Technische Universiteit Eindhoven University of Technology

Student name:

Student number:

Examination cover sheet

(to be completed by the examiner)

Course name: Software Specification and Architecture — Course code: 2IW80, exam 2IW81

Date: 14-04-2015

Start time: 9:00 — End time : 12:00

Number of pages: 19

Number of questions: 20 multiple choice questions; 4 modeling questions

Maximum number of points/distribution of points over questions:100; Part I: 20 multiple choice questions up to two points each; Part II: 4 modeling questions up to 20 points each. You should answer at least three of these questions. Should you decide to answer four questions, we will consider the three best solutions.

Method of determining final grade: divide total of points by 10

Answering style: multiple choice questions; open questions

Exam inspection: With your instructor

Other remarks: The exam questions have been inspected by Alexander Serebrenik, Kees Huizing, Sarmen Keshishzadeh and Anton Wijs.

Instructions for students and invigilators

Permitted examination aids (to be supplied by students):

- ☐ Notebook
- ☐ Calculator
- ☐ Graphic calculator
- ☐ Lecture notes/book
- ☐ One A4 sheet of annotations
- ☐ Dictionar(y)(ies). If yes, please specify: After inspection by the invigilator a dictionary is allowed from any language to English and from English to any language.
- ☐ Other:

Important:

- examinees are only permitted to visit the toilets under supervision
- it is not permitted to leave the examination room within 15 minutes of the start and within the final 15 minutes of the examination, unless stated otherwise
- examination scripts (fully completed examination paper, stating name, student number, etc.) must always be handed in
- the house rules must be observed during the examination
- the instructions of examiners and invigilators must be followed
- no pencil cases are permitted on desks
- examinees are not permitted to share examination aids or lend them to each other

During written examinations, the following actions will in any case be deemed to constitute fraud or attempted fraud:

- using another person's proof of identity/campus card (student identity card)
- having a mobile telephone or any other type of media-carrying device on your desk or in your clothes
- using, or attempting to use, unauthorized resources and aids, such as the internet, a mobile telephone, etc.
- using a clicker that does not belong to you
- having any paper at hand other than that provided by TU/e, unless stated otherwise
- visiting the toilet (or going outside) without permission or supervision

Student name: Student number:

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

Software specification and architecture (2IW81)
Tuesday, April 14, 2015, 9:00—12:00.

Part I (40 points)

Answer the following multiple-choice questions. Each question might have multiple correct answers; you should encircle all of them in the exam text below. For each question you get

- 2 points if all correct answers have been indicated and no incorrect answers have been indicated;
- 1 point if at least one correct answer have been given, some correct answers have been missed but no incorrect answers have been indicated;
- 0 points if at least one incorrect answer has been indicated.

Question 1

SysML extends UML by including additional types of diagrams such as the Requirement Diagram. This diagram can, for instance, display requirements, packages, test cases, rationale. Relations that can be shown on the Requirement Diagram include containment, satisfaction, verification, refinement, copy, and trace. By making these relations explicit, SysML can be seen as aiming at improvement of requirements'

a) specificity;

b) measurability;

c) attainability;

d) Answer traceability.

Question 2

Which of the following statements pertaining to actors in use cases are **FALSE**?

a) Answer In presence of generalization a more general actor can always replace the specialized one.

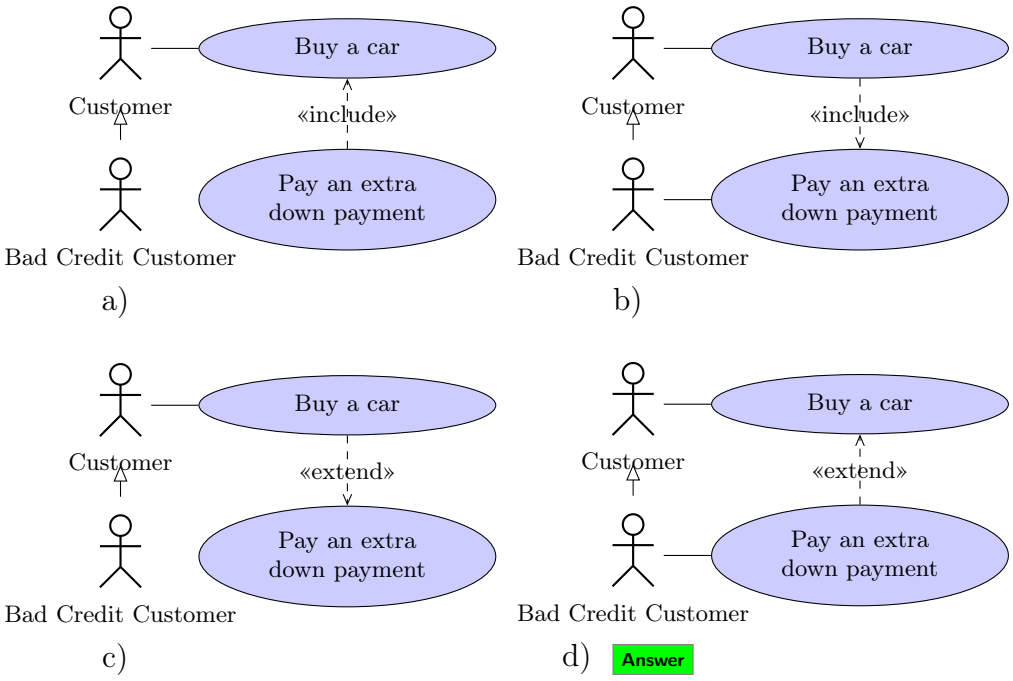
b) One user might be represented by multiple actors in the same use case.

c) Actor is a class of entities (human or computer), falling beyond the system boundaries and interacting with the system.

d) One use case might involve multiple actors.

Question 3

Read the following description. "Customers of the garage can buy cars. Customers with a bad credit should pay an extra down payment". Which of the following diagrams represent this description?



Answer

Question 4

Identify all **TRUE** statements:

a) Answer Connectors can be used to reduce visual clatter in an activity diagram.

b) Answer UML 2 supports multiple ways of indicating who is responsible for each group of activities.

c) Expansion regions are indicated with the rake symbol ⫛.

d) Activity diagrams focus on the steps/actions that should be taken and do not provide elements to represent the data being used.

Question 5

A controller for a cassette player is shown on Figure 1 on page 3. The cassette player can be controlled by buttons including play, pause, stop, flip and off. The player has a motor that can be started or stopped, and a playback head that can be engaged or disengaged. The head serves as an interface between the player and the cassette.

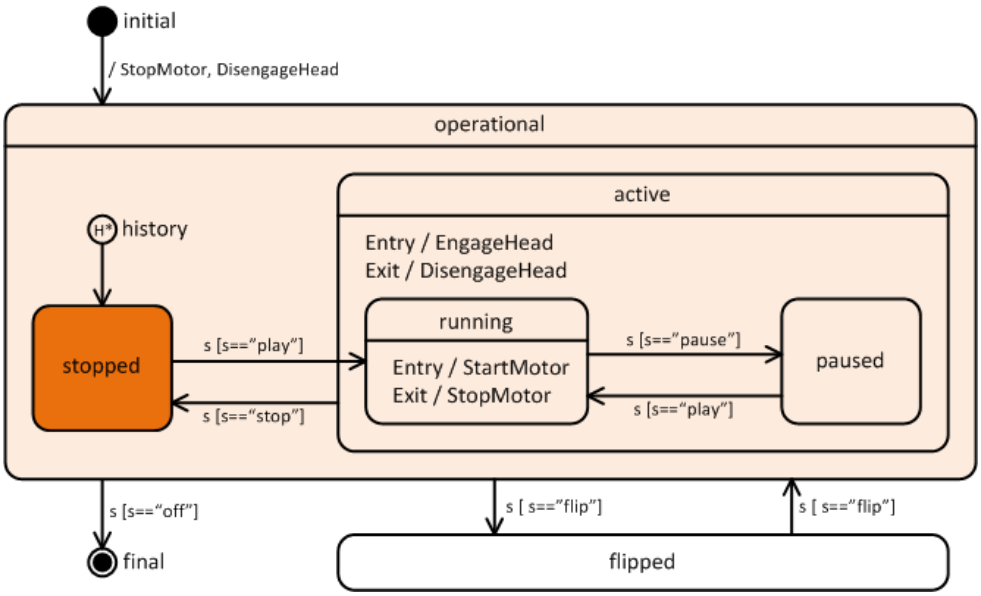


Figure 1: State machine of a controller for a cassette player

Which of the following statements are **TRUE**?

a) Answer If no previous history is available, then the *stopped* state is entered.

b) If the user presses play, then pause, followed by flip and flip, then the motor is running.

c) Correctness of answer a) would be changed if H^* would have been replaced by H.

d) Answer Whenever the motor is running, the head is engaged.

