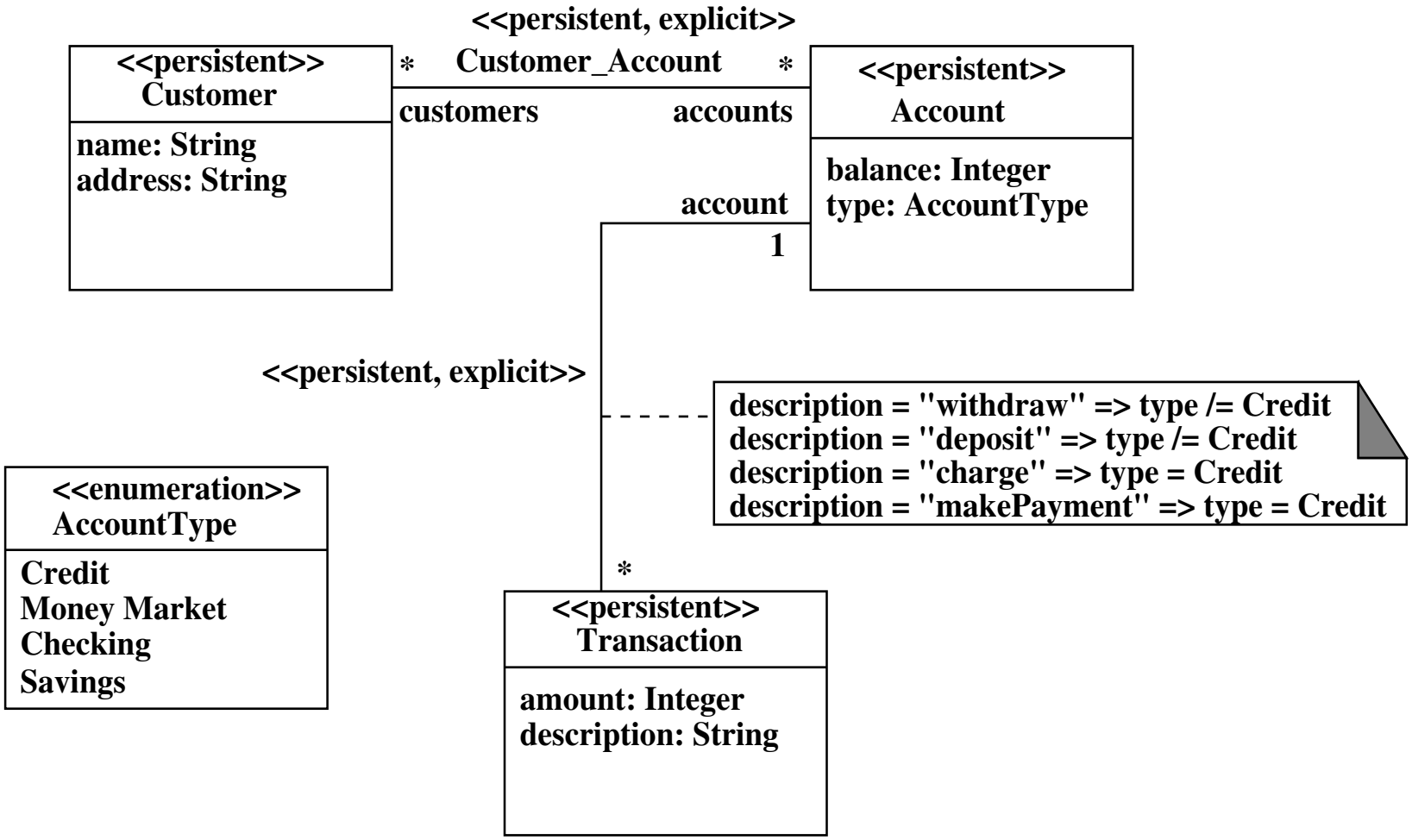
Online Bank System

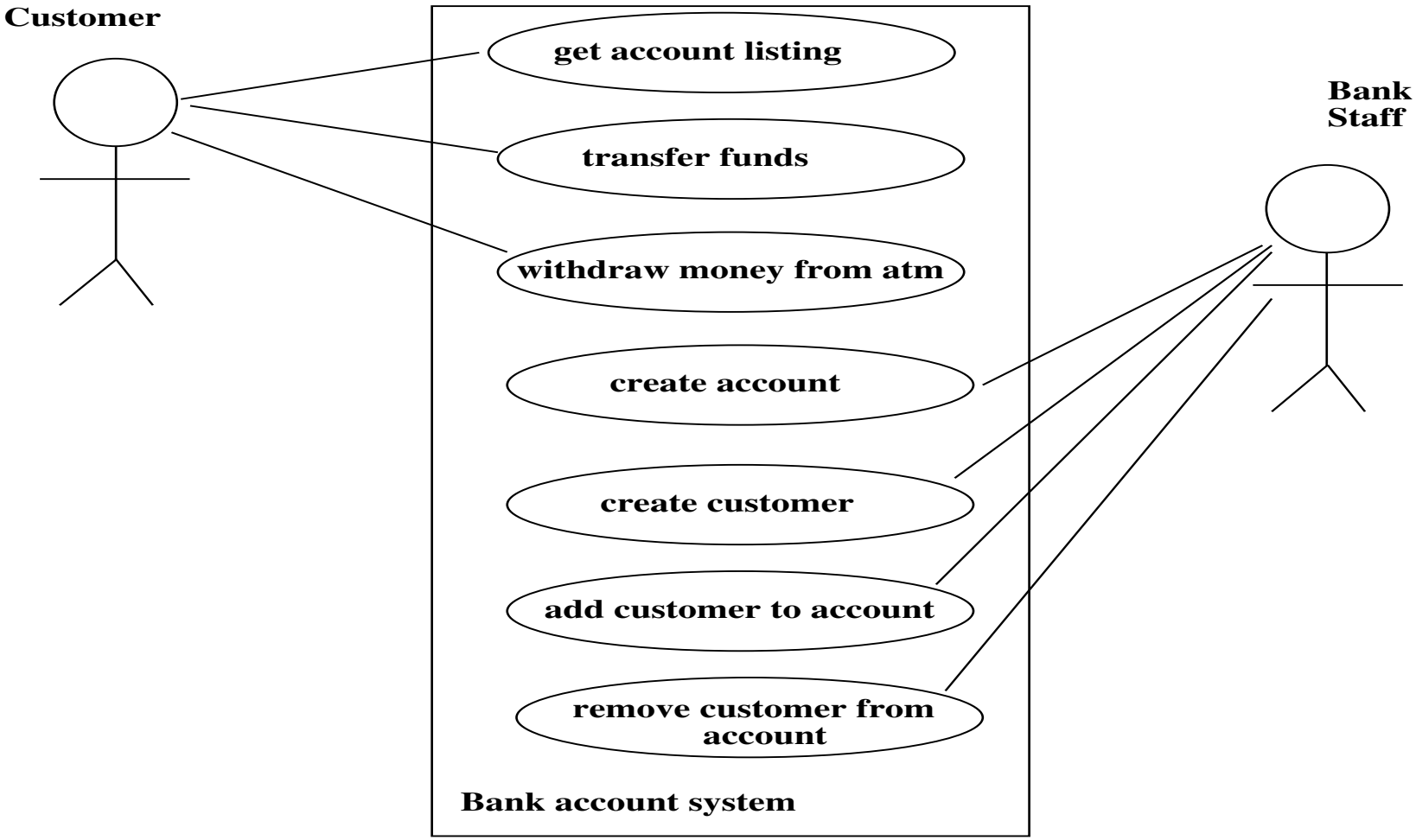
This is basic online banking system, with entities *Account*, *Customer* and *Tx* (transactions). Constraints of system are placed on *Account*-*Tx* association. Will be enforced by *TxControllerBean* session bean.

System has web interface for customers to view their accounts, and do transfers, and non-web interface for bank staff to create accounts and customers and to add or remove customers from accounts.

Constraint between *Tx* and *Account* is managed by *TxControllerBean* session bean, which only permits transactions to be processed if they satisfy the constraint.



