

Session 4

Collaboration (communication) diagrams

A collaboration diagram carries the same information as a sequence diagram but emphasises the links between objects instead of time. Numbering the messages is what makes it readable.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 9 to 11 of the manual: collaboration (communication) diagrams
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✕ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q9	Draw Collaboration Diagram for Student Registration Process in Masters Program of a...	Complete
Q10	Draw Collaboration Diagram for Online Banking System	Complete
Q11	Draw a Collaboration diagram for Employee Management System	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- The manual asks for a problem description of 300 to 500 words and a list of assumptions before every diagram. Write both first; they fix the scope the evaluator marks you against.

- Start from the sequence diagram of the same scenario and renumber messages 1, 1.1, 1.2, 2 in call order.
- Every message needs a link between the two objects; if there is no association in the class diagram, revisit Session 1.

Question 9

Problem Statement

■ Write in lab record

Draw Collaboration Diagram for Student Registration Process in Masters Program of a University.

Solution

■ Write in lab record

Assumptions

The university runs a two-year Masters programme (MCA) in four semesters. A student admitted to the programme registers at the start of every semester for the courses of that semester. Registration is done online through the university's Student Registration System, whose class diagram the manual shows in Figure 1.2 and Figure 1.15. The classes used here are Student (enrolment number, name, programme, current semester), Programme (code, name, list of semesters and their courses, maximum duration), Course (code, title, credits, seat limit, seats taken), Registration (one per student per semester: semester, date, fee, status), Enrolment (one per course inside a registration) and FeeReceipt (receipt number, amount, mode, date).

The scenario traced is the use case Register for Semester. The student logs in, picks the semester and the courses, and submits. The student object asks the programme whether this student is eligible: the previous semester's results are declared, no dues are pending and the maximum duration is not exceeded. If eligible, the student object creates a Registration for the semester. The registration asks the programme for the list of courses offered in that semester and, for each course the student chose, asks the course whether a seat is free. For every course with a seat it creates an Enrolment, which reserves the seat on the course. The registration then computes the fee from the credits of the enrolled courses and reports the fee due to the student.

The student pays the fee online. The registration creates a FeeReceipt with the amount and mode, marks itself Confirmed, adds itself to the student's list of registrations and returns a registration slip listing the enrolled courses and the receipt number. If the student does not pay within the deadline, the registration stays Provisional and the reserved seats are released by a batch job that is not part of this diagram.

Re-registration for failed courses, change of programme, scholarship fee waivers and hostel allotment are out of scope. The message numbering follows the order in which the calls happen, with nested numbers for calls made while an outer call is still active.

Assumptions:

- One registration per student per semester; a second submission edits the existing registration (not shown).
- Eligibility rules are owned by Programme so that Student stays a data holder.
- Seats are reserved at enrolment time and confirmed on payment; the diagram shows the paid path.
- Payment is a single online transaction; the gateway is not shown, the receipt records the mode.
- The fee is computed from credits at a fixed rate per credit set on the programme.

Steps

1. Write the scenario as the numbered list in the message table first; the numbers are the diagram.
2. Model, Add Diagram, Communication Diagram (StarUML's name for the collaboration diagram); name it `RegisterForSemester`.
3. Toolbox Lifeline: place the actor Student at the left and the six objects in a rough ring. In the Editor set Name and Type (`reg` and `Registration`).
4. Toolbox Connector: draw a line between every pair of objects that exchange a message.
5. Toolbox Forward Message: click the connector, type the message text, then set Sequence Number in the Editor to the value from the table (1, 1.1, 1.2 and so on). StarUML draws a small arrow beside the link showing direction.
6. Use Reverse Message for replies that carry data (1.6, 2.4). Save and export as PNG.

Diagram

Object	Class	Role in Register for Semester
student	actor Student	Chooses courses, pays the fee, receives the slip
s1	Student	Starts the registration, keeps the list of registrations
prog	Programme	Decides eligibility, lists the semester's courses
reg	Registration	Holds the enrolments, computes the fee, confirms
c1	Course	Knows its seat limit and reserves seats
enr	Enrolment	Ties one course into the registration
fee	FeeReceipt	Records the payment

No	From	To	Message
1	student	s1	register(semester, courseCodes)
1.1	s1	prog	isEligible(s1, semester)
1.2	s1	reg	create Registration(s1, semester)
1.3	reg	prog	getCourses(semester)
1.4	reg	c1	hasSeat() (for each chosen course)
1.5	reg	enr	create Enrolment(reg, c1) (when seat free)
1.5.1	enr	c1	reserveSeat()
1.6	s1	student	fee due (reply, from reg.computeFee())
2	student	reg	payFee(amount, mode)
2.1	reg	fee	create FeeReceipt(reg, amount, mode)
2.2	reg	reg	confirm()
2.3	reg	s1	addRegistration(reg)
2.4	reg	student	registration slip (reply)

