Team: Adam Wiemerslage,
Alex Killian,
Kenneth Hunter Wapman

Title: Multifaceted Dictionary

Project Summary: Our project is a dictionary application that manages dictionary entries. An entry is a collection of word senses. A word sense can have a definition, one or more word forms, and a part of speech identifier. This is much like a typical dictionary. The application will allow users to insert new entries, update existing entries, remove entries, and handle collisions when they attempt to insert or update an entry that matches an existing entry. This dictionary will go beyond the uses of a typical dictionary by: providing a number of word insertion methods such as adding entries in batch via a CSV, XML, or TSV file; allowing for quick lookup of entries via their constituent parts; and incorporating NLP inspired analyses on words and definitions, such as stemming via a Porter Stemmer. Using a stemming algorithm will allow for more robust collision detection, which will lead to fewer errors and a cleaner dictionary.

Project Requirements

Business Requirements:
Our application has no business requirements.

User Requirements:

User Requirements		
ID	**Requirement**	**Priority**
UR-01	User can add an Entry.	Critical
UR-02	User can add a Word Sense to an Entry.	Critical
UR-03	User can add a Definition to a Word Sense.	Critical
UR-04	User can add a Word Form to a Word Sense.	Medium
UR-05	User can add a Part of Speech to a Word Sense.	Low
UR-06	User can add entries in batch (Word/Definition pairs only) from a file.	Medium
UR-07	User can update an Entry word spelling.	Medium
UR-08	User can update a Word Form spelling.	Medium
UR-09	User can update a Definition.	Medium
UR-10	User can update a Part of Speech.	Low
UR-11	User can lookup Entries by word.	Critical
UR-12	User can lookup Entries by Definition.	Critical

UR-13	User can lookup Entries by Part of Speech.	Low
UR-14	User can remove an Entry.	High
UR-15	User can remove a Word Sense.	High
UR-16	User can remove a Part of Speech.	Low
UR-17	User can remove a Definition.	High
UR-18	User can remove a Word Form.	Medium
UR-19	User can resolve a Collision.	High

Non-Functional Requirements:

Non-Functional Requirements			
ID	**Requirement**	**Priority**	**Category**
NF-01	The commands/GUI should not require significant explanation for the user to understand how to use them.	Critical	Usability
NF-02	The commands and modules should be well documented so developers understand the system quickly and completely.	High	Maintainability
NF-03	CRUD operations on the dictionary should always leave the system in a consistent state, even after failure.	Critical	Reliability
NF-04	CRUD operations on the dictionary should have reasonable speed, given the size of the dictionary.	High	Performance

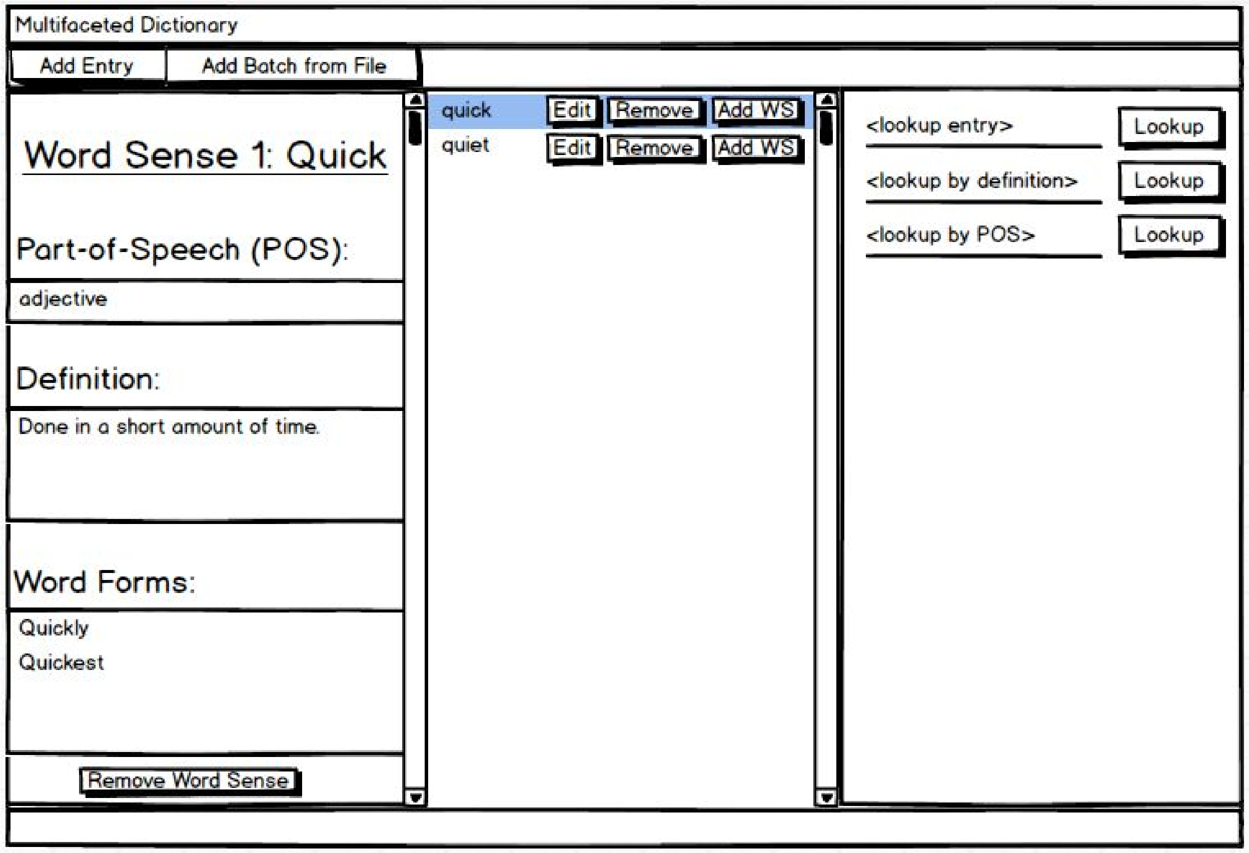
Data Storage

We will use MySQL with Hibernate to persist the user's dictionary.

