UML - MAGIC

UML Design Wizard for Clients and Software Engineers

Dillon Beck
Drew Hoover
Slade Meriwether
J. Wayne Nollen
F. Davis Quarles
Kullen Williams

UML Magic Analysis and Requirements

Table of Contents

1- Domain Analysis

- Concept Statement
- Conceptual Domain Model
- Domain State Model

Concept Statement

Design the software to support an automated UML production system to aid both designers and clients in creating and managing UML analysis and design documentation. This system is meant to facilitate the analysis and design phase of software development and should be used collaboratively between software developers and clients.

The system will first be provided a concept statement by the user that it will analyze to perform domain analysis and aid in development of a domain class model. During this analysis, the system will use a natural language parser to attempt to identify key terms that could be considered classes, and use these to build an application class model. The system will present the application class model to the user, and the user will be prompted to remove any classes that are not relevant or add any missing classes. The user is now prompted to write a small amount about each class to prepare a data dictionary.

Next, the system will use a natural language parser to identify stative verbs or verb phrases that can correspond to associations between the classes. From these possible associations, the user will be prompted to remove associations that are unnecessary or incorrect (the system will attempt to identify associations it thinks are less useful), and the user will be prompted to add associations that are missing (and identify the classes related by that association). Association attributes, such as name, end-names, and multiplicity, are also automatically generated by the system, and the user is allowed to add, modify, or delete these.

From this point, the system will attempt to refine the classes by inheritance by either bottom-up generalization or top-down specialization. Possible refinements in the class diagram will be shown to the user for consideration. The user can then make what changes they find necessary.

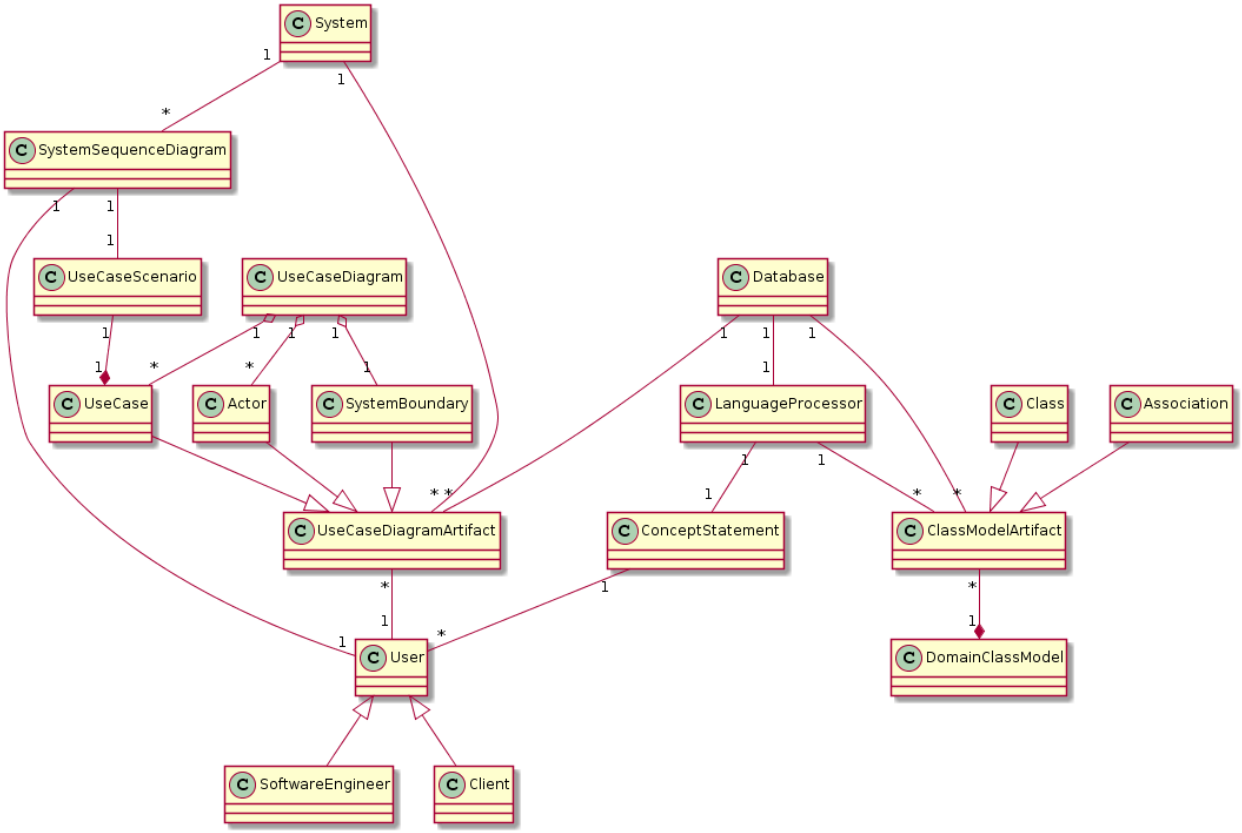
The system will now use the domain analysis information and domain class model to identify actors that interact with the system. System identified actors will be shown to the user to verify, add, change, or delete. By identifying these actors, the system boundary will be identified - any actors chosen here will be considered outside the system and everything else is considered part of the system.

The system will attempt to identify use cases from the information it has and will allow the user to add, change, or remove use cases. The system will prompt the user to prepare scenarios for the use cases identified. To facilitate scenario creation, the system will provide information including initial and final events, exception scenarios, and external events.

From these use cases and scenarios, the system will help the user generate sequence diagrams. For similar scenarios, the system will gather information from previously created sequence diagrams to aid in auto-generating sequence diagrams. Sequence diagrams can also be expanded to include boundary, entity, and controller objects.

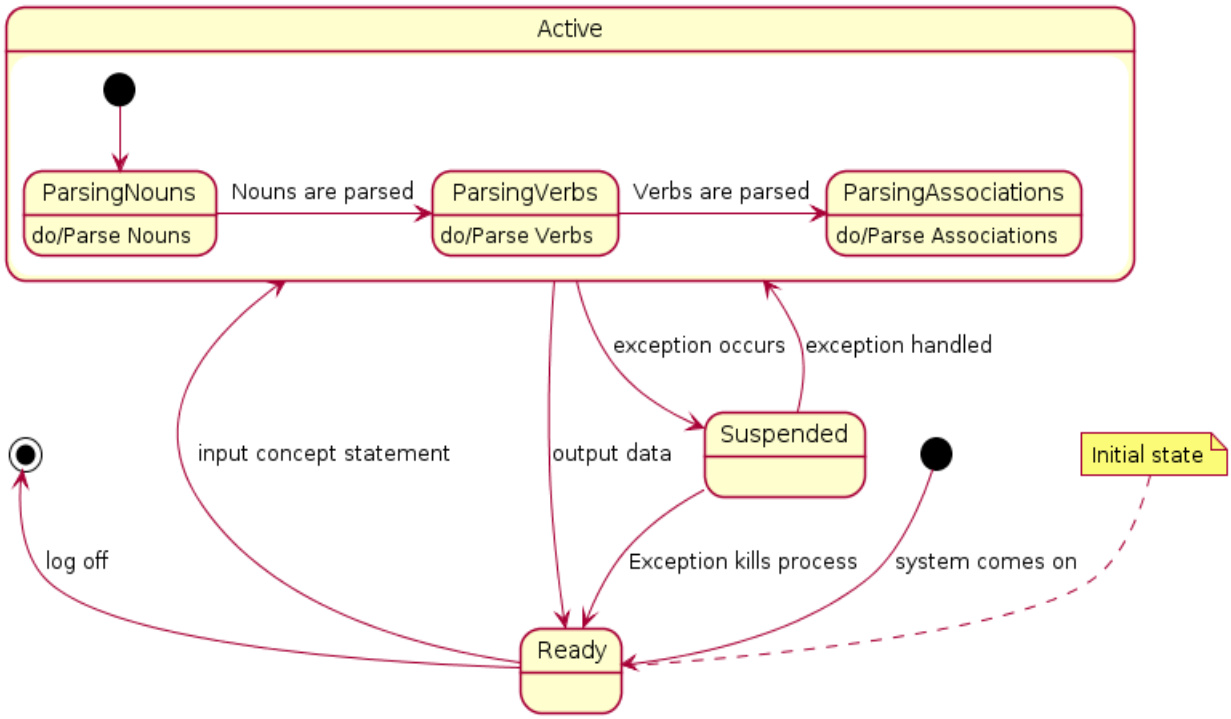
The final product from this system is a domain class model, use cases, and system sequence diagrams.

Domain Class Model

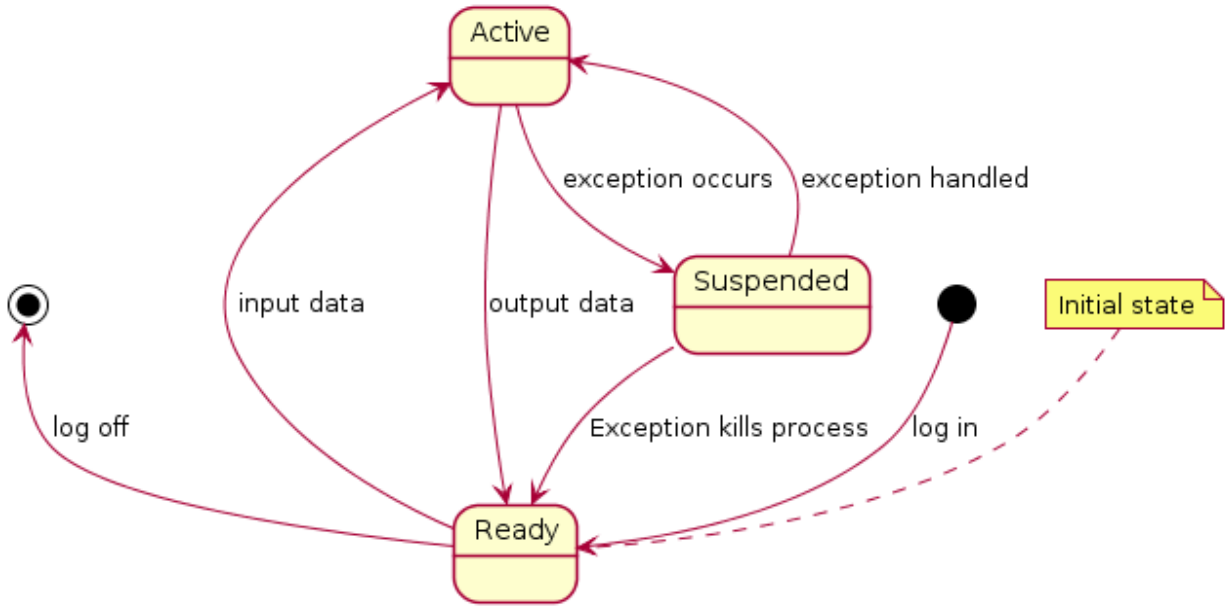


Domain State Model

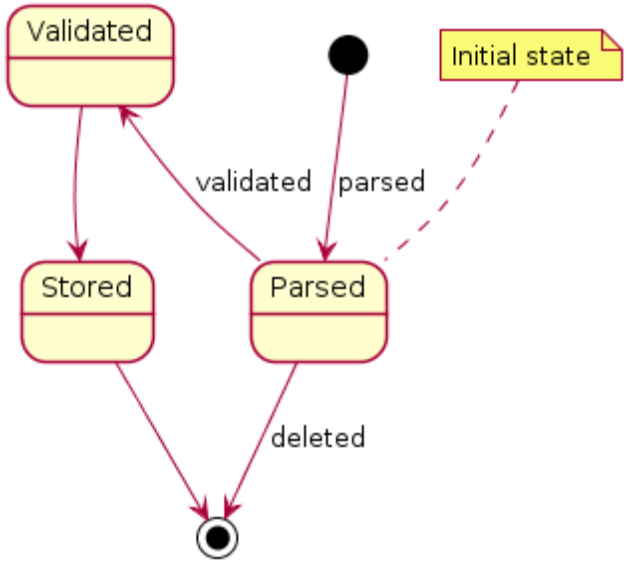
State Model for Natural Language Parser



State Model for System



State Model for Associations



2- Application Analysis

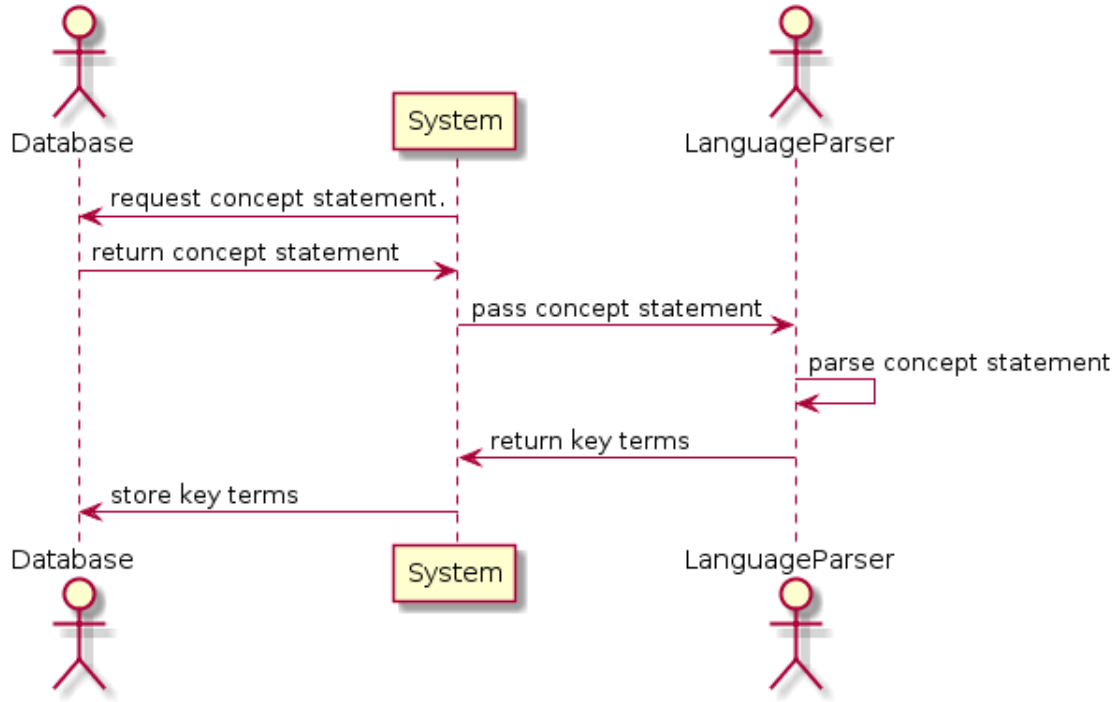
- Use cases
- Application Interaction Model
    - Essential Use Cases, Scenarios, and High-level SSD
    - Concrete Use Cases and Detailed SSD
- Application Class Model
- Application State Model

2.1 - Parse Concept Statement

Essential Use Case:
Summary: Reads concept statement from database sentence by sentence, evaluates each sentence, and extracts key terms and stores them in the database.
Actors: Language Parser and Database
Preconditions: A concept statement is already stored in the database.
Description: The system passes the concept statement from the database to the language parser. The language parser scans each sentence and extracts key terms. Extracted key terms are stored in the database.
Exceptions:
No key terms identified: No key terms are found in the concept statement. The system shows a message explaining this.
Postconditions: All key terms identified by the language parser are stored in the database.

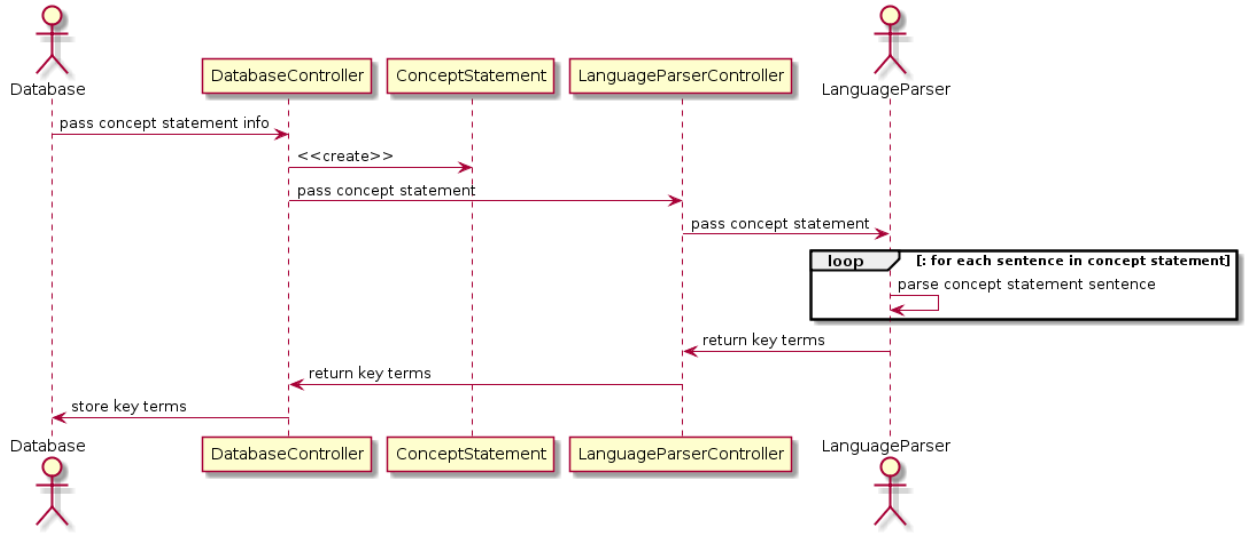
Scenario:
System requests concept statement from Database.
Database returns concept statement to System.
System passes concept statement to Language Parser.
Language Parser parses concept statement.
Language Parser returns key terms to System.
System passes key terms to Database.
Database stores key terms.