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

```

1          1: register(semester, courseCodes)
2  +-----+ -----> +-----+ 1
3  | student | ← - - 1.6: fee due - - - - - | s1 : Student | - -
4  | (actor) |                                     +-----+
5  +-----+                                     |      ^
6  |      ^                                     1.2: <<create>> Registration(s1, sem)
7  |      |                                     v      | 2.3:
8  |      | 2: payFee(amount, mode)             +-----+
9  +----|-----> | reg : Registration |
10      |             +-----+
11      +- - - 2.4: registration slip - - - - - |      |
12      |      |
13      |      | 1.4: hasSeat() |      |
14      |      | [each course] |      | +
15      |      | v      |      |
16      |      | +-----+ |      | +
17      |      | | c1 : Course | | 1.5: <<c
18      |      | +-----+ |
19      |      | ^      v
20      |      | |      +-----
21      |      | +---- | enr : Enrolme
22      |      | 1.5.1: reserveSeat() +-----

```

Explanation

Read the diagram by following the numbers, not the layout. Message 1 starts the outer activation on s1; everything numbered 1.x happens before s1 returns to the student with 1.6. Message 2 is a new activation started by the student after paying, so its sub-messages are 2.x. Nested numbers such as 1.5.1 mean the call is made while 1.5 is still executing: the enrolment reserves the seat inside its own constructor. Every arrow rides on a link, and every link corresponds to an association in the class diagram: Student to Programme, Student to Registration, Registration to Course, Registration to Enrolment, Enrolment to Course, Registration to FeeReceipt. If you find yourself needing an arrow without a link, the class diagram is missing an association. The same scenario drawn as a sequence diagram would have identical message labels; only the geometry changes.

Question 10

Problem Statement

■ Write in lab record

Draw Collaboration Diagram for Online Banking System.

Solution

■ Write in lab record

Assumptions

This diagram uses the Online Banking System described in Session 1 Question 3 and traces the scenario that produced the object diagram there: the use case Transfer Funds, in which customer Ravi moves five thousand rupees from his savings account SB1001 to Priya's savings account SB1002, which he has saved as beneficiary ben1. The objects are the same as in the object diagram: `sbi : Bank`, `login1 : Credential`, `acc1 : SavingsAccount`, `ben1 : Beneficiary`, `acc3 : SavingsAccount`, and the two transactions `t1` and `t2` that the scenario creates. Ravi himself is the actor.

Ravi logs in with his user id and password. The bank object finds the credential for that user id and asks it to verify the password; on success the bank returns a session to Ravi. Ravi then submits the transfer: source account, beneficiary and amount. The bank first asks the credential whether the amount is within the daily transfer limit that remains for today. It then asks the beneficiary for its target account, which resolves to `acc3` because both accounts are in the same bank. The bank generates a reference number and asks the source account to debit the amount. The savings account checks that the balance after the debit stays above its minimum balance, reduces its balance, and creates a Transaction of type DEBIT with the reference. The bank then asks the target account to credit the amount; the target account increases its balance and creates a Transaction of type CREDIT with the same reference. Finally the bank records the amount against the credential's daily usage and returns a receipt with the reference number to Ravi.

If the password fails, the login returns an error and the transfer cannot start. If the daily limit or the minimum balance check fails, the bank returns the reason and no transaction objects are created. Transfers to another bank go through a clearing system that is outside this diagram; only

the debit leg would appear. Adding a beneficiary, viewing statements and changing the password are separate use cases.

Assumptions:

- Ravi is already registered for net banking; login1 exists.
- The bank object reaches accounts directly; in the class model it navigates through Branch and Customer, and the direct links here are drawn as derived.
- Both legs of a transfer share one reference number so that they can be traced or reversed together.
- Debit and credit happen in one database transaction; the diagram shows the success path with the failure cases as guarded replies.
- The receipt is displayed on screen; SMS or e-mail alerts are not modelled.