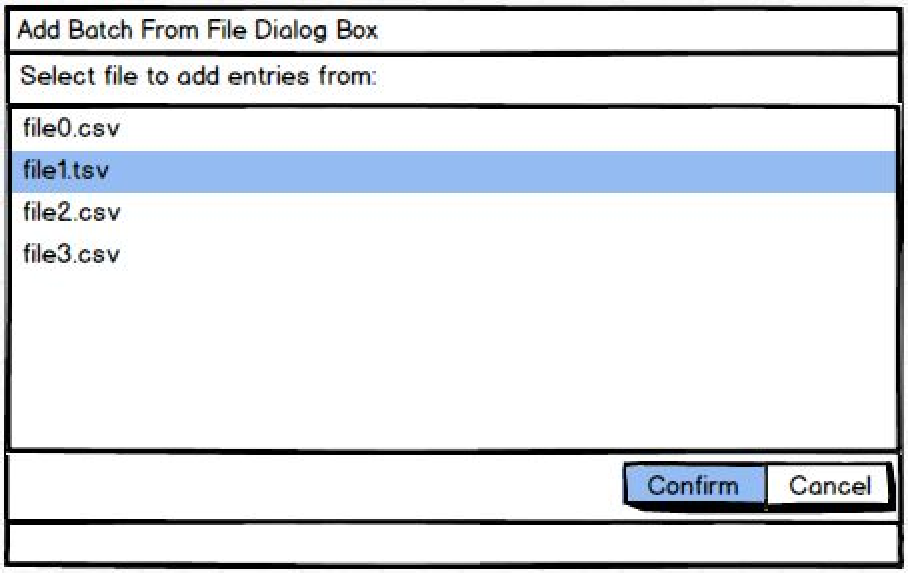
UI Mockups

We feel our primary UI will be best served by a command line interface. However, we also created these mockups to show how a GUI for our dictionary might look and behave.

This GUI is organized by a main window that is broken into three columns. The left column shows the details of a dictionary entry. The fields would become editable upon clicking the entry's corresponding "Edit" button in the middle column. This would allow this data to be updated, added to, and removed. The middle column shows all the dictionary entries that match a lookup operation done by the user. Each listed entry would have three buttons "Edit" (as discussed above), “Remove” (removes the entry), and “Add WS” (adds a new word sense to the entry). Finally, the right column provides the controls for doing the lookups. The top “<lookup entry>” text input field would allow the user to lookup entries by word form. The middle “<lookup by definition>” input field would allow the user to lookup entries by definition. And the last “<lookup by POS>” input field would allow the user to lookup entries by their part-of-speech. The “Lookup” buttons to the right would cause the middle column to update to reflect all entries that matched the lookup.



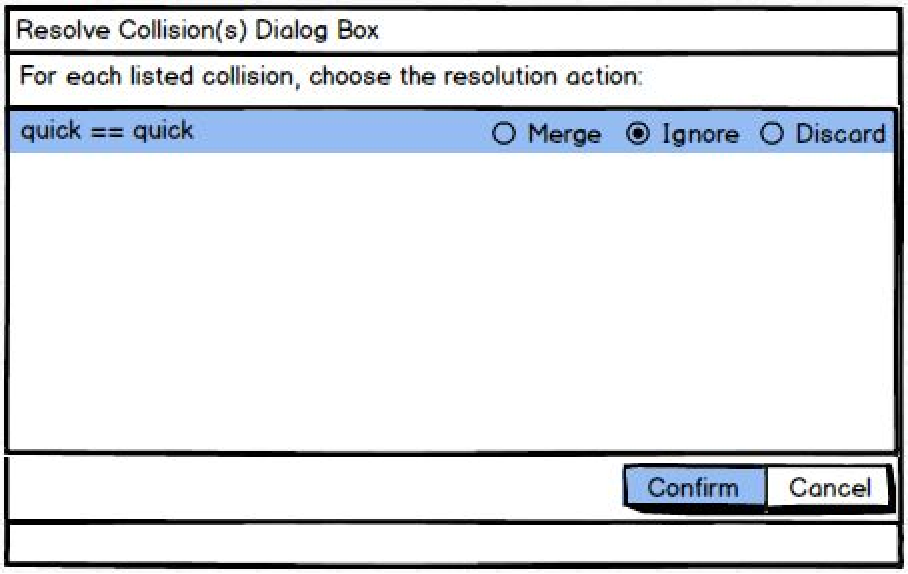
To facilitate the rest of our use cases, we would also have dialog boxes in our GUI. We would have an “Add Batch from File” dialog box, which would display when the user clicks the “Add Batch From File” button; a “Resolve Collision(s)” dialog box, which would display when the system detects collisions that the user needs to resolve; an “Add Entry” dialog box, which would display when the user clicks the “Add Entry” button; and a “Add Word Sense” dialog box, which would show when the user clicks one of the “Add WS” buttons. Finally, there would be a “Confirmation” dialog box that would display to allow the user to confirm remove operations.



Add Batch From File Dialog Box

Select file to add entries from:

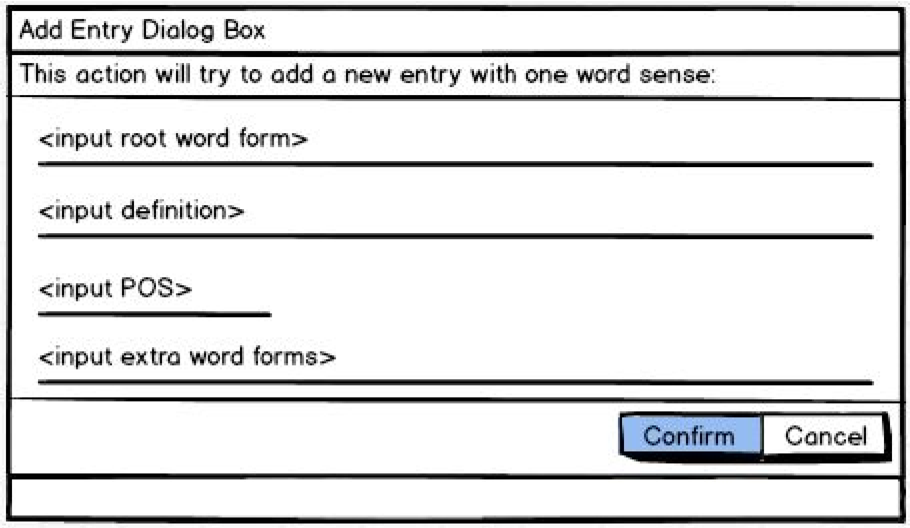
file0.csv
file1.tsv
file2.csv
file3.csv



Resolve Collision(s) Dialog Box

For each listed collision, choose the resolution action:

