Fundamental Banking Patterns

Lubor Sesera
SOFTEC & FIIT STU
Slovakia
lubor.sesera@softec.sk

ABSTRACT
The paper describes analysis patterns of software systems for banking. These patterns address complexity of banking products, effectiveness of computing account balances, and customer-orientation. The patterns are abstracted from real-world systems. Together, they form a fundamental pattern language.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures—*Patterns*

General Terms
Design

Keywords
Patterns

INTRODUCTION
Banking is the domain that is fairly known. People have accounts in banks and do at least several transactions in a month. However, there is a difference between knowing 'something' about banking (from the customer's point of view) and having 'deep knowledge' about a banking software system. From the point of view of 'deep knowledge' it is really surprising that only a few patterns have been published about this domain so far. Still, we do not have much more than accounting patterns [5], [8]. Unfortunately, although accounting is the basis of banking, banking itself is much more than accounting. The concepts of account and transaction [5] are valid but there are too many transactions in a bank each day so that one needs to cope with *effectiveness*. It influences the conceptual design of system. Furthermore, accounting is the simple low level mechanism and banking products and transactions are more *complex*.

PLoP '08, October 18–20, 2008, Nashville, TN, USA.

For instance, it is not enough to have just an account name and its balance. Additional information needs to be represented such as issued bank cards including their numbers, validity, and various limits. Bank card transactions do not address just accounts but bank cards and merchant devices as well. Last but not least, banking needs to be more '*customer oriented'* than accounting. For instance, instead of general account names the customer wants to see real product names. He is not interested in displaying counter accounts that are due to double-entry accounting. He expects just his products (accounts) to be displayed and in an integrated and attractive form to have a prompt overview. He is also happy to work in the 'customer session' mode with several transactions and only after he confirms them they are entered to accounts.

This paper addresses the issue of analysis patterns of software systems for banking. It starts from the ACCOUNT and TRANSACTION fundamental accounting patterns of Fowler and elaborates from the points of view mentioned above that are important in the banking system: effectiveness, complexity, and relationship to the customer. *Effectiveness* leads to patterns such as DAILY BALANCE, ARCHIVED HISTORICAL BALANCES, ARCHIVED HISTORICAL TRANSACTIONS, BANK DAYS and HIERARCHICAL BALANCING. *Complexity* of accounts and transactions results in patterns such as ACCOUNT TYPES, ACCOUNT CONTRACT, BUSINESS CASE, TRANSACTION TYPES and ACCOUNTING RULES. Furthermore, it includes fundamental patterns for fees and interests: ASSOCIATED FEE, BOOK RATE and PARAMETRIC RATE. *Relationship to the customer* results in patterns such as CUSTOMER PRODUCT, CUSTOMER SESSION, RATE PACKAGE, FRONT-END SYSTEM, and INTEGRATED CUSTOMER VIEW. Certain patterns such as ACCOUNT CONTRACT span more than one point of view and certain patterns e.g. FRONT-END SYSTEM are categorized a bit artificially in categories mentioned above.

The patterns mentioned above are *fundamental* patterns for banking and they form just a subset of an exhaustive pattern language. The domain of banking is too complex to be covered with a few patterns described in one article. For instance, these patterns do not address all subdomains of banking, e.g. they do not address securities or trade financing. (Although certain patterns such as ACCOUNT, TRANSACTION, and ACCOUNTING RULES can be used for these subdomains of banking.) I believe, however, that this article might be a good starting point.

The patterns described here are *analysis patterns* [5] for banking software systems. They are not business patterns [12], [13] providing business fundamentals. For instance, I do not discuss why it is important to have special rates for certain customers. I just say that to implement special rates effectively in

a software system it is beneficial to introduce the concept of RATE PACKAGE to this system. To give one more example: I do not discuss how to make fees 'less painful' to the customer, e.g. by transforming the transactional issuing bank card fee to the periodical monthly fee for the bank card; I just focus how to represent those fees to fulfill business requirements. On the other hand, the patterns presented in this paper are not architectural patterns [2], [6]. I do not address the architecture of the banking system[1] that may be a tiered system of a legacy host (core banking system), (JEE) integration and application middleware and a thin client (e.g. based on an Internet browser using AJAX). Also I do not address other modern buzzwords such as SOA, BPEL or a multichannel architecture of transactions. These patterns are not design patterns [7], although in certain places I mention that a particular issue can be solved by applying a design pattern, such as COMPOSITE or STRATEGY.

The patterns presented here are not 'ideal' patterns but they come from real world systems. Some of them, e.g. DAILY BALANCE, AVAILABLE BALANCE ARCHIVED HISTORICAL BALANCES, ARCHIVED HISTORICAL TRANSACTIONS, and BANK DAYS have been discovered in wide-spread legacy core banking systems. Other patterns, such as HIERARCHICAL BALANCING, CUSTOMER SESSION, BUSINESS CASE, RATE PACKAGE, FRONT-END SYSTEM, and INTEGRATED CUSTOMER VIEW have been abstracted from some up-to-date front-end systems. These systems are not wide-spread (but it might be nice to have them in all banks). Nevertheless, abstracted solutions come from the real-world systems and fulfill the prerequisite for patterns. In this article I do not describe ideas that might be nice but I have not found them in any software system. It seems to me that several times even I myself had a better solution for a problem (e.g. to use the STRATEGY pattern for calculating fees [1] or to generalize transaction types [19]) but I had not a chance trying to implement them in a system. I am sorry if some (or even all) patterns described here seem trivial to a reader. This is the world as it is.

The patterns presented here also do not claim to be abstract enough to cover domains other than banking. Obviously, certain patterns such as CUSTOMER PRODUCT, CUSTOMER SESSION, BUSINESS CASE, RATE PACKAGE and INTEGRATED CUSTOMER VIEW exceed the boundaries of banking and are usable in other types of financial systems (and even beyond). However, other types of financial systems may require additional fundamental patterns not covered here. For instance, in the insurance domain the concept of account is less important than the concept of contract and other fundamental patterns such as PARTIAL CLAIM, PARTIAL FULFILLMENT and PAYMENT METHODS are used [18]. I understand the challenge of Fowler to discover 'highly generic patterns' that 'cut across traditional boundaries of systems development and business engineering' but in analysis patterns, apart from a few exceptions, we are not advanced enough to do that. Patterns restricted to a particular domain may be the first step towards more ambitious goals. Keller's patterns [11] are a perfect example of patterns restricted to just one domain (insurance) and still valuable.

To confirm that an idea is a pattern one should add known uses. This is not so easy with banking systems, however. Banking systems are proprietary. First of all, they need to be protected from competitors. Second, they need to be protected from software architects to hide that they are not wonders of the world☺. To avoid investigation, I do not refer to real world banking systems explicitly. I use symbolic names of those systems, instead. Their acronyms are as follows:

- CBS 1 – One of the world leading core banking legacy systems. It is used in Bank 2 and Bank 3.
- CBS 2 – Another core banking system. Before a bank merge it was used in Bank 4.
- CMS 1 – The card management system implemented in Bank 1.
- CMS 2 – The card management system implemented in Bank 3.
- CRM 1 – The customer relationship management system of one of the world's leading banking software company, customized for Bank 3.
- LPS 1 – The loan process system designed and implemented for Bank 3.
- TS 1 – The JEE teller system of one of the world's leading software company, customized for Bank 2.
- TS 2 – The teller system implemented in Bank 1.

Table 1 provides a summary of the patterns. Every pattern is summarized using the Problem-Solution template and its main category: fundamental (F), effectiveness (E), complexity (C) or relationship to the customer (R).

Figure 1 shows the basic pattern map. For readability, this map does not show all relationships among patterns. To find all relationships of a pattern one should follow the *Related Patterns* section of the pattern description. In the diagram Czarnecki's modeling notation [3], [9] to describe pattern sequences is used: an open circle on a relationship indicates an optional relationship while a solid circle represents a recommended relationship. This is slightly different from [9] where the solid circle represents a mandatory relationship. However, mandatory relationships are a bit problematic in this pattern language. There is not a single system in a bank and whether a relationship is mandatory depends on the type of software system. For instance, DAILY BALANCE is a must in a core banking system but it is not needed at all in a card management system. Or one can argue that the relationship from ACCOUNT to ACCOUNT CONTRACT should be mandatory as data should be normalized but account contracts are rarely found in core banking systems that are in operation for decades! On the other hand, if a new card management system is developed from scratch this relationship should be considered mandatory. The diagram is a compromise addressing a complex banking system including accounts and assuming a 'nice' design of this system but admitting that it may not include everything.

[1] FRONT-END SYSTEM and INTEGRATED CUSTOMER VIEW might be an exception.

Table 1. Summary of patterns

#	Ct.	Pattern	Problem	Solution
-	*F*	*ACCOUNT (Fowler)*	*How do you record the history of changes to some quantity?*	*Create an account. Each change is recorded as an entry against the account. The balance of the account gives the current value.*
-	*F*	*TRANSACTION (Fowler)*	*How do you ensure that nothing gets lost from an account?*	*Use transactions to transfer items between accounts.*
1	E	DAILY BALANCE	How do you handle both the state (balance) and the state changes (transactions) on the account in an effective way?	Make persistent both the balance attribute and transactions. The balance attribute represents closing balance of the previous day. Calculate the actual balance dynamically.
2	E	AVAILABLE BALANCE	How do you handle both the actual balance and the available amount of money?	Separate the available balance from the actual balance. Calculate the available balance dynamically.
3	E	ARCHIVED HISTORICAL BALANCES	How do you provide historical balances?	Archive daily balances at specific time points.
4	C	ACCOUNT TYPES	How do you represent many types of accounts so that they can be manipulated in a consistent way?	Generalize various account types to a general account concept. Make each specific account type an extension to this general account.
5	C	ACCOUNT CONTRACT	How do you represent complex information on various bank products?	Separate an account contract from the account. The account should contain 'accounting' data while the account contract should contain the descriptive data of the product.
6	R	CUSTOMER PRODUCT	How do you make many account types transparent to the user and handle composite customer products?	Introduce a customer product that provides a façade to one or more accounting products.
7	C	BUSINESS CASE	How do you approach the product life cycle?	Introduce the concept of a business case aggregating business process objects. Create a process model of the business case.
8	C	TRANSACTION TYPES	How do you represent many types of transactions so that they can be manipulated in a consistent way?	Generalize various types of transactions to the general transaction concept. Make a specific transaction type with additional data an extension to this general transaction.
9	C	ACCOUNTING RULES	How do you map transactions to general ledger entries?	Define accounting rules. Make the structure of accounting rules specific on a business subdomain.
10	E	ARCHIVED HISTORICAL TRANSACTIONS	How do you manage historical transactions effectively?	Introduce an archive for transactions from the final accounts of the previous year. Make this an extra effort for the customer to search in the archive.
11	R	CUSTOMER SESSION	How do you implement several transactions of the same customer?	Introduce the concept of customer session. Keep transactions transient within the session until they are finally approved by the customer.
12	E	BANK DAYS	How do you balance job calculations and calendar days?	Introduce the concept of bank days to the system. Provide a dedicated procedure for the close of the day as it may not match the close of the calendar day.
13	E	HIERARCHICAL BALANCING	How do you coordinate various types of balancing?	Define hierarchical balancing with forced balancing based on layers.
14	C	ASSOCIATED FEE	How do you clarify fees?	Make an explicit reference from the fee to its

				originated transaction. Define meta-relationships between fee types and transaction codes / transaction types.
15	C	BOOK RATE	How do you handle constantly changing interest rates and fee rates?	Separate the fee rates / interest rates from their types. Allow to define the period of validity for values.
16	C	PARAMETRIC RATE	How do you approach various dependences of the book rate?	Represent parameters that have impact on the book rate. Implement the parametric function to calculate the actual book rate.
17	R	RATE PACKAGE	How do you manage different rates for different customers?	Introduce the concept of rate package to the system.
18	R	FRONT-END SYSTEM	How do you extend the banking system quickly enough to support new customer products and processes?	Develop or buy off-the-shelf a front-end system extending the particular functionality of the core banking system.
19	R	INTEGRATED CUSTOMER VIEW	How do you restore the integrated customer view using the 'quick win' strategy?	Provide the portal-like integrated customer view with the main attributes. Product maintenance remains in individual systems.

