The VCL model of Secure Simple Bank

Nuno Amálio
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi
L-1359 Luxembourg

TR-LASSY-11-10

Version	Date	Description
0.1	01/10/2010	1st release
0.2	30/09/2011	Uses VCL diagrams produced with the Visual Contract Builder Tool.
0.3	21/11/2012	Updates VCL diagrams to reflect changes to the syntax.
0.4	23/09/2013	Slight changes to case study, diagrams and resulting Z.

Table 1: Document Revision History

Contents

Chapter 1

Introduction

The Visual Contract Language (VCL) [AK10a, AKMG10, AK10b, AKM10] expresses software designs. Its approach to modelling is akin to object-oriented modelling. VCL's semantic basis is set theory; design by contract (pre- and post-conditions) is used to model behaviour. VCL is supported by a tool, the Visual Contract Builder (VCB) [AGK11][1].

This document presents the VCL model of the secure simple Bank case study together with the Z specification generated from the VCL model. The next section describes the requirements of this case study. Chapter 2 presents the VCL model. The Z specification that is generated from the VCL model is given in chapter 3.

1.1 Case Study

The *secure simple bank* case study extends the simple Bank case study used in [APS05, Amá07, AKM10, AK10a]. The extension covers the security concerns of *authentication* and *access-control*. The requirements of this system are listed in table 1.1.

Table 1.1: Requirements of the secure simple bank system.

R1	The bank system shall keep information of customers and their Bank accounts. A customer may hold many accounts; an account is held by one customer only.
R2	A customer record comprises a customer number, a name, an address and a type (either *corporate* or *personal*). Each customer has its own unique customer number.
R3	A Bank account shall have an account number, a balance indicating how much money there is in it, and its type (either *current* or *savings*). Each account has its own unique account number.
R4	Accounts of type savings cannot have negative balances.
R5	Customers of type corporate cannot hold savings accounts.
R6	Customers may hold savings accounts provided they also hold a current account with the Bank.

[1] `http://vcl.gforge.uni.lu`

R7	The system shall provide an operation to *create customers records*. This takes as input the customer's name, address and type; the customer number is to be assigned internally by the system.
R8	The system shall provide an operation to *open bank accounts* for some customer. This takes as input a customer number and a type of account; the account number is to be assigned internally by the system.
R9	The system shall provide an operation to *deposit money* into a bank account. This takes as input an account number and an amount to be deposited.
R10	The system shall provide an operation to *withdraw money* from some bank account. This takes as input an account number and an amount to be withdrawn.
R11	The system shall provide an operation to *view the balance of some bank account*. This takes as input an account number and outputs the account's balance.
R12	The system shall provide an operation to *obtain a list of all accounts of some customer*. This takes as input a customer number and outputs the set of accounts numbers corresponding to accounts held in the bank by the customer.
R13	The system shall provide an operation to *view a list of all accounts that are in debt* in the bank. This outputs the set of accounts numbers corresponding to accounts held in the bank that are in debt.
R14	The system shall provide an operation to *delete accounts* from the system. This takes as an input the number of the account to be deleted. A bank account may be deleted provided its balance is 0.
R15	Users have to authenticate themselves prior to opening a system session (*login*), which needs to be closed when they no longer need the system's services (*logout*).
R16	Users have their access to the system suspended if they miss a password for three consecutive times.
R17	There are two kinds of users: *clerks* and *managers*. Managers can execute *create customer records*, *open accounts* and *delete accounts*. Clerks can execute *deposit* and *withdraw*. Both managers and clerks can execute *get balance*, *get customer accounts* and *get accounts in debt*.
R18	A system service or operation may be used provided users have an open session and they have the required permissions to execute task.
R19	The system shall provide a functionality to suspend transactions for security reasons. If transactions are suspended, no deposits and withdrawals are allowed. The system shall allow transactions to be resumed once they have been suspended.

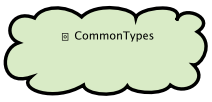
Chapter 2

The VCL Model

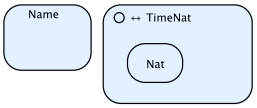
This chapter presents the VCL model of the secure simple Bank case study, which is divided in VCL packages. All diagrams presented here were drawn using the VCB tool. The following presents each packages of the VCL model.

2.1 Package `CommonTypes`

The container package `CommonTypes` encapsulates common structures of the VCL model. Figure 2.1 presents this package's package and structural diagrams. It introduces the set `Name` representing a set of names.



(a) Package Diagram



(b) Structural Diagram

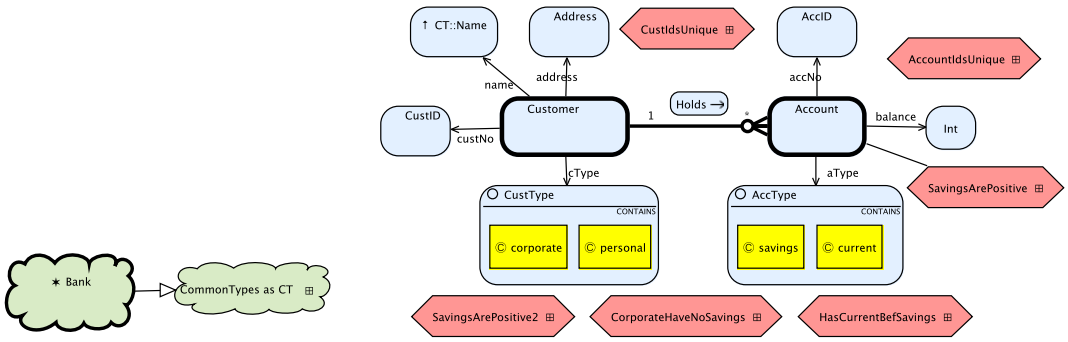
Figure 2.1: Package and structural diagrams of package `CommonTypes`

2.2 Package `Bank`

The ensemble package `Bank` localises the problem domain concern of banking. It introduces structures related with customer and their bank accounts. Figure 2.2 presents the package and structural diagrams of this package. `Bank` is an ensemble package that extends (or incorporates) the package `CommonTypes` (Fig. 2.2a).

2.2.1 Structural Diagrams

The VCL SD of package `Bank` (Fig. 2.2b) is as follows:



(a) Package Diagram (b) Structural Diagram

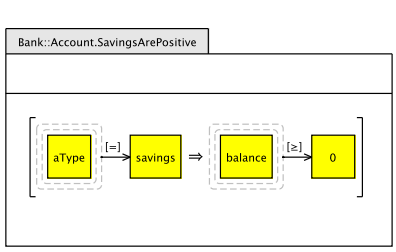
Figure 2.2: Package and structural diagrams of package `Bank`

- Values sets `CustID` and `AccID` represent the set of customer and account identifiers. Blob `Address` represents a set of postal addresses.
- Blobs `CustType` and `AccType` define customer and account types, respectivelly. They are defined by enumeration. `CustType` has elements `corporate` and `personal`; `AccType` has elements `savings` and `current`.
- Domain sets `Customer` and `Account` represent the main problem domain concepts (requirement $R1$). Property edges `name`, `cType`, `address` and `custNo` hold information of customers (Requirement $R2$); `accNo`, `balance` and `aType` hold information of accounts (Requirement $R3$).
- Relational edge `Holds` relates customers and their accounts. UML-style multiplicity constraints say that a customer may have many accounts and that an account is held by one `Customer` (Requirement $R1$).
- Several assertions represent constraints on the state space of the system. `SavingsArePositive` is local; it represents requirement $R4$. Remaining assertions are global: `CorporateHaveNoSavings` (represents requirement $R5$) and `HasCurrentBefSavings` (represents requirement $R6$).

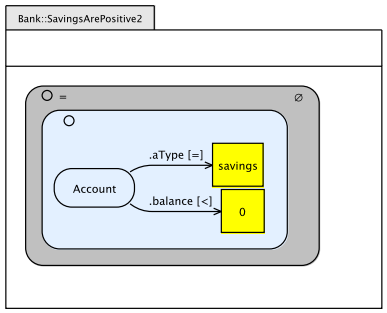
2.2.2 Assertion Diagrams of Invariants

The invariants identified in the SD of Fig. 2.2b are defined using VCL Assertion Diagrams (ADs) in Figs. 2.3 and 2.4.

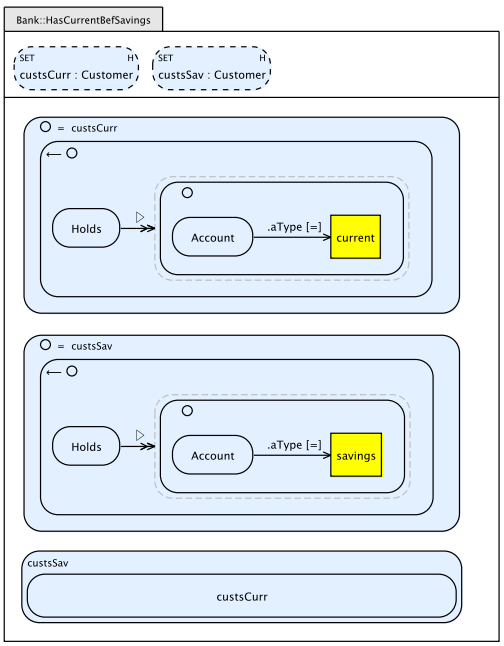
Figure 2.3a presents the AD of local invariant `SavingsArePositive` (of set `Account`), which says that savings accounts have positive balances. The declarations compartment is empty; no extra declarations of names are required to describe the assertion. The predicate compartment expresses the required constraint as an implication formula: if the account's type is savings then the balance must be greater or equal than 0. The same



(a) `SavingsArePositive`



(b) `SavingsArePositive2`



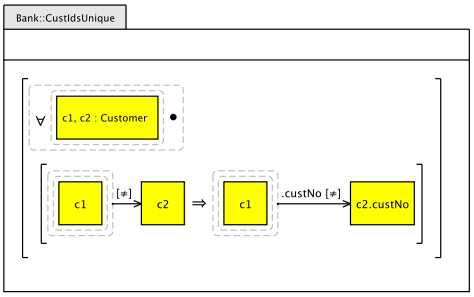
(c) `HasCurrentBefSavings`

Figure 2.3: ADs of invariants `SavingsArePositive`, `SavingsArePositive2` and `HasCurrentBefSavings`

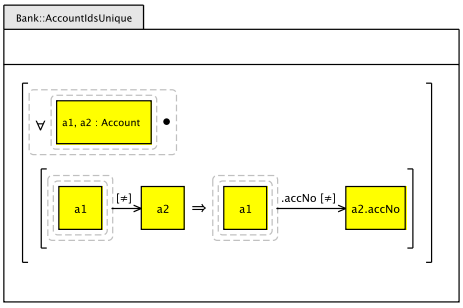
invariant is expressed globally using sets in Fig.2.3b; this defines the set of accounts with negative balances and then says that this set must be empty (outer set is shaded to say that it is empty).

The global invariant `HasCurrentBefSavings` is described in AD of Fig. 2.3c. The AD defines two hidden variables to hold the set of customers with current accounts (`custsCurr`) and set of customers with savings accounts (`custsSav`) and then says that the latter is a subset of former. Sets `custsCurr` and `custsSav` are defined by extracting the domain of the relation `Holds` restricted on the range to either current or savings accounts (the domain operator is $\leftarrow$ and the range restriction operator is $\rhd$).

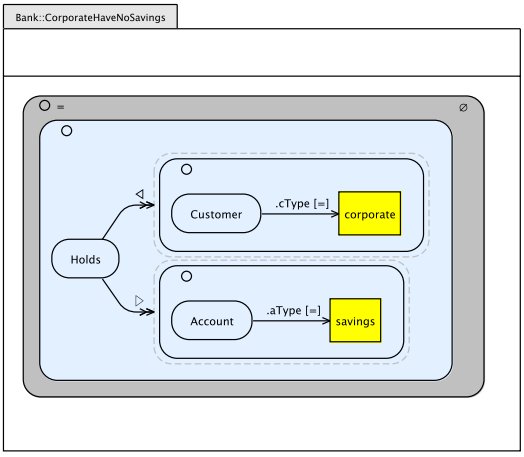
Figure 2.4c presents AD of global invariant `CorporateHaveNoSavings`. Again, the declarations compartment is empty. The predicate compartment describes this assertion by saying that the set of of all corporate customers with savings accounts (outer set) is empty (shading says that some set is empty), which gives the required meaning. This involves two sets that are then used to constrain the relation `Holds`. The first set restricts `Customer` to those objects whose property `cType` has value `corporate`. The second set restricts `Account` to those objects whose property `aType` has value `savings`. The two property edge modifiers are then used to restrict the relation `Holds` according to these two sets.