quick == quick   Merge [✓ Ignore] Discard



Add Entry Dialog Box

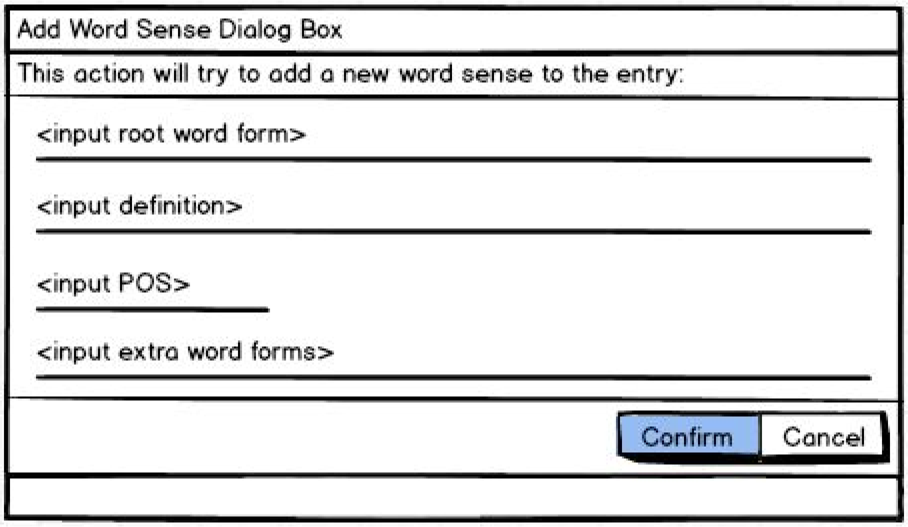
This action will try to add a new entry with one word sense:

<input root word form>

<input definition>

<input POS>

<input extra word forms>


Add Word Sense Dialog Box

This action will try to add a new word sense to the entry:

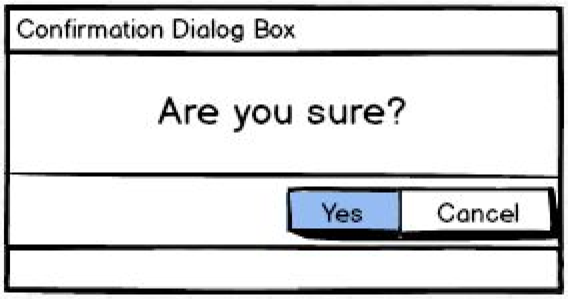
<input root word form>

<input definition>

<input POS>

<input extra word forms>

Confirm | Cancel

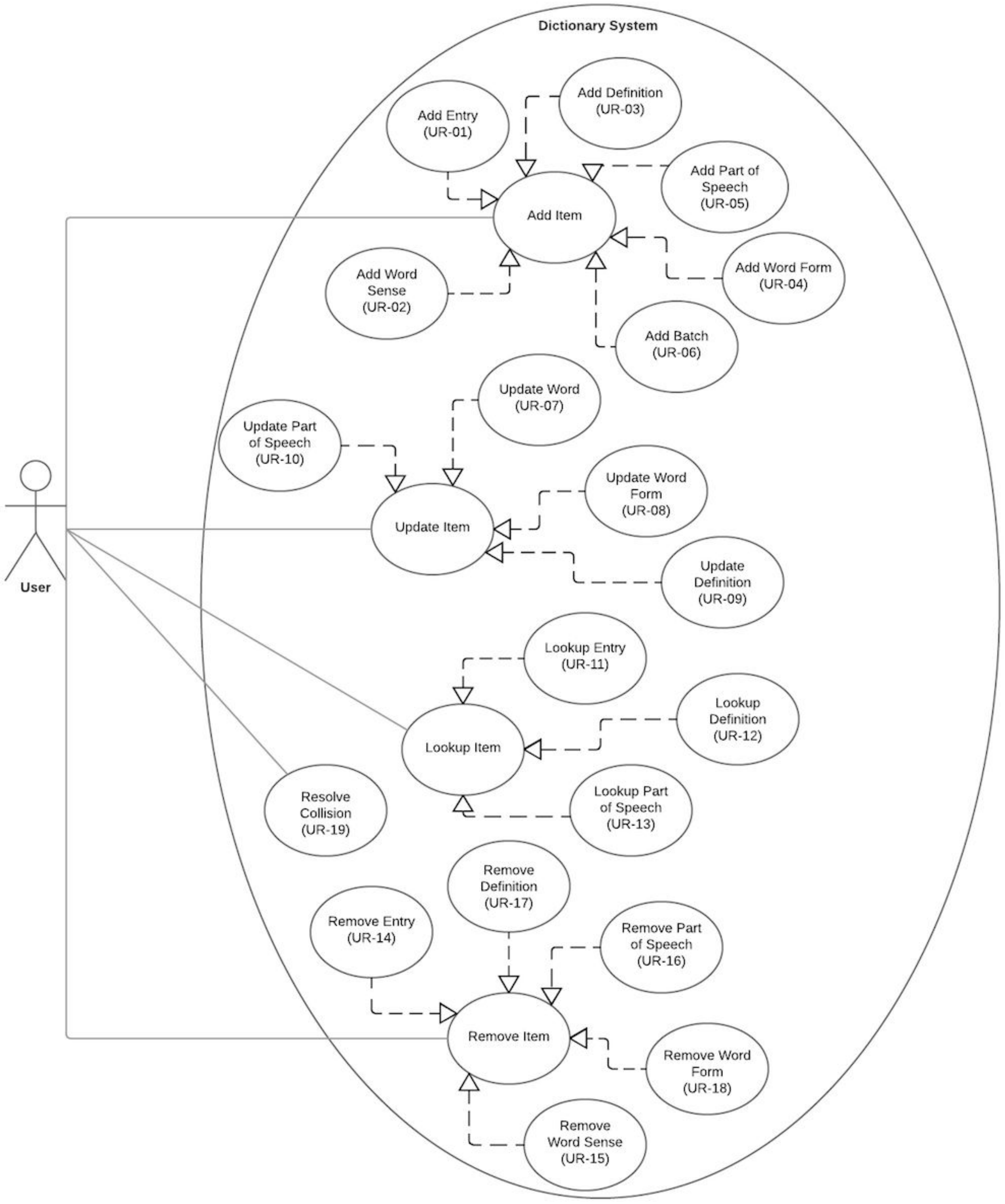

Confirmation Dialog Box

Are you sure?

