02291 System Integration

Goal-oriented Requirements Engineering: Solutions to exercises

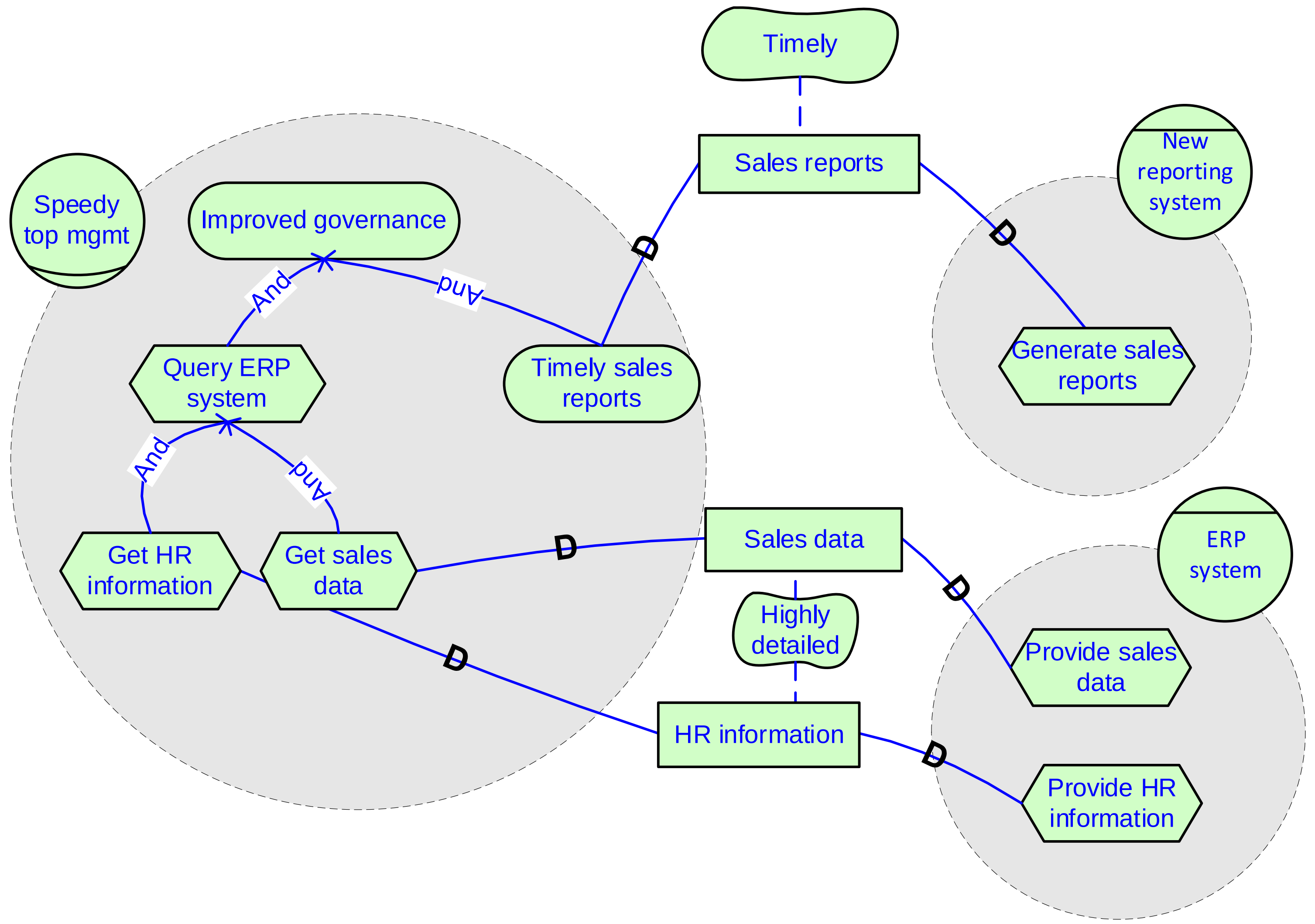
Exercise 1 – Speedy

Speedy is an international delivery company that needs to refocus on which markets it should operate. Indeed, Speedy lacks a clear understanding on which markets are the most profitable. Thus, to address this issue, Speedy’s top management aims at improving their governance. To achieve this, the top management requires sales reports containing timely information. Thus, the top management decided to build a new reporting system to automatically produce such reports. After inspecting the sales reports, the top management may also need to query the existing ERP system to get detailed sales and HR information.

Prepare an ArchiMate and an I-star model that captures the formalizes Speedy’s new architecture.

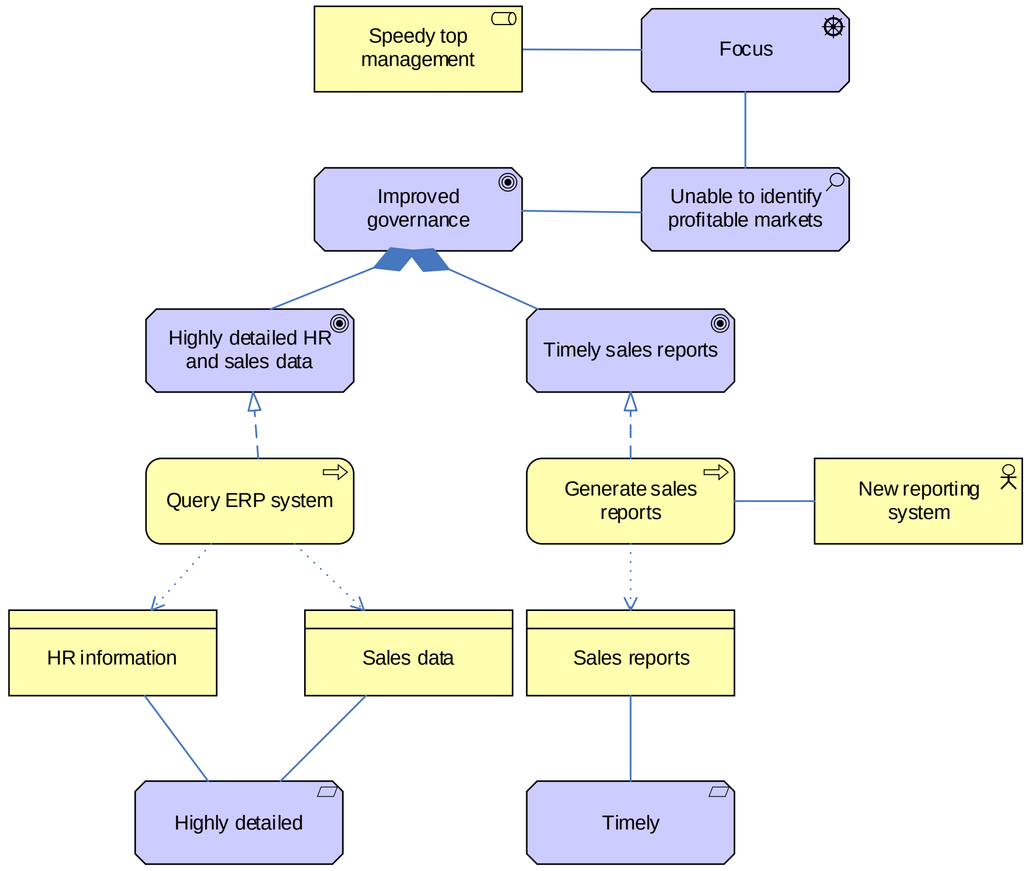
- Can you model all the elements and relations described in this exercise with both languages? If not, which elements can be captured only in ArchiMate? Which ones in I-star?