PIM class diagram of account system



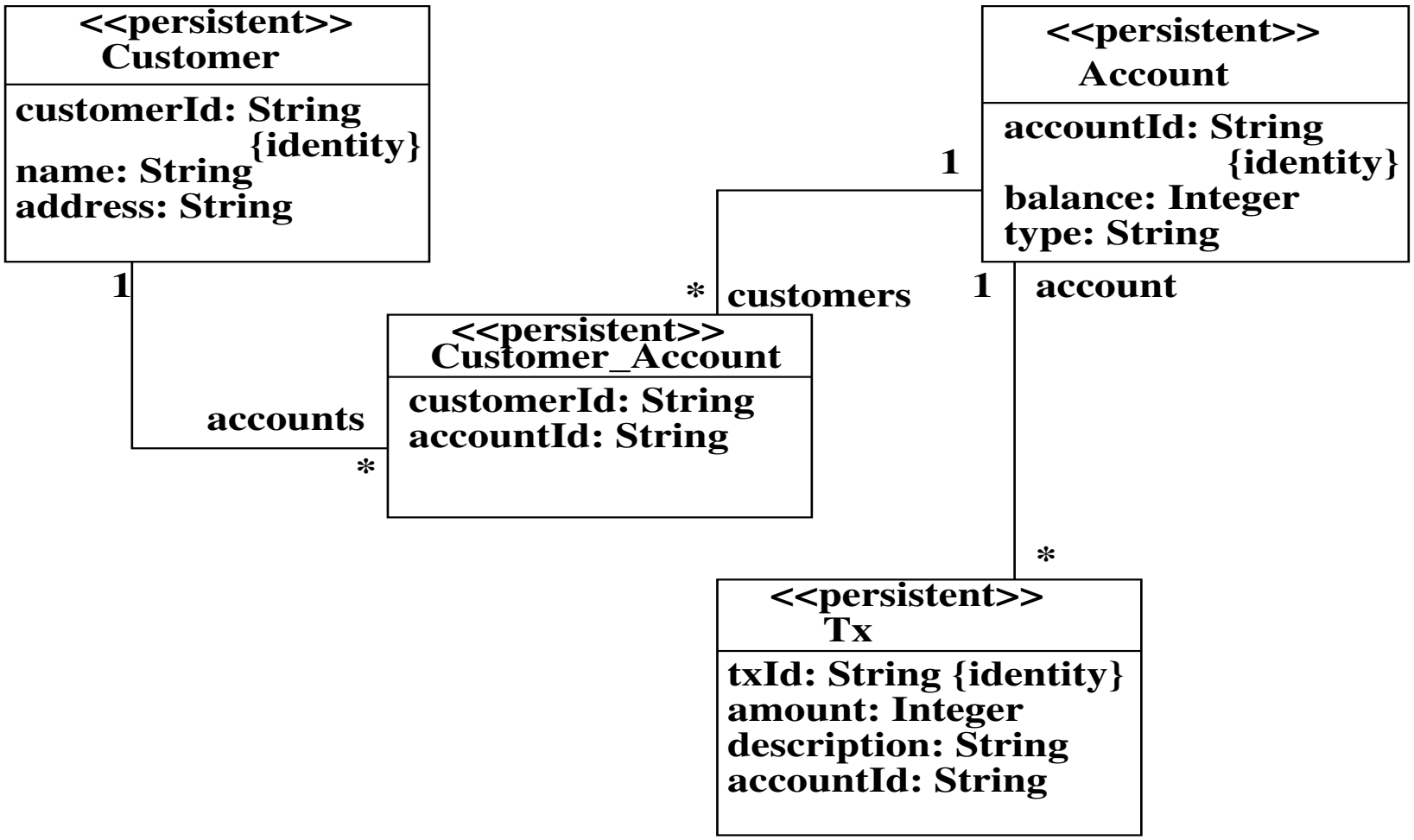
Use cases of account system

Design of account system

We need to take the following steps:

- Transform the PIM class diagram to a class diagram for a relational data model implementation.
- Identify components and architecture of the system.

The basic idea of the architecture is to use session beans to implement the use cases (cf. the Session Facade pattern), operating on entity beans for each entity.