(a) `CustIdsUnique`



(b) `AccountIdsUnique`



(c) `CorporateHaveNoSavings`

Figure 2.4: ADs of invariants `CustIdsUnique`, `AccountIdsUnique` and `CorporateHaveNoSavings`

2.2.3 Behaviour

The behaviour diagram (BD) of package `Bank` is given in Fig. 2.5. It introduces the following operations:

- The update operation `New` of set `Customer`, which creates customer objects (a constructor, denoted by symbol $\mathbb{N}$).
- The local operations of set `Account`. This includes the update operation `New` to create `Account` objects, the update operations `Deposit` and `Withdraw` to deposit and withdraw money from some account (update or modifier operations, symbol $\mathbb{U}$), the delete operation `Delete` to delete account objects (symbol $\mathbb{D}$), and the observe operation `GetBalance` to retrieve the current balance of some account.
- The global update operations `CreateCustomer` to create customer records in the system (requirement $R7$) , `OpenAccount` to create new customer accounts (requirement $R8$), `AccDeposit` to deposit money onto some account (requirement $R9$), `AccWithdraw` to withdraw money from account (requirement $R10$) and `AccDelete` to delete money from some account (requirement $R14$).

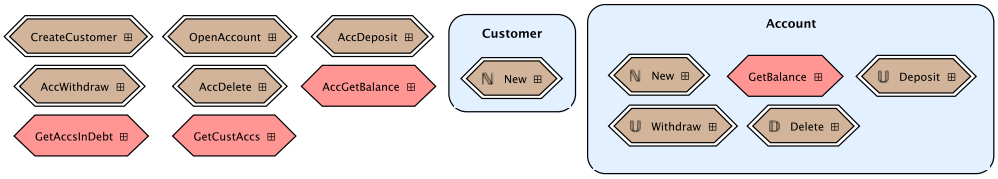


Figure 2.5: Behavioural diagram for the `Bank` package

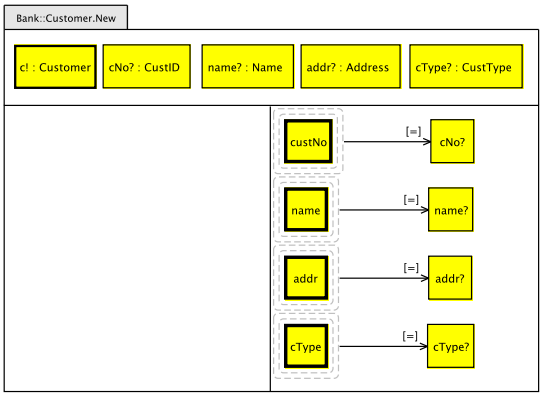


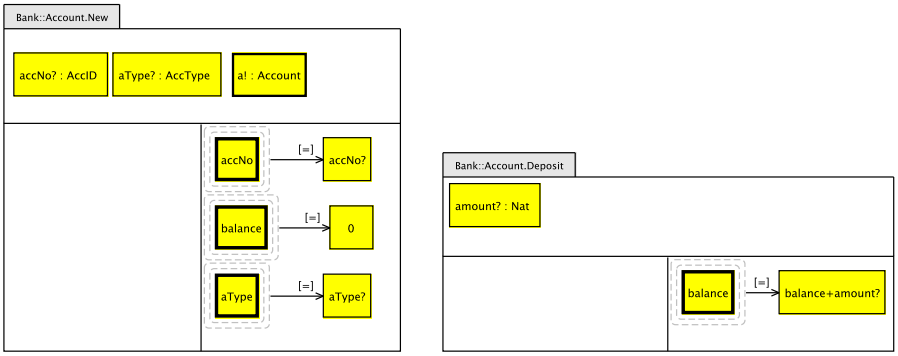
Figure 2.6: Contract diagram of operation `New` of set `Customer`

- The global observe operations `AccGetBalance` to get the balance of some account (requirement $R11$), `GetAccsInDebt` to retrieve the accounts that are in debt (requirement $R13$), and `GetCustAccs` to retrieve the accounts of some customer (requirement $R12$).

We now describe the operations of the operations of the BD of Fig. 2.5 using contract and assertion diagrams.

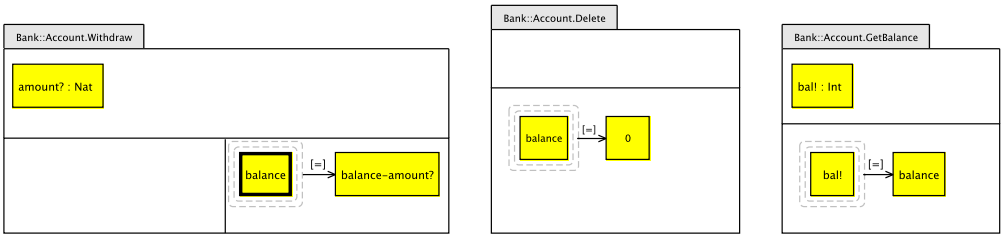
The CD of operation `New` of set `Customer` (Fig. 2.6) declares the output $c!$ of type `Customer` (marked in bold) to represents the `Customer` object that is returned as a result of the operation. It declares inputs `cNo?` of type `CustID`, `name?` of type `Name`, `addr?` of type `Address` and `cType?` of type `CustType`, which represent, respectively, the identifier, name, address and type of the new customer. The predicate assigns in the post-condition the properties `custNo`, `name`, `address` and `cType` to the declared inputs.

The CD of operation `New` of set `Account` (Fig. 2.7a) declares the output $a!$ of type `Account` (marked in bold), which represents the `Account` object returned as a result of the operation. It declares inputs `accNo?` of type `AccID` and `aType?` of type `AccType`, which represent, respectively, the identifier and type of the new account. The predicate assigns in the post-condition the properties `accNo` to the input `accNo?`, `balance` to 0 and `aType` to the input `aType?`.



(a) `Account.New` (b) `Account.Deposit`

Figure 2.7: Contract diagrams of operations `New` and `Deposit` of set `Account`



(a) `Account.Withdraw` (b) `Account.Delete` (c) `Account.GetBalance`

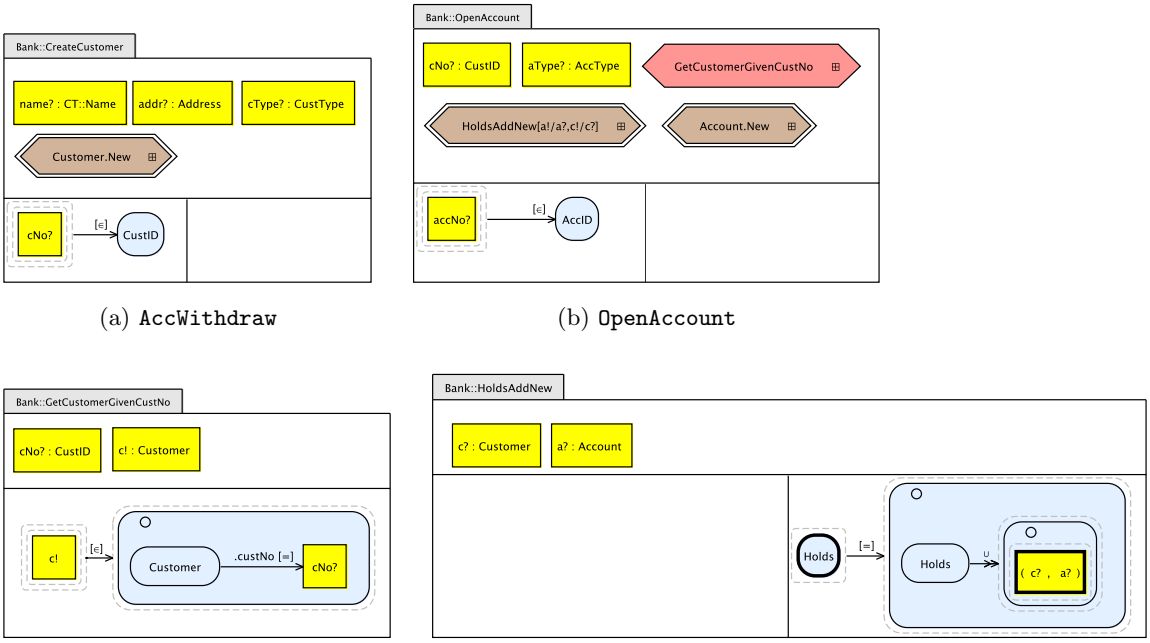
Figure 2.8: Operations `Withdraw`, `Delete` and `GetBalance` of set `Account`

The CD of operation `Deposit` of set `Account` (Fig. 2.7b) declares the input `amount` of the primitive type `Nat` (Natural numbers). The post-condition says that the property `balance` becomes the previous value of `balance` minus the amount given as input.

The CD of operation `Withdraw` of set `Account` (Fig. 2.8a) declares the input `amount` of the primitive type `Nat` (Natural numbers). The post-condition says that the property `balance` becomes the previous value of `balance` added with the amount given as input.

The AD of operation `Delete` of set `Account` (Fig. 2.8b) says that objects may be deleted provided its balance is 0.

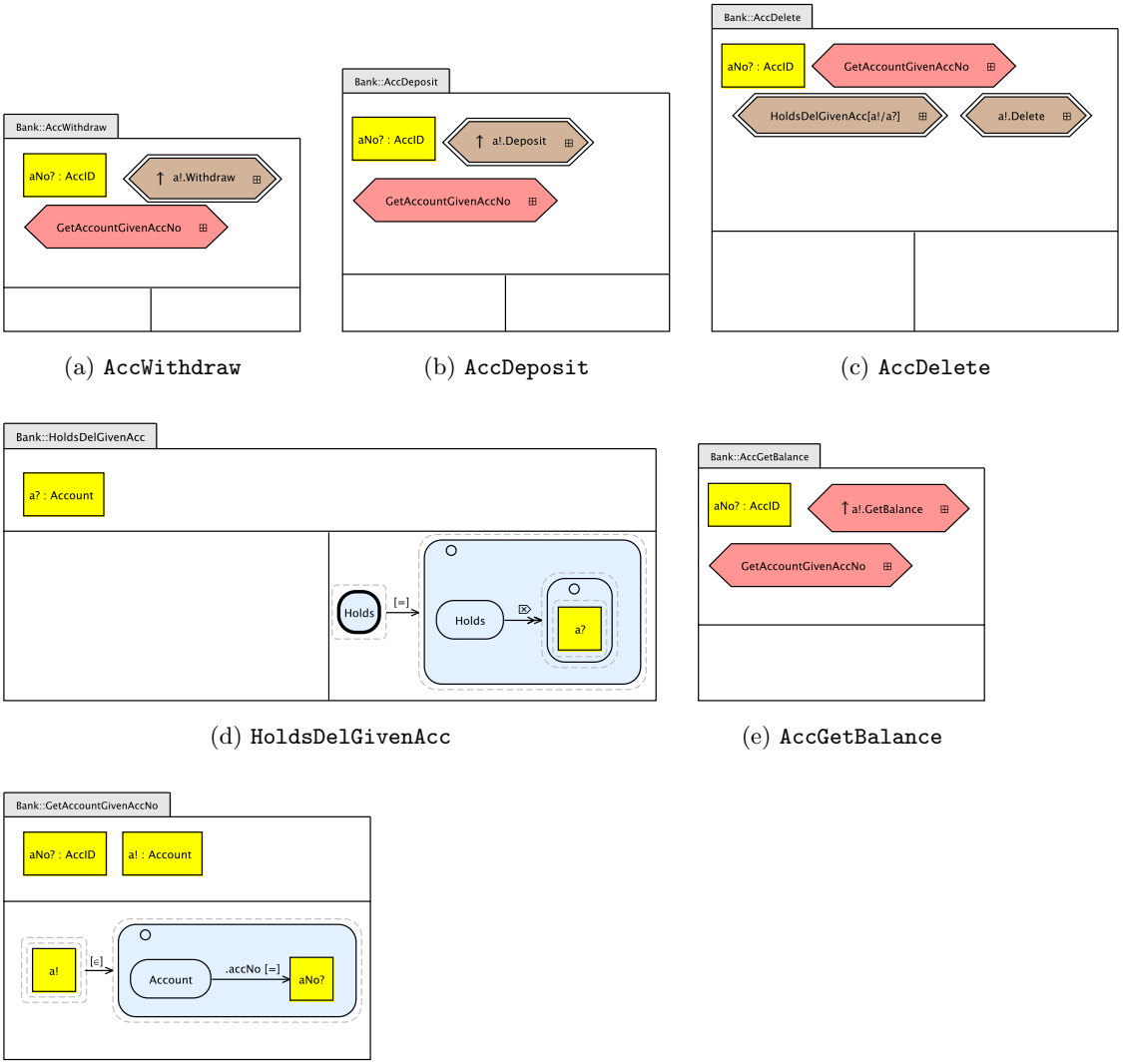
The AD of operation `GetBalance` of set `Account` (Fig. 2.8c) gives the output `bal!` the value of the property `balance`.



