



A SE430 Group Six Final Project Presentation

Presented by:
Nani Comenduley (SoaresPollack), Scott B. Bradley (DL), Larisa A. Bednyak-Kaykov, and Chang S. Kim

Contribution Summary

Ideas and creative input were contributed throughout the project by all group members. The following summary identifies specific areas of responsibility:

Larisa A. Bednyak-Kaykov

- **Technical: Use-Case Realization Sequence Diagrams**
- **Technical: RMS Use Case and Sequence Diagram**
- **Administration: Glossary Updates**

Scott B. Bradley (DL)

- **Technical: Domain Model Diagram**
- **Technical: Design Class Diagram**
- **Technical: Use Case Realization (prose)**
- **Administration: Glossary Updates**

Nani Comenduley (aka Haunani Soares-Pollack)

- **Technical: AMS Use Case and Sequence Diagram**
- **Administration: document formatting & design**
- **Administration: Glossary Updates**

Chang S. Kim

- **Technical: Project Overview**
- **Technical: GMS Use Case and Sequence Diagram**
- **Technical: CRC Cards**
- **Technical: Operation Contracts**
- **Administration: Glossary Creation & Updates**

Table of Contents

INTRODUCTION.....	4
OVERVIEW.....	4
USE CASE DIAGRAM FOR THE DETAILED USE CASES.....	5
GMS DETAILED USE CASE	6
GMS SYSTEM SEQUENCE DIAGRAM.....	7
AMS DETAILED USE CASE	8
AMS SYSTEM SEQUENCE DIAGRAM.....	9
RMS DETAILED USE CASE	10
RMS SYSTEM SEQUENCE DIAGRAM.....	11
OTHER POSSIBLE USE CASES FOR THE SKCS.....	12
DOMAIN MODEL.....	12
DESIGN SCENARIO DESCRIPTION	12
GRASP ANALYSIS	13
OPERATION CONTRACTS	13
DESIGN SCENARIO	14
GRASP ANALYSIS: FURTHER NOTES.....	19
ABSTRACTION SUMMARY	19
CRC CARDS	20
USE CASE REALIZATION SEQUENCE DIAGRAM.....	22
SEQUENCE DIAGRAM 1: USER REQUESTS MEAL GROUPS.....	22
SEQUENCE DIAGRAM 2: CHECK AVAILABLE GROCERIES.....	23
SEQUENCE DIAGRAM 3: GENERATE RECIPE LIST.....	23
SEQUENCE DIGRAM 4: USER SELECTS RECIPE	24
GRASP ABSTRACTIONS CLASS DIAGRAM.....	25
DESIGN PATTERN ANALYSIS.....	25
DESIGN PATTERN CLASS DIAGRAM	ERROR! BOOKMARK NOT DEFINED.7
GLOSSARY	28

Introduction

The Smart Kitchen Central System (SKCS) is designed to manage daily kitchen activities, including grocery inventory maintenance, recipe management and appliance control. The system is designed with today's busy homeowners in mind; simplifying and automating mundane tasks related to preparing daily meals. The system will help plan meals based on what's on hand in the kitchen, or notify them of any out-of-stock ingredients. Interactive displays and remote access to appliances will introduce homeowners to a higher level of enjoyment and quality assurance in the preparation of any meal. The SKCS offers homeowners convenience, safety, and fun.

The system will leverage the Internet for broadened applicability, and define a set of interfaces appropriate for modern electronically controlled appliances. The combination of web-based technologies and appliance integration built into the SKCS will position this system as a potential market leader in consumer-level kitchen control systems. By setting the standard for kitchen technology, manufacturers will join the kitchen automation revolution by manufacturing the SKCS interface compliant devices.

Overview

The drive of the cutting-edge innovation behind the SKCS is the ever-increasing desire of consumers to leverage modern technology to simplify daily routine tasks. The system accomplishes this goal by serving as a central control system for the most essential kitchen needs: (1) grocery management, (2) recipe management and (3) appliance management. The approach to these three basic needs is further detailed below:

Grocery management Subsystem (GMS)

The GMS consists of a specialized device connected to a barcode scanner and software that contains a user interface that features an internet connection. Grocery inventory is managed automatically as the user scans groceries into the system when newly purchased, and also scans them out when used for cooking. Grocery management features include:

- Automatic download of grocery/bar code database. Download updates can be automated or user initiated.
- Automated grocery inventory management through grocery item scan-in and scan-out.
- Internet Access capabilities such as recipe download, coupon download, and partner/sponsor links.
- Manual entry of non-standard grocery items.
- Optional use-date/sell-date entry for perishable items with advanced notice of items needing to be consumed by a given date.
- Automatic shopping list generation based on current grocery stock and selected recipes.
- Automated sending of shopping list and delivery request to partner grocery stores for home delivery service, with web-enabled payment service.

Recipe Management Subsystem (RMS)

The RMS generates complete recipes based on the grocery items in stock to the user. This subsystem houses its own recipe database. The data is input by the user either by creating new recipes or downloading them from the internet. Recipe management features include:

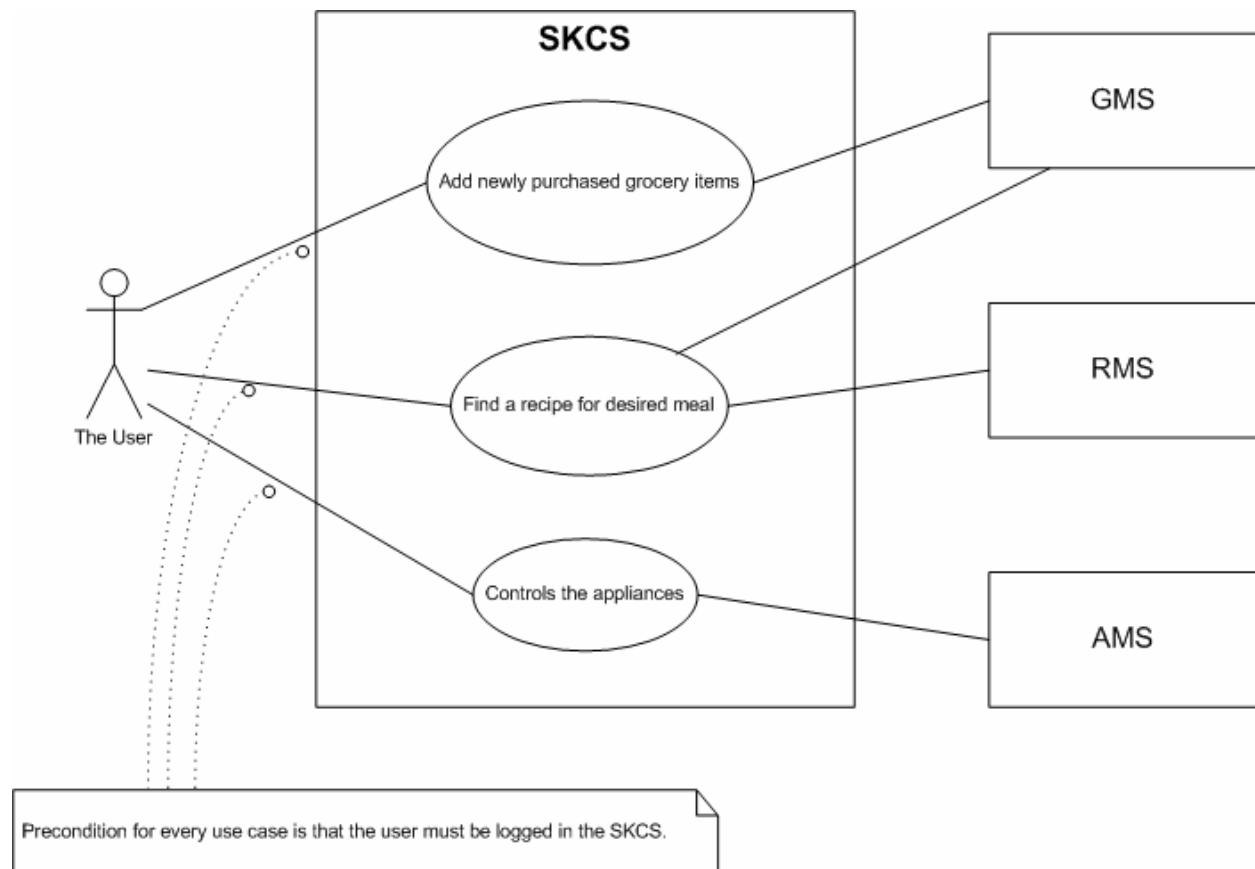
- System suggested recipes based on current grocery stock with related appliances' instruction.
- Manual entry of user's own recipe.
- System provides recipe nutrition information

Appliance Management Subsystem (AMS)

The AMS's main advantage to user will be its remote access feature. Remote access will enable users to check on the current status of each SKCS-enabled appliance, but also enable remote control of each appliance. Using standard protocols, manufacturers will be able to use the SKCS and internet standards to easily create appliances that user will be able to control over the internet. Feature of the AMS include:

- Automatic timed start and temperature set of controlled devices.
- Monitoring of essential appliance data such as temperature, humidity and smoke levels, with options for alarm settings.
- Telephone dial-in with phone keypad based controls
- Internet based remote access.

