# Requirements

## Scope

The scope of this document is to specify the requirements of OCRA Server application.

## Overview

OCRA Server is an application intended to offer authentication functionality to various clients requiring one time passwords. It implement the specifications described in RFC 6287 “OCRA: OATH Challenge-Response Algorithm”.

OCRA server acts as an authentication agent based on the challenge-response authentication scheme. The RFC 287 standardizes how the computations are performed and introduces sequences of messages exchanged. The integration of OCRA server is presented in Figure 1‑1 System components.

Figure 1‑1 System components



As presented in Figure 1‑1 System components OCRA Server is a software component that:

* Offers one time password functionality via a custom OCRA protocol to various OTP Clients; OTP Clients connect to OCRA Server and use the provided interface to exchange messages that implement the OCRA algorithm
* OCRA Server stores its data in a database server and acts as a client for the Database Server

## Requirements

### Functional Requirements

#### REQ-100 Platform integration

OCRA Server shall be developed as a Java standalone application.

#### REQ-200 Operating System

OCRA Server shall operate on all operating systems supporting Java platform 1.6 or above.

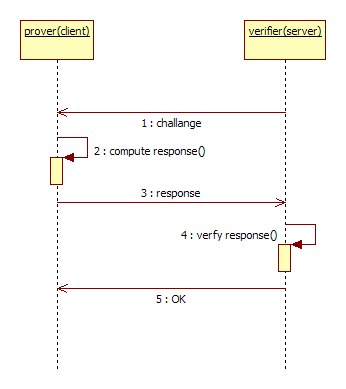
#### REQ-300 Authentication scenarios

OCRA Server shall support the authentication scenarios described in RFC 6287:

* One way challenge – response
* Mutual challenge – response
* Plain signature
* Signature with server authentication

#### REQ-310 Authentication scenarios – One Way Challenge Response

One way challenge – response scenario is described by the following flow

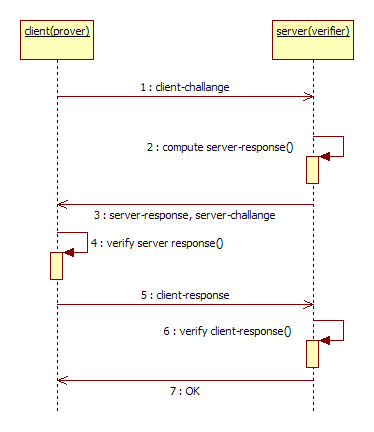


* + The verifier communicates a challenge value
  + The prover uses the challenge value to compute the response as described in RFC 6287
  + The prover communicates the response to the verifier to authenticate
  + The verifier checks the response and if the response is valid sends OK

#### REQ-320 Authentication scenarios – Mutual Challenge Response

Mutual challenge – response scenario is described by the following flow:

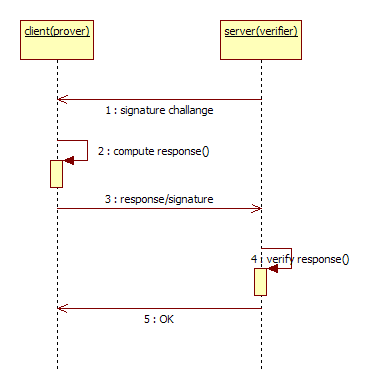
* + the client sends a random client- challenge to the server
  + The server computes the server-response
  + The server sends the server – response along with a server challenge
  + The client verifies the server response to be sure it is talking to the right server
  + The client computes a client – response
  + The client sends the client-response to the server
  + The server verifies the client-response
  + The server sends OK message to the client



#### REQ-330 Authentication scenarios – Plain Signature

Plain Signature scenario is described by the following flow

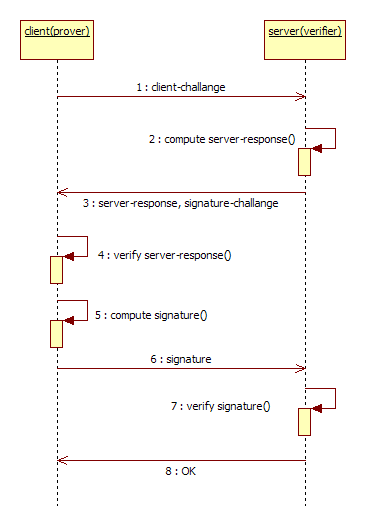
* + the server communicates a signature-challenge value to the client (signer)
  + client computes response
  + client sends response (signature) to the server
  + server authenticates



