16. Case studies of formal specifications

Two case studies are discussed in this part:

16.1. The formal specification of a Research Award Policy (RAP)

16.2. The formal specification of an Automated Teller Machine (ATM)
16.1 The formal specification of a Research Award Policy (RAP)

1. The background

- The Research Award Policy aims to encourage all the students in Shaoying Liu's laboratory to work hard toward their degrees.

- The specification that defines the policy is intended to serve as a fundamental and precise principle for managing students' research activities, and as a firm foundation for implementing a program system that assists to carry out the policy.
2. The process of constructing the formal specification:

Informal specification $\rightarrow$

semi-formal specification $\rightarrow$

formal specification

The informal and semi-formal specifications are for capturing functional requirements, while the formal specification is for abstract design.
Informal specification

1. Desired function:

Research Award Policy Performing System that awards properly the students who have made publications in recognized conferences or journals.
2. Award policy constraints:

2.1 If student A publishes a paper earlier than student B according to the time when their acceptance notification is received, student A will be awarded earlier than student B.

2.2 A regular paper is awarded more money than a short paper.

2.3 All the awards are made based on the availability of the usable and sufficient research grants in each fiscal year.
2.4 If student A cannot be awarded due to the lack of available research grants in the present fiscal year, he or she will be awarded when the usable and sufficient research grants become available.

2.5 The award policy is also effective to the students who have officially left the laboratory within one year time (e.g., graduation or stopped study), but will not be effective to those who have officially left the laboratory for more than one year (one year = 365 days).
1.1 Decomposition of the desired function:

1.1.1 Register students
1.1.2 Remove lab students
1.1.3 Remove left students
1.1.4 Receive notification
1.1.5 Award money
1.1.6 Update budget
Semi-formal specification

What to do in this stage:

- Organize the specification as a set of inter-related modules conforming to the SOFL syntax.
- Define all the necessary data types for defining the involved data resources.
- Relate data resources, which are declared as variables of appropriate types, with operations, which are defined as processes, and organize them properly in modules.
- Incorporate the policies on operations into either the pre and postconditions of the corresponding processes or the invariants of the related types and/or state variables.
Define each process and function (if any) with pre and postconditions, but leave the contents of the pre and postconditions informal.

When it is necessary, draw a CDFD for a module, but the CDFD may not be a complete one.

Define composite types in a way the common fields can be reused, that is, try to build a hierarchy of related composite types.
module SYSTEM_Research_Award_Policy;

type
StudentsInTheLab = set of Student;
StudentLeftTheLab = StudentsInTheLab;
/*StudentsLeftTheLab and StudentsInTheLab share
the same type, but they are used for defining
different variables and for keeping different data. */
IDtype = string;
Student = composed of
  id: IDtype
  name: string
  status: StudentStatus
end;
StudentStatus = given ;
UnpaidStudents = seq of WaitingStudent;
UsableResearchBudget = composed of
   award_money: nat0
   travel_money: nat0
end;

/*The unit of both award_money and travel_money is 1000 JPY */
AcceptNotification = given;
/*not sure what fields are needed values of this type at this stage */
WaitingStudent = AcceptNotification;
/*WaitingStudent shares the same structure as that of AcceptNotification */
TypeOfPublication = {<International_Conference>,
    <National_Conference>, <Journal>, <Others>};
JournalNames = {<IEEE TSE>,
    <Approved_Journal>};
ConferenceNames = {<ICFEM>, <FM>, <FEM>,
    <ICSE>, <ICECCS>, <APSEC>,
    <Approved_Conference>};
Location = {<Europe>, <North_America>,
    <South_America>, <Asia>, <Australia_NewZealand>,
    <Africa>, <Others>}
Award / UsableResearchBudget = composed of
    name: string
    end;
Grants = UsableResearchBudget;
var
ext #lab_students: StudentsInTheLab;
ext #left_students: StudentLeftTheLab;
ext #unpaid_students: UnpaidStudents;
ext #research_budget: UsableResearchBudget;
inv

(1) Every student in the store lab_students has a unique name.
(2) accept_note and award_com should become available exclusively to ensure that the related processes Receive_Notification and Award_Money do not execute concurrently.
(3) Award_com becomes available only when the store unpaid_students is not empty.
(4) award_com and grants should become available exclusively to ensure that the related processes Award_Money and Update_Budget do not execute concurrently.
behav CDFD_No1;
process Register_Student(student: Student)
ext wr lab_students: StudentsInTheLab
post add the input student to the store
    lab_students, if it does not exist in the
    lab_students.
end_process;