Steps

1. Model, Add Diagram, Communication Diagram; name it `TransferFunds`.
2. Place the actor Ravi at the left, `sbi : Bank` in the centre, and the other objects around it as in the sketch. Set Name and Type for each in the Editor.
3. Draw Connectors: ravi to sbi, sbi to login1, sbi to ben1, ben1 to acc3, sbi to acc1, sbi to acc3, acc1 to t1, acc3 to t2.
4. Add Forward Messages on the connectors with the sequence numbers from the table; use Self Message for 2.3.1 on acc1; mark 2.3.2 and 2.4.1 with the create stereotype.
5. Add Reverse Messages for 1.2 and 2.6 back to the actor. Save and export as PNG.

Diagram

Object	Class	Role in Transfer Funds
ravi	actor Customer	Logs in, submits the transfer, receives the receipt
sbi	Bank	Coordinates login, limit check, debit and credit
login1	Credential	Verifies the password, tracks the daily limit
acc1	SavingsAccount	Source account: checks minimum balance, debits
ben1	Beneficiary	Maps the nickname to the target account
acc3	SavingsAccount	Target account: credits
t1	Transaction	Debit leg, created by acc1
t2	Transaction	Credit leg, created by acc3

No	From	To	Message
1	ravi	sbi	login(userId, password)
1.1	sbi	login1	verify(password)
1.2	sbi	ravi	session ok (reply)
2	ravi	sbi	transferFunds(SB1001, ben1, 5000)
2.1	sbi	login1	withinDailyLimit(5000)
2.2	sbi	ben1	getTargetAccount() returns acc3
2.3	sbi	acc1	debit(5000, ref)
2.3.1	acc1	acc1	checkMinBalance(5000)
2.3.2	acc1	t1	create Transaction(DEBIT, 5000, ref)
2.4	sbi	acc3	credit(5000, ref)
2.4.1	acc3	t2	create Transaction(CREDIT, 5000, ref)
2.5	sbi	login1	addToDailyUsage(5000)
2.6	sbi	ravi	receipt(ref) (reply)

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

```

1                                     1.1: verify(password)
2                                     2.1: withinDailyLimit(5000)
3                                     2.5: addToDailyUsage(5000)
4                                     +----->
5      1: login(userId, password)      |
6      2: transferFunds(SB1001,ben1,5000)
7  +-----+ -----> +-----+ 2.2: getTargetAccount()
8  | ravi |           | sbi : Bank | ----->
9  | (actor)| ← - - - - - +-----+
10 +-----+ 1.2: session ok      |      |
11           2.6: receipt(ref)    |      |
12           |                    |      |
13           2.3: debit(5000,ref) | 2.4: credit(5000, ref)
14           |                    +----->
15           v
16           +-----+
17           | acc1 : SavingsAccount | --+
18           +-----+ | 2.3.1: checkMinBalanc
19           | ^-----+
20           | 2.3.2: <<create>>
21           | Transaction(DEBIT, 5000, ref)
22           v
23           +-----+
24           | t1 : Transaction |
25           +-----+

```

Explanation

The diagram has two top-level messages because the actor acts twice: once to log in and once to transfer. Three messages share the link between sbi and login1 (1.1, 2.1, 2.5); a collaboration diagram draws one link and stacks the messages beside it, which is exactly the compression it offers over a sequence diagram. The nested 2.3.1 and 2.3.2 are done by acc1 inside its debit operation, so the balance rule and the transaction record belong to the account class, not to the bank. After message 2.4.1 the objects match the Session 1 object diagram: acc1 at 40000, acc3 at 27000, t1 and t2 with reference TR26092601. That is the check to make in the viva: the collaboration diagram must leave the system in the state the object diagram shows.

Question 11

Problem Statement

■ Write in lab record

Draw a Collaboration diagram for Employee Management System.

Solution

■ Write in lab record

Assumptions

The Employee Management System is the one described in Session 3 Question 8: employees belong to departments, each department has a manager who is also an employee, every employee holds a LeaveBalance per leave type, and a LeaveRequest records one application with its dates, type and status. This diagram shows the same use case, Apply for Leave, as a collaboration diagram. The messages and their numbers are identical to the sequence diagram; only the arrangement changes from time-ordered columns to a graph of linked objects.

The scenario: an employee applies for leave giving the dates and type. The employee object asks its leave balance whether enough days remain. If so, it creates a LeaveRequest in status Pending, forwards it to the manager it reports to, and returns the request id to the employee. Later the manager opens the request and approves it. The manager object sets the status on the request; the request notifies the employee object; the employee object deducts the days from its balance. The manager receives confirmation that the decision has been recorded. If the balance is short, the application is refused at message 1 and nothing further happens; if the manager rejects, message 2.2.1 is skipped. Both conditions are written as guards in square brackets on the messages.