Exercise 1 – I-star model



We cannot model the driver and assessment concepts

Exercise 1 – ArchiMate model



We cannot explicitly model the social dependency between Speedy's top management and the new reporting systems

Exercise 2 – IC

IC is an insurance company which wants to offer a new insurance service for small assets (<2000$) managed completely online for reliable customers.

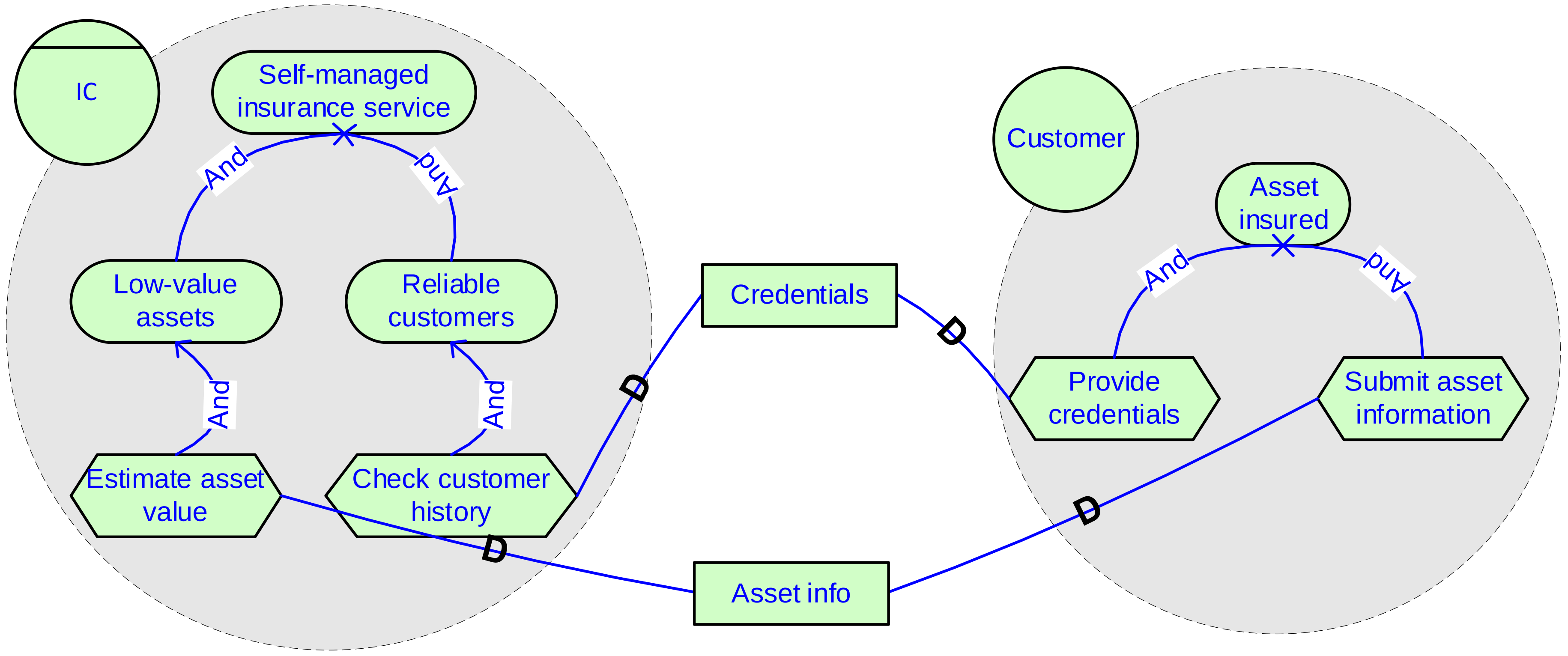
To achieve this, a customer who wants his assets to be insured has to provide its credentials and photo of the asset and its details (serial number, purchase date) to IC. To ensure that the customer is reliable and the asset inexpensive, IC will then check the customers credentials and past history and estimate the asset’s price.

Prepare an ArchiMate and an I-star model that captures the formalizes IC’s requirements.

- Can you model all the elements and relations described in this exercise with both languages? If not, which elements can be captured only in ArchiMate? Which ones in I-star?