High Level SSD:



Concrete Use Case:
Summary: Reads concept statement from database sentence by sentence, evaluates each sentence, and extracts key terms and stores them in the database.
Actors: Language Parser and Database
Preconditions: A concept statement is already stored in the database.
Description: The system passes the concept statement from the database to the language parser. The language parser scans each sentence and extracts key terms. Extracted key terms are stored in the database.
Exceptions:
No key terms identified: No key terms are found in the concept statement. The system shows a message explaining this.
Postconditions: All key terms identified by the language parser are stored in the database.

Detailed SSD:



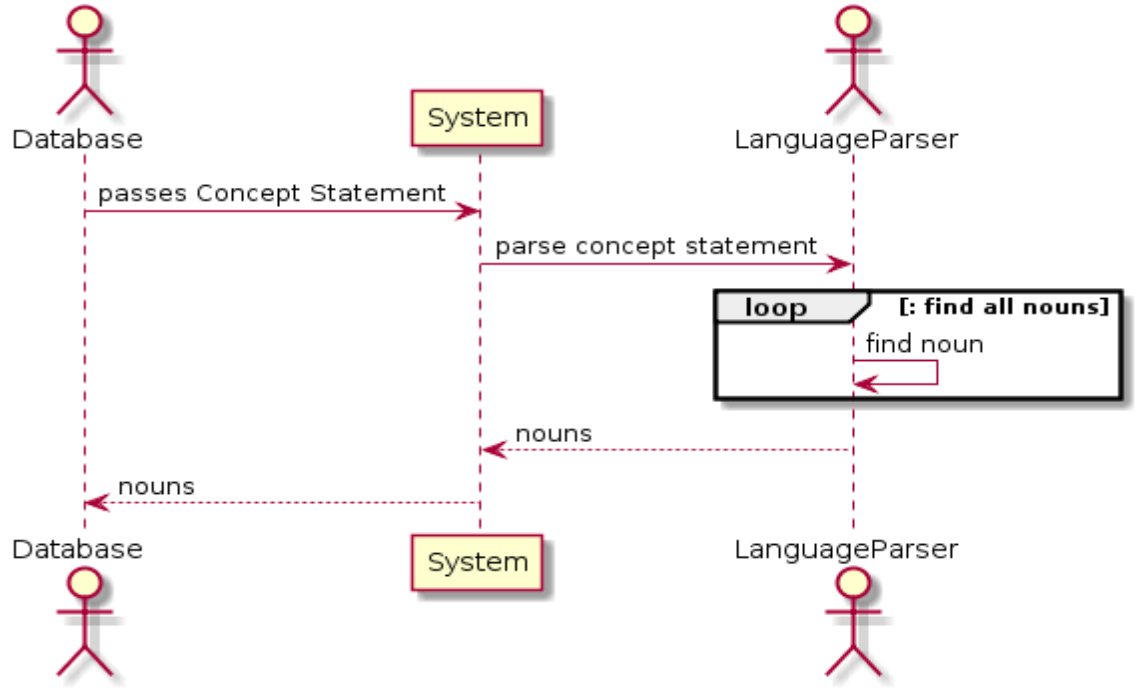
2.2 - Parse Nouns

Essential Use Case:

Summary: The Language parser identifies the nouns in the concept statement and saves them into the database.
Actors: Language Parser and Database
Preconditions: Concept statement is stored in the database.
Description: Language Parser scans the entire concept statement and identifies all nouns. Duplicate nouns are ignored. The identified nouns are saved to the database.
Exceptions:
No nouns identified: No nouns are identified by the Language Parser. A suitable error is shown to the user.
Postconditions: All unique nouns from the concept statement are stored in the database.

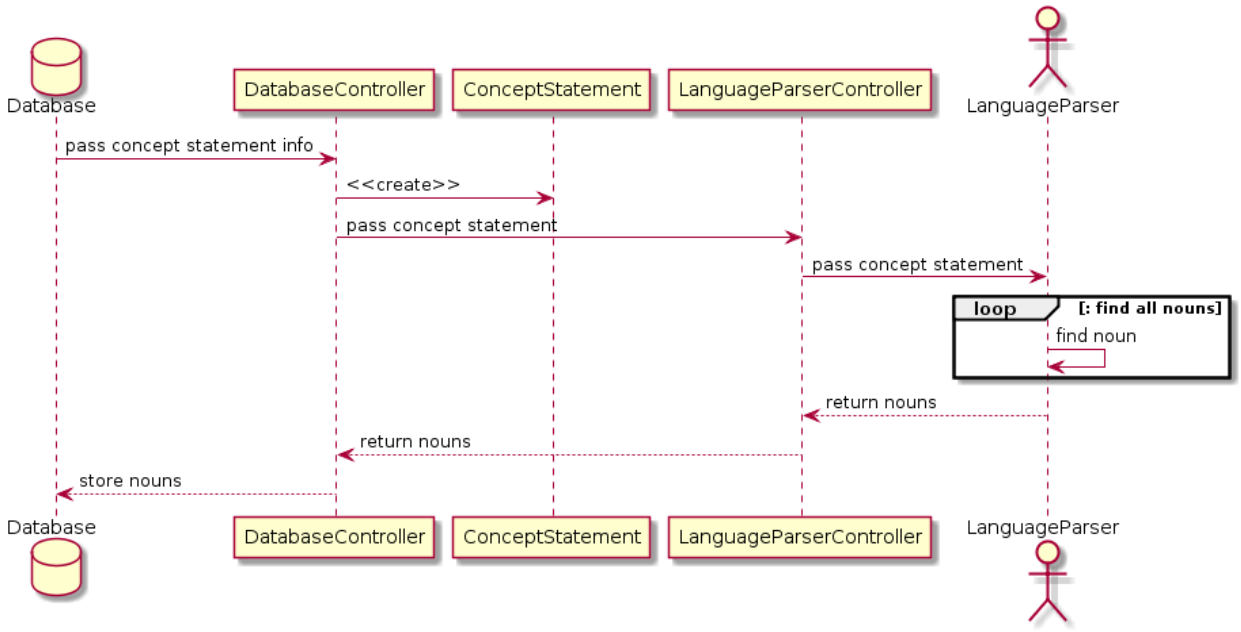
Scenario:
Database sends concept statement to System.
System passes concept statement to Language Parser.
Language Parser identifies all nouns in the concept statement.
Language Parser removes duplicate nouns.
Language Parser returns all nouns to the System.
System returns all nouns to the Database.
Database stores all nouns.

High Level SSD:



Concrete Use Case:
Use Case: Parse Nouns
Summary: The Language parser identifies the nouns in the concept statement and saves them into the database.
Actors: Language Parser and Database
Preconditions: Concept statement is stored in the database.
Description:
Language Parser scans a line in the concept statement.
All nouns from the line are identified.
Repeat previous steps for each line.
Duplicate nouns are removed.
All identified nouns are stored in the database.
Exceptions:
No nouns identified: No nouns are identified by the Language Parser. A suitable error is shown to the user.
Postconditions: All unique nouns from the concept statement are stored in the database.

Detailed SSD



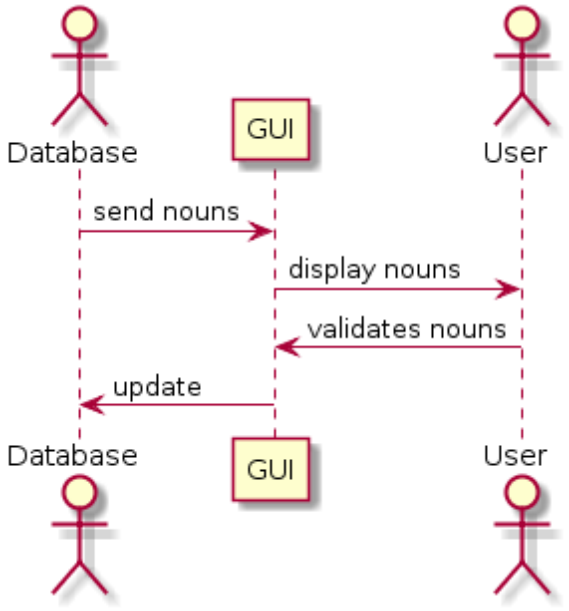
2.3 - Validate Nouns

Essential Use Case: Validate Nouns

- **Summary:** After nouns have been extracted from the concept statement and stored in the database. Our system will need help from the user to remove nouns that do not need to be included.
- **Actors:** User and Database
- **Preconditions:** Nouns have been parsed and stored in Database
- **Description:** The System shows the user a list of discovered nouns. and ask use to remove any nouns that need to be removed
- **Exceptions:**
    - No nouns were not found.
    - Nouns were not stored on database.
- **Postconditions:**
    - Nouns are validated and the Database is updated.

Scenario:
After the nouns have been found by the Language Parser.
Displays nouns that were stored on the Database
Users select valid nouns
Unselected nouns are removed from the database.

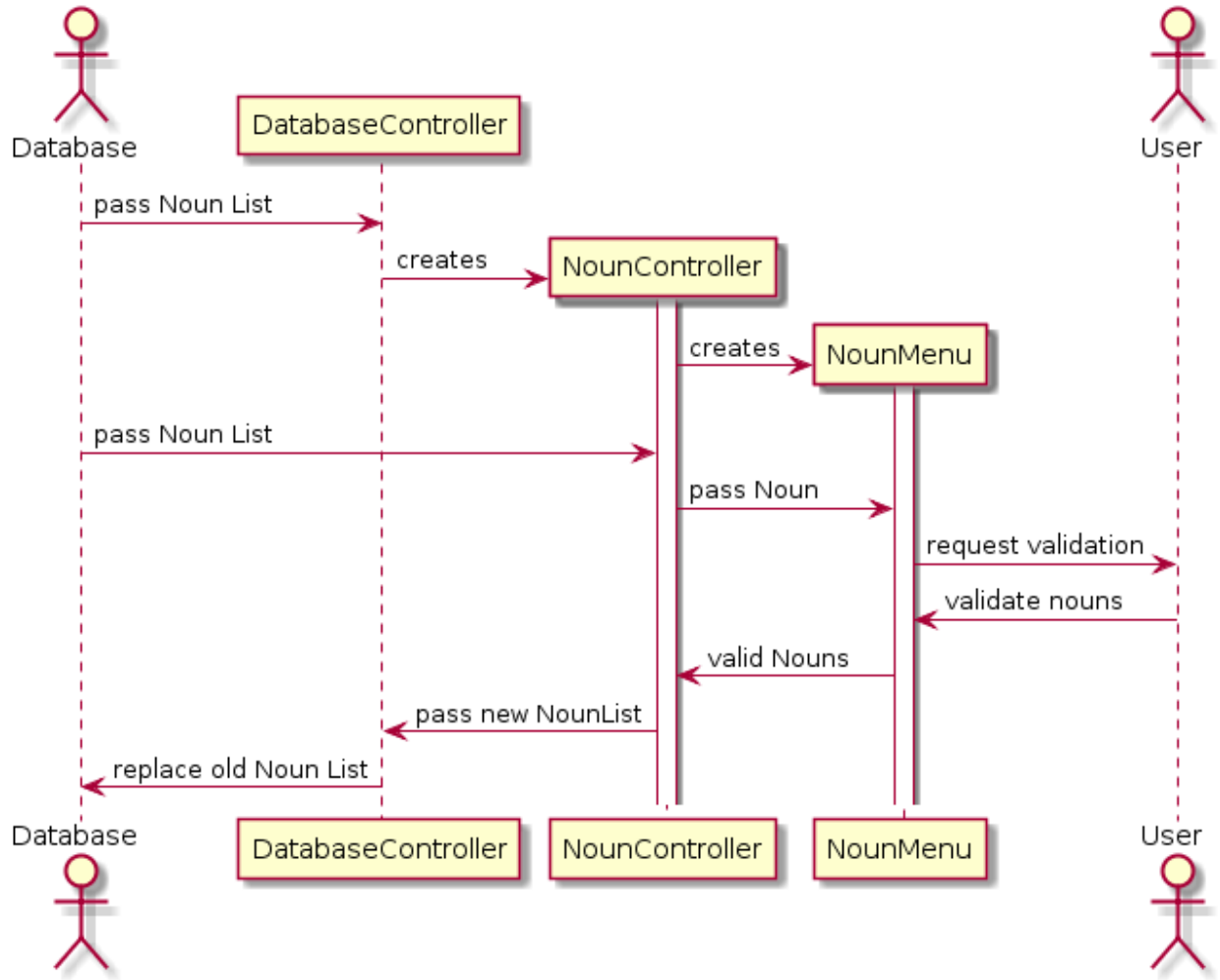
High Level SSD:



Concrete Use Case: Validate Nouns

- **Summary:** After nouns have been extracted from the concept statement and stored in the database. Our system will need help from the user to remove nouns that do not need to be included.
- **Actors:** User and Database
- **Preconditions:** Language Parser has found all nouns
- **Description:** Once the language parser finds all the nouns and removes duplicates. They are stored into the database. In the validation stage. The nouns are show on the screen and the user must select valid nouns to continue.
- **Exceptions:**
    - No nouns we're in the previous step
    - User doesn't select any nouns
- **Postconditions:** User validates nouns and working noun set is store in database

Detailed SSD:



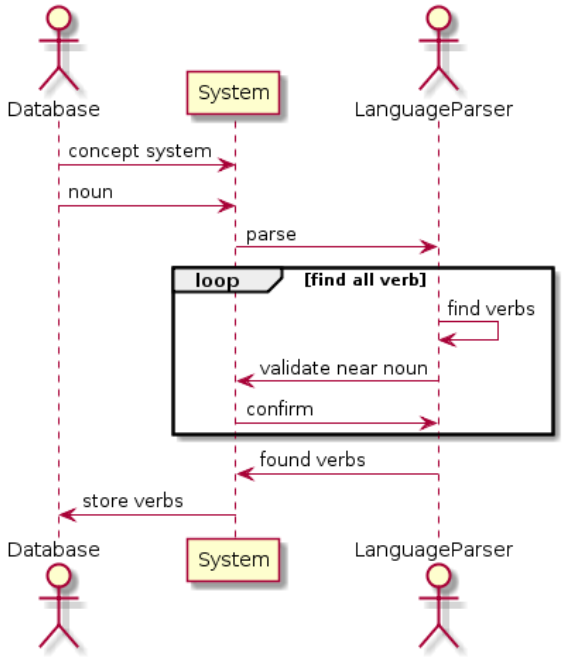
2.4 - Parse Verbs

Essential Use Case:

Summary: The Language Parser identifies the verbs between validated nouns in the concept statement and them into the database.
Actors: Language Parser and Database
Preconditions: Nouns have been validated.
Description: Language Parser scans back through concept statement that was stored on the database and identifies the all verbs that are located along with the nouns that they will associated with. Identified verbs will be saved to the database.
Exceptions:
No verbs identified: No verbs are identified by the Language Parser. A suitable error is shown to the user.
Postconditions: All verbs that have associations with nouns are found.

Scenario:
Database sends concept statement and validated nouns to System
System passes concept statement to Language Parser
Language parser identifies all verbs located in between validated nouns
Language parser returns verb relation table. (noun1, noun2, verb)
System return verb relation table to Database
Database stores relationship

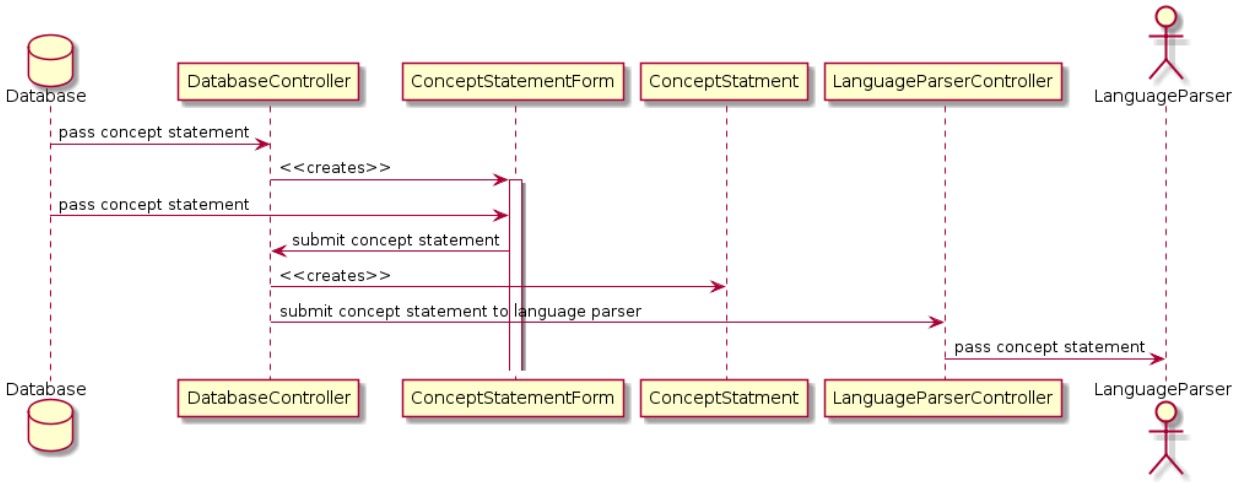
High Level SSD:



Concrete Use Case:
Summary: The Language Parser identifies the verbs between validated nouns in the concept statement and them into the database.

Actors: Language Parser and Database
Preconditions: Nouns have been resolved
Description:
Language parser scans a line from the concept statement.
Locates Noun that has been validated by the user then finds next validated noun.
Next it will identify Verb that is between each noun.
All Identified Verbs are stored along with their nouns.
Exceptions:
No Verbs can be found.
Postconditions:
All verbs are found.