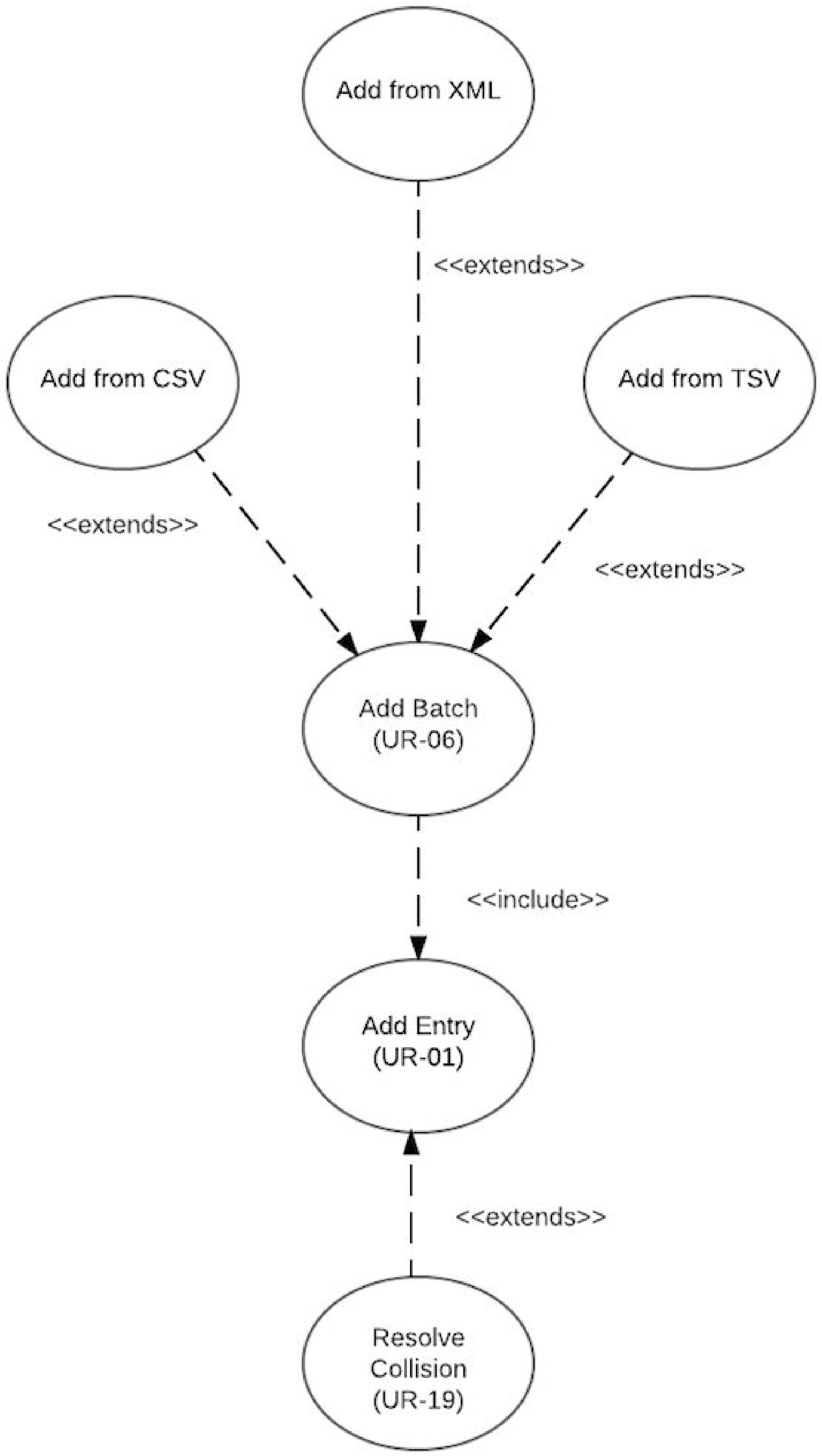
Yes | Cancel

Use Cases

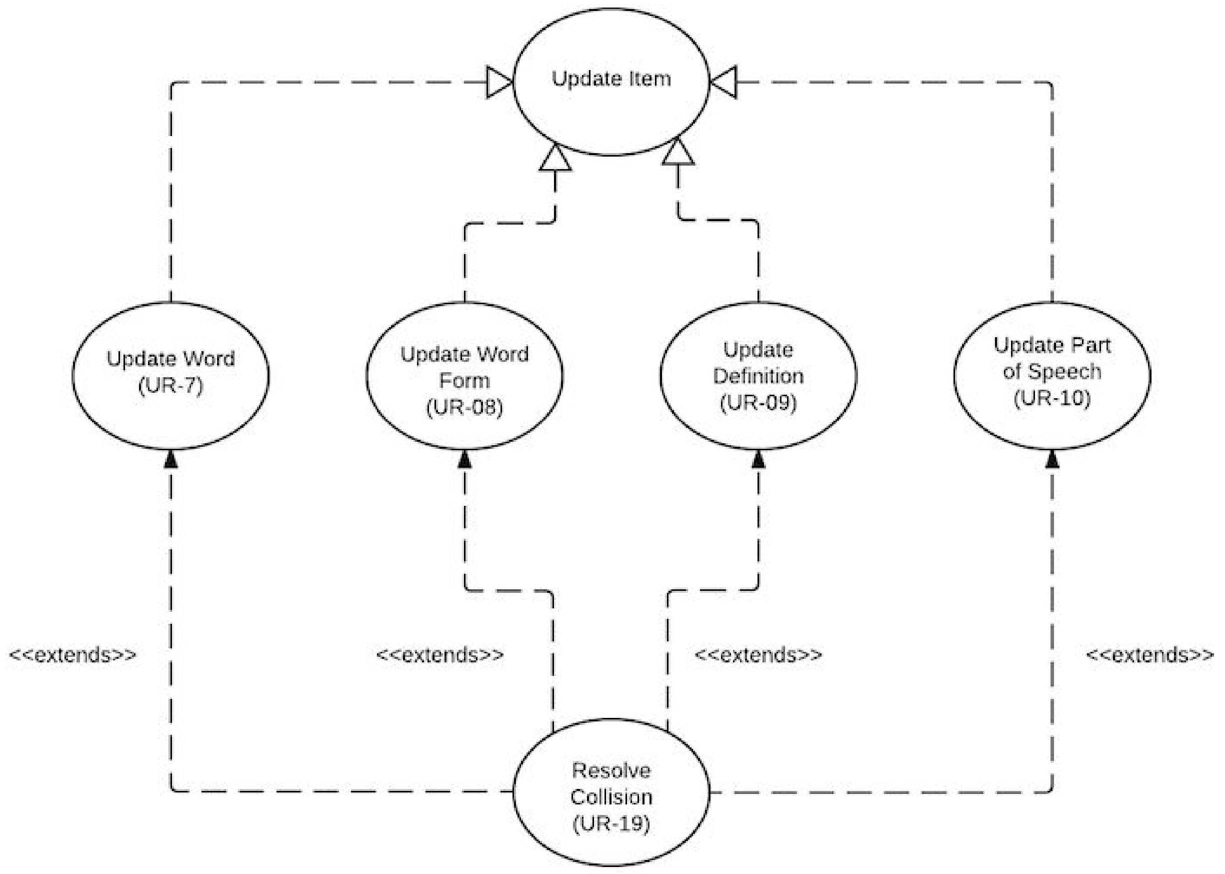
Use Case Overview:



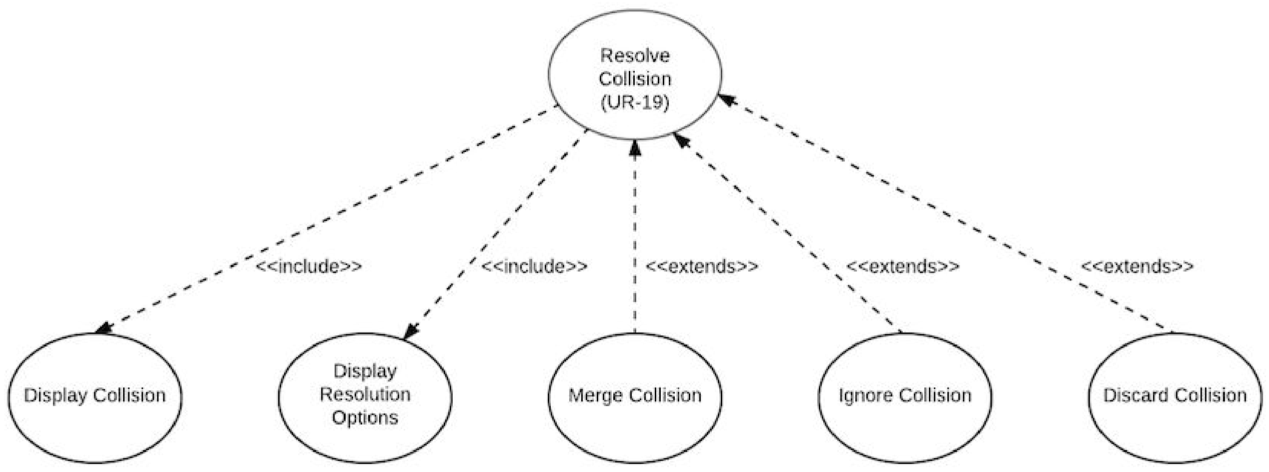
Add Batch Use Case Sub-diagram:



Update Item Use Case Sub-diagram:



Resolve Collision Use Case Sub-diagram:



Add Batch Use Case Document:

Use Case ID:	UC-01
Use Case Name:	Add Batch
Description:	User can add multiple dictionary entries from a specified file. The file may be of type CSV, TSV, or XML.

Actors:	User
Pre-conditions:	No pre-conditions, meaning this action can be done on an empty dictionary.
Post-conditions:	Entries are added correctly as specified by the file and the user. The dictionary should be in a consistent state.
Frequency of Use:	As often as the user needs, which could be daily.
Flow of Events:	

	Actor Action	**System Response**
1	Inputs add command.	
2	Specifies a file.	Checks file type and alerts user if there is an issue. User can retry if there is an issue.
3		Loads file.
4		Parses entries.
5		For each entry, checks if the entry collides; if so, prompts user (see Collision Resolution Use Case Document); if not, adds the entry.
6	Responds to collisions.	Fixes collisions.
7		Displays results of batch add.

Variations:	User can add from a CSV, a TSV, or a XML file. Could involve collision resolutions.
Exceptions:	If file read fails should fail gracefully. If collision occurs with existing entry, user should be given chance to choose how it is resolved.
Developer Notes:	Remember that this use case could involve collisions.

Resolve Collision Use Case Document:

Use Case ID:	UC-02
Use Case Name:	Resolve Collision
Description:	User can resolve a collision that occurs from a user action. A collision occurs when a modification to the database entails a change that would result in a duplicate entry or sense in our database.

Actors:	User
Pre-conditions:	There are existing entries in the database. A user action has been executed. The user action results in a duplicate entry or sense in the database.
Post-conditions:	The state of the database has been resolved as the user desires.
Frequency of Use:	As often as the user needs, which could be daily.
Exceptions:	If there is a character encoding failure, handle gracefully.
Developer Notes:	None

Variation 1: Entry Collision, User selects DISCARD		
1.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.
1.2	User	User selects DISCARD
1.3	System	System discards the user-input.
1.4	System	Terminates use case.

Variation 2: Entry Collision, User selects IGNORE		
2.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.
2.2	User	User selects IGNORE.
2.3	System	System adds the user-input Entry to the dictionary.
2.4	System	Terminates use case.

Variation 3: Entry Collision, User selects MERGE, System does not identify a Word Sense Collision		
3.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.
3.2	User	User selects MERGE.
3.3	System	System does not identify a Word Sense Collision.
3.4	System	System adds a new Word Sense to the entry that the user-input Word collided with.
3.5	System	Terminates use case.

Variation 4: Entry Collision, User selects MERGE, Word Sense Collision results, user selects DISCARD		
4.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.
4.2	User	User selects MERGE.
4.3	System	System identifies Word-Sense Collision.
4.4	System	System prompts user to MERGE, IGNORE, or DISCARD the Word Sense Collision.
4.5	User	User selects DISCARD.
4.6	System	System resumes at point 1.3.