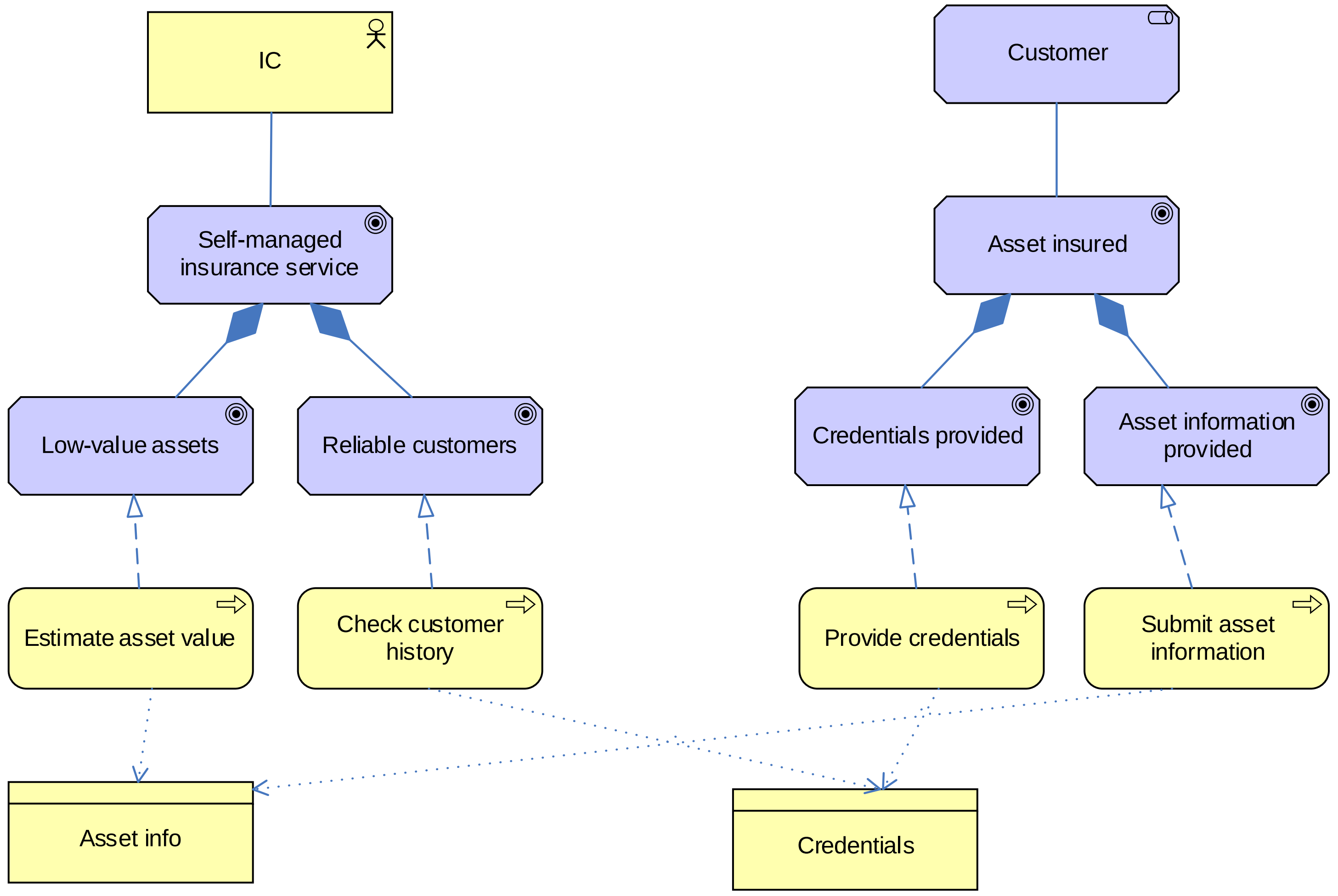
Exercise 2 – I-star model

The model captures all concepts and dependencies

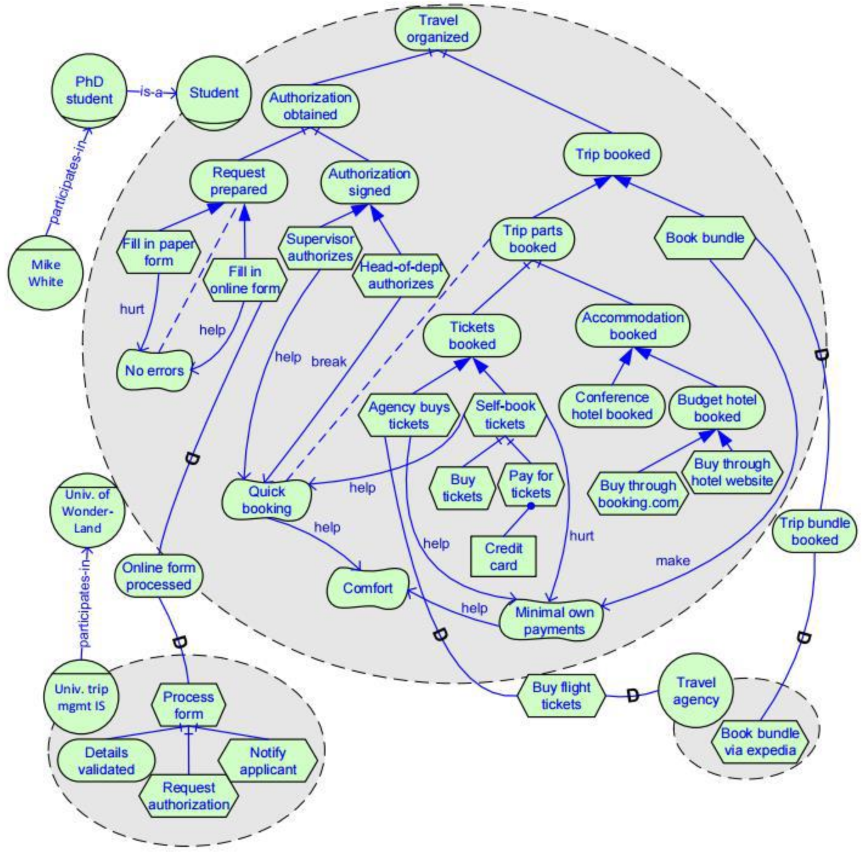


Exercise 2 – ArchiMate model

We cannot explicitly model the social dependencies between IC and the customer



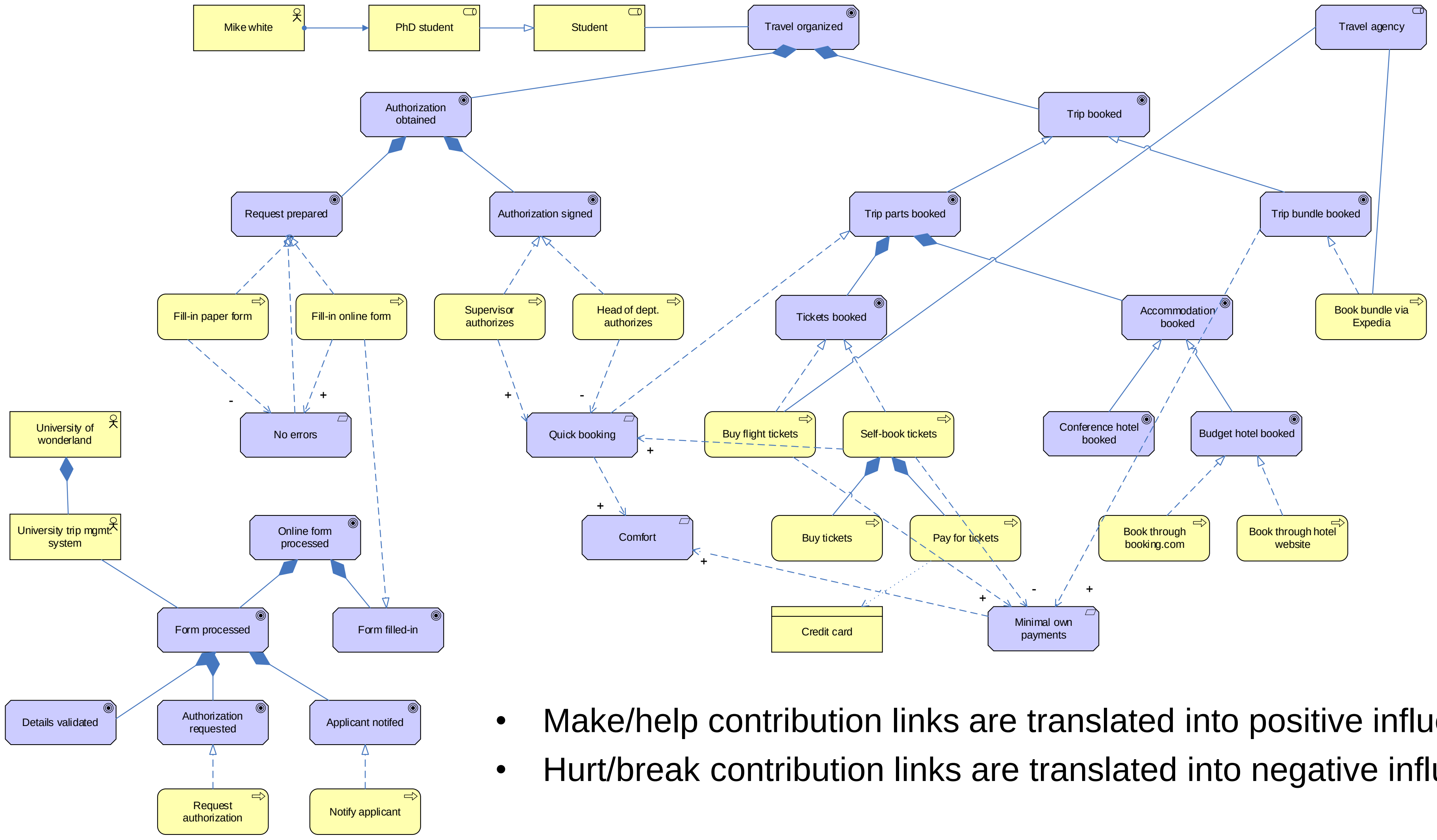
Exercise 3 – University travel reimbursement