Detailed SSD:



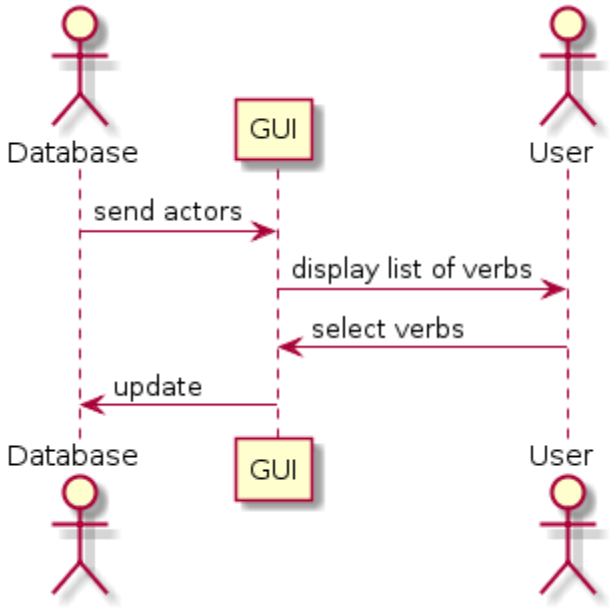
2.5 - Validate Verbs

Essential Use Case:

- **Summary:** Once the verbs have been extracted to the concept statement and stored in the database, the system will then display the verbs to the user and request that remove the verbs that are not needed.
- **Actors:** User and Database
- **Preconditions:** The language parser has parsed and stored the verbs in the database.
- **Description:** The system displays the verbs from the database in a GUI output. The user is then prompted to remove any verbs they find unnecessary.
- **Exceptions:**
    - No verbs were found in the database
- **Postconditions:**
    - Verbs are validated and the Database is updated.

Scenario:
Displays the list of verbs to the user
Prompts the user to select verbs
User selects valid verbs
New list is stored and unneeded verbs are removed

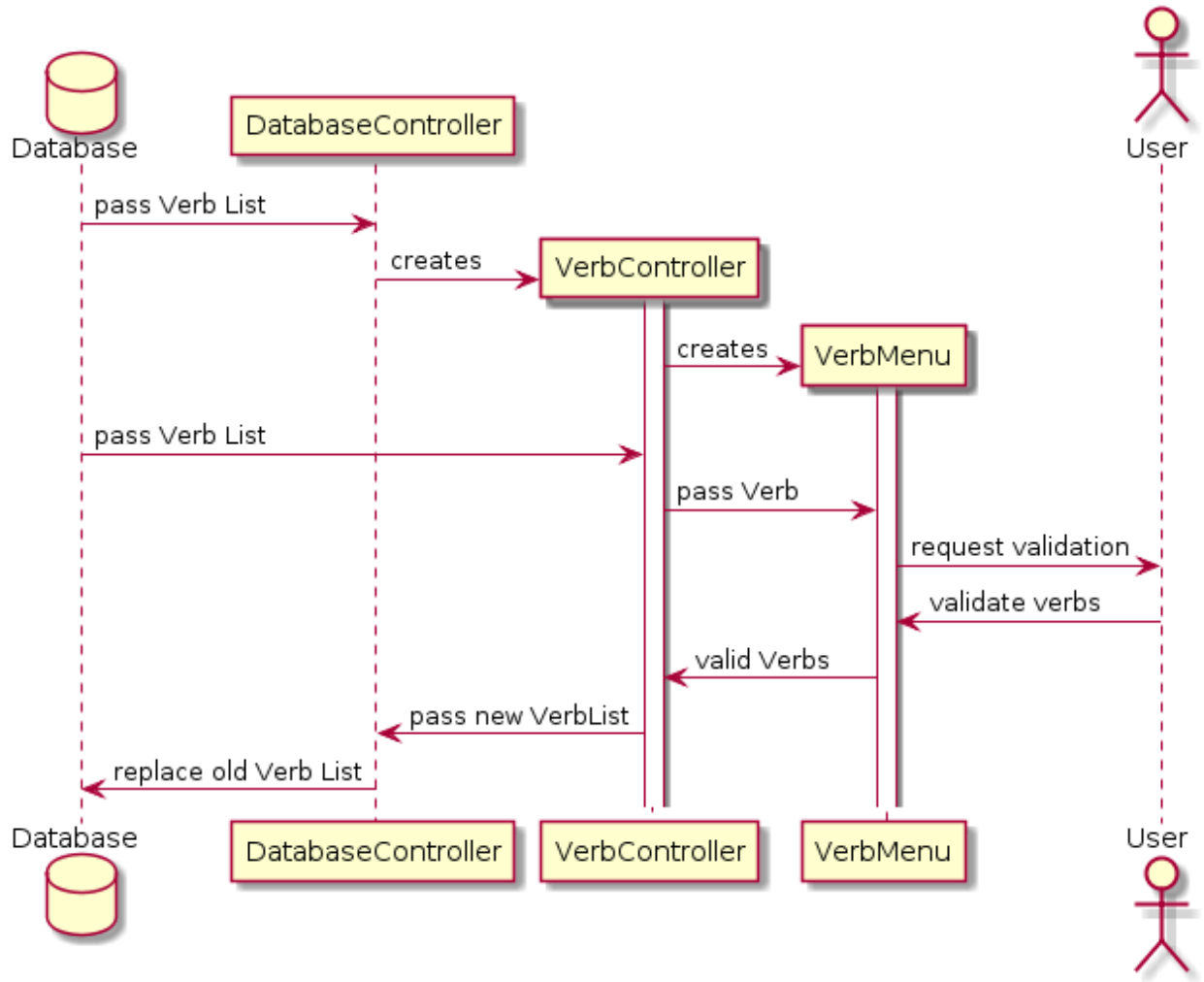
High Level SSD:



Concrete Use Case:

- **Summary:** Once the verbs have been extracted to the concept statement and stored in the database, the system will then display the verbs to the user and request that remove the verbs that are not needed.
- **Actors:** User and Database
- **Preconditions:** Language Parser has found all verbs
- **Description:** After the language parser has scanned through the concept statement and found all the verbs,they are then stored in the database. The system will then output a list of these verbs to the user. The user must then select the valid verbs from the list. The list of verbs is then returned to the database.
- **Exceptions:**
    - No verbs were found during the previous step
    - User selected no verbs
- **Postconditions:** User validates verbs and working verb set is stored in database

Detailed SSD:



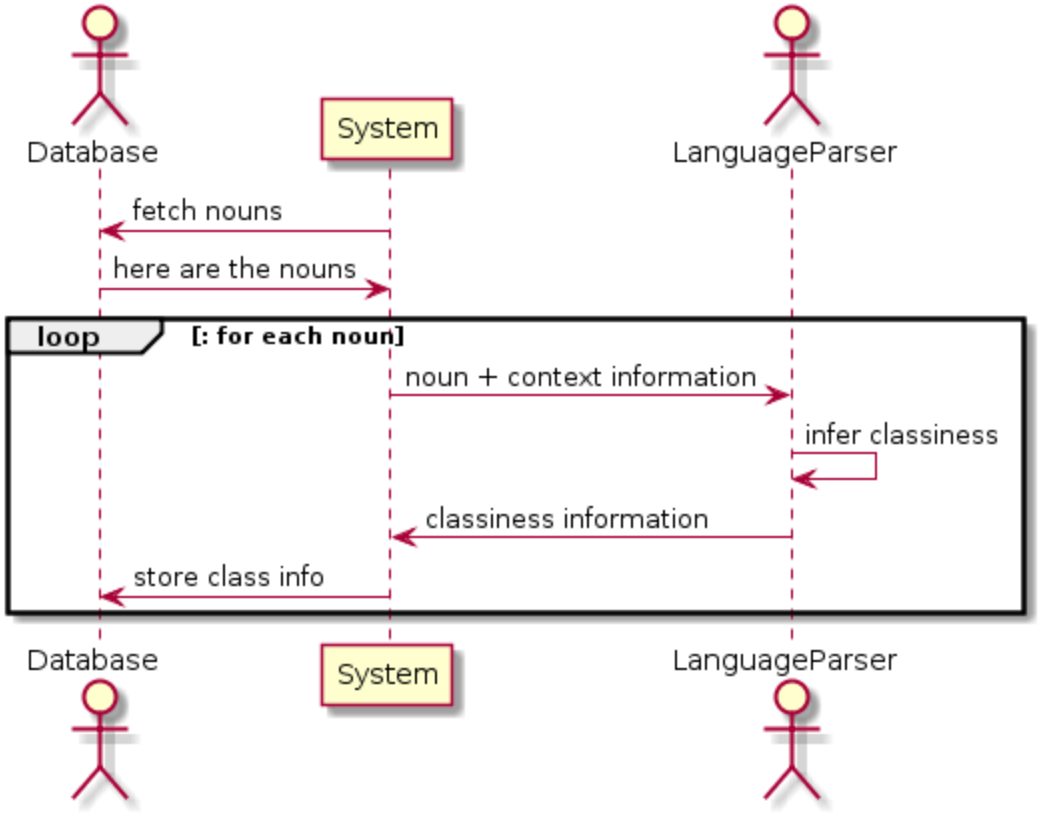
2.6 - Extract Classes from Concept Statement

Use Case:

Essential Use Case:

Summary: System gets nouns from database and analyses them against the domain-specific dictionary to determine classes.
Actors: LanguageParser, Database
Preconditions: Nouns have been parsed and validated
Description: LanguageParser will infer, based on importance/prominence of word (perhaps later based on the domain-specific knowledge-base), how likely a given noun or term is to be a class, and will create a class based on occurrence(s) of that term.
Exceptions: No nouns have been parsed, or incorrect words have been parsed
Postconditions: Classes are extracted
Scenario:
System grabs all nouns from database.
Database serves nouns to system
System analyses the 'classy' quality of a noun based on context
System presents the potential class to User
User modifies deletes or approves the class

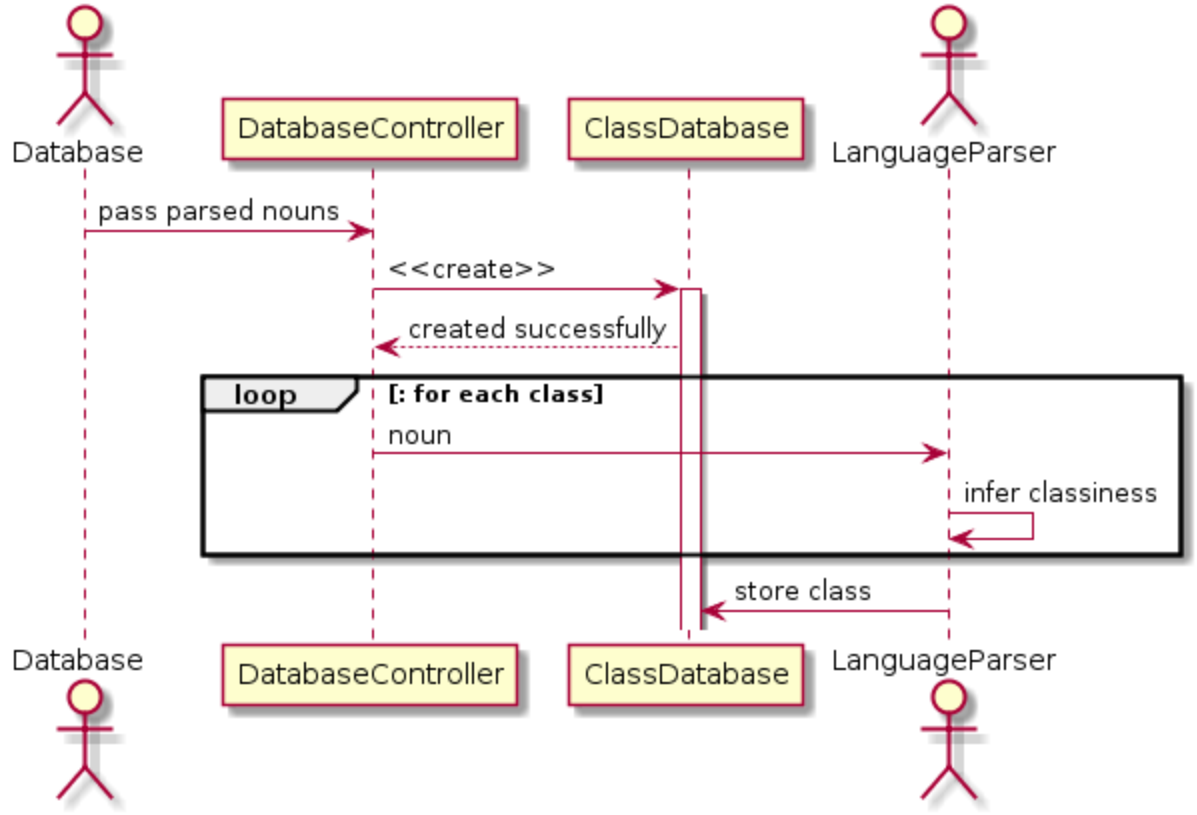
High Level SSD:



Concrete Use Case:

Summary: Database creates an Associations Database that the LanguageParser interacts with in order to infer associations between all established classes.
Actors: Database, LanguageParser, User
Preconditions: Nouns have been parsed and approved
Description: LanguageParser will use contextual information to determine if a noun is likely to be a class
Exceptions: If there are no associations or no nouns extracted
Postconditions: Potential associations are stored in the associations database

Detailed SSD:

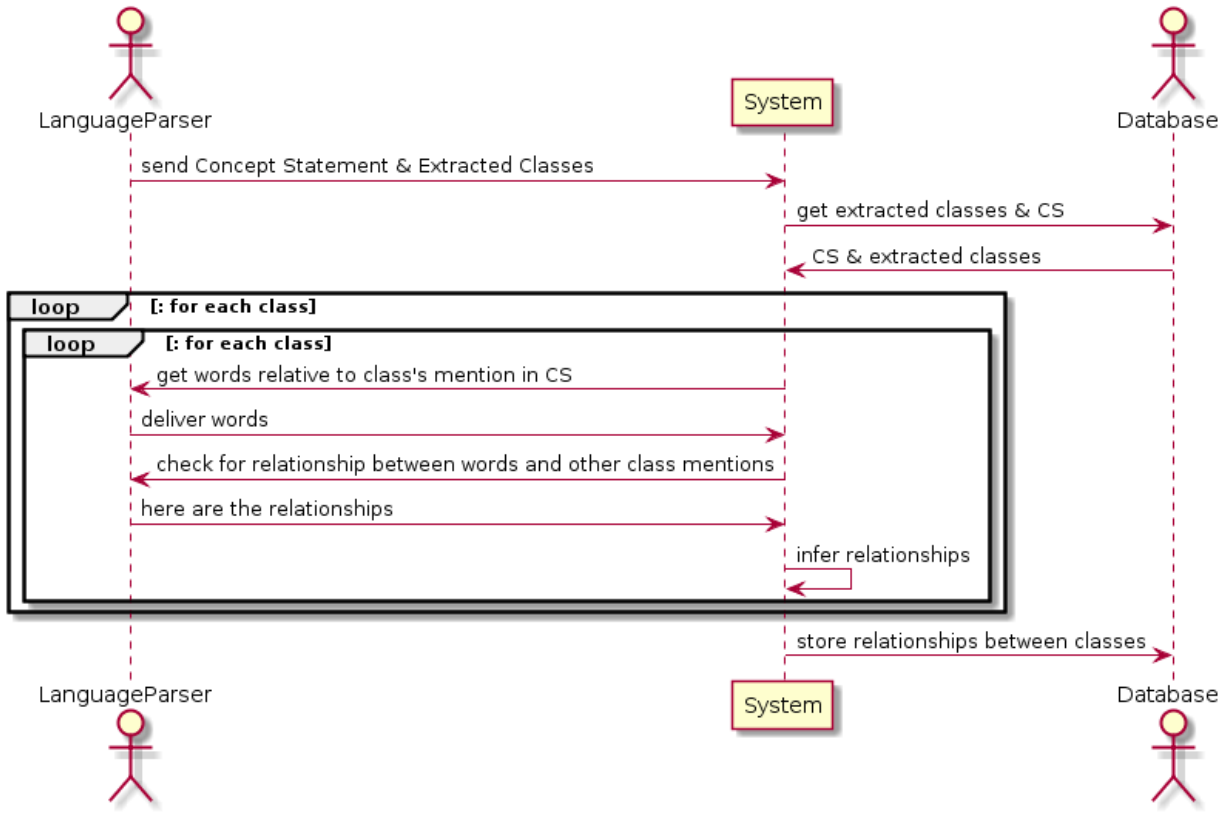


2.7 - Parse Associations

Essential Use Case:

Summary: Language Parser uses Extracted Classes and Concept Statement to parse relationships between classes
Actors: Language Parser, Database
Preconditions: Extracted Classes has been built
Description: System checks in the concept statement for relationships between classes in Extracted Classes based on information from the Language Parser, and it adds those relationships to the database.
Exceptions: Classes are not yet extracted
Postconditions: Associations are stored in the database
Scenario:
System asks for Concept Statement, Extracted Classes from database
Database send CS, extracted classes
Loop: for each class in Extracted Classes, Language Parser checks for a semantic association
System returns associations to database