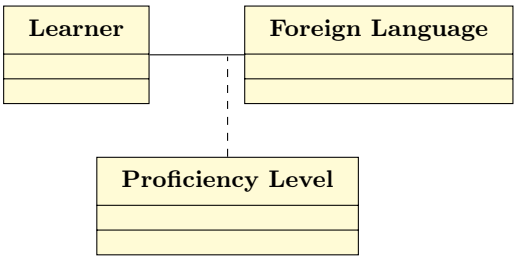


Figure 2: Class diagram for Question 6

Question 6

Which statements are implied by the class diagram in Figure 2 on page 3:

a) Every learner can study only one foreign language.

b) Answer Every learner can have only one proficiency level per foreign language.

c) Every foreign language can be studied only by one learner.

d) Every proficiency level can be recorded for at most one (learner, foreign language) pair.

Question 7

Read the following description and select the most appropriate class diagram from Figure 3 on page 4: "A company realizes projects; each project is executed by a team of employees". For the sake of simplicity, multiplicities and attributes are not indicated.

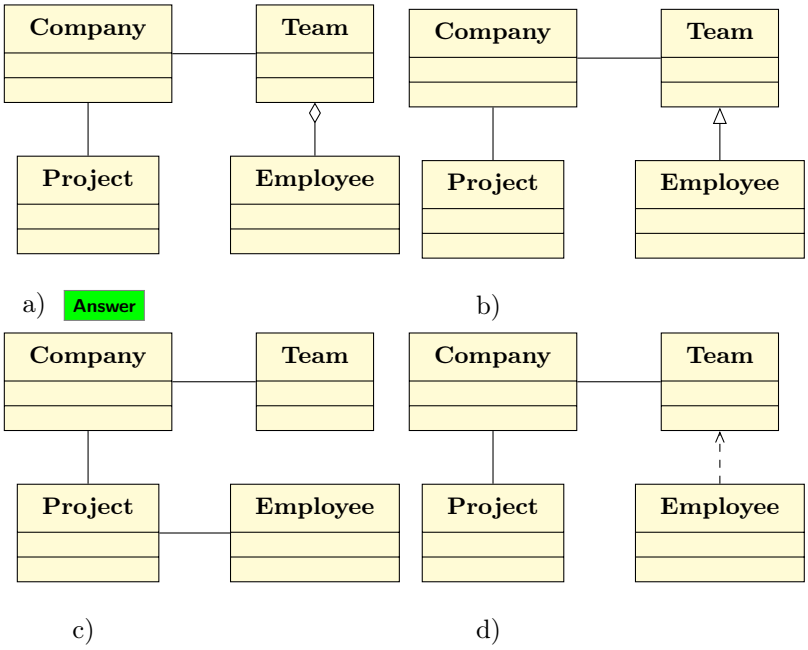
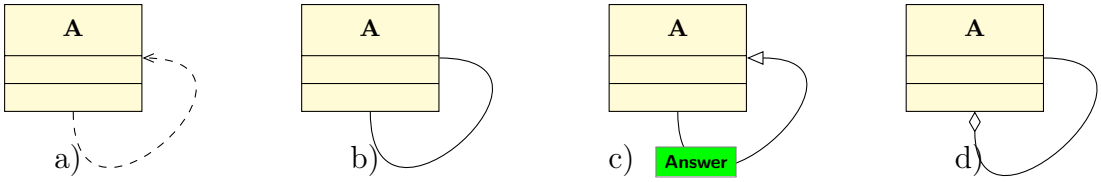


Figure 3: Class diagrams for Question 7

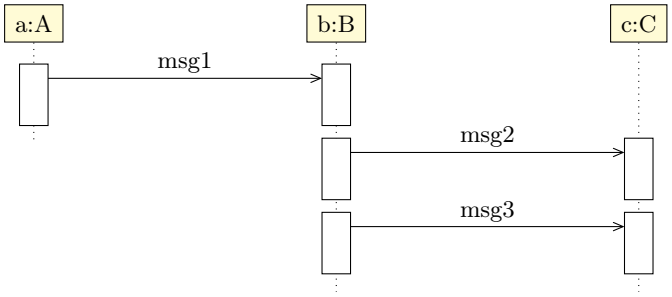
Question 8

Which of the following class diagrams is wrong?

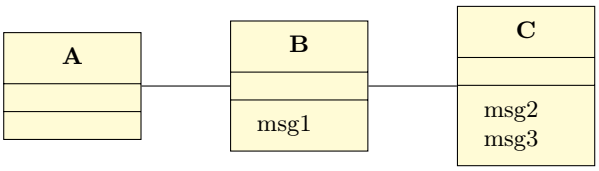


Question 9

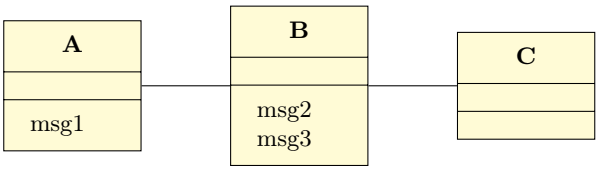
Consider the following sequence diagram.



Which of the following statements are **TRUE**?



a) Answer The class diagram above is consistent with the sequence diagram.



b) The class diagram above is consistent with the sequence diagram.

c) Message *msg2* is received before *msg3* has been sent.

d) Answer Message *msg2* is sent after *msg1* has been received.

Question 10

In UML deployment diagrams (indicate all correct answers):

a) artifacts are represented as pyramids.

b) Answer nodes are represented as cubes.

c) use cases are represented as cylinders.

d) information assets are represented as ellipses.

Question 11

On February 21, 2014 user *ettozyame* has asked the following question on the popular Q&A site StackOverflow: "I want to know in detail about the difference between *alt* and *opt* fragment in sequence diagram, they seem similar, I can't distinguish

them. Anyone knows about this thing?" Which of the following answers are correct?[1]

a) Answer *alt* is more used for several choices, like a *switch*, while with *opt* code will be executed or not!

b) Answer *alt* is used to described alternative scenarios. Only one of the options will be executed. *opt* is used to describe an optional step.

c) an *opt* fragment cannot be enclosed in other combined fragments such as *strict* or *par*, while *alt* can be enclosed in such fragments.

d) an *alt* fragment cannot be enclosed in other combined fragments such as *strict* or *par*, while *opt* can be enclosed in such fragments.

Question 12

Indicate all correct statements in the component diagram in Figure 4 on page 6.

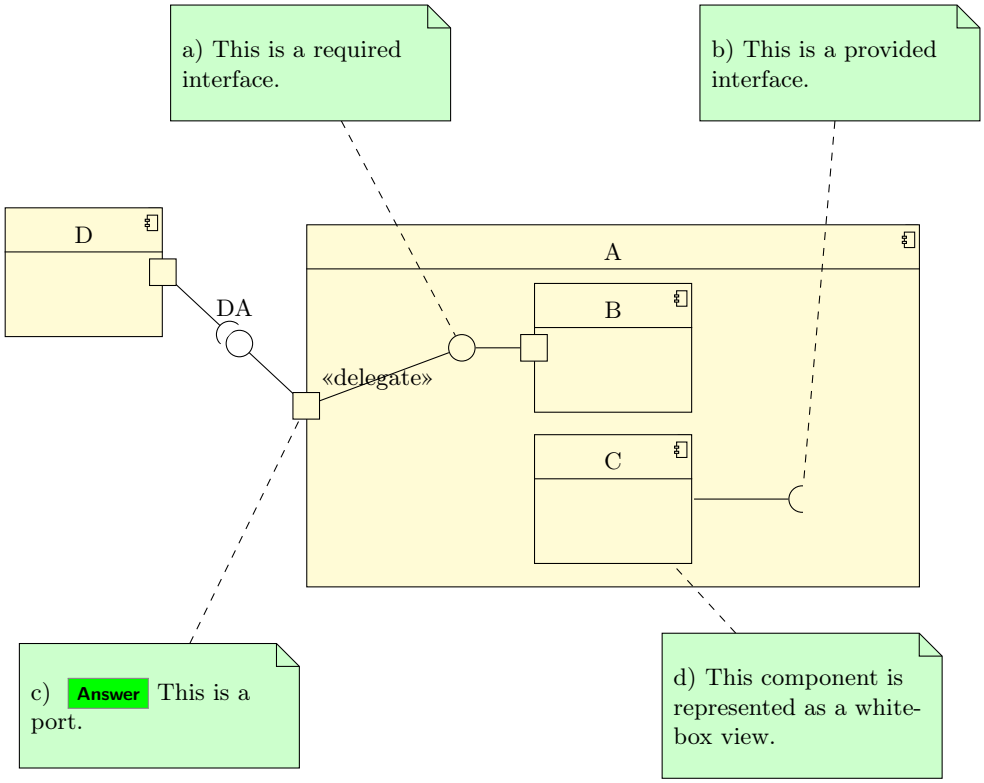


Figure 4: Component diagram.

