1. [15 points] Design a variability model corresponding to the following description by Hundt, Mehner, Pfeiffer and Sokenou (2007).

"The security component is intended to provide access control by user authentication and authorization. A user management stores login names and passwords.

Authentication is the minimal function required for access control. If only authentication is requested, the authenticated user has complete access to the application. The minimal requirement for authentication is the login feature. Login is carried out either through a user interface or through operating system user identification. An optional logout feature supports to explicitly log out from the system. After logout, a new user or the same must be able to re-login. Optionally, a timeout is available to logout a user automatically after a configurable time span has elapsed without user interaction. Logout and timeout are not supported if login is realized by operating system user identification.

Authorization is dependent on authentication. For authorization, it is assumed that access restrictions for each user can be specified. Authorization is supposed to cover business objects containing data (per value) but also work flows (per work flow) or certain operations (per operation). Different access rights and different user roles are distinguished."

Solution (inspired by Hundt, Mehner, Pfeiffer and Sokenou (2007)).
2. [15 points] Suppose we are developing a non-web-based information system. This software will be used to collect and process data about certain entities. The requirements on the user interface are not completely clear. We would like to minimize the risks regarding these requirements by splitting each interface into frames. We have two types of interfaces:

- data-entry interfaces: consist of two frames. The main frame displays existing data items in a data-grid. By selecting an item from this data-grid, the second frame will display a detailed view of the item and will allow the user to edit/remove the item. After making a change in the second frame, this change should be reflected immediately in the main frame.

- information display interfaces: consist of several frames. Each frame offers a different representation for the same data items, e.g., spreadsheet, pie chart, bar chart.

Which architectural patterns/styles would be useful in this situation?

What components of the system would you associate with each part of the chosen architectural patterns/styles?

How would you address the problems in this description by applying these patterns/styles?

Solution:

The general structure of the architecture will be based on MVC. For certain parts of the system applying Publisher/Subscriber would be useful. We are developing an information system that allows a high degree of interaction with the user. The requirements on the UI are not stable whereas the requirements related to the core functionality of the system will remain stable. We can:

- analyze the application domain and identify the data entities and the core functionality of the system (model in MVC);
- develop a view for each frame and compose the views to build the interfaces;
- establish the required communications between the views via controllers.

In particular controllers should synchronize:

- main and detailed views in data-entry interfaces and
- different views in an information display interface.

We use the Publisher/Subscriber pattern to establish this goal. Each view subscribes to be notified about changes applied to its underlying data source (publisher). After changing a data item the publisher will notify the subscribers about the change.
3. [15 points] Consider the following ATM system.

The ATM system allows clients to work with bank accounts. To use the ATM a client has to perform authentication. After authentication has been performed, the client gets access to the accounts that he/she is assigned to. The client can perform the following operations with his/her accounts:

- withdraw money from his/her account;
- transfer money from his/her account to any other account;
- deposit money in his/her account;
- stop working with the ATM and with his/her accounts.

The balance variable defines the amount of money stored on accounts. There are two types of accounts: debit and credit. Balance of a debit account must be greater than or equal to zero. Balance of a credit account can be negative, but must not exceed a predefined maximum credit sum.

Money is stored on accounts in different currency. There are four types of currency available: Dollar, Euro, Rupee, and Yuan. The exchange rate defines the rate at which one currency will be exchanged for another when withdrawing, transferring, or depositing money of different currency.

Compose an Event-B specification of this ATM system. Use the following Event-B specification as an initial version (here the specification is not split into the context and the machine for the sake of brevity):

**SETS**

- Clients
- Accounts
- AccountType

**CONSTANTS**

- Debit
- Credit
- MaxCredit

**AXIOMS**

\[
\begin{align*}
axm1 & : \text{partition(AccountType, \{Debit\}, \{Credit\})} \\
axm2 & : \text{MaxCredit} = 1000
\end{align*}
\]

**VARIABLES**

- authorized
- atm_user // the authorized client of the ATM
- clients2accounts // assigning accounts to clients
- balance
- access // accounts that the atm_user can access

**INVARINTS**

\[
\begin{align*}
\text{inv1} & : \text{clients2accounts} \in \text{Clients} \leftrightarrow \text{Accounts} \\
\text{inv2} & : \text{authorized} \in \text{BOOL} \\
\text{inv3} & : \text{atm_user} \in \text{Clients} \\
\text{inv4} & : \text{access} \in \text{Accounts} \\
\text{inv5} & : \text{authorized} = \text{FALSE} \implies \text{access} = \emptyset \\
\text{inv6} & : \text{authorized} = \text{TRUE} \implies \text{access} = \{x \cdot \text{atm_user} \mapsto x \in \text{clients2accounts} \mid x\} \\
\text{inv7} & : \text{balance} \in \text{Accounts} \rightarrow \mathbb{Z}
\end{align*}
\]