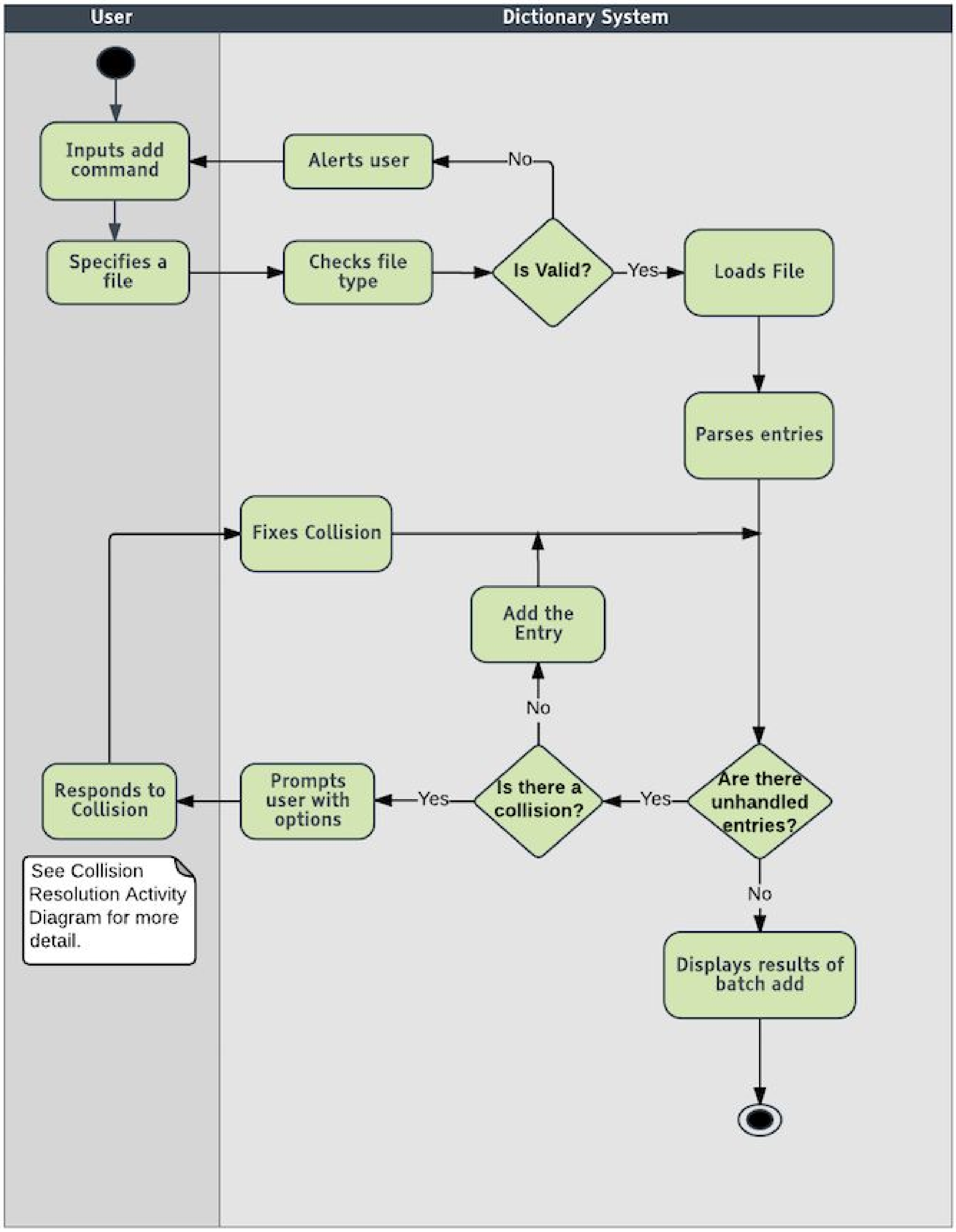
Variation 5: Entry Collision, User selects MERGE, Word Sense Collision results, user selects IGNORE		
5.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.

5.2	User	User selects MERGE.
5.3	System	System identifies a Word Sense Collision.
5.4	System	System prompts user to MERGE, IGNORE, or DISCARD the Word Sense Collision.
5.5	User	User selects IGNORE.
5.6	System	System adds a new Word Sense to the Entry that the user-input Word collided with.
5.7	System	Terminates use case.

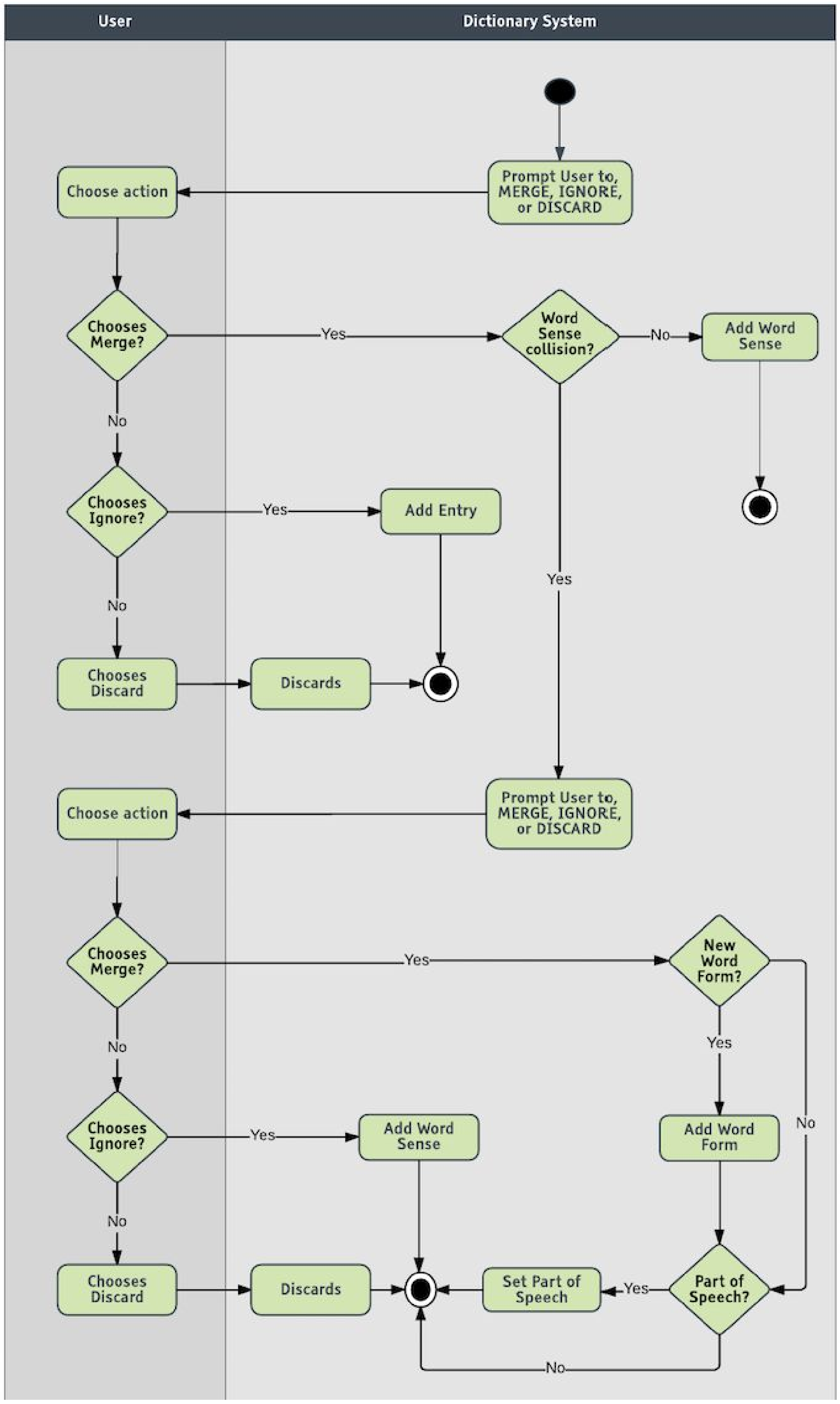
Variation 6: Entry Collision, User selects MERGE, Word Sense Collision results, User selects MERGE		
6.1	System	System prompts user to MERGE, IGNORE, or DISCARD the Entry Collision.
6.2	User	User selects MERGE.
6.3	System	System identifies Word Sense Collision.
6.4	System	System prompts user to MERGE, IGNORE, or DISCARD the Word Sense Collision.
6.5	User	User selects MERGE.
6.6	System	System adds a Word Form to the Word Sense that the user-input Entry collided with if the user-input Entry has a Word Form and that Word Form is not in the Entry the user-input Entry collided with.
6.7	System	System sets the Part of Speech of the Word Sense that the user-input Entry collided with to the Part of Speech in the user-input Entry if the user-input Entry has a Part of Speech.
6.8	System	Terminates use case.

