**Component Design**

For a Track & Field Meet Server

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Tracy Marshall

CIS 895 – MSE Project

Kansas State University

# Table of Contents

[Table of Contents 2](#_Toc467683588)

[1. Introduction 3](#_Toc467683589)

[2. Architecture 3](#_Toc467683590)

[3. Component Design 4](#_Toc467683591)

[3.1. Component Diagram 4](#_Toc467683592)

[3.2. Interface Specification 4](#_Toc467683593)

[4. Class Design 7](#_Toc467683594)

[4.1. Class Diagram 7](#_Toc467683595)

[4.1.1. Communicator 7](#_Toc467683597)

[4.1.2. Database Manager 8](#_Toc467683598)

[4.1.3. Database 9](#_Toc467683599)

[5. Sequence Design 10](#_Toc467683600)

[5.1. Registering User Sequence Diagram 10](#_Toc467683601)

[5.2. Requesting Data Sequence Diagram 12](#_Toc467683602)

[5.3. Updating Data Sequence Diagram 14](#_Toc467683603)

[5.4. Adding Data Sequence Diagram 16](#_Toc467683604)

# Introduction

The purpose of this document is to provide the component design for the Track & Field Meet Server (TFMS). The document will take a look at the component level design details and the interface specifications that support those. The document will also dive into the class diagram and sequence diagram design of the project to give some depth in details to support the component design.

# Architecture

The goal of the TFMS was to maintain a simple design while being able to provide a great deal of flexibility, capability and future growth. To do this the system was separated into three main components. These components work together to provide a framework that could easily be built upon to grow the TFMS as others develop the product. There are three main functions that drive the system and lent themselves nicely to decomposing my system. The TFMS communicates with clients in the outside world to be able to take in requests about looking up data, adding data and manipulating data. The system clearly has data that must be stored and organized for recall and modification and that data must have some sort of control mechanism to maintain some sort of protections for the data.

# Component Design

The following section will be looking at the component design for the TFMS. It will break down each component and how they interact with one another. The interfaces between each component will be described in the interface specification section.

## Component Diagram

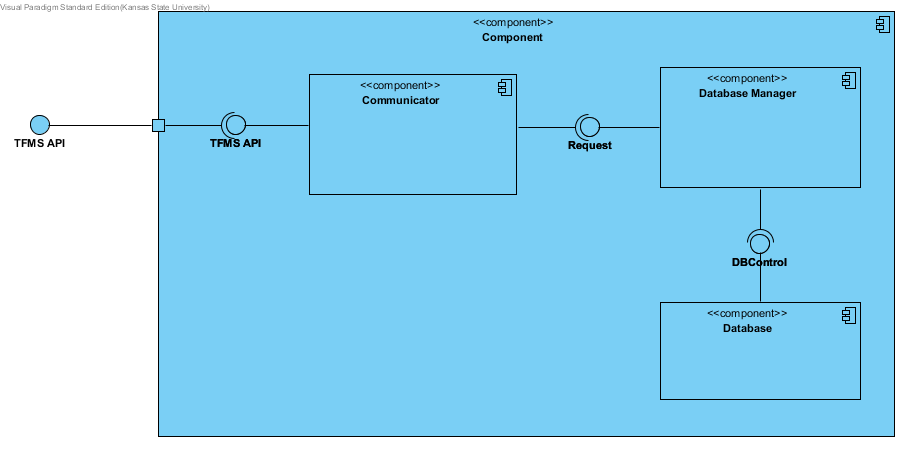


Figure 1 – Component Diagram

## Interface Specification

The following are the high level interfaces between the components of the component diagram. The external TFMS API interface will be defined in the user manual for client side development.

|  |  |
| --- | --- |
| **Communicator** |  |
| **Signature** | register( u: User) |
| **Purpose** | Registers a users account and establishes identity |
| **Pre-Conditions** | User is not registered in the system, User is in the system if not a spectator |
| **Post-Conditions** | User is registered, users public key is saved |
|  |  |
| **Signature** | requestData(u: User, d: data) |
| **Purpose** | Allows user the ability to request data that is stored in the server's database |
| **Pre-Conditions** | User must be registered |
| **Post-Conditions** | Server is queued for a response |
|  |  |
| **Signature** | addData(u: User, d: data) |
| **Purpose** | Allows users the capability to add data to the server given they have the correct permissions |
| **Pre-Conditions** | User must be registered |
| **Post-Conditions** | Server is queued for a response, data is added if the users permissions are correct |
|  |  |
| **Signature** | modifyData(u:User, d:data) |
| **Purpose** | Allows users the capability to modify data in the server given they have the correct permissions |
| **Pre-Conditions** | User must be registered |
| **Post-Conditions** | Server is queued for a response, data is modified if the users permissions are correct |
|  |  |
| **Signature** | response(u:user, d:data) |
| **Purpose** | Provides the users feedback on their requests or provides them with data depending on the nature of the request |
| **Pre-Conditions** | User must be resisted, a request must have been made |
| **Post-Conditions** | Response is removed from the server queue |