#### REQ-340 Authentication scenarios –Signature with Server Authentication

Signature with Server Authentication scenario is described by the following flow:

* + the client sends a random client - challenge to the server
  + server computes a server-response
  + server sends server-response along with the signature-challenge
  + the client first verifies the server-response
  + the client computes the signature
  + the client sends the signature to the server
  + the server authenticates



#### REQ-400 Data Storage

OCRA Server shall keep user data in a database server.

#### REQ-500 Data Access Extensibility

OCRA Server shall contain extensible components to ease adaptation to various other database servers.

#### REQ-600 Configuration

OCRA Server shall be configurable via an external file.

#### REQ-700 Multiple Simultaneous Dialogs

OCRA Server shall be able to handle multiple simultaneous dialogs with various clients.

# Design

Based on the requirements, the design describes the solution given to the problem.

A server application will be developed in order to implement the specified requirements.

In order to implement a working solutions several elements need to be clarified:

* in the RFC 6287, only the general steps are defined; there are no references regarding the underlying protocol based for communication; a protocol needs to be defined
* the use cases must be updated to match the concrete protocol specification
* state machines handling the dialog must be defined

Communication protocol considerations:

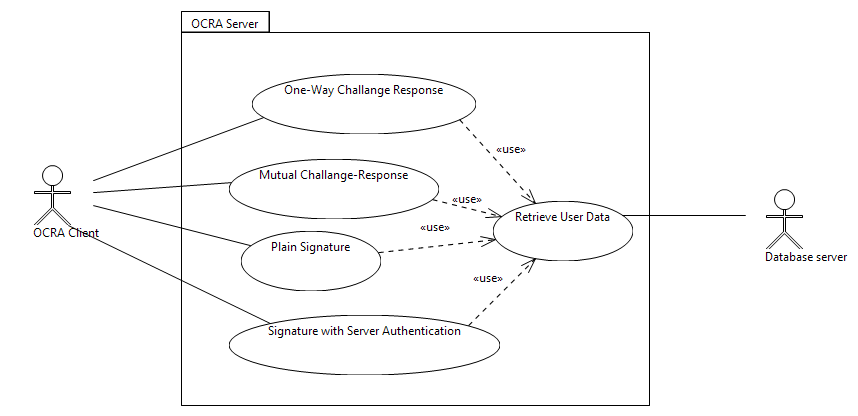
* over the socket, the data will be sent or received in plain text
* it may not be expected that only on key exists and it is shared between the server and an arbitrary number of clients; usually a key is shared between the client and the server for each client; in this case it is necessary that, in a first step, the client communicates its ID or username in order for the server to identify the corresponding key to use in the communication

It is assumed that the client and the server have a pre-shared key used for computation.

## Use cases

The following use case diagram shows the functionality that should be delivered by the server application. In the following the **one way challenge response** use case will be given details.

Figure 0‑1



### UC-100 One way challenge response

A challenge-response is a security mechanism in which the verifier presents a question (challenge) to the prover, who must provide a valid answer (response) to be authenticated.

In this case, OCRA Server authenticates a client using one way challenge response.

**Preconditions**:

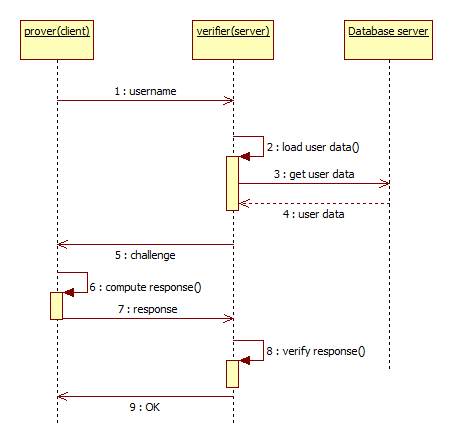
* OCRA Server is set up to operate in one way challenge response