PSM class diagram of account system

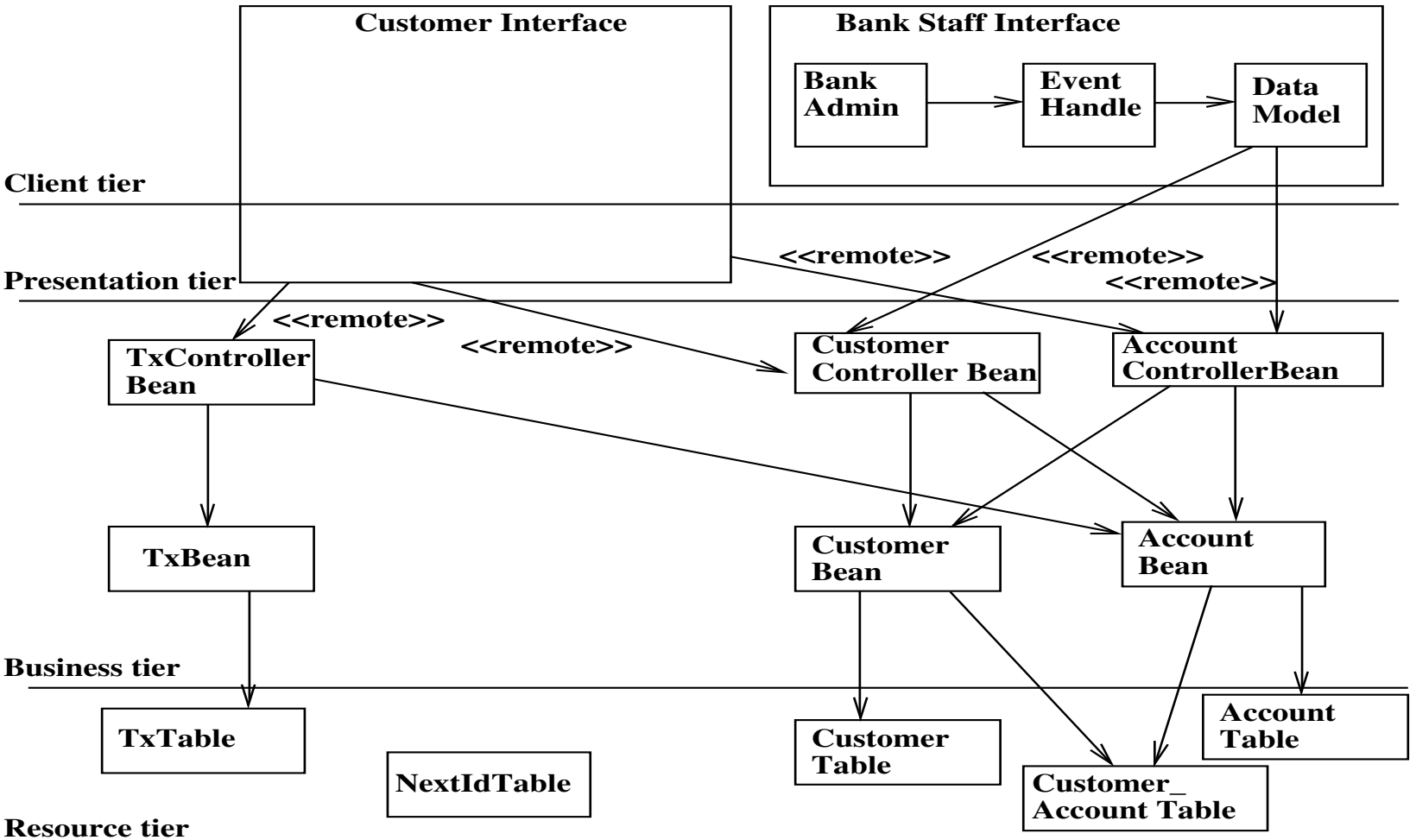
Components of account system

Introduce *AccountDetails*, *CustomerDetails* and *TxDetails* value object classes to transfer entity data.

Interface for bank staff will be a Swing GUI, sending commands to the session beans *AccountController*, etc.

Because *Customer*, *Account* are targets of associations (on the 'one' side of an association in design data model), their entity beans must be accessed locally, not remotely.

Current maximum id used in each entity table is stored in a *NextId* table. This is used to assign new (unused) ids for new instances of *Customer*, *Account* and *Transaction*.



Architecture of account system

Architecture validity

The session beans are *TxControllerBean*, *CustomerControllerBean*, *AccountControllerBean*.

Although there is shared write access in this example, by the session beans on *AccountBean*, the updates by *AccountControllerBean* and *CustomerControllerBean* on *AccountBean* cannot affect the constraints linking *Tx* and *Account*, so this architecture is valid.

Business tier components of account system

The session beans are:

- *AccountControllerBean*, implementing the *createAccount*, *addCustomerToAccount* and *removeCustomerFromAccount* use cases.
- *TxControllerBean*, implementing the *getAccountListing*, *transferFunds* and *withdrawMoney* use cases.
- *CustomerControllerBean*, implementing the *createCustomer* use case.

These components encapsulate the use cases of the system, as operations which make use of the entity beans. In particular all use cases for the Customer actor are implemented by *TxControllerBean*, and those for the Bank Staff are implemented by the other session beans.

Remote access is used for these beans because they may be used by presentation tier components on remote computers (eg, the web interface elements, which may reside on dedicated computers, separate to computers running business tier).

The entity beans are:

- *AccountBean*.
- *TxBean*.
- *CustomerBean*.
- *NextIdBean*.

Local access is used for these components, because they represent entities within the same database, connected by reference relationships. Such navigation between data would be very inefficient if carried out by remote method calls.

In addition there are auxiliary helper classes:

- *AccountDetails*, *CustomerDetails*, *TxDetails* value objects for the entities.
- *DBHelper* – used to generate next primary key values.
- *DomainUtil* – holds information about allowed types of account.
- *EJBGetter* – encapsulates bean lookup methods (cf, Service Locator pattern).

Data is passed between presentation and business tiers as $*Details$ objects, which hide details of the entity beans from higher tiers. They are examples of *value objects*.

An alternative architecture would combine the account and customer session beans into a single *StaffOperations* session bean. This would reduce the number of dependencies in the system, but decrease modularity.