The objects are the actor Employee, `emp : Employee`, `bal : LeaveBalance`, `req : LeaveRequest`, `mgr : Manager` and the actor Manager. The links are Employee to LeaveBalance (composition in the class model), Employee to LeaveRequest (creates), Employee to Manager (reports to), and Manager to LeaveRequest (approves). There is no link between LeaveBalance and Manager or between LeaveBalance and LeaveRequest, which is why the deduction goes through the employee object.

Attendance, payroll and department transfer are separate use cases. Multi-level approval and HR override are out of scope, as is the cancellation of an approved leave, which would need a credit message back to the balance.

Assumptions:

- Same as Question 8: one approval level, balance checked at application and deducted on approval, notification is an in-system message.
- The manager object is reached through the existing reports-to association; no Department object is needed on this diagram.
- Guards on 1.2 and 2.2.1 replace the alt fragments of the sequence diagram.
- Both actors are drawn because they send the two top-level messages at different times.

Steps

1. Copy the message table from Session 3 Question 8; the numbers do not change.
2. Model, Add Diagram, Communication Diagram; name it `ApplyForLeaveCollab`.
3. Place the actor Employee at the left, the actor Manager at the right, `emp : Employee` and `mgr : Manager` in the middle row, `bal : LeaveBalance` below emp and `req : LeaveRequest` between emp and mgr.
4. Draw Connectors: employee to emp, emp to bal, emp to req, emp to mgr, mgr to req, manager to mgr.
5. Add Forward Messages with sequence numbers 1, 1.1, 1.2, 1.3, 2, 2.1, 2.2, 2.2.1; type the guards `[balance sufficient]` on 1.2 and `[approved]` on 2.2.1 in the message text. Add Reverse Messages 1.4 and 2.3 to the actors.
6. Save and export as PNG.