[1] Slightly modified from the answers given at StackOverflow.

Question 13

We are modeling a taxi fleet company in Event-B. We consider the following variables: *drivers* representing the drivers working for the company, *taxis* for the taxis owned by the company, *assigned* representing the assignment of drivers to taxis at a given moment, *occupied* indicating whether the taxi is occupied or not, *i.e.*, is currently transporting a passenger or not.

The following Event B specification fragment[2] describes the relation between the variables:

$$\mathit{assigned} \in (\mathit{drivers} \;\text{⤔}\; \mathit{taxis}), \quad \mathit{occupied} \subseteq \mathrm{ran}(\mathit{assigned})$$

Which informal description corresponds to this fragment?

a) A driver can be assigned only to one taxi at the given moment, taxis can be driven by one or more drivers. Only taxis assigned to a driver can be occupied.

b) A driver can be assigned only to one taxi at the given moment, all taxis should be assigned to a driver. Only occupied taxis can be assigned.

c) One or more drivers can be assigned to the same taxi at the given moment. Only occupied taxis can be assigned.

d) **Answer** A driver can be assigned only to one taxi at the given moment, every taxi can be driven only by one driver. Only taxis assigned to a driver can be occupied.

Question 14

Rozanski and Woods identified the information viewpoint that “describes the way that the architecture stores, manipulates, manages, and distributes information”. Typical concerns of different classes of stakeholders that can be framed by this viewpoint pertain, therefore, to:

a) Acquirers: translation of the data architect’s models into database systems and information interfaces.

b) **Answer** Assessors: legal regulations and data quality.

c) **Answer** System administrators: management and support of real-wold components, *e.g.*, database systems.

d) Testers: types, specification and quantity of hardware required.

[2] For the sake of readability the font of the Event-B arrows has been increased.

Question 15

Kruchten's 4+1 deployment view

a) commonly discusses organization of the software system in files and folders;

b) can be represented as one of the UML interaction diagrams;

c) Answer should be consistent with scenarios;

d) Answer consists of one or more "model kinds" in the ISO/IEC/IEEE 42010-parlance.

Question 16

The following are examples of Domain-Specific Software Architectures:

a) Answer AUTOSAR,

b) Koala,

c) Answer MURA,

d) Wright.

Question 17

Domain-Specific Software Architecture consists of

a) Answer reference architecture,

b) architecture description language,

c) Answer component library,

d) Answer application configuration method.

Question 18

Consider the variability model [cf. Giraldo, Rincón-Perez, Mazo[3]] for mobile phones in Figure 7 on page 19. Which of the following statements are **TRUE**?

a) Answer No combination of features allowed by the variability model contains MSN.

b) Any combination of features allowed by the variability model contains MSN.

c) Combination of features {*Mobile Phone*, *Utility Function*, *Calls*, *Data*, *Games*, *Settings*, *Java support*, *OS*} is allowed by the variability model.

d) Optional Combination of features {*Mobile Phone*, *Utility Function*, *Calls*, *Voice*, *Games*, *Settings*, *Java support*, *OS*} is minimal, *i.e.*, no strict subset of this features is allowed by the variability model.

[3] `http://goo.gl/biPMcf`

Question 19

The diagnostic toolkit (DKIT) hybrid framework by Mylaraswamy (1996) "addressed the use and integration of multiple fault diagnostic techniques to meet the challenges of complex, industrial-scale diagnostic problems". DKIT integrates multiple fault diagnostic modules including a signed directed graph technique, qualitative trend analysis and a statistical classifier based on a probability density function. What architecture pattern/style would you apply to implement DKIT?

a) Model-View-Controller

b) Pipe-and-Filter

c) Answer Blackboard

d) Mobile code

Question 20

The following description is due to Brajendra Singh[4]. "There is a utility company which provides water and waste management services to citizens. It has a CRM system, billing system, messaging system and mapping (GIS) system. When a customer is created in the CRM system, the billing, messaging, and GIS systems need a copy of customer data to perform follow-up tasks and automation. For instance—the billing system will create a billing account for the customer, the messaging system will send a welcome message with customer account details on preferred communication channel and the GIS system will update the customer's address in the database for mapping purposes." What architecture pattern/style would you apply to implement the system described?

a) Sense-Compute-Control

b) Answer Publish-subscribe

c) Batch-sequential

d) Client-server

[4] http://goo.gl/2hGlCi

Part II (60 points)

This part consists of four modeling questions, 20 points each. You should answer at least three of these questions. Please read all the exercises first, choose the ones you are most confident in to work on, and work on those exercises.

Should you decide to answer four questions, we will consider the three best solutions. Please, keep in mind that three correct solutions will give you more points than four wrong ones!

Each question is presented on a separate page.

Question A

(20 points) Consider the following simplified description of a university where professors teach seminars in which students can enroll.

> A professors has a name, address, phone number, email address, and salary. A student has also a name, etc., but no salary (sorry). A student, however, has an average mark (of the final marks of his or her seminars). A seminar has a name and a number. When a student is enrolled in a seminar, the marks for this enrollment are recorded and the current average as well as the final mark (if there is one) can be obtained from the enrollment. From a student one can obtain a list of seminars he or she is enrolled in. Professors teach seminars. Each seminar has at least one and at most three teachers. There are two types of seminar: bachelor and master. From a bachelor seminar students can not withdraw. From a master seminar they can.

Draw a class diagram for this university. Add attributes and methods when necessary. You do not have to include getters and setters for attributes. Visibility modifiers (public, private, etc.) are not required.

Motivate your decisions.

Grading policy for this question

- Missing motivation ⇒ -5 points per important issue.
- Missing features (methods, attributes) ⇒ -3 points per issue.
- Small syntactic error ⇒ -2 points per error.
- Semantic error (inconsistency, missed feature, etc.) ⇒ -5 points per issue.

Answer

Figure 5

The domain categories *Student*, *Professor*, and *Seminar* are obviou candidates for classes. Professor and Student both have personal data such as name and address. This data could be factored out and placed in a superclass with a name like *Person* or *Academic*, but is not really necessary. Observing this will earn you a bonus point, whether the superclass is added or not.

The text mentions how many professors there can be teaching a seminar, so multiplicities are apparently important. This multiplicity is required and other multiplicities should be given in the diagram as well. Although the number of seminars a professor can teach is humanly limited, a clear upperbound can not be given, so $*$ seems appropriate here. Similarly, there may be practical or even official upperbounds to the number of seminars a student can take and the number of students that can be enrolled in a seminar, but since none was given in the text, $*$ is the right choice here.

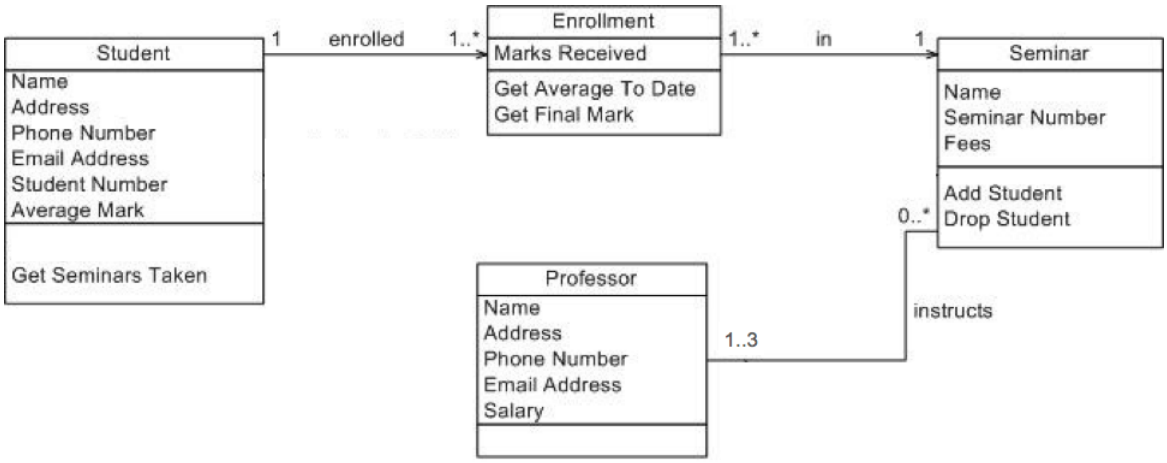


Figure 5: Class diagram for the university example.