Use Case Diagram For the detailed use cases



GMS Detailed Use Case



Name: Add newly purchased grocery items
Description: Add newly purchased grocery items information into the Grocery Management Subsystem (GMS).
Actors: The user; the SKCS and the GMS.
Trigger: The user wishes to enter all grocery items which are newly purchased into the GMS.
Preconditions:

1. The GMS must have all possible barcode DB for the grocery items.
2. The grocery items must have valid and readable barcode number.
3. The user must be logged into the SKCS.

Incoming information:

Grocery item's barcode number.

Results:

Grocery item information is entered into the GMS.

Postconditions:

1. The SKCS confirms that the GMS can maintain and update appropriate grocery items information.

Primary Event Flow:

1. The use case begins when the user selects 'Add new groceries' from the SKCS menus.
2. The SKCS requests the last shopping lists to the GMS.
3. The GMS sends back the last shopping list to the SKCS.
4. The SKCS displays the last shopping list.
5. The user selects 'Update' to add grocery items information.
6. The SKCS sends the GMS a message to add all grocery items from the last shopping list to the grocery inventory list.
7. The GMS sends back the updated inventory items list to the SKCS.
8. The SKCS displays the updated inventory items list.
9. The SKCS request the user to confirm the updated inventory items list
10. The use case ends when the user selects 'Confirm'.

Alternate Event Flows:

5a. The user wants to add the different grocery items than the last shopping list:

1. The user selects 'Update different items'.
2. The SKCS displays the barcode reader ready-status.
3. The user scans the grocery item
4. The SKCS displays the grocery item information that just scanned
5. Repeat step 2-4 until the user scans all grocery items.
6. The user selects 'End of adding groceries'
7. Primary event flow continues with step 6.

3a. The user cannot scan the grocery item where the barcode field on the grocery item is damaged:

1. The SKCS informs the user that the grocery item is not readable by the barcode scanner.
2. The SKCS displays the ready-to-input status.
3. The user enters the barcode number of that grocery item.

4. The user selects 'Submit'.
5. The SKCS displays the grocery item information that just entered.
6. Repeat step 2 – 5 until the user entered all grocery items information that is noticed by the SKCS.
7. Event flow continues with alternate event flow step 2 of 5a.

Special requirements:

The GMS must maintain and update the valid and up-to-date barcode DB.

Technology and data variation list:

The GMS must support various barcodes information.

Frequency of occurrence:

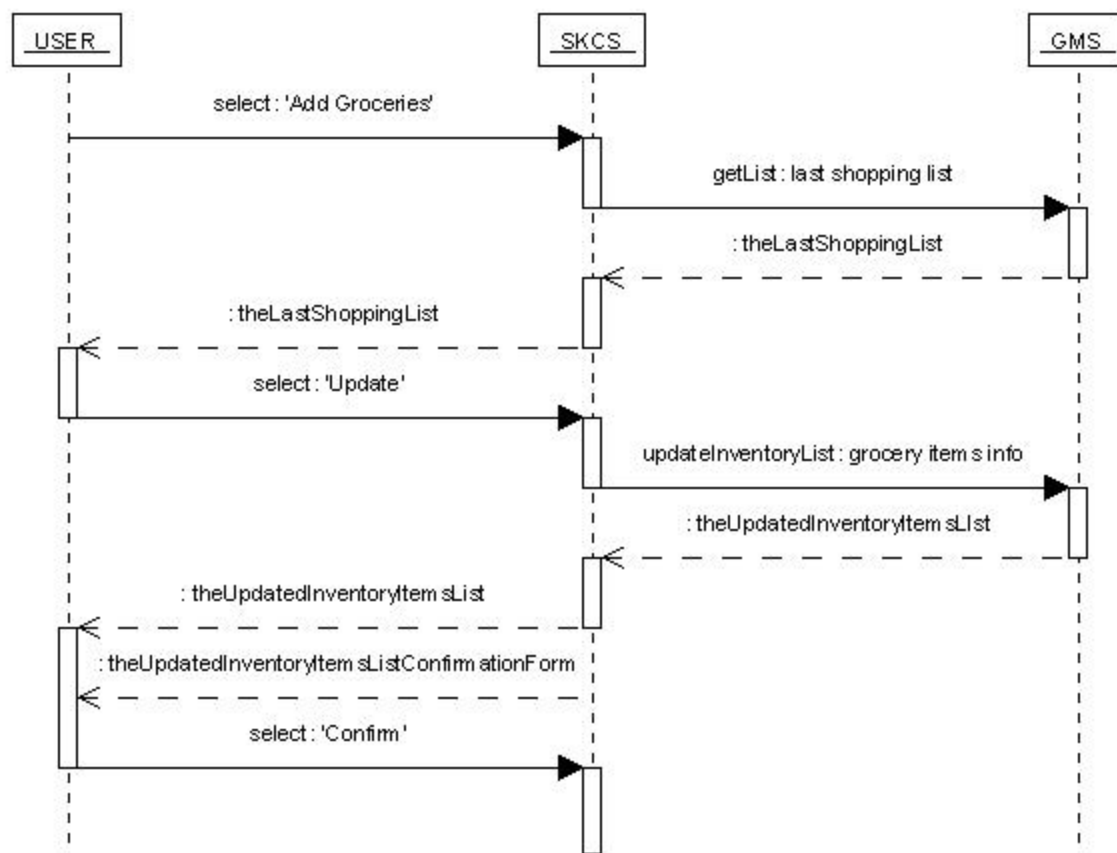
Very high while the user has the grocery items to add either manually or automatically.

Otherwise low.

Open issue:

None

GMS System Sequence Diagram



AMS Detailed Use Case



Name: Controls Kitchen Appliances using the SKC Website
Description: Remote Appliance Control using the Internet
Actors: Customer, SKC, Appliance, AMS, SKC website
Trigger: Customer wants to monitor or control their home appliance remotely
Pre-conditions:

1. The user is logged into the AMS website.
2. The appliance must be connected to the SKCS system

Incoming Information:

Appliance identification, action selection

Post-conditions:

1. Customer has knowledge of the status of their appliances.
2. Customer has updated the status of their appliances.
3. Company has records of customer system usage information.

Primary Event Flow:

1. The use case begins when the customer selects an appliance from a list.
2. SKC requests the current status from the appliance.
3. The appliance send its status to the AMS.
4. AMS displays the current status of the appliance to the customer.
5. AMS prompts the customer for a new status for the appliance.
6. The customer selects a new status for the appliance
7. The AMS sends the new instructions to the appliance
8. The appliance confirms that it is now in the new status.
9. The SKC displays a confirmation message to the customer
10. *Repeat steps 1-9 for each appliance selected.*
11. The AMS records new appliance status and any other visited page information.
12. The use case ends when the customer signs out of the AMS site.