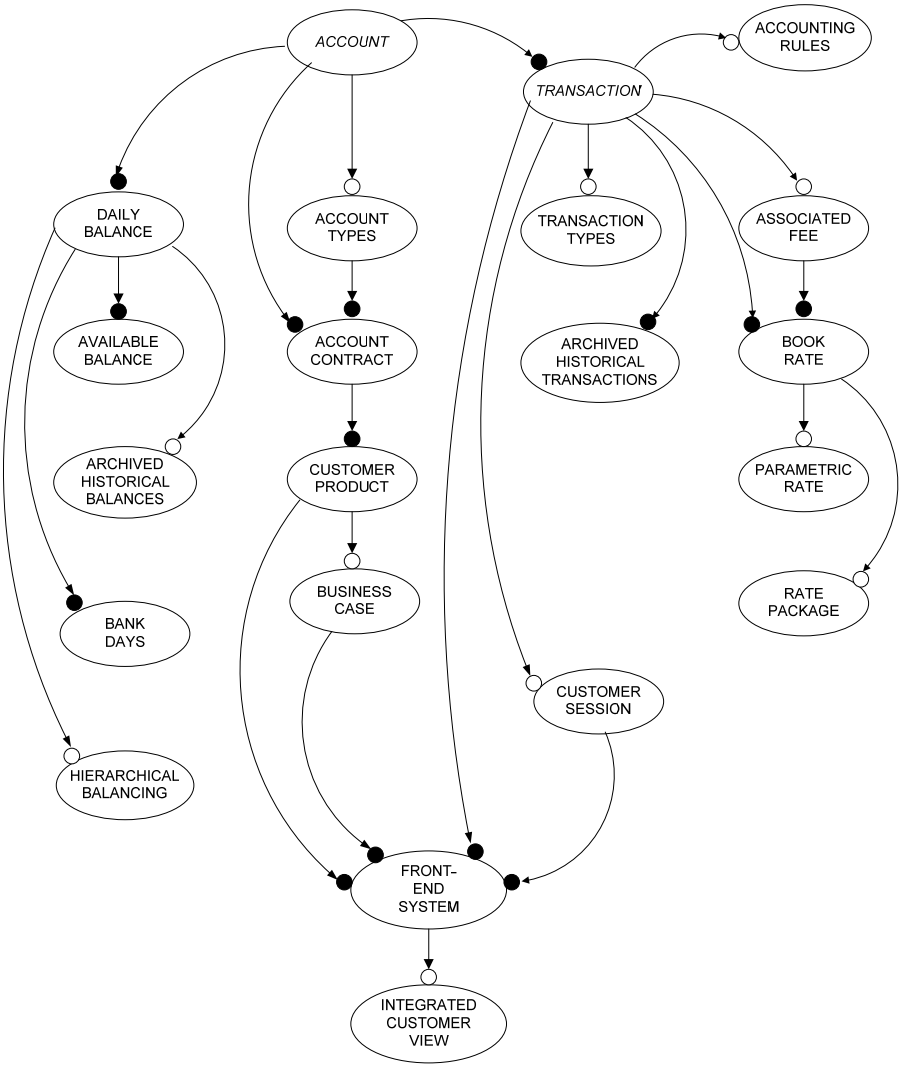

Figure 1 Pattern Map

1. DAILY BALANCE

1.1 Context

A large number of customers have current accounts in a bank. These customers perform many transactions (hundreds of thousands or even more) on their accounts each day. The account includes a balance that is the total of all transactions associated with the account. Customers expect short response time for displaying both the balance and account transactions at any time.

A simple conceptual model of this situation using the UML class diagram and the OCL language is shown in Figure 2.

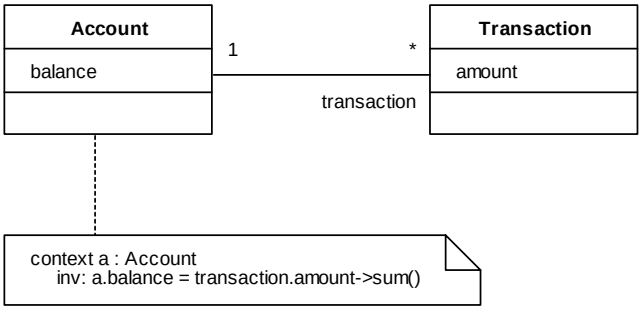

Figure 2 Account Balance

1.2 Problem

How do you handle both the state (balance) and the state changes (transactions) on the account in an effective way?

1.3 Forces

- The customer wants to see the balance on his account quickly at any time, but
- This customer also wants to perform and see account transactions at any time and
- The balance needs to be consistent with the total of all of the account transactions, but
- As there have been many transactions performed from the time this account was opened, it is not effective to calculate the balance again and again, but
- It is risky to make the balance attribute persistent due to possible inconsistency with the calculated total of account transactions.

1.4 Solution

Make both the balance attribute and transactions persistent. The balance attribute represents the closing balance of the previous day. Calculate the actual balance dynamically as the total of the closing balance and today's transactions.

1.5 Example

The solution is illustrated in Figure 3. In the diagram two invariants are shown. The first invariant describes how the `closing balance` is calculated based on the `closing balance` of the previous day and transactions dated from the previous `closing time` to this `closing time`. The second invariant shows how the `actual balance` is calculated based on the `closing balance` and transactions dated from its `closing time`. To represent the invariants the OCL language is extended with two operations: `old` and `sum`. The `old` operation returns the previous value of an attribute. The `sum` operation calculates the total of collection of numbers.

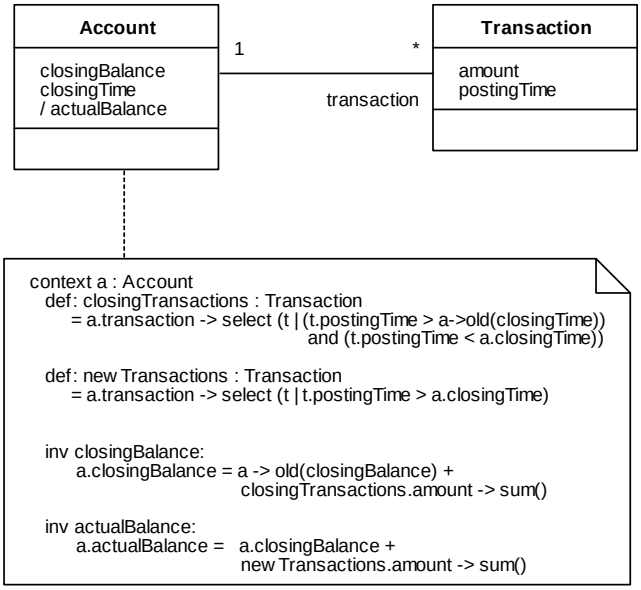

Figure 3 Daily Balance

1.6 Variants

A variant of the solution is that the actual balance attribute can be made persistent. This attribute is recalculated after a new transaction is posted. This variant solution provides faster responses to user queries on actual balance but it is more fragile as transactions may be canceled or rejected later on. The variant is used with the AVAILABLE BALANCE pattern in which fragile balances are represented in the form of `available balance.`

1.7 Consequences

+ Representation is effective as only a limited number of transactions are taken into consideration when computing the actual balance.
+ Inconsistency between the balance and the transactions is reduced.
+ The closing balance can be calculated out of peak times ('at night').
– Inconsistency is not fully eliminated as the derived value (i.e. `closing balance`) is persistent.

1.8 Related Patterns

- DAILY BALANCE elaborates the ACCOUNT pattern by splitting the balance into two balances.
- AVAILABLE BALANCE further elaborates DAILY BALANCE by adding one more balance.
- To calculate the closing balance or the available balance TRANSACTIONS are used.
- Balances include ASSOCIATED FEES.
- ARCHIVED HISTORICAL BALANCE adds a history of daily balances.
- The BANK DAYS pattern specifies balance days.
- Interests using BOOK RATE or RATE PACKAGE are calculated based on closing balances.

- The daily balancing can launch ACCOUNTING RULES.
- FRONT-END SYSTEM can and INTEGRATED CUSTOMER VIEW should display the closing and the actual balances.

1.9 Known Uses

In CBS 1 two account balances are used: the cleared balance and the ledger balance. The cleared balance corresponds to our closing balance and the ledger balance corresponds to our actual balance. Both of these balances are persistent attributes.

Similarly, two account balances are used also in CBS 2.

2. AVAILABLE BALANCE

2.1 Context

The actual balance may not correspond to the amount of money available to the customer. There may be certain transactions that are planned to be executed in a short time (during a day); there are certain holds on the account, a minimal balance required by a bank or an overdraft allowed, etc.

2.2 Problem

How do you handle both the actual balance and the available amount of money?

2.3 Forces

- The customer wants to see how much money he owns, but
- He also wants to see, how much money he can use to avoid penalties or failed transactions.

2.4 Solution

Separate the available balance from the actual balance. Calculate the available balance dynamically. Show both balances to the customer.

2.5 Example

In Figure 4 a sample UML class diagram with OCL invariants is shown. When compared to Figure 3 the `Held Item` class was added. Furthermore in the `Account` class certain new attributes were included. Finally, the `isProjected` attribute in the `Transaction` class classifies transactions to already executed and projected in the future. The `closing balance` invariant is the same as in Figure 3. The `actual balance` invariant is slightly modified so that it does not include projected transactions. The `available balance` invariant is new. It is calculated based on `actual balance`. From this, `minimal balance`, projected transactions and held items need to be subtracted and `overdraft` allowed needs to be added.

2.6 Consequences

+ The experienced customer can see both the actual balance and the available balance on his account.
– The inexperienced customer might be confused when he sees two or three balances on his accounts (with carefully encrypted acronym names ☺). However, the consequence may be reduced with education.

2.7 Related Patterns

- AVAILABLE BALANCE further elaborates daily balance by adding one more balance.
- The BANK DAYS pattern specifies the balance days.
- The customer should be warned if his planned TRANSACTION could overdraft the minimum balance.
- CUSTOMER SESSION needs to calculate and display AVAILABLE BALANCE. It should restrict further transactions when the available balance is to overdraft the minimum balance.
- FRONT-END SYSTEM can and INTEGRATED CUSTOMER VIEW should display the available balance.