(a) `AccWithdraw` (b) `OpenAccount`

(c) `GetCustomerGivenCustNo` (d) `HoldsAddNew`

Figure 2.9: Contract diagrams of global operations `CreateCustomer`, `OpenAccount` and auxiliary operations



(a) `AccWithdraw` (b) `AccDeposit` (c) `AccDelete`

(d) `HoldsDelGivenAcc` (e) `AccGetBalance`

(f) `GetAccountGivenAccNo`

Figure 2.10: Contract diagrams of global operations `AccWithdraw`, `AccDeposit` and `AccDelete` and auxiliary operations

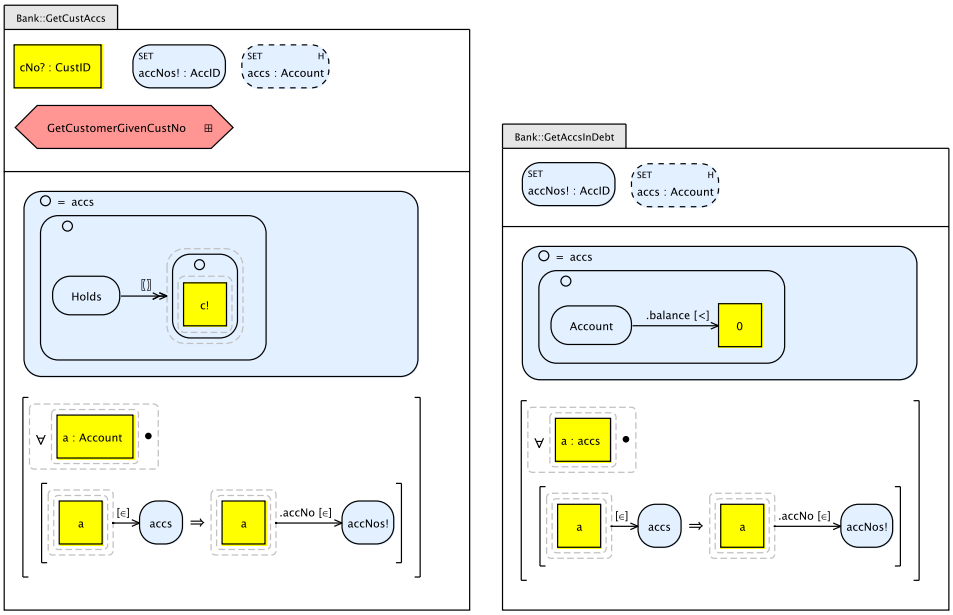
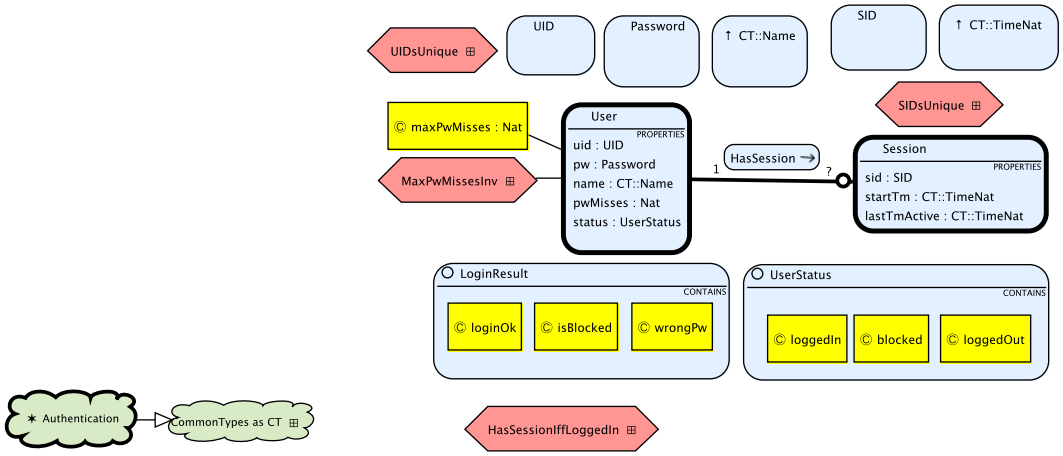


Figure 2.11: Assertion diagrams of global query operations `GetCustAccs` and `GetAccsInDebt`



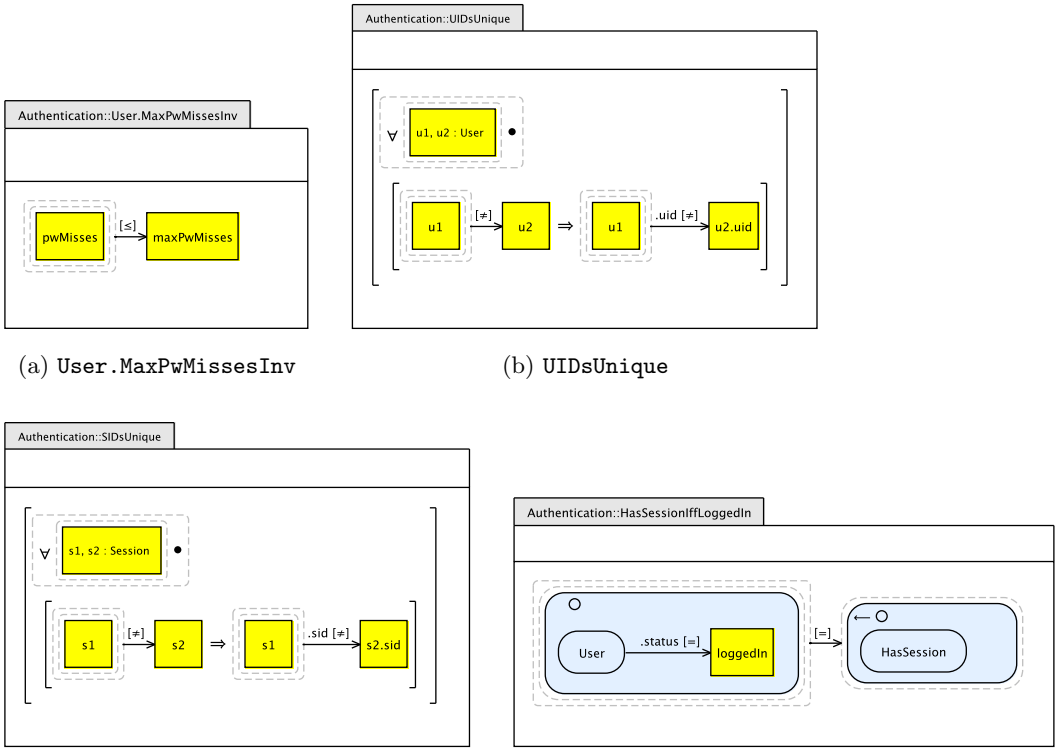
(a) Package Diagram (b) Structural Diagram

Figure 2.12: Package and structural diagrams of package `Authentication`

2.3 Package `Authentication`

The VCL package `Authentication` defines the core of a general solution to the concern of user authentication. This addresses requirement $R15$ of Table 1.1. Package `Authentication` is to be extended by other packages to define authentication-related functionality.

Figure 2.12 presents `Authentication`'s package and structural diagrams. Package `Authentication` extends package `CommonTypes` (Fig. 2.12a). The four invariants introduced in the SD of package `Authentication` (Fig. 2.12b) are given in Fig. 2.13.



(a) `User.MaxPwMissesInv` (b) `UIDsUnique`

(c) `SIDsUnique` (d) `HasSessionIffLoggedIn`

Figure 2.13: Assertion diagrams of package `Authentication`

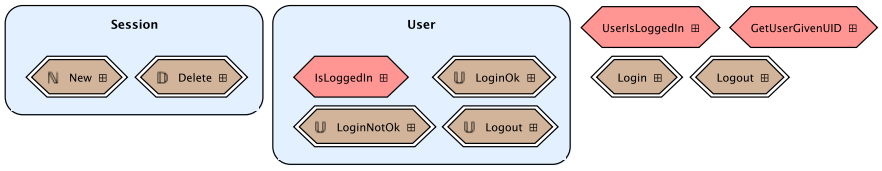
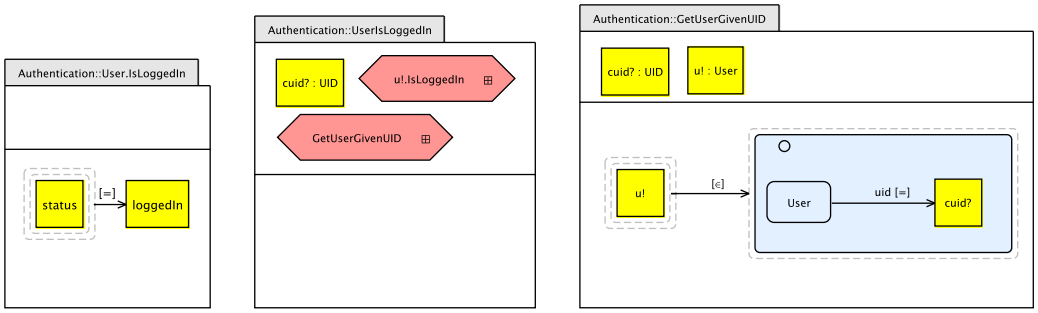


Figure 2.14: Behaviour diagrams of package `Authentication`



(a) Operation `User.IsLoggedIn` (b) Global Operation `UserIsLoggedIn` (c) Global Operation `GetUserGivenUID`

Figure 2.15: Assertion diagrams of observe operations of package `Authentication`

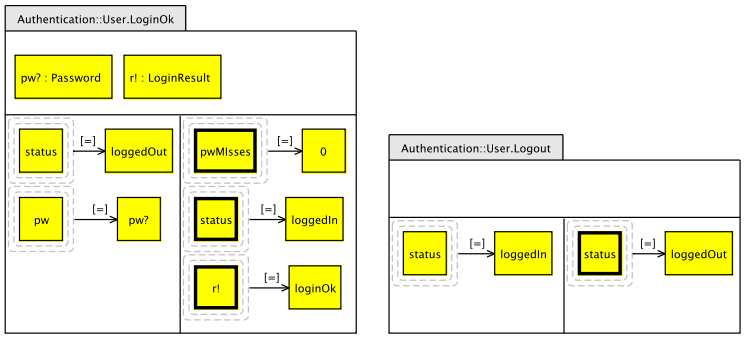


Figure 2.16: Contract Diagrams for local operations `LoginOk` and `Logout` of set `User`.

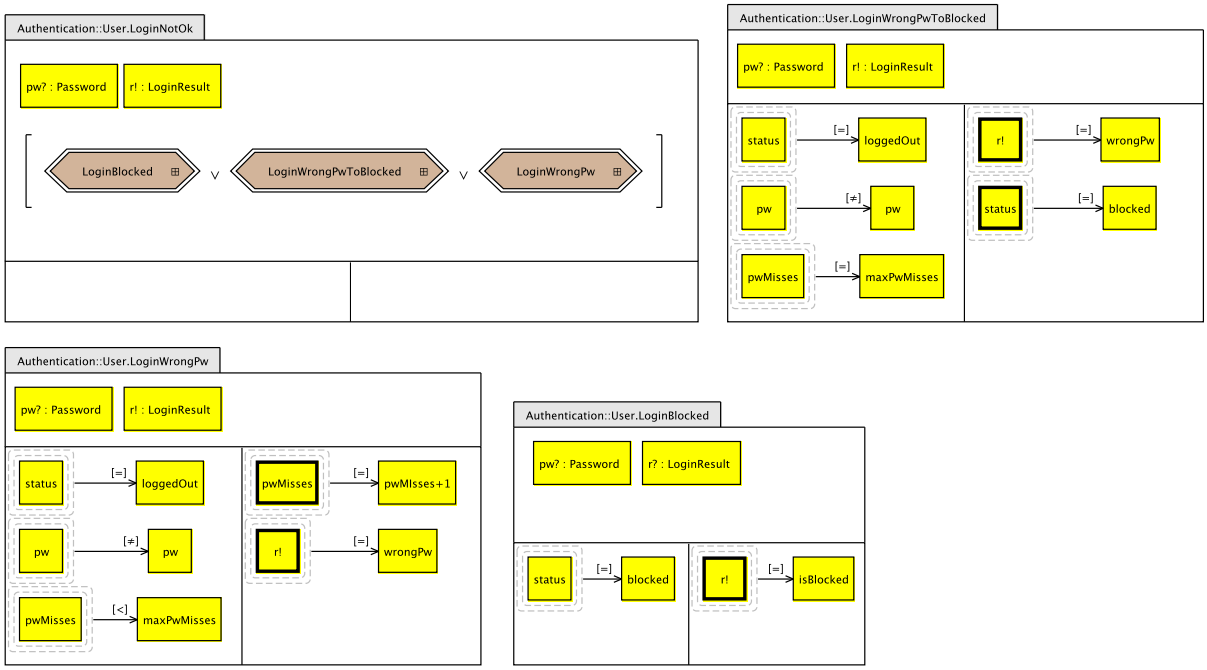


Figure 2.17: Contract Diagrams describing local operation `LoginNotOk` of set `User`

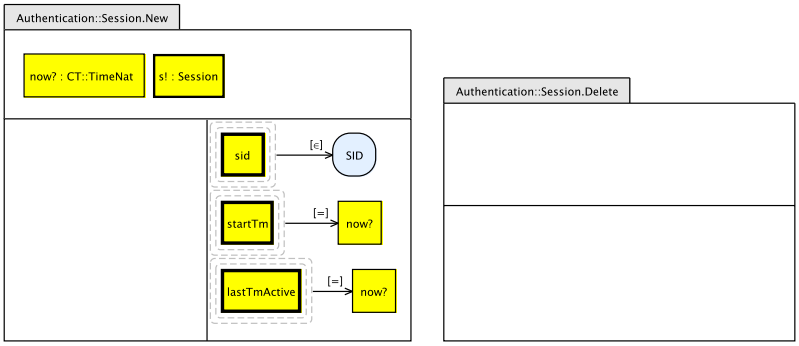


Figure 2.18: Contract Diagrams for local operations `New` and `Delete` of set `Session`.