process Remove_Student(students_to_remove: set of Student)
ext wr lab_students: StudentsInTheLab
    wr left_students: set of Student
post remove all the students in the set
    students_to_remove and add them to the store
    left_students.
end_process;
Process Remove_Left_Student(
    leftstudents_to_remove: set of Student)
ext wr left_students: set of Student
post remove all the students in the set
    leftstudents_to_remove from the store
    left_students.
end_process;
process Receive_Notification(
    accept_note: AcceptNotification)
ext rd lab_students: StudentsInTheLab
    wr unpaid_students: seq of WaitingStudent
pre the first author or second author (if the first one is "Shaoying Liu") must be contained in the store lab_students
post add the first author (if it is not "Shaoying Liu") or second author (if the first author is "Shaoying Liu") of the paper contained in the input accept_note to the end of the store unpaid_students (a sequence of WaitingStudents)
end_process;
process Award_Money(
    award_com: AwardCommand)
    award: Award
ext rd lab_students: StudentsInTheLab
    wr unpaid_students: UnpaidStudents
    wr research_budget:
        UsableResearchBudget
pre there exists one student in the store unpaid_students and there is sufficient budget for award
post determine the specific amount of money for travel and effort award based on the type of publication, long or short paper, and location of the related conference. Also, it removes the student who has been awarded from the store unpaid_student and updates the store research_budget.
end_process;
process Update_Budget(.grants: Grants)
ext wr research_budget:
    UsableResearchBudget
post add grants to the store research_budget
end_process
end_module.
module SYSTEM_Research_Award_Policy;

type
  StudentsInTheLab = set of Student;
  StudentLeftTheLab = StudentsInTheLab;
  /*StudentsLeftTheLab and StudentsInTheLab share the same type, but they are used for defining different variables and for keeping different data. */

IDtype = string;

Student = composed of
  id: IDtype
  name: string
  status: StudentStatus
end;

StudentStatus = {<Undergraduate_1>, <Undergraduate_2>,
  <Undergraduate_3>, <Undergraduate_4>, <MSc_1>,
  <MSc_2>, <PhD_1>, <PhD_2>, <PhD_3>, <Others>};
UnpaidStudents = seq of WaitingStudent;
UsableResearchBudget = composed of
  award_money: nat0
  travel_money: nat0
end;

/*The unit of both award_money and travel_money is 1000 JPY */
AcceptNotification = composed of
  paper_title: string
  first_author: string
  second_author: string
  type_of_publication: TypeOfPublication
  long_or_short: bool
  /*true for long paper and false for short paper */
  journal_name: JournalNames
  conference_name: ConferenceNames
  conference_place: Location
end;
WaitingStudent = AcceptNotification;
/*WaitingStudent shares the same structure as that of AcceptNotification */
TypeOfPublication = {<International_Conference>, <National_Conference>, <Journal>, <Others>};
JournalNames = {<IEEE TSE>, <Approved_Journal>};
ConferenceNames = {<ICFEM>, <FM>, <FEM>, <ICSE>, <ICECCS>, <APSEC>, <Approved_Conference>};
Location = {<Europe>, <North_America>, <South_America>, <Asia>, <Australia_NewZeland>, <Africa>, <Others>}
Award / UsableResearchBudget = composed of
   name: string
   end;
Grants = UsableResearchBudget;
var
define 
  ext #lab_students: StudentsInTheLab;
define 
  ext #left_students: StudentLeftTheLab;
define 
  ext #unpaid_students: UnpaidStudents;
define 
  ext #research_budget: UsableResearchBudget;

inv

forall [student: lab_students] | not exists [another_student: lab_students] | student.name = another_student.name;

/* All the students in lab_students have a different name. */
bound(award_com) and bound(accept_note) = false;

/* award_com and accept_note cannot be available at the same time. */
bound(award_com) => len(unpaid_students) > 0;

/* Only when the unpaid_students is not empty, can award_com becomes available. */
bound(award_com) and bound(grants) = false;

/* award_com and grants cannot be available at the same time */
behav CDFD_No1;

[Diagram with boxes and arrows, representing a behavioral diagram related to student registration, removal, and budget update processes.]
process Register_Student(student: Student)
ext wr lab_students: StudentsInTheLab
post  lab_students = union(~lab_students, 
                               {student})
comment
  This process adds the input student to the 
  store lab_students, if it does not exist in the 
  lab_students.
end_process;
process Remove_Student(students_to_remove: set of Student)
  ext wr lab_students: StudentsInTheLab
  wr left_students: set of Student
  post lab_students = diff(~lab_students, students_to_remove)
  and
  left_students = union(~left_students, students_to_remove)

comment
  This process removes all the students in the set students_to_remove and add them to the store left_students.
end_process;
process Remove_Left_Student(
    leftstudents_to_remove: set of Student)
    ext wr left_students: set of Student
    post left_students = diff(~left_students, 
                            leftstudents_to_remove)
    comment
    This process removes all the students in the 
    set leftstudents_to_remove from the store 
    left_students.
end_process;
process Receive_Notification(
    accept_note: AcceptNotification)
ext rd lab_students: StudentsInTheLab
wr unpaid_students: seq of WaitingStudent
pre exists[student: lab_students] |
    student.name = accept_note.first_author or
    student.name = accept_note.second_author and
    accept_note.first_author = "Shaoying Liu"
post let candidate_student =
    mk_WaitingStudent(accept_note.paper_title,
        accept_note.first_author,
        accept_note.second_author,
        accept_note.type_of_publication,
        accept_note.long_or_short,
        accept_note.journal_name,
        accept_note.conference_name,
        accept_note.conference_place)
    in
        unpaid_students = conc(~unpaid_students, [candidate_student])
end_process;
process Award_Money(
    award_com: AwardCommand)
    award: Award
ext rd lab_students: StudentsInTheLab
    wr unpaid_students: UnpaidStudents
    wr research_budget:
        UsableResearchBudget
pre len(unpaid_students) > 0 and
    research_budget.award_money > 0 and
    research_budget.travel_money > 0
post let candidate_student = hd(unpaid_students) in
    ( if candidate_student.type_of_publication = <International_Conference>
        then if candidate_student.conference_name
            inset {<ICSE>, <ICFEM>, <FM>}
            then if candidate_student.long_or_short
                then
                    award = Calculate_Money(candidate_student, conference_place, 1)
                else
                    award = Calculate_Money(candidate_student, conference_place, 2)
    else
        award = Calculate_Money(candidate_student, conference_place, 2)
else if candidate_student.long_or_short
    then award = Calculate_Money(candidate_student,
                                  conference_place, 3)