The *Enrollment* class can also be expresses as an association class for the association between Student and Seminar. It is not required, but it is a straightforward way to express that each student of for each seminar has a list of grades. Alternative approaches that I have seen often have only one grade per student per seminar.

The text is not very explicit about most of the services (methods) that should be provided, so we are lenient in this respect.

Question B

Consider the following required functionality of a print-on-demand service:

The print-on-demand service provides customers the possibility to print posters, flyers, or books on demand. The customer should be able to select a type of product (poster, flyer, or book), a desired quantity, and a paper type. In case a book has to be printed, additionally the customer can choose between hard cover and soft cover. Finally, the customer needs to provide a PDF file containing the desired content.

In order for the customer to be able to place an order, he or she must have an account. The customer can create an account by choosing a username/password combination. Furthermore, his or her address and credit card number can be linked to the account, which is required information when placing an order.

Once a customer has provided the information for an order, the system checks if all required information is there, either given in the order (type of product, quantity, etc.), or in the account (address and payment information). If any information is lacking, the system will inform the customer that it needs to be added before the order can be placed. Once all information is in place, the order is placed, and the credit card information is sent to the bank for approval. If the bank approves the card, the order is finalized.

A printing agent is in charge of actually performing the printing. He or she inspects the provided PDF files of finalized orders. If a file does not meet the quality requirements, the customer will be informed about this, and the order is temporarily put on hold until the customer has provided a new PDF file.

Finally, the administrator monitors if at all times, sufficient paper and ink stock is present. Whenever the amount of paper or ink is running low, an order must be placed at the appropriate supplier (either the paper or ink supplier).

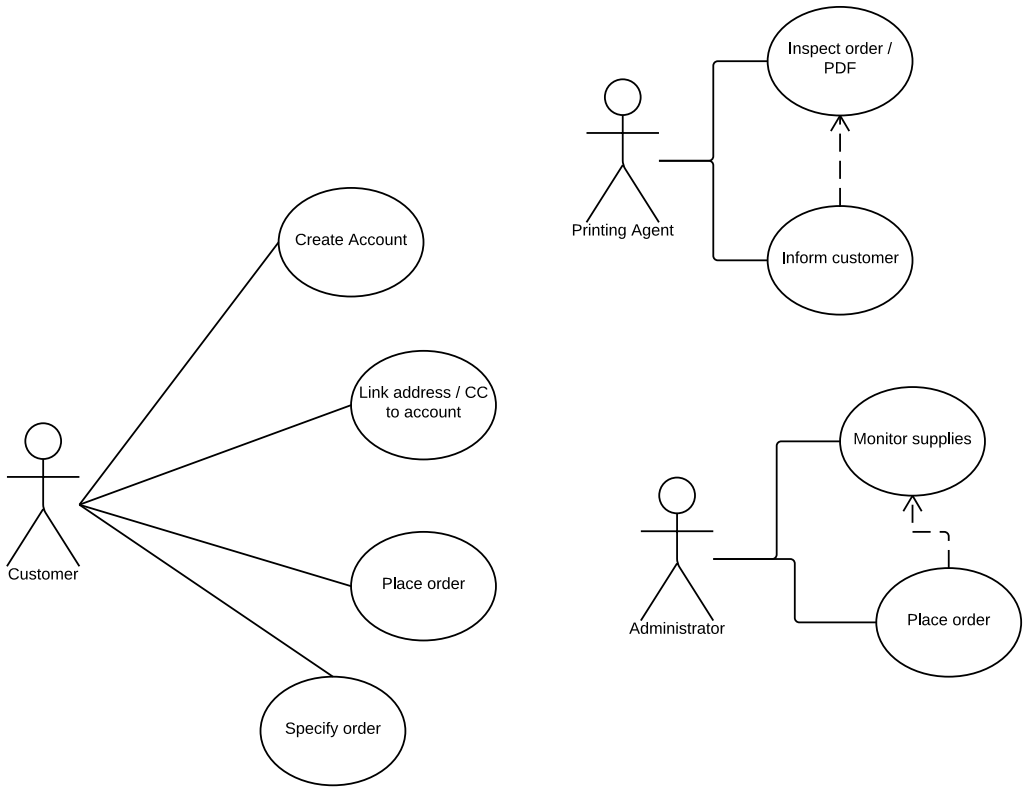
a) (7 points) Create a UML Use Case Diagram for the print-on-demand service. In addition, give a detailed scenario (pre-condition, trigger, guarantee, main scenario, ...) of the use case for "place an order".

b) (7 points) Based on your use case description of "place an order", create a UML Sequence Diagram.

c) (6 points) Create a UML Activity Diagram for processing an order. Once an order has been finalized, four parties are involved in processing it: Online Sales, Accounting, Shipping, and Printing. Online Sales sends the order to Printing, where the associated PDF file is inspected. If the file is not suitable, a new file is requested from the customer. Once the file is suitable, Accounting is informed to charge the credit card. While this is done, Printing carries out the actual printing, and sends the result to Shipping. When Shipping has both received the printed products and confirmation from Accounting that payment was successful, the products are shipped to the customer.

(Some possible) answers

a.



Place an Order

Pre: User has account
Trigger: User wants to place an order
Guarantee: Order is placed

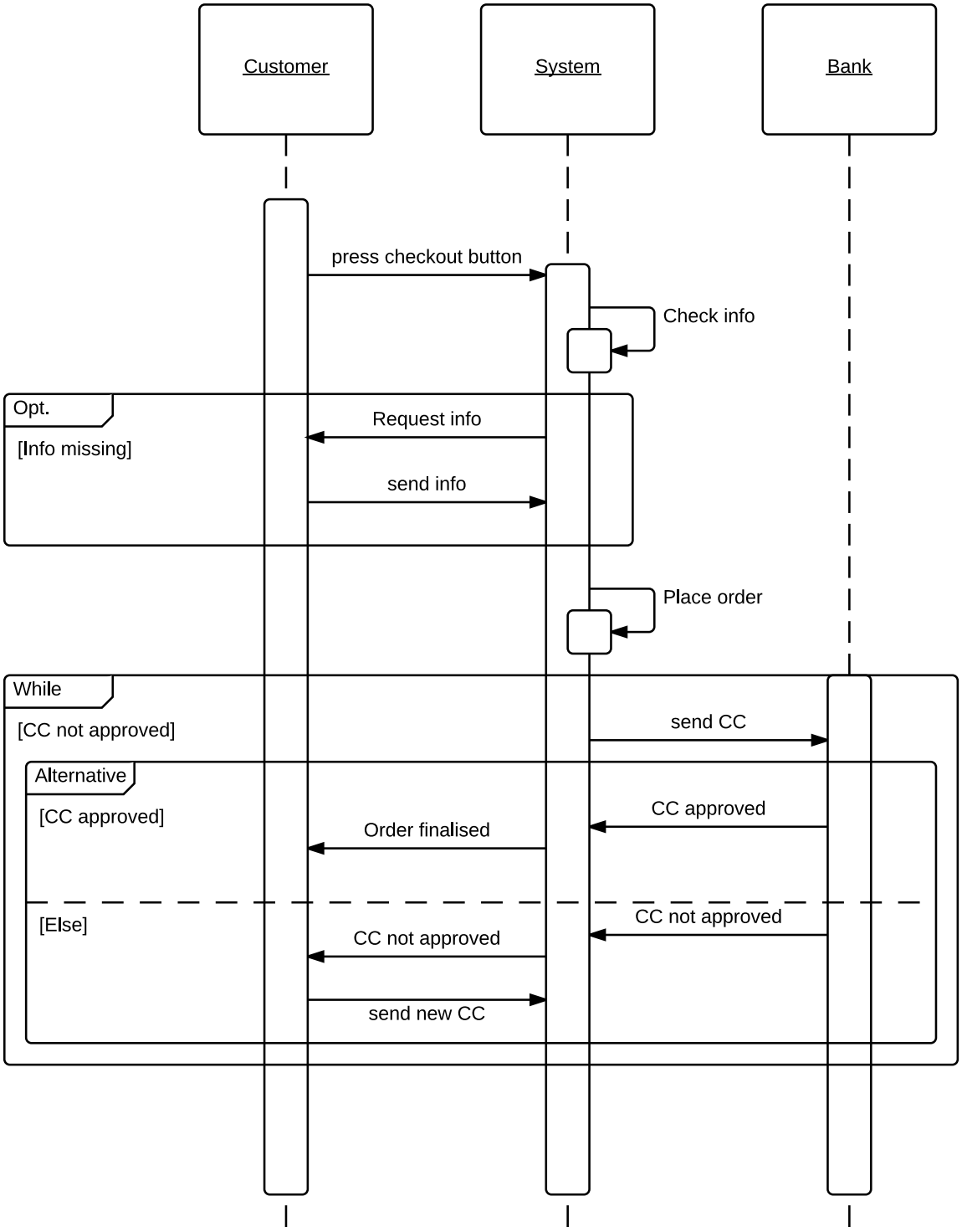
Main:

a. Send order
b. Information is checked
c. Order is placed
d. CC is sent to bank
e. Order is finalised