Alternate Flows:

- 6a. The appliance is unable to respond to new requests
 1. AMS sends the appliance a diagnostic request
 2. The appliance runs a diagnostic and sends the result to AMS
 3. AMS displays the results
 4. The user selects another appliance
 5. Continue from step 2

Special Requirements:

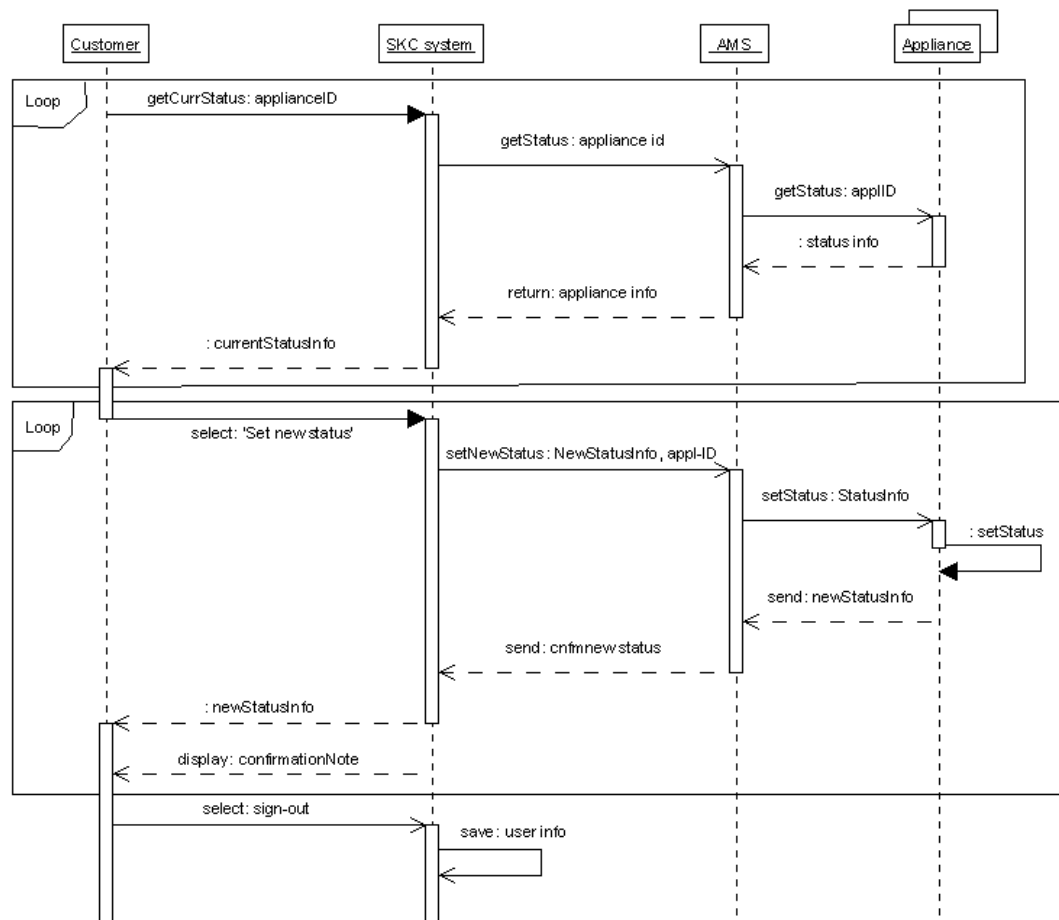
Any appliance to be remotely accessed must be able to interface with the SKC system. The appliances must also be able to conduct basic self diagnostics (i.e., temp, alarm, smoke levels).

Technology and Data variations list:

Frequency of Occurrence:

Multiple events per day

AMS System Sequence Diagram



RMS Detailed Use Case

Name: Find a Recipe for a desired meal
Description: User requests potential recipes based on groceries in stock.
Actors: User, Smart Kitchen Central System, Recipe Management, Grocery Management System
Trigger: User wishes to come up with recipe for a meal
Pre-conditions:

1. User is logged into the system
2. Grocery Management System is stocked with minimum items
3. Recipe Management System comes pre-loaded with recipes

Incoming Information: User's meal category selection

Results: A recipe for a meal

Post-conditions:

1. A recipe containing items in stock right now

Primary Event Flow:

1. The use case begins when the user selects "Find recipe" from the SKCS menu
2. SKCS sends a message to RMS requesting meal categories
3. RMS sends back a list of meal categories
4. SKCS presents the list of meal categories to the user
5. User selects a meal category
6. SKCS sends a chosen meal category to RMS
7. RMS sends required grocery list to GMS.
8. GMS sends back available matching grocery list
9. RMS sends a list of possible recipes to SKCS
10. SKCS presents a list of all recipes possible to make to the user
11. User selects a recipe
12. SKCS sends a request for the recipe selected to RMS
13. RMS sends back a step by step recipe to SKCS
14. The use case ends when SKCS presents the step-by-step recipe instruction to the user.

Alternate Even flow: (User does not like any of the Meal Categories presented)

- 5a. User selects "Meal Search" from SKCS menu
 1. SKCS provides a space for user to enter a name of the recipe they want
 2. User enters name of desired recipe
 3. SKCS sends desired recipe name to RMS
 4. RMS sends back a list of recipes similar to desired recipe to SKCS
 5. SKCS presents a list of matching recipe names to the user
 6. The user selects one recipe from the list
 7. Continue with Step 12 from the primary event flow

Special Requirements:

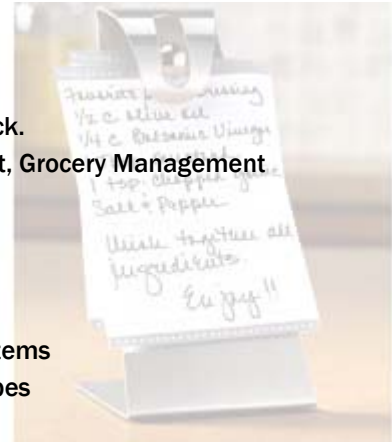
All items must have been entered with quantity

Technology and data variations list:

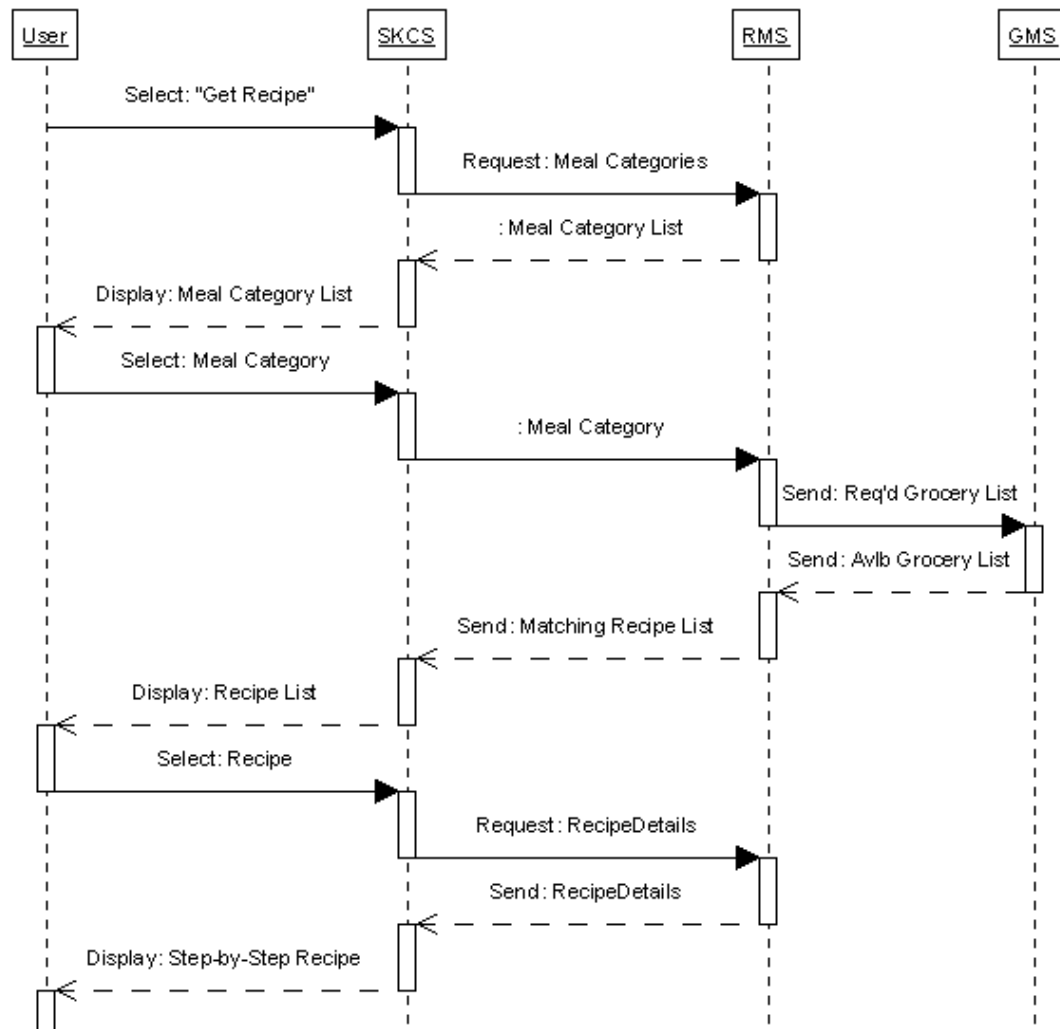
Recipe Management can be updated online with new recipes