- This I-star model represents a university travel reimbursement system
- Prepare an ArchiMate model representing the same system
  - Can you model all the elements and relations in this I-star model?
  - If not, which elements and relations cannot be modeled?

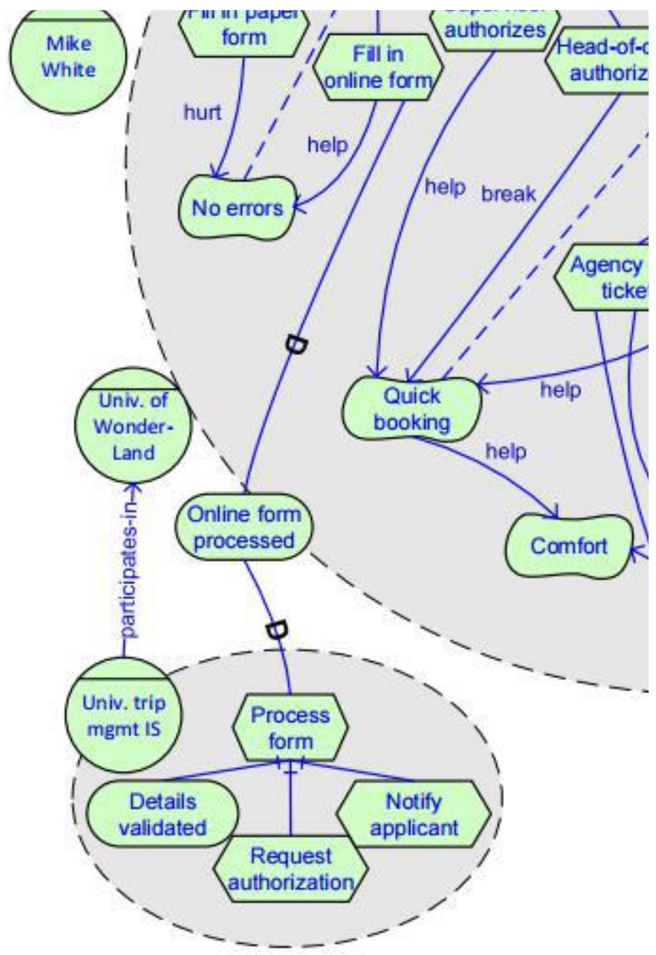
Model from: F. Dalpiaz, X. Franch, and J. Horko – iStar 2.0 Language Guide
https://arxiv.org/abs/1605.07767