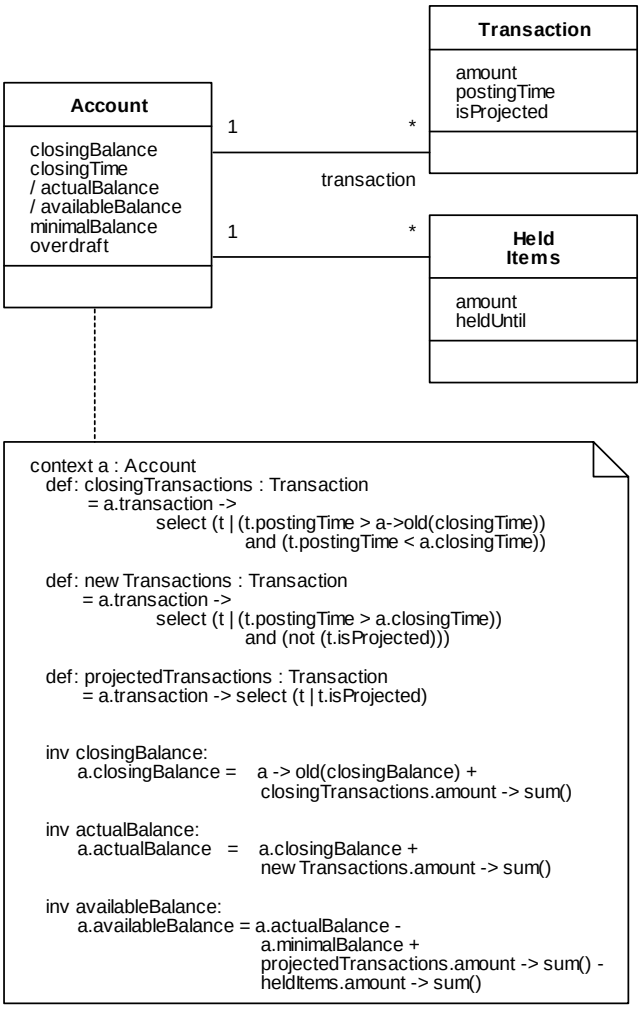

Figure 4 Available Balance

2.8 Known Uses

In CBS 1 three account balances are provided: the cleared balance, the ledger balance, and the available balance. The cleared balance and the ledger balance are persistent while the available balance is calculated dynamically.

Three account balances are quite common to many banking systems. I can see them in an on- line banking system of the bank in which I opened my accounts although I do not have deep knowledge of its core banking system to give more information.

3. ARCHIVED HISTORICAL BALANCES

3.1 Context

Apart from the daily balance, the customer would like to see certain historical balances. For instance, when a retail customer receives a monthly statement he would like to see both the opening balance of that month and the closing balance so that he could easily compare this opening balance to the closing balance of the previous statement and he can use this opening balance as the starting point to check the new closing balance. Or a court requires from the bank to provide the account balance at a specific date for a legal proceeding.

3.2 Problem

How do you provide historical balances?

3.3 Forces

- The customer wants to see historical balances regularly, but
- Computing the historical balance backward from the daily balance is time consuming, and
- Storing all daily balances for all accounts is wasteful as most of the retail customers do not perform transactions each day.

3.4 Solution

Archive[2] daily balances at specific time points. This time point is normally the end of month. Archive such end of month balances for several years. Compute the required historical balance backward from the nearest archived balance.

3.5 Variants

Archive all daily historical balances, but make the daily historical balance persistent only when it is different from the previous historical balance.

Because this variant method requires more disk space it is usually applied for short periods, usually for a month or several months. For long periods the standard solution is used.

3.6 Example

In Figure 5 the `Balance` class is decoupled from Account to store historical balances (including the closing balance of the previous day if it is different from the previous closing balance).

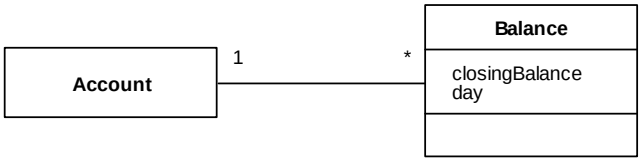

Figure 5 Historical Balances

3.7 Consequences

+ The solution is a compromise between required database space and time to compute any historical balance.
+ Making historical balances persistent further reduces the risk of inconsistency of the balance versus transactions.
– The solution slightly complicates the algorithm for computing the historical balance as the balance of the required date may not be stored.

3.8 Related Patterns

- ARCHIVED HISTORICAL BALANCE adds the history to DAILY BALANCE.
- FRONT-END SYSTEM and INTEGRATED CUSTOMER VIEW can display historical balances.

3.9 Known Uses

In CBS 1 there is a dedicated table of historical balances. The daily historical balance is stored only when it is different from the previous historical balance.

In CBS 2 the closing balance of each month is stored regardless of whether it is the same or different from the closing balance of the previous month. Apart from these and the closing balance of the previous day, other daily balances are not stored.

4. ACCOUNT TYPES

4.1 Context

A bank offers several types of accounts such as current accounts, saving accounts, term deposit accounts, passbook accounts, sweep accounts, etc. Furthermore, it keeps internal general ledger accounts. These types of accounts have many features in common but they also have their own specific features, such as maturity dates for term deposit accounts, passbook numbers for passbook accounts or thresholds for sweep accounts.

4.2 Problem

How do you represent many types of accounts so that they can be manipulated in a consistent way?

4.3 Forces

- Different types of accounts have their own specifics depending on their purposes, but
- Implementation of each specific type separately requires a lot of time and effort again and again and makes the system opaque, but
- Different types of accounts also have many things in common and
- Some of the account types are just bank internals and do not have ‘externally visible’ account numbers and
- The system should constrain types of accounts, but
- The system should also be extensible so that new account types can be added.

4.4 Solution

Generalize various account types to a general account concept. Make each specific account type an extension to this general account.

Note here that the account type represents the computational concept, i.e. the type having additional features to the general account. From the point of view of accounting, there can be other ‘account types’. These are usually called account codes. Ideally, the account type covers several account codes. However, this is not always the case. Usually, account codes do not need to be

[2] Here, the term *archive* means *make persistent*. It does not mean that balances need to be archived in a special archived device. Usually, these balances are archived in the same device and database as accounts.

represented with subclasses because from the point of view of modeling they do not bring additional features to account types.

4.5 Example

Figure 6 outlines the conceptual solution using inheritance (in relational core banking systems associations need to be used). Fundamental relationships such as the customer account ownership are specified on the general level, i.e. with the `Account` class. Specialized relationships are defined at the level of subclasses. For instance, there is a relationship between `Sweep Account` and its `Current Account` to allow automatic money transfers between these accounts. (In daily balancing when the closing balance of `Current Account` is above a limit defined by the account owner, the amount of money above the limit is automatically transferred to `Sweep Account` having an interest rate similar to a saving account. On the contrary, when the closing balance is below another limit defined by the customer, the money is automatically transferred from `Sweep Account` to `Current Account`). Similarly, the customer may specify `Current Account` for his `Term Deposit Account` to which an interest (and a principal) is automatically transferred at a maturity day.

The `Account Type` class represents the UML power type, i.e. its instances correspond to subclasses of `Account`. Account codes are in a separate class from `Account Type`. For one `Account Type` there are usually several instances of `Account Code`. For instance, for `Term Deposit Account` there are account codes depending on a type of customer (a physical person or a legal person), a type of currency (the domestic currency or a foreign currency) and a maturity period (1, 3, 6, 9, 12 or 24 months).

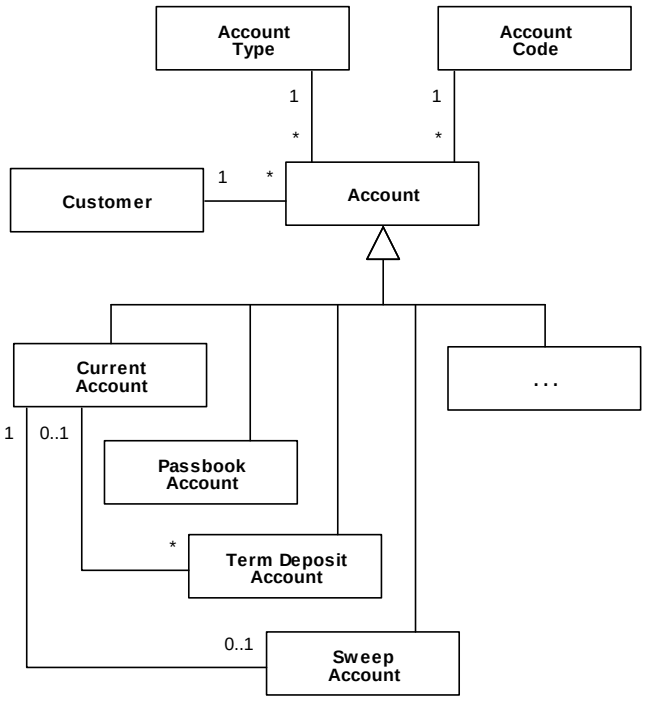

Figure 6 Account Types

4.6 Consequences

+ Different types of accounts have the same fundamental representation and can be manipulated in a similar way.
+ Specifics of an account type are stored in one place.
+ New account types can be added later on.
− Implementation is less efficient as there is a need to compound information from two objects.

4.7 Related Patterns

- ACCOUNT TYPES elaborates the ACCOUNT pattern.
- ACCOUNT CONTRACT is a more complex alternative to ACCOUNT TYPES.
- TRANSACTION TYPES can be associated with specific ACCOUNT TYPES. However, there is not one to one correspondence between an account type and a transaction type.
- ACCOUNT TYPES are many times parameters of ACCOUNTING RULES, BOOK RATES, and PARAMETRIC RATE. More frequently, account codes are those parameters instead.
- ACCOUNT TYPES can help in HIERARCHICAL BALANCING. However, account groups for hierarchical balancing are formed based on internal bank accounts instead of customer accounts.
- ACCOUNT TYPES facilitate displaying of products in FRONT-END SYSTEM and INTEGRATED CUSTOMER VIEW.

4.8 Known Uses

Although this pattern looks simple it is not easy to find it implemented in a pure form and still it is 'nice to have'.

In CBS 1 there is one core table for all types of accounts including general ledger accounts. There are also extension tables to store specific attributes. Unfortunately, there is no one to one correspondence of the extension table and the account type. The general account is specified by a compound key (customer number, account code, currency and others). The general account may or may not have an external (retail) number

5. ACCOUNT CONTRACT

5.1 Context

A bank sells products that are rich in information to be represented such as loans or bank cards. For instance, for the loan its type, the loan amount, the current principal, the accrued interest, the repayment amount and its frequency and other attributes need to be represented. Furthermore, one or more collaterals can be associated with the loan. The bank card is described by its number, the card product, the embossed name, validity, status, various limits, etc. There can be several bank cards associated with the same customer account.

5.2 Problem

How do you represent complex information on various bank products?

5.3 Forces

- The concept of account is the universal concept in banking, but
- Even with ACCOUNT TYPES, accounts would be overwhelmed with much information for complex products.

5.4 Solution

Separate an account contract from the account. The account should contain 'accounting' data such as the balance, the interest, the date of maturity, and associated transactions. The account

contract should contain the descriptive data of the product such as the product type, the product name, associated objects, etc.

5.5 Example

Figure 7 shows the separation of `Contract` from `Account`. `Contract` has its own `contracting party` that is usually the same but can also be different from `account owner`. `Contract` can be a complex entity consisting of `Contract Items`. If `Object of Contract` can be shared among several `Contracts` it is separated from `Contract Item`.

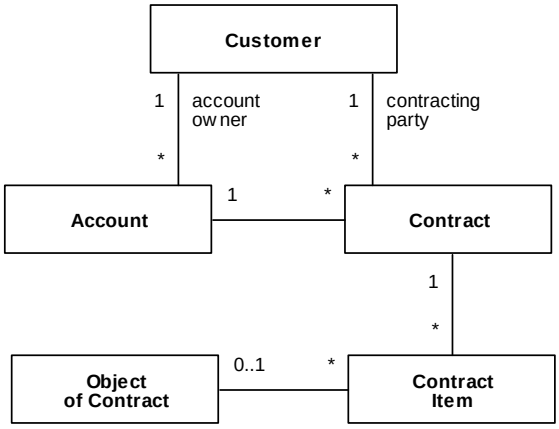

Figure 7 Account Contract

A bank card contract is an application (and a simplified 'version') of this pattern. `Bank Card Contract` specifies various bank card limits. It is associated with `Account` (the current account or the credit account). There might be several `Bank Card Contracts` associated with the same `Account`, e.g. the customer has two debit cards. `Card holder` may be the same or he may be different from `account owner`. For instance, the customer who is `account owner` is `card holder` of the first debit card, while his wife is `card holder` of the second debit card. Several `Contract Items, i.e. Bank Cards` can be associated with `Bank Card Contract.` For instance, bank cards have a revolving nature and due to historical transactions it is good to store not only the valid `Bank Card` but also historical `Bank Cards` (furthermore, validity periods of the current and historical `Bank Cards` may overlap by one month). `Bank Card` has attributes such `card number` or `valid thru`. (The embossed name attribute is usually the attribute of the `Customer` class.)