Frequency of occurrence:

Every day, 2xday



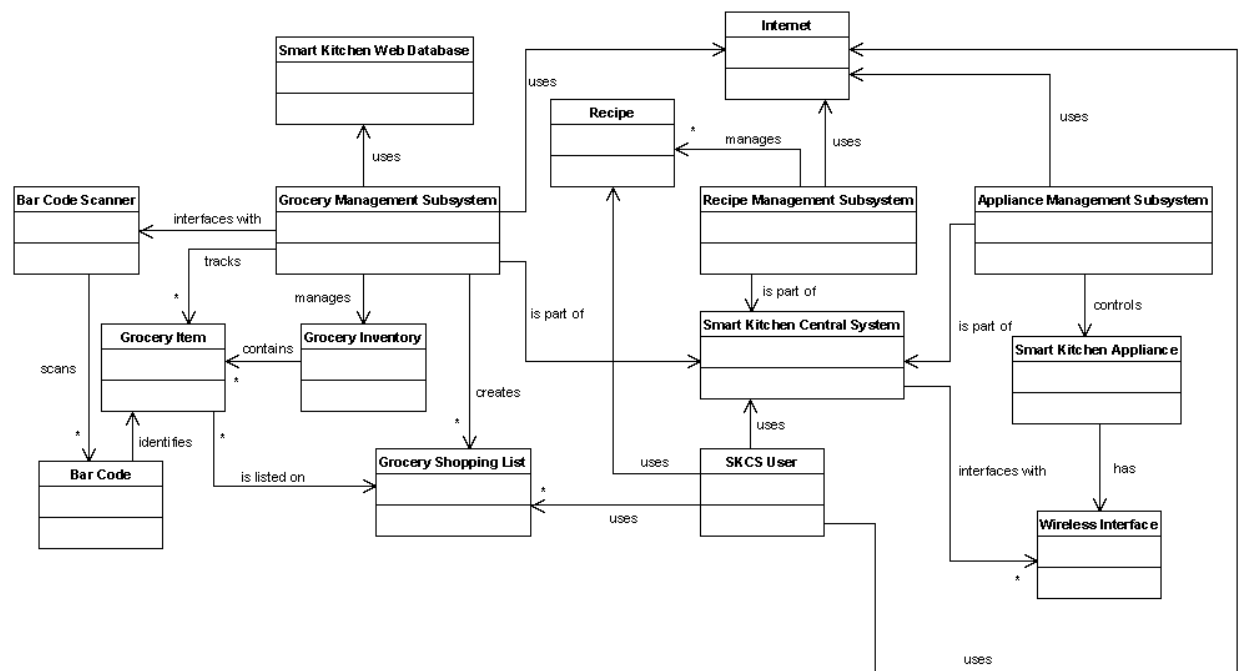
RMS System Sequence Diagram



Other Possible Use Cases For the SKCS

1. Add newly grocery items based on the barcode
2. Find a Recipe for a desired meal
3. Check on the status of the kitchen appliances
4. Add non-standard grocery items
5. Log on to website
6. Deduct grocery usage (system level)
7. Grocery bar code download (system level)
8. Get recipe Nutrition Information
9. Get list of items that are low
10. Generate the shopping list
11. Place an order based on the shopping list.
12. Set appliance temperature
13. Activate meal ready signal

Domain Model



Design Scenario Description

Methodology

The following design scenario ensues from the “Find a Recipe for a Desired Meal” use case, which we might more simply refer to as the “Retrieve Recipe” use case. The approach to creating software abstractions associated with this use case will be a double iteration approach requiring the following stages:

1. Perform a GRASP analysis, determining appropriate responsibilities and assigning them to GRASP abstractions. This is done through a thorough analysis of the individual steps of the use case event flow, determining for each the appropriate responsibilities.
2. Perform a design pattern analysis, re-factoring the GRASP abstractions into a more detailed design-level model according to well-known and proven software design patterns.

For a completed detailed design, further iterations and assignment of fields and methods to the class abstractions would be necessary. These third and fourth stages of the applied methodology, however, lie outside the scope of this document.

GRASP Analysis

Use Case: Find a Recipe for a Desired Meal

We will use the associated use case as the basis for the design scenario. The event flow of the use case is repeated here for clarity:

Primary Event Flow:

1. The use case begins when the user selects “Find recipe” from the SKCS menu
2. SKCS sends a message to RMS requesting meal categories
3. RMS sends back a list of meal categories
4. SKCS presents the list of meal categories to the user
5. User selects a meal category
6. SKCS sends a chosen meal category to RMS
7. RMS sends required grocery list to GMS.
8. GMS sends back available matching grocery list
9. RMS sends a list of possible recipes to SKCS.
10. SKCS presents a list of all recipes possible to make to the user
11. User selects recipe
12. SKCS sends a request for the recipe selected to RMS
13. RMS sends back a step by step, detailed recipe to SKCS
14. The use case ends when SKCS presents the step-by-step recipe instruction to the user

Alternate Event flow: (User does not like any of the Meal Categories presented)

- 5a. User selects “Meal Search” from SKCS menu
 1. SKCS provides a space for user to enter a name of the recipe they want
 2. User enter a name of desired meal
 3. SKCS sends desired recipe name to RMS
 4. RMS sends back a list of recipes similar to desired recipe to SKCS
 5. SKCS presents a list of matching recipe names to the user
 6. The user selects one recipes from the list
 7. Continue with Step 6 from the primary event flow

Operation Contracts

Operation name	select(“Find a recipe”)
Cross reference	Find a recipe for a desired meal
Preconditions	The system must have valid recipe DB
Postconditions	Available recipe list is returned

Operation name	requestMealCategories
Cross reference	Find a recipe for a desired meal
Preconditions	There are valid meal categories
Postconditions	Meal Category List is returned

Operation name	requestGroceryList(selectedMealCategory)
Cross reference	Find a recipe for a desired meal
Preconditions	There are grocery items with that meal category
Postconditions	Available Grocery List is returned

Operation name	select(recipe)
Cross reference	Find a recipe for a desired meal
Preconditions	There are valid instruction for the selected recipe
Postconditions	Recipe instruction is returned

Operation name	requestRecipeInstruction(selectedRecipe)
Cross reference	Find a recipe for a desired meal
Preconditions	Selected recipe's instruction must be existed
Postconditions	Recipe instruction is returned

Design Scenario

1. The use case begins when the user selects "Get recipe" from the SKCS menu

Responsibility: Capture user input

Design Thoughts: A UI layer will be implemented based on the model-view-controller pattern. In this configuration, the model will be the class which handles message controls to the RMS.

Assignment:

- SKCSView abstraction to represent the graphical user interface.
- SKCSViewController to observe the view and to update the RMS.

Responsibility: Represent a Recipe

Design Thoughts: A recipe object will need to be created which can contain the recipe to be returned to the user. While the recipe itself will be static text, the object might have fields representing recipe meta-data such as the title or the meal group.

Assignment:

- Recipe

Responsibility: Create (that is, instantiate) the recipe object

Design Thoughts: GRASP analysis indicates that the responsibility of object creation is handled by a factory abstraction.

Assignment:

- RecipeFactory

2. SKCS sends a message to RMS requesting meal categories

Responsibility: Forwarding of captured user input to appropriate subsystem

Design Thoughts: From the simple model so far (UI – RecipeFactory – Recipe), and from our knowledge of the overall system architecture, with the RMS and GMS subsystems, it is clear that a controller will be required for each of the subsystems. We will delegate this functionality to a Dispatcher object.

Assignment:

- RMSDispatcher
- GMSDispatcher

Responsibility: Request creation of a meal group list

Design Thoughts: This will be handled by the RMSDispatcher which serves as a controller to the RMS.

Assignment:

- RMSDispatcher

Responsibility: Create a MealGroupList

Design Thoughts: The list of meal groups will itself be an object apart from the list of recipes according to group. This is because the user interface allows the user to select a category of meals first, requiring interaction with the group list, then to select a recipe from the group.