|  |  |
| --- | --- |
| **Request** |  |
| **Signature** | isRegistered(u:User): Boolean |
| **Purpose** | Checks if a given user is registered in the system |
| **Pre-Conditions** | Request is made by a user |
| **Post-Conditions** | State of registration is returned |
|  |  |
| **Signature** | isAutheticated(u:User): Boolean |
| **Purpose** | Authenticates the user to prove identity |
| **Pre-Conditions** | User is registered |
| **Post-Conditions** | State of authentication is returned |
|  |  |
| **Signature** | hasPermissions(u:User, d:Data): Boolean |
| **Purpose** | Checks if the given user is allowed to make a given request to the server |
| **Pre-Conditions** | User is registered and authenticated |
| **Post-Conditions** | State of permissions are returned |
|  |  |
| **Signature** | queryDB(u:User, d:Data): Data |
| **Purpose** | Accepts a request to query the DB |
| **Pre-Conditions** | User is registered, authenticated, and has permissions |
| **Post-Conditions** | Data that was being queried is returned |
|  |  |
| **Signature** | updateDB(u:User, d:Data) |
| **Purpose** | Accepts a request to update data in the DB |
| **Pre-Conditions** | User is registered, authenticated, and has permissions |
| **Post-Conditions** | DB is queued for an update of the given data |
|  |  |
| **Signature** | addToDB(u:User, d:Data) |
| **Purpose** | Accepts a request to add data to the DB |
| **Pre-Conditions** | User is registered, authenticated, and has permissions |
| **Post-Conditions** | DB is queued to add the new data |

|  |  |
| --- | --- |
| **DBControl** |  |
| **Signature** | queueUpdate(d:Data) |
| **Purpose** | Adds an update to the DB queue so that when the DB is in a safe state the udpate will be made |
| **Pre-Conditions** | Data exists |
| **Post-Conditions** | Upadate is queued |
|  |  |
| **Signature** | dataExists(d:Data): Boolean |
| **Purpose** | Allows the ability to check existence of given data |
| **Pre-Conditions** | DB has initialized |
| **Post-Conditions** | The state of existence of the data is returned |
|  |  |
| **Signature** | queueAdd(d:Data) |
| **Purpose** | Adds an add to the DB queue so that when the DB is in a safe state the data will be added |
| **Pre-Conditions** | DB has initialized |
| **Post-Conditions** | Add is queued |
|  |  |
| **Signature** | getData(d:Data):Data |
| **Purpose** | Grabs the data in the DB described by d |
| **Pre-Conditions** | Data exists |
| **Post-Conditions** | Data is returned |

# Class Design

The following section will take a quick look at the overall class diagram for the system. It will then take a closer look at each of these classes and look at how they break down.

## Class Diagram

## 

Figure 2 -Class Diagram

## Communicator

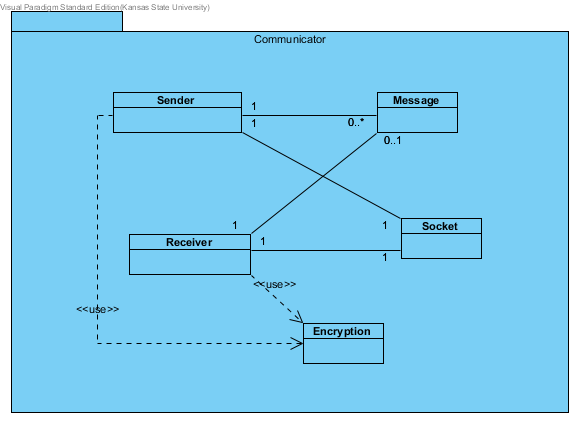


Figure 3- Communicator Package

The communicator package is responsible for communication with all of the clients of the server. This means that it has to be able to receive and send messages. To do this the package builds upon existing socket packages and utilizes encryption packages to do the symmetric and asymmetric encryption/decryption necessary to keep data secure.

