The Account Analysis Pattern

Eduardo B. Fernandez and Ying Liu
Dept. of Computer Science and Engineering,
Florida Atlantic University
Boca Raton, FL 33431

Intent

The Account pattern keeps track of accounts of customers in institutions. These customers can perform transactions of different types against the accounts.

Example

Consider a banking institution, where customers have accounts of different types, e.g., checking, savings, loan, mortgage, etc. Customers use these accounts to handle their financial transactions, e.g., they sign checks for payments, they deposit money in their accounts, etc. Customers receive bank cards to facilitate their transactions. For the convenience of their customers, the bank may have several branches or offices located in different places. Usually, an account is in charge of a specific branch.

Context

There are many institutions, e.g., banks, libraries, clubs, and others, that need to provide their customers or members with convenient ways to handle financial obligations, charge meals, buy articles, reserve and use materials, etc.

Problem

Without the concept of account users need to carry large amounts of cash, may have trouble reserving items to buy or borrow, and would have serious problems sending funds to remote places.

An effective solution to this problem uses the concept of account and requires balancing the following forces:

- Accounts are very convenient for the customer, who doesn't need to carry large amounts of cash and needs to borrow books, sports equipment, or other items. Handling accounts efficiently and correctly is important for the institution to keep its customers happy.
- There is a wide variety of accounts and we need a generic model to describe the handling of all kinds of accounts, without a need to specify these accounts in detail.
- Accounts have dynamic as well as static aspects. For example, accounts usually have several states that affect the way they are used, one cannot withdraw money from a frozen account for example. Any solution must reflect and enforce the contexts defined by these states.
- There should be an explicit representation in the model for documents normally used in practice, e.g., a loan record should correspond to an object of a class. This makes the generation of those documents easier and the model is now more understandable.

- Different accounts require different types of transactions, the model should describe only abstract, common aspects of transactions.
- We need to give the authorized users a convenient way to prove their authorities.
- Most institutions have several branches or offices that can be used by their customers. Normally, one of these branches is in charge of an account and the users need to know about the location and facilities of all the branches.

Users need to perform the following generic use cases on accounts:

- <u>Open an account.</u> The customer's information (name, ID, address, etc.) is recorded, and credit is usually verified. An account is created and authorization cards are provided to the customers so they can access their accounts.
- <u>Perform a transaction.</u> Customers can perform transactions of different types against the accounts. For example, in a library users charge books to their accounts, renew a loan, pay fines.
- <u>Close an account.</u> The account is permanently deactivated and a historical record may be created.

These use cases are generic in that they correspond to a variety of situations or applications. They can be used to build a *semantic analysis pattern* [Fer00a], a type of pattern that tries to capture the semantics of a set of related applications. In a previous paper [Fer00], we showed this type of analysis pattern to be useful in building complex conceptual models. A good collection of these patterns makes creating conceptual models easier and less prone to error.

Solution

Start from class Account and add relevant entities; in this case customers, cards, and transactions. Build an institution hierarchy describing the branches of the institution and relate accounts to the branches.

a) Class Model

Figure 1 is a class diagram for the realization of these Use Cases. Abstract class **Account** may have a variety of concrete accounts. Customers are given a certain number of authorization cards and are owners of their accounts. With these cards customers can perform different types of transactions. We collect all the transactions in an account for a given period in class **TransactionSet (TXSet)**. The generic **Institution** may have several branches described by class **BranchInstitution**. Accounts are in charge of specific branches.

b) Dynamic aspects

Figure 2 shows a sequence diagram showing how a customer opens an account in an institution. This assumes that this is a new customer (indicated by the *addCustomer* operation). An account is created for this customer (possibly indicating an initial deposit) and added to a branch office (*addAccount* operation). A card is created and an account number is returned to the customer.

Figure 3 shows the state diagram for class Account. This assumes that an account can be deactivated at the request of the customer (e.g., if the customer is absent for some time), or frozen by the institution (e.g., if the account is delinquent). The superstate indicates that the account can be closed from any state.