Assignment:

- MealGroupList
- MealGroupListFactory

3. RMS sends back a list of meal categories

Responsibility: Return the MealGroupList object

Design Thoughts: This will be handled by the RMSDispatcher

Assignment:

- RMSDispatcher

4. SKCS presents the list of meal categories to the user

Responsibility: Display information returned by the subsystem

Design Thoughts: All user display will be handled by the UI layer which will see data only abstractly. Display is provided by the view which is updated by the view controller

Assignment:

- SKCSView
- SKCSViewController

5. User selects a meal category

Responsibility: Select item from UI list

Design Thoughts: This is a user responsibility

Design Thoughts: Assignment:

- SKCS User (not to be abstracted)

6. SKCS sends a chosen meal category to RMS

Responsibility: Handle user selection from abstract list

Design Thoughts: According to the model-view-controller pattern, the view is observed by the controller which in turn updates the model.

Assignment:

- SKCSViewController

Responsibility: Dispatch meal category selection action.

Design Thoughts: All RMS related activities will be managed by the RMSDispatcher

Assignment:

- RMSDispatcher

7. RMS sends required grocery list to GMS

Responsibility: Pass GroceryList object to the GMS subsystem

Design Thoughts: The subsystems of the SKCS are primarily conceptual in nature. They are based on divisions of functionality rather than any inherent divisions in the system's physical architecture. Therefore, it is assumed at this point that the subsystem controllers should have direct access to one another for the passing of related messages.

Assignment:

- The RMSDispatcher will handle this responsibility

8. GMS sends available matching grocery list

Responsibility: Representation of all grocery items in stock

Design Thoughts: This is another example of a grocery list. Thus the GroceryListFactory and GroceryList abstraction should be accessible also to the GMS. This is a GMS requested grocery list because it represents grocery stock rather than recipe-related groceries. The intersection of these grocery lists will result in a list of groceries available for the desired recipe list.

Assignment:

- GroceryListFactory
- GroceryList

Responsibility: Filter the current grocery list according to available groceries in stock

Design Thoughts: It is more likely that the newly instantiated GroceryList will be one built dynamically from a match of needed groceries to in-stock groceries (rather than building two complete lists and then applying a filter). At this level of analysis, such information sourcing is handled by the information expert.

Assignment:

- SKCSInformationExpert
- SKCSDataSource

Responsibility: Return grocery list object to the RMS

Design Thoughts: The GMSDispatcher handles forwarding of messages into and out of the GMS subsystem

Assignment:

- GMSDispatcher

9. RMS sends a list of possible recipes to SKCS

Responsibility: Request construction of a list of recipes

Design Thoughts: The RMSDispatcher handles controls for recipe management.

Assignment:

- RMSDispatcher

Responsibility: Construct a list of available recipes based on recipe group and available groceries

Design Thoughts: Again, consideration is for another type of list object created by an object factory, utilizing the services of an information expert.

Assignment:

- RecipeListFactory
- RecipeList
- SKCSInformationExpert
- SKCSDataSource

10 SKCS presents a list of all recipes possible to make to the user

Responsibility: Provide display update information to the UI layer.

Design Thoughts: The RMSDispatcher is the model within the model-view-controller pattern which provides updates to the view

Assignment:

- RMSDispatcher

Responsibility: Display abstract selectable list to the user.

Design Thoughts: General display of abstract information to the user.

Assignment:

- SKCSView

11. The user selects a recipe from a list presented on the SKCS menu.

Responsibility: Select recipe

Design Thoughts: This is a user responsibility

Assignment: SKCS User (not abstracted)

12. SKCS sends a request for the recipe selected from to RMS.

Responsibility: Handle user selection from abstract list

Design Thoughts: The SKCSViewController provides updates of the user interface to the RMS

Assignment: SKCSViewController

13. RMS sends back a step by step recipe to SKCS.

Responsibility: Request creation of a recipe object.

Design Thoughts: The RMSDispatcher handles controls for recipe management.

Assignment:

- RMSDispatcher

Responsibility: Provide instantiation of a recipe object

Design Thoughts: We recognized the need for this responsibility above in step 1. However, it is only here that the recipe will finally be instantiated.

Assignment:

- RecipeFactory
- Recipe

Responsibility: Forward recipe text for display

Design Thoughts: The RMSDispatcher handles updating of display view according to model-view-controller pattern.

Assignment:

- RMSDispatcher

14. The use case ends when the SKCS presents the step-by-step recipe instruction to the user.

Responsibility: Display abstract text

Design Thoughts: The view provides all information display to the user

Assignment: SKCSView

Alternate Event Flow:

5a. User selects "Meal Search" from SKCS menu

5.a.1. SKCS provides a space for user to enter a name of the recipe they want

Responsibility: Display text box.

Design Thoughts: This is a responsibility of the view.

Assignment: SKCSView.

5.a.2. User enters a name of the meal

Responsibility: Enter a keyword.

Design Thoughts: This is a user responsibility.

Assignment: SKCS User (not to be abstracted)

5.a.3. SKCS sends name of requested meal to RMS

Responsibility: Observe the view and update the RMS

Design Thoughts: This is assigned according to the model-view-controller pattern.

Assignment: SKCSViewController

5.a.4. RMS sends back a list of recipes similar to meal requested to SKCS

Responsibility: Request creation of a recipe list.

Design Thoughts: RMSDispatcher handles management of recipe-subsystem.

Assignment:

- RMSDispatcher

Responsibility: Create the recipe list

Design Thoughts: A factory will handle instantiation of the concrete list object

Assignment:

- RecipeListFactory
- RecipeList

Responsibility: Return the list to the SKCS

Design Thoughts: The RMSDispatcher updates the UI layer view.

Assignment:

- RMSDispatcher

5.a.5. SKCS presents a list of matching recipe names to the user

Responsibility: Display abstract list to the user.

Design Thoughts: This is a result of the update provided by the RMSDispatcher to the view

Assignment: SKCSView

5.a.6. The user selects a recipe from the list

Responsibility: Select item from abstract list.

Design Thoughts: This is a user responsibility.

Assignment: SKCS User (not to be abstracted)

5.a.7. Continue with Step 6 from the primary event flow

GRASP Analysis: Further Notes

We have deferred discussion of the nature of data persistence in our system. Some observations should be noted from our work so far:

- System must be able to store recipes, their associated groceries, their associated groups, and related abstract forms (such as a recipe title, or full recipe text, or short recipe description).
- The system must be able to store grocery items, including:
 - All possible groceries
 - All groceries in stock
 - All groceries needed for a recipe
- Subsystems must be able to access persisted data independently

This is clearly a candidate for a database-oriented solution. However, at this juncture, we will not commit to a particular technological implementation of data persistence. Rather, the information expert has been assigned for this responsibility, according to standard GRASP analysis methodology to handle data sourcing.

Abstraction Summary

The abstractions gathered from the GRASP analysis are summarized here for clarity:

- SKCSView
- SKCSViewConroller
- RMSDispatcher
- GMSDispatcher
- MealGroupListFactory
- MealGroupList
- RecipeListFactory
- RecipeList
- RecipeFactory
- Recipe
- GroceryListFactory
- GroceryList
- SKCSInformationExpert

CRC Cards

SKCSView	
<i>Responsibility</i>	<i>Collaborators</i>
Capture user input	SKCS User
Provide input data	SKCSViewController
Display updates from the model (RMS)	RMSDispatcher

SKCSViewController	
<i>Responsibility</i>	<i>Collaborators</i>
Observe the view	SKCSView
Update the model (RMS)	RMSDispatcher

RMSDispatcher	
<i>Responsibility</i>	<i>Collaborators</i>
Update the view	SKCSView
Receive updates from the view controller	SKCSViewController
Request creation of a recipe	RecipeFactory
Request creation of a meal group list	MealGroupListFactory
Request creation of a grocery list	GroceryListFactory
Request creation of a recipe list	RecipeListFactory
Request grocery-stock related services	GMSDispatcher
Get meal group list	MealGroupList
Get recipe list	RecipeList
Get grocery list	GroceryList
Get recipe	Recipe

GMSDispatcher	
<i>Responsibility</i>	<i>Collaborators</i>
Request creation of a grocery list	GroceryListFactory
Receive RMS requests	RMSDispatcher
Get grocery list	GroceryList