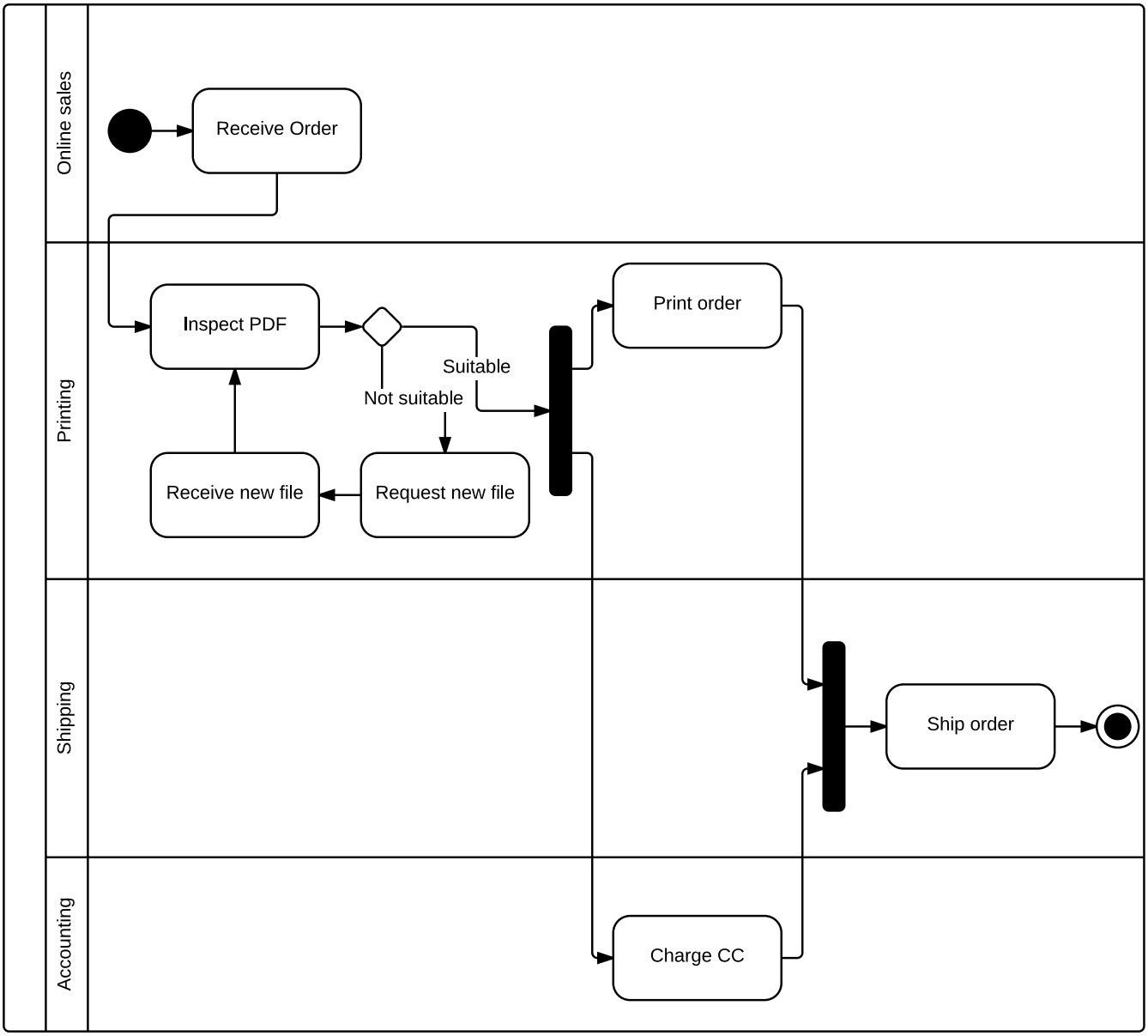
Alternatives:

c-1. If information is missing, request it from the user. Retry step b.
e-1. If CC is not approved, request new CC from the user. Retry step d.

b.


Customer
System
Bank
press checkout button
Check info
Opt.
[Info missing]
Request info
send info
Place order
While
[CC not approved]
send CC
Alternative
[CC approved]
CC approved
Order finalised
[Else]
CC not approved
CC not approved
send new CC

c.


Online sales
Receive Order
Printing
Inspect PDF
Suitable
Not suitable
Print order
Receive new file
Request new file
Shipping
Ship order
Accounting
Charge CC

Question C

Company "Valco Unlimited" intends to sell a broad range of software systems handling payment transactions in supermarkets. 'Valco" intends to support different cash desk systems serving varying purposes and shop scenarios. To address this challenge'the company considers to design an engineering product line. '

a) (5 points) Select two *stakeholders categories* from the classification of Rozanski and Woods, and describe advantages and disadvantages of the engineering product line approach from *their* points of view.

b) (15 points) Design a variability model for the systems "Valco" intends to sell based on the following description:[5]

A generic cash desk system allows a cashier to perform payment transactions. The cashier enters products into the system using a number of input controllers. Handing product input to the system can be done using a bar code scanner or a keyboard. In case a scanner is not present, a keyboard has to be connected to the system for entering the respective product IDs manually. On the other hand, a scanner is required should no keyboard be present. If both input options are available they may be used interchangeably. Based on the product IDs entered by the cashier, the cash desk polls a central store server to obtain the product's data set, calculates the overall value of customer's desired products and displays the information to the cashier.

The system allows paying the goods by different payment options. These payment options are cash, credit card, a prepaid card or an electronic cash system. The keyboard used for product input is also required for cash payment, as the cashier needs to enter the value of the cash money she received from the customer. This allows the system to calculate and display the difference. Allowing cash payment additionally requires a cash box being connected to the cash desk.

In case the system is used for selling goods, which don't have their price calculated on a per piece base, as in a supermarket selling fruit and vegetable for example, there are two options: One possibility is to let each cash desk be provided with an additional scale to measure the weight of a product. In this case the product data in the store inventory indicates, if a specific product needs to be weighed. Alternatively, weighing the goods may happen at a specialized weighing machine in the respective part of the shop. This machine allows specifying the product type, weighing the goods, producing a bar code label, which is to be attached to the goods, and sending the bar code information with the corresponding price to the inventory on the store server.

[5]Example is due to

- Sebastian Herold, Holger Klus, Yannick Welsch, Constanze Deiters, Andreas Rausch, Ralf Reussner, Klaus Krogmann, Heiko Koziolek, Raffaela Mirandola, Benjamin Hummel, Michael Meisinger, and Christian Pfaller, "CoCoME—The Common Component Modeling Example, Common Component Modeling Example" (Andreas Rausch et al., eds.), Springer Verlag, 2008, pp. 16–53.
- Alexander Worret, "Automated Product Derivation for the CoCoME Software Product Line: From Feature Models to CoBoxes," Master's thesis, University of Kaiserslautern, March 2009.

For franchise supermarkets the system might need to support an enterprise server to create a connection between several stores and to allow performing a number of enterprise-oriented tasks, like product transfer and synchronization between stores. If a store is not part of a bigger franchise, the enterprise server can be omitted. Additionally, small businesses don't require a high number of cash desks. For these stores having just one cash desk is often sufficient. In that case, this one cash desk may also take over the role of the store server and the enterprise server, if it is present. If the enterprise server is present, it can be a Windows, Linux or Mac machine.

Another optional part of the system is self service for cash desks. At self service desks customers process their payment on their own, without the assistance of a cashier, which speeds up their payment process and saves costs for the store. To control this process, the products entered by a customer are weighed and compared to the weight of the product that is archived on the inventory. In case weighed goods are supported by the system, self service desks may be equipped with a camera system. This camera analyzes which type of product is potentially on the scales and displays the possible choices to the customer.

As a final requirement, some customers require the express checkout mode. When the express checkout mode is switched on the cash desk allows only customers with a few goods and also only cash payment to speed up the clearing.

Grading policy for this question:

- The basic structure of the variability model is syntactically incorrect ⇒ -15.
- Missing features ⇒ -2 (per feature).
- Wrong optionality indication ⇒ -1 (per error).
- And/or/xor confusion ⇒ -1 (per error).
- Missing cross-feature relations ⇒ -2 (per feature).