#### UC-100 Main Flow

**Steps**:

* + The prover connects and informs verifier about its claimed identity (username)
  + Verifier asks for user data from the database server (this step is common to all use cases so it will be included in Retrieve User Data)
  + The database server responds with retrieved user data
  + The verifier communicates a challenge value based on the retrieved user data
  + The prover uses the challenge value to compute the response as described in RFC 6287
  + The prover communicates the response to the verifier to authenticate
  + The verifier checks the response and validation succeeds and the reply is OK

The sequence diagram for this use case is presented in the following sequence diagram

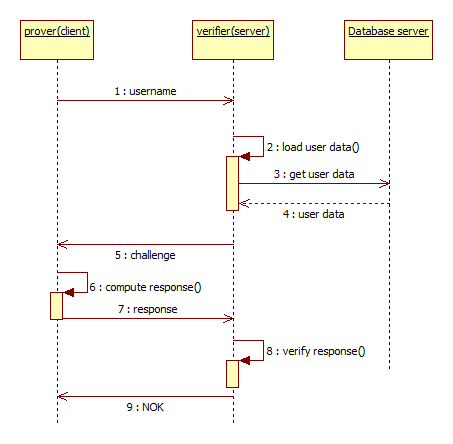


#### UC-100 Alternative Flow – verification fails

**Steps**:

* + The prover connects and informs verifier about its claimed identity (username)
  + Verifier asks for user data from the database server (this step is common to all use cases so it will be included in Retrieve User Data)
  + The database server responds with retrieved user data
  + The verifier communicates a challenge value based on the retrieved user data
  + The prover uses the challenge value to compute the response as described in RFC 6287
  + The prover communicates the response to the verifier to authenticate
  + The verifier checks the response and but validation fails therefore replies with NOK

The sequence diagram for this use case is presented in the following sequence diagram

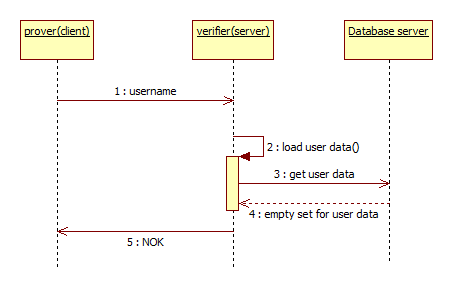


#### UC-100 Alternative Flow – User Data not found

**Steps**:

* + The prover connects and informs verifier about its claimed identity (username)
  + Verifier asks for user data from the database server (this step is common to all use cases so it will be included in Retrieve User Data)
  + The database server responds with retrieved user data(this step is common to all use cases so it will be included in Retrieve User Data)
  + User data is not found/empty set
  + The verifier checks the response and but validation fails therefore replies with NOK

The sequence diagram for this use case is presented in the following sequence diagram



#### UC-200 Mutual challenge response

**Not detailed**

#### UC-300 Plain signature

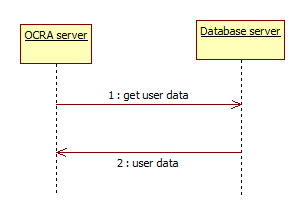
**Not detailed.**

#### UC-400 Signature with server authentication

**Not detailed.**

#### UC-500 Retrieve User Data

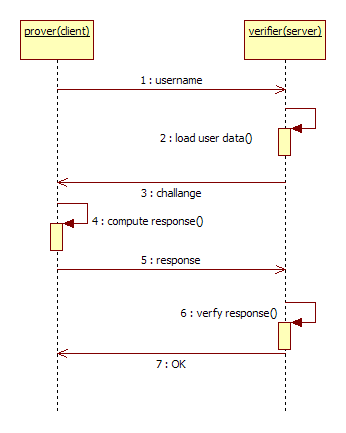
This is use case groups a common sequence of events included in UC-100, UC-200, UC-300, UC-400, therefore, to reduce redundancy, the group of messages is modeled as a separate use case included in the flow of other use cases.