**EVENTS**

- Authentication
  - ANY client
WHERE ...
THEN ...
END

Transfer
  ANY source, destination, amount
WHERE ...
THEN ...
END

Deposit
  ANY destination, amount
WHERE ...
THEN ...
END

Withdraw
  ANY source, amount
WHERE ...
THEN ...
END

Stop
  WHERE ...
THEN ...
END

END

Add guards (WHERE-section) and actions (THEN-section) to the listed events to specify the ATM functionality. Specify types of the proposed parameters in the guards of the events. Ensure that the invariants hold after each of these events is executed. Add necessary variables, invariants and guards to describe debit/credit types of accounts. Add necessary sets, constants, axioms, variables, invariants, parameters and guards to describe currencies and exchange of currencies.

Solution:

SETS
  Clients
  Accounts
  AccountType
  Currency

CONSTANTS
  Debit
  Credit
  Euro
  Dollar
  Yuan
  Rupee
  MaxCredit

AXIOMS
  axm1 :  partition(AccountType, {Debit}, {Credit})
  axm2 :  MaxCredit = 1000
  axm3 :  partition(Currency, {Euro}, {Dollar}, {Yuan}, {Rupee})
VARIABLES

atm_user
authorized
balance
accounts
access
accountTypes
exchangeRate
accountCurrency

ININVARIANTS

inv7 : access ∈ Clients → Accounts
inv1 : atm_user ∈ Clients
inv2 : authorized ∈ BOOL
inv3 : balance ∈ Accounts → N
inv5 : accounts ∈ Accounts
inv4 : authorized = FALSE → accounts = ø
inv6 : authorized = TRUE → accounts = {x : atm_user ⊃ x ∈ access | x}
inv8 : accountTypes ∈ Accounts → AccountType
inv9 : exchangeRate ∈ (Currency × Currency) → N1
inv10 : accountCurrency ∈ Accounts → Currency

EVENTS

INITIALISATION

BEGIN

act5 : access ∈ Clients × Accounts
act1 : atm_user ∈ Clients
act2 : accounts = ø
act3 : balance = Accounts × {100}
act4 : authorized = FALSE
act6 : accountTypes = Accounts × {Credit}
act7 : exchangeRate = (Currency × Currency) × {10}
act8 : accountCurrency = Accounts × {Euro}

END

Authentication

BEGIN

client

WHERE

grd1 : client ∈ Clients
grd2 : authorized = FALSE

THEN

act3 : authorized = TRUE
act1 : atm_user = client
act2 : accounts = access[{client}]

END
Transfer

\[ \text{STATUS ordinary} \]
\[ \text{ANY} \]
\[ \text{source} \]
\[ \text{destination} \]
\[ \text{amount} \]
\[ \text{WHERE} \]
\[ \text{grd1} : \text{authorized} = \text{TRUE} \]
\[ \text{grd2} : \text{source} \in \text{accounts} \]
\[ \text{grd3} : \text{destination} \in \text{Accounts} \]
\[ \text{grd4} : \text{amount} \leq \text{N} \]
\[ \text{grd5} : \text{accountTypes(source)} = \text{Credit} \implies \text{amount} \leq \text{balance(source)} + \text{MaxCredit} \]
\[ \text{grd7} : \text{accountTypes(source)} = \text{Debit} \implies \text{amount} \leq \text{balance(source)} \]
\[ \text{grd6} : \text{source} \neq \text{destination} \]
\[ \text{THEN} \]
\[ \text{act1} : \text{balance} = \text{balance} \leftarrow \text{balance(source)} - \text{amount,} \]
\[ \text{destination} \rightarrow \text{balance(destination)} + \text{amount} \]
\[ \text{END} \]

Deposit

\[ \text{STATUS ordinary} \]
\[ \text{ANY} \]
\[ \text{destination} \]
\[ \text{amount} \]
\[ \text{WHERE} \]
\[ \text{grd3} : \text{authorized} = \text{TRUE} \]
\[ \text{grd1} : \text{amount} \leq \text{N} \]
\[ \text{grd2} : \text{destination} \in \text{accounts} \]
\[ \text{THEN} \]
\[ \text{act1} : \text{balance(destination)} \leftarrow \text{balance(destination)} + \text{amount} \]
\[ \text{END} \]