## Database Manager

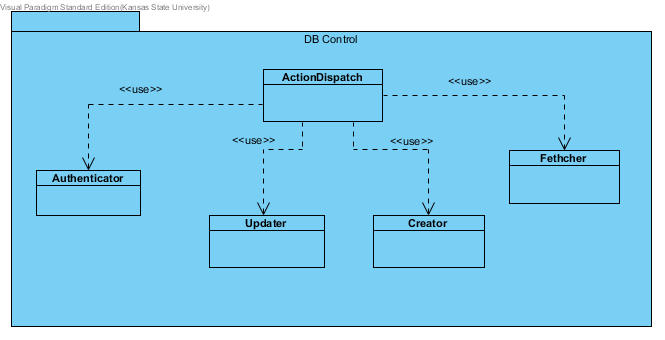


Figure 4- DB Control Package

The database manager package is responsible for being the intelligent one in the system. This package contains the interpreter of the ICD and is able to determine what certain requests from clients mean. On top of that the manager makes determinations about user’s privileges so that not just anyone can modify and add information to the server. The controller dispatches the requests from the client into queries to the database.

## Database

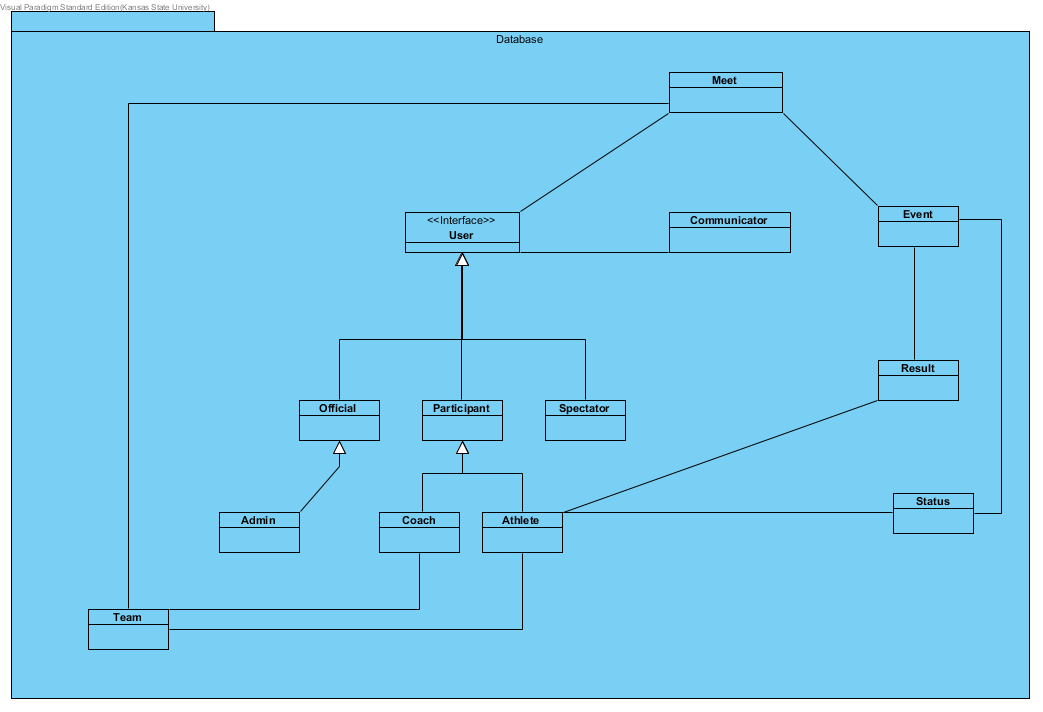


Figure 5- Database Package

The database package is the heart of the system and it contains all of the data that makes the server so important. This package is responsible for managing the actual data objects and providing interfaces to looking up data as well as adding and modifying data in the system.