    else award = Calculate_Money(candidate_student,
                                   conference_place, 4)

else if candidate_student.type_of_publication = "Journal"
then if candidate_student.journal_name = "IEEE TSE"
    then if candidate_student.long_or_short
        then
            award = Calculate_Money(candidate_student,
                                      conference_place, 5)

        else
            award = Calculate_Money(candidate_student,
                                      conference_place, 6)
else if candidate_student.long_or_short
    then
        award = Calculate_Money(candidate_student,
                                conference_place, 7)
    else
        award = Calculate_Money(candidate_student,
                                conference_place, 8)
else if candidate_student.type_of_publication = <National_Conference>
    then if candidate_student.long_or_short
        then
            award = Calculate_Money(candidate_student,
                                    conference_place, 9)
        else
            award = Calculate_Money(candidate_student,
                                    conference_place, 10)
else if candidate_student.long_or_short
   then
      award = Calculate_Money(candidate_student, conference_place, 11)
   else
      award = Calculate_Money(candidate_student, conference_place, 12)

) and
unpaid_students = tl(~unpaid_students) and
research_budget = mk_UsableResearchMoney(
   ~research_budget.award_money -
   award.award_money,
   ~research_budget.travel_money -
   award.travel_money)
comment

The precondition of this process requires that unpaid_students contains at least one student and both the award money and travel money of the research_budget must be greater than 0. Under this precondition, the process determines the specific amount of money for travel and effort award based on the type of publication, long or short paper, and location of the related conference. Also, it removes the student who has been awarded from the store unpaid_student and updates the store research_budget.

end_process;
process Update_Budget(grants: Grants) 

ext wr research_budget:
   UsableResearchBudget

post research_budget.award_money = ~research_budget.award_money + grants.award_money and
   research_budget.travel_money = ~research_budget.travel_money + grants.travel_money

end_process;
function Calculate_Money(candidate_student: WaitingStudent, 
    conference_place: Location, paper_situation: nat): Award 
post let travel_expense: nat0 in 
    if candidate_student.first_author = "Shaoying Liu" 
then Calculate_Money = mk_Award(50, 0, candidate_student.second_author) /*when the student is the second author */ 
else 
    case paper_situation of 
    1 --> if conference_place = <Asia> 
        then Calculate_Money = mk_Award(200, travel_expense, 
                                        candidate_student.name) 
                                        and travel_expense <= 150 
                                        else Calculate_Money = mk_Award(200, travel_expense, 
                                        candidate_student.name) 
                                        and travel_expense <= 200; 
    2 --> if conference_place = <Asia> 
        then Calculate_Money = mk_Award(0, travel_expense, 
                                        candidate_student.name) 
                                        and travel_expense <= 150 
                                        else Calculate_Money = mk_Award(0, travel_expense, 
                                        candidate_student.name) 
                                        and travel_expense <= 200;
3 --> if conference_place = <Asia>
then Calculate_Money = mk_Award(150, travel_expense, candidate_student.name)
   and travel_expense <= 150
else Calculate_Money = mk_Award(150, travel_expense, candidate_student.name)
   and travel_expense <= 200;
4 --> if conference_place = <Asia>
then Calculate_Money = mk_Award(0, travel_expense, candidate_student.name)
   and travel_expense <= 150
else Calculate_Money = mk_Award(0, travel_expense, candidate_student.name)
   and travel_expense <= 200;
5 --> Calculate_Money = mk_Award(400, 0, candidate_student.name);
6 --> Calculate_Money = mk_Award(150, 0, candidate_student.name);
7 --> Calculate_Money = mk_Award(300, 0, candidate_student.name);
8 --> Calculate_Money = mk_Award(50, 0, candidate_student.name);
9 --> Calculate_Money = mk_Award(10, travel_expense, candidate_student.name)
    and travel_expense <= 50;
10 --> Calculate_Money = mk_Award(0, travel_expense, candidate_student.name)
    and travel_expense <= 50;
11, 12 --> Calculate_Money = mk_Award(0, 0, candidate_student.name)
    end_case
end_function;
end_module.
16.2. The formal specification of an Automated Teller Machine (ATM)

1. The background:

The case study is intended to show the entire process of developing a formal detailed design specification from an informal requirements specification based on Mizuho Bank ATMs services available at the website:

http://www.mizuhobank.co.jp/start/atm/index.html
2. The process of constructing the formal detailed design specification:

Informal specification →

semi-formal specification →

formal abstract design specification →

formal detailed design specification
Informal specification

1. The desired functional services: the top-level module:

(1) Operations on current account.
(2) Operations on savings account
(3) Transfer money between accounts
(4) Manage foreign currency account
(5) Change password
2. Decomposition of function (1) in the top-level module

2.1 Operations

(1.1) Deposit
/* put money into the current account */
(1.2) Withdraw
/* get money out of the current account */
(1.3) Show balance
/* display the balance of the current account */
(1.4) Print out transaction records
/* print a list of transactions so far */
2.2 Policy on operations

(1) Withdraw:
   (1.1) Maximum amount to be withdrawn each time is 1,000,000 JPY. /* JPY = Japanese yen */
   (1.2) Maximum amount to be withdrawn each day is 5,000,000 JPY.
   (1.3) No overdraw is allowed.

(2) Deposit: at most 1,000,000 JPY can be deposited each time.

(3) Password is required for all the four operations given above.

(4) Bank-card is required for all the four operations.

(5) Bank-book is required only for operation 1.4: print out transaction records.
2.3 Data resources
(1) Each customer has ONE current account.
(2) It is necessary to record the following data items in the system for each customer:
   (2.1) full name
   (2.2) account number
   (2.3) password
3. Decomposition of function (2) in the top-level module

3.1 Operations

(2.1) Deposit /* put money into the savings account */

(2.2) Application of withdrawing money from the savings account. /* withdrawing money from the savings account needs application in advance */

(2.3) Withdraw /*only after a customer submits an application, can he withdraw money from the savings account. */

(2.4) Show balance

(2.5) Print out transaction records
3.2 Data resources
(1) Each customer has ONE savings account.
(2) Each customer needs the following data items to be recorded in the system:
  (2.1) full name
  (2.2) account number
  (2.3) password
3.3 Policy on operations

(1) After every 6 months the customer can withdraw money and money cannot be withdrawn without application in advance.

(2) The maximum amount to be withdrawn each time is 3,000,000 JPY. That is, when applying for the withdraw, the customer can apply for up to 3,000,000 JPY.

(3) The maximum amount to deposit is 3,000,000 JPY.
4. Decomposition of function (3) in the top-level module

4.1 Operations
(1) Transfer money between the current and the savings account using cash-card

4.2 Data resources
(1) The current and savings accounts.

4.3 Policy on operations
(1) The maximum amount of each transfer transaction is 1,000,000 JPY.
5. Decomposition of function (4) in the top-level module

5.1 Operations

(1) Purchase US dollars using the money of the current account.
(2) Sell US dollars to JPY and deposit the money into the current account.
(3) Purchase US dollars using cash and deposit the dollars into the foreign currency account.
(4) Withdraw JPY from the foreign currency account. /*The JPY is converted from US dollars*/
(5) Show balance.
5.2 Data resources

(1) Each customer needs a foreign currency account.

(2) Each customer's following data items needs to be recorded in the account:
   (2.1) full name
   (2.2) account number
   (2.3) password
6. Decomposition of function (5) in the top-level module

6.1 Operations

(1) Change password for the current account.
(2) Change password for the savings account.
(3) Change password for the foreign currency account.
module SYSTEM_ATM;
const
maximum_withdraw_once = 1,000,000;
/*The unit is JPY, likewise for the following constants.*/
maximum_withdraw_day = 5,000,000;
maximum_deposit_once = 1,000,000;
maximum_withdraw_application = 3,000,000;
ATM_no = i; /*i is any natural number*/
type
CustomerInf = composed of
  account_no: nat0
  pass: Password
end;
Password = nat0;
/*A password is a natural number or zero*/
AccountInf = composed of
  name: string
  /*The customer's full name */
  balance: nat0
  /*The unit is JPY*/
  transaction_history: seq of Transaction
end;
CurrentAccountInf = AccountInf;
SavingsAccountInf / AccountInf =
    composed of
    withdraw_application_amount: nat0
    application_status: bool
    /*true for yes, false for no */
end;
ForeignCurrencyAccountInf / CustomerInf =
    composed of
    name: string
    balance: real /*The unit is US dollar */
end;
CurrentAccountFile =
    map CustomerInf to CurrentAccountInf;

SavingsAccountFile =
    map CustomerInf to SavingsAccountInf;

ForeignCurrencyAccountFile =
    map CustomerInf to
        ForeignCurrencyAccountInf;
ApplicationNotice =
    composed of
    application_amount: nat0;
    appication_successful: bool;
    end;
Transaction =
    composed of
    date: Date
    time: Time
    payment: nat0
    deposit: nat0
    balance: nat0
    atm_no: nat0
    end;
Date = Day * Month * Year;
Day = nat0;   Month = nat0;   Year = nat0;
var

ext #current_accounts: CurrentAccountFile;
ext #savings_accounts: SavingsAccountFile;
ext #foreign_currency_accounts: ForeignCurrencyAccountFile;

ext #today: Date; /*The variable today is assumed to change to reflect the date of today in calendar.*/

ext #current_time: Time; /*This variable represents a clock telling the current time */
inv