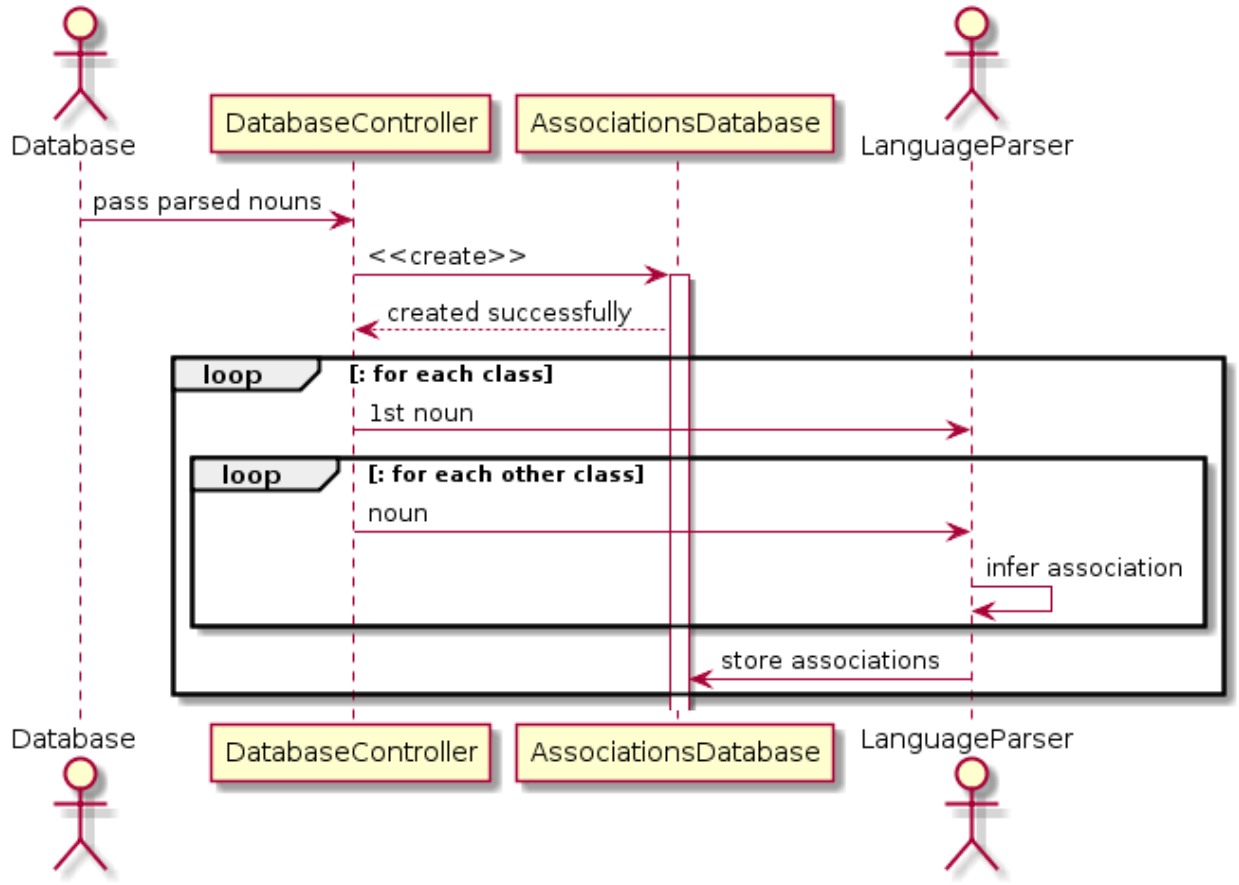
High Level SSD:



Concrete Use Case:

Summary: Database creates an Associations Database that the LanguageParser interacts with in order to infer associations between all established classes.
Actors: Database, LanguageParser
Preconditions: Nouns have been parsed and approved
Description: LanguageParser will use contextual information to determine if a relationship/association is likely to exist between two classes by analyzing the words that have relevance to the words that are associated with the names of two different classes.
Exceptions: If there are no associations or no nouns extracted
Postconditions: Potential associations are stored in the associations database

Detailed SSD:



2.8 - Validate Associations

Essential Use Case:

Summary: The system, through a GUI, presents the parsed associations to the user for approval/modification

Actors: System, User

Preconditions: Associations are parsed

Description: System presents every association between classes to the user with an interface through which the user may choose to modify, delete, or add an association between classes.

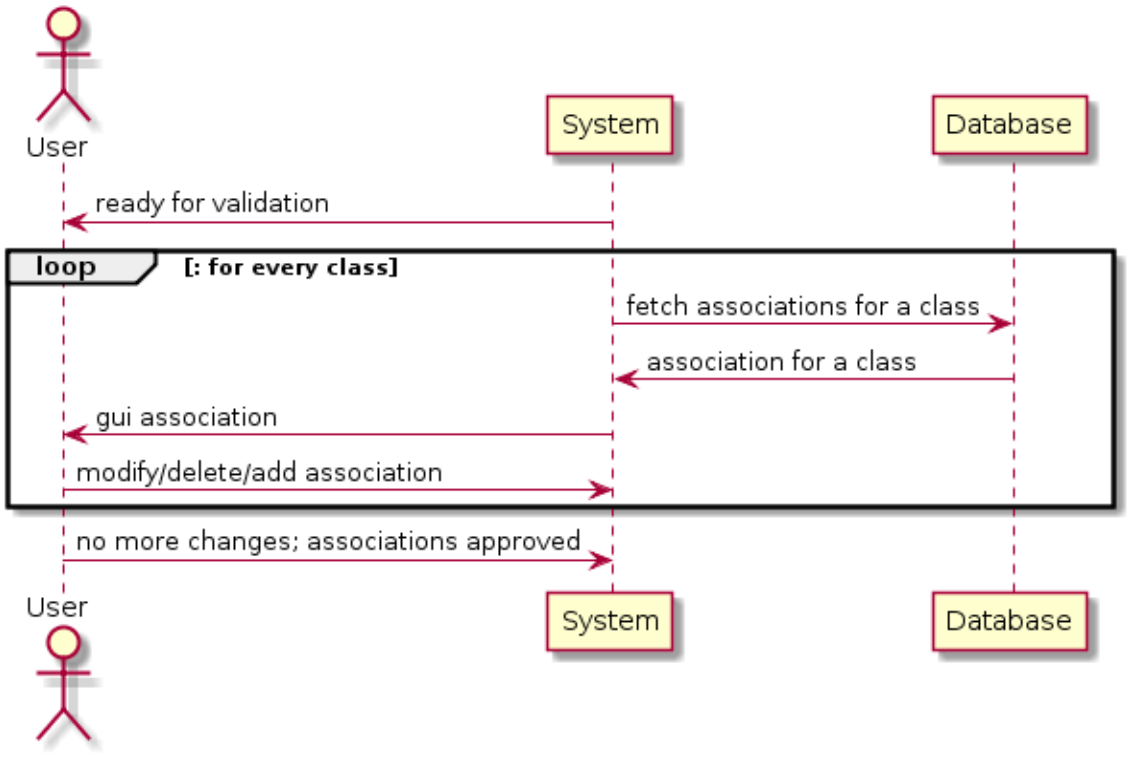
Exceptions: If associations not parsed

Postconditions: Associations are stored in database and system is ready to generate class diagrams.

Scenario:

System signals user that it is ready for validation.
User reviews all associations in a loop
User modifies the association
User deletes the association
User adds an association

High Level SSD:



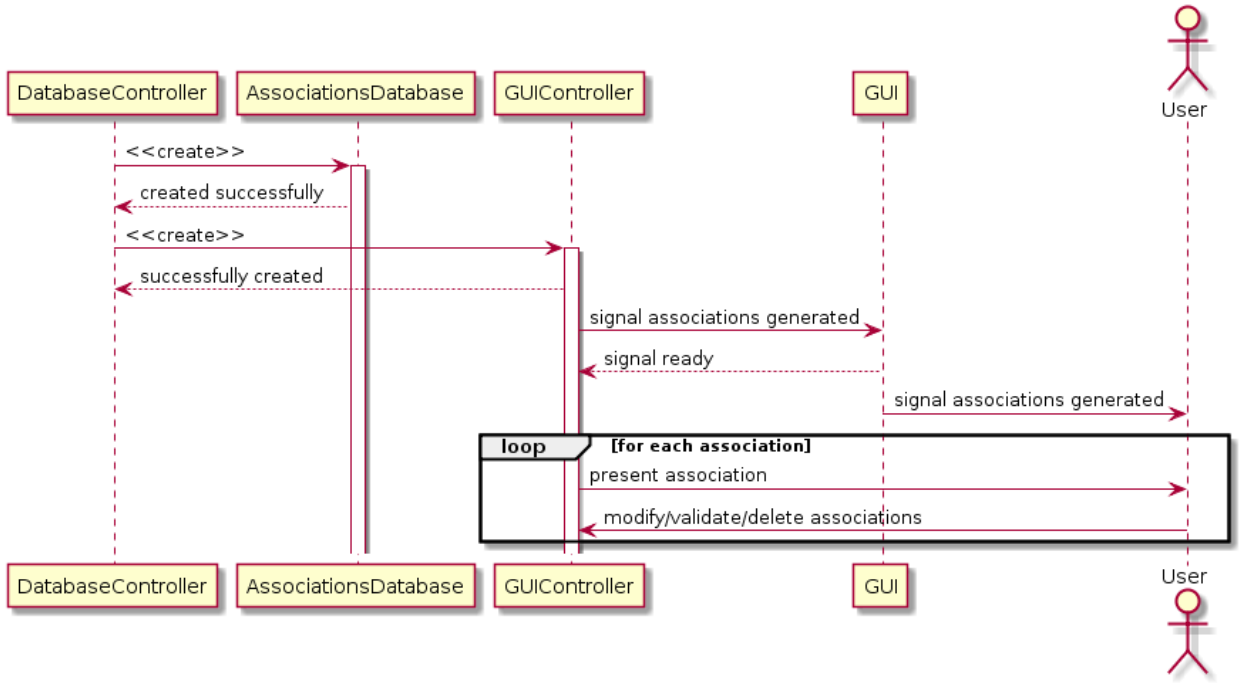
Concrete Use Case:

Summary: System prompts user to validate associations

Actors: User, System

Preconditions: Associations have been parsed from the problem statement
Description: System delivers, through GUI all of the found associations that for the approval/modification of the user. System allows user to create associations also.
Exceptions: power failure or other event that prevents User from finalizing all associations
Postconditions: All associations are validated or added, thus finalized

Detailed SSD:



2.9 - Build Class Diagram

Essential Use Case:

Summary: The System takes the list of nouns and verbs identified by the user after the system has parsed the concept statement. The system then generates a plantUML format text file, and displays the class diagram to the user

Actors: User, Database, System

Preconditions: Concept statement has been parsed and verbs and nouns identified and accepted by the user.

Description: The system takes the lists of nouns and verbs approved by the user, and their associations generated from parsing the concept statement, and generates a UML class diagram.

Exceptions: *Too many items*: plantuml has a limit to the number of objects it can handle, if this limit is exceeded, the user should be informed and advised to reduce the number of classes.

Postconditions: A UML class diagram has been generated as well its associated plantUML source code stored in the Database

Scenario:

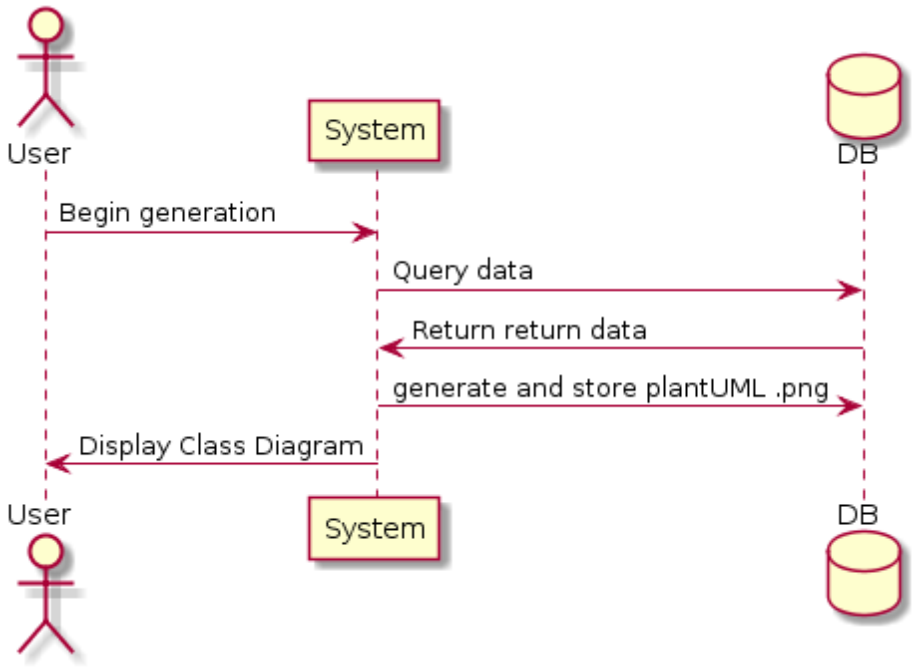
The user clicks proceed.

The system then queries the database for needed data

The system creates a plantUML source file

The system generates a plantUML .png file of the class diagram

This diagram is displayed to the user.



Concrete Use Case:

Summary: The user clicks the GUI icon to begin generation of a class diagram

Actors: User, Database, Database Controller System, GUI

Preconditions: Concept statement has been parsed and verbs and nouns identified and accepted by the user.

Description: The user selects proceed. The system then queries the database for nouns and their verb associations, storing these in a data structure for processing. The system then generates the plantUML source code for the nouns and verb associations. The system then generates a .png file of the source file, and displays it to the user.

Exceptions:

Too many items: plantuml has a limit to the number of objects it can handle, if this limit is exceeded, the user should be informed and advised to reduce the number of classes.

Postconditions: A plantUML source code file has been generated as well as its associated .png and both are stored in the database

Scenario:

The user clicks proceed.

The system then queries the database for the list of nouns it stores this for later processing

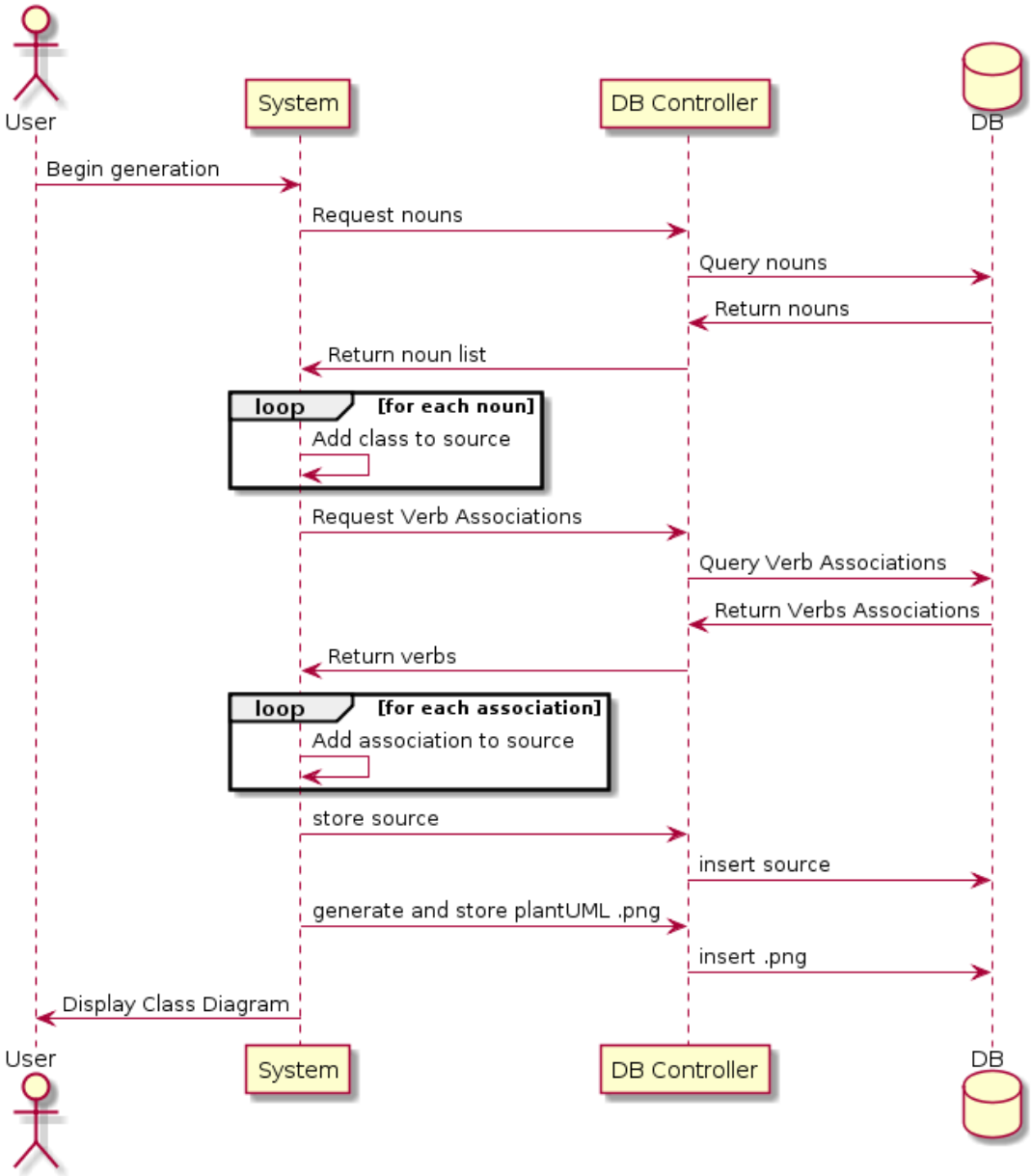
The system then queries the DB for the list of verb associations for the nouns it has retrieved and stores these for processing.

The system then generates plantUML code for the classes based on the noun list.

The system connects these nouns using the list of verb associations, through plantUML.

The system generates a plantUML .png file of the class diagram

This diagram is displayed to the user.

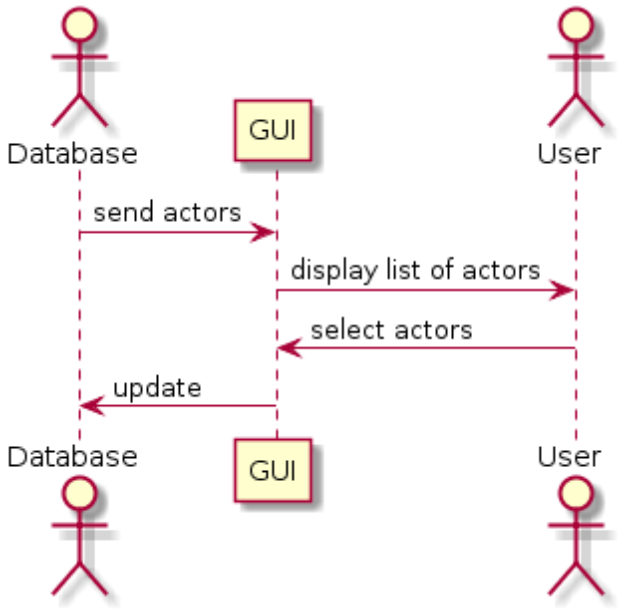


2.10 - Identify Actors

Essential Use Case: Identify Actors

- **Summary:** After the class diagram has been built and stored in the database. The system will them prompt the user to select the names from the diagram that will serve as actors when the use case diagrams are created.
- **Actors:** User and Database
- **Precondition:** Class diagram has been built
- **Description:** The system prompts the User to select from the class diagram class names that will also serve as actors for the use case diagram.
- **Exceptions:**
    - No class diagram found
- **Postconditions:**
    - A list of actors is created for the use case diagram.