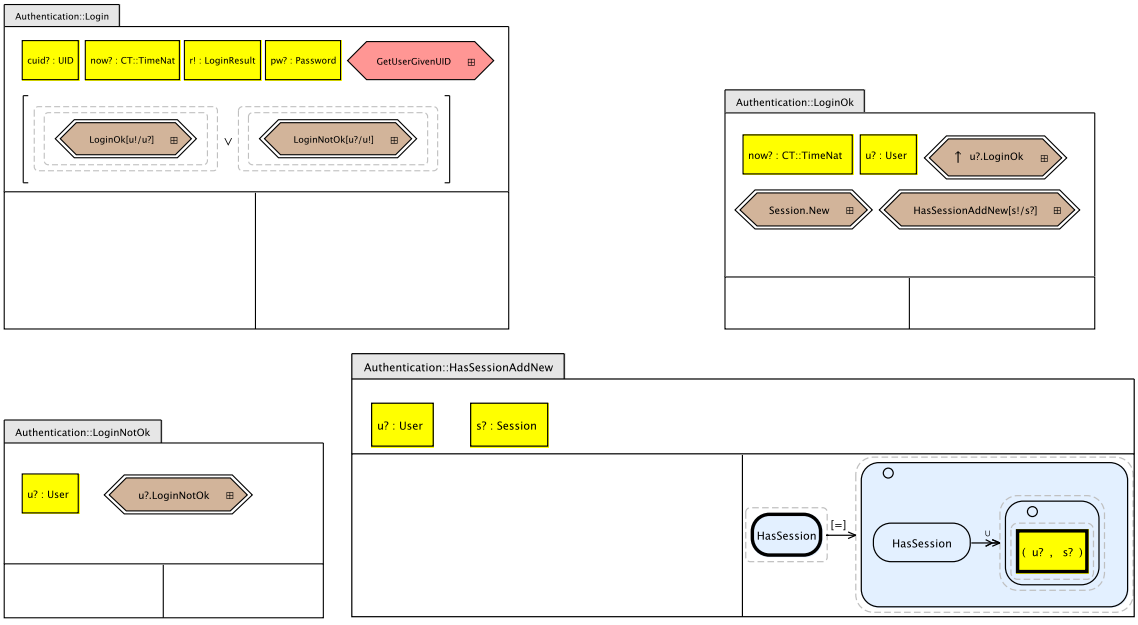


Figure 2.19: Contract Diagrams defining global operations `Login`

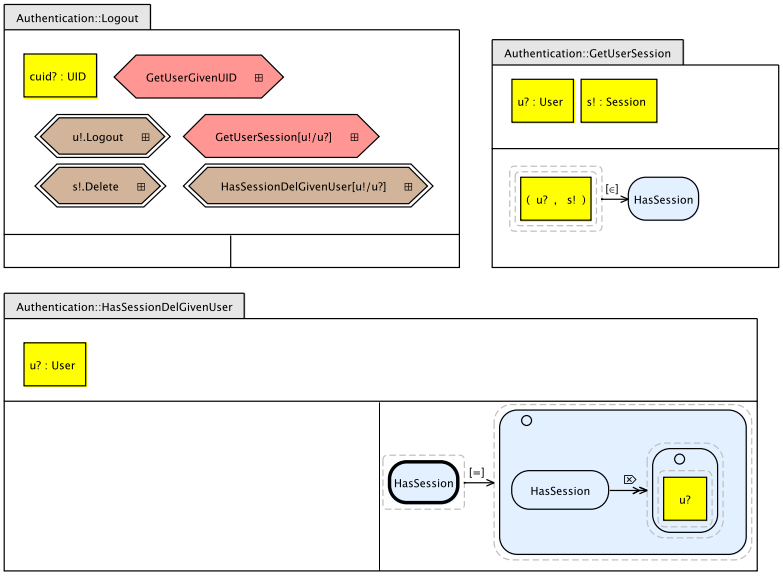
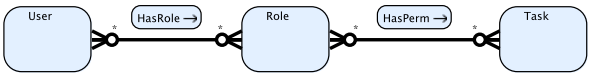


Figure 2.20: Contract Diagrams of global operation `Logout` and auxiliary definitions

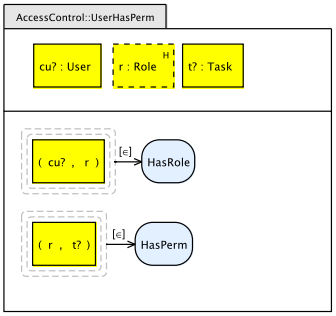
(a) Package Diagram



(b) Structural Diagram

(c) Behaviour Diagram

Figure 2.21: Package structural and behaviour diagrams of package `AccessControl`



(a) Operation `UserHasPerm`

Figure 2.22: Assertion diagrams of global observe operation `UserHasPerm`

2.4 Package `AccessControl`

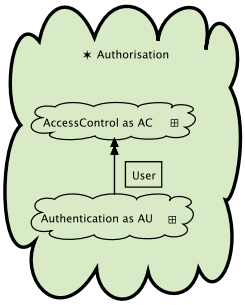
The VCL package `AccessControl` (Fig. 2.21a) defines the core of a general solution to the concern of access control following a rôle-based access control [SCFY96] scheme. This addresses requirement $R16$ of *secure simple Bank* (Table 1.1).

Figure 2.21 presents `AccessControl`'s package, structural and behaviour diagrams. Figure 2.22 presents the AD that defines observe operation `UserHasPerm`.

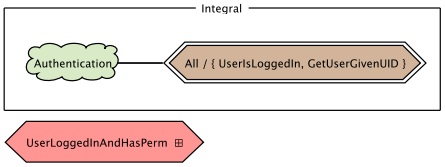
2.5 Package `Authorisation`

VCL Package `Authorisation` puts together the concerns of authentication and acess-control. This is naturally expressed in the package's PD (Fig. 2.23a); the ensemble package `Authorisation` extends both `Authentication` and `AccessControl`.

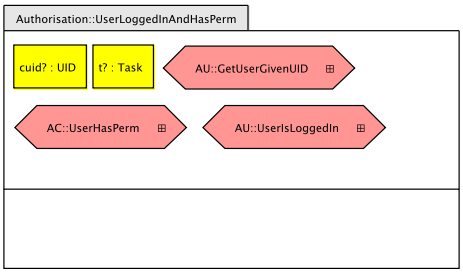
`Authorisation`'s BD (Fig. 2.23b) introduces observe operation `UserLoggedInAndHasPerm`, which checks if some user is logged in the system and has permission to execute some task. This is defined in the AD of Fig. 2.23c, which puts together the operations `UserIsLoggedIn` of package `Authentication` and `UserHasPerm` of package `AccessControl`.



(a) Package Diagram



(b) Behaviour Diagram

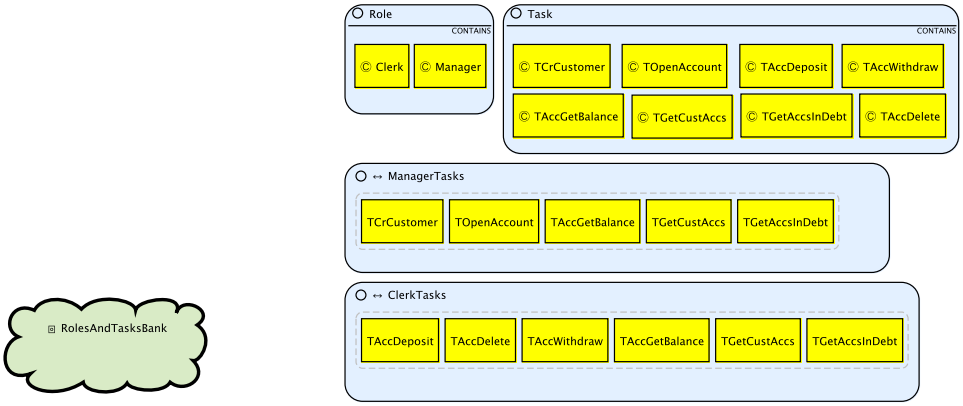


(c) Query operation `UserLoggedInAndHasPerm`

Figure 2.23: VCL diagrams of package `Authorisation`

2.6 Package `RolesAndTasksBank`

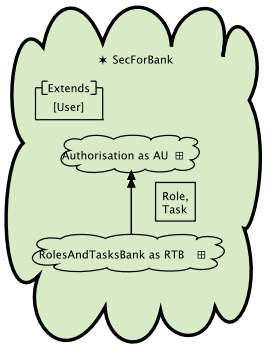
The VCL package `RolesAndTasksBank` makes some customisations concerning *access control*. Package `RolesAndTasksBank` is an ensemble package (Fig. 2.24a). Its SD (Fig. 2.24b) defines the sets `Task` and `Role` according to the needs of the secure simple bank system. In addition, it defines the derived sets `ManagerTasks` and `ClerkTasks` (subsets of `Tasks`), which describe the tasks to be executed by clerks and managers.



(a) Package Diagram (b) Structural Diagram

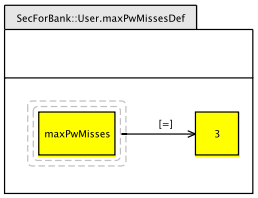
Figure 2.24: Package and structural diagrams of package `RolesAndTasksBank`

2.7 Package `SecForBank`

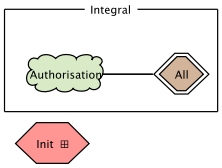


(a) Package Diagram

(b) Structural Diagram



(c) Definition of constant `maxPwMisses`



(d) Behaviour diagram

Figure 2.25: VCL Diagrams of package `SecForBank`

Package `SecForBank` customises generic package `Authorisation` for the purpose of *secure simple bank*.

`SecForBank`'s PD (Fig. 2.25a) says that the ensemble package `Authorisation` extends the packages `Authorisation` and `RolesAndTasksBak`, and that the package `RolesAndTasksBak` overrides `Authorisation` on sets `Role` and `Task`. This basically says that the access control scheme defined in package `AccessControl` (incorporated in `Authorisation`) is to use the sets Role and `Task` from package `RolesAndTasksBak`. The SD (Fig. 2.25b) introduces a definition of the constant `maxPwMisses` defined in package `Authentication` (incorporated in `Authorisation`); the value for this constant is defined in the AD of Fig. 2.25c.