forall[x: CustomerInf] | not exists[y: CustomerInf] | x.account_no = y.account_no; /*Each customer's account is unique */

forall[x, y: Transaction] | x <> y;
/*All the transactions are different. */

/* No CDFD is drawn for this module in this stage for there is no concern about the relations among the processes involved in this module. */
process Manage_Current_Account(current: sign)
end_process;

process Manage_Savings_Account(savings: sign)
end_process;

process Manage_Transfer(transfer: sign)
end_process;

process Manage_Foreign_Currency_Account(foreign_currency: sign)
end_process;

process Change_Password(change_pass: sign)
end_process;

end_module;
module Manage_Current_Account_Decom / SYSTEM_ATM;

type
  Notice = composed of
    transaction_account: nat0
    updated_balance: nat0
  end;

var
  ext current_accounts: CurrentAccountFile;
process Current_Authorization(
    current_inf: CustomerInf)
permission: sign | e_mesg1: string
ext rd current_accounts
post if the input account_no and password
    match those of the customer's current
    account in the store current_accounts
    then generate output permission
    else output an error message
end_process;
process Current_Deposit(
    permission: sign,
    current_inf: CustomerInf,
    deposit_amount: nat0)
    notice: Notice | warning: string

ext wr current_accounts;
post if the input deposit_amount is less than or equal to
    the maximum_deposit_once
    then (1) add the deposit_amount to current_account
    (2) give the customer a notice showing the amount
        of deposit and the updated balance
    (3) update the transaction history of the account
    else give a warning message to indicate that the
        amount is over the limit.
end_process;
process Current_Withdraw(permission: sign,
current_inf: CustomerInf,
amount: nat0)
notice: Notice | warning2: string
ext wr current_accounts
post if the input amount is less than or equal to the
balance of the account and the
maximum_withdraw_once
then (1) output the cash of the requested amount
(2) reduce the withdraw amount from the
balance
(3) update the transaction history of the
account
(4) give the notice
else generate the warning message
end_process;
process Current_Show_Balance(
    permission: sign,
    current_inf: CustomerInf)
    balance: nat0 | warning3: string
ext rd current_accounts
post if the input account_no and pass match
    those of the customer in the store
    current_accounts
    then display the balance of the customer's
    current account
else issue an error message
end_process;
process Current_Print_Transaction_Records(
    permission: sign,
    current_inf: CustomerInf,
    date: Date)
transaction_records: TransactionRecords
ext rd current_accounts
post print out the transaction records since the input date
end_process;
end_module;
module Manage_Savings_Account_Decom / SYSTEM_ATM;

var ext savings_accounts:
    SavingsAccountFile;

    ... /* process specifications */
end_module;
module Manage_Foreign_Currency_Account / SYSTEM_ATM;
  type   ExchangeNotice = composed of
         amount_in_yen: nat0
         current_balance: nat0
         foreign_balance: real
         exchange_rate: nat0 /*US$1 = n JPY */
     end;

CashExchangeNotice = composed of
         amount_in_yen: nat0
         amount_in_dollar: real
         foreign_balance: real
     end;

var
  ext current_accounts: SavingsAccountFile;
  ext foreign_currency_accounts: ForeignCurrencyAccountFile;
      ... /* process specifications */
end_module;
module Change_Password_Decom / SYSTEM_ATM;

var
  ext #current_accounts
  ext #savings_accounts
  ext #foreign_currency_accounts
  ext #all_used_passwords: set of Passwords

  ... /* process specifications */

end_module.
module SYSTEM_ATM;

... /* the same as those in the semi-formal module */

inv

... /*the same as those in the semi-formal module */

forall[x, y: {current, savings, transfer, foreign_currency, change_pass}] | bound(x) and bound(y) = false; /*Any two of the input control data flows cannot become available at the same time */

behav CDFD_No1;
process Manage_Current_Account(
current: sign)
ext wr current_accounts
end_process;
process Manage_Savings_Account(
savings: sign)
ext wr savings_accounts
end_process;
process Manage_Transfer(transfer: sign)
ext wr current_accounts
wr savings_accounts
end_process;
process Manage_Foreign_Currency_Account (foreign_currency: sign)
ext wr foreign_currency_accounts
end_process;

process Change_Password(change_pass: sign)
ext wr all_used_passwords
  wr foreign_currency_accounts
  wr savings_accounts
  wr current_accounts
end_process;
end_module;
module Manage_Current_Account_Decom / SYSTEM_ATM;

type
  ... /* omit the same type declarations */
  OutputDevice = seq of universal;
  ServiceCollection = {<1>, <2>, <3>, <4>};

var
  ... /* omit the same variable declarations */
  ext #output_device: OutputDevice;

inv
  forall[x, y: {deposit, withdraw, s_balance, p_transactions}] | bound(x) and bound(y) = false;

behav CDFD_No2;
process Select_Services(deposit, a: sign | b, withdraw: sign | c, s_balance: sign | d, p_transactions: sign)
    sel: ServiceCollection
post bound(deposit) and sel = <1> or bound(withdraw) and sel = <2> or bound(s_balance) and sel = <3> or bound(p_transactions) and sel = <4>
comment
    The output data flow sel is decided to take different value depending on the availability of the input data flows.
end_process;
process Current_Authorization(sel: ServiceCollection,
current_inf: CustomerInf)
current_inf1: CustomerInf |
current_inf2: CustomerInf |
current_inf3: CustomerInf |
current_inf4: CustomerInf |
e_msg1: string

ext rd current_accounts
post if current_inf inset dom(current_accounts)
  then case sel of
    <1> --> current_inf1 = current_inf;
    <2> --> current_inf2 = current_inf;
    <3> --> current_inf3 = current_inf;
    <4> --> current_inf4 = current_inf;
  end_case
  else e_msg1 = "Your password or account number is incorrect."

