titleSearchResults : String
A string holding the results of an title search.

Public Operations

searchTitleCollection(String title)
Operation to search for an title in the database.

KeywordSearch - *represents the software application used to perform searches on the artist, song, and title databases.*

Public Attributes:

keywordSearchResults : String
A string holding the results of an keyword search.

Public Operations

getArtistSearch(String artist)
Operation to get the results of an artist search.
getSongSearch(String song)
Operation to get the results of an song search.
getTitleSearch(String title)
Operation to get the results of an title search.

Description for the Database Classes

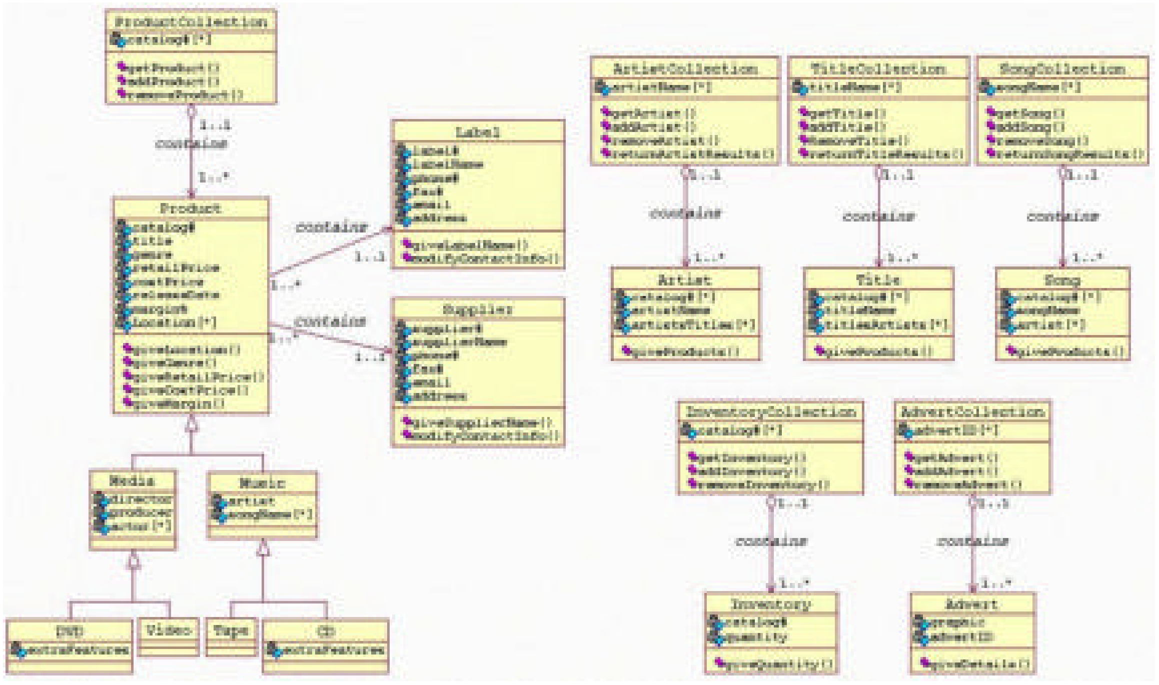
All classes with the suffix "Collection" are indices of the object designated by the prefix. The Collection classes exist to facilitate searching through products, artists, etc...

HMV sells media and music products: DVD's, videos, tapes, and CD's. Each of these products is represented by a class. When a customer searches for a product, they are trying to search for it using its artist, its title, or a particular song. Artist, song, and title classes exist to optimize the searching operations. These objects hold the catalog# of the particular products so that the product's details can be accessed through the product object.

There is also an inventory item associated with each product where inventory information is stored.

Each artist, and hence music product, has an associated label and supplier. These objects are not indexed because searches are seldom performed on them.

Finally, adverts are displayed with search results. Advert information is also held in this database.



Glossary of Database Classes

ProductCollection - represents a database of products, sorted by catalog#.

Public Attributes:

Catalog#[*] : Integer[*]
An array of catalog#'s.

Public Operations

getProduct(String product)
Returns product if it exists in the product database.
addProduct(Product product)
Adds a new product to the database.
removeProduct(Integer catID)
Removes a product from the database.

Product - a generalization of records stored in the ProductCollection database.

Public Attributes:

Catalog# : Integer
The product's catalog number.
Title : String
The product's title.
Genre : String
The product's genre.
RetailPrice : Float
The product's retail price.
CostPrice : Float
The amount the supplier charges for the product.
ReleaseDate : Time
The release date of the product.
Margin% : Float
CostPrice / RetailPrice
Location[*] : String[*]
A list of locations the product is available at.

Public Operations

giveLocation() : String[*]
Returns a list of locations.
giveGenre() : String
Operation to return the genre of the product.
giveRetailPrice() : Float
Operation to return the retail price of the product.
giveCostPrice() : Float
Operation to return the cost price of the product.
giveMargin() : Float
Operation to return the margin% of the product.

Media - *a specialisation of the Product superclass. Used to store information about media products in the product database.*

Public Attributes:

Director : String
The product's director.
Producer : String
The product's producer.
Actor[*] : String[*]
A list of actors that appeared in this product.

DVD - *a specialisation of the media class. Used to store information about DVD products in the product database.*

Public Attributes:

extraFeatures : String
A description of the extra features included on the DVD.

Video - *a specialisation of the media class. Used to store information about videos in the product database.*

Music - *a specialisation of the Product superclass. Used to store information about music products in the product database.*

Public Attributes:

Artist : String
The product's artist.
SongName[*] : String[*]
A list of the songs.

Tape - *a specialisation of the Music class. Used to store information about tapes in the product database.*

CD - *a specialisation of the Music class. Used to store information about CD's in the product database.*

Public Attributes:

extraFeatures : String
A description of the extra features included on the CD.

InventoryCollection - *represents a database of store inventories.*

Public Attributes:
C**atalog#[*]** : Integer[*]
An array of catalog#'s.

Public Operations
getInventory(Integer catID) : Integer
Returns the quantity of a product in stock.
addInventory(Integer catID)
Increases the quantity of a product in stock.
removeInventory(Integer catID)
Decreases the quantity of a product in stock.

Inventory - *a generalisation of records stored in the InventoryCollection database.*

Public Attributes:
Catalog# : Integer
The catalog number of this inventory.
Quantity : Integer
The quantity of this inventory in stock.

Public Operations
giveQuantity() : Integer
Returns the number of a particular product that are in stock.

ArtistCollection - *represents a database of artists, sorted by ArtistName.*

Public Attributes:

ArtistName[*] : String[*]
An array of artists.

Public Operations

getArtist(String artist)
Operation to find an artist in the database.
addArtist(Artist artist)
Operation to add an artist to the database.
removeArtist(Artist artist)
Operation to remove an artist from the database.
returnArtistResults() : String
Operation to return the results of an artist search.

Artist - *a generalization of records stored in the ArtistCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products this artist has released.
ArtistName : String
The artist's name.
ArtistsTitles[*] : String[*]
An array of the titles of the products this artist has released.

Public Operations

giveProducts() : Product[*]
Returns a list of product's associated with this artist.

TitleCollection - *represents a database of titles, sorted by TitleName.*

Public Attributes:

TitleName[*] : String[*]
An array of titles.

Public Operations

getTitle(String title)
Operation to find a title in the database.
addTitle(Title title)
Operation to add a title to the database.
removeTitle(Title title)
Operation to remove a title from the database.
returnTitleResults() : String
Operation to return the results of a title search.

Title - *a generalization of records stored in the TitleCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products with the same title.
TitleName : String
The title.
TitlesArtists[*] : String[*]
An array of the artists associated with this title.

Public Operations

giveProducts() : Product[*]
Returns a list of product's associated with this title.

SongCollection - *represents a database of songs, sorted by SongName.*

Public Attributes:

SongName[*] : String[*]
An array of songs.

Public Operations

getSong(String song)
Operation to find a song in the database.
addSong(Song song)
Operation to add a song to the database.
removeSong(Song song)
Operation to remove a song from the database.
returnSongResults() : String
Operation to return the results of a song search.

Song - *a generalization of records stored in the SongCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products containing this song.
SongName : String
The song's name.
Artist[*] : String[*]
An array of the artists associated with this song.

Public Operations

giveProducts() : Product[*]
Returns a list of product's associated with this song.

AdvertCollection - *represents a database of advertisements.*

Public Attributes:
AdvertID[*] : Integer[*]
An array of advertisement IDs.

Public Operations
getAdvert(String genre)
Operation to find an advertisement in the database, for a particular genre.
addAdvert(Advert advert)
Operation to add an advertisement to the database.
removeAdvert(Advert advert)
Operation to remove an advert from the database.

Advert - *a generalization of records stored in the AdvertCollection database.*

Public Attributes:
Graphic : Graphic
The advertisement's banner.
AdvertID : Integer
The advertisement's ID.

Public Operations
giveDetails() : String
Returns a description of the advertisement.

Supplier - *a generalization of records stored in the database, used to store information specifically about suppliers.*

Public Attributes:
Supplier# : Integer
The supplier's ID.
SupplierName : String
The supplier's name.
Phone# : Integer
The supplier's phone number.
Fax# : Integer
The supplier's fax number.
Email : String
The supplier's email address.
Address : String
The supplier's address.

Public Operations
giveSupplierName() : String
Operation to return the supplier's name.
modifyContactInfo(Integer sID, String name, Integer phone, Integer fax, String email, String address)
Operation to change information stored about a supplier.

Supplier - *a generalization of records stored in the database, used to store information specifically about labels.*

Public Attributes:

Label# : Integer
The label's ID.
LabelName : String
The label's name.
Phone# : Integer
The label's phone number.
Fax# : Integer
The label's fax number.
Email : String
The label's email address.
Address : String
The label's address.

Public Operations

giveLabelName() : String
Operation to return the label's name.
modifyContactInfo(Integer sID, String name, Integer phone, Integer fax, String email, String address)
Operation to change information stored about a label.

Justification That the Class Diagrams Meet the Relevant Requirements

The Relevant Functional Requirements are:

1. Processing Requirements
   The system must be able to translate and relay queries to the stores existing database, and translate and relay the results of the query back to the user.
2. Input Requirements
   The system must be able to accept queries from users and must be able to receive information from the stores existing database.
3. Output Requirements
   The system must be able to display the results of queries immediately to the user, as well as relevant location information.

The class diagrams demonstrate the processing requirements are met in the following ways: there are various search classes representing search applications in the Application Tier, and there are classes in the interface with operations to submit data to the Application Tier.

The GoButton Class, has a submitData() operation which passes the search string, and the search choice, to the Search Class in the Application Tier. The Search Class is able to choose the appropriate search function to call using chooseSearchFunction(), and can pass the results back to the ResultsDialog Class in the Interface Tier using returnSearchResults().

The class diagrams demonstrate the input requirements are met in the following ways: they show that functions exist to handle any sort of input from the user, and handle any sort of input from the store's existing database.

The TouchscreenDialog Class has the function getTouchcoordinate() to get the coordinates of a user's touch, and can correctly select which window to pass the coordinates to, using selectWindow(). The KeyboardWindow() can take coordinates and correctly choose which key has been pressed using getActiveKeyID(). The RadioButton Class can successfully be chosen by activate(). The GoButton Class can successfully submit data using submitData(). Input from the server is handled by the ResultsDialog Class using getProductDetails().

The class diagrams demonstrate the output requirements since they contain functions to display output to the user. All input from the server is displayed by ResultsDialog Class using the print() operation.

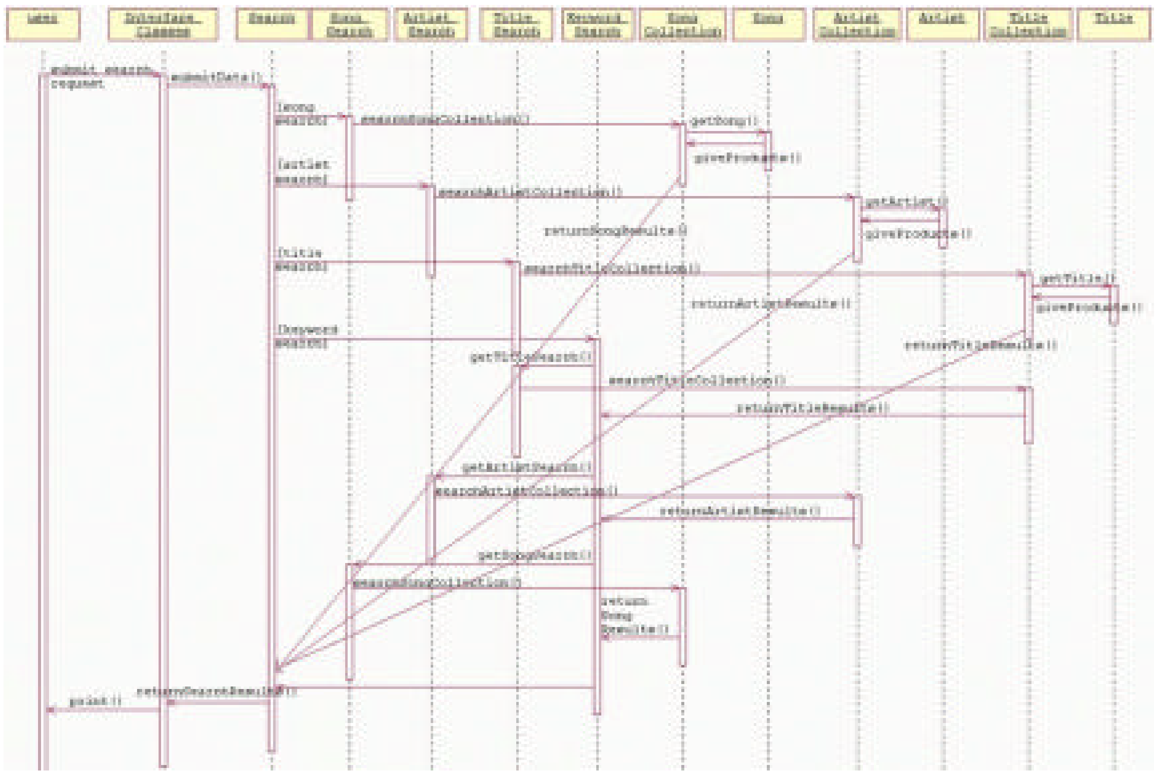
Sequence diagrams

There are four sequence diagrams describing the Product Search Sequence, the Interface Sequence, the Product Details Sequence, and the Product Selection Sequence.

Description of the Product Search Sequence Diagram

The users first submit their search to the Interface Classes. The Interface Component then submits the search Data to the Search class in the Application Component. Depending on the type of search, either the SongSearch, ArtistSearch, TitleSearch or KeywordSearch class performs the search. The designated search classes iterates through the corresponding objects (Artist, Title or Song) held in the Database through their Collection class (either ArtistCollection, SongCollection or TitleCollection). If the user chooses to perform a Keyword search, then the KeywordSearch class cycles through the Collection classes of all the types of objects. The Collection class then returns the search results, to the main Search classes. The Search class then sends these results to the Interface Component to be displayed to the users.