A safe deposit box rental is another simple application of this pattern. The rental can be paid from `Account` (or by cash). `Safe Deposit Box` is `Object of Contract` that is used again and again by the following contracts.

5.6 Consequences

+ The universal concept of account is preserved.
+ The complex products can be represented.
+ The account is not overwhelmed with too many details.
– There is a need to compound information from two sources.

5.7 Related Patterns

- ACCOUNT CONTRACT elaborates the ACCOUNT pattern. It separates the 'non-accounting' data from the account.

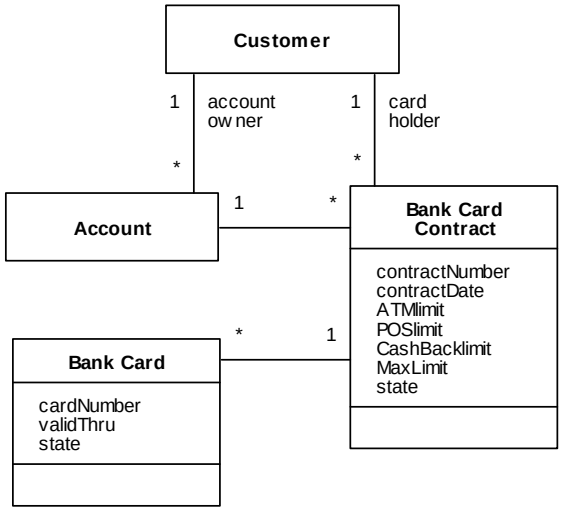

Figure 8 Bank Card Contract

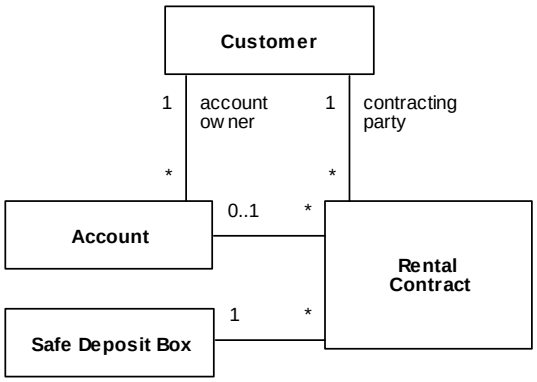

Figure 9 Safe Deposit Box Rental Contract

- ACCOUNT CONTRACT is a more complex alternative to ACCOUNT TYPES.
- CUSTOMER PRODUCT provides FAÇADE [7] to complex account contracts.
- Contract types are many times parameters of ACCOUNTING RULES, BOOK RATES, and PARAMETRIC RATE.
- FRONT-END SYSTEM should allow an easy manipulation of account contracts.
- INTEGRATED CUSTOMER VIEW displays account contracts.

5.8 Known Uses

In CBS 1 the dedicated loan record is separated from the account and it contains loan specific data as outlined above. Accounts are assigned to loans based on accounting rules. In the bank in which I was engaged usually two accounts were opened for the loan: one representing the principal and another one for interests.

In our CMS 1 we designed the card contract concept that stores the card product, card limits, and other attributes. There can be more than one contract associated with the account. Individual bank cards described by its number and validity are represented as individual records and assigned to the card contract. These bank cards include both valid and historical cards. (In the overlapping

period two bank cards, both the older one and the new one, are valid.)

6. CUSTOMER PRODUCT

6.1 Context

There are too many account (contract) types and subtypes. For instance, in Bank 3, there are more than a dozen subtypes of current accounts (account codes) depending on the customer category; whether the customer is a private person, a company, a government institution, etc.; and if he is a private person, whether he is a resident or non-resident, a bank employee, a student, a retiree, etc.

Furthermore, the bank offers packages for customers, e.g. sweep accounts, current accounts with one or two bank cards, etc.

6.2 Problem

How do you make many account types transparent to the user and handle composite customer products?

6.3 Forces

- Due to accounting many account subtypes are needed, but
- Too many account types and subtypes makes the system difficult to understand and
- The bank offers packages that aggregate several account types and subtypes.

6.4 Solution

Introduce a customer product that provides FAÇADE [7] to one or more accounting products. The customer product can aggregate several accounting products. Make the customer product to represent the business and marketing view, while the account type represents the information point of view and the account subtype (account code) represents the accounting point of view.

Note that it would be possible to apply the COMPOSITE design pattern [7] for customer products so that packages can be recursively composed of other packages. Unfortunately, we have not seen this pattern used for bank products (due to efficiency).

6.5 Example

Figure 10 shows the sample class diagram for defining customer products. (It is just a sample diagram as product packages are too complex to be represented with just a few classes.) In the diagram the 'traditional' approach is used: every customer product has both a header (instance of `Customer Product`) and a detail (instance of `Product Item`). `Product Item` can be based on another `Product Item` of the same `Customer Product`. `Product Item` references 'technical' `Account Type` (or even `Account Code`) and/or `Object Type`.

Figure 11 shows CUSTOMER PRODUCT on the instance level.

Some examples for using the schema in Figure 10 for real-world product packages:

- Package 1: current account + ATM card. The package is defined as instance of `Customer Product`. It is the composition of two instances of `Product Items`. The first instance of `Product Item` is associated with the current account instance of `Account Type`. The second instance of `Product Item` is based on the first instance of `Product Item` (the ATM card is issued to the specific current account). This second instance of `Product Item` is associated with the ATM card type instance of `Object Type`.
- Package 2: current account + sweep account + ATM card + credit card. The package is instance of `Customer Product` and the composition of four `Product Items`. The first instance is associated with the current account instance of `Account Type`. The second instance is based on the first instance and associated with the sweep instance of `Account Type`. The third instance is again based on the first instance but it is associated with the ATM card instance of `Object Type`. The fourth instance is not based on any other previous instances. It is associated with the credit account instance of `Account Type` and the credit card instance of `Object Type`.

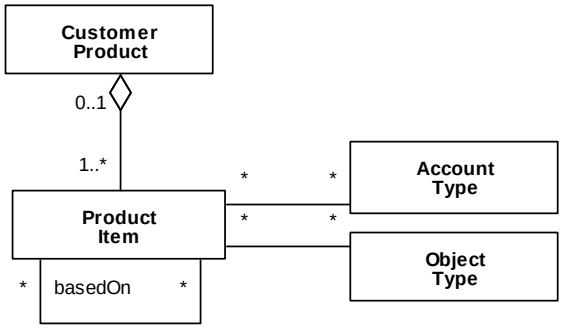

Figure 10 Customer Product (meta-level)

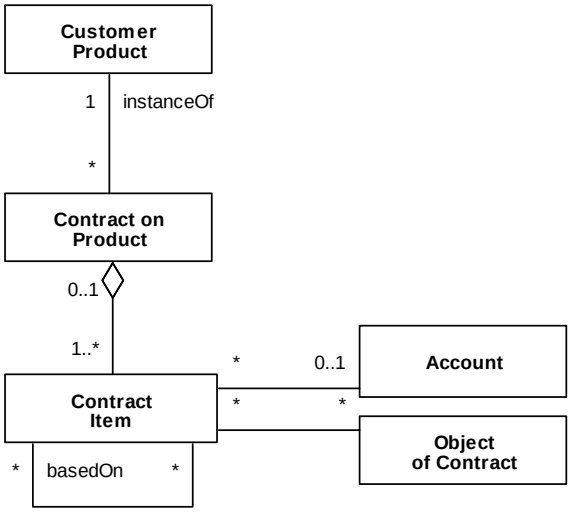

Figure 11 Contract on Product

6.6 Consequences

+ Account types and account codes for accounting are preserved.
+ Customers have a manageable number of products to choose from.
+ The marketing department is happy because packages are supported and customers buy more products in total.
+ Customers can buy less expensive composite products (and they do not realize they sometimes also buy what they do not need ☺).
– It requires some effort to implement customer products.

6.7 Related Patterns

- CUSTOMER PRODUCT provides FACADE [7] to complex account contracts.
- Customer products are many times parameters of ACCOUNTING RULES, BOOK RATES, and PARAMETRIC RATE.
- BUSINESS CASE elaborates a complex CUSTOMER PRODUCT having a non-trivial life cycle.
- FRONT-END SYSTEM is usually developed to handle customer products.
- INTEGRATED CUSTOMER VIEW displays customer products.

6.8 Known Uses

In CMS 1 we implemented four views of the bank card: the view of the card company (similar to the customer's view), authorization view (contains more card types), the card management system view (all the details) and the central registry of customers view (only restricted number of types needed). Relationships among the views were defined using meta-level entities. The concept of card packages was implemented in the system.

In CBS 1 bank cards may be assigned to accounts. Nevertheless, this core banking system does not fully support the concept of packages.

7. BUSINESS CASE

7.1 Context

Certain bank products have a fairly complex life cycle. For instance the loan application process of a complex loan product starts with a meeting with customer who is given one or more (non-binding) offers. If an offer is accepted, the customer brings the required documents and applies for the loan. The application is formally verified and the customer is rated. Then a credit officer prepares a proposal. Afterward, the risk management department may assess the proposal. Then a committee decides the proposal. Finally, contract documents are prepared and signed and the loan can be disbursed. This is not the end of the process, however. After disbursement, the customer is regularly (e.g. once a year) monitored; his accounting documents are checked, the customer is rated and loan parameters may be changed. In the life cycle described above the subsequent process can change an attribute value set by the predecessor such as the loan amount, the interest rate, or the collateral value.

7.2 Problem

How do you approach the product life cycle?

7.3 Forces

- Life cycle information is temporal and only the contract data counts, but
- Until the contract is signed and the product opened in the core banking system credit it is important to keep as much information as possible, but
- This may result in a complex net of objects.

7.4 Solution

Introduce the concept of a business case that aggregates objects representing information on business processes associated with a product. These business process objects should have an origin in business documents. Create a process model of the business case.

7.5 Variants

Business process objects can have their own life cycles. This variant is used when the life cycle of business case is too complex. The solution has two alternatives. In a 'hierarchical' alternative, an object life cycle extends an activity of the business case life cycle. In a 'distributed' alternative, there is not a central business case process model and transitions are between activities of life cycles of individual objects. The 'hierarchical' alternative gives a better view for the user but it is more difficult to implement.

7.6 Example

Figure 12 shows fundamental business process objects of the loan approval process aggregated to the `Business Case` class. Associations are optional as the corresponding subprocesses may be skipped or the loan process may stop at any moment. Certain aggregations have one to many multiplicity. There may be several options of (non-binding) `Offers` given to the customer. There may be several `Contracts` of different types associated with `Business Case` (e.g. a loan contract, collateral contracts with various parties, etc.). Finally, there can be several `Disbursements` of the loan.

Figure 13 outlines the process model of the business case using the UML Activity diagram. The diagram elaborates activities described in the context part of this pattern up to the disbursement (excluding monitoring and loan updates). For simplicity, objects that are outputs of activities are not drawn and names of activities correspond to names of the business case objects. Apart from two exceptions, backward transitions are omitted as well. The model includes many decision points on the next steps of the loan process.