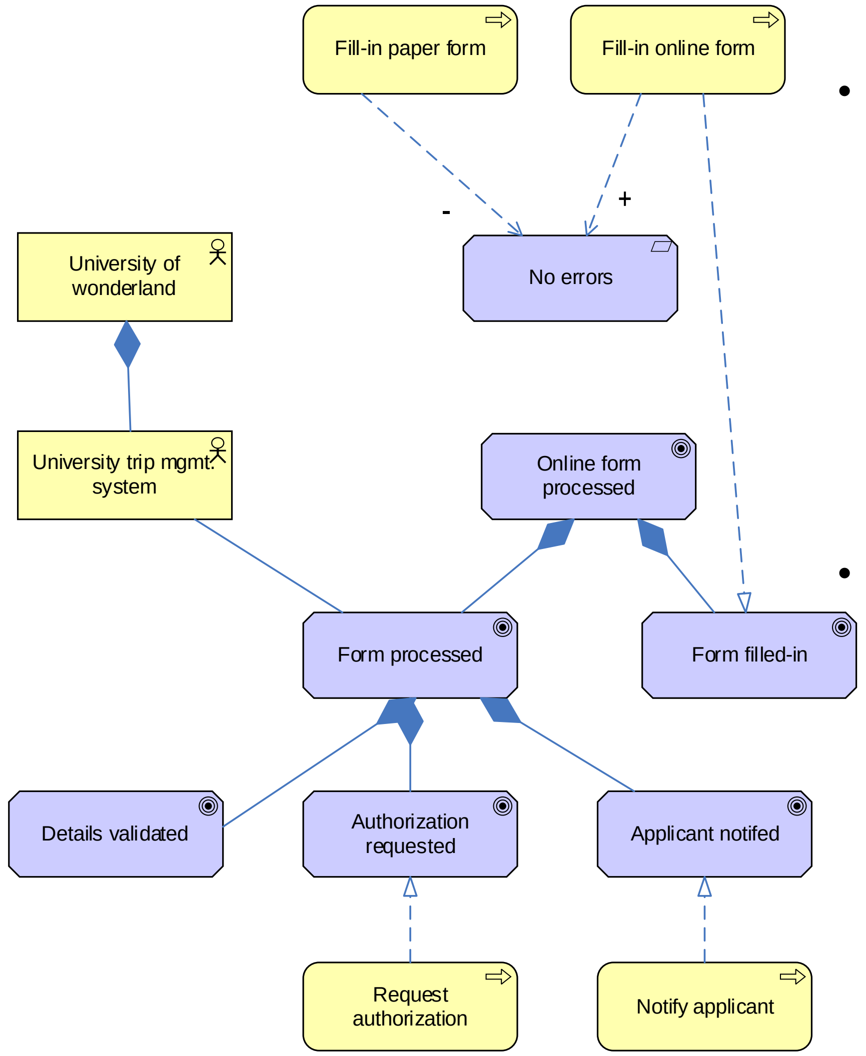
Exercise 3 – ArchiMate model



- Make/help contribution links are translated into positive influence relations
- Hurt/break contribution links are translated into negative influence relations