The Product Search Sequence Diagram



Glossary of Classes within Sequence Diagram

GoButton - *a specialization of the Button superclass, used to initiate searches.*

Public Operations

submitData(String data)
Submits data for a search.
getInputText()
Gets the input text from the input-field.
getRadioButtonStatus()
Gets the status of the radio buttons.

Search - *represents the software application used by the kiosk system to perform searches. Search's implementation depends on being able to call on the* ArtistSearch, SongSearch, TitleSearch, and KeywordSearch applications.

Public Attributes:

searchText : String
The text which is being searched for.

Public Operations

ChooseSearchFunction(int choice)
Calls the appropriate search function, based on "choice".
returnSearchResults() : String
Returns the results of a search.

ArtistSearch - *represents the software application used to perform searches on the artist database.*

Public Attributes:

artistSearchResults : String
Holds the results of an artist search.

Public Operations

searchArtistCollection(String artist)
Operation to search for an artist in the database.

SongSearch - *represents the software application used to perform searches on the song database.*

Public Attributes:

songSearchResults : String
A string holding the results of an song search.

Public Operations

searchSongCollection(String song)
Operation to search for an song in the database.

TitleSearch - *represents the software application used to perform searches on the title database.*

Public Attributes:

titleSearchResults : String
A string holding the results of an title search.

Public Operations

searchTitleCollection(String title)
Operation to search for an title in the database.

KeywordSearch - *represents the software application used to perform searches on the artist, song, and title databases.*

Public Attributes:

keywordSearchResults : String
A string holding the results of an keyword search.

Public Operations

getArtistSearch(String artist)
Operation to get the results of an artist search.
getSongSearch(String song)
Operation to get the results of an song search.
getTitleSearch(String title)
Operation to get the results of an title search.

ArtistCollection - *represents a database of artists, sorted by ArtistName.*

Public Attributes:

ArtistName[*] : String[*]
An array of artists.

Public Operations

getArtist(String artist)
Operation to find an artist in the database.
addArtist(Artist artist)
Operation to add an artist to the database.
removeArtist(Artist artist)
Operation to remove an artist from the database.
returnArtistResults() : String
Operation to return the results of an artist search.

Artist - *a generalization of records stored in the ArtistCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products this artist has released.
ArtistName : String
The artist's name.
ArtistsTitles[*] : String[*]
An array of the titles of the products this artist has released.

Public Operations

giveProducts() : Product[*]

Returns a list of product's associated with this artist.

TitleCollection - *represents a database of titles, sorted by TitleName.*

Public Attributes:

TitleName[*] : String[*]

An array of titles.

Public Operations

getTitle(String title)

Operation to find a title in the database.

addTitle(Title title)

Operation to add a title to the database.

removeTitle(Title title)

Operation to remove a title from the database.

returnTitleResults() : String

Operation to return the results of a title search.

Title - *a generalization of records stored in the TitleCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products with the same title.
TitleName : String
The title.
TitlesArtists[*] : String[*]
An array of the artists associated with this title.

Public Operations

giveProducts() : Product[*]
Returns a list of product's associated with this title.

SongCollection - *represents a database of songs, sorted by SongName.*

Public Attributes:

SongName[*] : String[*]
An array of songs.

Public Operations

getSong(String song)
Operation to find a song in the database.
addSong(Song song)
Operation to add a song to the database.
removeSong(Song song)
Operation to remove a song from the database.
returnSongResults() : String
Operation to return the results of a song search.

Song - *a generalization of records stored in the SongCollection database.*

Public Attributes:

Catalog#[*] : Integer[*]
An array of the catalog numbers of the products containing this song.
SongName : String
The song's name.
Artist[*] : String[*]
An array of the artists associated with this song.

Public Operations

giveProducts() : Product[*]
Returns a list of product's associated with this song.

Description of the Interface Sequence Diagram

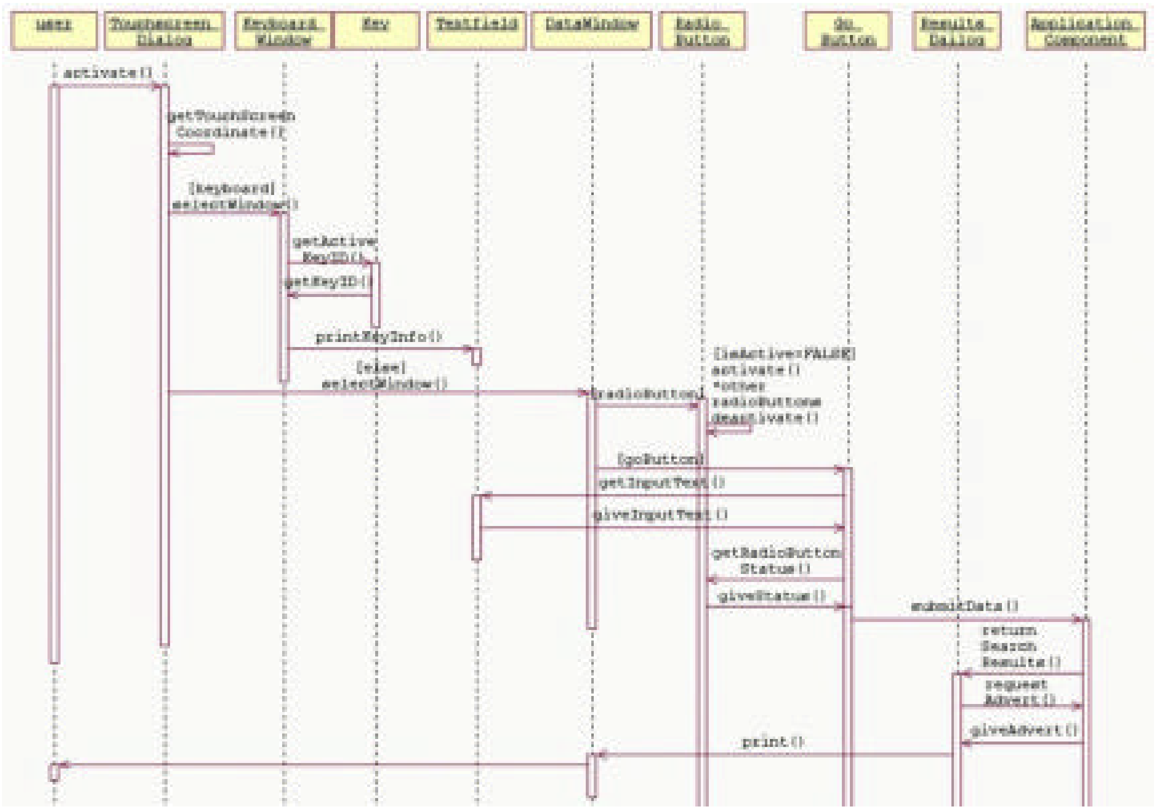
Initially, the user touches the screen to activate the TouchScreenDialog. The TouchScreenDialog then gets the coordinates of where the user had contact with the screen. This determines which window becomes active.

If the users touch the lower half of screen where the keyboard is located, then KeyboardWindow is selected. Then, the ID of the key that the users touch is sent to the Textfield. This allows the character selected by the users to be displayed in the Textfield area of the screen.

The users can also select areas in the upper half of the screen: a RadioButton, or the GoButton. If the coordinates of the user contact correspond to a RadioButton, then that RadioButton becomes active, and all other RadioButtons become deactivated. If the GoButton is selected, then the users wish to submit their query.

Submitting a query consists of getting the InputText from the TextField, getting the status of the RadioButtons, and submitting this Data to the Application Classes. The results of the search are returned to the ResultsDialog, which in turn requests a related Advert to display along with the results. Once the Advert has been sent to the ResultsDialog from the Application Classes, it is displayed to the users along with the search results.

The Interface Sequence Diagram



Glossary of Classes within Sequence Diagram

TouchScreenDialog - *represents a generalization of the physical touchscreen with abilities to analyze user input.*

Public Attributes:

isActive : boolean
Stores whether or not the touchscreen is active.
screenHeight : Integer
The vertical resolution of the screen.
screenWidth : Integer
The horizontal resolution of the screen.
touchXcoordinate : Integer
The x-coordinate of a user touch.
touchYcoordinate : Integer
The y-coordinate of a user touch.

Public Operations

getTouchcoordinate()
Operation to determine the screen coordinates of a touch, when the screen is touched.
selectWindow(Integer x, Integer y)
Operation to determine which window is being accessed.
activate()
Operation to activate the touchscreen.
deactivate()
Operation to deactivate the touchscreen.

KeyboardWindow - *represents the software application emulating an on-screen keyboard for the touch screen.*

Public Attributes:

key[*] : Key[]
An array of all the keys of the keyboard.
height : Integer
Vertical height of the KeyboardWindow, in pixels.
width : Integer
Horizontal width of the KeyboardWindow, in pixels.

Public Operations

getActiveKeyID(Integer x, Integer y) : Integer
Returns the ID of the key at coordinates (x,y).
printKeyInfo(Integer id)
Prints the key's info.

Key - *represents a software emulated key utilized by the KeyboardWindow.*

Public Attributes:

keyID : Integer
The key's ID.
graphic : Graphic
The key's graphic.
height : Integer
The vertical height of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.

Public Operations

getKeyID() : Integer
Returns the key's ID.

DataWindow - *represents the browser used to perform searches, and display results.*

height : Integer
The vertical height of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.
displayObject[*] : DisplayObject[]
An array of objects being displayed on the screen.

Public Operations

followLink(String link)
Operation to follow a hyperlink.
print()
Operation to print information to the display area.

RadioButton - *a specialization of the Button superclass, used to present choices for different searches*.

Public Attributes:

isActive : Boolean
Holds whether or not this radiobutton has been selected.

Public Operations

activate()
Activates the button.
deactivate()
Deactivates the button.
giveStatus() : Boolean
Returns the status of the button.

GoButton - *a specialization of the Button superclass, used to initiate searches.*

Public Operations

submitData(String data)
Submits data for a search.
getInputText()
Gets the input text from the input-field.
getRadioButtonStatus()
Gets the status of the radio buttons.

Description of the Product Inventory Sequence Diagram

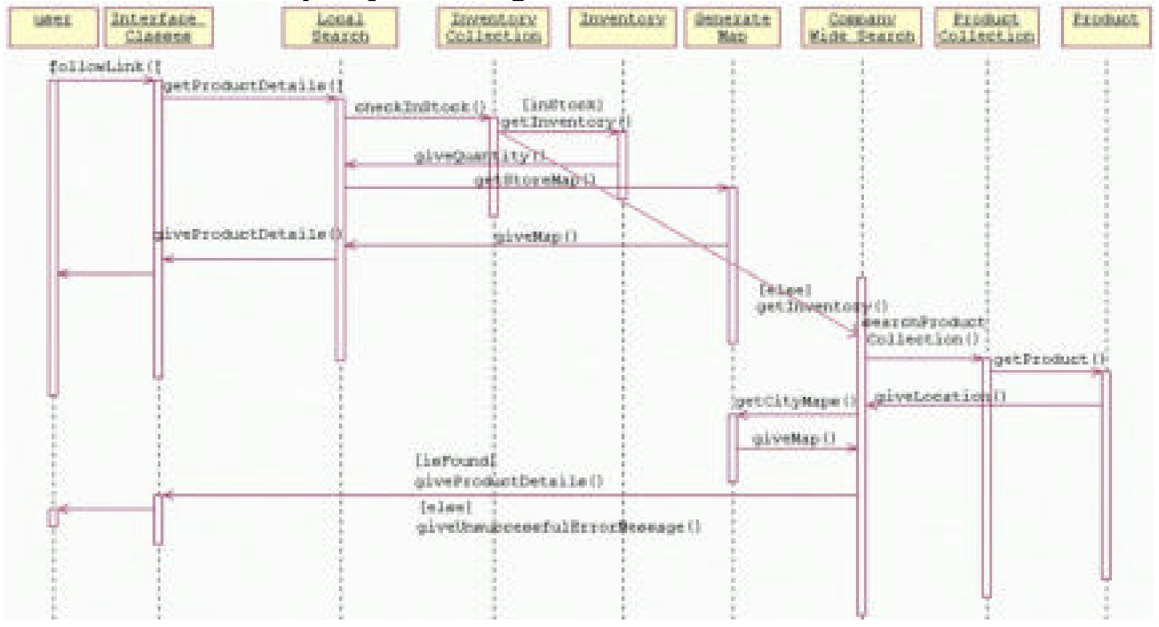
Users are displayed a list of products that match their search query. These products are displayed as links to the users. The users touch the link of the product to see if this product is in stock, and where it can be found.

The Interface Classes request the Product Details from the LocalSearch class. The LocalSearch class searches the InventoryCollection database to find the Inventory object associated with the product in question.

If the Product is carried in the store, then inStock is true, and the corresponding Inventory object returns the quantity of the product in the store. After this information is returned, the LocalSearch class requests a map corresponding to the Product's location in the store (based on the Product's genre). The GenerateMap class then returns the map, and LocalSearch returns the Product Deatils (quantity and map) to the Interface Classes to display to the users.

If the Product is not carried in this branch of HMV, then inStock is false. Then, CompanyWideSearch searches for the stores that carry the specified product. CompanyWideSearch does this by searching through the ProductCollection for the Product. If the product is not found, an unsuccessful error message is displayed to the users. If the product is found, then the location attribute is returned to CompanyWideSearch. CompanyWideSearch sends this information to the GenerateMap class to create a map with highlights the relevant stores in a city map. Then, CompanyWideSearch sends the ProductDeatils to the Interface Component. This operation returns the quantity of the Product (0 in the current branch) and the map showing other locations.

The Product Inventory Sequence Diagram



Glossary of Classes within Sequence Diagram

GenerateMap - *represents the software application used by the kiosk system to create maps.*

Public Attributes:
graphic : Map
The map to be returned.

Public Operations
giveMap() : Map
Returns a map.

Link - *represents a hyperlink.*

Public Attributes:
linkID : Integer
The link's ID.
title : String
The link's title.
destinationAddress : String
The destination of the link.

Public Operations
giveLinkDetails()
Returns the details of the address being linked.

LocalSearch - *a specialisation of the InventorySearch software, used to search the local store's inventory.*

Public Attributes:
isInStock : Boolean
Stores whether or not the product is in stock.

Public Operations
getStoreMap(String store) : Map
Returns a map of the store.

CityWideSearch - *a specialisation of the InventorySearch software, used to search the local area inventory.*

Public Attributes:
isFound : Boolean
Stores whether or not a product is in stock in any store in the local area.

Public Operations
searchProductCollection(String product)
Searches the master database for a product.
getCityMap(String city) : Map
Returns a map of the city.

InventoryCollection - *represents a database of store inventories.*

Public Attributes:
C**atalog#[*]** : Integer[*]
An array of catalog#'s.

Public Operations
getInventory(Integer catID) : Integer
Returns the quantity of a product in stock.
addInventory(Integer catID)
Increases the quantity of a product in stock.
removeInventory(Integer catID)

Decreases the quantity of a product in stock.
Inventory - *a generalisation of records stored in the InventoryCollection database.*

Public Attributes:
Catalog# : Integer
The catalog number of this inventory.
Quantity : Integer
The quantity of this inventory in stock.