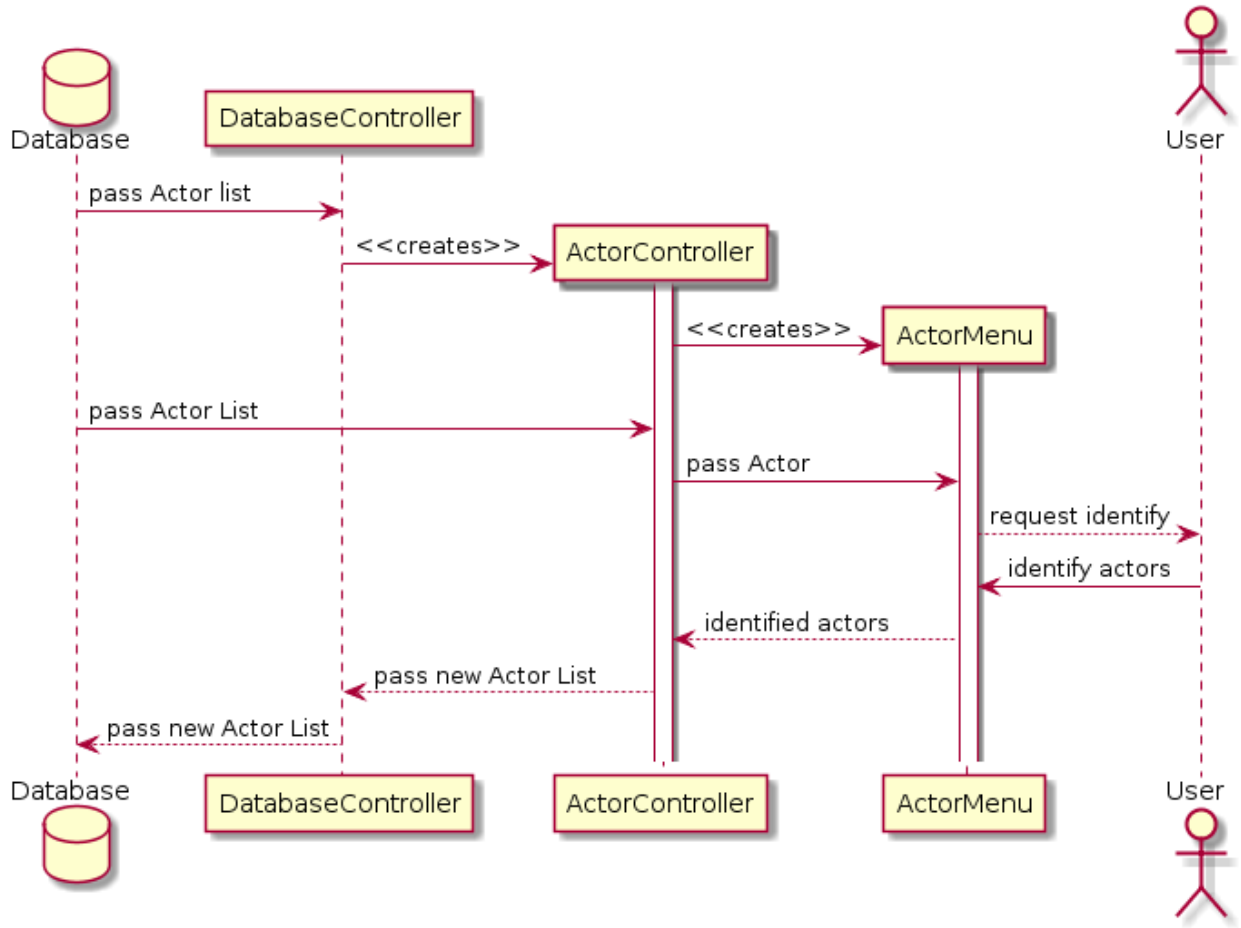
Scenario:
Displays the list of actors to the user
Prompts the user to select actors
User selects valid actors
Database is updated with new list of actors.
High Level SSD:



Concrete Use Case: Identify Actors

- **Summary:** After the class diagram has been built and stored in the database. The system will them prompt the user to select the names from the diagram that will serve as actors when the use case diagrams are created
- **Actors:** User and Database
- **Preconditions:** A class diagram is currently stored in the database
- **Description:** After the class diagram has been created, it is then stored in the database. The class names are displayed on the screen and the user is prompted to choose the names that will be used as actors.
- **Exceptions:**
    - No class diagram has been created
- **Postconditions:**
    - A list of actors is created and is stored in the database

Detailed SSD:

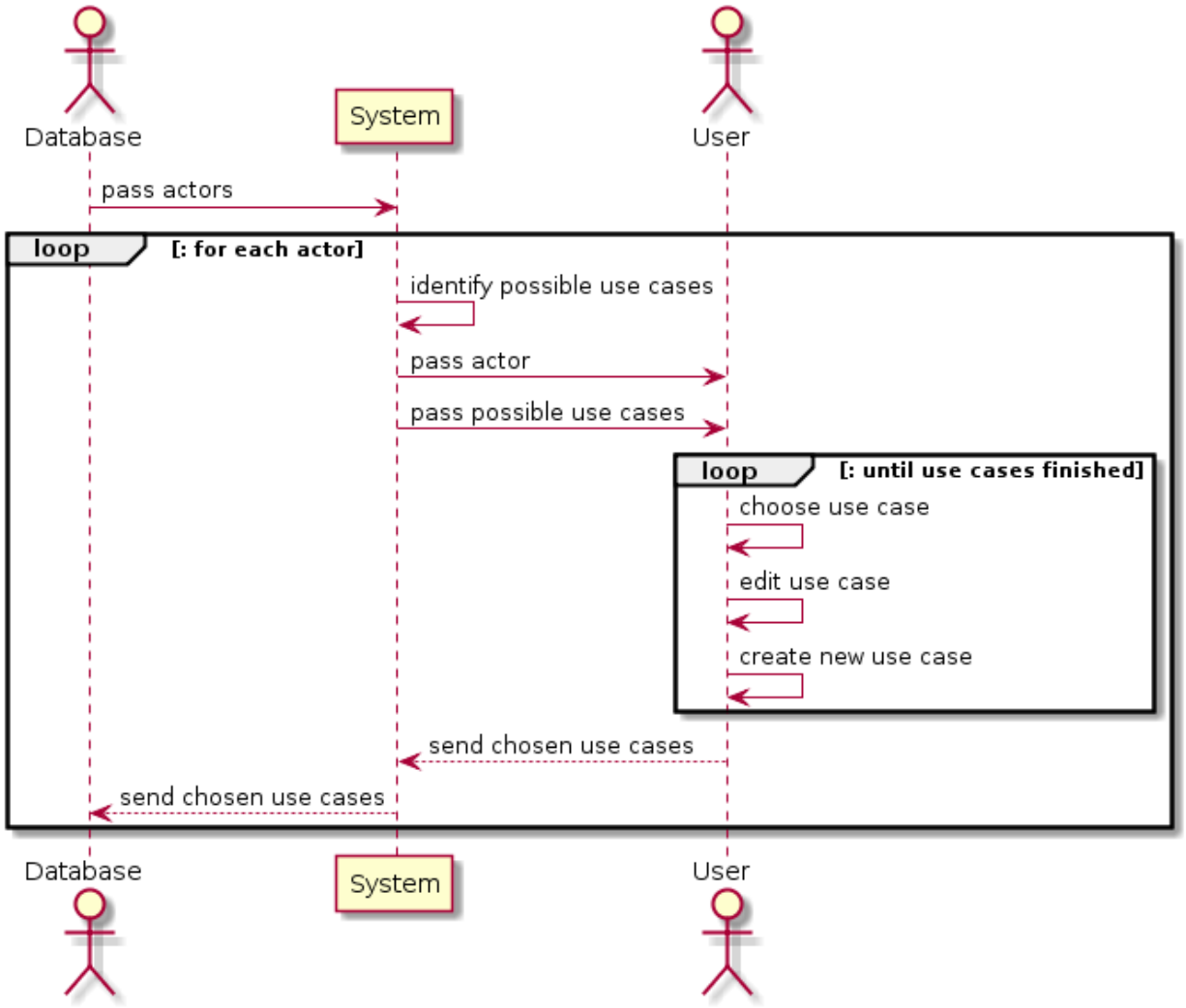


2.11 - Generate Basic Use Case

Essential Use Case:
Summary: The system uses the list of actors and class diagram, along with user input, to identify and create use cases.
Actors: User and Database
Preconditions: Class diagram has been created and is stored in the database, and actors have been identified and are stored in the database.
Description:
For each actor, the system identifies, and shows to the user, possible use cases. Possible use cases are classes in the class diagram connected to the current actor, which are not themselves actors. The user selects and/or edits use cases and, if necessary, creates any use cases for the actor.
Exceptions:
No Use Cases for Actor: No use cases for an actor are identified by the user. This is not allowed since actors only exist to participate in use cases. A suitable error message is shown to the user.
Actor-Name Use Case: A use case can not have the same name (or be the same thing) as an actor. Actor to actor associations are not allowed for use cases. A suitable error message is shown to the user.
Postconditions: Use Cases for each actor have been generated and are stored in the database.

Scenario:
Database sends actors to the System.
System chooses an actor and sends it to the User.
System identifies possible use cases.
System sends possible use cases to the User.
User chooses applicable use cases from those displayed.
User edits one of the displayed use cases to be more accurate.
User creates a new use case for the actor.
User sends identified use cases for the actor to the System.
System sends identified use cases from the actor to the Database.
Database stores use cases for the actor.
System chooses the next actor and repeats the steps above until all actors have use cases.

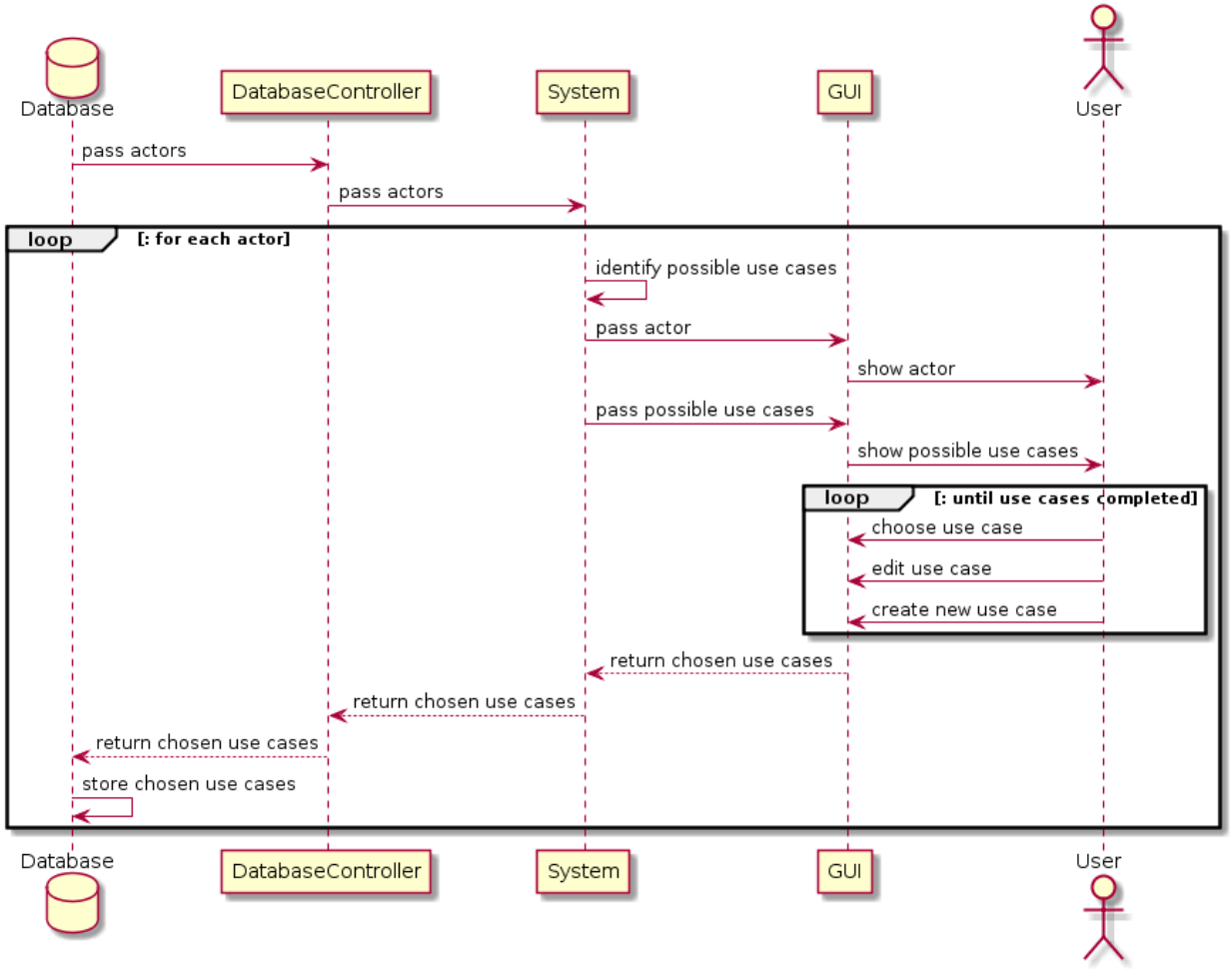
High Level SSD:



Concrete Use Case:

Summary: The system uses the list of actors and class diagram, along with user input, to identify and create use cases.
Actors: User and Database
Preconditions: Class diagram has been created and is stored in the database, and actors have been identified and are stored in the database.
Description:
For each actor, the system identifies, and shows to the user, possible use cases.
To identify a possible use case, the class associated with the current actor, from the class diagram, is identified. All classes associated with this class (linked by an association) are possible use cases. From these classes, those that share a name with an actor are removed, because an actor to actor link is not allowed. These classes are now the possible use cases for the original actor.
The user chooses from the list of possible use cases those that they think are correct use cases. The user can also add their own use cases that are not in the list of possible use cases. The user can also edit use cases so they are more correct.
The system saves these identified use cases into the database.
Exceptions:
No Use Cases for Actor: No use cases for an actor are identified by the user. This is not allowed since actors only exist to participate in use cases. A suitable error message is shown to the user.
Actor-Name Use Case: A use case can not have the same name (or be the same thing) as an actor. Actor to actor associations are not allowed for use cases. A suitable error message is shown to the user.
Postconditions: Use Cases for each actor have been generated and are stored in the database.

Detailed SSD:

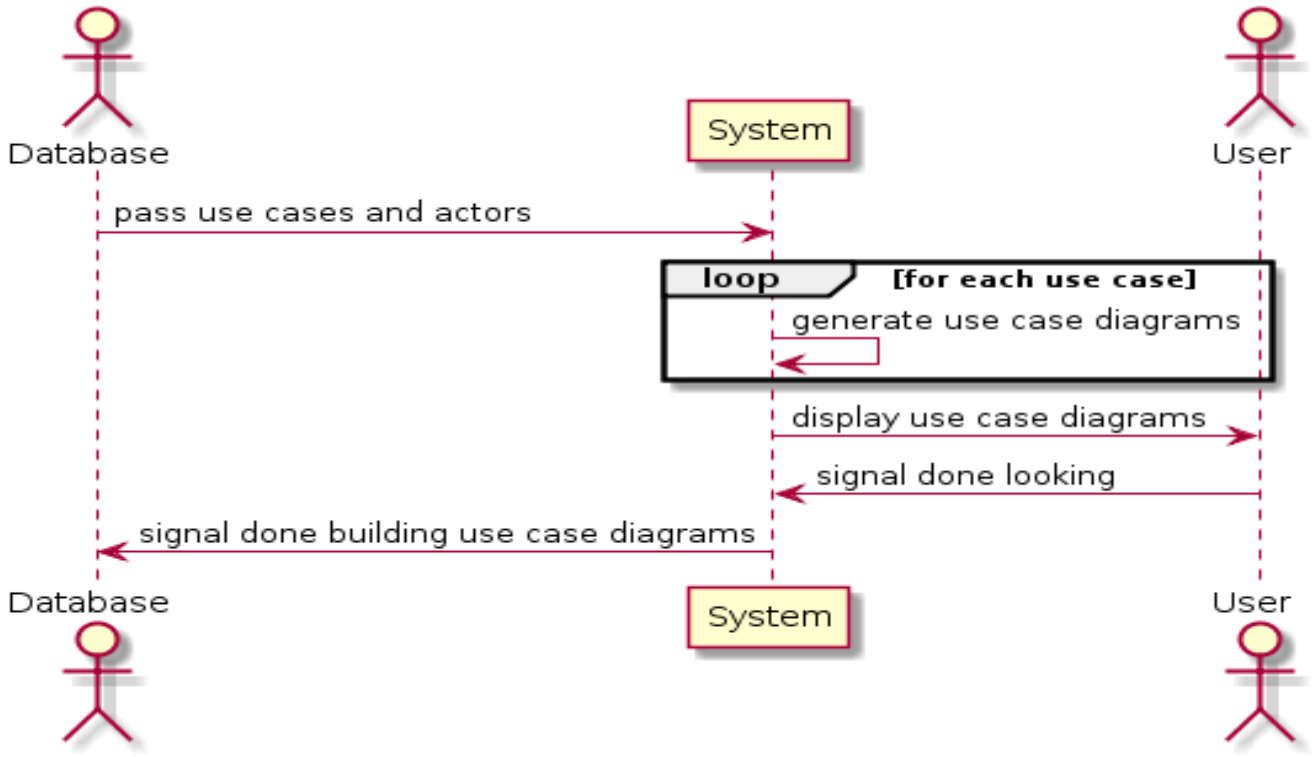

Database
DatabaseController
System
GUI
User
pass actors
pass actors
loop
[: for each actor]
identify possible use cases
pass actor
show actor
pass possible use cases
show possible use cases
loop
[: until use cases completed]
choose use case
edit use case
create new use case
return chosen use cases
return chosen use cases
return chosen use cases
store chosen use cases
Database
DatabaseController
System
GUI
User