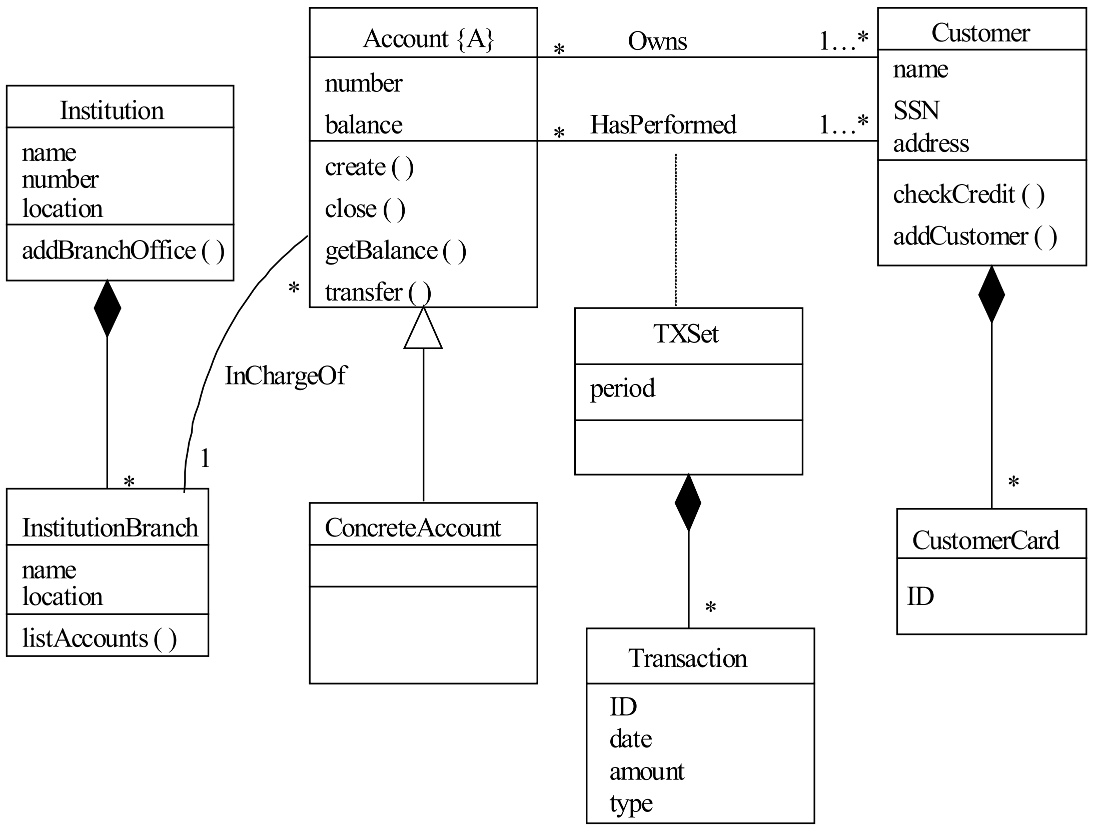

Figure 1. Class diagram for the Account pattern

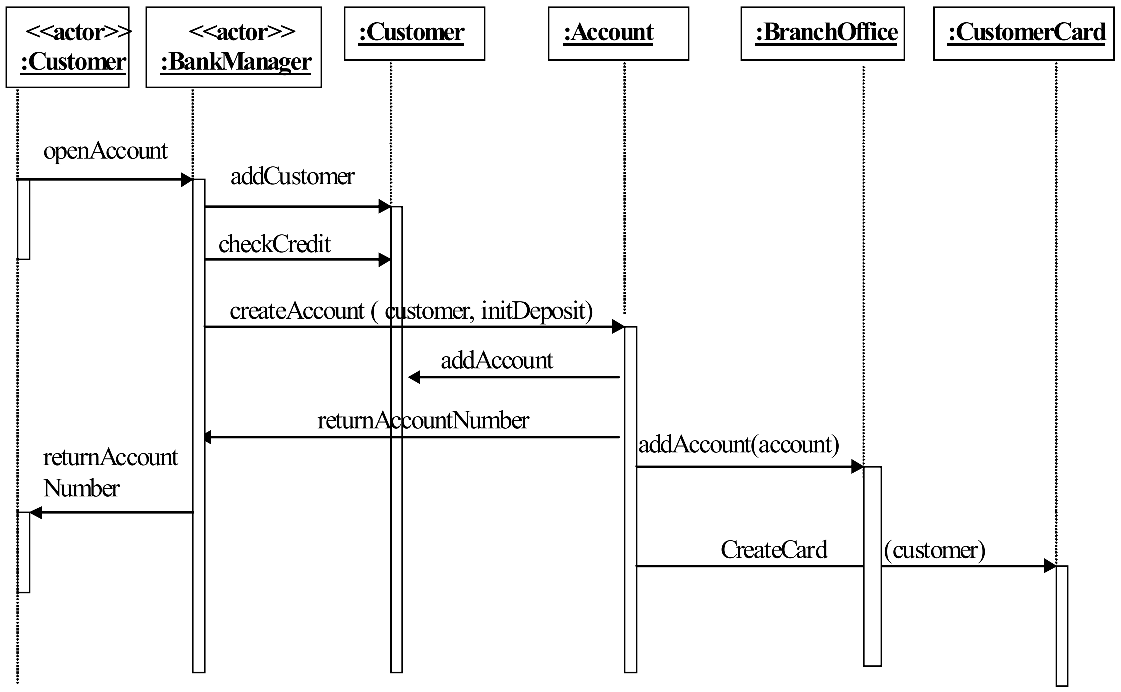

Figure 2. Sequence diagram for opening an account

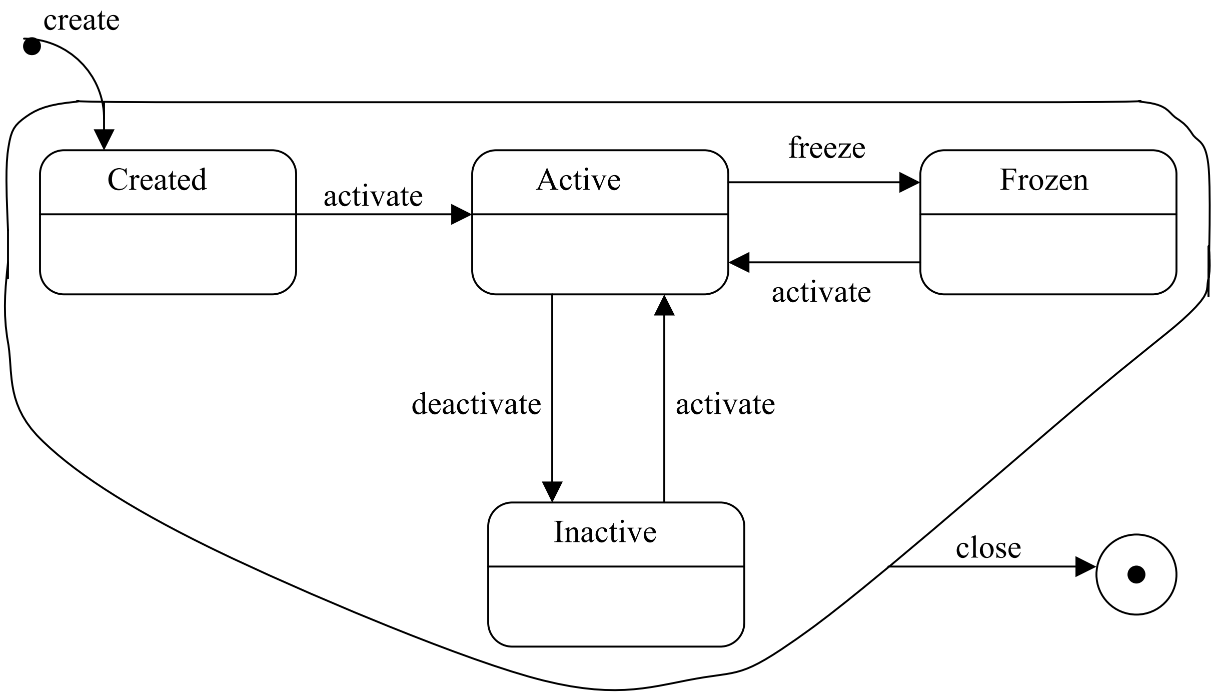

Fig. 3. State diagram for class Account

Example resolved

An example for Bank accounts is shown in Figure 4. The classes contained in the model include **Bank**, **BranchOffice**, **Account, CheckingAccount**, **Customer**, **BankCard**, **TransactionSet** (TXSet), and **Transaction**, with their obvious meanings. Class **TXSet** collects all the transactions for a user on his account for a given period of time. There are, of course, other types of accounts.