Public Operations
giveQuantity() : Integer
Returns the number of a particular product that are in stock.

ProductCollection - represents a database of products, sorted by catalog#.

Public Attributes:
Catalog#[*] : Integer[*]
An array of catalog#'s.

Public Operations
getProduct(String product)
Returns product if it exists in the product database.
addProduct(Product product)
Adds a new product to the database.
removeProduct(Integer catID)
Removes a product from the database.

Product - a generalization of records stored in the ProductCollection database.
Public Attributes:
Catalog# : Integer
The product's catalog number.
Title : String
The product's title.
Genre : String
The product's genre.
RetailPrice : Float
The product's retail price.
CostPrice : Float
The amount the supplier charges for the product.
ReleaseDate : Time
The release date of the product.
Margin% : Float
CostPrice / RetailPrice
Location[*] : String[*]
A list of locations the product is available at.

Public Operations
giveLocation() : String[*]
Returns a list of locations.
giveGenre() : String
Operation to return the genre of the product.
giveRetailPrice() : Float
Operation to return the retail price of the product.
giveCostPrice() : Float
Operation to return the cost price of the product.
giveMargin() : Float

Operation to return the margin% of the product.

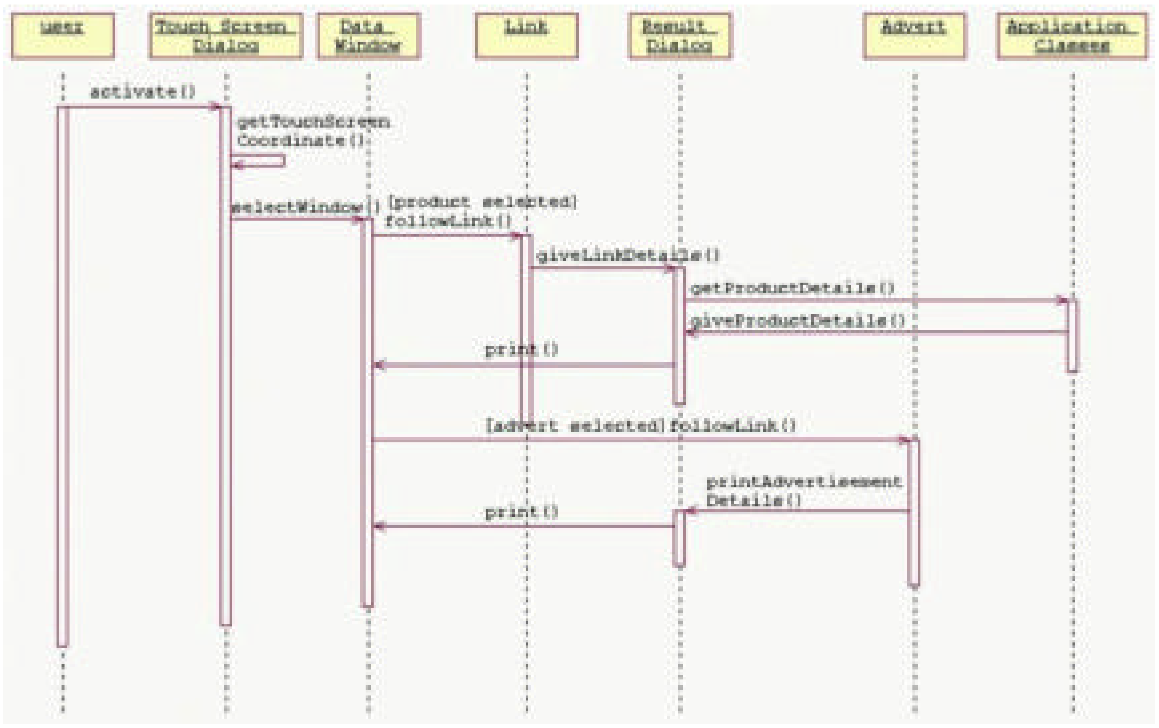
Description of the Product Selection Sequence Diagram

The users are returned a list of products that match their search query. This list is in the form of links. The users select a link to find out more about the desired product.
The users touch the screen, and this first activates the TouchScreenDialog class, which gets the coordinates of the users' contact. TouchScreenDialog selects the DataWindow, which is where links are displayed.

If the coordinates correspond to a product link, then DataWindow follows the link. The Link object then gives its details to the ResultDialog class. This initiates a request from the ResultDialog to get the product details, held in the database class. Then, ResultDialog prints these results to the DataWindow, which displays them to the users.

If the coordinates correspond to an advertisement, then the Advert Link is followed. This means that the Advert object sends its details to the ResultDialog, which prints the information to the screen for the users.

The Product Selection Sequence Diagram



Glossary of Classes within Sequence Diagram

TouchScreenDialog - *represents a generalization of the physical touchscreen with abilities to analyze user input.*

Public Attributes:

isActive : boolean
Stores whether or not the touchscreen is active.
screenHeight : Integer
The vertical resolution of the screen.
screenWidth : Integer
The horizontal resolution of the screen.
touchXcoordinate : Integer
The x-coordinate of a user touch.
touchYcoordinate : Integer
The y-coordinate of a user touch.

Public Operations

getTouchcoordinate()
Operation to determine the screen coordinates of a touch, when the screen is touched.
selectWindow(Integer x, Integer y)
Operation to determine which window is being accessed.
activate()
Operation to activate the touchscreen.
deactivate()
Operation to deactivate the touchscreen.

DataWindow - *represents the browser used to perform searches, and display results.*

height : Integer
The vertical height of the key, in pixels.
width : Integer
The horizontal width of the key, in pixels.
displayObject[*] : DisplayObject[]
An array of objects being displayed on the screen.

Public Operations

followLink(String link)
Operation to follow a hyperlink.
print()
Operation to print information to the display area.

Link - *represents a hyperlink.*

Public Attributes:

linkID : Integer
The link's ID.
title : String
The link's title.
destinationAddress : String
The destination of the link.

Public Operations

giveLinkDetails()
Returns the details of the address being linked.

Advert - *represents an advertisement banner which can be displayed in the DataWindow.*

Public Attributes:

advertID : Integer
The advertisement's ID.
graphic : Graphic
The advertisement's banner graphic.
advertDetails : String
The advertisement's description.

Public Operations

getAdvertID(String genre)
Gets an advertisement for a specific genre.
printAdvertisementDetails()
Prints the advertisement's details to DataWindow.

Inventory Search - *represents the software application used by the kiosk system to check for the existence of products.*

Public Operations

checkInStock(String product) : boolean
Return true if the product is in the inventory collection.
giveProductDetails() : Product,Quantity,Map
Returns the details of a product, and a map to where it can be found.

Justification That the Sequence Diagrams Meet the Relevant Requirements

The relevant Requirements for the sequence diagrams are:

Functional Requirements:

1. Processing Requirements
   The system must be able to translate and relay queries to the stores existing database, and translate and relay the results of the query back to the user.

2. Input Requirements
   The system must be able to accept queries from users and must be able to receive information from the stores existing database.

3. Output Requirements
   The system must be able to display the results of queries immediately to the user, as well as relevant location information.

Non-Functional Requirements:

2. Efficiency Requirements
   The system must be fast because customers can be very impatient.

Justification that the design meets relevant requirements:

The program design meets the all processing requirements outlined in the requirements analysis document. The Search Sequence diagram demonstrates how a search is successfully translated and relayed to the store's existing database, and how the results are translated and displayed to the user. When the user when the user presses the goButton, the query is translated by the browser into a format recognised by the HMV servers . The server then performs a search on the existing database, and relays the relays the results to the ResultsDialog class which translates the results into HTML and prints them to the DataWindow class .

All input and output requirements are met by the kiosk system. The touchscreen dialog allows users to input queries, and the browser is able to translate search results from the local server into HTML pages for viewing.

The diagrams for the Interface Sequence,and the Product Selection Sequence ,the manner in which input is collected from the user. The Interface Sequence diagram shows how the user's physical contact with the screen allows them to select various windows, and input search string. The Product Selection Sequence demonstrates that links can be selected by the user.

The diagrams for the Product Inventory Sequence (, the Product Selection Sequence,, the Search Sequence ,and the Interface Sequence , demonstrate the output requirements since they show how the results of the queries are successfully displayed to the user. In the diagrams, all information passed back from the server is collected by ResultsDialog, which is responsible for displaying search results and product location to users.

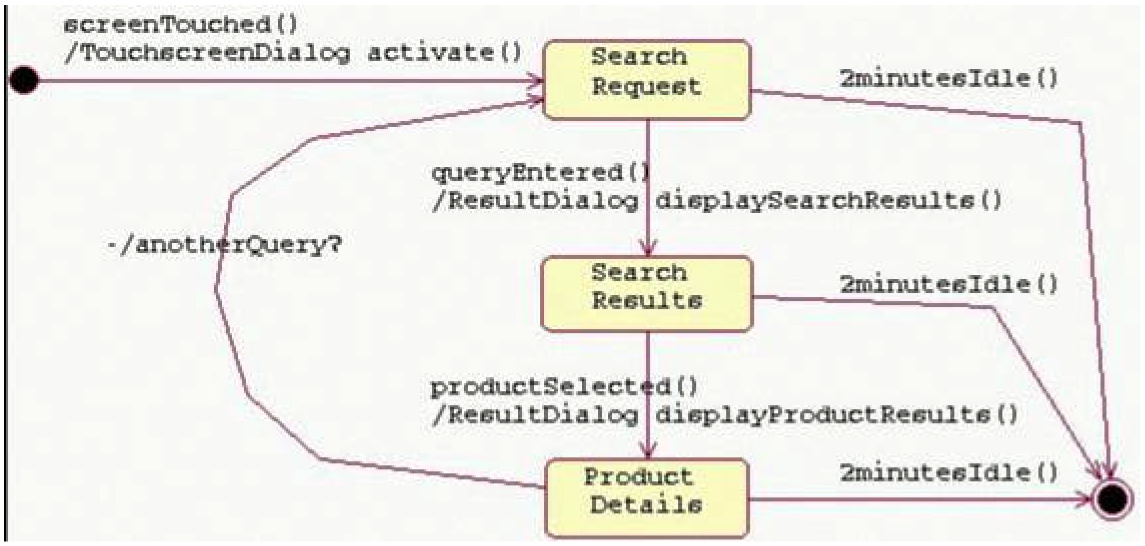
The program design meets the efficiency requirements outlined in the requirements analysis document. Search requests are submitted directly to the store's local server that performs the search on the existing database. Additional information about products is not downloaded until request, thus minimizing search time and maximizing efficiency.

State diagrams

There are five State diagrams: Interface State, Radio Button State, Result Dialog State, Artist Search State and Touch Screen Dialog State.

Description of the Interface Dialog Statechart Diagram

The users touch the screen to begin their search. This contact activates the TouchScreenDialog. The users now have the opportunity to input a Search Request. Once the query is entered, the ResultDialog displays a list of Products that match the users query (the search results). The users then have the ability to select a product in the results list. If they select a product, then the ResultsDialog displays the Product's details. If, at any point, the TouchscreenDialog is not touched for two consecutive minutes, then it is deactivated, and the users' session is ended.



TouchScreenDialog - *represents a generalization of the physical touchscreen with abilities to analyze user input.*

Public Attributes:

isActive : boolean
Stores whether or not the touchscreen is active.
screenHeight : Integer
The vertical resolution of the screen.
screenWidth : Integer
The horizontal resolution of the screen.
touchXcoordinate : Integer
The x-coordinate of a user touch.
touchYcoordinate : Integer
The y-coordinate of a user touch.

Public Operations

getTouchcoordinate()
Operation to determine the screen coordinates of a touch, when the screen is touched.
selectWindow(Integer x, Integer y)
Operation to determine which window is being accessed.
activate()
Operation to activate the touchscreen.
deactivate()
Operation to deactivate the touchscreen.

ResultsDialog - *represents the software application handling input from the store's existing database.*

Public Attributes:

resultText : String
The result of a search.
numberOfResults : Integer
The number of products found.
displayObject[*] : DispalyObject[*]
An array of objects to be displayed on the screen.

Public Operations

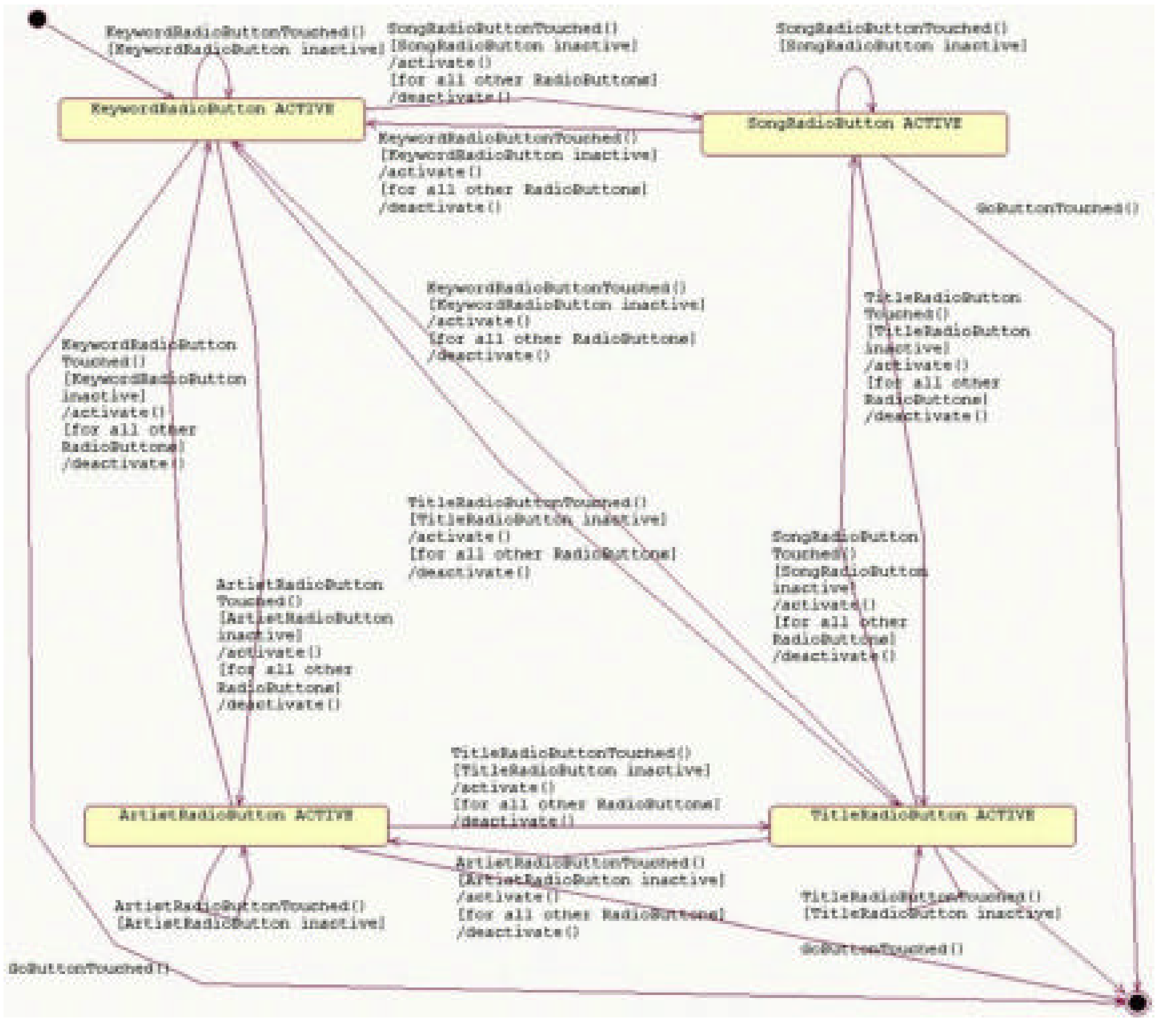
print()
Prints display objects to the screen.
requestAdvert()
Requests an advertisement from the database.
getProductDetails()
Retrieves product details from the database.