MealGroupListFactory	
<i>Responsibility</i>	<i>Collaborators</i>
Accept request to create a meal group list	RMSDispatcher
Create a meal group list	MealGroupList
Request data required for meal group instantiation	SKCSInformationExpert

MealGroupList	
<i>Responsibility</i>	<i>Collaborators</i>
Represent meal group list by instantiation	MealGroupListFactory
Provide meal group list	RMSDispatcher

RecipeListFactory	
<i>Responsibility</i>	<i>Collaborators</i>
Accept request to create a recipe list	RMSDispatcher
Create a recipe list	RecipeList
Request data required for recipe list instantiation	SKCSInformationExpert

RecipeList	
<i>Responsibility</i>	<i>Collaborators</i>
Represent a recipe list by instantiation	RecipeListFactory
Provide recipe list	RMSDispatcher

RecipeFactory	
<i>Responsibility</i>	<i>Collaborators</i>
Accept request to create a recipe	RMSDispatcher
Create a recipe	Recipe
Request data required for recipe instantiation	SKCSInformationExpert

Recipe	
<i>Responsibility</i>	<i>Collaborators</i>
Represent a recipe by instantiation	RecipeFactory
Provide recipe	RMSDispatcher

GroceryListFactory	
<i>Responsibility</i>	<i>Collaborators</i>
Accept request to create a grocery list	RMSDispatcher
Accept request to create a grocery list	GMSTDispatcher
Create a grocery list	GroceryList
Request data required for grocery list instantiation	SKCSInformationExpert

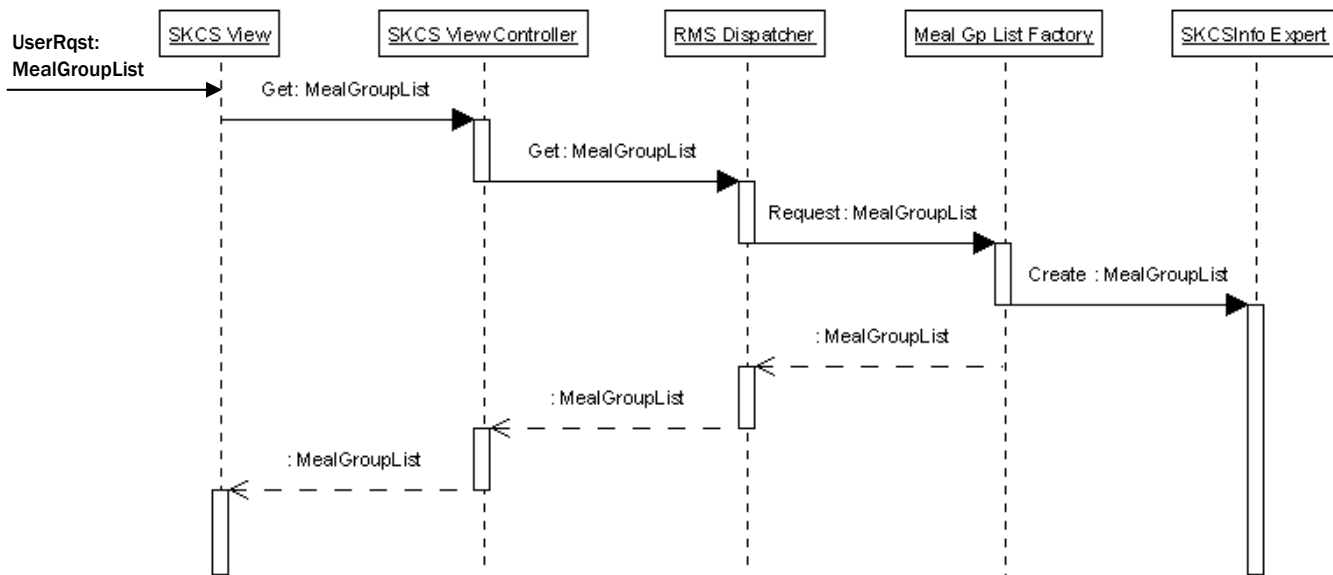
GroceryList	
<i>Responsibility</i>	<i>Collaborators</i>
Represent a grocery list by instantiation	GroceryListFactory
Provide grocery list	RMSDispatcher
Provide grocery list	GMSTDispatcher

SKCSInformationExpert	
Responsibility	Collaborators
Accept data request	MealGroupListFactory
Accept data request	RecipeListFactory
Accept data request	RecipeFactory
Accept data request	GroceryListFactory
Request data from data source	SKCSDataSource

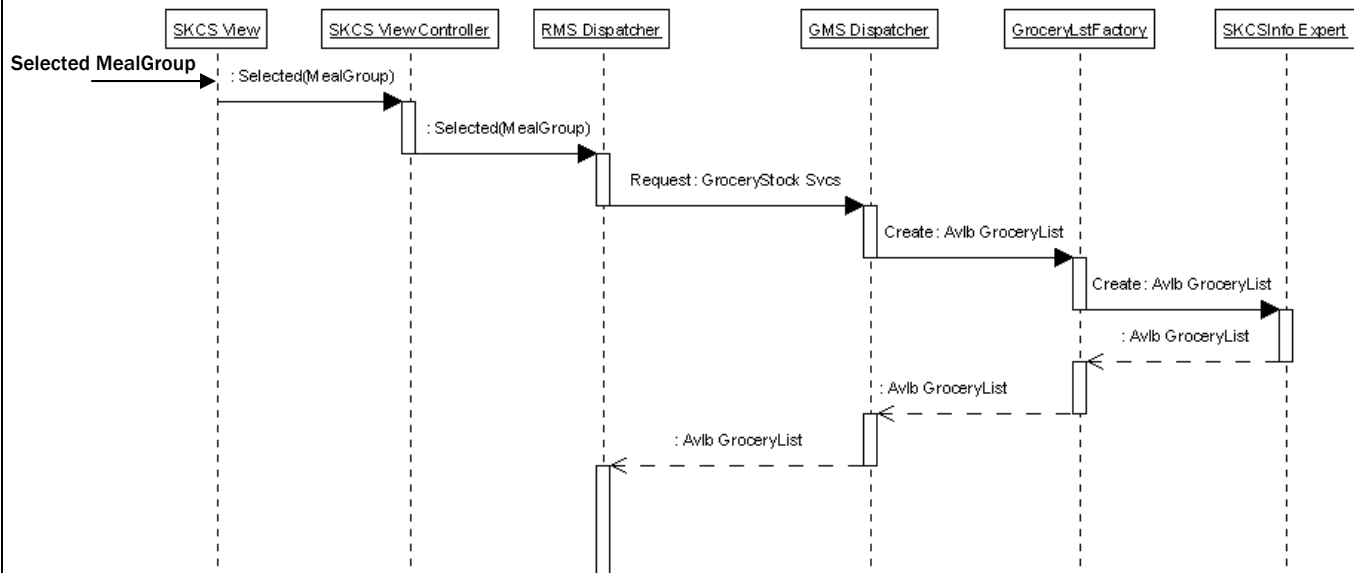
SKCSDataSource	
Responsibility	Collaborators
Provide requested data	SKCSInformationExpert

Use-case Realization Sequence Diagram

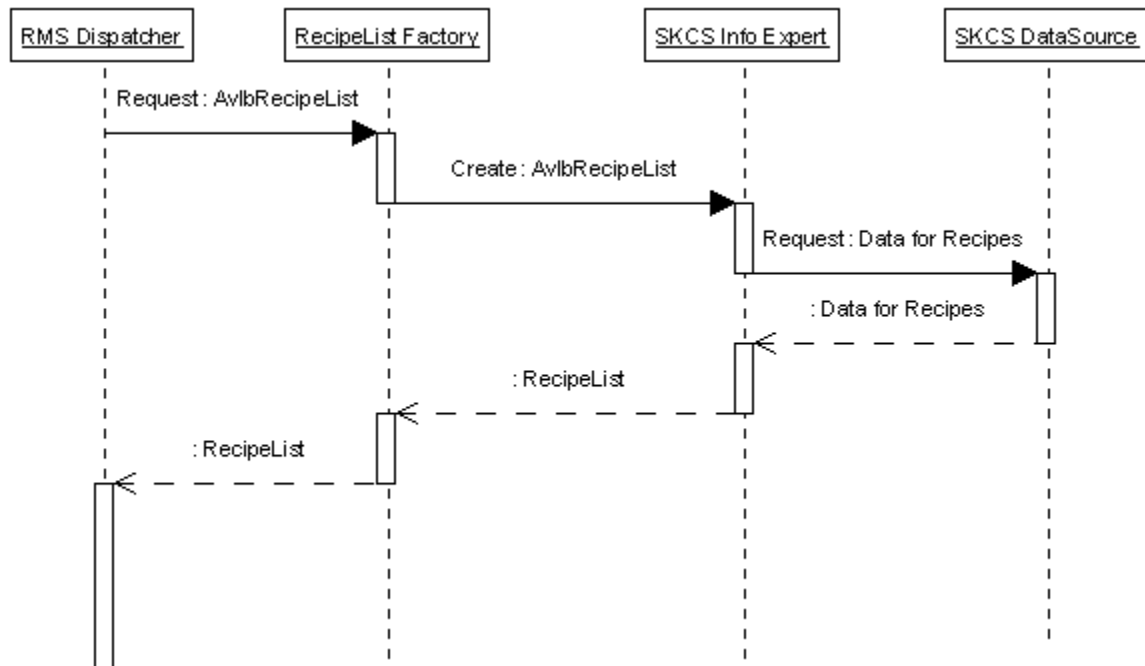
Sequence Diagram 1: User Requests Meal Groups