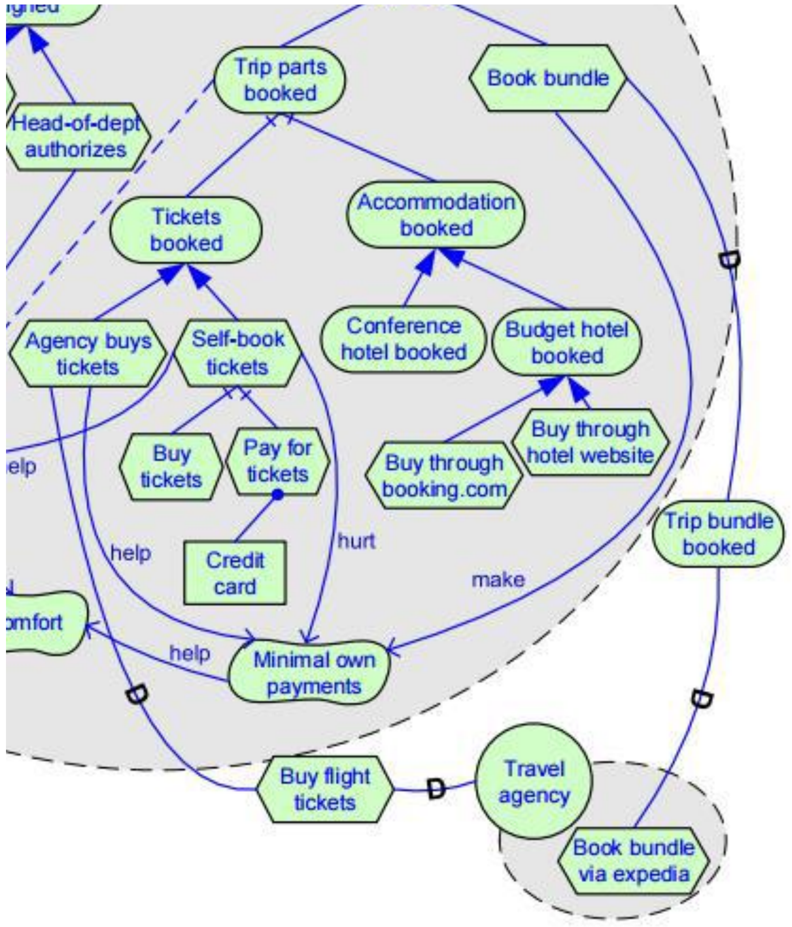
Exercise 3 – Comparison

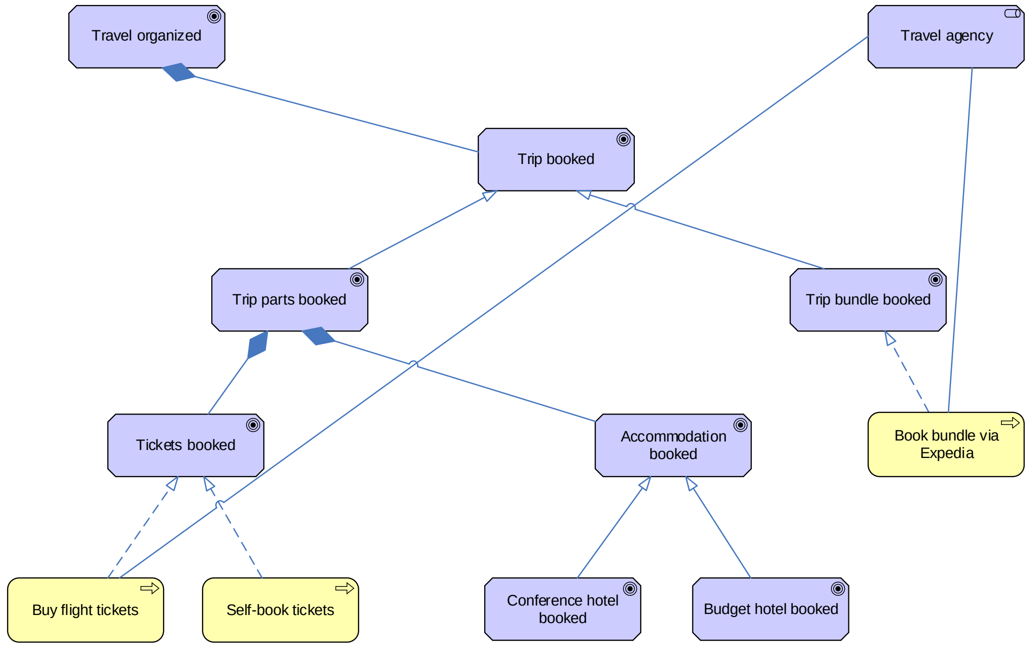




- We cannot model the social dependency between IS and student
  - We can approximate it by breaking the dependum (Online form processed) into subgoals, which are inclusively realized by subgoals belonging to the two entities
- We cannot model that goals are refined from tasks:
  - We either turn Process form into a goal (as depicted), or turn Details validated into a task

Exercise 3 – Comparison





- We cannot model the social dependencies between travel agency and student
- We can approximate them:
  - We replace task Agency buys tickets with dependum Buy flight ticketst, and we assign it to the travel agency
  - We replace task Book bundle with dependum Trip bundle booked, which is realized by task Book bundle with Expedia

02291 System Integration

Introduction to ArchiMate: Solutions to exercises

© Giovanni Meroni

Exercise 1 – Speedy

Speedy is a delivery company that wants to create a new reporting system for the top management. After inspecting the sales reports, the top management may also need to query the existing ERP system, based on Oracle Fusion, to get detailed sales and HR information.

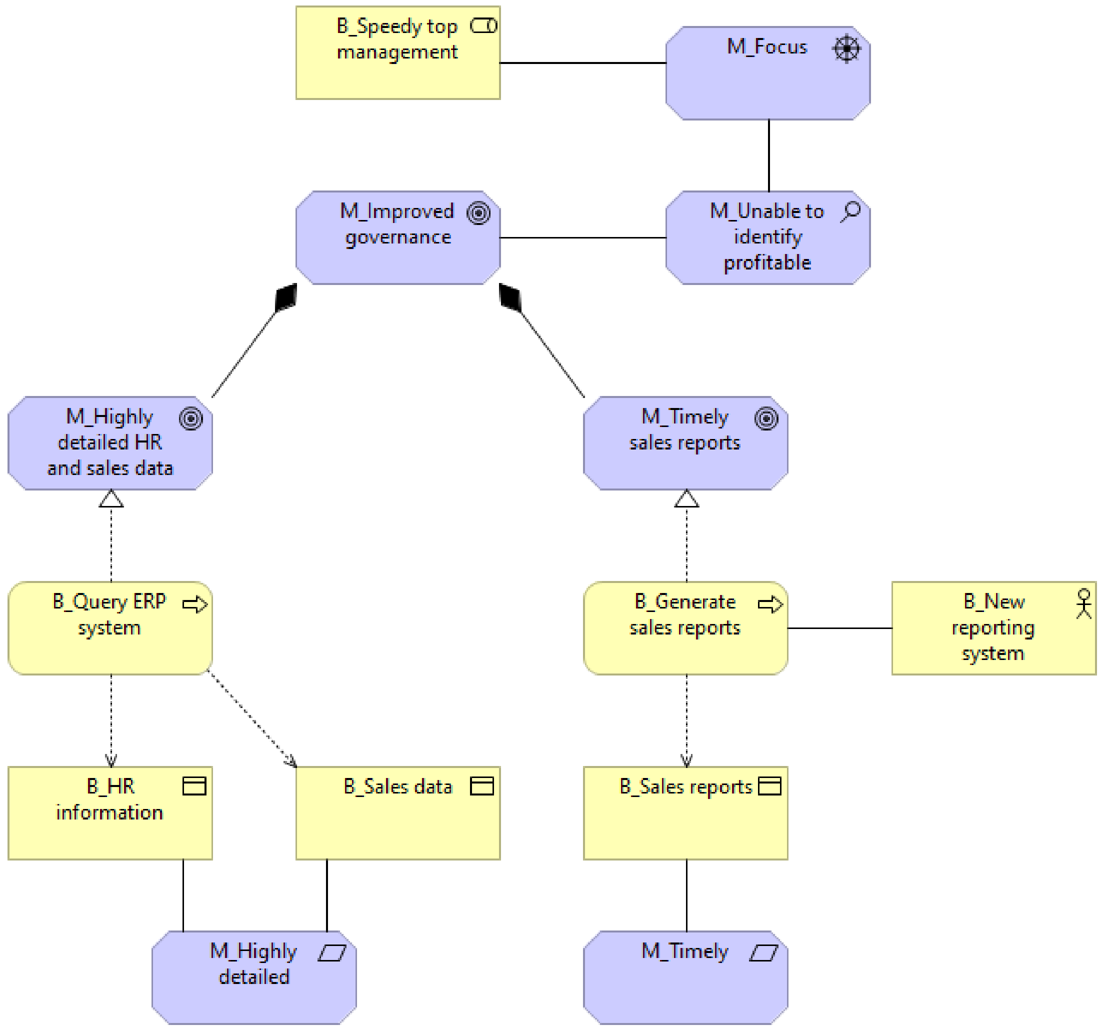
To implement the reporting system, a data warehouse (DW) based on Microsoft SQL Server 2022 will be used. The DW will run three components responsible for extracting sales data from the ERP database, managing analysis data, and generating sales reports.