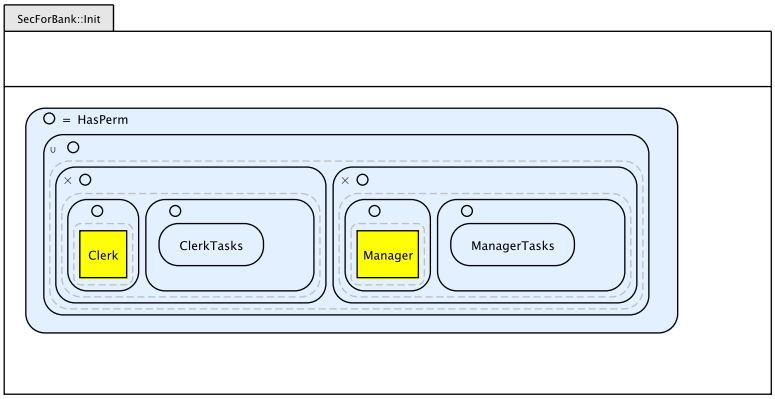
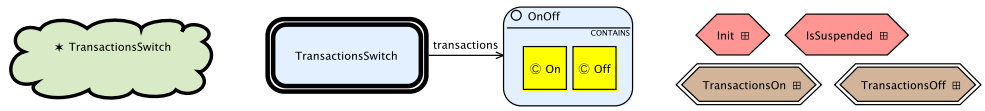
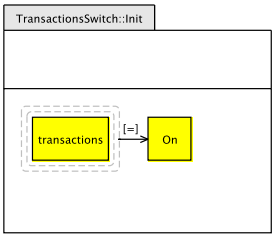


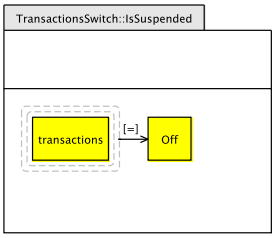
Figure 2.26: Assertion diagram describing `Init`, the initialisation of package `SecForBank`



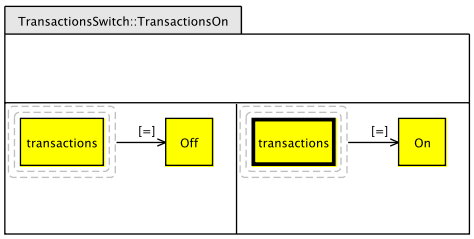
(a) Package Diagram (b) Structural Diagram (c) Behaviour Diagram



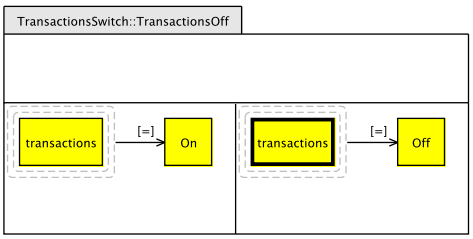
(d) `Init`



(e) `IsSuspended`



(f) `TransactionsOn`



(g) `TransactionsOff`

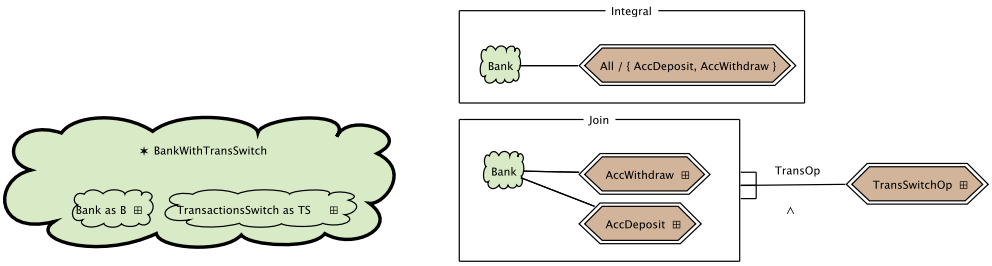
Figure 2.27: VCL diagrams of package `TransactionsSwitch`

The BD of package `SecForBank` (Fig. 2.25d) uses an integral extension to say that all global operations of package `Authorisation` are to be used unaltered in the context of `SecForBank`. In addition, `SecForBank` introduces `Init`, which defines the packages initialisation. `Init` is defined in the AD of Fig. 2.26; this says that the relation-edge `HasPerm` of `AccessControl` is initialised to the union of two sets: (a) the cross product of the set with object `Clerk` and the set `ClerkTasks` (the tasks of `Clerk` are those defined in `ClerkTasks`) and the cross product of the set with object `Manager` and the set `ManagerTasks` (the tasks of `Manager` are those defined in `ManagerTasks`).

2.8 Package `TransactionsSwitch`

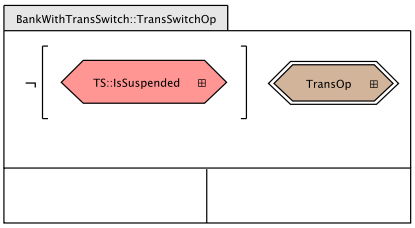
This package defines a switch to turn transactions *on* and *off*. This addresses requirement R17 of *secure simple Bank* (Table 1.1).

The PD of `TransactionsSwitch` (Fig. 2.27a) introduces this ensemble package. The SD (Fig. 2.27a) introduces the package set `TransactionsSwitch` (singleton representing the whole package) and its property `transactions` of type `OnOff` (an enumeration with values `On` and `Off`).



(a) Package Diagram

(b) Behaviour Diagram



(c) CD of Join operation `TransSwitchOp`

Figure 2.28: VCL diagrams of package `BankWithTransSwitch`

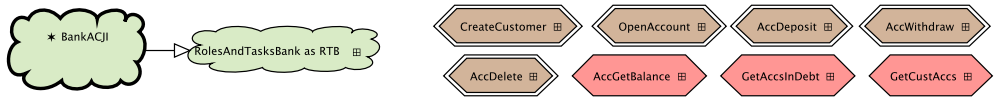
The package's BD (Fig. 2.27c) introduces `Init` (the package's initialisation), the observe operation `IsSuspended` (says whether transactions are suspended), and the updates operations `TransactionsOn` (turns transactions on) and `TransactionsOff` (turns transactions off).

The initialisation (Fig. 2.27d) says that initially transactions are not suspended. Operation `IsSuspended` (Fig. 2.27e) defines a predicate that is true when transactions are suspended. Finally, operations `TransactionsOn` (Fig. 2.27f) and `TransactionsOff` (Fig. 2.27g) turn transactions on and off.

2.9 Package `BankWithTransSwitch`

VCL package `BankWithTransSwitch` adds the transactions switch mechanism to the package `Bank`. This ensures that transactions are executed only if they have not been suspended.

`BankWithTransSwitch`'s PD (Fig. 2.28a) says that the ensemble package `BankWithTransSwitch` extends both `Bank` and `TransactionsSwitch`. The BD (Fig. 2.28b) says that all operations of `Bank` except `AccDeposit` and `AccWithdraw` are to be integrally extended (brought into the new context unaltered), and that the `Bank` operations `AccDeposit` and `AccWithdraw` are to be composed with the contract `TransSwitchOp` (Fig. 2.28c), which adds a precondition to these operations: they may be executed provided transactions are not suspended.



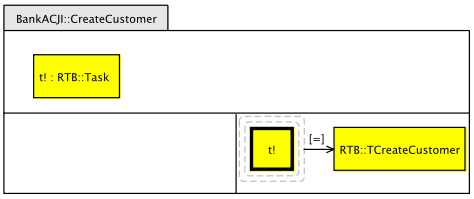
(a) Package Diagram (b) Behaviour Diagram

Figure 2.29: Package and behaviour diagram of package `BankACJI`

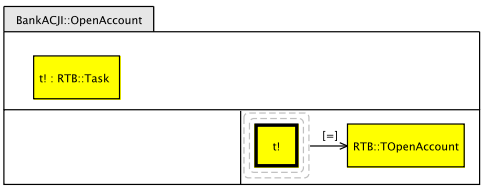
2.10 Package `BankACJI`

This section introduces package `BankACJI` (*Bank Access-Control Join Interface*), which defines the interface of package `Bank` (section 2.2) to the access control concern. This consists of defining an outer interface to be merged with package `Bank` that will enable composition with the access control concern.

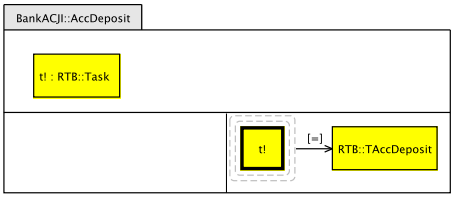
The ensemble package `BankACJI` (Fig. 2.29a) extends package `RolesAndTasksBank`. `BankACJI`'s BD (Fig. 2.29a) comprises the same global operations of package `Bank`. The CD of these operations are defined in CDs of Figs .2.30a to 2.30h.



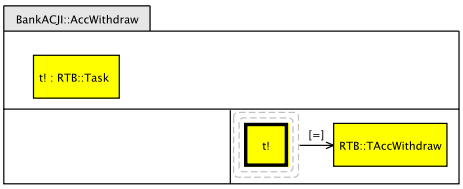
(a) `CreateCustomer`



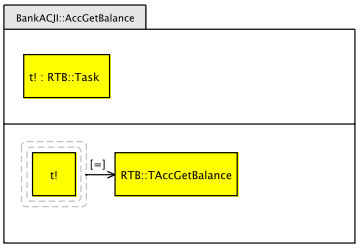
(b) `OpenAccount`



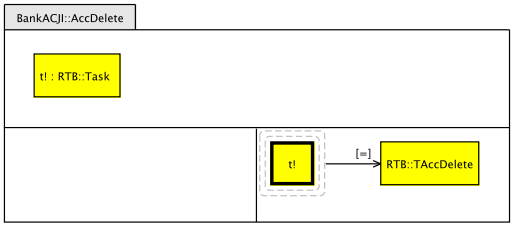
(c) `AccDeposit`



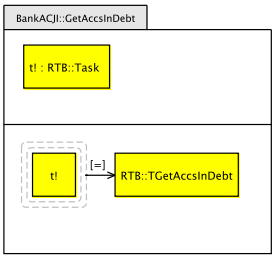
(d) `AccWithdraw`



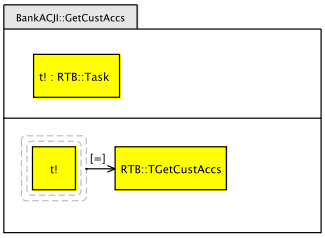
(e) `AccGetBalance`



(f) `AccDelete`

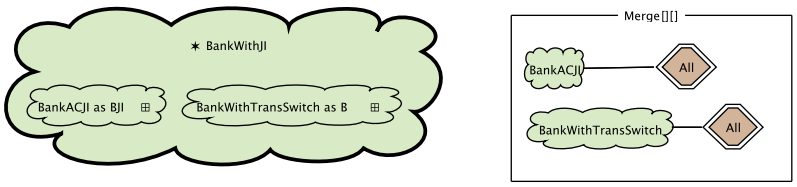


(g) `GetAccsInDebt`



(h) `GetCustAccs`

Figure 2.30: Operations of package `BankACJI`



(a) Package Diagram (b) Behaviour Diagram

Figure 2.31: Package and behaviour diagrams of package `BankWithJI`

2.11 Package `BankWithJI`

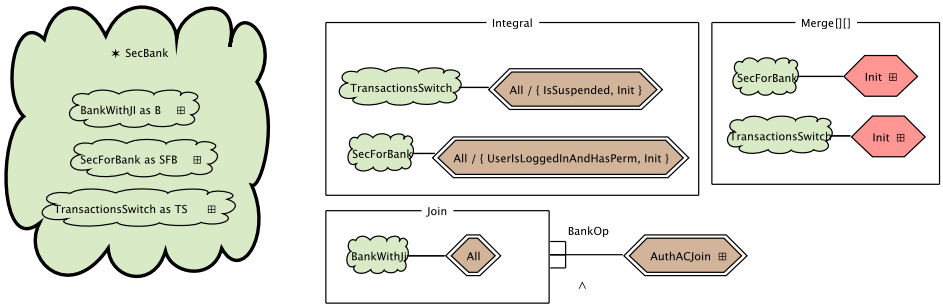
This section introduces package `BankWithJI` (*Bank With Join Interfaces*), which adds the access-control join interface, package `BankACJI`, to the package `BankWithTransSwitch`.

PD of package `BankWithJI` (Fig 2.31a) says that the ensemble package `BankWithJI` extends packages `BankACJI` and `BankWithTransSwitch`. The BD (Fig:BankWithJI:BD) specifies the merge by saying that all operations of package `BankWithJI` are to be merged with all operations of package `BankACJI`; this means that all operations with the same name are merged into a single operation.

2.12 Package `SecBank`

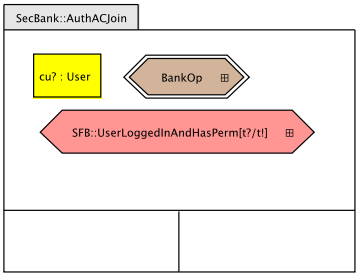
The package `SecBank` encapsulates the overall secure simple bank system with all its concerns.

PD of package `SecBank` (Fig. 2.32a) says that the ensemble package `SecBank` extends packages `SecForBank`, `AuthenticationOps`, `BankWithJI` and `TransactionsSwitch`. The BD (Fig. 2.32b) says that all operations of packages `TransactionsSwitch` and `AuthenticationOps` are to be used integrally in the new context, and that all operations of package `BankWithJI` are to be joined with the behaviour specified in the join contract `AuthACJoin`. The join contract `AuthACJoin` (Fig. 2.32c) adds a precondition to the joined operation; it says that the operation may be executed provided the current user is logged in and has permissions to execute some given task.