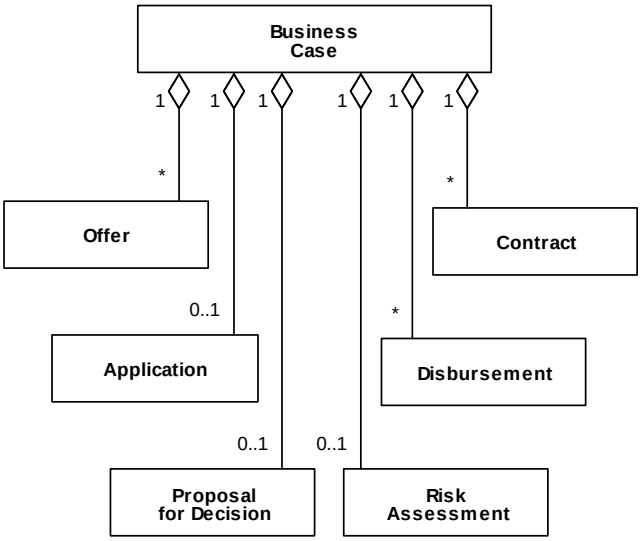

Figure 12 Loan Business Case

7.7 Consequences

+ Life cycle snapshots of complex products are available.
+ Objects are aggregated to the business case.
– Business process objects bring complexity to implementation.

7.8 Related Patterns

- BUSINESS CASE elaborates a complex CUSTOMER PRODUCT having a non-trivial life cycle.
- BUSINESS CASE can calculate fees using BOOK RATE or PARAMETRIC RATE.
- To handle BUSINESS CASE a sophisticated FRONT-END SYSTEM needs to be developed.
- The state of BUSINESS CASE can be displayed in INTEGRATED CUSTOMER VIEW.

7.9 Known Uses

In LP 1 we have designed the system to support the loan application process. The system contains both workflows and life cycle objects. For every loan application subprocess there is a dedicated user screen and the life cycle object behind. Objects are aggregated to the loan business case. Subsequent process entities may be created from the preceding entities with the copy of meaningful values. Documents can be attached to entities (word / pdf documents, scanned documents that are stored in the central repository, etc.). The process model is associated with the business case (although for readability it is drawn in several diagrams).

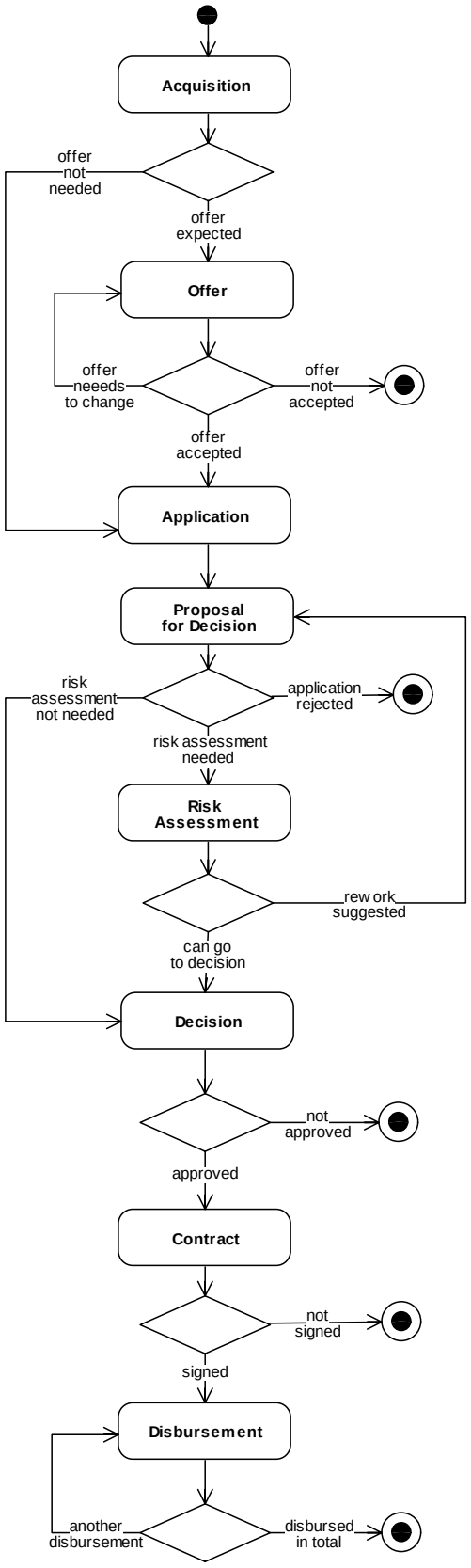

Figure 13 Loan Process Model

8. TRANSACTION TYPES

8.1 Context

There are many types of account transactions such as a cash withdrawal, an ATM withdrawal, a bank transfer, a foreign currency exchange, a loan repayment, a safe deposit box rental payment, and a purchase of travelers' checks. These types of transactions have many features in common, but they also have their own specific features, such as a cash box number for the cash withdrawal, an ATM number and a card number for the ATM withdrawal, a counter account for the bank transfer, the amount of foreign currency for the currency exchange, a loan contract number for the loan repayment, a safe deposit box number for the safe deposit box rental payment, check numbers for the purchase of travelers' checks, etc.

8.2 Problem

How do you represent many types of transactions so that they can be manipulated in a consistent way?

8.3 Forces

- Different types of transactions have their own specifics depending on their purposes, but
- Implementation of each specific type separately requires a lot of time and effort again and again, but
- Different types of transactions also have many things in common and
- The system should constrain types of transactions, but
- The system should also be extensible so that new transaction types can be added.

8.4 Solution

Generalize various types of transactions to the general transaction concept. This transaction has the relationship to the primary account and common attributes such as the transaction amount, the credit/debit indicator, the date of transaction, a narrative. Make a specific transaction type with additional data an extension to this general transaction.

Note that similarly to ACCOUNT TYPES transaction types and transaction codes are distinguished. (Transaction types are from the point of view of modeling and computation while account codes are from the point of view of accounting. Ideally, transaction codes are subtypes of transaction codes. However, this is not always the case.)

Academically, the concept of transaction types can be elaborated further. In [19] I proposed to view bank transactions as transformations in three-dimensional space: the location of the money (e.g. the account, the cash box) **x** the form of the money (e.g. the cash, the electronic money, the travelers' checks) **x** the exchange rate (cash/electronic x buy/sell). Based on this, the meta-level is designed defining transaction types with their constraints. Nevertheless, this proposal was too late to be implemented and it is not a pattern so far ☹.

8.5 Example

Figure 14 shows the similar approach to ACCOUNT TYPES (Figure 6). Fundamental relationships such as the relationship to `Account` are associated with the superclass, i.e. with `Transaction`. Specific relationships, such as the relationship to the bank card are represented on the subclass level. `Transaction Type` represents the UML power type. `Transaction Codes` are separated from `Transaction Types`.

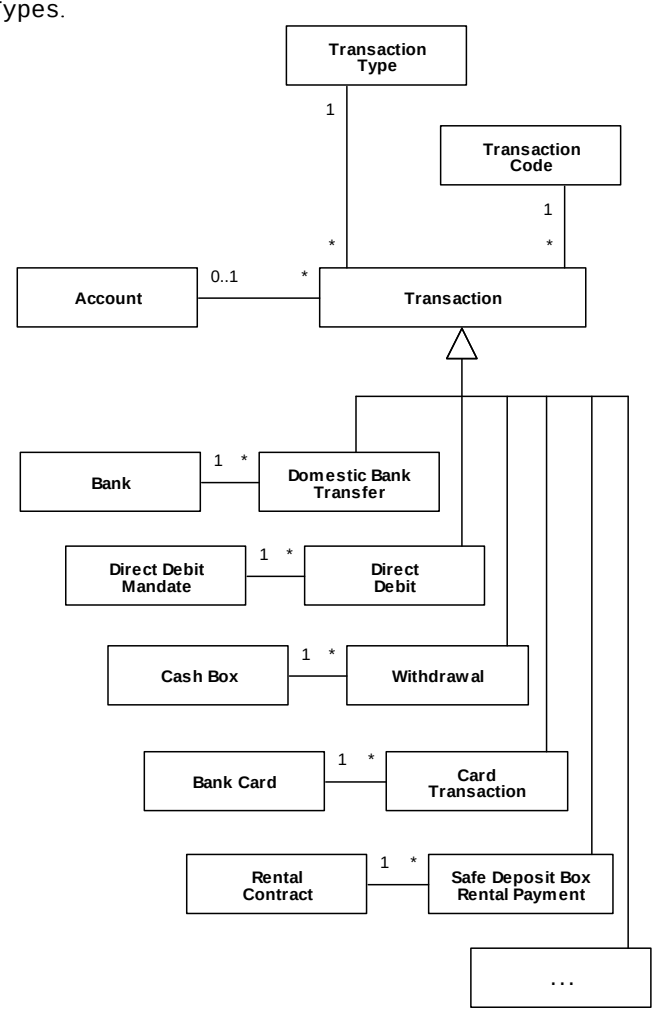

Figure 14 Transaction Types

8.6 Consequences

+ Different types of transactions have the same fundamental representation and can be manipulated in a similar way.
+ Specifics of a transaction type are stored in one place.
+ New transaction types can be added later on.
− Implementation is less efficient as there is a need to compound information from two objects.

8.7 Related Patterns

- TRANSACTION TYPES elaborates the TRANSACTION pattern.
- TRANSACTION TYPES can be associated with specific ACCOUNT TYPES. However, there is not one to one correspondence between an account type and a transaction type.
- TRANSACTION TYPES or transaction codes are the main parameters for ACCOUNTING RULES.
- TRANSACTION TYPES or transaction codes define fee types in ASSOCIATED FEE.
- TRANSACTION TYPES or transaction codes can be parameters for BOOK RATE and PARAMETRIC RATE.

8.8 Known Uses

In CBS 1 transactions are stored in one table. They are distinguished by the account code attribute. Additional is stored in the narrative attribute.

In TS 2 we defined several extension tables to transactions to store information on multicurrency transactions, travelers' checks, commemorative coins, etc.

9. ACCOUNTING RULES

9.1 Context

Transactions are input for the double-entry accounting. This accounting depends not only on the transaction code but other parameters as well. For instance, the accounting of an ATM withdrawal transaction depends on the card network (Visa, MasterCard, etc.), whether it is domestic or foreign, and in the case of domestic withdrawal whether the ATM is owned by this bank or not. If the ATM is owned by the bank it has its own account in the bank general ledger, etc.

9.2 Problem

How do you map transactions to general ledger entries?

9.3 Forces

- Mapping of transactions to general ledger entries depends on transaction types, but
- This mapping depends on other specific parameters and
- Accounting rules in the bank may be the subject of change.

9.4 Solution

Define declarative accounting rules to map transactions to accounting. Make the structure of the accounting rules specific to a business subdomain since these rules depend on various specific business parameters (e.g. the structure of accounting rules for card transactions is different from the structure of accounting rules for factoring).

9.5 Example

Figure 15 shows the model for accounting rules for card transactions. The accounting rule depends on (the 'left side' of the rule):

- Card Network, e.g. a mirror of 'nostro' account for Visa is different from a mirror of 'nostro' account for MasterCard.
- Customer Affiliation, e.g. the ATM withdrawal of the bank customer is the entry to a different account from the ATM withdrawal of a customer of another bank.
- Transaction Type, e.g. the counter account of the withdrawal from ATM of that bank is the ATM account.
- Source of Transaction, e.g. internal ATM withdrawal (i.e. using ATM owned by the bank) is treated differently from the withdrawal done by this customer abroad using an ATM of another bank.