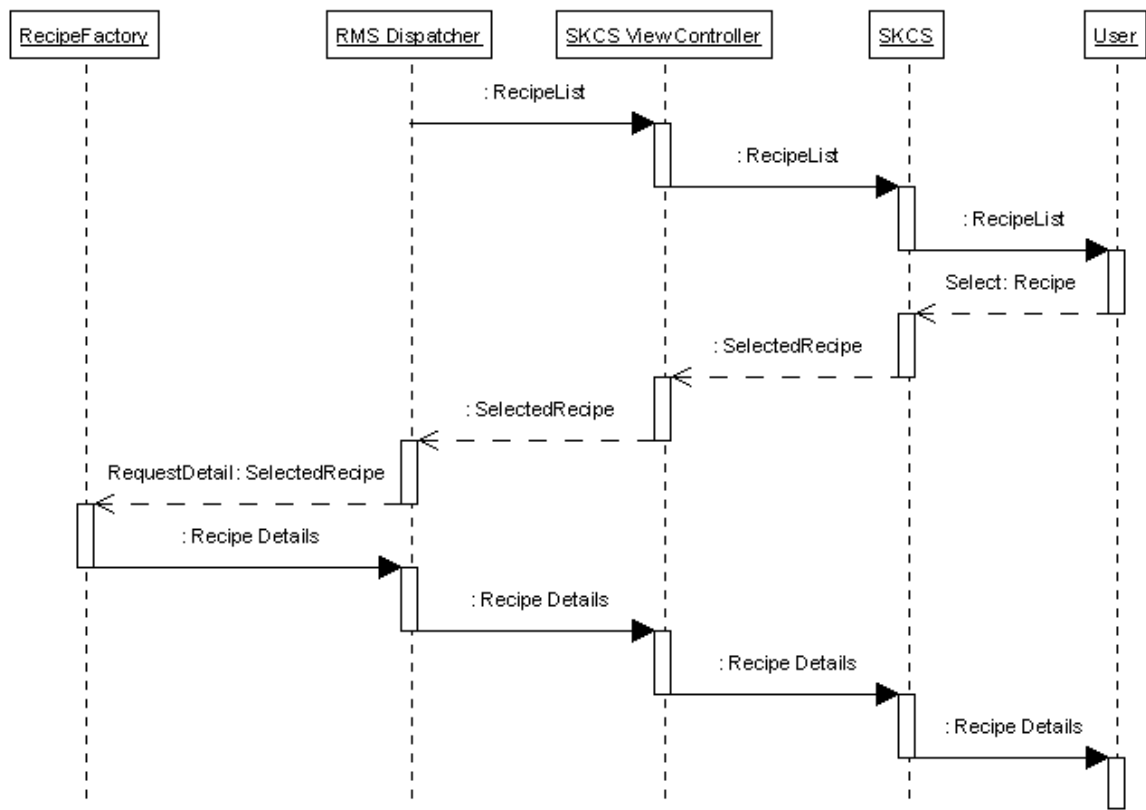
Sequence Diagram 2: Check Available Groceries



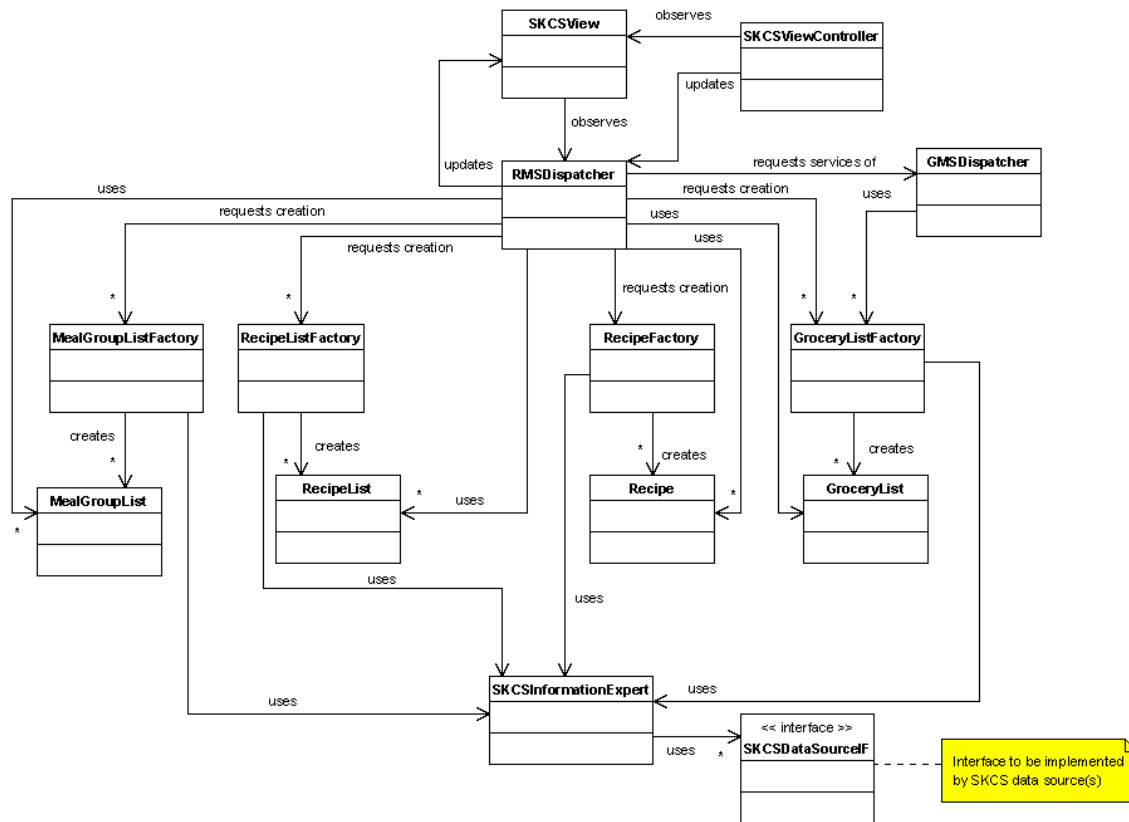
Sequence Diagram 3: Generate Recipe List



Sequence Diagram 4: User Selects Recipe



Class Diagram



GRASP Abstractions Class Diagram

Design Pattern Analysis

Software design patterns provide a way in which to implement proven designs which are known to adhere to fundamental object-oriented principles. Full detail of the design pattern analysis will not be documented as was the GRASP pattern analysis stage, however some notes on particular GRASP abstractions will help to illuminate the transition from a GRASP pattern architecture to one based on software design patterns. These notes follow:

UI Layer:

A mediator will be needed to handle UI events and to pass them to the system. Since each subsystem will have its own controller and since the full scope of use case across the system has not yet been assessed, yet is certain to have a great deal of complexity (judging from the current use case), a mediator model by which the mediator serves to handle a particular use case will be used. For more information on this approach to mediation, see Larman, p. 303. This mediator will be called RetrieveRecipeMediator, and will pass messages from the UI layer to the RMSDispatcher. This mediator will need to implement the interfaces necessary for listening to the UI layer elements.