Answer

a) The question asked for advantages and disadvantages of the engineering product line approach, hence the stakeholders should be aware that the engineering product line approach has been chosen to implement the system. From the perspective of the *developers* the approach chosen allows them to reduce the development time of new system variants. Implementing new features, however, might require more effort if those features do not extend the expected variation points. Similarly, *testers* can find the testing effort spent on testing individual features decreasing since the tests of those features can be reused; however, ensuring that no unintended feature interaction occurs might be challenging as testing all possible feature combinations might become prohibitive.

b) The variability model is shown in Figure 6. Please note that different variability models can be correct. We did not model dependency of the "separate weighing"

on the "scanner" since bar code labels produced by the weighing machine might be eventually entered manually.

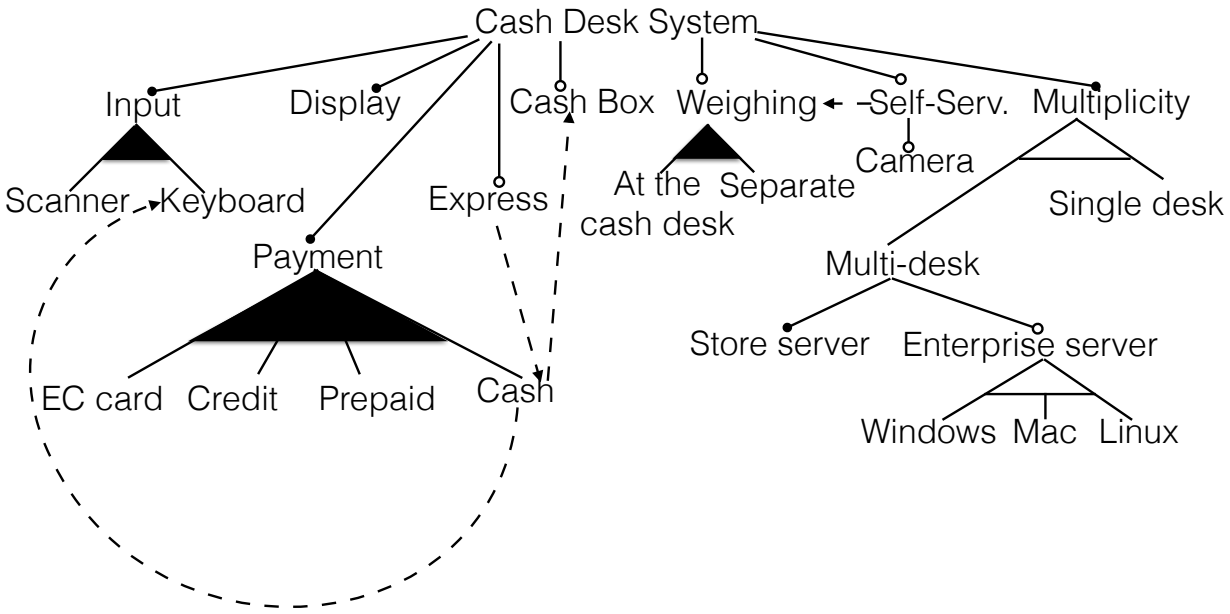


Figure 6: Feature model for the cash desk system of "Valco Unlimited".

Question D

Suppose we have designed a new programming language called L. Our task is to develop a compiler that receives a program in L and generates machine instructions for a specific processor. The compilation process consists of different steps. Flexible error handling in these steps is of utmost importance; we aim to have error handling strategies that are specific to different stages of the compilation process. To this end, we have decided to split the compilation process into three stages: *front end*, *middle end*, and *back end*.

The *front end* reads the source file, parses it, and converts it into an abstract syntax tree. The *front end* performs semantic analysis on the constructed abstract tree, augments it with additional information and passes the augmented tree to the *middle end*.

The purpose of the *middle end* is to generate code in an intermediate language and to perform optimization on the generated code. The optimization phase consists of different steps. The architecture should offer design flexibility in the optimization phase such that we can add, remove, or reorder the optimization steps in different releases of the compiler.

The *back end* receives the optimized code in the intermediate language and translates it to machine instructions for a specific processor.

a) (4 points) Identify the flexibility concerns that should be addressed by the architecture.

b) (3 points) Which architectural patterns/styles would be useful in this situation? Please exclude language-influenced styles (object-oriented and main program with subroutines) from consideration.

c) (6 points) Identify the components and connectors of the chosen styles.

d) (7 points) Explain how the chosen patterns/styles address the flexibility issues in the description.

Answer

a) The main problems that should be addressed by the architecture:

- splitting the compilation process into three parts to facilitate error handling;
- flexibility of the middle end.

b) The general structure of the system will be based on layers. The middle end will be structured with Pipes and Filters.

c) The front end, the middle end, and the back end will be the layers of the compiler. The input program is received by the front end. The front end transforms the program into a new shape (i.e., an abstract syntax tree) and passes it to the middle end. This process continues until the (transformed) program arrives at the lowest level and machine code is generated. We apply Pipes and Filters in

the middle end layer. The optimization steps will be the filters and adjacent steps will be connected with pipes. Intermediate programs will flow between the steps through the pipes.

d) The Layers pattern helps us to decompose the compiler into parts with well-defined functionalities. Applying Pipes and Filters in the middle end layer allows us to change or reorder the steps without significant efforts.

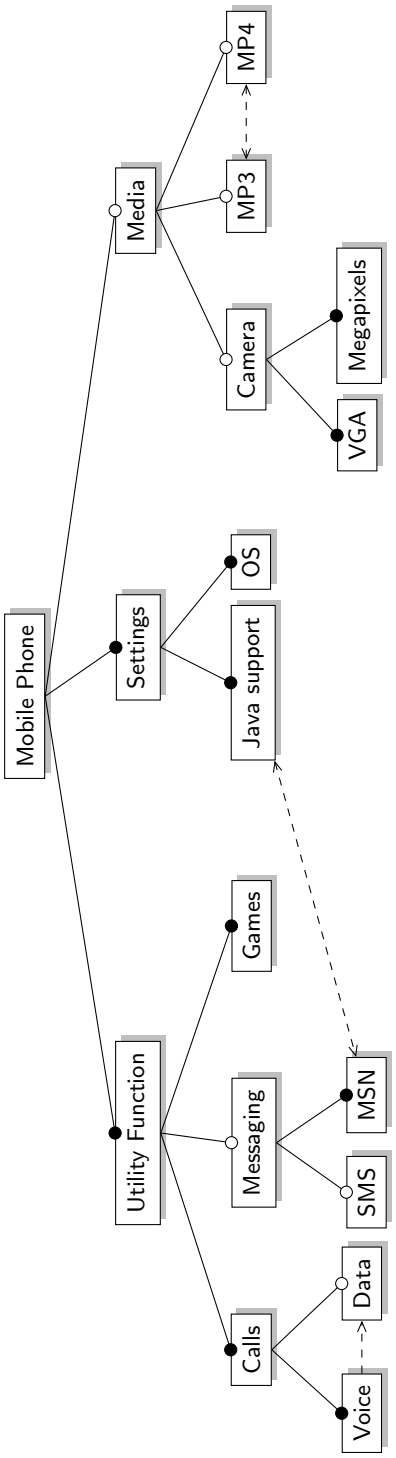


Figure 7: Variability model.

A Concise Summary of the Event B mathematical toolkit [1]

Each construct will be given in its presentation form, as displayed in the Rodin toolkit, followed by the ASCII form that is used for input to Rodin.

In the following: P, Q and R denote *predicates*;
x and y denote single variables;
z denotes a single or comma-separated list of variables;
p denotes a pattern of variables, possibly including $\mapsto$ and parentheses;
S and T denote set expressions;
U denotes a set of sets;
m and n denote integer expressions;
f and g denote functions;
r denotes a relation;
E and F denote expressions;
E, F is a recursive pattern, *ie* it matches e_1, e_2 and also $e_1, e_2, e_3 \ldots$; similarly for x, y;

Freeness: The meta-predicate $\neg free(z, E)$ means that none of the variables in z occur free in E. This meta-predicate is defined recursively on the structure of E, but that will not be done here explicitly. The base cases are: $\neg free(z, \forall z \cdot P \Rightarrow Q)$, $\neg free(z, \exists z \cdot P \wedge Q)$, $\neg free(z, \{z \cdot P \mid F\})$, $\neg free(z, \lambda z \cdot P | E)$, and $free(z, z)$.

In the following the statement that P *must constrain* z means that the type of z must be at least inferrable from P.

In the following, parentheses are used to show syntactic structure; they may of course be omitted when there is no confusion.

Note: Event-B has a formal syntax and this summary does not attempt to describe that syntax. What it attempts to do is to *explain* Event-B *constructs*. Some words like *expression* collide with the formal syntax. Where a syntactical entity is intended the word will appear in *italics*, *e.g. expression*, *predicate*.