comment
  if the input account_no and password match those of the customer's current
  account in the store current_accounts
  then generate output permission
  else output an error message.
end_process;
process Current_Deposit(deposit_amount: nat0,
current_inf1: CustomerInf)
notice1: Notice |
warning1: string

ext wr current_accounts;
post if deposit_amount <= maximum_deposit_once
then current_accounts =
    override(~current_accounts,
        {current_inf1 -->
            modify(~current_accounts(current_inf1),
                balance -->
                ~current_accounts(current_inf1).balance + deposit_amount,
                transaction_history -->
                conc(~current_accounts(current_inf1).transaction_history,
                    [Get_Transaction(current_accounts, today, current_time, 0,
                        deposit_amount,
                        current_inf1)])
        } ) ) and
    notice1 = mk_Notice(deposit_amount,
        current_accounts(current_inf1).balance))
else warning1 = "Your amount is over 1000000 yen limit."
if the input deposit_amount is less than or equal to the maximum_deposit_once
then   (1) add the deposit_amount to the current_account (2) give the customer a notice showing the amount of deposit and the updated balance (3) update the transaction history of the account
else give a warning message to indicate that the amount is over the limit.
end_process;
process Current_Withdraw(current_inf2: CustomerInf, amount: nat0)
    notice2: Notice |
    warning2: string

ext wr current_accounts
post if amount <= maximum_withdraw_one and
    amount <= ~current_accounts(current_inf2).balance
then  current_accounts =
    override(~current_accounts, 
        {current_inf2 --> 
            modify(~current_accounts(current_inf2), 
                balance -->  ~current_accounts(current_inf2).balance - 
                amount, 
                transaction_history --> 
                conc(~current_accounts(current_inf2).transaction_history, 
                    [Get_Transaction(current_accounts, today, current_time, 
                        amount, 0, 
                        current_inf2)])} ) )
      and 
    notice2 = mk_Notice(amount, current_accounts(current_inf2).balance))
else warning2 = "Your withdraw amount is over the limit."
comment
if the input amount is less than or equal to the balance of the account and the maximum_withdraw_once
then (1) output the cash of the requested amount (2) reduce the withdraw amount from the balance (3) update the transaction history of the account (4) give a notice
else generate a warning message
end_process;
process Current_Show_Balance(
    current_inf3: CustomerInf)
    balance: nat0
ext rd current_accounts
post balance =
    current_accounts(current_inf3).balance
comment
display the balance of the customer's current account
end_process;
process Current_Print_Transaction_Records(
    current_inf4: CustomerInf, date: Date)
    transaction_records: TransactionRecords
ext rd current_accounts
post let transactions =
    current_accounts(current_inf4).transaction_history
    in let i = get({i | i: inds(transactions) &
        transactions(i).date = date})
    in  transaction_records =
        transactions(i, ..., len(transactions))
comment
    print out the transaction records since the input date
end_process;
process Display_Information(notice1: Notice | notice2: Notice | balance: nat0 | transaction_records: TransactionRecords)

ext wr output_device
post bound(notice1) and
    output_device = conc(~output_device, [notice1]) or
post bound(notice2) and
    output_device = conc(~output_device, [notice2]) or
post bound(balance) and
    output_device = conc(~output_device, [balance]) or
post bound(transaction_records) and
    output_device = conc(~output_device, [transactions_records])

comment
Display the input data flows onto the output device based on their availability.
end_process;
process Display_Message(warning1: string | 
    warning2: string |
    e_mesg1: string)

ext wr output_device
post bound(warning1) and
    output_device = conc(~output_device, [warning1])
or
bound(warning2) and
    output_device = conc(~output_device, [warning2])
or
bound(e_mesg1) and
    output_device = conc(~output_device, [e_mesg1])

comment
    Display the input data flows onto the output device based on their availability.
end_process;
function Get_Transaction(
current_accounts: CurrentAccountFile,
to_day: Date,
time: Time,
pay_amount: nat0,
deposit_amount: nat0,
customer_inf: CustomerInf):
  Transaction
  == mk_Transaction(to_day, time, pay_amount,
                      deposit_amount,
                      current_accounts(customer_inf).balance,
                      ATM_no)
end_function
end_module;
module Manage_Savings_Account_Decom / SYSTEM_ATM;
...
end_module;
module Manage_Foreign_Currency_Account / SYSTEM_ATM;
...
end_module;
module Change_Password_Decom / SYSTEM_ATM;
...
end_module.
Two major tasks in writing the detailed design specification:

- Transform the structured design specification resulting from the abstract design into an object-oriented design specification by converting and developing all the involved composite types (as well as product types and union types) into classes.