# Sequence Design

## Registering User Sequence Diagram

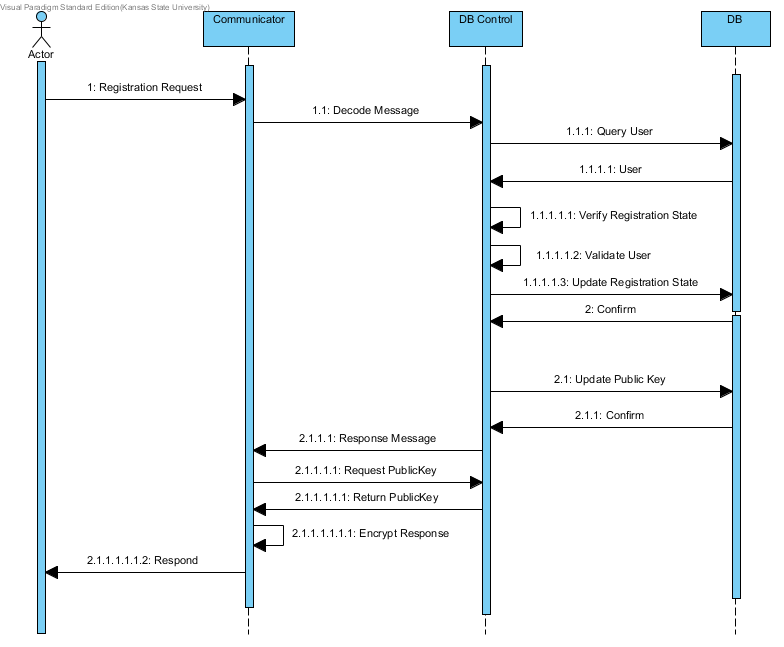


Figure 6- Register User Sequence Diagram

Prerequisites

1. Client has obtained the servers public key
2. Client has obtained user ID

Sequence of Events

1. Client sends a registration request
2. The message is sent to the control to be decoded
3. Database is queried for a matching user object
4. The user is returned to the controller
5. Registration state is verified for the user by checking the public key on record
6. Validate that the user credentials received match with the user credentials that the database knows about
7. Update the database with the registration state
8. Database confirms the update
9. Update the public key of the user
10. Database confirms the update
11. A response message is formed and given to the communicator
12. Users public key is requested
13. The Public Key is returned
14. The public key is used to encrypt the response
15. Response is sent back to the client

Post-Conditions

1. Client is registered

## Requesting Data Sequence Diagram

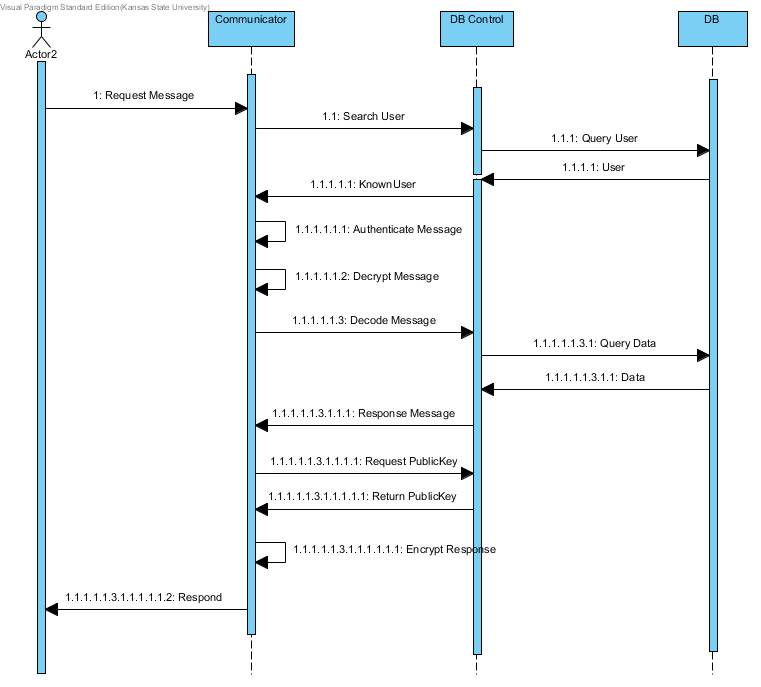


Figure 7- Request Data Sequence Diagram

Prerequisites

1. Client has registered

Sequence of Events

1. Client sends a request message
2. The user info is retrieve and sent to the controller to search the user
3. A database query is made for the user
4. The user object that matches the user data is returned
5. Known user status and data is returned to communicator
6. Message is authenticated
7. Message is decrypted
8. Message is sent to controller for decoding
9. Decoded message triggers a query to the database
10. Data is returned to the controller
11. A response message is formed and given to the communicator
12. Communicator request the users public key
13. The public key is returned
14. The response is encrypted using the users public key
15. The response is sent back to the client