Activity Diagrams

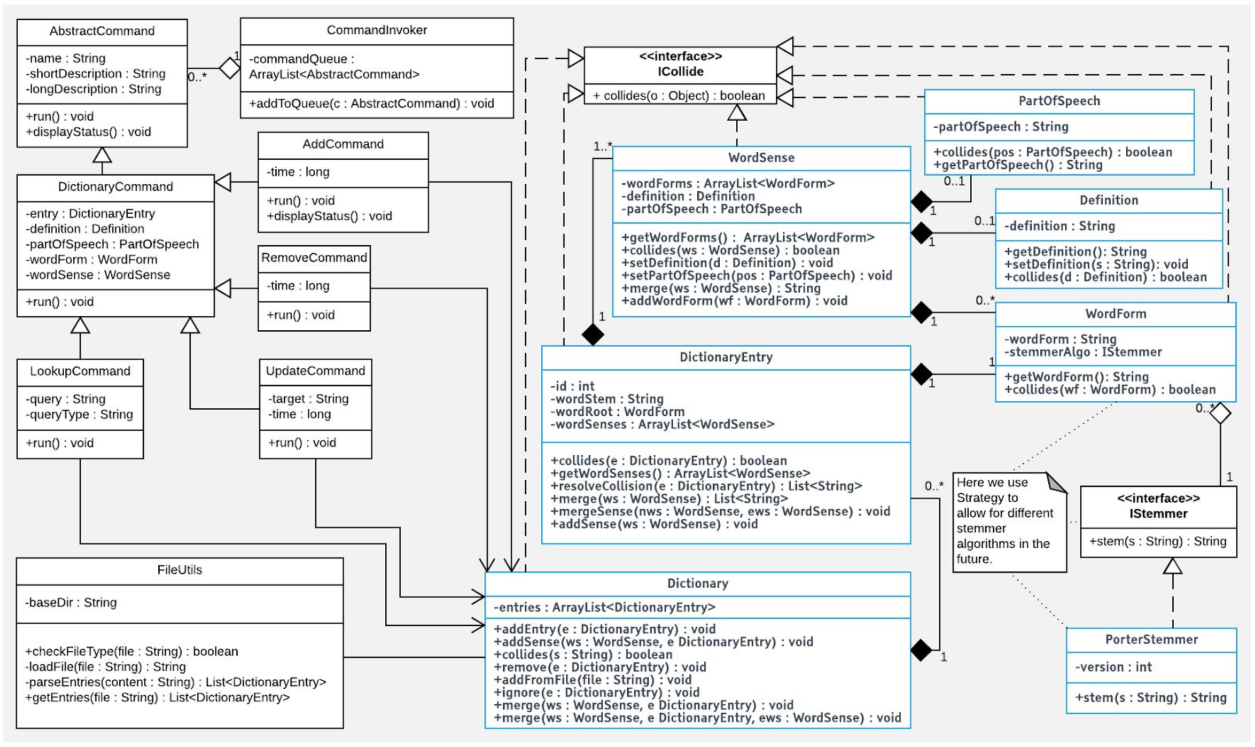
Add Batch (UR-06 / UC-01) Activity Diagram:



Resolve Collision (UR-19 / UC-02) Activity Diagram:

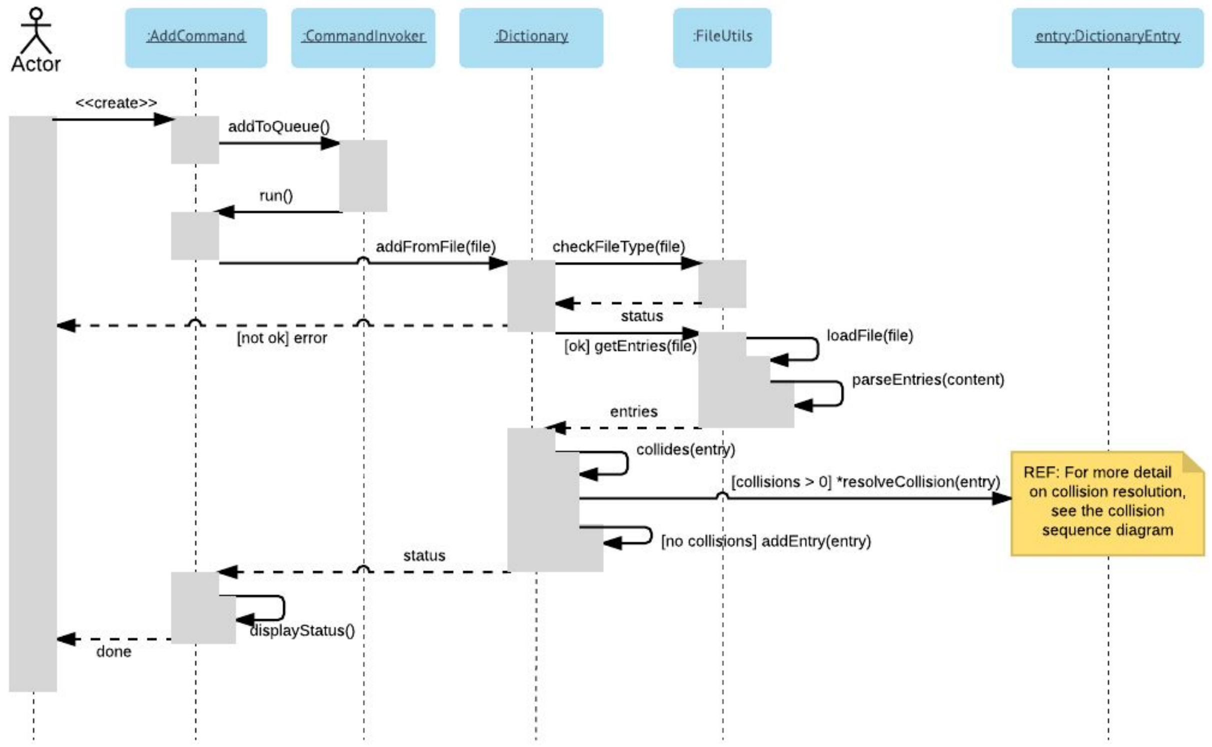


Class Diagram

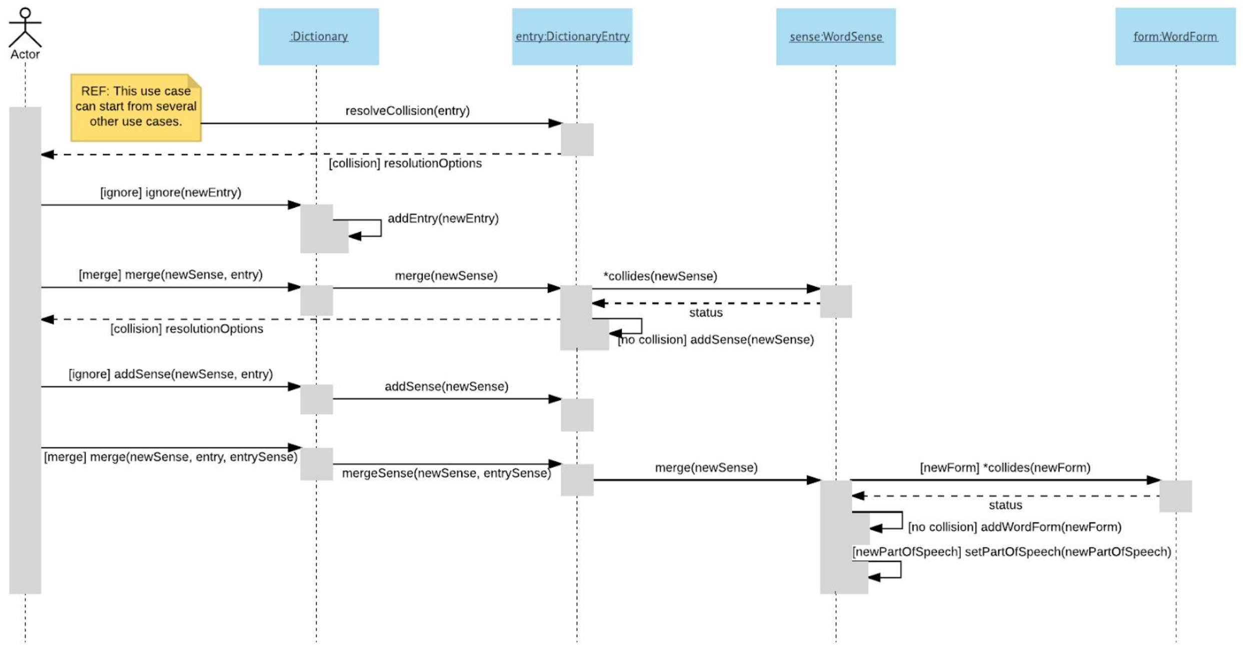

AbstractCommand
-name : String
-shortDescription : String
-longDescription : String
+run() : void
+displayStatus() : void
0..*
1
CommandInvoker
-commandQueue : ArrayList<AbstractCommand>
+addToQueue(c : AbstractCommand) : void
AddCommand
-time : long
+run() : void
+displayStatus() : void
DictionaryCommand
-entry : DictionaryEntry
-definition : Definition
-partOfSpeech : PartOfSpeech
-wordForm : WordForm
-wordSense : WordSense
+run() : void
RemoveCommand
-time : long
+run() : void
LookupCommand
-query : String
-queryType : String
+run() : void
UpdateCommand
-target : String
-time : long
+run() : void
FileUtils
-baseDir : String
+checkFileType(file : String) : boolean
-loadFile(file : String) : String
-parseEntries(content : String) : List<DictionaryEntry>
+getEntries(file : String) : List<DictionaryEntry>
<<interface>>
ICollide
+ collides(o : Object) : boolean
PartOfSpeech
-partOfSpeech : String
+collides(pos : PartOfSpeech) : boolean
+getPartOfSpeech() : String
1..*
WordSense
-wordForms : ArrayList<WordForm>
-definition : Definition
-partOfSpeech : PartOfSpeech
+getWordForms() : ArrayList<WordForm>
+collides(ws : WordSense) : boolean
+setDefinition(d : Definition) : void
+setPartOfSpeech(pos : PartOfSpeech) : void
+merge(ws : WordSense) : String
+addWordForm(wf : WordForm) : void
0..1
1
Definition
-definition : String
+getDefinition(): String
+setDefinition(s : String): void
+collides(d : Definition) : boolean
0..1
1
0..*
1
WordForm
-wordForm : String
-stemmerAlgo : IStemmer
+getWordForm(): String
+collides(wf : WordForm) : boolean
1
1
DictionaryEntry
-id : int
-wordStem : String
-wordRoot : WordForm
-wordSenses : ArrayList<WordSense>
+collides(e : DictionaryEntry) : boolean
+getWordSenses() : ArrayList<WordSense>
+resolveCollision(e : DictionaryEntry) : List<String>
+merge(ws : WordSense) : List<String>
+mergeSense(nws : WordSense, ews : WordSense) : void
+addSense(ws : WordSense) : void
1
0..*
1
Here we use Strategy to allow for different stemmer algorithms in the future.
<<interface>>
IStemmer
+stem(s : String) : String
1
Dictionary
-entries : ArrayList<DictionaryEntry>
+addEntry(e : DictionaryEntry) : void
+addSense(ws : WordSense, e DictionaryEntry) : void
+collides(s : String) : boolean
+remove(e : DictionaryEntry) : void
+addFromFile(file : String) : void
+ignore(e : DictionaryEntry) : void
+merge(ws : WordSense, e DictionaryEntry) : void
+merge(ws : WordSense, e DictionaryEntry, ews : WordSense) : void
PorterStemmer
-version : int
+stem(s : String) : String

User Interactions

Add Batch (UR-06 / UC-01) Sequence Diagram:



Resolve Collision (UR-19 / UC-02) Activity Diagram:


Actor
:Dictionary
entry:DictionaryEntry
sense:WordSense
form:WordForm
REF: This use case can start from several other use cases.
resolveCollision(entry)
[collision] resolutionOptions
[ignore] ignore(newEntry)
addEntry(newEntry)
[merge] merge(newSense, entry)
merge(newSense)
*collides(newSense)
status
[collision] resolutionOptions
[no collision] addSense(newSense)
[ignore] addSense(newSense, entry)
addSense(newSense)
[merge] merge(newSense, entry, entrySense)
mergeSense(newSense, entrySense)
merge(newSense)
[newForm] *collides(newForm)
status
[no collision] addWordForm(newForm)
[newPartOfSpeech] setPartOfSpeech(newPartOfSpeech)