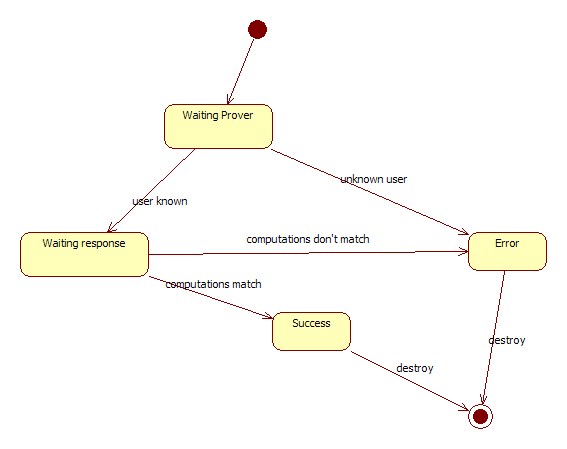
# Processing design

## State Machine

Taking into account the actual sequence of events for one way challenge response functionality, described in the following use case diagram, a dialog state machine will be designed.



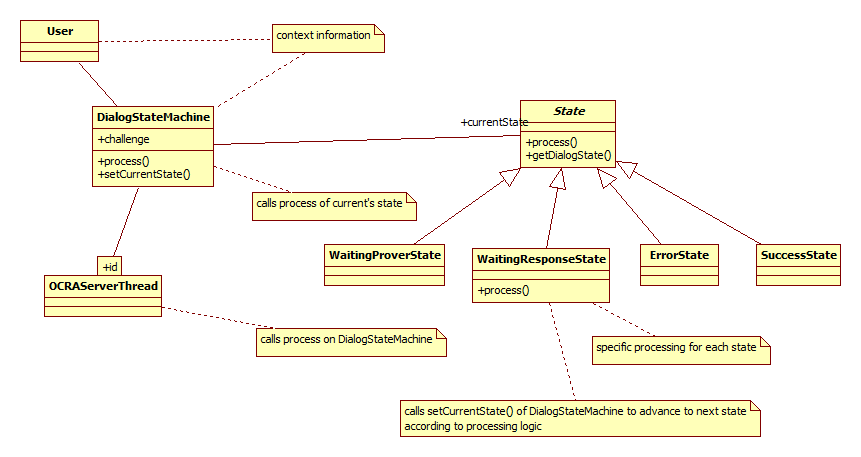
To handle this dialog, the corresponding state machine for the dialog is described in the following state diagram.



The behavior of the machine is as follows:

* Waiting Prover – in this state it is expected to receive as input, the username and the server will verify if it exists among the already provisioned users; the output is the challenge is sent to the prover
* Waiting Response – in this state it is expected as input a client-response; the match will be computed; the output of the processing is the message to be sent to the user, OK or NOK
* Error – in this state the dialog enters from the previous states when the user is unknown or the computations do not match
* Success – this state shows the dialog has completed
* After entering either Error or Success states, the dialog is expected to be closed.

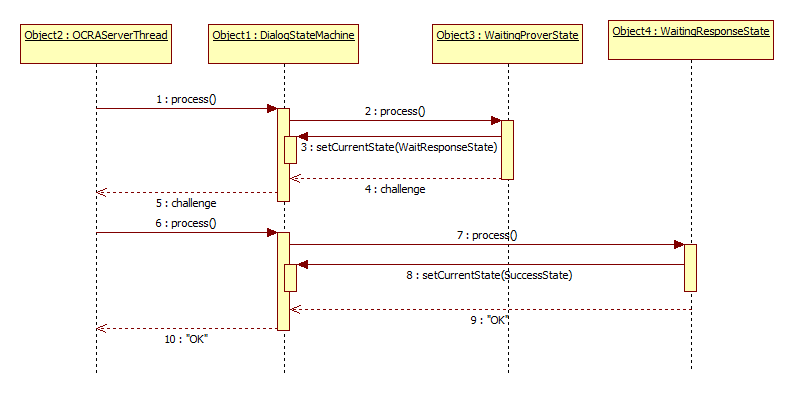
The state pattern used to implement this state machine is depicted in the following class diagram.