1 Predicates

1. False $\bot$ `false`
2. True $\top$ `true`
3. Conjunction: $P \wedge Q$ `P & Q`
   Left associative.
4. Disjunction: $P \vee Q$ `P or Q`
   Left associative.
5. Implication: $P \Rightarrow Q$ `P => Q`
   Non-associative: this means that $P \Rightarrow Q \Rightarrow R$ must be parenthesised or an error will be diagnosed.
6. Equivalence: $P \Leftrightarrow Q$ `P <=> Q`.
   $P \iff Q = P \Rightarrow Q \wedge Q \Rightarrow P$
   Non-associative: this means that $P \Leftrightarrow Q \Leftrightarrow R$ must be parenthesised or an error will be diagnosed.
7. Negation: $\neg P$ `not P`
8. Universal quantification:
   $\forall z \cdot P \Rightarrow Q$ `!z.P => Q`
   Strictly, $\forall z \cdot P$, but usually an implication.
   For all values of z, satisfying P, Q is satisfied.
   The types of z must be inferrable from the *predicate* P.
9. Existential quantification:
   $\exists z \cdot P \wedge Q$ `#z.P & Q`
   Strictly, $\exists z \cdot P$, but usually a conjunction.
   There exist values of z, satisfying P, that satisfy Q.
   The type of z must be inferrable from the *predicate* P.
10. Equality: $E = F$ `E = F`
11. Inequality: $E \neq F$ `E /= F`

2 Sets

1. Singleton set: $\{E\}$ `{E}`
2. Set enumeration: $\{E, F\}$ `{E, F}`
   See note on the pattern E, F at top of summary.
3. Empty set: $\varnothing$ `{}`
4. Set comprehension: $\{ z \cdot P \mid F \}$ `{ z . P | F }`
   General form: the set of all values of F for all values of z that satisfy the *predicate* P. P must *constrain* the variables in z.
5. Set comprehension: $\{ F \mid P \}$ `{ F | P }`
   Special form: the set of all values of F that satisfy the *predicate* P. In this case the set of bound variables z are all the free variables in F.
   $\{ F \mid P \} = \{ z \cdot P \mid F \}$, where z is all the variables in F.
6. Set comprehension: $\{ x \mid P \}$ `{ x | P }`
   A special case of item 5: the set of all values of x that satisfy the *predicate* P.
   $\{ x \mid P \} = \{ x \cdot P \mid x \}$
7. Union: $S \cup T$ `S \/ T`
8. Intersection: $S \cap T$ `S /\ T`

[1] Version October 7, 2010©1996-2010 Ken Robinson

9. Difference: $S \setminus T$ `S \ T`
   $S \backslash T = \{x \mid x \in S \wedge x \notin T\}$

10. Ordered pair: $E \mapsto F$ `E |-> F`.
    $E \mapsto F \neq (E, F)$
    Left associative.
    In all places where an ordered pair is required, $E \mapsto F$ must be used. E, F will not be accepted as an ordered pair, it is always a list. $\{x, y \cdot P \mid x \mapsto y\}$ illustrates the different usage.

11. Cartesian product: $S \times T$ `S ** T`
    $S \times T = \{x \mapsto y \mid x \in S \wedge y \in T\}$
    Left-associative.

12. Powerset: $\mathbb{P}(S)$ `POW(S)`
    $\mathbb{P}(S) = \{s \mid s \subseteq S\}$

13. Non-empty subsets: $\mathbb{P}_1(S)$ `POW1(S)`
    $\mathbb{P}_1(S) = \mathbb{P}(S) \backslash \{\varnothing\}$

14. Cardinality: card(S) `card(S)`
    Defined only for *finite*(S).

15. Generalized union: union(U) `union(U)`
    The union of all the elements of U.
    $\forall U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
    $\text{union}(U) = \{x \mid x \in S \wedge \exists s \cdot s \in U \wedge x \in s\}$
    where $\neg$*free*(x, s, U)

16. Generalized intersection: inter(U) `inter(U)`
    The intersection of all the elements of U.
    $U \neq \varnothing$,
    $\forall U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
    $\text{inter}(U) = \{x \mid x \in S \wedge \forall s \cdot s \in U \Rightarrow x \in s\}$
    where $\neg$*free*(x, s, U)

17. Quantified union:
    $\cup z \cdot P \mid S$ `UNION z.P | S`
    P must *constrain* the variables in z.
    $\forall z \cdot P \Rightarrow S \subseteq T \Rightarrow$
    $\cup(z \cdot P \mid E) = \{x \mid x \in T \wedge \exists z \cdot P \wedge x \in S\}$
    where $\neg$*free*(x, z, T), $\neg$*free*(x, P), $\neg$*free*(x, S), $\neg$*free*(x, z)

18. Quantified intersection:
    $\cap z \cdot P \mid S$ `INTER z.P | S`
    P must *constrain* the variables in z,
    $\{z \mid P\} \neq \varnothing$,
    $(\forall z \cdot (P \Rightarrow S \subseteq T)) \Rightarrow$
    $\cap z \cdot P \mid S = \{x \mid x \in T \wedge (\forall z \cdot P \Rightarrow x \in S)\}$
    where $\neg$*free*(x, z), $\neg$*free*(x, T), $\neg$*free*(x, P), $\neg$*free*(x, S).

2.1 Set predicates

1. Set membership: $E \in S$ `E : S`
2. Set non-membership: $E \notin S$ `E /: S`
3. Subset: $S \subseteq T$ `S <: T`
4. Not a subset: $S \not\subseteq T$ `S /<: T`
5. Proper subset: $S \subset T$ `S <<: T`
6. Not a proper subset: $s \not\subset t$ `S /<<: T`
7. Finite set: *finite*(S) `finite(S)`
   finite(S) $\Leftrightarrow$ S *is finite.*
8. Partition: *partition*(S, x, y) `partition(S,x,y)`
   x and y partition the set S, *ie* $S = x \cup y \wedge x \cap y = \varnothing$
   Specialised use for enumerated sets: *partition*$(S, \{A\}, \{B\}, \{C\})$.
   $S = \{A, B, C\} \wedge A \neq B \wedge B \neq C \wedge C \neq A$

3 BOOL **and** bool

BOOL is the enumerated set: {FALSE, TRUE}
and bool is defined on a predicate P as follows:

1. P is provable: bool(P) = TRUE
2. $\neg P$ is provable: bool(P) = FALSE

4 Numbers

The following is based on the set of integers, the set of natural numbers (non-negative integers), and the set of positive (non-zero) natural numbers.

1. The set of integer numbers: $\mathbb{Z}$ `INT`
2. The set of natural numbers: $\mathbb{N}$ `NAT`
3. The set of positive natural numbers: $\mathbb{N}_1$ `NAT1`
   $\mathbb{N}_1 = \mathbb{N} \backslash \{0\}$
4. Minimum: min(S) `min(S)`
   $S \subset \mathbb{Z}$ and *finite*(S) or S must have a lower bound.
5. Maximum: max(S) `max(S)`
   $S \subset \mathbb{Z}$ and *finite*(S) or S must have an upper bound.
6. Sum: $m + n$ `m + n`
7. Difference: $m - n$ `m - n`
   $n \leq m$
8. Product: $m \times n$ `m * n`
9. Quotient: m/n `m / n`
   $n \neq 0$
10. Remainder: $m \, mod \, n$ `m mod n`
    $n \neq 0$
11. Interval: $m \mathinner{..} n$ `m .. n`
    $m \mathinner{..} n = \{\, i \mid m \leq i \wedge i \leq n \,\}$

4.1 Number predicates

1. Greater: $m > n$ `m > n`
2. Less: $m < n$ `m < n`
3. Greater or equal: $m \geq n$ `m >= n`
4. Less or equal: $m \leq n$ `m <= n`

5 Relations

A relation is a set of ordered pairs; a many to many mapping.

1. Relations: $S \leftrightarrow T$ `S <-> T`
   $S \leftrightarrow T = \mathbb{P}(S \times T)$
   Associativity: relations are *right associative*:
   $r \in X \leftrightarrow Y \leftrightarrow Z = r \in X \leftrightarrow (Y \leftrightarrow Z)$.