The association between Account and Customer shows that each account may be owned by one or more customers and a customer may have zero or more accounts. A customer can perform transactions to deposit or withdraw money.

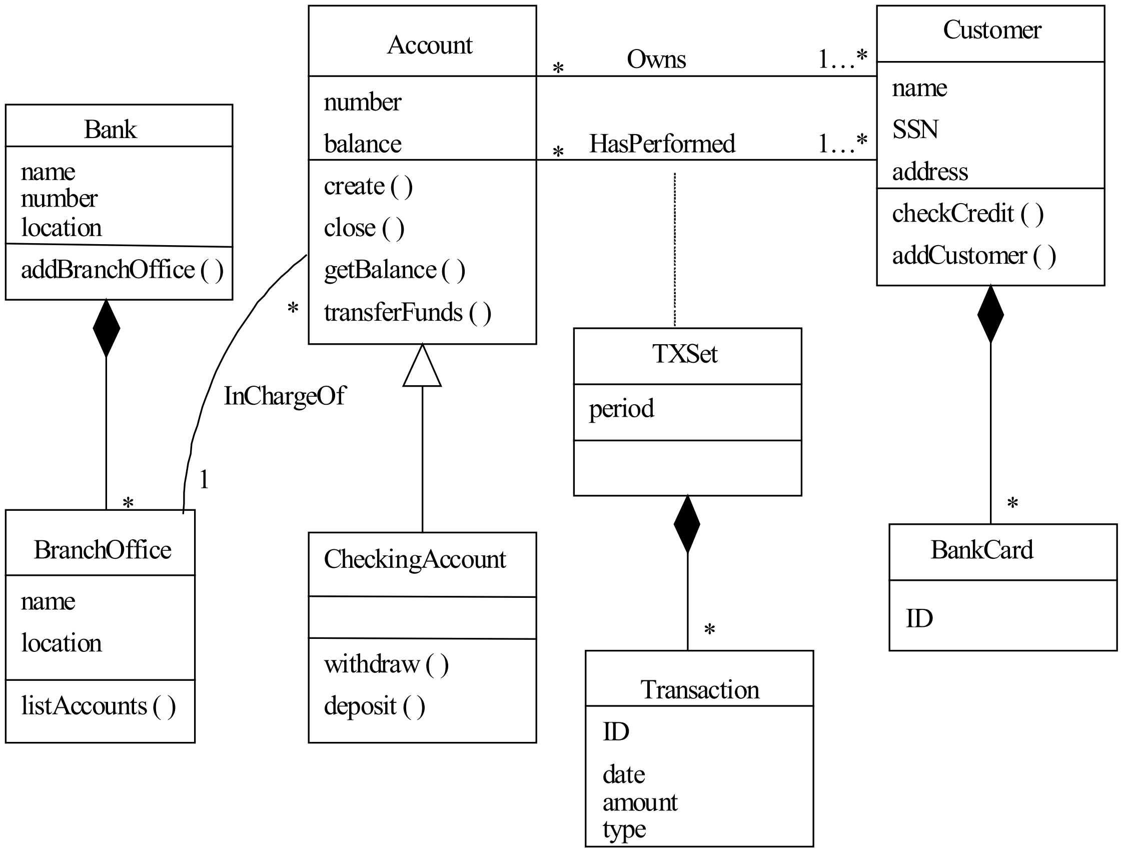

Figure 4. Class diagram for bank example

Known Uses

The following are examples of uses of this pattern:

- Banks, where customers have financial accounts of different types.
- Libraries, where patrons can borrow books and tapes.
- Manufacturing accounts, where materials are charged.

- Video stores, where customers can borrow or buy movies.
- Clubs, where members can charge meals and objects to their accounts.
- Hotels, where customers staying at the hotel can charge meals and telephone calls.
- Frequent flyer accounts in airlines.

Consequences

The pattern has the following advantages:

- It is clear that this model provides an effective description of the needs and can be used to drive the design and implementation of the software system. Not using a similar model would result in code that is hard to extend and probably incorrect.
- One can easily add other use cases: freeze account and activate/deactivate account.
- It can describe account handling in many types of institutions.
- Documents such as transaction records (e.g., a deposit slip) and physical cards are represented in the model by objects of specific classes.
- The model includes information to describe branch offices and their account responsibilities.
- Adding instances of the Authorization pattern [Fer01] the pattern can include the required authorizations for the roles in its use cases. Security is a very important aspect for accounts because of the monetary aspects involved.