Description of the Radio Button State Diagram

Users have a choice for the type of search that they are going to perform: either an artist search, a song search, a title search or a keyword search. The users select a radio button to specify the type of search.

The Keyword RadioButton is the default RadioButton selected, since any search can be performed as a Keyword search. When users select any other RadioButton, the Keyword RadioButton becomes inactive, and the selected RadioButton becomes active. In general, when a RadioButton is selected:
1- if it is inactive, it becomes active, and all other RadioButtons become inactive;
2- if it is active, it remains active, and the other RadioButtons remain inactive.

This diagram illustrates these properties with respect to the four RadioButtons in our interface. When the users select the GoButton to submit their search, then the end state is reached.


KeywordRadioButtonTouched() [KeywordRadioButton inactive]
SongRadioButtonTouched() [SongRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
SongRadioButtonTouched() [SongRadioButton inactive]
KeywordRadioButton ACTIVE
SongRadioButton ACTIVE
KeywordRadioButtonTouched() [KeywordRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
GoButtonTouched()
KeywordRadioButtonTouched() [KeywordRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
TitleRadioButton Touched() [TitleRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
KeywordRadioButton Touched() [KeywordRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
TitleRadioButtonTouched() [TitleRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
SongRadioButton Touched() [SongRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
ArtistRadioButton Touched() [ArtistRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
TitleRadioButtonTouched() [TitleRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
ArtistRadioButton ACTIVE
TitleRadioButton ACTIVE
ArtistRadioButtonTouched() [ArtistRadioButton inactive] /activate() [for all other RadioButtons] /deactivate()
ArtistRadioButtonTouched() [ArtistRadioButton inactive]
TitleRadioButtonTouched() [TitleRadioButton inactive]
GoButtonTouched()
GoButtonTouched()

RadioButton - *a specialization of the Button superclass, used to present choices for different searches*.

Public Attributes:

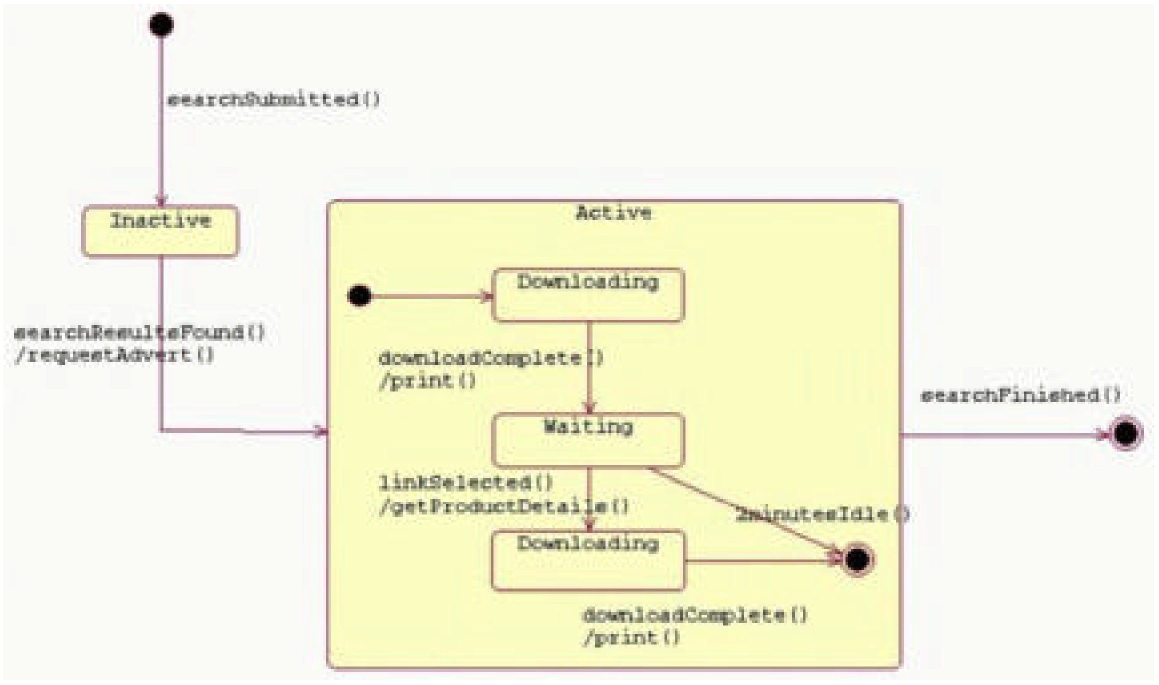
isActive : Boolean
Holds whether or not this radiobutton has been selected.

Public Operations

activate()
Activates the button.
deactivate()
Deactivates the button.
giveStatus() : Boolean
Returns the status of the button.

Description of the Result Dialog State Diagram

The ResultDialog remains inactive while the search is progressing. Once the results are found, ResultDialog becomes active. In the active state, ResultDialog first requests an advertisement that is related to the products that the users are requesting. When the ResultDialog has downloaded the advertisement, it prints the results of the search along with the advertisement. At this point, the ResultDialog is waiting for the users to select one of the links representing a product. If the user does not select a link within two minutes, the ResultDialog exits the active state and the search is terminated. If the user does select a link, the ResultDialog requests the details of the product selected. The ResultDialog is now in a downloading state. Once the details are downloaded, ResultDialog displays these details to the users, and the search is finished.



ResultsDialog - *represents the software application handling input from the store's existing database.*

Public Attributes:

resultText : String
The result of a search.
numberOfResults : Integer
The number of products found.
displayObject[*] : DispalyObject[*]
An array of objects to be displayed on the screen.

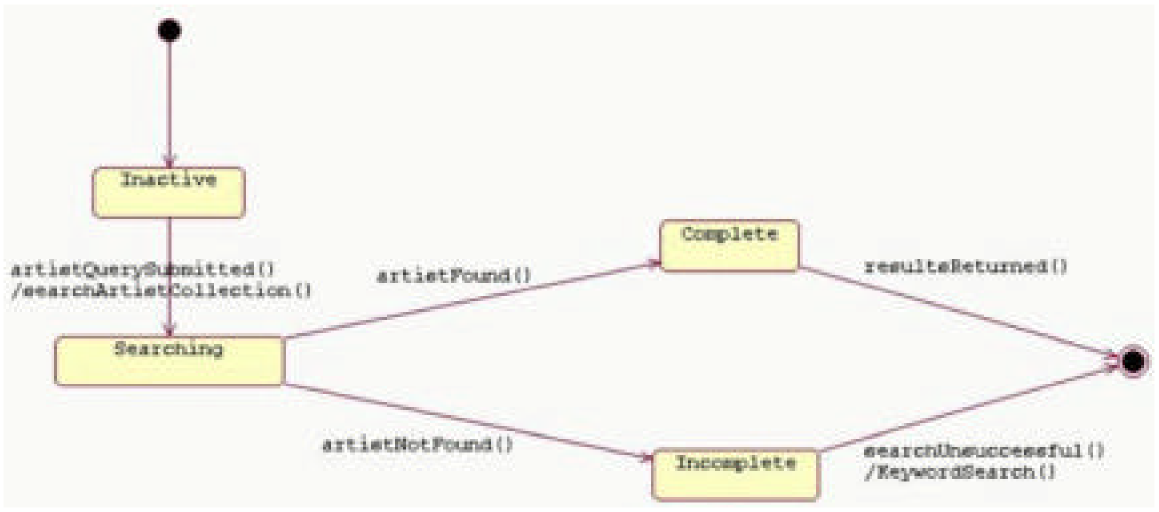
Public Operations

print()
Prints display objects to the screen.
requestAdvert()
Requests an advertisement from the database.
getProductDetails()
Retrieves product details from the database.

Description of Artist Search State Diagram

The users initially type in their search string. During this time, the ArtistSearch class is inactive. The users will then select the ArtistRadioButton, and touch the GoButton to submit their query. The ArtistSearch class is now searching through the ArtistCollection index to find the closest matches to the users' query. If the artist is found, then the search is complete and the results are returned. If the artist is not found, then a keyword search is implemented to attempt to match the users' query with either a Title or a Song. This brings the artistSearch to an end.

Artist Search State Diagram



Search - *represents the software application used by the kiosk system to perform searches. Search's implementation depends on being able to call on the* ArtistSearch, SongSearch, TitleSearch, and KeywordSearch applications.

Public Attributes:

searchText : String
The text which is being searched for.

Public Operations

ChooseSearchFunction(int choice)
Calls the appropriate search function, based on "choice".
returnSearchResults() : String
Returns the results of a search.

ArtistSearch - *represents the software application used to perform searches on the artist database.*

Public Attributes:

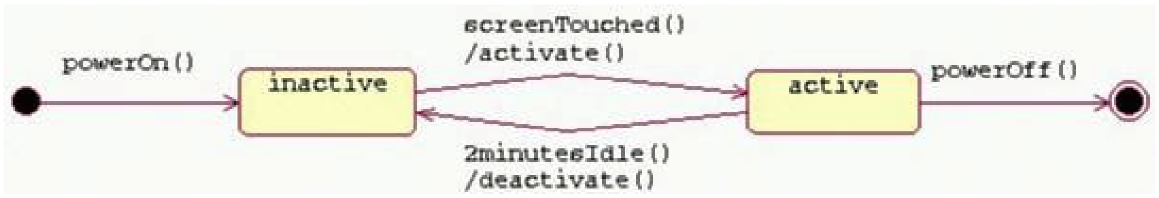
artistSearchResults : String
Holds the results of an artist search.

Public Operations

searchArtistCollection(String artist)
Operation to search for an artist in the database.

Description of Touch Screen Dialog State Diagram

The TouchscreenDialog class deals with the users' input and output. Initially, when the kiosk is turned on, the TouchscreenDialog is inactive. It becomes activated when a user touches the screen. Then, the TouchscreenDialog is active and dialogs within the TouchscreenDialog are interacting with the user. If the screen is left untouched for two minutes, then the TouchscreenDialog reverts back to an inactive state. This state conversion conserves energy, and brings the screen back to a welcome screen for the next user. When the machine is turnedoff, Touchscreen has reached the end state.



TouchScreenDialog - *represents a generalization of the physical touchscreen with abilities to analyze user input.*

Public Attributes:

isActive : boolean
Stores whether or not the touchscreen is active.
screenHeight : Integer
The vertical resolution of the screen.
screenWidth : Integer
The horizontal resolution of the screen.
touchXcoordinate : Integer
The x-coordinate of a user touch.
touchYcoordinate : Integer
The y-coordinate of a user touch.

Public Operations

getTouchcoordinate()
Operation to determine the screen coordinates of a touch, when the screen is touched.
selectWindow(Integer x, Integer y)
Operation to determine which window is being accessed.
activate()
Operation to activate the touchscreen.
deactivate()
Operation to deactivate the touchscreen.

Justification That the State Diagrams Meet the Relevant Requirements

The Relevant Requirements are:

2. Input Requirements
   The system must be able to accept queries from users and must be able to receive information from the stores existing database.

3. Output Requirements
   The system must be able to display the results of queries immediately to the user, as well as relevant location information.

The statechart diagrams demonstrate the input requirements are met in the following ways: they show that input can be successfully received from the users, which changes the states of certain classes, and they show that the Interface component can communicate successfully with the existing database by accepting the search results from the Application Component, which also results in changes of state in certain classes.

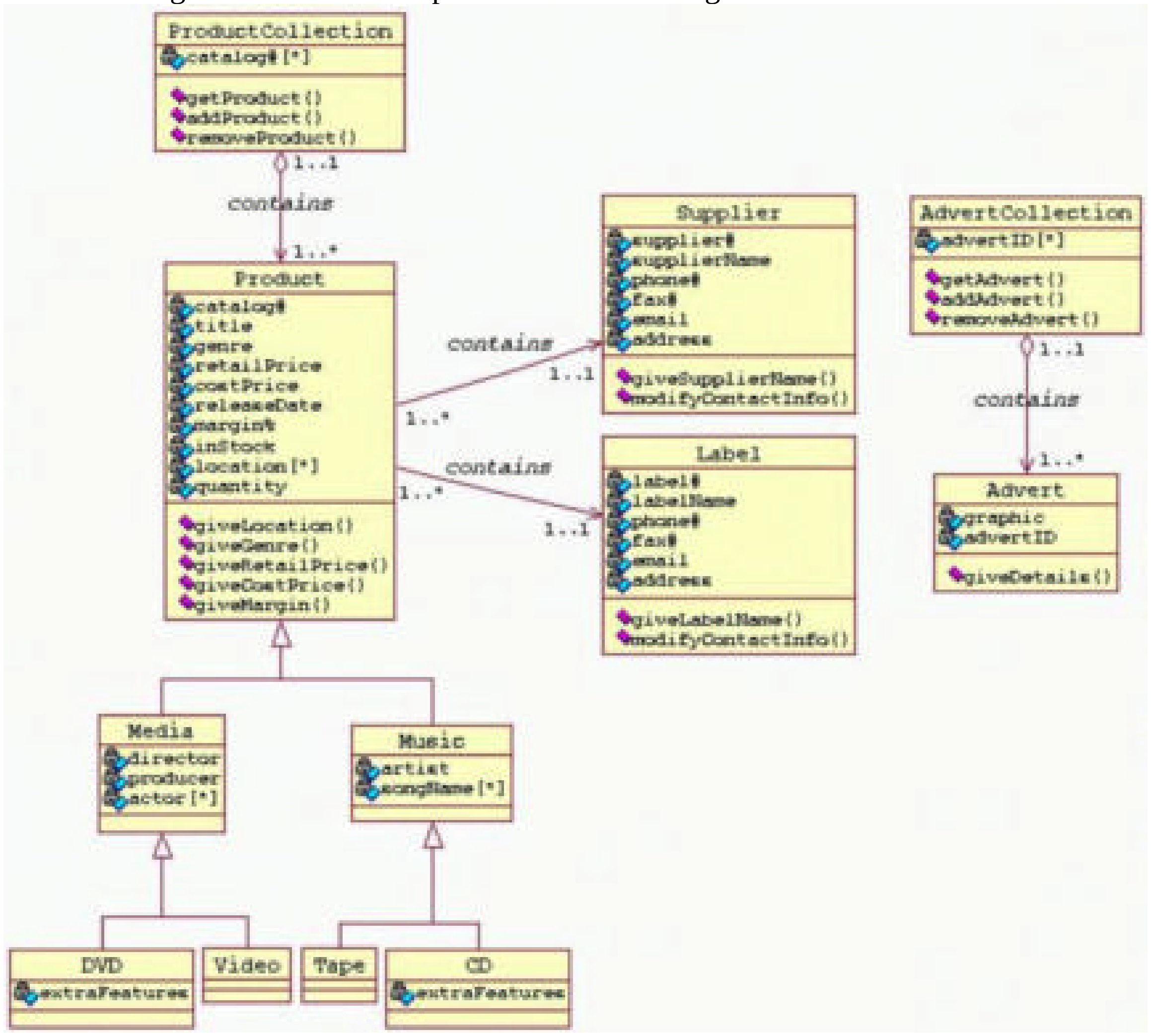
The behaviour of the TouchscreenDialog class, the RadioButton class, the Search class and the ResultDialog class responds to user input and results input (from the database classes). These responses show that our system accepts the users' queries successfully. Specifically, The TouchscreenDialog Statechart Diagram shows that users physical contact with the screen initiates the search. The RadioButton Statechart diagram shows that the users' choice of search is recognized through their choice of the RadioButton selection. The ArtistSearch Statechart Diagram is representative of each type of search. This statechart diagram demonstrates that queries are accepted from users, and that the results of the queries are accepted from the database component. The ResultDialog Statechart diagram shows the successful communication between the interface component and the application/database component.