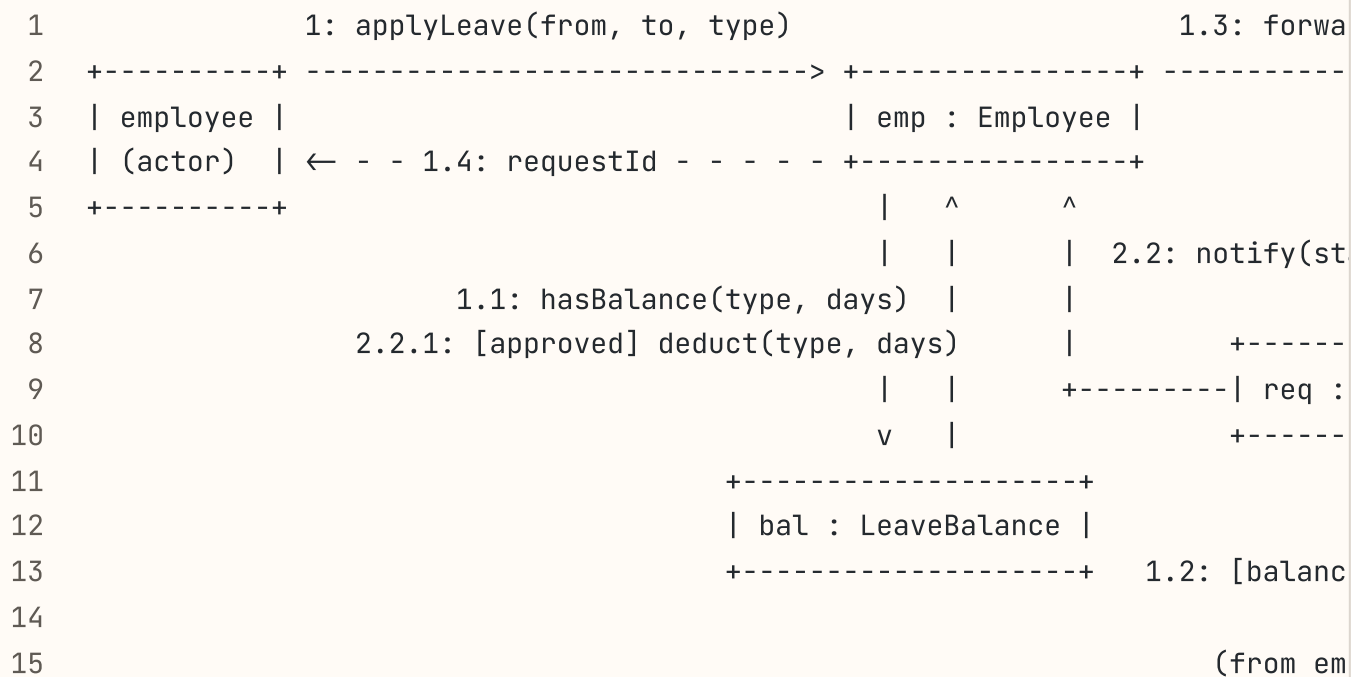
Diagram

Object	Class	Role in Apply for Leave
employee	actor Employee	Applies, receives the request id and outcome
emp	Employee	Checks balance, creates and forwards the request, deducts on approval
bal	LeaveBalance	Days remaining per leave type
req	LeaveRequest	Dates, type, status; notifies the employee
mgr	Manager	Records the decision
manager	actor Manager	Approves or rejects

No	From	To	Message
1	employee	emp	applyLeave(from, to, type)
1.1	emp	bal	hasBalance(type, days)
1.2	emp	req	[balance sufficient] create LeaveRequest(from, to, type)
1.3	emp	mgr	forward(req)
1.4	emp	employee	requestId (reply)
2	manager	mgr	approve(requestId)
2.1	mgr	req	setStatus(Approved)
2.2	req	emp	notify(status)
2.2.1	emp	bal	[approved] deduct(type, days)
2.3	mgr	manager	decision recorded (reply)

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch



Explanation

Put this diagram next to the Session 3 sequence diagram and check that every numbered label appears once in each. The sequence diagram needed two alt fragments; here the same conditions are guards written before the message name on 1.2 and 2.2.1, because a collaboration diagram has no fragments. Messages 1.1 and 2.2.1 travel the same link between emp and bal in opposite phases of the scenario, which is only visible in this form: the link is the composition from the class diagram, and the balance never talks to anyone but its employee. The diagram also makes the design choice explicit that the manager does not touch the balance; if an examiner asks why, the answer is that there is no association between Manager and LeaveBalance and adding one would let a manager change balances outside the leave process.

Viva Questions

✗ Do not copy. Read for understanding and the viva

Q: What is a collaboration diagram? **A:** An interaction diagram that shows the objects taking part in a scenario, the links between them and the messages sent along the links, with sequence numbers giving the time order. UML 2 calls it a communication diagram.

Q: How do you convert a sequence diagram into a collaboration diagram? **A:** Keep the same objects and messages. Draw a link for every pair that communicates, place each message beside

its link with an arrow for direction, and number the messages in the order they occur using nested numbers for calls made inside another call.

Q: What does message number 2.3.1 mean? **A:** The first message sent while message 2.3 is still executing, which itself was the third message sent during top-level message 2.

Q: Where do the links come from? **A:** From the associations in the class diagram. A link is an instance of an association; if a message needs a link that has no association, the class diagram is incomplete.

Q: How do you show a condition without an alt fragment? **A:** Write a guard in square brackets before the message name, for example `2.2.1: [approved] deduct(type, days)`. Iteration is shown with an asterisk, for example `1.4: *[each course] hasSeat()`.

Q: Can a collaboration diagram show object creation? **A:** Yes, label the message with the create stereotype and optionally stereotype the object as new. The Registration, Enrolment and Transaction objects here are created inside the scenario.

Q: Which diagram is better, sequence or collaboration? **A:** They carry the same information. Sequence diagrams are easier to read for long scenarios with many steps; collaboration diagrams show structure and are easier to check against the class diagram.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Numbering messages 1, 2, 3 in a flat list even when calls are nested; the nesting is what shows which object is in control.
- Drawing an arrow between two objects with no link, or a link with no message on it.
- Changing message names between the sequence and the collaboration diagram of the same scenario; the evaluator compares them.
- Placing time-ordering by position on the page; only the numbers order the messages in this diagram.
- Forgetting the reply messages that carry results back to the actor (1.6, 2.4 in Question 9), so the diagram never shows what the user gets.
- Adding alt or loop boxes; those belong to sequence diagrams. Use guards and the asterisk instead.

Session Summary

■ Write in lab record

- Question 9: Register for Semester collaboration diagram, seven objects including the actor, thirteen numbered messages, three object creations
- Question 10: Transfer Funds collaboration diagram on the Session 1 banking objects, eight objects, thirteen numbered messages, ends in the state shown by the Session 1 object diagram
- Question 11: Apply for Leave collaboration diagram, six objects including two actors, ten messages numbered identically to the Session 3 sequence diagram, guards in place of alt fragments
- Object and message tables, ASCII sketch and PlantUML file for each diagram

[✎ Edit this page](#)

[< Previous](#)
Session 3

Session 5 [Next >](#)