(a) Package Diagram

(b) Behaviour Diagram



(c) Join contract `AuthACJoin`

Figure 2.32: VCL diagrams of package `SecBank`

Chapter 3

Z Specification generated from the VCL model of secure simple Bank

This chapter presents the Z specification generated from the VCL model of secure simple Bank. Appendix A presents some Z definitions from the ZOO toolkit that are used in the Z specifications presented here. The Z specification presented here has been type-checked using the *CZT* (Community Z Tools) Z type-checker.

3.1 Preamble

$$CLASS ::= CustomerCl \mid AccountCl \mid UserCl \mid SessionCl$$

$$\begin{array}{|l} subCl : CLASS \leftrightarrow CLASS \\ abstractCl : \mathbb{P}\, CLASS \\ rootCl : \mathbb{P}\, CLASS \\ \hline subCl = \{\} \\ abstractCl = \{\} \\ rootCl = CLASS \setminus \mathrm{dom}\, subCl \end{array}$$

$$\mathbb{O} : CLASS \to \mathbb{P}_1\, OBJ$$
$$\mathbb{O}_x : CLASS \to \mathbb{P}_1\, OBJ$$

$$\text{disjoint}\, \mathbb{O}_x$$

$$\forall\, cl : CLASS \bullet$$
$$\quad \mathbb{O}\, cl = \mathbb{O}_x\, cl \cup \bigcup (\mathbb{O}_x\, (\!|\, (subCl^{+})^{\sim}\, (\!|\{cl\}|\!)\, |\!))$$

$$\forall\, cl, cl' : CLASS \mid cl \mapsto cl' \in subCl \bullet$$
$$\quad \mathbb{O}\, cl \subseteq \mathbb{O}\, cl'$$

3.2 Package `CommonTypes`

section *CommonTypes* **parents** *ZOO_Toolkit*, *Model_Preamble*

[*Name*]

$$TimeNat == \mathbb{N}$$

3.3 Package `Bank`

This section presents Z specification generated for the VCL package `Bank` (section 2.2).

section *Bank* **parents** *ZOO_Toolkit*, *Model_Preamble*, *CommonTypes*

Set `CustType`

$$CustType ::= corporate \mid personal$$

Set `Address`

[*Address*]

Set `CustID`

[*CustID*]

Set `Customer`

*Customer*0

$$
\begin{array}{l}
name : Name \\
address : Address \\
custNo : CustID \\
cType : CustType
\end{array}
$$

Customer

$$Customer0$$

SCustomer

$$
\begin{array}{l}
sCustomer : \mathbb{P}(\mathbb{O} CustomerCl) \\
stCustomer : \mathbb{O} CustomerCl \pfun Customer \\
\hline
\dom stCustomer = sCustomer
\end{array}
$$

SCustomerInit

$$
\begin{array}{l}
SCustomer' \\
\hline
sCustomer' = \varnothing \\
stCustomer' = \varnothing
\end{array}
$$

CustomerNew

$$
\begin{array}{l}
Customer' \\
name? : Name \\
address? : Address \\
cType? : CustType \\
\hline
name' = name? \\
address' = address? \\
cType' = cType?
\end{array}
$$

$\Phi BankSCustomerN$

$\Delta SCustomer$
$Customer'$
$c! : \mathbb{O}\ CustomerCl$

$c! \in \mathbb{O}_x\ CustomerCl \setminus sCustomer$
$sCustomer' = sCustomer \cup \{c!\}$
$stCustomer' = stCustomer \cup \{(c! \mapsto \theta\ Customer')\}$

$SCustomerNew == \exists\ Customer' \bullet \Phi BankSCustomerN \land CustomerNew$

Set `AccID`

$[AccID]$

Set `AccType`

$AccType ::= savings \mid current$

Set `Account`

$Account0$

$accNo : AccID$
$aType : AccType$
$balance : \mathbb{Z}$

$AccountSavingsArePositive$

$Account0$

$(aType = savings) \Rightarrow (balance \geq 0)$

$Account$

$Account0$

$AccountSavingsArePositive$

SAccount

$sAccount : \mathbb{P}(\mathbb{O}\ AccountCl)$

$stAccount : \mathbb{O}\ AccountCl \pfun Account$

$\dom stAccount = sAccount$

SAccountInit

$SAccount'$

$sAccount' = \varnothing$

$stAccount' = \varnothing$

AccountNew

$Account'$

$accNo? : ACCID$

$aType? : AccType$

$accNo' = accNo?$

$aType' = aType?$

$balance' = 0$

AccountDelete

$Account$

$balance = 0$

AccountDeposit

$\Delta Account$

$amount? : \mathbb{N}$

$accNo' = accNo'$

$aType' = aType'$

$balance' = balance + amount?$

AccountWithdraw

$\Delta Account$
$amount? : \mathbb{N}$

$accNo' = accNo'$
$aType' = aType'$
$balance' = balance - amount?$

AccountGetBalance

$\Xi Account$
$accBal! : \mathbb{Z}$

$accBal! = balance$

$\Phi BankSAccountN$

$\Delta\ SAccount$
$Account'$
$a! : \mathbb{O}\ AccountCl$

$a! \in \mathbb{O}_x\ AccountCl \setminus sAccount$
$sAccount' = sAccount \cup \{a!\}$
$stAccount' = stAccount \cup \{(a! \mapsto \theta\ Account')\}$

$\Phi BankSAccountU$

$\Delta\ SAccount$
$\Delta\ Account$
$a? : \mathbb{O}\ AccountCl$

$a? \in sAccount$
$\theta\ Account = stAccount\ a?$
$sAccount' = sAccount$
$stAccount' = stAccount \oplus \{(a? \mapsto \theta\ Account')\}$

$\Phi BankSAccountO$

$\Xi SAccount$
$\Xi Account$
$a? : \mathbb{O}\ AccountCl$

$a? \in sAccount$
$\theta\ Account = stAccount\ a?$

$\Phi BankSAccountD$

$\Delta SAccount$
$Account$
$a? : \mathbb{O}\ AccountCl$

$a? \in sAccount$
$\theta\ Account = stAccount\ a?$
$sAccount' = sAccount \setminus \{a?\}$
$stAccount' = \{a?\} \lhd stAccount$

$SAccountNew == \exists\ Account' \bullet \Phi BankSAccountN \land AccountNew$
$SAccountDelete == \exists\ Account \bullet \Phi BankSAccountD \land AccountDelete$
$SAccountDeposit == \exists \Delta Account \bullet \Phi BankSAccountU \land AccountDeposit$
$SAccountWithdraw == \exists \Delta Account \bullet \Phi BankSAccountU \land AccountWithdraw$
$SAccountGetBalance == \exists \Delta Account \bullet \Phi BankSAccountO \land AccountGetBalance$

Relational Edge Holds

$Holds$

$rHolds : \mathbb{O}\ CustomerCl \leftrightarrow \mathbb{O} AccountCl$

$HoldsInit$

$Holds'$

$rHolds' = \varnothing$

$HoldsAddNew$

$\Delta Holds$
$a? : \mathbb{O}\ AccountCl$
$c? : \mathbb{O}\ CustomerCl$

$rHolds' = rHolds \cup \{(a?, c?)\}$

$HoldsDelGivenAccount$

$\Delta Holds$
$a? : \mathbb{O}\ AccountCl$

$rHolds' = rHolds \rhd \{a?\}$

Global State

BankGblSt

$$SCustomer$$
$$SAccount$$
$$Holds$$

BankHoldsGCnt

$$BankGblSt$$

$$mult\,(rHolds,\ sCustomer,\ sAccount, om, \varnothing, \varnothing)$$

BankSavingsArePositive2

$$BankGblSt$$

$$\{o : sAccount \mid (stAccount\ o).aType = savings \land (stAccount\ o).balance < 0\} = \varnothing$$

BankCustIdsUnique

$$BankGblSt$$

$$\forall\, c1, c2 : \mathbb{O}\ CustomerCl \bullet$$
$$((c1 \neq c2) \Rightarrow ((stCustomer\ c1).custNo \neq (stCustomer\ c2).custNo))$$

BankAccountIdsUnique

$$BankGblSt$$

$$\forall\, a1, a2 : \mathbb{O}\ AccountCl \bullet$$
$$((a1 \neq a2) \Rightarrow ((stAccount\ a1).accNo \neq (stAccount\ a2).accNo))$$

BankCorporateHaveNoSavings

$$BankGblSt$$

$$(\{o : sCustomer \mid (stCustomer\ o).cType = corporate\}$$
$$\lhd rHolds) \rhd \{o : sAccount \mid (stAccount\ o).aType = savings\} = \varnothing$$

*BankHasCurrentBefSavings*0

$$
\begin{array}{l}
BankGblSt \\
custsCurr : \mathbb{P}\,\mathbb{O}\ CustomerCl \\
custsSav : \mathbb{P}\,\mathbb{O}\ CustomerCl \\
\hline
custsCurr = \mathrm{dom}\,(rHolds \rhd \{o : sAccount \mid (stAccount\ o).aType = current\}) \\
custsSav = \mathrm{dom}\,(rHolds \rhd \{o : sAccount \mid (stAccount\ o).aType = savings\}) \\
custsCurr \subseteq custsSav
\end{array}
$$

$$BankHasCurrentBefSavings == BankHasCurrentBefSavings0 \setminus (custsCurr,\ custsSav)$$

BankGbl

$$
\begin{array}{l}
BankGblSt \\
\hline
BankHoldsGCnt \\
BankCorporateHaveNoSavings \\
BankCustIdsUnique \\
BankHasCurrentBefSavings \\
BankAccountIdsUnique \\
BankSavingsArePositive2
\end{array}
$$

BankAHoldsGCnt

$$
\begin{array}{l}
BankSt \\
\hline
\mathrm{mult}(rHolds, sCustomer, sAccount, om, \{\}, \{\})
\end{array}
$$

$$BankInit == Bank' \land SCustomerInit \land SAccountInit \land HoldsInit$$

Operation `CreateCustomer`

$$\Psi BankCreateCustomer == \Delta Bank \land \Xi SAccount \land \Xi Holds$$

$$BankCreateCustomer == \Psi BankCreateCustomer \land SCustomerNew$$

Operation `GetCustomerGivenCustNo`

$$
\begin{array}{|l}
\hline
BankGetCustomerGivenCustNo \\
\hline
BankGblSt \\
cNo? : CustID \\
c! : \mathbb{O}\ CustomerCl \\
\hline
c! \in \{o : sCustomer \mid (stCustomer\ o).custNo = cNo?\} \\
\hline
\end{array}
$$

Operation `OpenAccount`

$$\Psi BankOpenAccount == \Delta Bank \land \Xi Customer$$

$$BankOpenAccount0 == [c? : \mathbb{O}\ CustomerCl;\ \Delta Bank \mid c? \in sCustomer]$$

$$BankOpenAccount == \Psi BankOpenAccount \land SAccountNew \land BankOpenAccount0 \land HoldsAddNew[a!/a?] \land BankGetCustomerGivenCustNo \setminus (accNo?, a!, c?)$$

Operation `GetAccountGivenAccNo`

$$
\begin{array}{|l}
\hline
BankGetAccountGivenAccNo \\
\hline
BankGblSt \\
aNo? : AccID \\
a! : \mathbb{O}\ AccountCl \\
\hline
a! \in \{o : sAccount \mid (stAccount\ o).accNo = aNo?\} \\
\hline
\end{array}
$$

Operation `AccDelete`

$$\Psi BankAccDelete == \Delta Bank \land \Xi Customer$$

$$BankAccDelete == \Psi BankAccDelete \land GetAccountGivenAccNo \land SAccountDelete \land HoldsDelGivenAccount \setminus (a?)$$

Operation `AccDeposit`

$$\Psi BankAccDeposit == \Delta Bank \land \Xi Customer \land \Xi Holds$$

$$BankAccDeposit == \Psi BankAccDeposit \land GetAccountGivenAccNo \land SAccountDeposit \setminus (a?)$$

Operation `AccWithdraw`

$$\Psi BankAccWithdraw == \Delta Bank \land \Xi Customer \land \Xi Holds$$
$$BankAccWithdraw == \Psi BankAccWithdraw \land GetAccountGivenAccNo \land SAccountWithdraw \setminus (a?)$$

Operation `AccGetBalance`

$$BankAccGetBalance == \Xi Bank \land\land GetAccountGivenAccNo \land SAccountGetBalance \setminus (a?)$$