The statechart diagrams demonstrate the output requirements since they show how the results of the queries are successfully displayed to the users. The ResultDialog class is responsible for displaying search results and product location to the users. The ResultDialog Statechart Diagram demonstrates this by the print() statement that leads to the end state within the Active state.

Part C: The Database Component

The HMV kiosk system consists essentially of Interface and Application classes. The most cost-effective and efficient implementation of our system uses the database that is already in existence in each HMV store. Since DataPear is not designing a database, Professor Mylopoulos suggested that we analyze the existing HMV system and try to reconstruct the process they may have gone through in their design phase. The following section describes a potential database and the related analysis that leads to its restructuring into a database that resembles the existing one.

The class diagram below could represent an Initial design for the database.



Identifiers of Database Classes

The following tuples represent the database classes. The unique identifier of each class is underlined.

CD(<u>catalog#</u>, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Tape(<u>catalog#</u>, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

DVD(<u>catalog#</u>, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Video(<u>catalog#</u>, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

Advert(<u>advertID</u>, graphic)

Supplier ***(<u>supplier#</u>, supplierName, phone#, fax#, email, address)***

Label(<u>label#</u>, labelName, phone#, fax#, email, address)

Database Constraints

Referential Integrity: This ensures that an object identifier in an object is actually referring to an object that exists. The referential Integrity of the database is always maintained because in the ObjectCollection class, there is an addObject() and removeObject(). These operations ensure that whenever an object is added or deleted from the database, all objects that correspond to the deleted objects are updated.

Dependency Constraints: This ensures that attribute dependencies, where one attribute may be calculated from other attributes, are maintained consistently. The only attribute that is dependent in the database is the margin% attribute in all Products. The accuracy of this number is verified whenever the giveMargin() operation is performed, thus ensuring that this field remains correct.

Domain Constraints: This ensures that attributes only hold permissible values. This constraint is kept because the addObject() operation in the ObjectCollection class verifies each parameter value for correctness before adding the object to the database.

Workload Data

The operations that are performed in the database are summarized in the tables below.

Workload Data

Concept	Type	Volume
Title	E	7 800 000
Artist	E	4 000 000
Song	E	78 000 000
Inventory	E	5 000
CD	E	7 500 000
Tape	E	5 000 000
DVD	E	100 000
Video	E	300 000
Label	E	10 000
Supplier	E	50

Table of Operations

#	Operation	Type	Frequency per day
1	Search for song title	I	250.00
2	Search for artist name	I	250.00
3	Search for product title	I	250.00
4	Search for keyword	I	250.00
5	Insert DVD	I	0.05
6	Insert video	I	0.05
7	Insert tape	I	0.15
8	Insert CD	I	0.20
9	Get inventory	B	1.00

Redundancy Analysis

The addition of some collection classes, namely: Inventory; Artist; Title and Song, could create redundancy, which is analyzed with the representative operations two, eight and nine. These operations were chosen because operations one through four have the same basic functionality and operations five through eight have the same basic functionality.

· Operation 2 – Search artist: This operation could be made redundant because we presently look through all the product checking the artist field when we could add a separate look-up table and simply do a search on that table.

Operation 2 with redundancy				**Operation 2 without redundancy**			
Concept	**Type**	**Access**	**Type**	**Concept**	**Type**	**Access**	**Type**
ArtistCollection	E	Log(4 million)	R	ProductCollection	E	12.9 million	R
Artist	E	3 *	R				
ProductCollection	E	3 x Log(12.9 million)	R				
Product	E	3	R				
Total	**138 Reads**			**Total**	**12.9 million Reads**		

*Each artist has, on average, three CDs

· Operation 8 – Insert CD: Having the different look-up tables would result in increased redundancy while inserting new information into the system.

Operation 8 with redundancy				**Operation 8 without redundancy**			
Concept	**Type**	**Access**	**Type**	**Concept**	**Type**	**Access**	**Type**
CD	E	1	W	CD	E	1	W
ProductCollection	E	Log(12.9 million)	R	ProductCollection	E	Log(12.9 million)	R
ProductCollection	E	1	W	ProductCollection	E	1	W
Title	E	1	W				
TitleCollection	E	Log(7.8 million)	R				
TitleCollection	E	1	W				
Artist	E	1	W				
ArtistCollection	E	Log(4 million)	R				
ArtistCollection	E	1	W				
Song	E	6	W				
SongCollection	E	Log(78 million)	R				
SongCollection	E	1	W				
Inventory	E	1	W				
InventoryCollection	E	Log(5000)	R				
Total	**103 Reads, 15 Writes (133)**			**Total**	**24 Reads, 2 Writes (28)**		

· Operation 9 – Get Inventory: This operation would be made redundant if we add a separate look-up table; we could avoid the separate look-up table and simply look through all the product checking an inStock field.

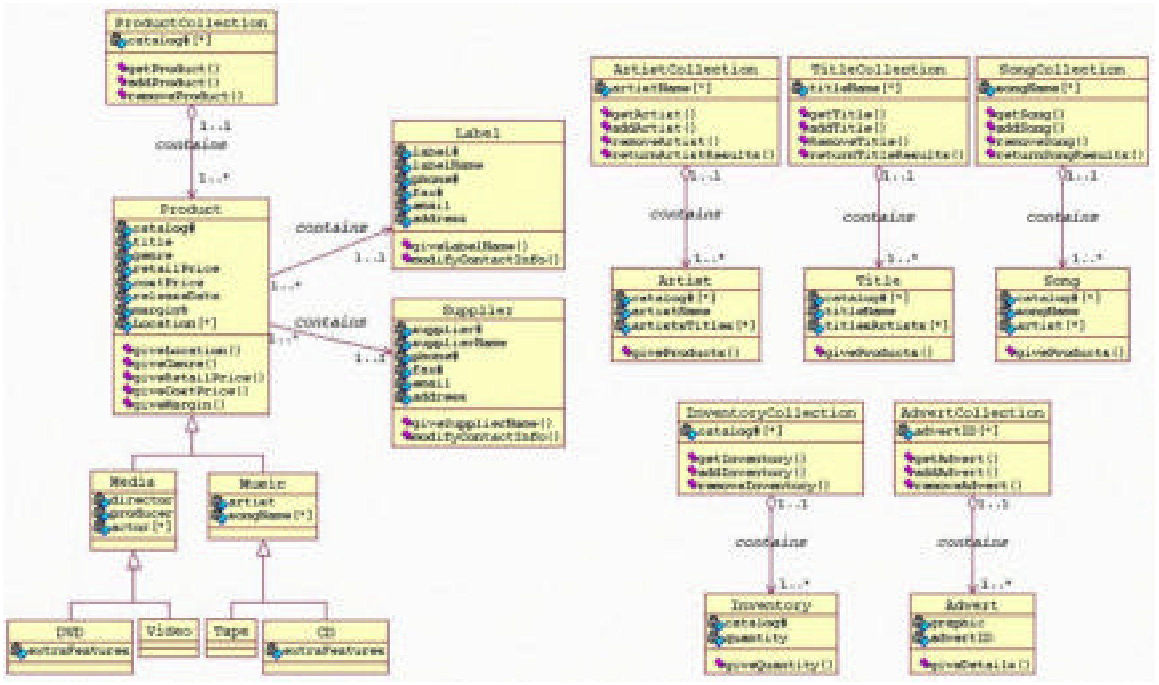
Operation 9 with redundancy				**Operation 9 without redundancy**			
Concept	**Type**	**Access**	**Type**	**Concept**	**Type**	**Access**	**Type**
Inventory	E	5000	R	Product	E	12.9 million	R
Total	**5000 Reads**			**Total**	**12.9 million Reads**		

Workload Analysis

	With Redundancy			Without Redundancy		
	accesses	frequency per day	accesses x frequency	accesses	Frequency per day	accesses x frequency
Operation 2	138	250	34500	12.9 million	250	3 225 000 000
Operation 8	133	0.20	26.6	28	0.20	5.6
Operation 2 + Operation 8	**34 526.6**			**3 225 000 005.6**		
Space Analysis	The added look-up tables increase the space required, but speed is a much larger concern.			Obviously this takes up much less space, but the speed trade-off is tremendous.		

	With Redundancy			Without Redundancy		
	accesses	frequency per day	accesses x frequency	accesses	Frequency per day	accesses x frequency
Operation 9	5000	1	5000	12.9 million	1	12.9 million

Using the analysis above we arrive at a database that can be represented with the class diagram below.



Generation of the Relational Schema

Removing Generalizations

The generalization that exists in our database with redundancy is the following:

- DVD and Video inherit from the Media class, which inherits from the Product class.
- Tape and CD inherit from the Music class, which inherits from the Product class.

We remove these generalizations as follows:

- DVD will include Media's and Product's operations and attributes, in addition to its own.
- Video will include Media's and Product's operations and attributes, in addition to its own.
- Tape will include Music's and Product's operations and attributes, in addition to its own.
- DVD will include Music's and Product's operations and attributes, in addition to its own.

Partitioning and Merging of Classes

Accesses are reduced by separating attributes of the same concept that are accessed by different operations and by merging attributes of different concepts that are accessed by the same operations.

A careful analysis of the existing classes of our database shows that classes exist with attributes that group together concepts accessed by the same operations. Specifically, the ObjectCollection classes have attributes that are partitioned from the Product classes based on the type of search performed.

However, the partitioning process was not complete when creating the ObjectCollection classes because the attributes partitioned still remain in the Product classes. The Product classes still contain the partitioned attribute in order to maintain the integrity of the entire object. The Product can be referenced by several searches and so all the relevant attributes must remain with the Product.

The following describes the relevant operations considered and how, in relation to these operations, the database with redundancy is already partitioned.

There are four main search-operations performed on the products available at HMV by the customers: an artist search, a title search, a song search, and a keyword search. HMV management regularly performs general product searches and inventory searches. These searches are each dealt with separately below.

The ArtistSearch takes in an artistName to look up, and returns the results of the search: a list of catalog#s and titles associated with the artist' s products. To deal efficiently with this search, an ArtistCollection class holds an array of Artist objects: which each have an artistName, a catalog#[*] and Artiststitle[*]. The ArtistCollection class partitions the artist, catalog# and title attributes of the CD and Tape objects. This allows the artist search operation to be performed more efficiently. As previously mentioned, the artist, catalog# and title attribute remains in the object so that it can be easily referenced during other searches.

The title search and the song search are dealt with in a similar manner.

The keyword search effectively performs an artist search, followed by a title search, followed by a song search to attempt to match the user's query to any of these fields in an HMV product. The addition of the aforementioned Collection classes allows this search to be performed more efficiently.

A general Product search is conducted when a store employee or manager requires the full details of a particular product. A ProductCollection class exists with the partitioned catalog# attribute of the DVD, Video, CD and Tape objects. This allows the general product search operation to be performed more efficiently. The catalog# attribute remains in the object so that products can be referenced across searches.

A product inventory search is conducted when the inventory details of a particular product are requested. An InventoryCollection class exists with the partitioned catalog# and quantity attributes of the DVD, Video, CD and Tape objects. This allows the inventory search operation to be performed more efficiently. The addition of this class has partitioned out the quantity and inStock attributes that existed in the database without redundancy.

Selection of Primary Identifiers

Similar classes are grouped so that their primary identifiers can be evaluated together. The final relational schema is found at the end of this section.

The following tuples represent different Products at HMV:

CD(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Tape(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

DVD(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Video(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

These tuples were created with a unique catalog#, and this attribute serves as a unique identifier.

The next classes evaluated are the Advert class, the Supplier class and the Label class.

Advert(advertID, graphic)
*Supplier****(supplier#, supplierName, phone#, fax#, email, address)***
*Label****(label#, labelName, phone#, fax#, email, address)***
Inventory(catalog#,quantity)
Artist(artistName, catalog#[*], artistTitles[*])
Title(titleName, catalog#[*], titleArtists[*])
Song(songName, catalog#[*], artist[*])

Advert's attributes include the advertID, and the graphic associated with the advert. The advertID is a unique code assigned to each instantiation of the Advert class. Thus, the advertID is Advert's unique identifier. The Supplier and Label also have a unique code: supplier# and label#, along with other attributes. Thus, supplier# and label# are the unique identifiers for there objects. The Inventory object is referenced by the unique catalog#, so this is its key. The Artist, Title and Song object are all referenced with a string, their Name, and this is their unique identifier.

The following classes are Collection classes. They represent several indices of the HMV products. There is only one instantiation of each of these classes, therefore no unique identifier is needed.

ArtistCollection(artistName[*])
TitleCollection(titleName[*])
SongCollection(songName[*])
ProductCollection(catalog#[*])
InventoryCollection(catalog#)
AdvertCollection(advertID[*])

Relational Schema

CD(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Tape(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

DVD(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], extraFeatures, supplier#, label#)

Video(catalog#, title, genre, retailPrice, costPrice, releaseDate, margin%, location[*], supplier#, label#)

Advert(advertID, graphic)

Supplier ***(supplier#, supplierName, phone#, fax#, email, address)***

Label(label#, labelName, phone#, fax#, email, address)

Inventory(catalog#,quantity)

Artist(artistName, catalog#[*], artistTitles[*])

Title(titleName, catalog#[*], titleArtists[*])

Song(songName, catalog#[*], artist[*])

ArtistCollection(artistName[*])*

TitleCollection(titleName[*])*

SongCollection(songName[*])*

ProductCollection(catalog#[*])*

InventoryCollection(catalog#)*

AdvertCollection(advertID[*])*

*There is only one instantiation of each of these classes; therefore no unique identifier is needed.

Normalization:
The classes Advert, Supplier, and Label are very simple classes, identified by a unique code as their key. Thus, they are already in 3NF form. The Collection classes only have one instantiation of each class, and hence they also do not need to be normalized. Therefore, the only classes remaining that require normalization are the DVD, Video, CD and Tape classes. The following normalization uses three Tape objects to represent the normalization of all Product classes.