- Refine the implicit specification of each process into an explicit specification providing more algorithmic expression of the defined behavior of the process.
module SYSTEM_ATM;
    /* updated declarations, omitted for brevity */
behav CDFD_No1;
process Manage_Current_Account(current: sign) ext wr current_accounts
end_process;
process Manage_Savings_Account(savings: sign) ext wr savings_accounts
end_process;
process Manage_Transfer(transfer: sign)
    ext wr current_accounts
    wr savings_accounts
end_process;
process Manage_Foreign_Currency_Account(
    foreign_currency: sign)
    ext wr foreign_currency_accounts
end_process;
process Change_Password(change_pass: sign)
    ext wr all_used_passwords
    wr foreign_currency_accounts
    wr savings_accounts
    wr current_accounts
end_process;
end_module;
Notice:
The top-level module SYSTEM_ATM is almost unchanged, except the elimination of the composite types that are converted into the classes definitions given below.
class CustomerInf;

var
    account_no: nat0;
    pass: nat0;

method Init()
post account_no = 0 and pass = 0
end_method;
end_class;
class AccountInf;
var
  name: string;
  balance: nat0;
  transaction_history: seq of Transaction;
method Init()
  explicit
  begin
    name := ""; /*empty string*/
    balance := 0;
    transaction_history := [ ] /*empty sequence*/
  end
end_method;
method Increase_Balance(amount: nat0)
ext wr balance
post balance = ~balance + amount
end_method;

method Decrease_Balance(amount: nat0)
ext wr balance
post balance = ~balance - amount
end_method;

method Update_Transaction_History(transaction: Transaction)
ext wr transaction_history
post transaction_history = conc(~transaction_history, [transaction])
end_method;
end_class;
class CurrentAccountInf / AccountInf;
end_class;

class SavingsAccountInf / AccountInf;
var
    withdraw_application_amount: nat0;    application_status: bool;

method Init()
    post withdraw_application_amount = 0 and
        application_status = true
end_method;

method Set_Application_Amount(amount: nat0)
    ext wr withdraw_application_amount
        wr application_status
    post withdraw_application_amount = amount and application_status = true
end_method;
end_class;
class ForeignCurrencyAccountInf / CustomerInf;

var

  name: string;
  balance: real;

method Init()

post name = "" and balance = 0.0
end_method;
end_class;
class ApplicationNotice;

var
  application_amount: nat0;
  application_successful: bool;

method Init()
  post application_amount = 0 and
    application_successful = true
end_method;
end_class;
class Transaction;
  type
    CurrentAccountsFile = SYSTEM_ATM.CurrentAccountsFile;
  var
    date: Date;
    time: Time;
    payment: nat0;
    deposit: nat0;
    balance: nat0;
    atm_no: nat0;
  method Init()
    explicit
    begin
      date := new Date;
      time := new Time;
      payment = 0;
      deposit = 0;
      atm_no = 0;
    end
end_method;
method Get_Transaction(current_accounts: CurrentAccountsFile, 
  date1: SYSTEM_ATM.Date, 
  time1: SYSTEM_ATM.Time, 
  pay1: nat0, 
  deposit1: nat0, 
  balance1: nat0, 
  current_inf: CustomerInf)

  ext wr date 
    wr time 
    wr payment 
    wr deposit 
    wr balance 
  explicit 
  begin 
    date := date1; 
    time := time1; 
    payment := pay1; 
    deposit := deposit1; 
    balance := current_accounts(current_inf).balance; 
  end 
end_method;
Method Get_Savings_Transaction(savings_accounts: SavingsAccountsFile, 
    date1: SYSTEM_ATM.Date, 
    time1: SYSTEM_ATM.Time, 
    pay1: nat0, 
    deposit1: nat0, 
    balance1: nat0, 
    customer_inf: CustomerInf) 

ext wr date 
    wr time 
    wr payment 
    wr deposit 
    wr balance 
explicit 
    begin 
        date := date1; 
        time := time1; 
        payment := pay1; 
        deposit := deposit1; 
        balance := savings_accounts(customer_inf).balance; 
    end 
end_method; 
end_class;
class Date;
  var
    day: nat0;
    month: nat0;
    year: nat0;
  method Init()
    post day = 0 and month = 0 and year = 0
  end_method;
end_class;
class Notice;
    var
        transaction_amount: nat0;  updated_balance: nat0;
    method Init()
    post transaction_amount = 0 and
        updated_balance = 0
    end_method;
    method Make_Notice(amount: nat0, balance: nat0)
    ext wr transaction_amount
    ext wr updated_balance
    explicit
    begin
        transaction_amount := amount;
        updated_balance := balance
    end
    end_method;
end_class;
class TransferNotice;

    var
        transaction_amount: nat0
        from_account_balance: nat0
        to_account_balance: nat0
    method Init()
        post transaction_amount = 0 and from_account_balance = 0
            and to_account_balance = 0
    end_method;

    method Make_TransferNotice(transfer_amount1: nat0,
        from_balance: nat0,
        to_balance: nat0)
        ext wr transaction_amount
            wr from_account_balance
            wr to_account_balance
        post transaction_amount = transfer_amount1 and
            from_account_balance = from_balance and
            to_account_balance = to_balance
    end_method
end_class;
module Manage_Current_Account_Decom /
  SYSTEM_ATM;