Operations `GetAccsInDebt` **and** `GetCustAccounts`

*BankGetAccsInDebt*0

$$BankGbl$$
$$accs : \mathbb{P}\,\mathbb{O}\ AccountCl$$
$$accNos! : \mathbb{P}\ AccID$$

$$accs = \{o : sAccount \mid (stAccount\ o).balance < 0\}$$
$$\forall a : \mathbb{O}\ AccountCl \bullet ((a \in accs) \Rightarrow ((stAccount\ a).accNo \in accNos!))$$

$$BankGetAccsInDebt == BankGetAccsInDebt0 \setminus (accs)$$

BankGetCustomerGivenCustNo

$$BankGblSt$$
$$cNo? : CustID$$
$$c! : \mathbb{O}\ CustomerCl$$

$$c! \in \{o : sCustomer \mid (stCustomer\ o).custNo = cNo?\}$$

*BankGetCustAccs*0

$$BankGbl$$
$$cNo? : CustID$$
$$accs : \mathbb{P}\,\mathbb{O}\ AccountCl$$
$$c! : \mathbb{O}\ CustomerCl$$
$$accNos! : \mathbb{P}\ AccID$$

$$BankGetCustomerGivenCustNo$$
$$accs = rHolds\ (\!|\ \{c!\}\ |\!)$$
$$\forall a : \mathbb{O}\ AccountCl \bullet ((a \in accs) \Rightarrow ((stAccount\ a).accNo \in accNos!))$$

$$BankGetCustAccs == BankGetCustAccs0 \setminus (accs, c!)$$

3.4 Package `Authentication`

The following presents the Z specification that is generated for VCL package `Authentication` (section 2.3).

section *Authentication* **parents** *ZOO_Toolkit*, *Model_Preamble*, *CommonTypes*

Set `LoginResult`

$$LoginResult ::= loginOK \mid wrongPW \mid isBlocked$$

Set `UID`

$$[SID]$$

Set `Session`

*Session*0

$$sid : SID$$
$$startTm : TimeNat$$
$$lastTmActive : TimeNat$$

Session

$$Session0$$

SSession

$$sSession : \mathbb{P}(\mathbb{O}\ SessionCl)$$
$$stSession : \mathbb{O}\ SessionCl \pfun Session$$

$$\mathrm{dom}\ stSession = sSession$$

SSessionInit

$SSession$

$sSession = \varnothing \land stSession = \varnothing$

SessionNew

$Session'$
$sid? : SID$
$now? : Time$

$sid' = sid?$
$startTm' = now?$
$lastTmActive' = now?$

SessionDelete

$Session$

$\Phi SSessionNew$

$\Delta SSession$
$Session'$
$s! : \mathbb{O}\, SessionCl$

$s! \in \mathbb{O}_x\, SessionCl \setminus sSession$
$sSession' = sSession \cup \{s!\}$
$stSession' = stSession \cup \{(s! \mapsto \theta\, Session')\}$

$\Phi SSessionUpd$

$\Delta SSession$
$\Delta Session$
$s? : \mathbb{O}\, SessionCl$

$s? \in sSession$
$\theta\, Session = stSession\, s?$
$sSession' = sSession$
$stSession' = stSession \oplus \{(s?, \theta\, Session')\}$

$\Phi SSessionO$

$SSession$
$Session$
$s? : \mathbb{O}\, SessionCl$

$s? \in sSession$
$\theta\, Session = stSession\, s?$

$\Phi SSessionDel$

$\Delta SSession$
$Session$
$s? : \mathbb{O} SessionCl$

$s? \in sSession$
$\theta\, Session = stSession\, s?$
$sSession' = sSession \setminus \{s?\}$
$stSession' = \{s?\} \lhd stSession$

$SSessionNew == \exists\, Session' \bullet \Phi SSessionNew \land SessionNew$

$SSessionDelete == \exists\, Session \bullet \Phi SSessionDel \land SessionDelete$

Set `UserStatus`

$UserStatus ::= loggedIn \mid blocked \mid loggedOut$

Set `Password`

$[Password]$

Set `UID`

$[UID]$

Set `User`

This defines constant `maxPWMisses`.

$UsermaxPwMisses : \mathbb{N}$

*User*0

$uid : UID$
$pw : Password$
$name : Name$
$pwMisses : \mathbb{N}$
$status : UserStatus$

UserMaxPwMissesInv

*User*0

$pwMisses \leq UsermaxPwMisses$

User

*User*0

UserMaxPwMissesInv

SUser

$sUser : \mathbb{P}(\mathbb{O}\ UserCl)$
$stUser : \mathbb{O}\ UserCl \pfun User$

$\mathrm{dom}\ stUser = sUser$

SUserInit

SUser

$sUser = \varnothing \land stUser = \varnothing$

UserIsLoggedIn

User

$status = loggedIn$

SUserGetUserGivenID

$SUser$
$u! : \mathbb{O}\ UserCl$
$uid? : UID$

$(stUser\ u!).uid = uid?$

UserLoginOk

$\Delta\ User$
$pw? : Password$
$r! : LoginResult$

$status = loggedOut$
$pw = pw?$
$pwMisses' = 0$
$status' = loggedIn$
$pw' = pw$
$name' = name$
$uid' = uid$
$r! = loginOK$

UserLogout

$\Delta\ User$

$status = loggedIn$
$status' = loggedOut$
$pwMisses' = pwMisses$
$pw' = pw$
$name' = name$
$uid' = uid$

UserLoginBlocked

$\Delta\ User$
$r! : LoginResult$

$status = blocked$
$status' = status$
$pwMisses' = pwMisses$
$pw' = pw \land name' = name$
$uid' = uid$
$r! = isBlocked$

UserLoginWrongPW

$\Delta\, User$
$pw? : Password$
$r! : LoginResult$

$status = loggedOut$
$pw \neq pw?$
$pwMisses < maxPwMisses$
$status' = status$
$pwMisses' = pwMisses + 1$
$pw' = pw \land name' = name$
$uid' = uid$
$r! = wrongPW$

UserLoginWrongPWToBlocked

$\Delta\, User$
$pw? : Password$
$r! : LoginResult$

$status = loggedOut$
$pw \neq pw?$
$pwMisses = maxPwMisses$
$status' = blocked$
$pwMisses' = pwMisses$
$pw' = pw \land name' = name \land uid' = uid$
$r! = wrongPW$

$\Phi SUserNew$

$\Delta\, SUser$
$User'$
$u! : \mathbb{O}\ UserCl$

$u! \in \mathbb{O}_x\ UserCl \setminus sUser$
$sUser' = sUser \cup \{u!\}$
$stUser' = stUser \cup \{(u!, \theta\ User')\}$

$\Phi SUserDel$

$\Delta SUser$
$User$
$u? : \mathbb{O}\ UserCl$

$u? \in sUser$
$\theta\ User = stUser\ u?$
$sUser' = sUser \setminus \{u?\}$
$stUser' = \{u?\} \lhd stUser$

$\Phi SUserUpd$

$\Delta SUser$
$\Delta User$
$u? : \mathbb{O}\ UserCl$

$u? \in sUser$
$\theta\ User = stUser\ u?$
$sUser' = sUser$
$stUser' = stUser \oplus \{(u?, \theta\ User')\}$

$UserLoginNotOk == UserLoginBlocked \lor UserLoginWrongPW$
$\quad \lor UserLoginWrongPWToBlocked$

$SUserLoginOk == \exists \Delta User \bullet \Phi SUserUpd \land UserLoginOk$

$SUserLogout == \exists \Delta User \bullet \Phi SUserUpd \land UserLogout$

$SUserLoginNotOk == \exists \Delta User \bullet \Phi SUserUpd \land UserLoginNotOk$

Relation Edge `HasSession`

$HasSession$

$rHasSession\ :\ \mathbb{O}\ UserCl \leftrightarrow \mathbb{O}\ SessionCl$

$HasSessionInit$

$HasSession'$

$rHasSession' = \varnothing$

HasSessionAddNew

$\Delta HasSession$
$u? : \mathbb{O}\ UserCl$
$s? : \mathbb{O}\ SessionCl$

$rHasSession' = rHasSession \cup \{(u?, s?)\}$

HasSessionDelGivenUser

$\Delta HasSession$
$u? : \mathbb{O}\ UserCl$

$rHasSession' = \{u?\} \lhd rHasSession$

HasSessionGetUserSession

$HasSession$
$u? : \mathbb{O}\ UserCl$
$s! : \mathbb{O}\ SessionCl$

$(u?, s!) \in rHasSession$

Global State

AuthenticationGblSt

$SUser$
$SSession$
$AHasSession$

AuthenticationHasSessionGCnt

$AuthenticationGblSt$

$mult\ (rHasSession,\ sUser,\ sSession, ozo, \varnothing, \varnothing)$

AuthenticationHasSessionIffLoggedIn

$AuthenticationGblSt$

$\{o : sUser \mid (stUser\ o).status = loggedIn\} = \mathrm{dom}\ (rHasSession)$

AuthenticationSIDsUnique

AuthenticationGblSt

$\forall\, s1, s2 : \mathbb{O}\ SessionCl \bullet$
$((s1 \neq s2) \Rightarrow ((stSession\ s1).sid \neq (stSession\ s2).sid))$

AuthenticationUIDsUnique

AuthenticationGblSt

$\forall\, u1, u2 : \mathbb{O}\ UserCl \bullet$
$((u1 \neq u2) \Rightarrow ((stUser\ u1).uid \neq (stUser\ u2).uid))$

AuthenticationGbl

AuthenticationGblSt

AuthenticationHasSessionGCnt

AuthenticationHasSessionIffLoggedIn

AuthenticationUIDsUnique

AuthenticationSIDsUnique

$AuthenticationInit == AuthenticationGbl' \land SUserInit \land SSesionInit \land HasSessionInit$

Global Behaviour

AuthenticationGetUserGivenUID

AuthenticationGbl
$cuid? : UID$
$u! : \mathbb{O}\ UserCl$

$u! \in \{o : sUser \mid (stUser\ o).uid = cuid?\}$

SUserOF

$$SUser$$
$$User$$
$$o? : \mathbb{O}\ UserCl$$

$$o? \in sUser$$
$$\theta\ User = stUser\ o?$$

$$SUserIsLoggedIn == \exists\ User \bullet SUserOF \land UserIsLoggedIn$$

AuthenticationUserIsLoggedIn0

$$AuthenticationGbl$$
$$cuid? : UID$$
$$u! : \mathbb{O}\ UserCl$$
$$SUserIsLoggedIn[u!/o?]$$

$$AuthenticationGetUserGivenUID$$

$$AuthenticationUserIsLoggedIn == AuthenticationUserIsLoggedIn0 \setminus (u!)$$

$$\Psi AuthenticationLoginOk == \Delta AuthenticationOps$$

$$AuthenticationLoginOk == \Psi AuthenticationLoginOk \land SUserLoginOk \land SSessionNew \land HasSessionAddNew[s!/s?] \setminus (sid?, s!)$$

$$\Psi AuthenticationLoginNotOk == \Delta Authentication \land \Xi SSession \land \Xi HasSession$$

$$AuthenticationLoginNotOk == \Psi AuthenticationLoginNotOk \land SUserLoginNotOk$$

$$AuthenticationLogin == SUserGetUserGivenID[u?/u!] \land (AuthenticationLoginOk \lor AuthenticationLoginNotOk) \setminus (u?)$$

$$\Psi AuthenticationLogout == \Delta AuthenticationOps$$

$$AuthenticationLogout == \Psi AuthenticationLogout \land SUserGetUserGivenID[u?/u!] \land SUserLogout \land HasSessionDelGivenUser \land HasSessionGetUserSession[s?/s!] \land SessionDelete \setminus (u?, s?)$$

3.5 Package `AccessControl`

The following presents the Z specification that is generated for VCL package `AccessControl` (section 2.4).

section *AccessControl* **parents** *ZOO_Toolkit*, *Model_Preamble*

Set `User`

$[User]$

Set `Role`

$[Role]$

Set `Task`

$[Task]$

Relation Edge `HasPerm`

HasPerm

$$rHasPerm : Role \leftrightarrow Task$$

HasPermInit

$$HasPerm'$$

$$rHasPerm' = \varnothing$$

Relation Edge `HasRole`

HasRole

$$rHasRole : User \leftrightarrow Role$$

HasRoleInit

$$HasRole'$$

$$rHasRole' = \varnothing$$

Global State

AccessControlGblSt

$$HasPerm$$
$$HasRole$$

AccessControlHasPermGCnt

$$AccessControlGblSt$$

$$mult\,(rHasPerm,\ Role,\ Task, mm, \varnothing, \varnothing)$$

AccessControlHasRoleGCnt

$$AccessControlGblSt$$

$$mult\,(rHasRole,\ User,\ Role, mm, \varnothing, \varnothing)$$

AccessControlGbl

$$AccessControlGblSt$$

$$AccessControlHasPermGCnt$$

$$AccessControlHasRoleGCnt$$

$$AccessControlInit == AccessControlGbl' \land HasRoleInit \land HasPermInit$$

3.5.1 Global Behaviour

$$
\begin{array}{|l}
\hline
AccessControlUserHasPerm0 \\
AccessControlGbl \\
r : Role \\
t? : Task \\
cu? : User \\
\hline
(cu?, r) \in rHasRole \\
(r, t?) \in rHasPerm \\
\hline
\end{array}
$$

$$AccessControlUserHasPerm == AccessControlUserHasPerm0 \setminus (r)$$

3.6 Package `RolesAndTasksBank`

The following presents the Z specification that is generated for VCL package `RolesAndTasksBank` (section 2.6).