0 NF

Catalog#	Title	Artist	Genre	Supplier#	Label#	Retail Price	Cost Price	Margin%	Release Date	Song[
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	64%	06/01/96	-Dry tl Rain- -Sumn day- -Sprin Water -Bitter Tree- -No O Know -Were Makin This U
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	64%	11/12/00	-So Fr So Cle -B.O.I -Mrs. Jackso -Toile Tisha- -Yam Powda -Ghett Blues-
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	68%	01/01/81	- Norma your MOJC -I Lea Nothin from U MaryI – -666N -Used Case- -Seedi Monte Carlo- - Kill SSR-

1NF

A table is in first normal form IFF all row/column intersections contain atomic values.

Remove Margin, because it can be derived from Retail Price and Cost Price then expand the repeating group Song.

Catalog#	Title	Artist	Genre	Supplier#	Label#	Retail Price	Cost Price	Release Date	Song
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	Dry the Rain
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	Summer day
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	Spring Water
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	Bitter Tree
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	No One Knows
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96	Were Making This Up
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	So Fresh, So Clean
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	B.O.B.
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	Mrs. Jackson
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	Toilet Tisha
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	Yam Powdah
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00	Ghetto Blues
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	Normalise your MOJO!
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	I Learned Nothing from U of MaryLand
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	666NF
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	Used Case
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	Seeding Monte Carlo
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81	Kill SSR

2NF

A relation is in second normal form IFF it is in first normal form and every non-key attribute is fully dependent on the primary key.

Separate Song into a separate table because it is not fully dependent on the primary key.

Catalog#	Title	Artist	Genre	Supplier#	Label#	Retail Price	Cost Price	Release Date
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$22.99	$14.34	06/01/96
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$22.99	$14.34	11/12/00
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$18.99	$12.82	01/01/81

Song	Catalog#
Dry the Rain	War987
Summer day	War987
Spring Water	War987
Bitter Tree	War987
No One Knows	War987
Were Making This Up	War987
So Fresh, So Clean	BM7654
B.O.B.	BM7654
Mrs. Jackson	BM7654
Toilet Tisha	BM7654
Yam Powdah	BM7654
Ghetto Blues	BM7654
Normalise your MOJO!	Sorny67
I Learned Nothing from U of MaryLand	Sorny67
666NF	Sorny67
Used Case	Sorny67
Seeding Monte Carlo	Sorny67
Kill SSR	Sorny67

3NF

A relation is in third normal form IFF it is in second normal form and every attribute is dependent on the primary key and not on another non-key attribute.

Separate Retail Price out of the table because it is dependent on Cost Price (but is not just a function of cost price)

Normalise your MOJO!	Sorny67
I Learned Nothing from U of MaryLand	Sorny67
666NF	Sorny67
Used Case	Sorny67
Seeding Monte Carlo	Sorny67
Kill SSR	Sorny67

Song	**Catalog#**
Dry the Rain	War987
Summer day	War987
Spring Water	War987
Bitter Tree	War987
No One Knows	War987
Were Making This Up	War987
So Fresh, So Clean	BM7654
B.O.B.	BM7654
Mrs. Jackson	BM7654
Toilet Tisha	BM7654
Yam Powdah	BM7654
Ghetto Blues	BM7654

Catalog#	**Title**	**Artist**	**Genre**	**Supplier#**	**Label#**	**Cost**	**Release**

						Price	Date
War987	Three EP's	Beta Band	Alternative	45RZ	FM83	$14.34	06/01/96
BM7654	Stankonia	Outkast	Rap	32VB	AM64	$14.34	11/12/00
Sorny67	Shaking the Tree	Pear Juice	Thrash Metal	7SH	DL2	$12.82	01/01/81

Cost Price	Retail Price
$14.34	$22.99
$12.82	$18.99

Justification that the Design Meets the Relevant Requirements

The most relevant requirement for any database is efficiency. Our database design emphasises speed over space with the inclusion of the separate search tables/classes, as would be required by the HMV database. The design of our database is simple because the minimum amount of redundancy was added to optimize the most frequent operations. By having a simple database design we satisfy Economy requirements, because it will be easier to implement, and we also satisfy the Reliability requirements because a simpler design will contain less errors.

The Input and Output requirements of the system have been met because our design stresses simplicity in the interaction between the database and the application(s) using the database. This emphasis on cohesion also meets the Interoperability and Lifecycle requirements of the database.

Part D: Interface Design

The Target Users

The target users of our system encompass a large variety of people. The main users of our system will be customers and employees.

Our customer users are essentially any person that inside an HMV store and interested in purchasing a CD. The age of customers range from approximately 10 years to 85 years since persons within that age group are able to purchase CDs. Customers are middle class and upper class citizens since they are able to afford CDs. Customers in this class bracket often have some exposure to computers. Thus, the majority of this group is familiar with the Windows operating system and browsers. Since our customer users will be using a kiosk before purchasing a CD, it is likely that most of them are in a hurry.

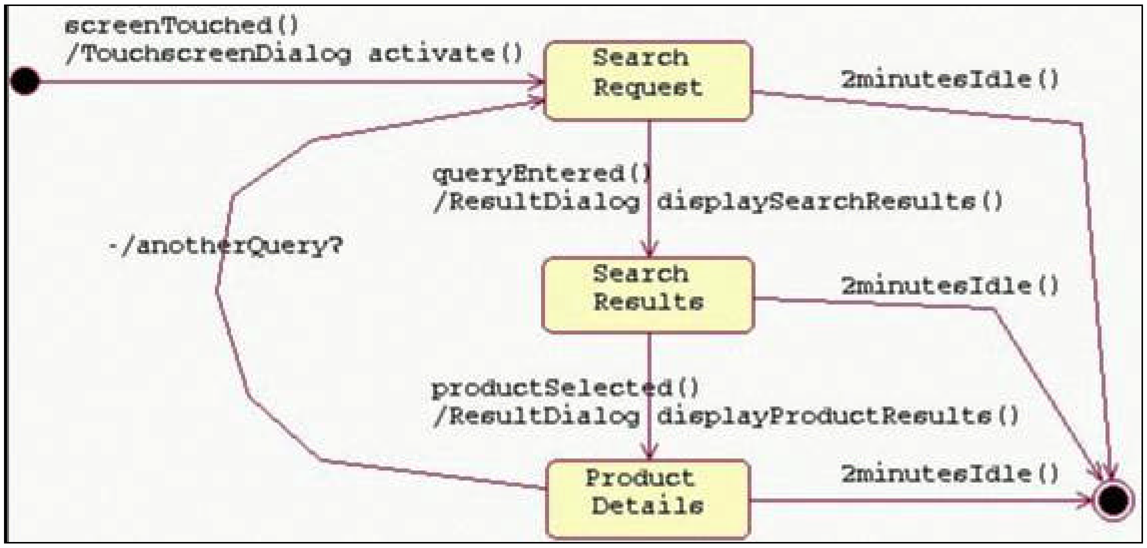
The second group of users for our system is the HMV store employees. Employees range in age from approximately 16 to 60. Employees deal with a computer system daily. Thus, they are quite familiar with searches and queries. Furthermore, persons from 16 to 60 are often users of computers. Thus, this user group is also familiar with the Windows operating system and browsers. When using our system, the employees are usually in the process of helping a customer, so it is likely that they are in a hurry.

Our users will be approximately 50% male and 50% female. The users of our system are literate, and able to understand English.

Description of the Interface Dialog State Diagram

The users touch the screen to begin their search. This contact activates the TouchScreenDialog. The users now have the opportunity to input a Search Request. Once the query is entered, the ResultDialog displays a list of Products that match the users query (the search results). The users then have the ability to select a product in the results list. If they select a product, then the ResultsDialog displays the Product's details. If, at any point, the TouchscreenDialog is not touched for two consecutive minutes, then it is deactivated, and the users' session is ended.

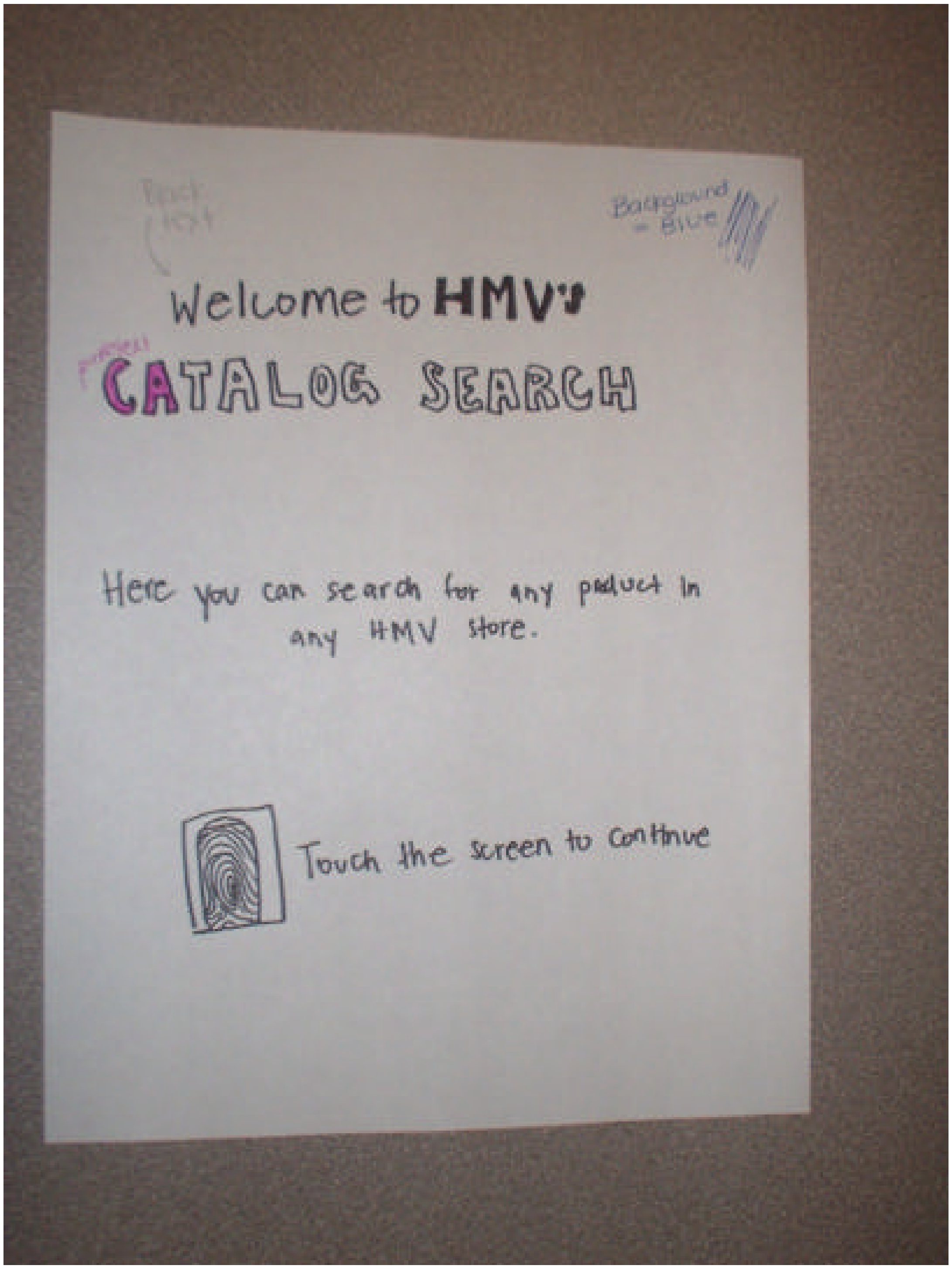
State Diagram Describing Interface Dialog



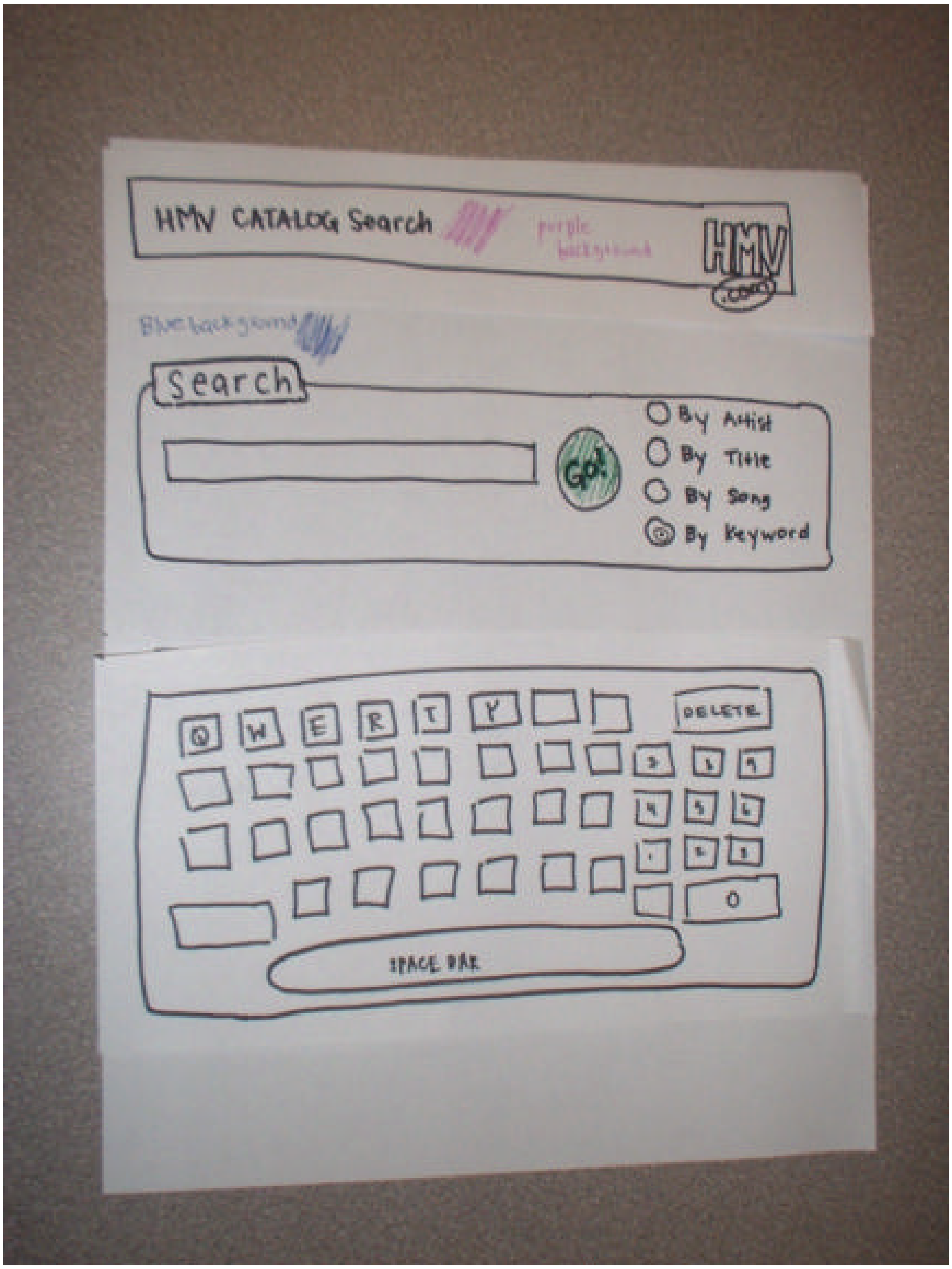
Mockups of Windows

The following paper mock-ups represent low-fidelity prototypes for the appearance of the screen displays of the HMV Kiosk Search System. They consist of the "welcome screen", the "input query" screen, the "display results" screen and the "product details" screen.