  ... /* the declarations are omitted */

beHAV CDFD_No2;

process Select_Services(deposit, a: sign | b, withdraw: sign | c, s_balance: sign | d, p_transactions: sign)
  sel: ServiceCollection

post bound(deposit) and sel = <1> or bound(withdraw) and sel = <2> or bound(s_balance) and sel = <3> or bound(p_transactions) and sel = <4>
explicit

if bound(deposit)
then sel := <1>
else if bound(withdraw)
    then sel := <2>
else if bound(s_balance)
    then sel := <3>
else if bound(p_transactions)
    then sel := <4>

comment
The output data flow sel is decided to take different value depending on the availability of the input data flows.

end_process;
process Current_Authorization(sel: ServiceCollection, 
current_inf: CustomerInf) 
current_inf1: CustomerInf |
current_inf2: CustomerInf |
current_inf3: CustomerInf |
current_inf4: CustomerInf |
e_mesg1: string

ext rd current_accounts 
explicit 
if current_inf inset dom(current_accounts) 
then case sel of 
    <1> --> current_inf1 := current_inf; 
    <2> --> current_inf2 := current_inf; 
    <3> --> current_inf3 := current_inf; 
    <4> --> current_inf4 := current_inf; 
end_process 
else e_mesg1 := "Your password or account number is incorrect." 
comment 
if the input account_no and password match those of the customer's current account in the store current_accounts then generate output permission 
else output an error message. 
end_process;
process Current_Deposit(deposit_amount: nat0, current_inf1: CustomerInf)
    notice1: Notice | warning1: string
    ext wr current_accounts;
    explicit
    account_inf: CurrentAccountInf;     transaction: Transaction;
    begin
    account_inf := new CurrentAccountInf;
    transaction := new Transaction;
    if deposit_amount <= maximum_deposit_once
    then
    begin
    account_inf := current_accounts(current_inf1);
    account_inf.Increase_Balance(deposit_amount);
    account_inf.Update_Transaction_History(
        transaction.Get_Transaction(current_accounts, today, current_time, 0, deposit_amount, current_inf1));
current_accounts := override(current_accounts,
    {current_inf1 --> account_inf});

notice1 := new Notice;
notice1.Make_Notice(deposit_amount,
    current_accounts(current_inf1).balance)
end
else warning1 := "Your amount is over 1000000 yen limit."
end
comment
  ...  
end_process;
process Current-Withdraw(current_inf2: CustomerInf, amount: nat0) notice2: Notice | warning2:

string
ext wr current_accounts explicit
account_inf: CurrentAccountInf;
transaction: Transaction;
begin
account_inf := new CurrentAccountInf;
transaction := new Transaction;
if amount <= maximum_withdraw_once and
amount <=
current_accounts(current_inf2).balance
then
begin
  account_inf := current_accounts(current_inf2);
  account_inf.Decrease_Balance(amount);
  account_inf.Update_Transaction_History(
    transaction.Get_Transaction(current_accounts,
      today, current_time, amount, 0, current_inf2));
  current_accounts :=
    override(current_accounts,
      {current_inf2 => account_inf});
  notice2 := new Notice;
  notice2.Make_Notice(amount,
    current_accounts(current_inf2).balance)
end
else
  warning2 := "Your amount is over 1000000 yen limit."
end
comment
...
end_process;
process Current_Show_Balance(
    current_inf3: CustomerInf)
balance: nat0
ext rd current_accounts
explicit
    balance :=
    current_accounts(current_inf3).balance
comment
    ...
end_process;
process Current_Print_Transaction_Records(
    current_inf4: CustomerInf, date: Date)
transaction_records: TransactionRecords

ext rd current_accounts
explicit
    transactions: seq of Transaction;
    index: nat0;
begin
    transactions :=
        current_accounts(current_inf4).transaction_history;
    index := get({i | i: inds(transactions) &
        transactions(i).date = date});
    transaction_records := transactions(index, ..., len(transactions))
end
comment ...
end_process;
process Display_Information(notice1: Notice | notice2: Notice | balance: nat0 | transaction_records: TransactionRecords)

ext wr output_device

explicit

if bound(notice1)
    then output_device := conc(output_device, [notice1])
else if bound(notice2)
    then output_device = conc(output_device, [notice2])
else if bound(balance)
    then output_device = conc(output_device, [balance])
else output_device = conc(output_device, [transactions_records])

comment
...
end_process;
process Display_Message(warning1: string | warning2: string | e_mesg1: string)

ext wr output_device
explicit
  if bound(warning1)
    then output_device := conc(~output_device, [warning1])
  else if bound(warning2)
    then output_device := conc(output_device, [warning2])
  else output_device := conc(output_device, [e_mesg1])

comment
  ... 
end_process;
end_module;
Exercise 16

1. Give a semi-formal specification for the module Manage_Savings_Account_Decom.

2. Give a formal abstract design specification for the module Manage_Savings_Account_Decom.

3. Write a formal detailed design specification for the module Manage_Savings_Account_Decom.