The 'right side' of the rule represents meta-entries (`Card accounting rule entry`) to accounts. This can be either a specific account (e.g. a mirror of Visa 'nostro' account) or accounts that are found dynamically based on attributes of card transactions (e.g. the customer account, the account of specific ATM, etc.).

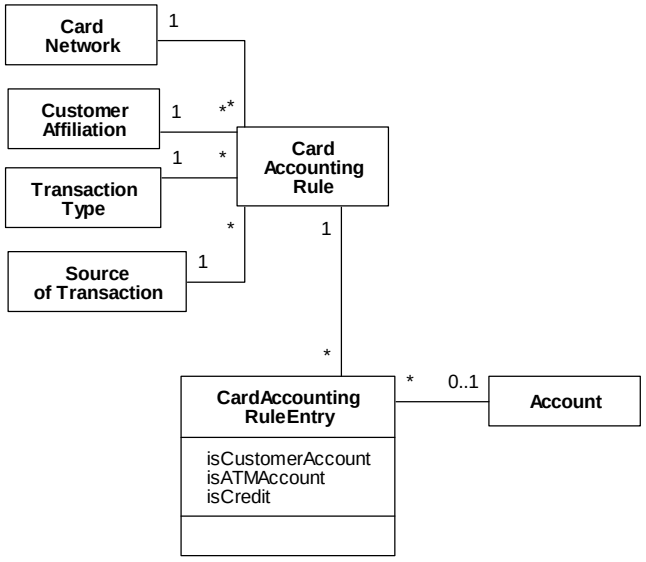

Figure 15 Accounting Rules of Card Transactions

9.6 Consequences

+ Mapping of transactions to general ledger accounts is mostly declarative (not buried in the programming code) and therefore is readable to business experts.
+ Declarative mapping is flexible to 'standard' changes of accounting rules in the bank.
– The mapping rules are simple so that they are not resistant to fundamental change in the accounting rules.

9.7 Related Patterns

- The daily balancing (see DAILY BALANCE) can launch ACCOUNTING RULES.
- ACCOUNT TYPES and TRANSACTION TYPES are many times parameters of ACCOUNTING RULES. More frequently, account codes and transaction codes are those parameters instead.
- FRONT-END SYSTEM can allow defining ACCOUNTING RULES.

9.8 Known Uses

In Bank 3 the existing system for processing card transactions uses a simple proprietary language to map transactions to general ledger entries, based on a set of rules. The left side of the rule refers to the transaction type, the transaction category (on-us, domestic, foreign), and the card type. The right side of the rule specifies accounts (symbolic names) in CBS 1 and parameters of their entries.

10. ARCHIVED HISTORICAL TRANSACTIONS

10.1 Context

Customers do a huge number of transactions. Despite the fact that transactions are indexed it continually slows down the system.

10.2 Problem

How do you manage historical transactions effectively??

10.3 Forces

- There are a huge number of transactions every year and this slows down the system, but
- The bank needs to store transactions for many years due to government regulations, but
- Customers mostly check transactions for the previous month or two, but
- Sometimes they need to search for the transactions further back.

10.4 Solution

Introduce an archive for transactions from the final accounts of the previous year. Archive transaction of the previous year only after several months of the current year has passed. Normally, allow customers to search for transactions in the non archived transactions. Provide a tool to search for transactions in the archive as well, but make this extra effort for the customer so that he uses this option rarely.

10.5 Variants

1. If possible (not slowing down the system) transactions may be archived after more than a year or two.
2. If the transaction archive is too big it can be split to several archives; each one for a specific year.

10.6 Example

Figure 16 shows two entities for transaction. The `Transaction` entity represents non-archived transactions while the `Archived Transaction` entity represents archived transactions. Both entities are associated with the same `Account` entity.

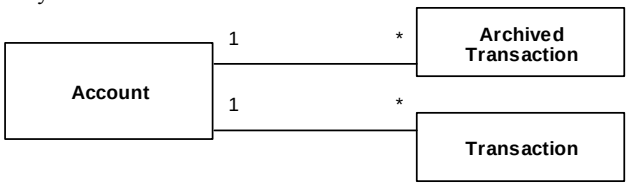

Figure 16 Archived Transactions

10.7 Consequences

+ Most of the search operations are effective as they look up in a reduced number of transactions.
+ It is possible to search for historical transactions, too.

– An extra tool is needed to search for historical transactions.

10.8 Related Patterns

- ARCHIVED HISTORICAL TRANSACTIONS extend the TRANSACTION pattern.
- Archiving transactions in ARCHIVED HISTORICAL TRANSACTIONS implies archiving ASSOCIATED FEEs.
- FRONT-END SYSTEM and INTEGRATED CUSTOMER VIEW can display historical transactions.

10.9 Known Uses

In Bank 3 the standard CBS 1 mechanism was recoded by developers. Transactions are stored in files on a yearly basis. The files have the same names with the suffix of the year.

11. CUSTOMER SESSION

11.1 Context

When the customer comes to the bank he often makes more than one transaction. For instance, he can pay a bill, make several bank transfers, make cash withdrawals from two accounts. After he performs several transactions he may realize he has not enough money on the account and he would like to decrease the amount of one of the previous transactions. Furthermore, he appreciates getting one receipt.

11.2 Problem

How do you implement several transactions of the same customer?

11.3 Forces

- Every transaction is individual and needs to be treated this way, but
- Transactions may have something in common, e.g. they decrease the same account and
- The customer would like to get one receipt.

11.4 Solution

Introduce the concept of customer session. Keep transactions transient within the session until they are finally approved by the customer. Calculate the subtotals after every transaction to provide the 'big picture' for customer.

The concept of customer session differentiates 'banking transactions' from 'software transactions'. Here, there is one 'software transaction' composed of one or several 'banking transactions'. To avoid confusion the term 'transaction' refers to the 'banking transaction' all over the paper.

11.5 Example

Figure 17 shows the simplified sample of customer session. The `Customer Session` instance is created for `Customer` (standing in front of the counter[3] or on the phone). This `Customer` may not be the `Account` owner but he needs to be the entitled person to all `Accounts` within the session. `Customer Session` includes several `Transactions` being prepared and finally executed.

[3] In general, there can be several `Customers` 'in front of the counter' and in `Customer Session`, e.g. several entitled persons of `Account`. We omit it here for simplicity.

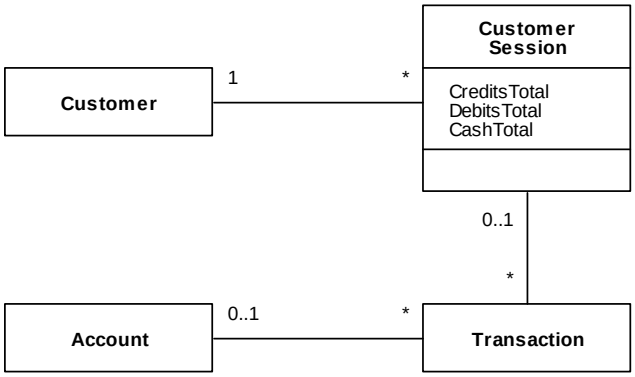

Figure 17 Customer Session

11.6 Consequences

+ Transactions are brought to accounts individually.
+ The customer is happy to have a better control on his transactions.
+ The concept of customer session may be utilized further, e.g. for the purpose of statistics.
– It requires some effort to implement the concept and the communication with the transaction engine of the core banking system.

11.7 Related Patterns

- CUSTOMER SESSION provides ACCOUNTs that can be manipulated.
- Within CUSTOMER SESSION its TRANSACTIONs and their ASSOCIATED FEEs are created and manipulated.
- CUSTOMER SESSION needs to calculate and display AVAILABLE BALANCE. It should restrict further transactions when the available balance is to overdraft the minimum balance.
- CUSTOMER SESSION can be provided by FRONT-END SYSTEM.

11.8 Known Uses

The TS 1 tool includes the mechanism of customer session, including a browser of transactions and subtotals. Unfortunately, transactions are not kept transient, but every transaction is sent to host immediately after it is entered. In the project in Bank 2 substantial effort was put into extending the mechanism so all transactions can be sent to the core banking system as a unit after the final transaction.

12. BANK DAYS

12.1 Context

Normally, banks are closed on weekends and public holidays. Traditionally, daily balances are not calculated on these days. Even on working days, (most) bank branches close their service in the afternoon or in the evening. When customers arrive home the number of electronic transactions decreases rapidly. Batch jobs in the bank can be launched.

12.2 Problem

How do you balance job calculations and calendar days?

12.3 Forces

- Daily balances are calculated on working days only and

- Batch jobs in the bank may be launched some time after most branches are closed, but
- Due to 24x7 internet banking when daily balances reflect calendar days, they should be postponed after midnight, but
- Postponing is a waste of time.

12.4 Solution

Introduce the concept of bank days to the system. Provide a dedicated procedure for the close of the day. It may not match the close of the calendar day.

12.5 Example

Bank days are represented as instances of a specific entity.

12.6 Consequences

+ Batch jobs are not restricted by calendar hours.
+ Customers are not affected too much as most of their transactions are done before the close of the bank day.
– Some night owls may be surprised to see their before midnight transactions with the date of the following day. (This has an impact e.g. on exchange rates, etc.)

12.7 Related Patterns

- The BANK DAYS pattern specifies days of DAILY BALANCE, AVAILABLE BALANCE and ARCHIVED HISTORICAL BALANCES.
- FRONT-END SYSTEM and INTEGRATED CUSTOMER VIEW can display the time span of BANK DAYS.

12.8 Known Uses

In CBS1 there is a specific database table for bank days. Bank days can be specified specific to countries and location.

In TS 1 there are dedicated procedures for closing the day.

13. HIERARCHICAL BALANCING

13.1 Context

There are various types of balancing: balancing the cash box, balancing the branch, balancing customer accounts, etc. Every balancing should be executed at a proper time. For instance, the cash box balancing should be performed when a cashier finishes his duty period and hands back the cash box to the vault (or to another cashier); branch balancing should be executed shortly after the branch is closed, and customer accounts are balanced at the bank close of day. Some balancing depends on other balancing, e.g. the branch balancing should be executed only after all cash boxes of the branch are balanced, the general ledger balancing should be launched only after all branches are balanced. However, some branches may be opened overnight.

13.2 Problem

How do you coordinate various types of balancing?

13.3 Forces

- Due to security, every balancing should be executed shortly after there are no more associated transactions expected that day[4], but
- Some balancing depends on other balancing, but
- There is no guarantee that balancing of accounts which is required for other balancing is finished on time.

13.4 Solution

Define hierarchical balancing: balancing of the certain group of accounts is dependent on completed balancing of other groups of accounts. The hierarchy of groups may have several layers. Use forced balancing based on these layers: if the balancing of the lower level group has not been started and completed to the time set, the higher level balancing procedure can launch the lower level balancing procedure.[5] The higher level balancing procedure should send messages to people responsible at a specified time that lower level balancing is required.

13.5 Consequences

+ Every balancing can be launched at the proper time.
+ If the lower level balancing was not started when required it is started automatically by the upper level procedure.
– In forced balancing, balancing of certain accounts (e.g. cash boxes) may be artificial and accounts need to be rebalanced later.

13.6 Related Patterns

- HIERARCHICAL BALANCING elaborates the method how DAILY BALANCE is calculated.
- The BANK DAYS pattern specifies days for HIERARCHICAL BALANCING.
- FRONT-END SYSTEM can display accounts that should be rebalanced due to forced balancing.