2.12 - Build Use Case

Essential Use Case:

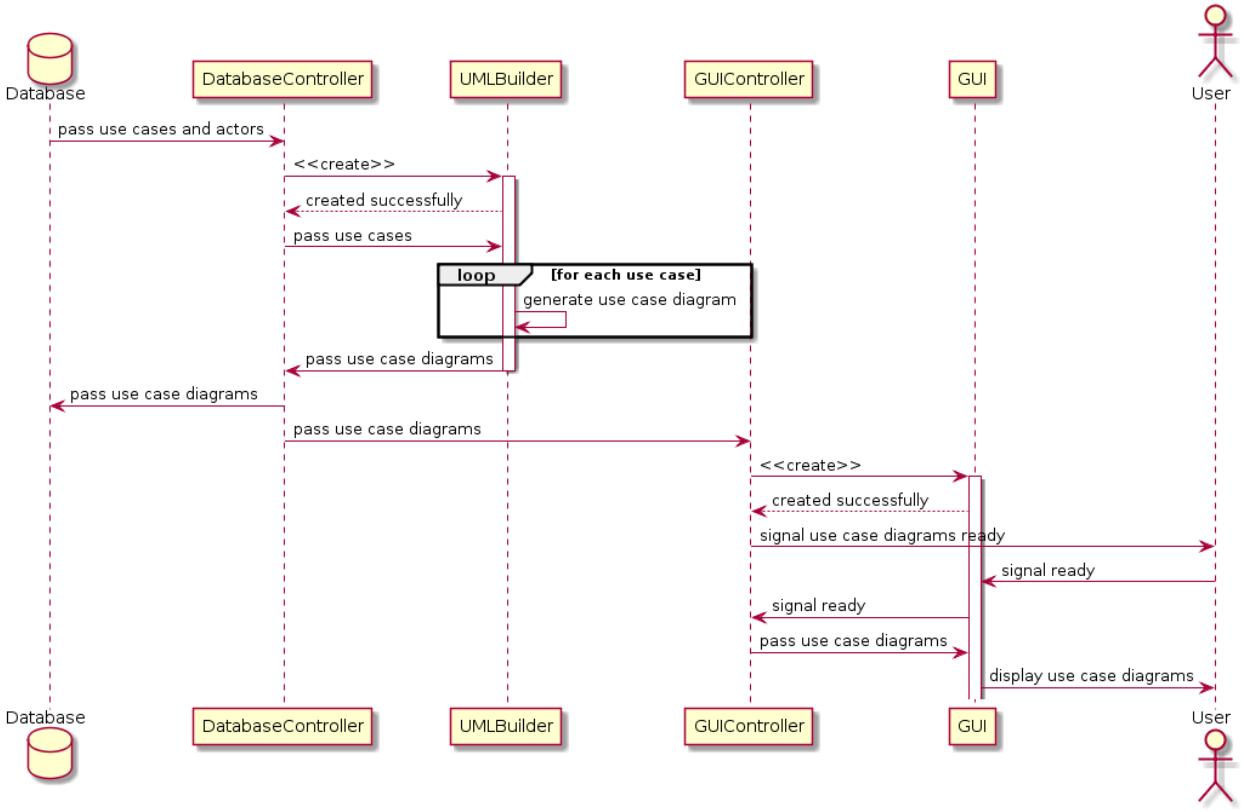
Summary: The database passes the system use cases and the system generates use case diagrams.
Actors: User, Database
Preconditions: Use cases have been written.
Description: System generates use case diagrams from the use cases.
Exceptions:
No nouns or verbs identified: No nouns or verbs are identified by the Language Parser. A suitable error is shown to the user.
Postconditions: Use case diagram is built.

Scenario:
Database passes the system use cases.
System generates use cases.
System displays use cases to user.
User signals that they are done looking.
System signals the database that use case diagrams are complete.



Concrete Use Case:

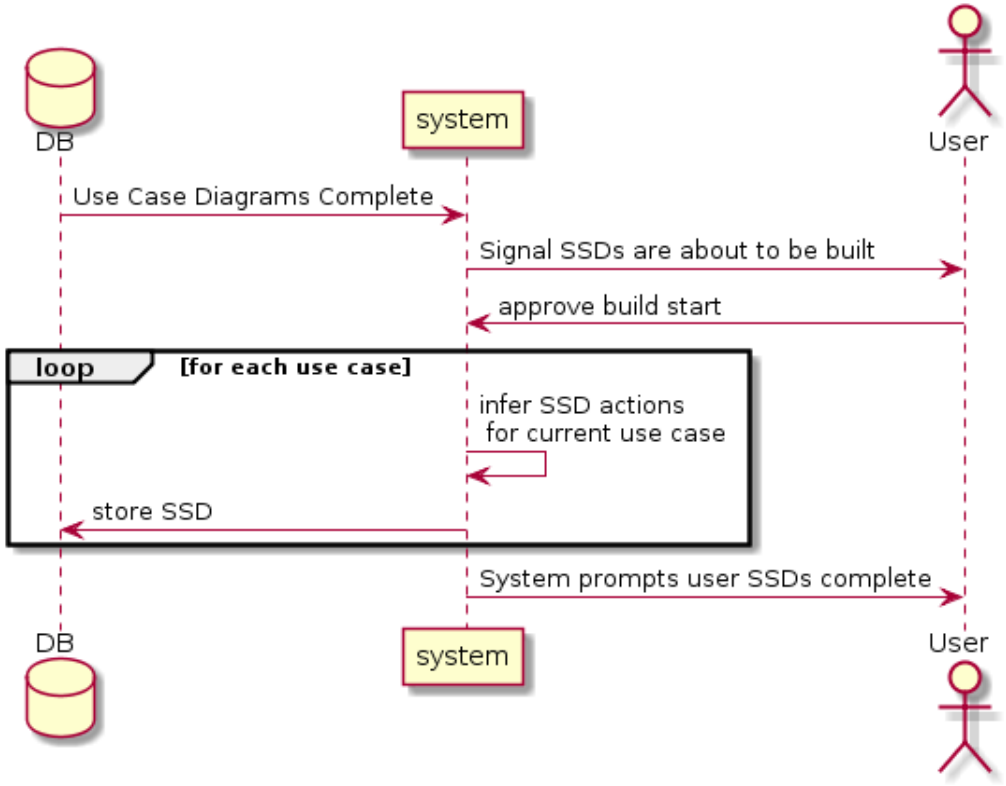
Summary: The database passes the system use cases and the system generates use case diagrams.
Actors: Database, User
Preconditions: Use cases have been written.
Description: System generates use case diagrams from the use cases.
Exceptions:
No nouns or verbs identified: No nouns or verbs are identified by the Language Parser. A suitable error is shown to the user.
Postconditions: Use case diagram is built.



2.13 - Build System Sequence Diagram

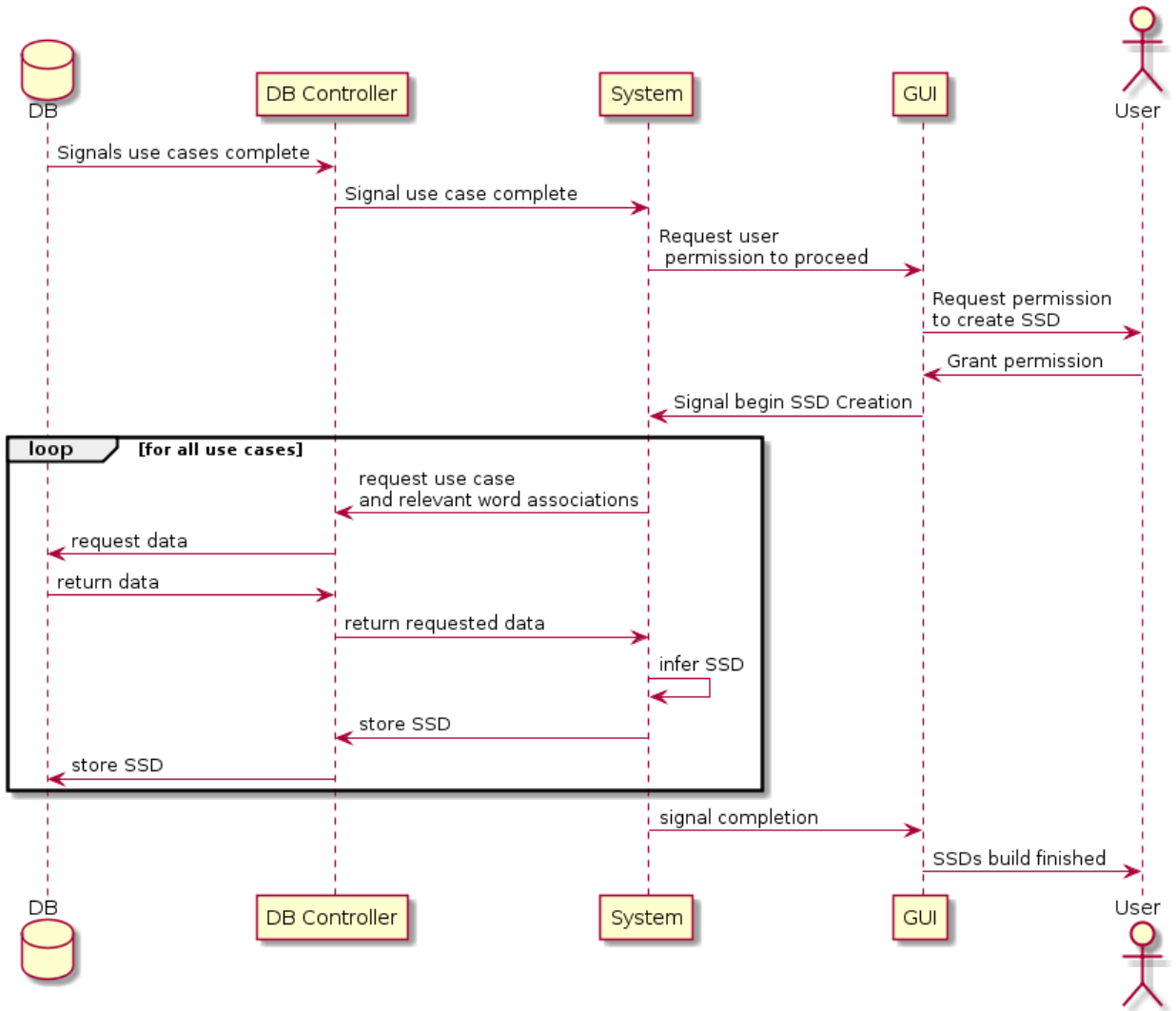
Essential Use Case:

Summary: The System uses the Database and user approved use cases to build system sequence diagrams.
Actors: User, Database, System
Preconditions: Use Cases Diagram(s) have been been built.
Description: System generates System Sequence Diagrams (SSDs) from the user approved use cases and previous analysis of concept statement.
Exceptions:
Noun verb interactions unclear: The system is unable to differentiate which verbs are acting on which nouns, and cannot create their SSD.
Postconditions: System Sequence Diagram(s) has been built.
Scenario:
Database signals that the use cases have been completed.
System informs the user it will begin building SSDs.
System infers interactions between actors from user approved use cases and previous analysis of concept statement.
System creates a SSD for each use case.
System prompts user that the SSDs are complete.



Concrete Use Case:

Summary: The system uses the users approved use case diagrams and previous concept statement analysis stored in the database to infer and create SSDs.
Actors: User, Database, System, GUI
Preconditions: Use Case Diagrams have been built
Description: System prompts user for permission to begin building SSDs. It then accesses the Database and
Exceptions:
Noun verb interactions unclear: The system is unable to differentiate which verbs are acting on which nouns, and cannot create their SSD.
Postconditions: SSDs have been created for each use case.



2.14 - Create System Design

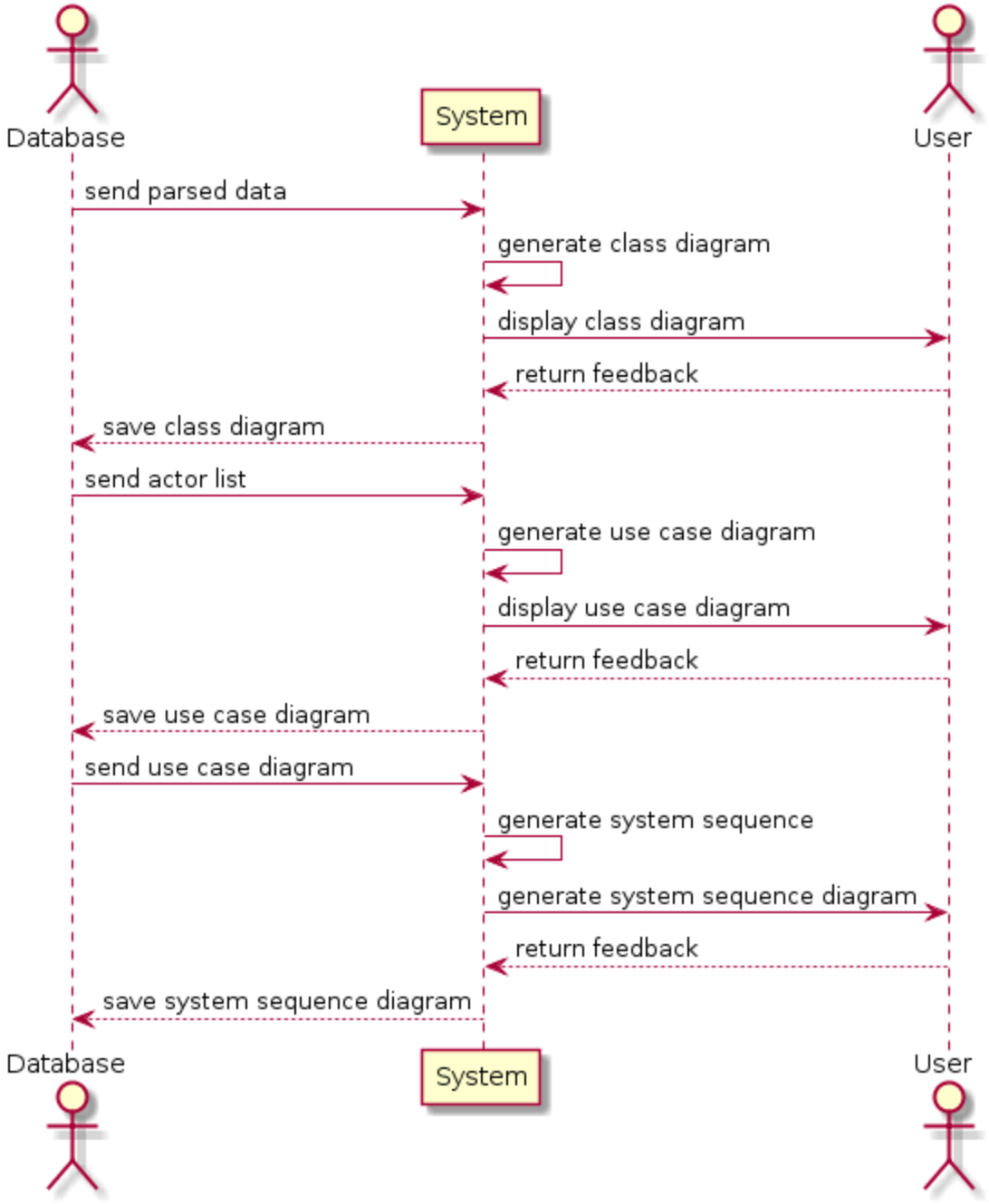
Essential Use Case: Create System Design

- **Summary:** After the user has read in a concept statement into the UML Magic wizard
- **Actors:** User and Database
- **Preconditions:** A concept statement has been read into the database.
- **Description:** The User interacts with the System through the gui to coordinate the development of all components of the UML Magic design package.
- **Exceptions:**
    - No concept statement found in the database.
- **Postconditions:** UML Magic diagram package complete.