2. Domain: $\mathrm{dom}(r)$ `dom(r)`
   $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$
   $\mathrm{dom}(r) = \{x \cdot (\exists y \cdot x \mapsto y \in r)\}$

3. Range: $\mathrm{ran}(r)$ `ran(r)`
   $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$
   $\mathrm{ran}(r) = \{y \cdot (\exists x \cdot x \mapsto y \in r)\}$

4. Total relation:$S \nleftrightarrow T$ `S <<-> T`
   if $r \in S \nleftrightarrow T$ then $\mathrm{dom}(r) = S$

5. Surjective relation:$S \leftrightarrow\!\!\!\!\rightarrow T$ `S <->> T`
   if $r \in S \leftrightarrow\!\!\!\!\rightarrow T$ then $\mathrm{ran}(r) = T$

6. Total surjective relation:$S \nleftrightarrow\!\!\!\!\rightarrow T$ `S <<->> T`
   if $r \in S \nleftrightarrow T$ then $\mathrm{dom}(r) = S$ and $\mathrm{ran}(r) = T$

7. Forward composition: $p\,;q$ `p ; q`
   $\forall p, q \cdot p \in S \leftrightarrow T \wedge q \in T \leftrightarrow U \Rightarrow$
   $p\,;q = \{x \mapsto y \mid (\exists z \cdot x \mapsto z \in p \wedge z \mapsto y \in q)\}$

8. Backward composition: $p \circ q$ `p circ q`
   $p \circ q = q\,;p$

9. Identity: id `id`
   $S \lhd \mathrm{id} = \{x \mapsto x \mid x \in S\}$.
   id is generic and the set S is inferred from the context.

10. Domain restriction: $S \lhd r$ `S <| r`
    $S \lhd r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \in S\}$.

11. Domain subtraction: $S \lhd\!\!\!- r$ `S <<| r`
    $S \lhd\!\!\!- r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin S\}$.

12. Range restriction: $r \rhd T$ `r |> T`
    $r \rhd T = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in T\}$.

13. Range subtraction: $r \rhd\!\!\!- T$ `r |>> T`
    $r \rhd\!\!\!- T = \{x \mapsto y \mid y \in r \wedge y \notin T\}$.

14. Inverse: r^{-1} `r~`
    $r^{-1} = \{y \mapsto x \mid x \mapsto y \in r\}$.

15. Relational image: $r[S]$ `r[S]`
    $r[S] = \{y \mid \exists x \cdot x \in S \wedge x \mapsto y \in r\}$.

16. Overriding: $r_1 \lhd\!\!+ r_2$ `r1 <+ r2`
    $r_1 \lhd\!\!+ r_2 = r_2 \cup (\mathrm{dom}(r_2) \lhd r_1)$.

17. Direct product: $p \otimes q$ `p >< q`
    $p \otimes q = \{x \mapsto (y \mapsto z) \mid x \mapsto y \in p \wedge x \mapsto z \in q)\}$.

18. Parallel product: $p \parallel q$ `p || q`
    $p \parallel q = \{x, y, m, n \cdot x \mapsto m \in p \wedge y \mapsto n \in q \mid (x \mapsto y) \mapsto (m \mapsto n)\}$.

19. Projection: prj_1 `prj1`
    prj_1 is generic.
    $(S \times T) \lhd \mathrm{prj}_1 = \{(x \mapsto y) \mapsto x \mid x \mapsto y \in S \times T\}$.

20. Projection: prj_2 `prj2`
    prj_2 is generic.
    $(S \times T) \lhd \mathrm{prj}_2 = \{(x \mapsto y) \mapsto y \mid x \mapsto y \in S \times T\}$.

5.1 Iteration and Closure

Iteration and closure are important functions on relations that are not currently part of the kernel Event-B language. They can be defined in a Context, but not polymorphically.

Note: iteration and irreflexive closure will be implemented in a proposed extension of the mathematical language. The operators will be non-associative.

1. Iteration: r^n □
   $r \in S \leftrightarrow S \Rightarrow r^0 = S \lhd \mathrm{id} \wedge r^{n+1} = r\,;r^n$.
   Note: to avoid inconsistency S should be the finite *base* set for r, ie the smallest set for which all $r \in S \leftrightarrow S$.
   Could be defined as a function $iterate(r \mapsto n)$.

2. Reflexive Closure: r^* □
   $r^* = \cup n \cdot (n \in \mathbb{N} \mid r^n)$.
   Could be defined as a function $rclosure(r)$.
   Note: $r^0 \subseteq r^*$.

3. Irreflexive Closure: r^+ □
   $r^+ = \cup n \cdot (n \in \mathbb{N}_1 \mid r^n)$.
   Could be defined as a function $iclosure(r)$.
   Note: $r^0 \not\subseteq r^+$ by default, but may be present depending on r.

5.2 Functions

A function is a relation with the restriction that each element of the domain is related to a unique element in the range; a many to one mapping.

1. Partial functions: $S \nrightarrow T$ `S +-> T`
   $S \nrightarrow T = \{r \cdot r \in S \leftrightarrow T \wedge r^{-1}\,; r \subseteq T \lhd \mathrm{id}\}$.

2. Total functions: $S \rightarrow T$ `S --> T`
   $S \rightarrow T = \{f \cdot f \in S \nrightarrow T \wedge \mathrm{dom}(f) = S\}$.

3. Partial injections: $S \rightarrowtail T$ `S >+> T`
   $S \rightarrowtail T = \{f \cdot f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S\}$.
   One-to-one relations.

4. Total injections: $S \rightarrowtail T$ `S >-> T`
   $S \rightarrowtail T = S \rightarrowtail T \cap S \rightarrow T$.

5. Partial surjections: $S \twoheadrightarrow T$ `S +->> T`
   $S \twoheadrightarrow T = \{f \cdot f \in S \nrightarrow T \wedge \mathrm{ran}(f) = T\}$.
   Onto relations.

6. Total surjections: $S \twoheadrightarrow T$ `S -->> T`
   $S \twoheadrightarrow T = S \twoheadrightarrow T \cap S \rightarrow T$.

7. Bijections: $S \rightarrowtail\!\!\!\!\twoheadrightarrow T$ `S >->> T`
   $S \rightarrowtail\!\!\!\!\twoheadrightarrow T = S \rightarrowtail T \cap S \twoheadrightarrow T$.
   One-to-one and onto relations.

8. Lambda abstraction:
   $(\lambda p \cdot P \mid E)$ `(%p.P|E)`
   P must *constrain* the variables in p.
   $(\lambda p \cdot P \mid E) = \{z \cdot P \mid p \mapsto E\}$, where z is a list of variables that appear in the pattern p.

9. Function application: $f(E)$ `f(E)`
   $E \mapsto y \in f \Rightarrow E \in \mathrm{dom}(f) \wedge f \in X \nrightarrow Y$, where $type(f) = \mathbb{P}(X \times Y)$.
   Note: in Event-B, relations and functions only ever have one argument, but that argument may be a pair or tuple, hence $f(E \mapsto F)$ `f(E |-> F)` $f(E, F)$ is never valid.

6 Models

1. Contexts: contain sets and constants used by other contexts or machines.

CONTEXT	Identifier
EXTENDS	Machine_Identifiers
SETS	Identifiers
CONSTANTS	Identifiers
AXIOMS	Predicates
END	

   Note: *theorems* can be presented in the AXIOMS part of a context.

2. Machines: contain events.

MACHINE	Identifier
REFINES	Machine_Identifiers
SEES	Context_Identifiers
VARIABLES	Identifiers
INVARIANT	Predicates
VARIANT	Expression
EVENTS	Events
END	

   Note: *theorems* can be presented in the INVARIANT section of a machine and the WHERE part of an event.

6.1 Events

Event_name	
REFINES	Event_identifiers
ANY	Identifiers
WHERE	Predicates
WITH	Witnesses
THEN	Actions
END	

There is one distinguished event named *INITIALISATION* used to initialise the variables of a machine, thus establishing the invariant.

6.2 Actions

Actions are used to change the state of a machine. There may be multiple actions, but they take effect concurrently, that is, in parallel. The semantics of events are defined in terms of *substitutions*. The substitution $[G]P$ defines a predicate obtained by replacing the values of the variables in P according to the action G. General substitutions are not available in the Event-B language.

Note on concurrency: any single variable can be modified in at most one action, otherwise the effect of the actions would, in general, be inconsistent.

1. *skip*, the null action:
   skip denotes the empty set of actions for an event.

2. Simple assignment action: $z := E$ `x := E`
   $:=$ = *"becomes equal to"*: replace free occurrences of x by E.

3. Choice from set: $x :\in S$ `x :: S`
   $:\in$ = *"becomes in"*: arbitrarily choose a value from the set S.

4. Choice by predicate: $z :| P$ `z :| P`
   $:|$ = *"becomes such that"*: arbitrarily choose values for the variable in z that satisfy the predicate P. Within P, x refers to the value of the variable x before the action and x' refers to the value of the variable after the action.

5. Functional override: $f(x) := E$ `f(x) := E`
   Substitute the value E for the function/relation f at the point x.
   This is a shorthand:
   $f(x) := E \;=\; f := f \Leftarrow \{x \mapsto E\}$.

Acknowledgement: Jean-Raymond Abrial, Laurent Voisin and Ian Hayes have all given valuable feedback and corrections at various stages of the evolution of this summary.