The “welcome screen”

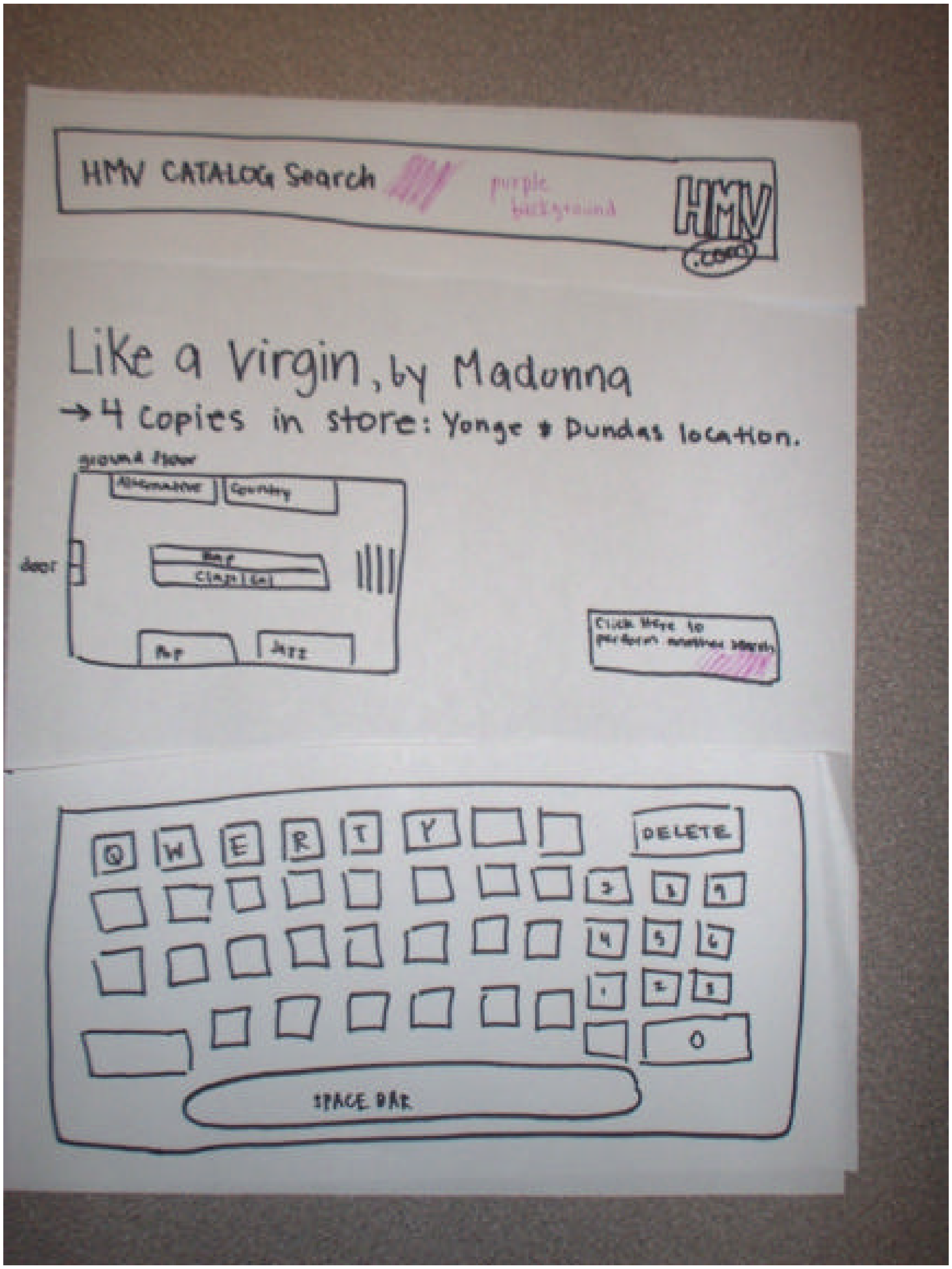

Black text
Background = Blue
Welcome to HMV's
CATALOG SEARCH
Here you can search for any product in any HMV store.
Touch the screen to continue

The "input query" screen



The “display results” screen

The "product details" screen



Input/Output Design

Input in made through the touch screen. There are two main types of input: Data input from the user, and Data capture. Data input from the user gathered in several ways:

1. Touching hotspots on the touchscreen (the keys), allowing the user to type.
2. Entering text in the search field.
3. Selecting a radio button to specify the type of query.
4. Selecting the Go button to submit the query.
5. Selecting links or an advertisement with a link in the data window (the display portion of the screen), bringing the user to the linked page.

The second type of input, data capture, involves the identification of new data sent to the database. At midnight each night, the database is updated so that the inventory and list of products is current. There is not physical input medium for the data capture since the information is transferred over a Wide Area Network. This information is then made available to the kiosks through a Local Area Network.

Output is done through screen displays and reports. The screen display consists of query results of a kiosk Catalog search. This display is an internal output. The medium of this output could be paper is the user requests that their search results be printed. The format of this output is in the form of a well-organized table.

The second type of output is a statistical report. The statistical report summarizes the frequencies of various queries. The reports consist of several formats: pie charts, tables and histograms. Thus, the output format is part graphical, and part narrative. These reports are sent electronically to the HMV Main Office but there are also printed out locally for the store manager to review.

The Relevant Requirements to Interface Design:

Functional Requirements

1. Input Requirements

The system must be able to accept queries from users and must be able to receive information from the stores existing database.

2. Output Requirements

The system must be able to display the results of queries immediately to the user, as well as relevant location information.

Non-Functional Requirements

1. Interface Requirements

The system must be user friendly with an intuitive and unambiguous interaction mechanism. The system must have a familiar design so customers will feel comfortable using the system.

Justification that the Interface Design meets the requirements:

The interface design meets the input requirements in both of the specified respects: it is able to accept queries from the users and it is able to receive information from the store's existing database. The interface is designed to accept queries from the users through the touchscreen. The TouchScreenDialog class gathers the information of the screen contact from the users. As shown in the Dialog Statechart Diagram, the TouchscreenDialog is activated by the users' first contact, and is then ready to accept the users' query. The interface is able to receive information from the store's existing database by communicating with the database classes in a modular way. The Application Component parses the database's results of each query and sends this information to the interface to display to the users.

The interface design meets the output requirements by being able to display the search results to the user, and by being able to display the location information of the selected product. The Dialog Statechart diagram shows that the Result Dialog enters the state 'SearchResults' where it displays the best matches to the users' search. This behaviour fulfills the first part of the requirement. A subsequent state, 'Product Details', is entered if the users select a product. Thus, the requirement of displaying relevant location information to the users is met if and only if the users select a product from the list given as the search results.

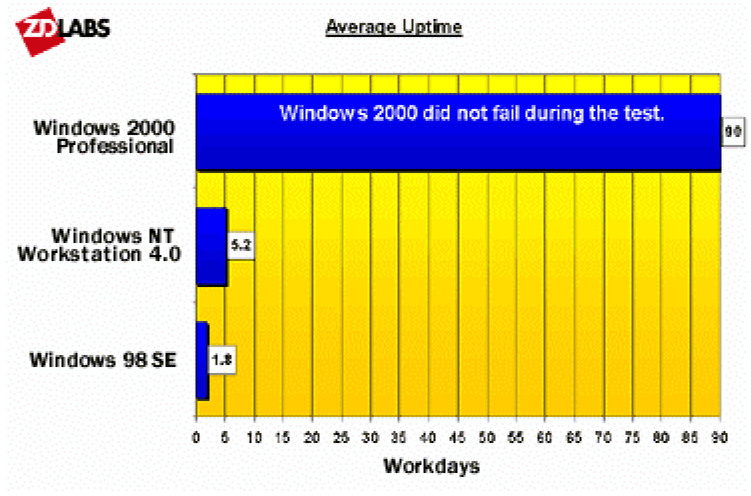
The Interface Requirements are met by the interface design because the system is very user-friendly and easy-to-use. It is made obvious to the user how to interact with the system. The touchscreen affords to be touched since there are no other peripherals for input. The welcome screen of the system shows a graphic of a thumbprint (See Mock ups section), conveying the idea that the screen responds to physical contact. The system is very easy-to-use since it is inspired from internet browser style screens, which users are generally familiar with. The steps of the search are obvious: a keyboard affords touching keys, and so when it is displayed for the first time, the users know this is how they can enter their query; buttons afford being pressed, and the labels of the buttons displayed inform of their functionality. Also, instructions are given to the users: for example, in the SearchResults state, the ResultDialog instructs the users to 'select any Product for inventory details'. These intructions make each step in the search process very obvious to the users.

Conclusion:

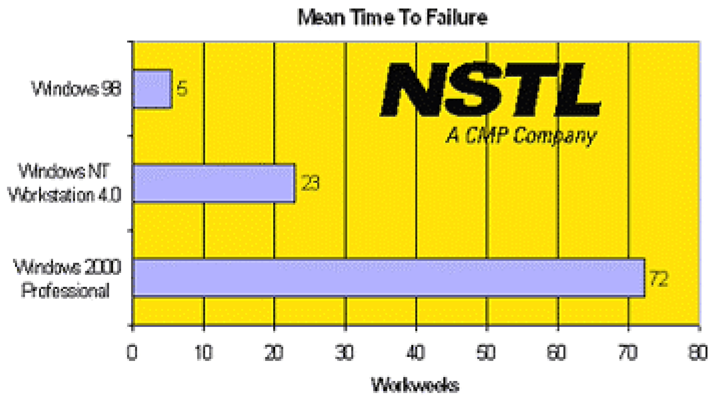
These design recommendations ensure that new HMV Kiosk Search System meets all functional and non-functional requirements specified in the Requirements Analysis Phase (see Appendix B). The new system is fast, efficient, easy to use, and scalable, allowing HMV's continued dominance of media sales in North America. DataPear focused on user needs and on system constraints in the design of our interface, application and database components. The result is a quality system implemented in an economic manner, guaranteeing customer and HMV employee satisfaction.

Appendix A: Justification for Network, Hardware and Software Choices

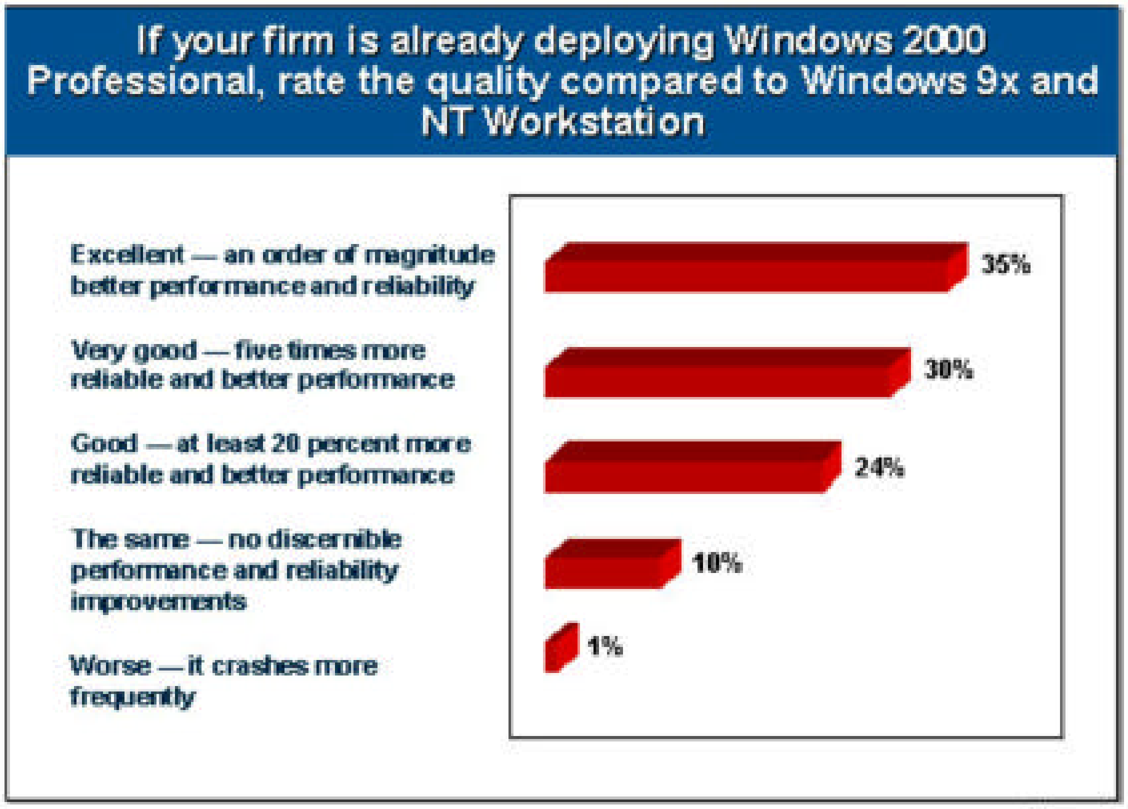
A.1.1 Comparison of Average Uptime Across Windows Operating Systems