The DW will run on-premise on a different node than the one running the ERP. Both nodes will share the same LAN.

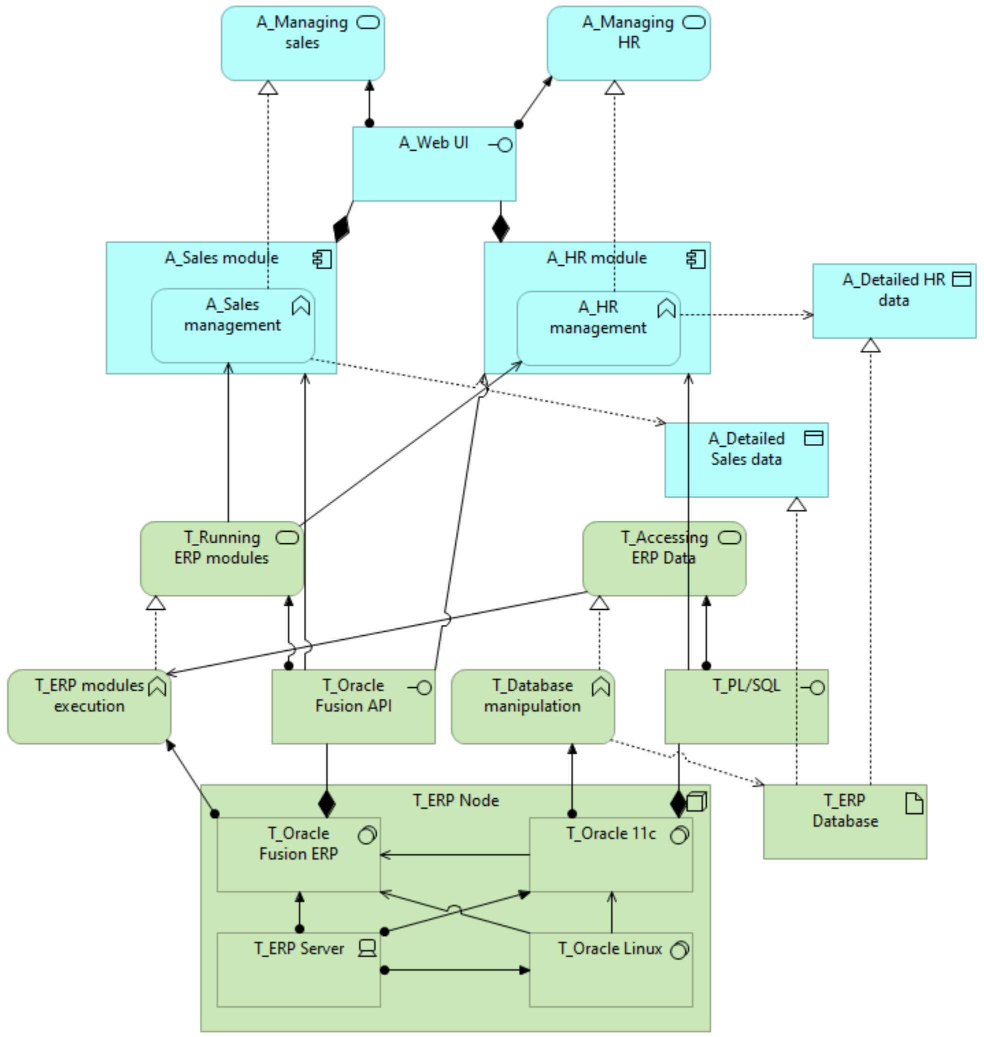
ArchiMate models representing the business goals and the existing ERP system are enclosed in the next slides.

Starting from these models, create a complete ArchiMate model that: 1) fully represents the business layer and links it to the application layer, 2) extends the application and technology layers by adding the elements required for the new reporting service.