Classess:

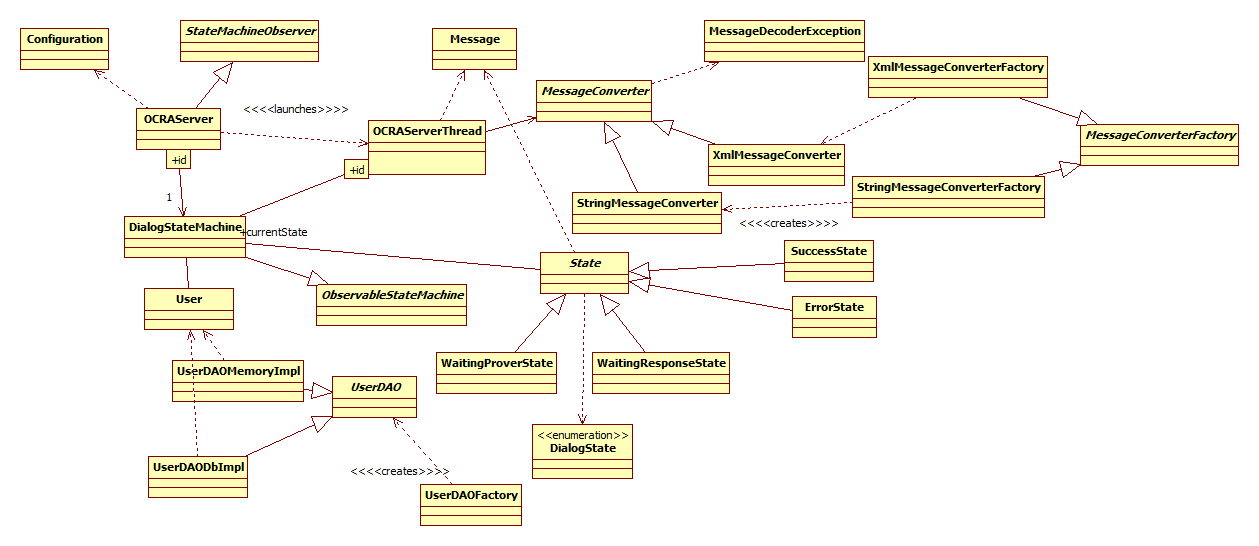
* OCRAServerThread and OCRAServer
  + Offers low level functionality for the dialog, e.g. sending/receiving data over the socket
  + Holds a map to enable multiple simultaneous dialogs
    - dialogId as key
    - DialogStateMachine object as value
  + Uses the Dialog State Machine for the current dialog identified by dialogId and sends it received data buffers for processing
  + the low level communication sends the data buffers created by the state machine
* Dialog State Machine
  + Represents the context of the state machine pattern
  + Processes the received data buffers in each state of the dialog
  + Holds a pointer to the current state and calls the appropriate process method that will trigger a processing in the context defined by the corresponding State class
* State
  + Base class for the states of the dialog
  + Offers an interface for processing the received data
* Concrete States: Waiting Prover, Waiting Response, Success, Error implement the concrete behavior of handling the incoming data
* User Data
  + Class holding user data, shared among the states
* Dialog State
  + Enumeration for identifying the state types by a value

The dynamic aspects for the state machine are included in the following diagram



## Application Class Overview

The overview class diagram for the application is presented in the following figure.



## Patterns used in the implementation

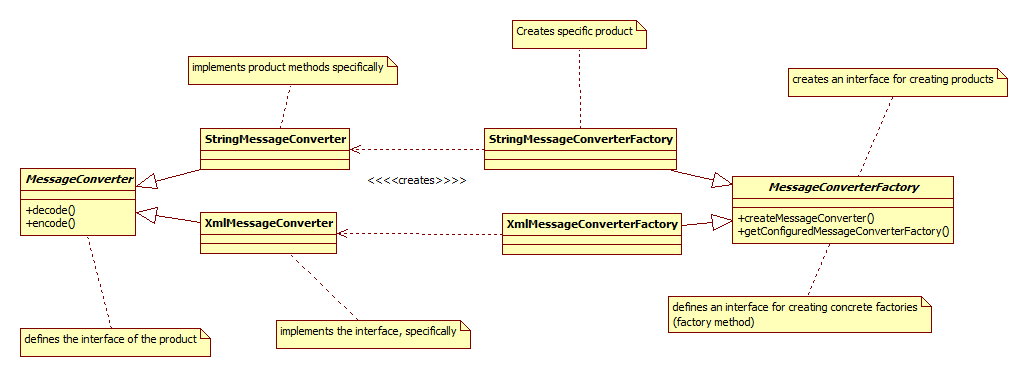
### State Machine

See chapter 3.1.