The UI Layer will also need to handle the following widgets for this use case, each of which is identified as a needed abstraction:

- RetrieveRecipeButton

- RetrieveRecipeListBox
- RetrieveRecipeTextBox

Note that the list and text boxes will deal only with abstract data, thus allowing dynamic manipulation of that data. This manipulation is handled outside the UI Layer.

The RMSDispatcher:

As shown in the GRASP analysis, the RMSDispatcher will handle distribution of messages throughout the RMS, particularly interfacing to the GMS, the various RMS-related class factories, and to the UI Layer (through the aforementioned RetrieveRecipeHandler mediator class).

The RMS in general:

Looking again at the factories that are needed, it is determined that there are two types of data classes being generated in this use case: dynamic list data, and static text data. These will be common data types also needed by other use cases. Two factory interfaces are proposed:

- ListFactoryIF
- TextFactoryIF

Using the Factory Method design pattern, the following abstractions result:

- MealGroupListFactory
- MealGroupList
- RecipeListFactory
- RecipeList
- GroceryListFactory
- GroceryList
- RecipeTextFactory
- RecipeText
- AbstractList
- ListIF
- AbstractText
- TextIF

Note that in this particular use case, the grocery list is not passed as an abstract list to the user interface, but is merely used by the system for data manipulation. In other use cases, however, a grocery list object might in fact need to pass abstract data to the user interface.

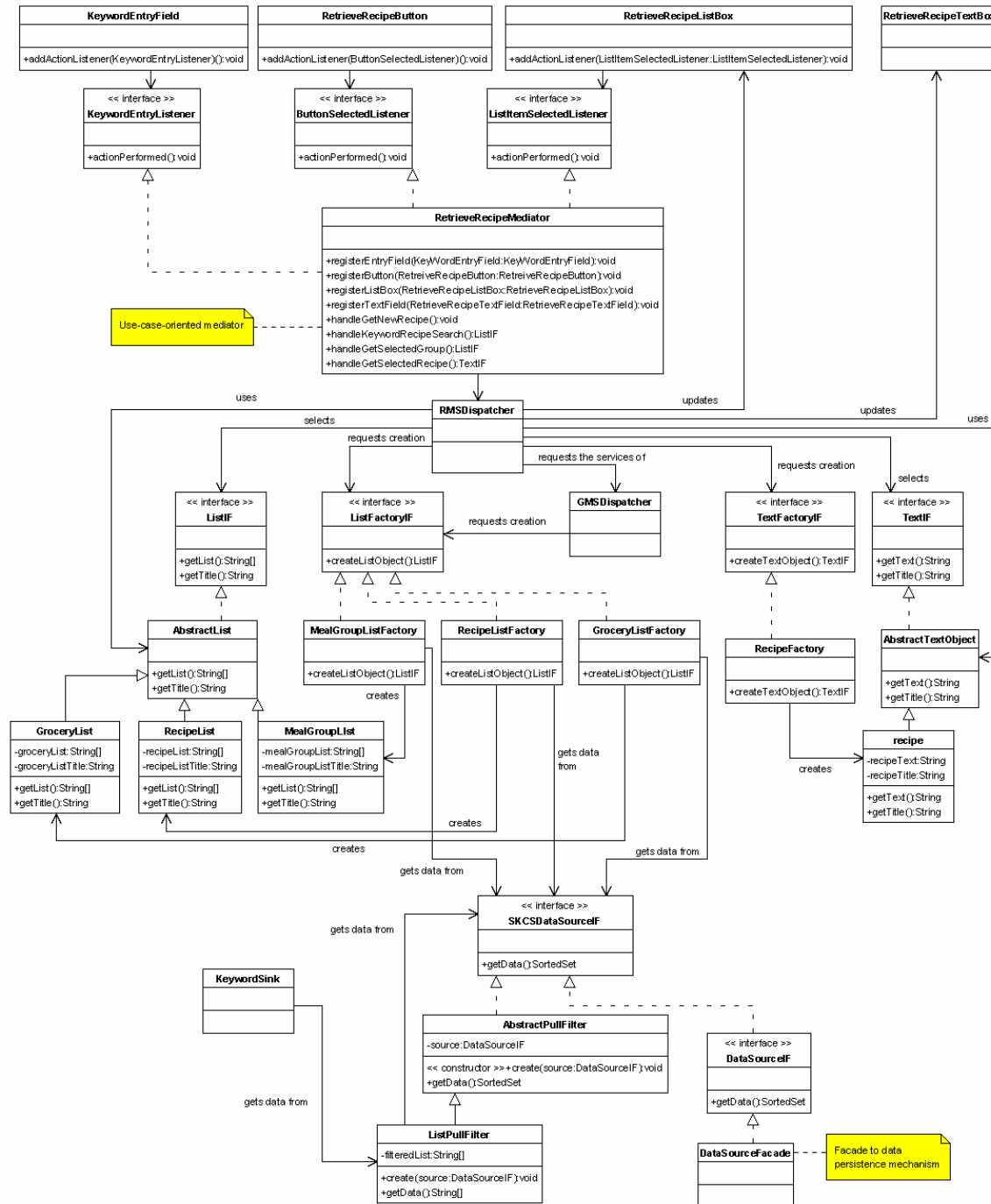
The Information Expert

This GRASP level abstraction must now be detailed according to known design patterns. In particular, we know that we will need functionality to pull data from a still unknown data source, as well as to filter that data according to known lists and keywords. To achieve this functionality, we will create a data source interface which must be implemented by any data source objects providing services to the SKCS subsystems. The filter pattern will provide the data filtering functionality required and a data source interface coupled with a data source façade will provide a data source implementation which is decoupled from the SKCS subsystems and which may potentially be made up of multiple components. From the related patterns, the following classes result:

- SKCSDataSourceIF
- AbstractPullFilter
- ListPullFilter

- KeywordSink
- DataSourceIF
- DataSourceFacade

Class Diagram



Design Patterns Class Diagram

Glossary

Appliance Refers to a kitchen appliance that is able to interface with the AMS subsystem. An appliance in this domain is able to send and receive information from the AMS.

Appliance Management Sub-System (AMS):
Manages appliances in the SKCS

Barcode DB: Keeps valid and up-to-date barcode information. The user can get downloaded and updated periodically through Internet.

Barcode number:
An identifier for each grocery item

Barcode Scanner:
A physical part of the system, but it is built into the SKCS. It is used to scan in grocery items.

GMSDispatcher
Serves as a controller for GMS, delegates tasks and responsibilities within and outside the GMS

Grocery Item:
An item Smart Kitchen System user brings home and wants to add to their stock of items. It could be delivered.

Grocery Inventory List:
A list of current items in stock

GroceryList
Generates a list of groceries needed for a recipe

GroceryListFactory
Generates a list of available groceries

Grocery Management System (GMS):
A subsystem of the Smart Kitchen Central System used to manage groceries

Grocery Shopping List:
A list created by the user, contains all the items they wish to buy at the store, it is kept in the GMS. It is based on barcode information of grocery items.

MealGroupList
Generated by the MealGroupListFactory, its a list of available meal groups, i.e. breakfast, lunch, dinner....

MealGroupListFactory
Servers for the RMS to generate available Meal Group Categories

Recipe: An element of the Recipe management Subsystem. The user can make customized recipe or look up the recipe database.

RecipeFactory

Servers for the RMS to generate available recipes

RecipeList

Generated by the RecipeListFactory, its a list of available recipes

RecipeListFactory

Servers for the RMS to generate recipes

Recipe Management Subsystem (RMS)

A subsystem of the Smart Kitchen Central System used to manage recipes

RMSDispatcher

Serves as a controller for RMS, delegates tasks and responsibilities within and outside the RMS

SKCSInformationExpert

Serves as an informational point for the RMS, contains recipes, instructions for recipes, quantity needed, etc.

SKCS User: The customer who installed the SKCS at home

Smart Kitchen Appliance:

An element of Appliance Management Sub-System

Smart Kitchen Central System (SKCS):

The main system, which includes Grocery Management Sub-System, Recipe Management Sub-System and Appliance Management sub-System

Smart Kitchen Grocery Database: see Barcode DB

Smart Kitchen Recipe Database:

Contains all personal recipes entered by the SKCS user as well as the ones downloaded from the web

SKCSView

The UI layer used to present information to the user, receive update from SKCSViewController

SKCSViewController

Part of the MVC design pattern, updates the SKCSView

Wireless Interface:

Used by the Smart Kitchen Central System, but mainly focusing on control the appliances so that the user can control