Post-Conditions

1. Data requested by the client is returned in a response message

## Updating Data Sequence Diagram

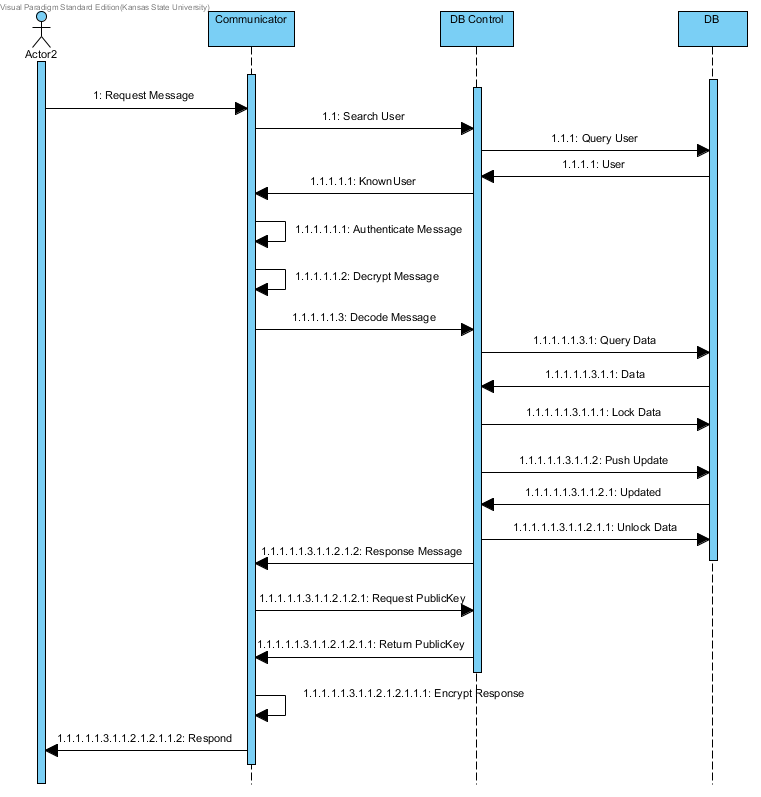


Figure 8- Update Data Sequence Diagram

Prerequisites

1. Client has registered

Sequence of Events

1. Client sends a request message
2. The user info is retrieve and sent to the controller to search the user
3. A database query is made for the user
4. The user object that matches the user data is returned
5. Known user status and data is returned to communicator
6. Message is authenticated
7. Message is decrypted
8. Message is sent to controller for decoding
9. Decoded message triggers a query to the database
10. Data is returned to the controller
11. Controller commands the database to lock the data that was requested
12. Controller sends updated data to the database
13. Database sends a data updated response
14. The controller unlocks the data
15. A response message is formed and given to the communicator
16. Communicator request the users public key
17. The public key is returned
18. The response is encrypted using the users public key
19. The response is sent back to the client

Post-Conditions

1. Data is updated in the database

## Adding Data Sequence Diagram

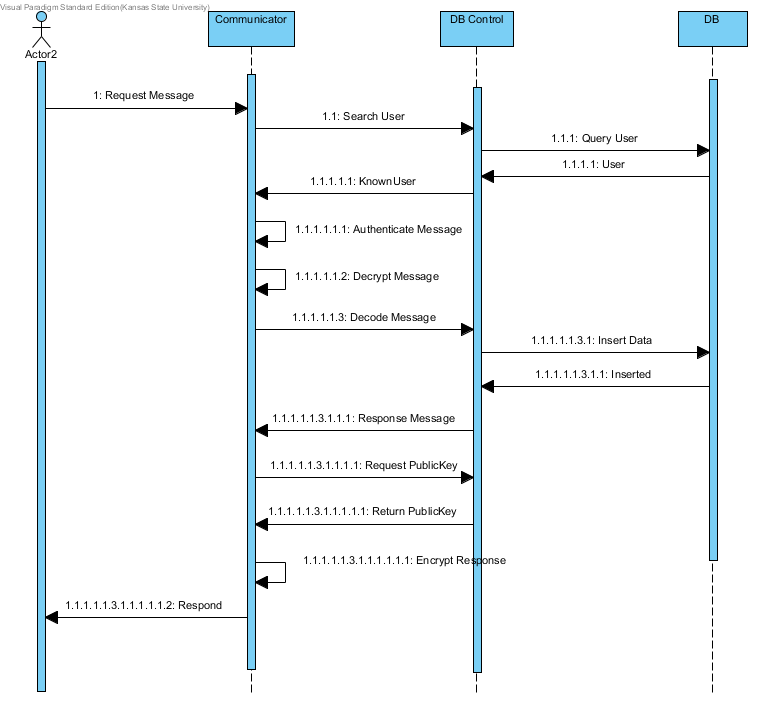


Figure 9- Add Data Sequence Diagram

Prerequisites

1. Client has registered

Sequence of Events

1. Client sends a request message
2. The user info is retrieve and sent to the controller to search the user
3. A database query is made for the user
4. The user object that matches the user data is returned
5. Known user status and data is returned to communicator
6. Message is authenticated
7. Message is decrypted
8. Message is sent to controller for decoding
9. Decoded message identifies new data and an insert is sent to the database
10. Database sends back an inserted resposne
11. A response message is formed and given to the communicator
12. Communicator request the users public key
13. The public key is returned
14. The response is encrypted using the users public key
15. The response is sent back to the client

Post-Conditions

1. Data is added to the database