A.1.2 Comparison of Mean Time to Failure Across Windows Operating Systems

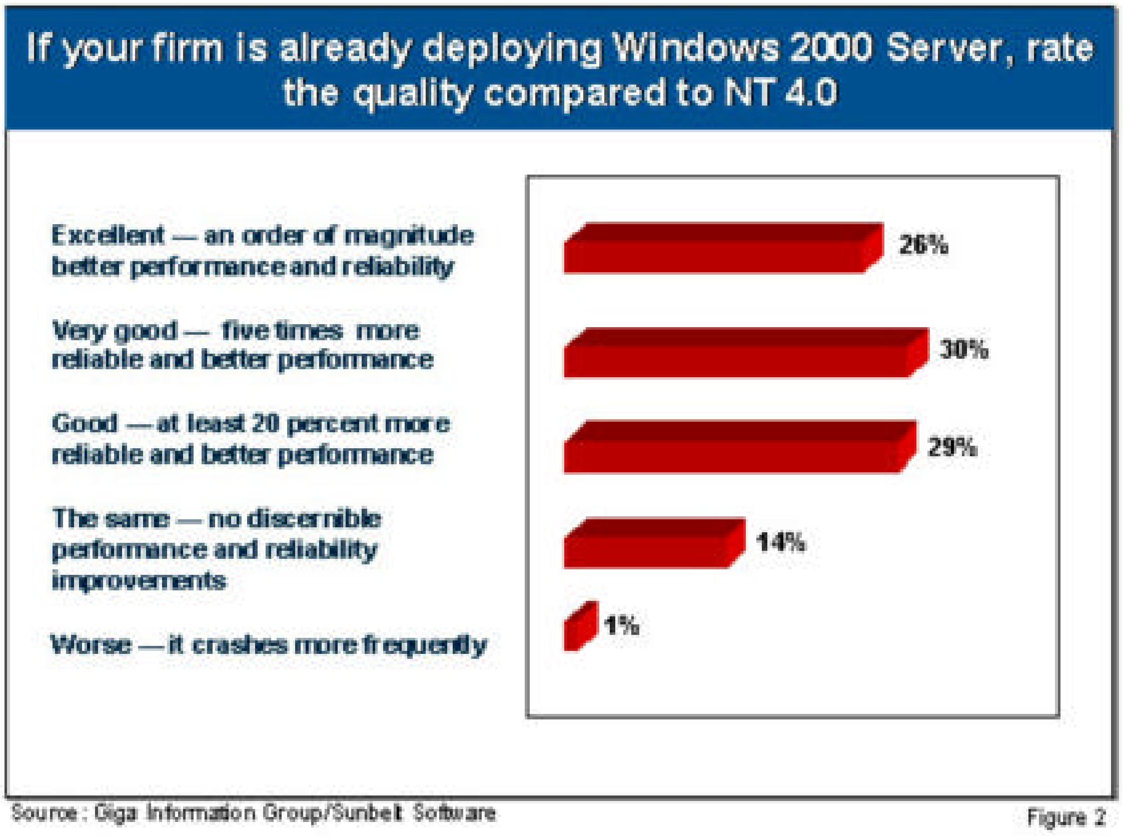


A.1.3 Comparison of Performance Across Windows Operating Systems

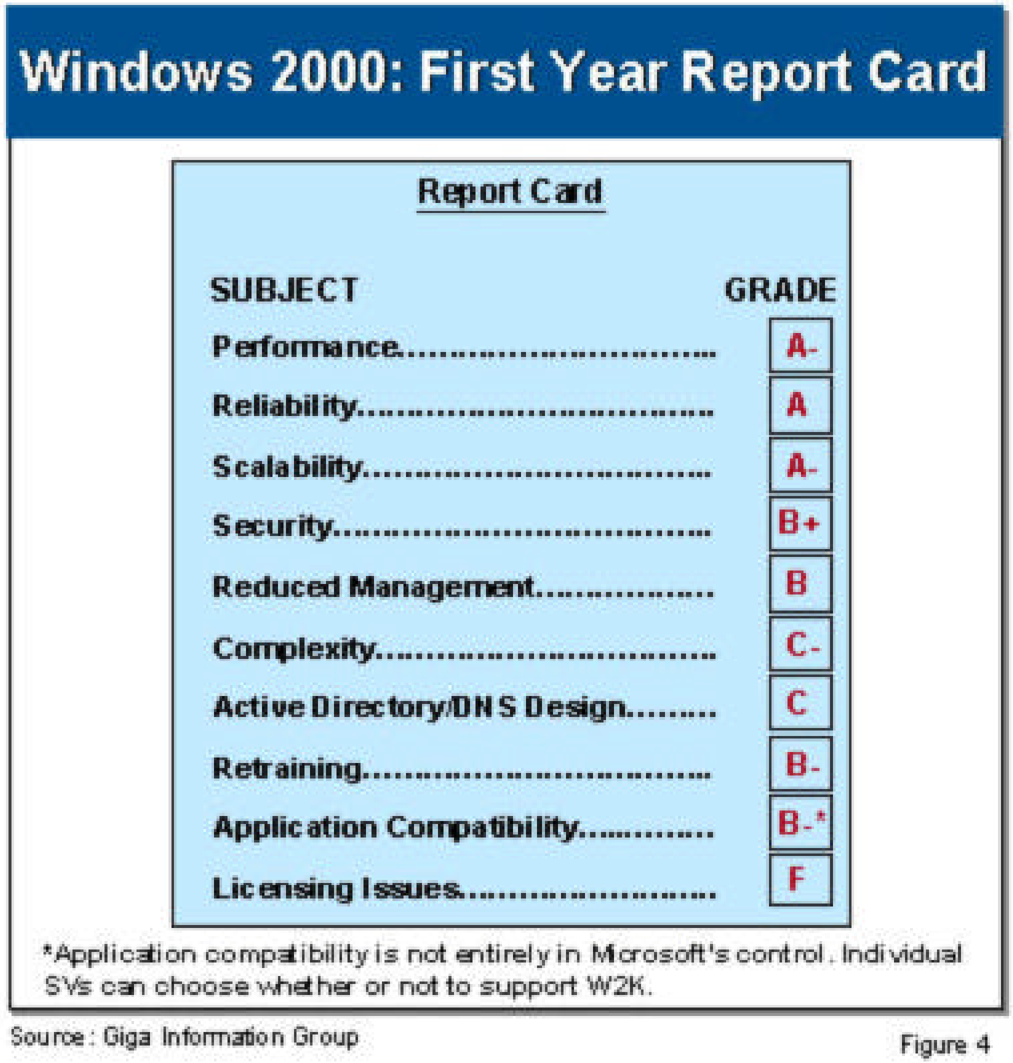


Source: Giga Information Group/Sunbelt Software Figure 1

A.1.4 Comparison of Performance Across Windows Operating Systems



A.2 Windows 2000 Report Card



Windows 2000: First Year Report Card

Report Card

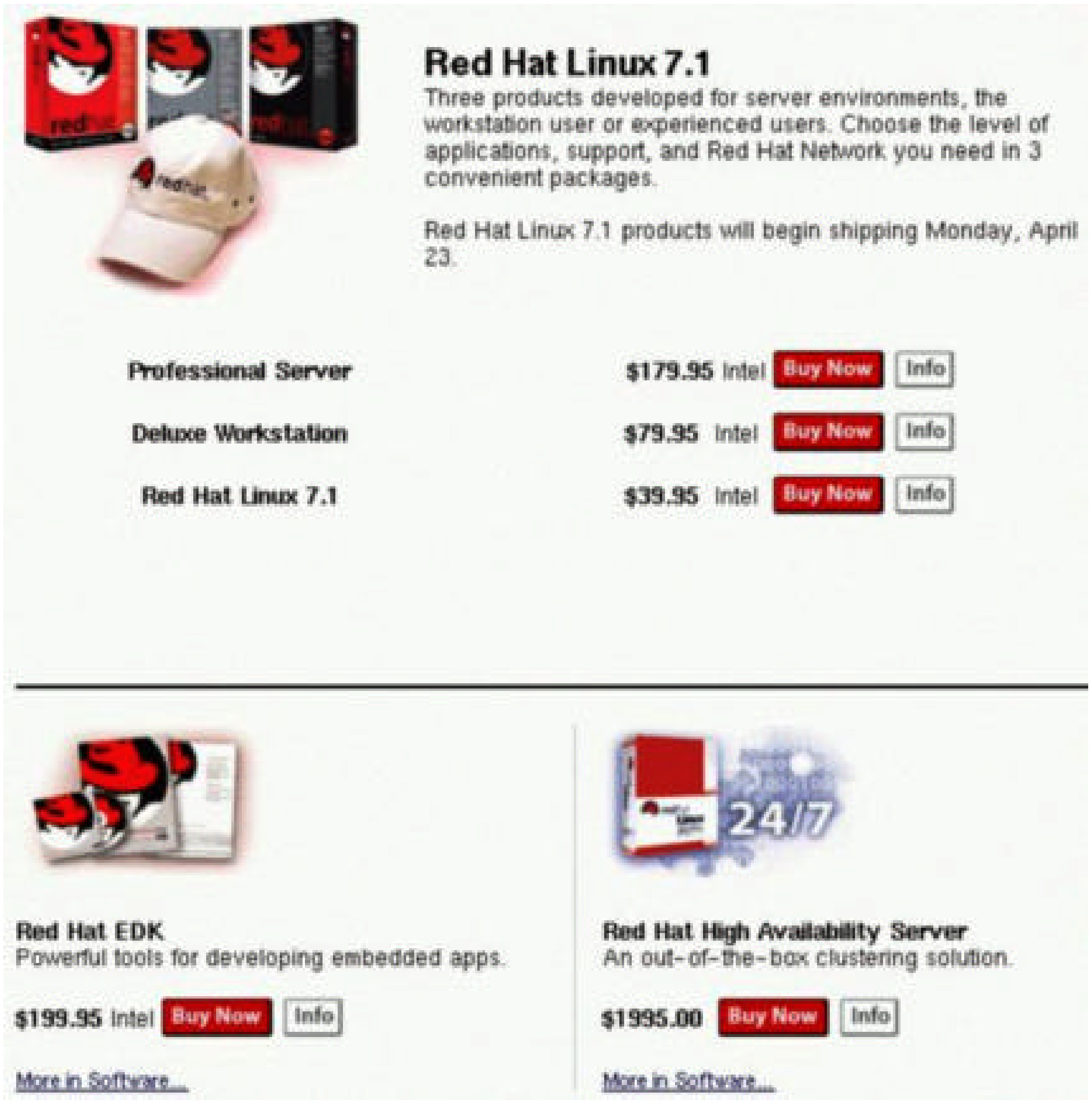
SUBJECT	GRADE
Performance	A-
Reliability	A
Scalability	A-
Security	B+
Reduced Management	B
Complexity	C-
Active Directory/DNS Design	C
Retraining	B-
Application Compatibility	B-*
Licensing Issues	F

*Application compatibility is not entirely in Microsoft's control. Individual SVs can choose whether or not to support W2K.

Source: Giga Information Group

Figure 4

A.3 Details on the Linux Operation System



Red Hat Linux 7.1

Three products developed for server environments, the workstation user or experienced users. Choose the level of applications, support, and Red Hat Network you need in 3 convenient packages.

Red Hat Linux 7.1 products will begin shipping Monday, April 23.

Professional Server	$179.95	Intel	Buy Now	Info
Deluxe Workstation	$79.95	Intel	Buy Now	Info
Red Hat Linux 7.1	$39.95	Intel	Buy Now	Info

Red Hat EDK
Powerful tools for developing embedded apps.

$199.95 Intel Buy Now Info

More in Software...

Red Hat High Availability Server
An out-of-the-box clustering solution.

$1995.00 Buy Now Info

More in Software...

A.4 Possible Touchscreen Choices

Appendix B: Functional and Non-Functional Requirements

Functional Requirements

1. **Processing Requirements**
   The system must be able to translate and relay queries to the stores existing database, and translate and relay the results of the query back to the user.
2. **Input Requirements**
   The system must be able to accept queries from users and must be able to receive information from the stores existing database.
3. **Output Requirements**
   The system must be able to display the results of queries immediately to the user, as well as relevant location information.

Non-Functional Requirements

4. **Interface Requirements**
   The system must be user friendly with an intuitive and unambiguous interaction mechanism. The system must have a familiar design so customers will feel comfortable using the system.
5. **Efficiency Requirements**
   The system must be fast because customers can be very impatient.
6. **Durability Requirements**
   The system must be able to survive heavy daily use, and be resistant to petty vandalism.
7. **Lifecycle Requirements**
   There is no foreseeable obsolescence of the service provided by the system so a long life span is required. The system must be able to have both the hardware and the software upgraded.
8. **Economic Requirements**
   The system must be implemented in over one hundred stores, which means any savings that do not detract from the quality of the system are of great value. Additionally the system should have a low maintenance and replacement cost for the same reason.

Appendix C: Justifications for the Interface Design Decisions

We referred to the textbook “Human Computer Interaction: Toward the Year 2000”, by Baecker, Grudin, Buxton and Greenberg, from the Morgan Kaufmann Publishers, Inc. (California, 1995) In the following section, this book will be referred to as BGBG.

1. Colour in the HMV Kiosk Interface

Friendly warmth is a central concern in designing our website. To achieve this goal, we need a captivating yet calming background colour. We chose a warm shade of blue as background to fulfill this need. In his article “Principles of Effective Visual Communication for Graphical User Interface Design”, Marcus states that blue is good for screen backgrounds since blue-sensitive colour receptors are the least numerous in the retina and are especially infrequent in the eye’s central focusing area, the fovea (BGBG, p.431). Furthermore, our choice of the warm shade of blue is appropriate because "most people experience warm colours advancing toward them- hence forcing attention" (BGBG, p.443).

We choose the text on the page to be black. Black is a darker colour than our blue background; this should make the text clearly visible and easy to read.

We limit our overall colouring of fonts, background and graphics to less than four colours because we have learned in our csc318 Interface Design class that too many colours only make the display ‘busier’ and more confusing rather than more organized. “When too many figures or background fields compete for the viewer’s attention, confusion arises, as can happen in the Las Vegas approach to colour design”. (Marcus, BGBG, p.431)

Our colours are bright yet not too forceful. We chose bright colours because we are aware that "older viewers need higher brightness levels to distinguish colours" (BCBG, p.442).

2. Font in the HMV Kiosk Interface

We use a sans-serif font for the title “HMV Catalog Search”. We use another sans-serif font for the main text. We choose two different fonts because we want our users to be able to easily distinguish between the titles and the text. Sans- serif font is appropriate in both cases since the bodies of text are very short.

We use a much larger font for our main title because we wanted to keep it visible at all times with the purpose of reminding the users of where they are and what the web page is about.

We limit the overall use fonts to two: the main headings and the main text. In this way our users can easily differentiate between the two different classes of information for a better overall conceptual model.

3. Layout in the HMV Kiosk Interface

We inspire ourselves from the Internet Explorer browser look for the layout of our interface. In general, users are familiar with browsers and this will aid them navigate our system.

Appendix D

Summary of Team Meetings

Date	Subject	Attendance	Accomplishment	Time Spent	Homework
Mar 11, 2001	Brainstorming, Discussion on Users, Classes, Functionality	Simon, Danielle, Joshua	Informal Class Diagram, Refinement of solution	2 hours	Everyone develop ideas for Users and Classes
Mar 18, 2001	Brainstorming ideas about, Sequence, State Diagrams	Simon, Danielle, Joshua	Refine Class Diagram, Develop Sequence Diagram	2 hours	Develop a checklist to evaluate if Design meets Functional, Non-Functional Requirements
Mar 25, 2001	Brainstorming ideas about diagrams	Danielle, Joshua, Simon	Refine Class Diagram, Perfect Requirements	2 hours	Develop a list of functional requirements
Apr 12, 2001	Refine diagrams	Danielle, Joshua, Simon	Write up descriptions	3 hours	Danielle: Develop ideas for Sequence Simon: Develop ideas for State Diagrams Joshua: develop glossary for Diagrams
Apr 13, 2001	Develop a Sequence and State Diagram, Gather notes and write final copy Design Deatils	Simon, Danielle, Joshua	Developed Rough Sketch of Sequence and State Diagrams, Developed Functional Requirement support,	7.5 hours	Simon does intro. Danielle will do final copy of all diagrams for submission Joshua will put everything together for the write-up for submission, etc.
Apr 15	Write Report	Simon, Danielle, Joshua		15 hours	

Appendix E: Team Report Form

Description of roles and contributions of each team member:

All team members contributed equally to all aspects of the project.

Name	% of Team Effort
Danielle Lottridge	33 1/3 %
Simon Hatch	33 1/3 %
Joshua Collings	33 1/3 %

Date submitted: Monday, April 16, 2001.