13.7 Known Uses

In TS 1 there is a virtual hierarchy of close of days. The branch close of the day requires a finish of the close of the day of all tellers. The bank close of the day requires finish of the close of the day of all branches. If they are not finished they use a forced close of the day on unfinished tellers or branches.

14. ASSOCIATED FEE

14.1 Context

A transaction can have one or several fees. For instance, the cash deposit transaction can have fees such as the standard fee for a transaction ('accounting entry fee'), the exchange rate fee in the case of multicurrency transaction, the fee when the number of coins exceeds the limit, etc. Many times fees are not accounted on-line but using the batch processing in the end of statement period (monthly). Often, ordinary customers and even junior tellers have problems understanding the correspondence between transactions and fees in the account statement.

14.2 Problem

How do you clarify fees so that they can be understood by bank employees and customers?

14.3 Forces

- Fees are separated from transactions for computational and accounting reason, but

[4] There are some other circumstances for certain balancing, e.g. a shift of the person responsible, a suspicion of error, etc.

[5] When 'forced balancing' was executed a manual check is required later on (e.g. a cash box is manually rebalanced) to avoid inconsistencies.

- Most of the fees are outcomes of transactions.
- The transaction type may have several fees in general, but
- Only some of them may apply for the particular transaction instance.

14.4 Solution

Make an explicit reference from the fee to its associated transaction and show it in the account statement. Define meta-relationships between fee types and transaction types / transaction codes so that it is clear which fees can be applied for which types of transaction.

14.5 Variants

Also 'periodic fees', i.e. fees associated with the product and not associated with any transaction (e.g. monthly account fee) can be changed to transaction fees. These fees can be associated with a job 'transaction' that calculated them.

14.6 Example

In Figure 18 both the meta-level and the operational level are outlined. The meta-level restricts `Fee Types` that can be applied for transactions of the `Transaction Code` type. The operational level stores `Fees` that were actually applied for the specific `Transaction`. For every `Fee` there is an explicit reference to its `Transaction`. `Fees` applied in the operational level should 'obey' restrictions defined in the meta-level.

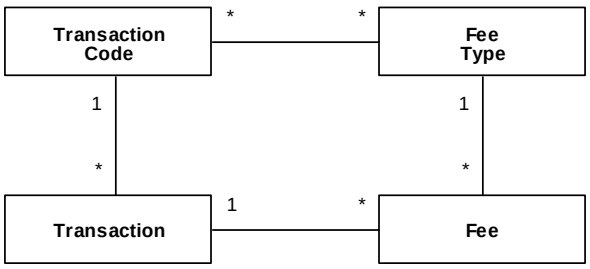

Figure 18 Associated Fee

14.7 Consequences

+ Fees are kept separate from transactions.
+ Fees are associated with their associated transactions and are easier to understand.
+ The fee instance level can be inspected against (and controlled by) the fee meta-level.

14.8 Related Patterns

- Fees are associated with TRANSACTIONS.
- TRANSACTION TYPES or transaction codes define fee types in ASSOCIATED FEE.
- ASSOCIATED FEES are included in DAILY BALANCE and AVAILABLE BALANCE.
- ASSOCIATED FEES are displayed and manipulated within CUSTOMER SESSION.
- Archiving transactions in ARCHIVED HISTORICAL TRANSACTIONS implies archiving ASSOCIATED FEES.

14.9 Known Uses

In CMS 1 we designed the fee instance table associated with the transaction instance table. The fee instance contains also a reference to the fee type record. On the meta-level there is a mapping table between fee types and transaction types.

15. BOOK RATE

15.1 Context

Periodically, interest rates and fee rates change due to the market.

15.2 Problem

How do you handle constantly changing interest rates and fee rates?

15.3 Forces

- In ASSOCIATED FEE the type was separated from the fee so the amount can be changed easily, but
- Sometimes there is a need to recalculate the fee in the past and
- It may be risky to change amounts on the close of the last day of the validity of old fee rates, but
- Introducing new fee types leads to fee type explosion.

15.4 Solution

Separate the fee rates / interest rates[6] from their types. The rate may be dependent on the product and other parameters such as an access channel. Define the period of validity for values so that both historic rates can be kept and future rates prepared.

We showed [1] how fees can be calculated using the STRATEGY [7] design pattern. Unfortunately, this was not implemented and it is not a banking pattern so far.

15.5 Example

Figure 19 outlines the separation of `Fee Rate` from `Fee Type`. Fee Rate instances form the timing queue with possible 'holes' but no overlaps. `Fee Rate` may depend on `Product Item`. (For instance, in bank cards `Fee Rate` may depend on the type of card: the ATM withdrawal fee for the ATM card is different from the credit card fee.) `Fee Rate` may also depend on other parameters. In the diagram, the dependency on `Channel` is shown. (For instance, `Fee Rate` for a bank transfer transaction is different when the transaction is executed manually by a branch teller than when the transaction is performed by the customer using on-line banking.) The `Calculation Method` class represents an enumeration of possible methods for calculating fee: the absolute value, the percentage of value, the percentage with the minimum and the maximum absolute value, the layered value (such as for Western Union money transfer), etc.

15.6 Consequences

+ Historical rates are kept.
+ There is no need to change fee codes after the bank publishes new rates.
– The book rate introduces one more level of indirection.
– Periods of validity complicate retrieval of values (SQL statements).

[6] We use the general term of the book rate for both the book of interest rates and the book of fee rates.

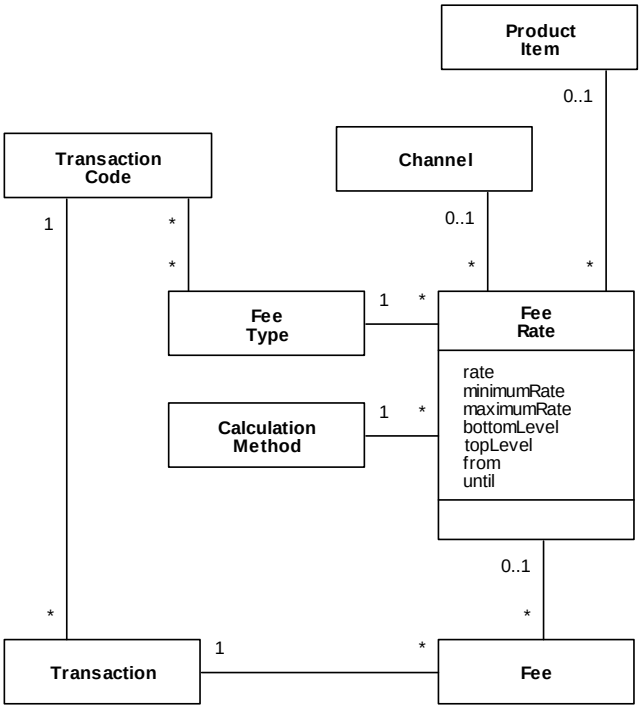

Figure 19 Fee Rate

15.7 Related Patterns

- BOOK RATE extends the fee type in ASSOCIATED FEE.
- BOOK RATEs include interest rates. They are used in calculating interests based on DAILY BALANCE.
- BUSINESS CASE can calculate fees using BOOK RATE.
- TRANSACTION TYPES or transaction codes can be parameters for BOOK RATE.
- BOOK RATE can be a part of RATE PACKAGE.
- PARAMETRIC RATE adds a specific method how to calculate a fee when declarative representation of BOOK RATE is too complicated.

15.8 Known Uses

In CMS 1 we designed and implemented the fee type amount table containing fee rates including historical rates. This table refers to the fee type table and the fee instance refers to the fee type amount. Fee type amount depends on the product.

16. PARAMETRIC RATE

16.1 Context

The book rate may not just depend on the type and the product but other parameters as well. For instance, the interest rate may depend on the amount of money the customer has in his account or the fee rate may depend on various optional services of the product.

16.2 Problem

How do you approach various dependencies of the book rate?

16.3 Forces

- The book rate may depend on various conditions and
- It would be useful to represent these conditions declaratively, but
- These conditions may be of very different nature.

16.4 Solution

Represent parameters that have impact on the book rate. Implement PARAMETRIC FUNCTION [5] to calculate the actual book rate.

16.5 Consequences

+ Parameters for calculating book rates are represented declaratively and can be changed.
– Sophisticated constrains are buried in the programming code.

16.6 Related Patterns

- PARAMETRIC RATE is a specific case of PARAMETRIC FUNCTION [5].
- PARAMETRIC RATE adds a specific method how to calculate a fee when declarative representation of BOOK RATE is too complicated.
- ACCOUNT TYPES, ACCOUNT CONTRACT types, CUSTOMER PRODUCTs and TRANSACTION TYPES are many times parameters of PARAMETRIC RATE.
- BUSINESS CASE can calculate fees using PARAMETRIC RATE.

16.7 Known Uses

In CMS 1 the calculated fee depends not only on the type of fee and the product but also some optional services, e.g. the urgency of issuing. We designed the relationship table between the fee type amount and card service tables and implemented the parametric function to calculate the fee.

17. RATE PACKAGE

17.1 Context

Customers are formed in various categories: standard private customers, affluent private customers, VIP customers, companies, etc. Important customers may be given higher interest rates and lower fee rates. Usually, lower rates are set not only for periodic fees but for transaction fees as well. They may be set differently for different transaction types.

17.2 Problem

How do you handle different rates for different customers?

17.3 Forces

- Rates should be as universal as possible, but
- Special rates may be offered to important customers, but
- Handling rates individually for customers is not transparent and it is time consuming.

17.4 Solution

Introduce the concept of rate package to the system. The rate package includes rates for the set of products (account types) and/or transaction types. Assign the customer the package for his category or CUSTOMER PRODUCT by default. The account officer may change the package for another package for the special customer.

17.5 Example

The concept of the rate package is outlined in Figure 20 and Figure 21. Figure 20 shows the meta-level (i.e. classes for defining the fee package) while Figure 21 is devoted to the operational level. It needs to be emphasized that the diagram is a

sample diagram rather than the complete solution. (Fee packages are too complex to be represented using just a few classes.)

`Fee Package` is composed of several `Fee Package Items`. Each `Fee Package Item` represents either `Fee Rate` for `Product Item` or `Fee Rate` for the specific `Transaction Code`. These `Fee Rates` can be:

1. Equal to null (not specified). For instance, the fee for issuing the ATM card is included in the fee for package or the package includes the fee for the specified number of ATM withdrawal transactions.
2. Reduced. For instance, the fee for the second ATM card (ATM card for a spouse) is reduced or fees for on-line banking transactions are reduced.
3. Standard. For instance, bank transfers performed on the branch have standard fee rates. (Alternatively, this may not be specified in the fee package and the 'fee engine' should apply the general standard fee for all `Fee Types` not covered by the fee package.)

The association between `Customer Product` and `Fee Package` restricts fee packages that are permitted for this product.

In the meta-level other restrictions may be specified that are not shown here, for instance, dependency of `Fee Packages` to the category of customer, and others.

In the operational level (Figure 21) when creating `Contract on Product` also `Fee Package` from permitted `Fee Packages` for `Customer Product` needs to be selected.