Scenario:

After the User has read in the concept statement
The Database sends the parsed data to the System
System generates the class diagram and displays to the User
User returns necessary feedback to the System
System saves class diagram to Database
Database sends actor list to the System
System generates the use case diagram and displays to the User
User returns necessary feedback to the System
System saves use case diagram to the Database
Database sends use case diagram to the System
System generates the system sequence diagram and displays to the User
User returns necessary feedback to the System
System saves system sequence diagram to the Database

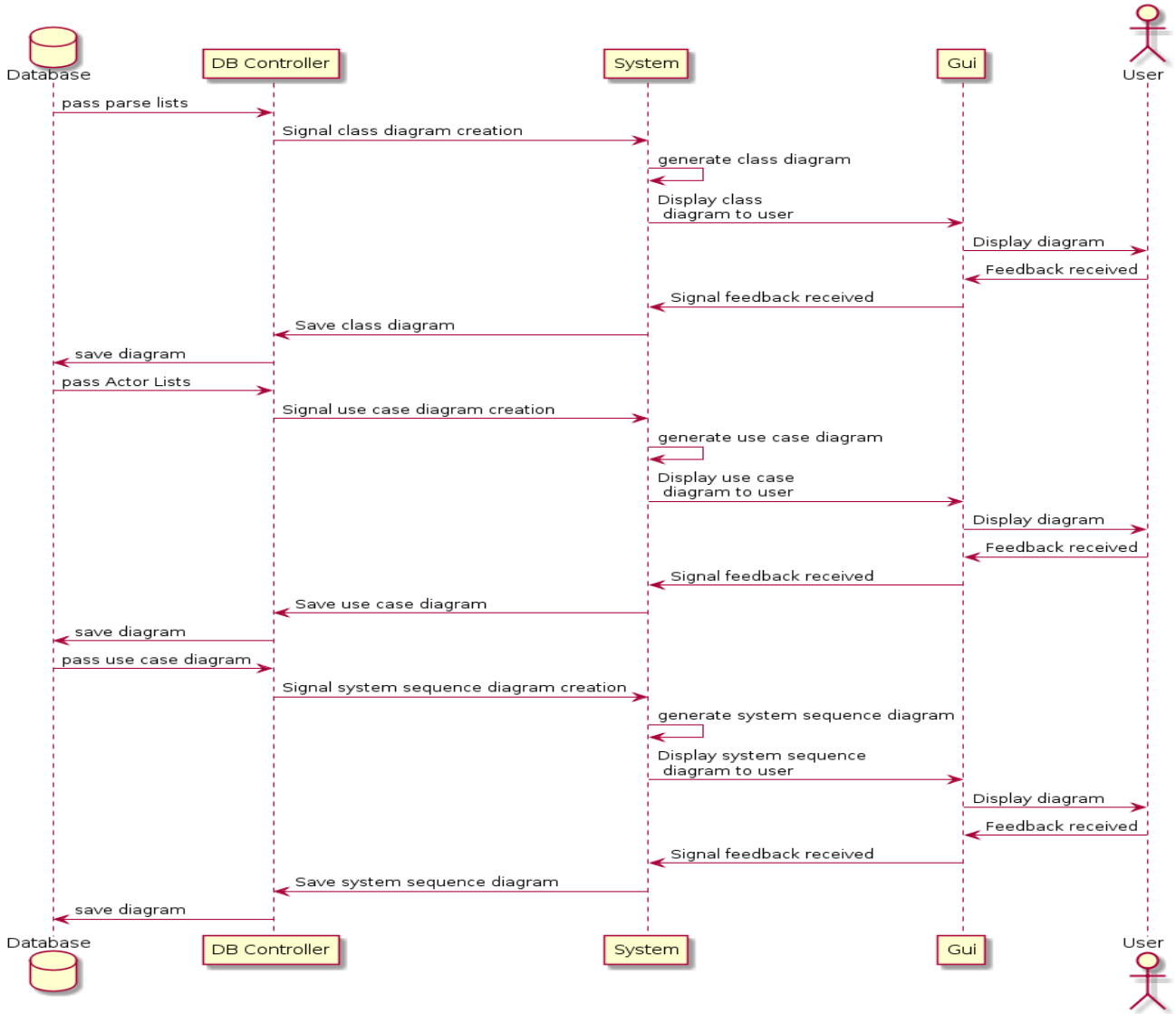
High Level SSD:



Concrete Use Case:

- **Summary:** After the user has read in a concept statement into the UML Magic wizard
- **Actors:** User and Database
- **Preconditions:** A concept statement has been read into the UML Magic wizard.
- **Description:** After the concept statement has been parsed into the database, the system is ready to generate the required diagrams. The system the displays the diagrams to the user to view and make the required feedback.
- **Exceptions:**
    - No concept statement found
    - No parsed noun list found
    - No parsed verb list found
- **Postconditions:**
    - UML package diagram complete

Detailed SSD:



2.15 - Print Output

Essential Use Case:

Summary: The System displays a report of all the documents created, bundles them into a file, and presents the user with the opportunity to physically print or save the output as a pdf or similar file format.

Actors: User, Database, System

Preconditions: At least one diagram has been completed

Description: After the user has created diagrams the option to display them in a printable format becomes available. The user selects print or export. The diagrams are bundled into a pdf document and the user may either save or print them.

Exceptions:

No Diagrams have been created. If the user has not created any diagrams the program will alert the user that print/export is unavailable.

Postconditions: pdf (or similar format) bundle of diagrams will be available for the user to print or save to disk.

Scenario:

User selects print/export

System queries database for a list of completed documents

System displays a selection window for the user to select which diagrams they would like to export. (check all option)

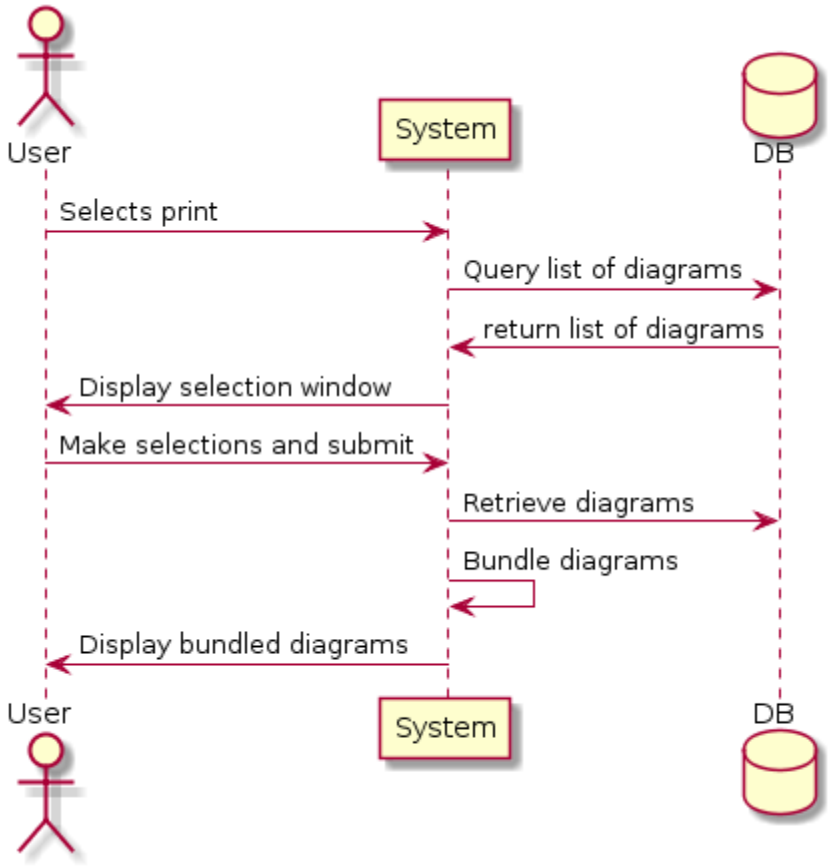
The user makes their selections and clicks submit.

The system bundles these diagrams and generates a pdf (or similar format).

The system notifies the user upon completion, either through message box or by auto opening the created document.

The user now has the option to save, print , or both.

High Level SSD



Concrete Use Case:

Summary: User selects print/export. The system presents the user with a list of available diagrams. The user selects all diagrams for export and submits their request. The system bundles all available documents into a pdf, and opens it.

Actors: User, Database, System, GUI

Preconditions: at least one diagram has been completed

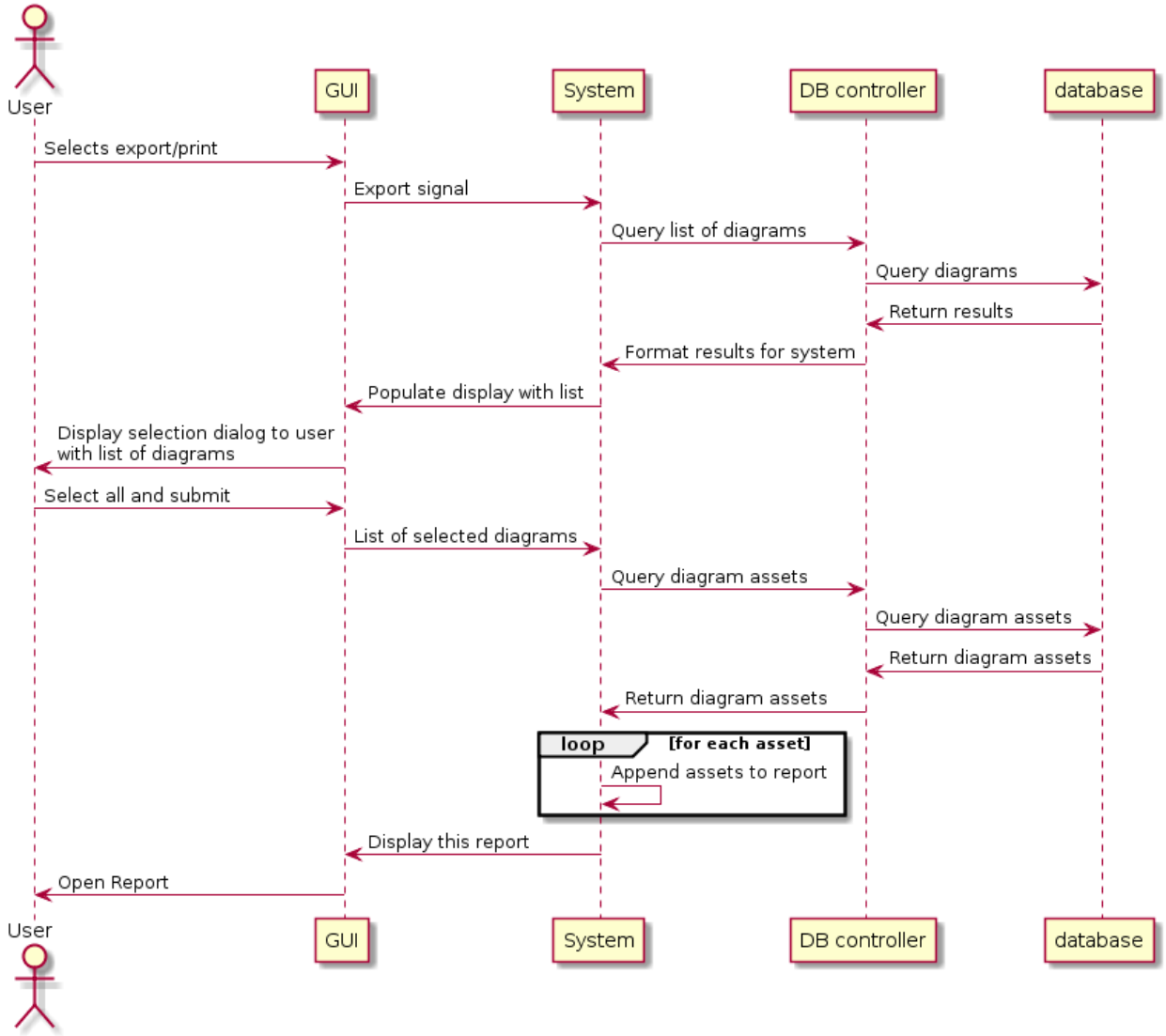
Description:

Exceptions:

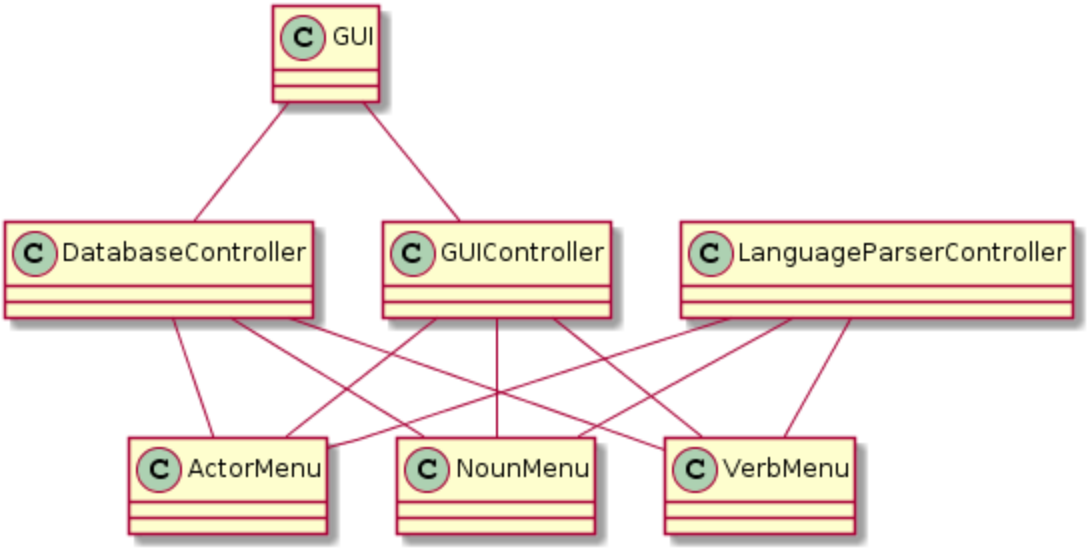
No diagrams available. The system displays a message box informing the user they must first create a diagram. (the option should be greyed out in the gui until this point, but this error handling should be built in as well)

Postconditions: A report style file has been created which the user may print, save, or both.