Withdraw

\[ \text{STATUS ordinary} \]
\[ \text{ANY} \]
\[ \text{source} \]
\[ \text{amount} \]
\[ \text{currency} \]
\[ \text{result} \]
\[ \text{WHERE} \]
\[ \text{grd4} : \text{authorized} = \text{TRUE} \]
\[ \text{grd1} : \text{source} \in \text{accounts} \]
\[ \text{grd2} : \text{amount} \leq \text{N} \]
\[ \text{grd3} : \text{accountTypes(source)} = \text{Credit} \implies \text{amount} \leq \text{balance(source)} + \text{MaxCredit} \]
\[ \text{grd5} : \text{accountTypes(source)} = \text{Debit} \implies \text{amount} \leq \text{balance(source)} \]
\[ \text{grd7} : \text{currency} \in \text{Currency} \]
\[ \text{grd5} : \text{result} = \text{amount} + \text{exchangeRate(accountCurrency(source)} \rightarrow \text{currency}) \]
\[ \text{result} = \text{exchangeRate(currency) \rightarrow accountCurrency(source))} \]
\[ \text{THEN} \]
\[ \text{act1} : \text{balance(source)} = \text{balance(source)} - \text{result} \]
\[ \text{END} \]

Stop

\[ \text{STATUS ordinary} \]
\[ \text{WHEN} \]
\[ \text{grd1} : \text{authorized} = \text{TRUE} \]
\[ \text{THEN} \]
\[ \text{act1} : \text{authorized} \leftarrow \text{FALSE} \]
\[ \text{act2} : \text{accounts} \leftarrow \emptyset \]
\[ \text{END} \]

END
4. [15 points] Give a class diagram that models the domain of the card game Klondike. Give only classes, associations, and multiplicities. No attributes or methods required.

Klondike is a solitaire game where part of the cards are initially placed on the table in a harp shape, the so-called tableau (see picture). The rest of the cards are placed face down in the deck pile and one by one turned over and either added to the tableau or to the discard pile. Cards have a rank and a suit. During the game, four suit piles are built (in the top right of the picture), one for each suit.

Solution:
5. [15 points] Consider the following required functionality of an online DVD shop:

Customers should be able to search for DVDs by Title and Category (some examples of categories are: Series, Movies, Music). Once they find a product they like, they can add a DVD to their shopping cart. This requires that they are logged in, therefore if they do not yet have an account, they should be able to create one. When creating an account, a username and password should be chosen, and optionally, they can enter their home address and/or the credit card information they want to use for paying. Credit card information consists of the name on the credit card, the credit card company, the credit card number, and the expiration date.

They can add as many DVDs to the shopping cart as they want, and they can also remove DVDs. If they decide at some point to check out the shopping cart, then they are asked to check the shipping address and credit card information supplied, and if either of those was not supplied, they are asked to supply it. Next, they can review the contents of the shopping cart, and if they agree, they can send the order. Then, the credit card information is sent to the bank, and, if approved, the order is finalised. If the information was not approved, then the customer is made aware of this, so he/ she can change the payment option.

A delivery agent can check the status of online orders, and process an order, meaning that it changes its status from “pending” to “delivered”.

Finally, the administrator is responsible for the DVD inventory. She can check the current inventory, order new DVDs from the supplier, and add DVDs to the inventory.

a. Create a UML Use Case Diagram for the online DVD shop. In addition, give a detailed description (pre-condition, trigger, guarantee, main scenario,…) of the use case for “check out shopping cart”.

b. Based on your use case description of “check out shopping cart”, create a UML Sequence Diagram.

c. Create a UML Activity Diagram to depict the business process for processing a DVD order. There are three parties involved in processing an order: Shipping, Online Sales, and Accounting. The process starts when Online Sales receives an order for DVDs from a user. To complete the order, the store needs to charge the credit card and deliver the DVDs. To charge the card, Online Sales sends the credit card information to Accounting, who will then validate and process the credit card. To deliver the DVDs, Shipping will first fill the order, then prepare the package, and finally deliver it. Once the DVDs are delivered and the credit card is charged, the order is closed.
(Some possible) answers

a.

Check out Shopping Cart

*Pre:* User is logged in

*Trigger:* User wants to check out

*Guarantee:* Order finalised

*Main:*

a. Start checkout
b. Check shipping address (SA) and credit card (CC) information
c. Review contents of cart
d. Send the order
e. CC is sent to bank
f. Order is finalised

*Alternatives:*

b-1. If either SA or CC is not supplied, supply it.
f-1. If CC is not approved, choose a different payment option. Retry step e.