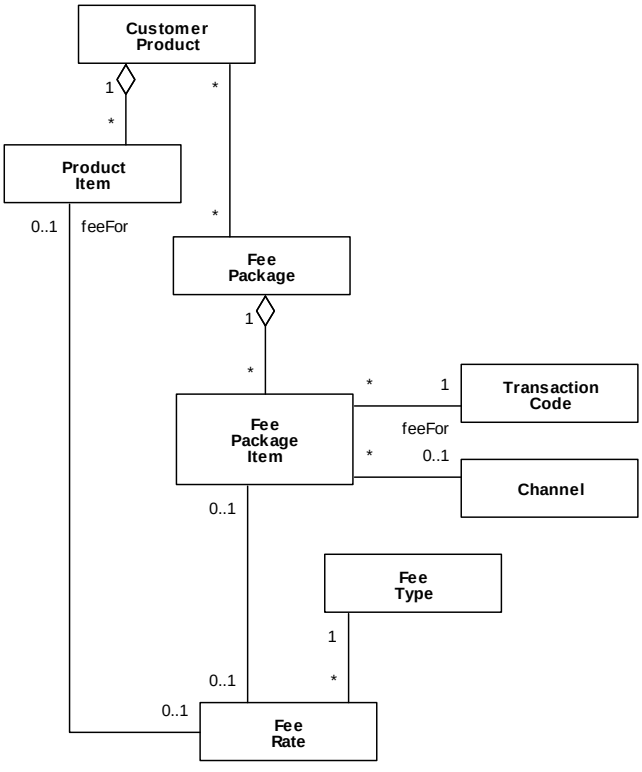

Figure 20 Fee Package (meta-level)

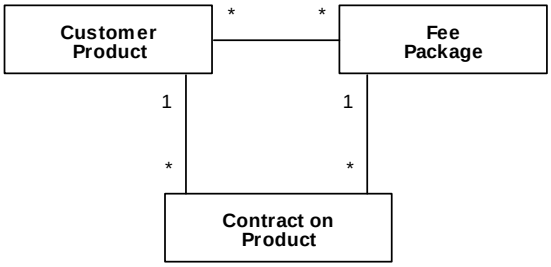

Figure 21 Contract Using Fee Package

17.6 Consequences

+ The customer can be assigned the set of rates depending on his importance.
+ The number of rate packages is restricted, which makes the mechanism transparent and easy to maintain.
– Rate packages add complexity to the system.

17.7 Related Patterns

- RATE PACKAGE aggregates several BOOK RATEs.
- BOOK RATEs usually include interest rates that are calculated based on DAILY BALANCE.
- An instance of RATE PACKAGE can be assigned to ACCOUNT CONTRACT, CUSTOMER PRODUCT or BUSINESS CASE. However, this is a rare case as RATE PACKAGE is usually assigned to a customer.

17.8 Known Uses

The extension to CMS 1 implemented by a third party includes fee packages. The fee package includes fee rates for periodic account fees and certain transaction types.

18. FRONT-END SYSTEM

18.1 Context

In the past, banking systems were designed to be off-the-shelf systems with the aim of supporting the complete portfolio of bank products. Due to the market, customer products have become more and more complex and there is a continuous need to add more functionality to banking systems. Furthermore, these systems were implemented using old technology, e.g. COBOL, RPG, etc. and it is not easy to extend them quickly enough.

18.2 Problem

How do you extend the banking system quickly enough to support new customer products and processes?

18.3 Forces

- There is an operating banking system providing much functionality, but
- This system requires permanent extensions due to the market, but
- It is not easy to extend the system quickly enough as it is implemented using old technology and
- With extension of many submodules the system is becoming too big, but
- The system is not easy to supersede with a new technology system as it contains too many sophisticated extensions developed for specific needs of the bank.

18.4 Solution

Keep the system as the core banking system that supports fundamental functionality. Develop or buy off-the-shelf a modern technology system that extends the particular functionality of the core banking system. Install this system as the front-end system of the core banking system. This front-end system stores most of data in its repository and sends only a limited subset of its data to the core banking system.

18.5 Example

The core banking system contains a loan submodule. In this submodule basic attributes of a loan contract such as a loan type, a loan amount, an interest type, a maturity date, and a fee are represented. The loan submodule is integrated with the general ledger submodule using ACCOUNTING RULES. In the general ledger interests of loan accounts are calculated. From the general ledger values such as a current principal, a principal already paid, an interest already paid, an outstanding principal and an outstanding interest can be obtained.

The loan submodule, however, does not support the loan approval process having activities such as preparing offers for the customer, preparing a proposal for decision, a risk assessment, a decision, preparing contract documentation, preparing a disbursement and others (see also the BUSINESS CASE pattern). To support these activities a sophisticated front-end system can be developed using a workflow engine. This front-end system is integrated with the core banking system. When the loan approval process ends and the loan is approved the front-end system can automatically create the loan record, the loan account and fees in the core banking system.

The front-end system can also support post approval processes such as a loan monitoring and a work-out. Here, the integration with the core banking system also includes reading data from the loan submodule and the general ledger.

18.6 Consequences

+ Useful functionality of the core banking system is preserved.
+ There is no need to reimplement the sophisticated functionality into a new core banking system.
+ Front-end system brings new functionality.
+ Front-end system can be extended quite easily as it is relatively small and implemented in the new technology.
– The strategy usually leads to several front-end systems (each system for a particular subdomain) that are of different architectures and implemented in different technologies.

18.7 Related Patterns

Related patterns to FRONT-END SYSTEM depend on the type of front-end system. Here we assume that the front-end system is a teller system. In such a way most of other patterns described in this article can be used.

- FRONT-END SYSTEM allows manipulating with ACCOUNTS and TRANSACTIONS.
- FRONT-END SYSTEM should display DAILY BALANCE, AVAILABLE BALANCE, HISTORICAL BALANCES and HISTORICAL TRANSACTIONS.
- ACCOUNT TYPES facilitate displaying of products in FRONT-END SYSTEM.
- FRONT-END SYSTEM should allow an easy manipulation of ACCOUNT CONTRACTS and CUSTOMER PRODUCTS.
- A sophisticated FRONT-END SYSTEM needs to be developed to handle BUSINESS CASE.
- FRONT-END SYSTEM can provide CUSTOMER SESSION.
- FRONT-END SYSTEM can display the time span of BANK DAYS.
- FRONT-END SYSTEM can display accounts that should be rebalanced due to forced balancing in HIERARCHICAL BALANCING.
- INTEGRATED CUSTOMER VIEW is a sophisticated FRONT-END SYSTEM to several source systems.

18.8 Known Uses

This is a wide spread architecture that can be found in banks nowadays.

Bank 3 uses CBS 1 to support general ledger, customer accounts, loans and other basic functionality. Apart from this, about half dozen of major front-end systems are used: a teller system, an on-line banking system, a home banking system, a card management system, a private banking system and others. We ourselves implemented the CMS 2 card management system as CBS 1 provides only a very limited functionality for bank cards.

Currently, we are implementing the LPS 1 loan workflow system.

We implemented the TS 2 teller system for bank 1.

19. INTEGRATED CUSTOMER VIEW

19.1 Context

Several front-end systems are used in a bank. These systems extend the functionality of the core banking system. They are essential to support complex processes in the bank. Unfortunately, these front-end systems were developed by different vendors and are not integrated. When a customer comes to a branch, the account officer does not have a global picture of this customer quickly enough and needs to search for customer products in several systems. This limits cross-selling opportunities.

19.2 Problem

How do you restore the integrated customer view using the ‘quick win’ strategy?

19.3 Forces

- The account officer needs one modern bank system to work with customers and their products, but
- There are several front-end systems and
- The complete integration of all these systems (using SOA, etc.) is too costly and in the distant future.

19.4 Solution

Provide the CRM-like (portal-like) integrated view of the customer. The account officer can see the main attributes of all customer products in one or several screens. On demand detail information on the product chosen can be provided or the user is redirected to the system responsible. Product maintenance remains in the individual systems.

This is just the ‘light’ version of integration but from the business point of view it is the revolutionary step. Further levels of integration, although desirable, require much more effort and less ‘visible’ effects.

The integrated customer view can be augmented further with the other CRM features, e.g. recording interactions with

customers, the Q&A repository, search criteria for marketing campaigns, etc.

19.5 Consequences

+ The account officer quickly has the global picture of the customer.
+ The solution is not too expensive.
– It is not a complete integration of systems. Detailed information and the opening of new products or changing existing ones needs to be done specifically in the individual systems.

19.6 Related Patterns

- INTEGRATED CUSTOMER VIEW is a sophisticated FRONT-END SYSTEM to several source systems.
- INTEGRATED CUSTOMER VIEW is the INFORMATION PORTAL [10] type of integration.
- INTEGRATED CUSTOMER VIEW should display DAILY BALANCE, AVAILABLE BALANCE, HISTORICAL BALANCES and HISTORICAL TRANSACTIONS.
- ACCOUNT TYPES, ACCOUNT CONTRACTS and CUSTOMER PRODUCTS facilitate displaying of products in INTEGRATED CUSTOMER VIEW.
- The state of BUSINESS CASE can be displayed in INTEGRATED CUSTOMER VIEW.
- The system for INTEGRATED CUSTOMER VIEW can provide CUSTOMER SESSION.
- The system for INTEGRATED CUSTOMER VIEW can display the time span of BANK DAYS.

19.7 Known Uses

There are a lot of CRM tools and CRM systems implemented.

In the CRM 1 project we used a CRM tool with powerful integration facilities to host systems. The tool also provides certain features for cross–selling, recording interaction with customers and marketing campaigns. We implemented the integrated customer view providing view on data from ten systems and the management of customer data.

20. ACKNOWLEDGMENTS

I would like to thank my shepherd Uwe Zdun and members of the working group at PLoP 2008 moderated by Linda Rising for suggestions that makes this paper much more precise. I am also thankful to the shepherds and reviewers of my previous papers especially Andy Longshaw, Allan Kelly, Ed Fernandez, and Kyle Brown. Andy was my shepherd at EuroPLoP 2005 and taught me how to write domain patterns so that they are understandable to people outside the domain. Allan helped me to be more precise. Ed was my first shepherd and introduced me to the world of patterns. Kyle encouraged me at PloP'2000 to write pattern languages instead of small patterns.

21. REFERENCES

[1] Blstak, P., L. Sesera. Model Driven Software Development Using Patterns. DATAKON 2005 (in Slovak).

[2] Buschmann, F. et al. Pattern-oriented Software Architecture – A System of Patterns. J. Willey and Sons, 1996.

[3] Czarnecki, K. and U. W. Eisenecker. Generative Programming. Addison-Wesley, 2000.

[4] Fernandez, E., Y. and Y. Liu. The Account Analysis Pattern. EuroPLoP 2002.

[5] Fowler, M. Analysis Patterns: Reusable Object Models, Reading, MA: Addison-Wesley, 1997.

[6] Fowler, M. Patterns of Enterprise Application Architecture. Reading, MA: Addison-Wesley, 2003.

[7] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA: Addison-Wesley, 1995.

[8] Hay, D. Data Model Patterns. Conventions of Thought. Dorset House, 1996.

[9] Henney, K. Context Encapsulation. Three Stories, a Language, and Some Sequences. EuroPLoP 2005.

[10] Hohpe, G., B. Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, 2004.

[11] Keller, W. Some Patterns for Insurance Systems. PLoP'98.

[12] Kelly, A. Business Strategy Design Patterns. EuroPLoP 2004.

[13] Kelly, A. A Few More Business Patterns. EuroPLoP 2005.

[14] Sesera, L. A Recurring Fulfillment Analysis Pattern. PLoP 2000.

[15] Sesera, L. Analysis Patterns. (Invited talk.) In: SOFSEM'2000. Lecture Notes in Computer Science series, Vol. 1963, Springer Verlag, 2000.

[16] Sesera, L., A. Micovsky and J. Cerven, J. Data Modeling in Examples. Grada, 2001 (in Czech).

[17] Sesera, L. Hierarchical Patterns: A Way to Organize (Analysis) Patterns. CITSA 2004. Orlando, FL, 2004.

[18] Sesera, L. Obligation-Fulfillment: A Pattern Language for Certain Financial Information Systems. EuroPLoP'05.

[19] Sesera, L. Extendible Banking Model. DATAKON 2007 (in Slovak).