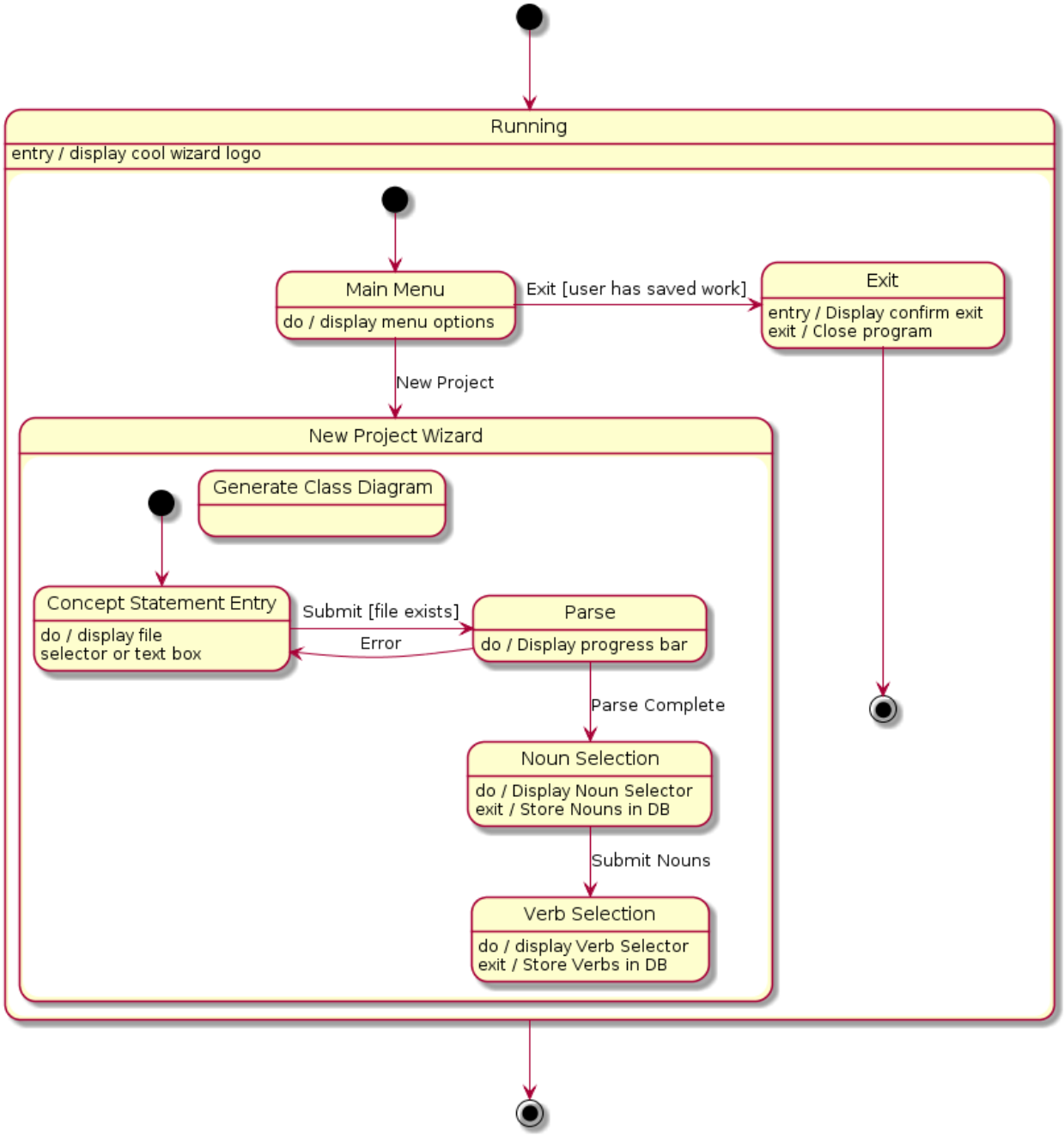
Detailed SSD

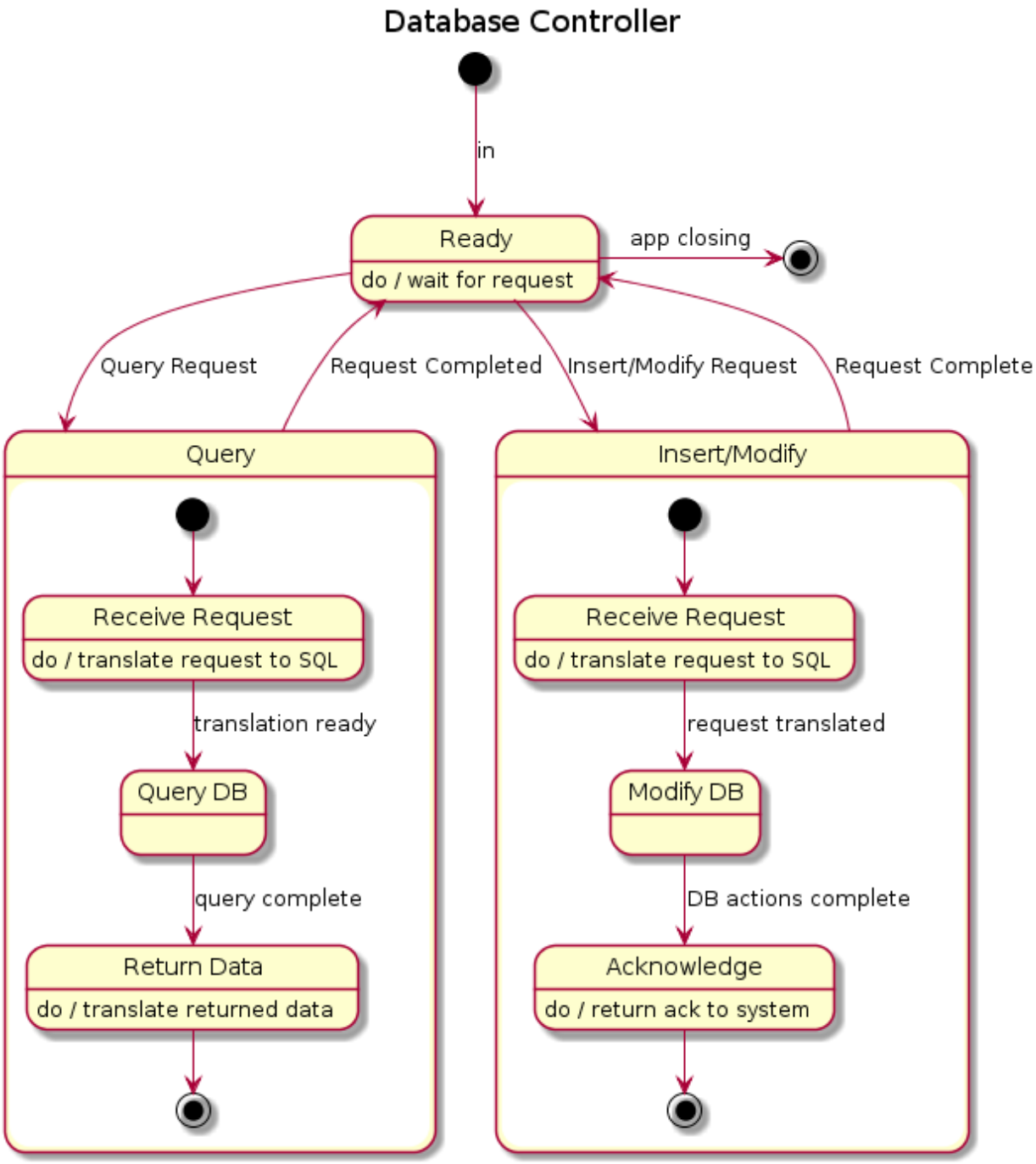

User
GUI
System
DB controller
database
Selects export/print
Export signal
Query list of diagrams
Query diagrams
Return results
Format results for system
Populate display with list
Display selection dialog to user with list of diagrams
Select all and submit
List of selected diagrams
Query diagram assets
Query diagram assets
Return diagram assets
Return diagram assets
loop
[for each asset]
Append assets to report
Display this report
Open Report
User
GUI
System
DB controller
database

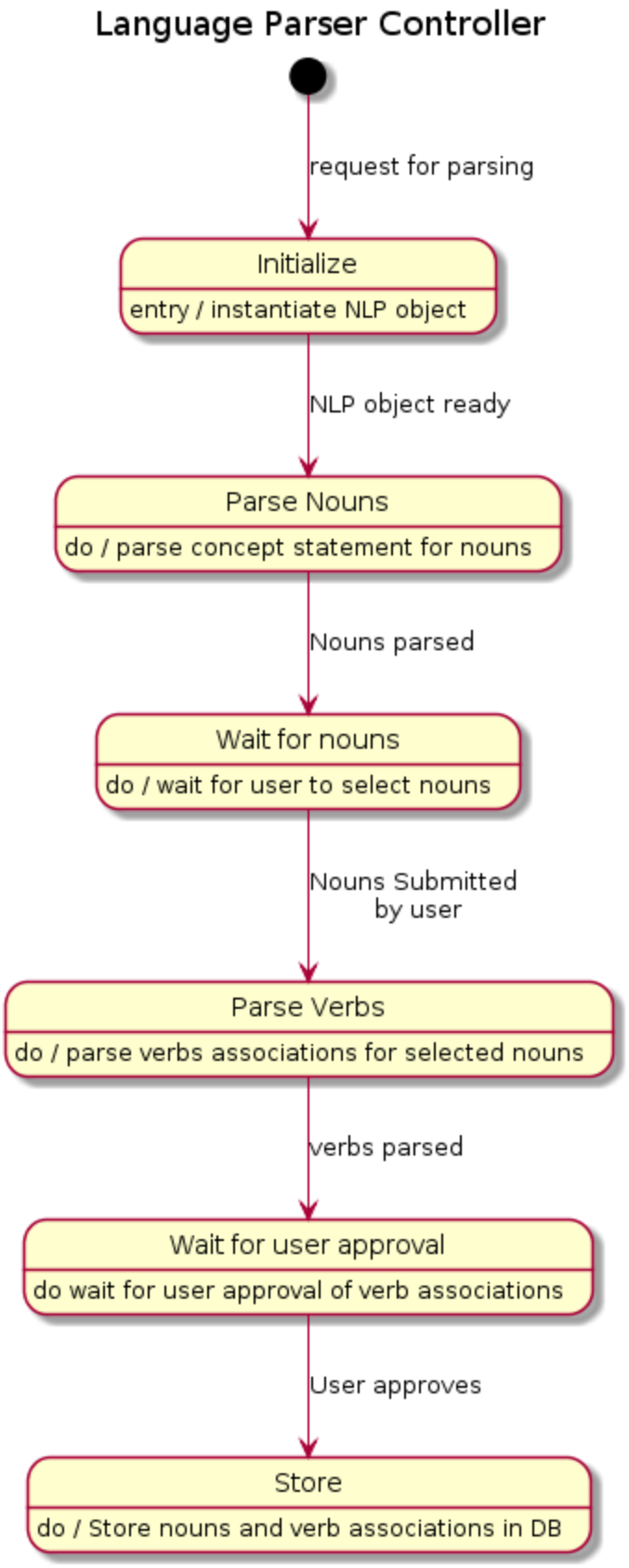
Application Class Model



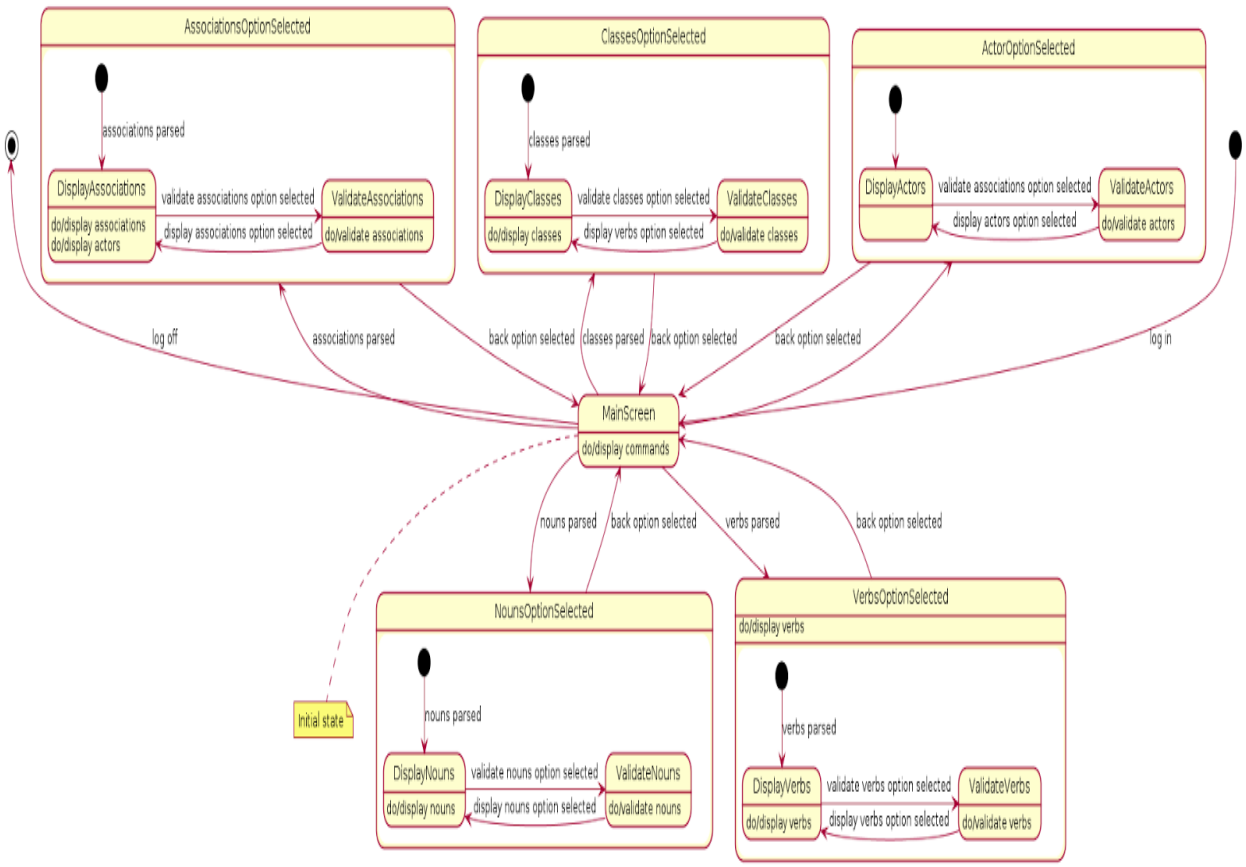
Application State Model




Database Controller
in
Ready
do / wait for request
app closing
Query Request
Request Completed
Insert/Modify Request
Request Complete
Query
Receive Request
do / translate request to SQL
translation ready
Query DB
query complete
Return Data
do / translate returned data
Insert/Modify
Receive Request
do / translate request to SQL
request translated
Modify DB
DB actions complete
Acknowledge
do / return ack to system

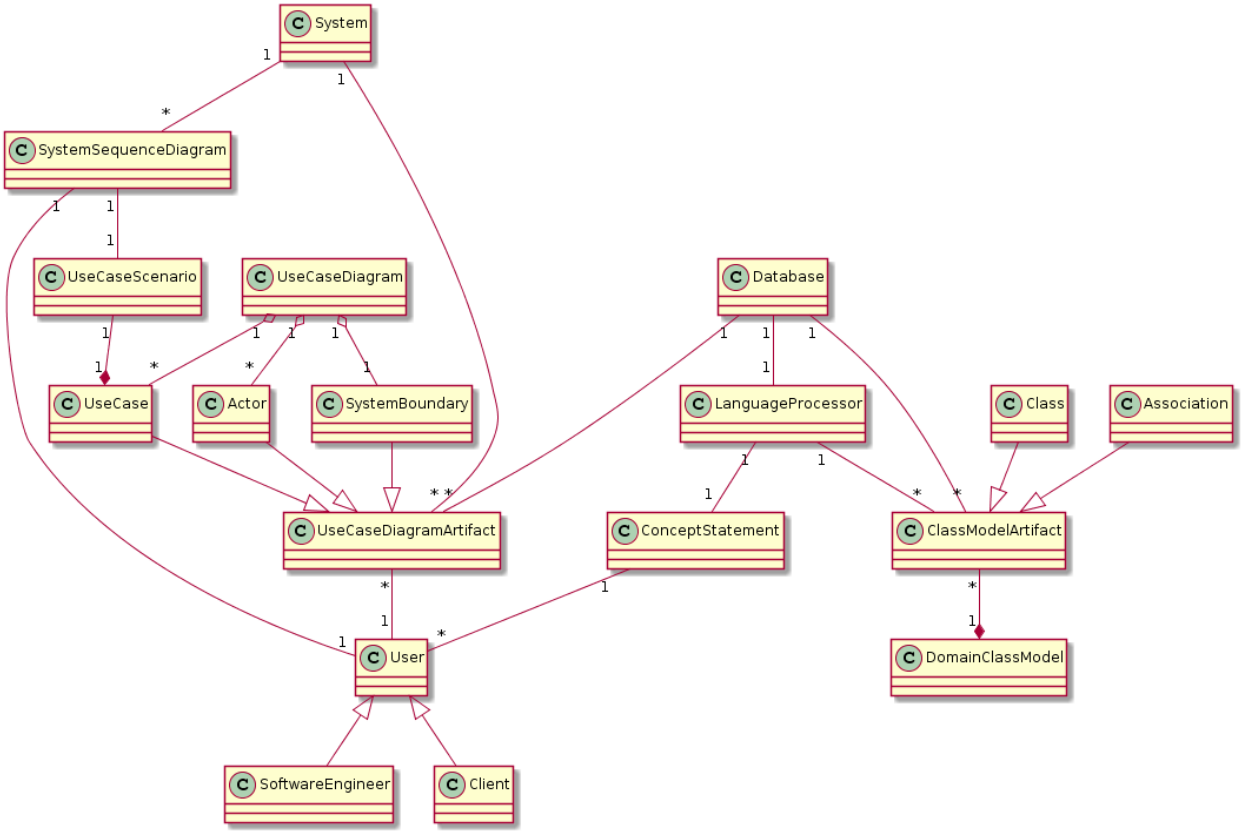

Language Parser Controller
request for parsing
Initialize
entry / instantiate NLP object
NLP object ready
Parse Nouns
do / parse concept statement for nouns
Nouns parsed
Wait for nouns
do / wait for user to select nouns
Nouns Submitted
by user
Parse Verbs
do / parse verbs associations for selected nouns
verbs parsed
Wait for user approval
do wait for user approval of verb associations
User approves
Store
do / Store nouns and verb associations in DB

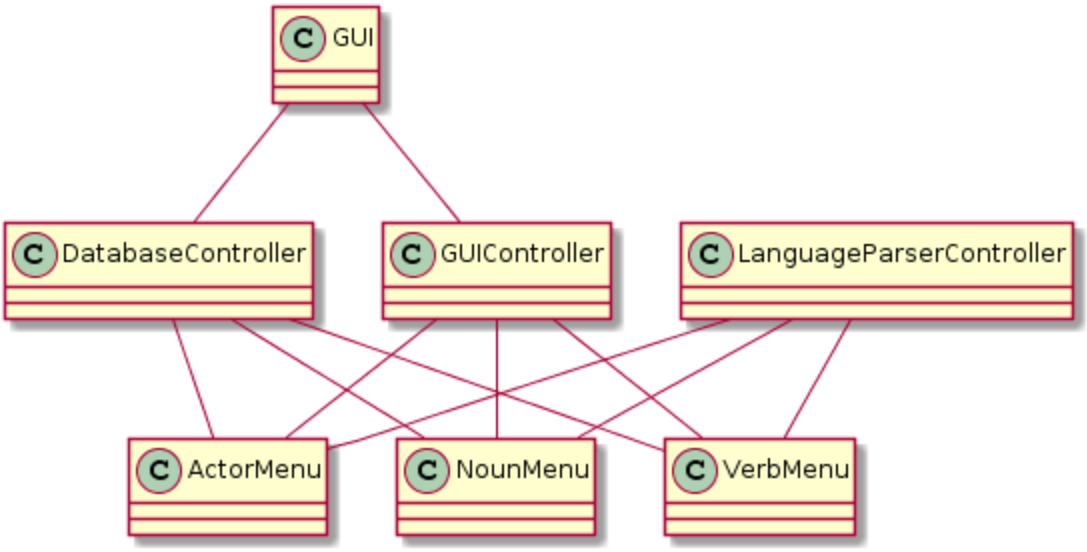
GUI Controller



3 - Consolidated Class Model

Consolidated Class Model





4 - Model Review

Model Review:

Analysis Model Correctness:

(1) Is the glossary of concepts and entity objects understandable by the user?

The concepts and entity objects should make sense in combination with all the diagrams provided in the Analysis Phase.

(2) Do abstract classes correspond to user-level concepts?

The abstract classes correspond to the most basic level entities in the project's domain scope.

(3) Are all descriptions in accordance with the user definitions?

Descriptions in the analysis diagrams correspond to the definitions detailed in the concept statement.

(4) Do all entity and boundary objects have meaningful noun phrases as names?

Entity and boundary objects do have names that are adequately descriptive, meaningful noun phrases for names.

(5) Do all use cases and controller objects have meaningful verb phrases as names?

Use cases and controller objects do have names that are adequately descriptive, meaningful verb phrases for names.

(6) Are most critical error cases described and handled?

All possible, critical error cases are identified in the corresponding use cases and their handling is described.

Analysis Model Completeness:

(1) For each class: Is it needed by any use case or the concept statement? In which use case is it defined, modified, created?

From the Domain Class Model, each class is represented in at least 1 use case. Also, each class is described in some way in the original concept statement.

(2) For each association: When is it traversed? Why was the specific multiplicity chosen?

From the analysis models, each association displayed is necessary and has a multiplicity that corresponds to its needs.

(3) For each attribute: type (optional)? Should it be a qualifier?

No attributes are provided in the models at this time. These have been avoided until a time where they are necessary for further information.

(4) For each controller object: Does it have the necessary associations to access the objects participating in its corresponding use case.

Each controller object is associated with all objects that it requires access to in its use cases.

Analysis Model Consistency:

(1) Are there multiple classes or use cases with the same name?

No classes or use cases have the same names.

(2) Do entities with similar names denote similar concepts?

Similar names were avoided; all names should be disparate enough to denote different concepts entirely.

(3) Are there classes with similar associations and attributes that are not in the same generalization hierarchy?

Classes that were similar enough in regards to associations were merged together, and now all classes should have enough distinct associations to require separate classes.

Inter-diagram Consistency:

All diagrams that share objects should access the entities and have associations that are consistent. All objects in the sequence diagrams should have corresponding classes in the consolidated class model.