3.6.1 Blob *Role*

$$Role ::= Clerk \mid Manager$$

3.6.2 Blob *Task*

$$Task ::= TCreateCustomer \mid TOpenAccount \mid TAccDeposit \mid TAccWithdraw \mid TAccGetBalance \mid TAccDelete \mid TGetAccsInDebt \mid TGetCustAccs$$

$$
\begin{array}{|l}
\hline
ClerkTasks, ManagerTasks : \mathbb{P}\ Task \\
\hline
ClerkTasks = \{TAccDeposit, TAccWithdraw\} \\
ManagerTasks = \{TCreateCustomer, TOpenAccount, TAccDelete\} \\
\hline
\end{array}
$$

3.7 Package `Authorisation`

The following presents the Z specification that is generated for VCL package `Authorisation` (section 2.5).

3.7.1 Global State

Authorisation

Authentication
AccessControl

3.7.2 Global Behaviour

AuthorisationUserLoggedInAndHasPerm

Authorisation
AccessControlUserHasPerm
AuthenticationUserIsLoggedIn

3.8 Package `SecForBank`

The following presents the Z specification that is generated for VCL package `SecForBank` (section 2.7).

3.8.1 Global State

This gives a value to constant `maxPwMisses` of package `Users`:

$$maxPwMisses = 3$$

SecForBank

Authorisation

$$AccessControlInitMod == AccessControlInit \setminus (rHasPerm')$$

SecForBankInit

SecForBank'
AuthenticationInit
AccessControlInitMod

$$rHasPerm' = (\{Clerk\} \times (ClerkTasks \cup TasksOfBoth)) \cup (\{Manager\} \times (ManagerTasks \cup TasksOfBoth))$$

3.8.2 Global Behaviour

$$SecForBankUserLoggedInAndHasPerm == SecForBank \\ \land\ AuthorisationUserLoggedInAndHasPerm$$

3.9 Package `BankACJI`

The following presents the Z specification that is generated for VCL package `BankACJI` (section 2.10).

3.9.1 Global Behaviour

BankACJICreateCustomer

$$t! : Task$$

$$t! = TCreateCustomer$$

BankACJIOpenAccount

$$t! : Task$$

$$t! = TOpenAccount$$

BankACJIAccDeposit

$$t! : Task$$

$$t! = TAccDeposit$$

BankACJIAccWithdraw

$$t! : Task$$

$$t! = TAccWithdraw$$

BankACJIAccGetBalance

$$t! : Task$$

$$t! = TAccGetBalance$$

BankACJIAccDelete

$t! : Task$

$t! = TAccDelete$

BankACJIGetAccsInDebt

$t! : Task$

$t! = TGetAccsInDebt$

BankACJIGetCustAccs

$t! : Task$

$t! = TGetCustAccs$

3.10 Package `BankWithJI`

The following presents the Z specification that is generated for VCL package `BankWithJI` (section 2.11).

3.10.1 Global State

BankWithJI

$Bank$

$BankWithJIInit == BankWithJI' \land BankInit$

3.10.2 Global Behaviour

Defines the frame for *update* operations.

$CommonWithBank == BankWithJI \upharpoonright Bank$

$BankWithJIWithoutBank == \exists\, CommonWithBank \bullet BankWithJI$

$\Psi BankWithJIMergeOps == \Delta BankWithJI \land \Xi BankWithJIWithoutBank$

BankWithJICreateCustomer

$\Psi BankWithJIMergeOps$
$BankCreateCustomer$
$BankACJICreateCustomer$

BankWithJIOpenAccount

$\Psi BankWithJIMergeOps$
$BankOpenAccount$
$BankACJIOpenAccount$

BankWithJIAccDeposit

$\Psi BankWithJIMergeOps$
$BankAccDeposit$
$BankACJIAccDeposit$

BankWithJIAccWithdraw

$BankAccWithdraw$
$BankACJIAccWithdraw$

BankWithJIAccGetBalance

$BankWithJI$
$BankAccGetBalance$
$BankACJIAccGetBalance$

BankWithJIAccDelete

$\Psi BankWithJIMergeOps$
$BankAccDelete$
$BankACJIAccDelete$

BankWithJIGetAccsInDebt

$BankWithJI$
$BankGetAccsInDebt$
$BankACJIGetAccsInDebt$

BankWithJIGetCustAccs

BankWithJI
BankGetCustAccs
BankACJIGetCustAccs

3.11 Package `SecBank`

The following presents the Z specification that is generated for VCL package `SecBank` (section 2.12).

3.11.1 Global State

SecBank

SecForBank
BankWithJI
AuthenticationOps

SecBankInit

SecForBankInit
BankWithJIInit
AuthenticationOpsInit

3.11.2 Global Bebhaviour

$$CommonWithAuthenticationOps == SecBank \upharpoonright AuthenticationOps$$

$$SecBankWithoutAuthenticationOps == \exists\, CommonWithAuthenticationOps \bullet SecBank$$

$$\Psi SecBankOpsFromAuthenticationOps == \Delta SecBank \land \Xi SecBankWithoutAuthenticationOps$$

Login

$\Psi SecBankOpsFromAuthenticationOps$
AuthenticationOpsLogin

Logout

$\Psi SecBankOpsFromAuthenticationOps$
AuthenticationOpsLogout

$$CommonWithBankWithJI == SecBank \upharpoonright BankWithJI$$

$$SecBankWithoutBankWithJi == \exists\, CommonWithBankWithJI \bullet SecBank$$

$$\Psi SecBankOpsFromBankWithJI == \Delta SecBank \land \Xi SecBankWithoutBankWithJi$$

CreateCustomer

$\Psi SecBankOpsFromBankWithJI$
$cu? : \mathbb{O}\, UserCl$
$BankWithJICreateCustomer$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

OpenAccount

$\Psi SecBankOpsFromBankWithJI$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIOpenAccount$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccDeposit

$\Psi SecBankOpsFromBankWithJI$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccDeposit$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccWithdraw

$\Psi SecBankOpsFromBankWithJI$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccWithdraw$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccGetBalance

$SecBank$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccGetBalance$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccDelete

$\Psi SecBankOpsFromBankWithJI$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccDelete$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccGetAccsInDebt

$SecBank$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccDeposit$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

AccGetCustAccs

$SecBank$
$cu? : \mathbb{O}\, UserCl$
$BankWithJIAccWithdraw$

$SecForBankUserLoggedInAndHasPerm[t!/t?]$

Appendix A

ZOO Toolkit

section *ZOO_Toolkit* **parents** *standard_toolkit*

$[OBJ]$

relation$(opt\ \_)$

$[X]$

$$opt\ \_ : \mathbb{P}(\mathbb{P}\ X)$$
$$the : \mathbb{P}\ X \pfun X$$

$$\forall\ S : \mathbb{P}\ X \bullet opt\ S \Leftrightarrow (\exists\ x : X \bullet S = \{x\}) \lor S = \{\}$$
$$\forall\ x : X \bullet the\ \{x\} = x$$

$[L]$

$$\Sigma : (L \pfun \mathbb{Z}) \to \mathbb{Z}$$

$$\Sigma\ \{\} = 0$$
$$\forall\ l : L;\ n : \mathbb{Z} \bullet \Sigma\ \{(l, n)\} = n$$
$$\forall\ l : L;\ n : \mathbb{Z};\ S : L \pfun \mathbb{Z} \mid \neg\ l \in \mathrm{dom}\ S \bullet \Sigma\ (\{(l, n)\} \cup S) = n + \Sigma\ S$$

$MultTy ::= mm \mid mo \mid om \mid mzo \mid zom \mid mlo \mid lom \mid lolo \mid loo \mid olo \mid lozo \mid zolo \mid oo \mid zozo \mid zoo \mid ozo \mid ms \mid sm \mid ss \mid so \mid os \mid szo \mid zos$

relation(*mult* _)

$[X, Y]$

$mult\_ : \mathbb{P}((X \leftrightarrow Y) \times \mathbb{P}\, X \times \mathbb{P}\, Y \times MultTy \times \mathbb{F}\, \mathbb{N} \times \mathbb{F}\, \mathbb{N})$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, mm, s_1, s_2)) \Leftrightarrow r \in sx \leftrightarrow sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, mo, s_1, s_2)) \Leftrightarrow r \in sx \to sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, om, s_1, s_2)) \Leftrightarrow r^{\sim} \in sy \to sx$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, mzo, s_1, s_2)) \Leftrightarrow r \in sx \nrightarrow sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, zom, s_1, s_2)) \Leftrightarrow r^{\sim} \in sy \nrightarrow sx$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, mlo, s_1, s_2)) \Leftrightarrow r \in sx \leftrightarrow sy \land \operatorname{dom} r = sx$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, lom, s_1, s_2)) \Leftrightarrow r \in sx \leftrightarrow sy \land \operatorname{ran} r = sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, lolo, s_1, s_2)) \Leftrightarrow r \in sx \leftrightarrow sy \land \operatorname{dom} r = sx \land \operatorname{ran} r = sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, loo, s_1, s_2)) \Leftrightarrow r \in sx \twoheadrightarrow sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, olo, s_1, s_2)) \Leftrightarrow r^{\sim} \in sy \twoheadrightarrow sx$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, lozo, s_1, s_2)) \Leftrightarrow r \in sx \twoheadrightarrow sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, zolo, s_1, s_2)) \Leftrightarrow r^{\sim} \in sy \twoheadrightarrow sx$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, oo, s_1, s_2)) \Leftrightarrow r \in sx \rightarrowtail sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, zozo, s_1, s_2)) \Leftrightarrow r \in sx \rightarrowtail sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, zoo, s_1, s_2)) \Leftrightarrow r \in sx \rightarrowtail sy$

$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$
$(mult(r, sx, sy, ozo, s_1, s_2)) \Leftrightarrow r^{\sim} \in sy \rightarrowtail sx$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, ms, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, mm, s_1, s_2))$$
$$\land\ (\forall\, x : \mathrm{dom}\, r \bullet \#(\{x\} \lhd r) \in s_1)$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, sm, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, mm, s_1, s_2))$$
$$\land\ (\forall\, y : \mathrm{ran}\, r \bullet \#(r \rhd \{y\}) \in s_1)$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, ss, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, ms, s_1, \{\}))$$
$$\land\ (mult(r, sx, sy, sm, s_2, \{\}))$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, so, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, mo, s_1, s_2))$$
$$\land\ (mult(r, sx, sy, sm, s_1, s_2))$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, os, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, om, \{\}, \{\}))$$
$$\land\ (mult(r, sx, sy, ms, s_1, \{\}))$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, szo, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, mzo, \{\}, \{\}))$$
$$\land\ (mult(r, sx, sy, sm, s_1, \{\}))$$

$$\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\, \mathbb{N} \bullet$$
$$(mult(r, sx, sy, zos, s_1, s_2)) \Leftrightarrow (mult(r, sx, sy, zom, \{\}, \{\}))$$
$$\land\ (mult(r, sx, sy, ms, s_1, \{\}))$$

References

[AGK11] Nuno Amálio, Christian Glodt, and Pierre Kelsen. Building VCL models and automatically generating Z specifications from them. In *FM 2011*, volume 6664 of *LNCS*, pages 149–153. Springer, 2011.

[AK10a] Nuno Amálio and Pierre Kelsen. Modular design by contract visually and formally using VCL. In *VL/HCC*, 2010.

[AK10b] Nuno Amálio and Pierre Kelsen. VCL, a visual language for abstract specification of software systems formally and modularly (short paper). In *Diagrams 2010*, volume 6170 of *LNAI*. Springer, 2010.

[AKM10] Nuno Amálio, Pierre Kelsen, and Qin Ma. Specifying structural properties and their constraints formally, visually and modularly using VCL. In *EMMSAD 2010*, volume 50 of *LNBIP*, pages 261–273. Springer, 2010.

[AKMG10] Nuno Amálio, Pierre Kelsen, Qin Ma, and Christian Glodt. Using VCL as an aspect-oriented approach to requirements modelling. *TAOSD LNCS*, VII:151–199, 2010.

[Amá07] Nuno Amálio. *Generative frameworks for rigorous model-driven development*. PhD thesis, Dept. Computer Science, Univ. of York, 2007.

[APS05] Nuno Amálio, Fiona Polack, and Susan Stepney. An object-oriented structuring for Z based on views. In *ZB 2005*, volume 3455 of *LNCS*, pages 262–278, 2005.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2), 1996.