The liabilities of this pattern come from the fact that to limit the size of the pattern and to make it more generic we have left out:

- Different types of customers. Each variety of customer could be handled in a special way. In particular, Hay [Hay96] and Fowler [Fow97] use the concept of Party, which can be an individual or an organization.
- Historical aspects, except for the history of transactions performed in a given time period. When an account is closed, its information may be transferred to a history file.
- Account administration functions, such as auditing.
- Policies for handling account closing, freezing, and activation. These vary between institutions and can be handled through OCL constraints [Wan98] or through business rules [Ars00].

There are also some modeling choices or variants:

- The institution card could be related to class Account instead of Customer. This may be more convenient for cases where different types of accounts require different types of authorizations.
- It could be possible to relate the transactions to a card instead of a customer. This could be useful for a customer who has several cards for his family members and wants to monitor their use.

Related Patterns

This pattern is based on the Accountability pattern of Fowler [Fow97]. Fowler's pattern is too abstract for many purposes and we have made the pattern more concrete while preserving its ability to be applied to many situations. Fowler's concept of accountability includes responsibilities such as being a manager or an employee and the corresponding accountability of the position. Also, Fowler's model doesn't consider dynamic aspects and considers transactions as a separate aspect. By considering accounts and transactions together we capture more closely the meaning of accounts in a variety of institutions. Johnson and Yoder have developed a set of banking patterns that include accounting patterns [JY]. They do not consider dynamic aspects either but have some complementary points that emphasize monetary aspects. Hay [Hay96] has a chapter on accounting, discussing economic aspects and is also complementary to our pattern. Silverton [Sil01] has a chapter on accounting and budgeting and considers details of transactions and budget aspects. Similarly to Hay the emphasis is on data modeling and no dynamic aspects are considered. IBM's Patterns for e-business include an account access pattern [ibm] but its description has few details. A design model for accounts, considering the use of components, can be found in [Lev02].

As indicated above, the Party and the Business Rule patterns can be combined with this pattern. Other complementary (in the sense that they appear frequently together with this pattern) patterns include Reservation and Use of Entities [Fer99], and inventory (Stock Manager) [Fer00b].

Acknowledgements

This paper has been significantly improved by the comments of our shepherd Ali Arsanjani and the Writers Workshop at EuroPLoP 2002.

References

[Ars00] A. Arsanjani, "Rule Object: A Pattern Language for Adaptable and Scalable Business Rule Construction", *Procs. of Pattern Languages of Programs Conference, PloP2000.* http://jerry.cs.uiuc.edu/~plop/plop2k

[Fer99] E. B. Fernandez and X. Yuan. "An analysis pattern for reservation and use of reusable entities", *Procs. of Pattern Languages of Programs Conference, PLoP99.* http://jerry.cs.uiuc.edu/~plop/plop99

[Fer00a] E. B. Fernandez and X. Yuan. "Semantic Analysis patterns", *Procs. of 19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195.

[Fer00b] E.B. Fernandez, "Stock Manager: An analysis pattern for inventories", *Procs. of PLoP 2000*, http://jerry.cs.uiuc.edu/~plop/plop2k

[Fer01] E.B.Fernandez and R. Pan, "A pattern language for security models", *Procs. of PLoP 2001,* http://jerry.cs.uiuc.edu/~plop/plop2001

[Fow97] M. Fowler, *Analysis patterns – Reusable object models*, Addison-Wesley, 1997.

[Hay96] D.Hay, *Data model patterns-- Conventions of thought*, Dorset House Publ., 1996.

[ibm] IBM Corp., Patterns for e-business, http://www-106.ibm.com/developerworks/patterns/

[JY] R. Johnson and J. Yoder, "Inventory and accounting", Part of "Banking patterns", http://www.joeyoder.com/marsura/banking/

[Lev02] K. Levi and A. Arsanjani, "A goal-driven approach to enterprise component identification and specification", *Comm. of the ACM*, vol. 45, No 10, October 2002, 45-52.

[Sil01] L. Silverton, *The data model resource book* (revised Edition), Vol. 1, J. Wiley & Sons, 2001.

[Wan98] J. Wanner and A. Kloppe, *The OCL: Precise modeling with UML*, Addison-Wesley 1998