### Abstract Factory

For protocol extensibility, an abstract factory creates different decoders for incoming messages.

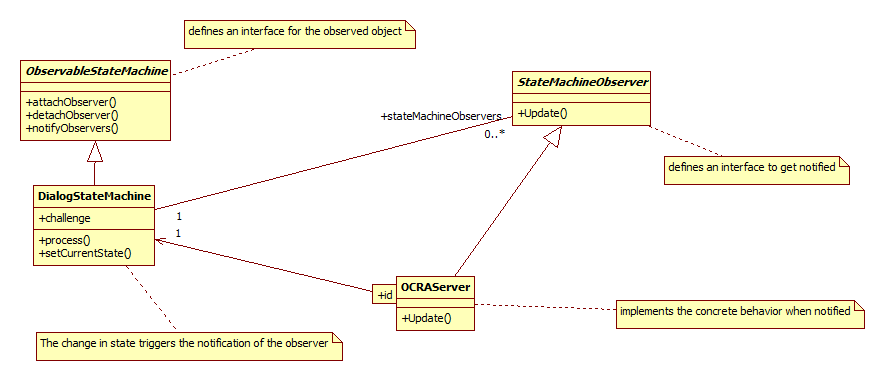
Static aspects:



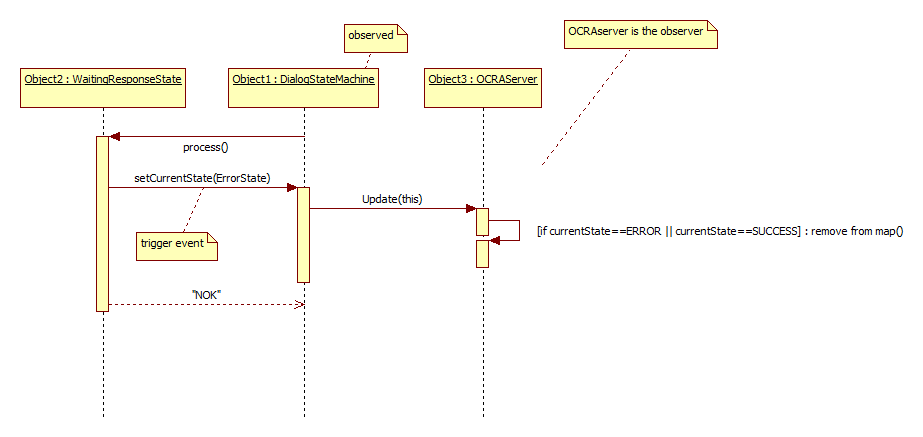
### Observer pattern

State Machines are observed in order to detect dialog finishing.

Static aspects:

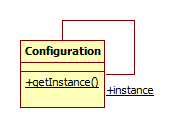


Dynamic aspects:



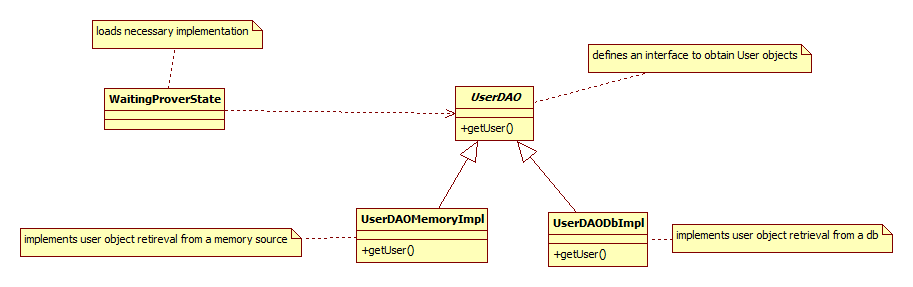
### Singleton

Configuration is an object intended to exist as a single instance.



### Strategy

Implements the required extensibility mechanism for data access. It is easy to implement access for new data providers. Initially there are two, one in-memory data provider and a dummy database provider.



### Factory method

Easily customizable method that creates a default User DAO.