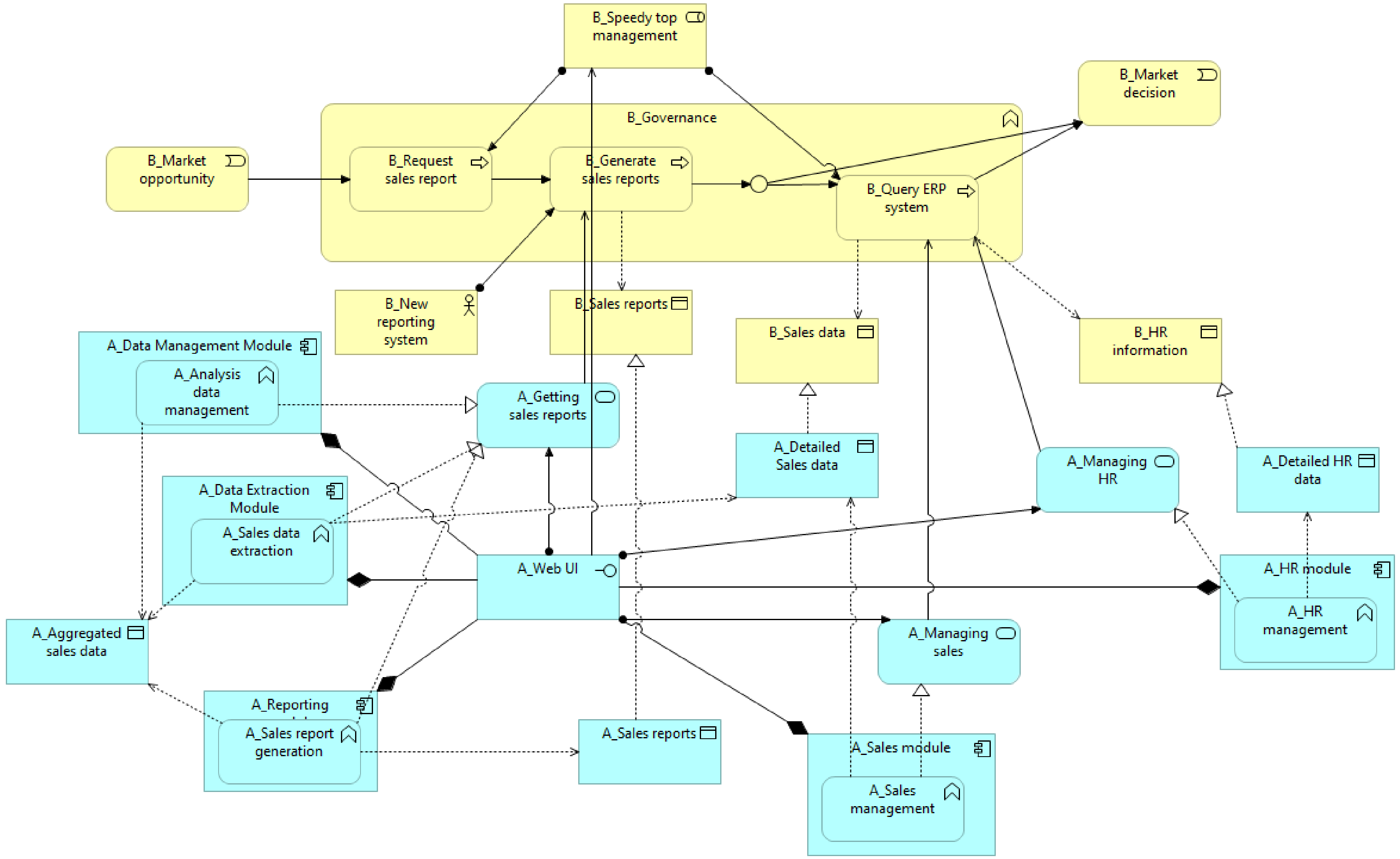
Exercise 1 – Motivation



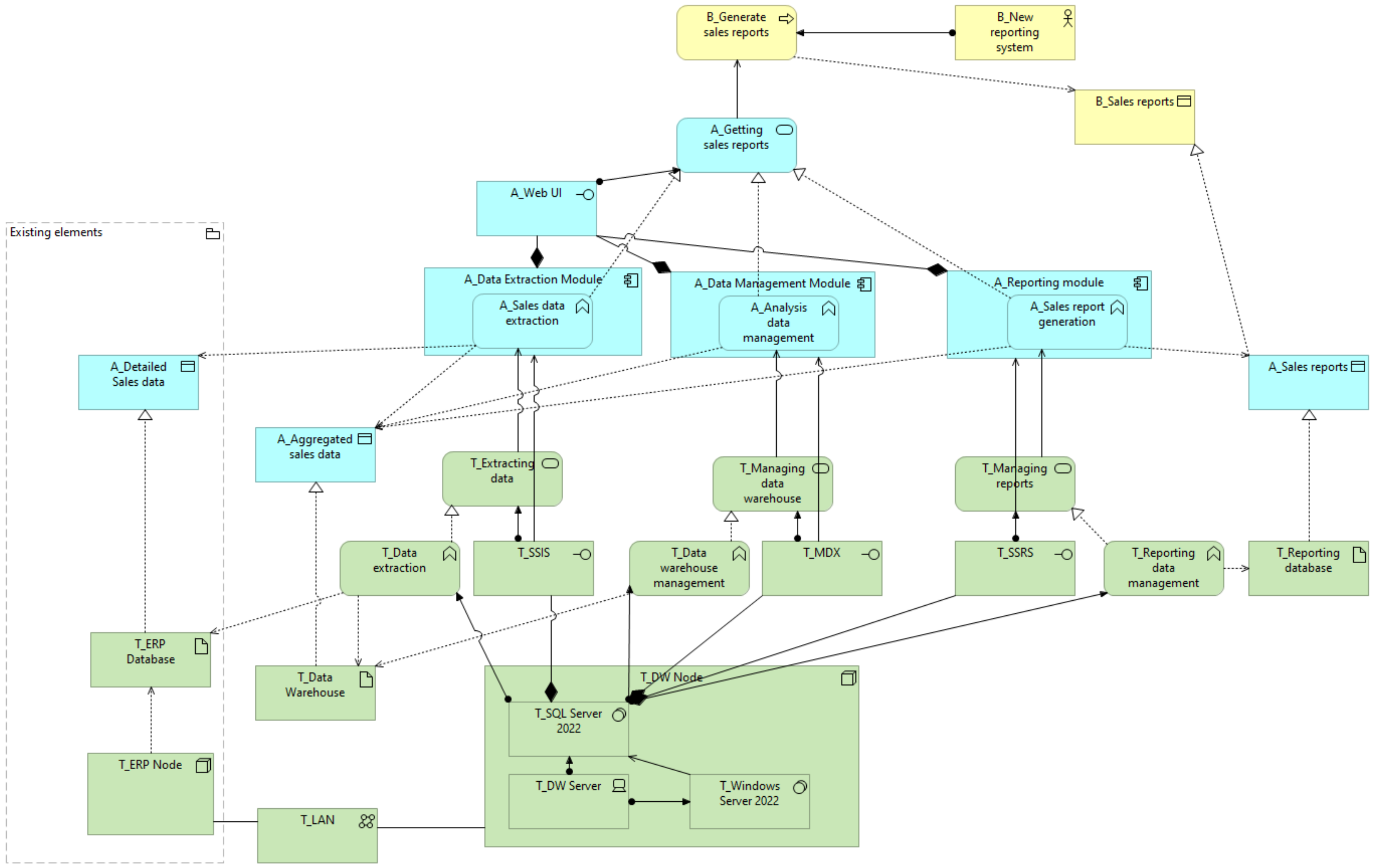
Exercise 1 – Existing ERP system



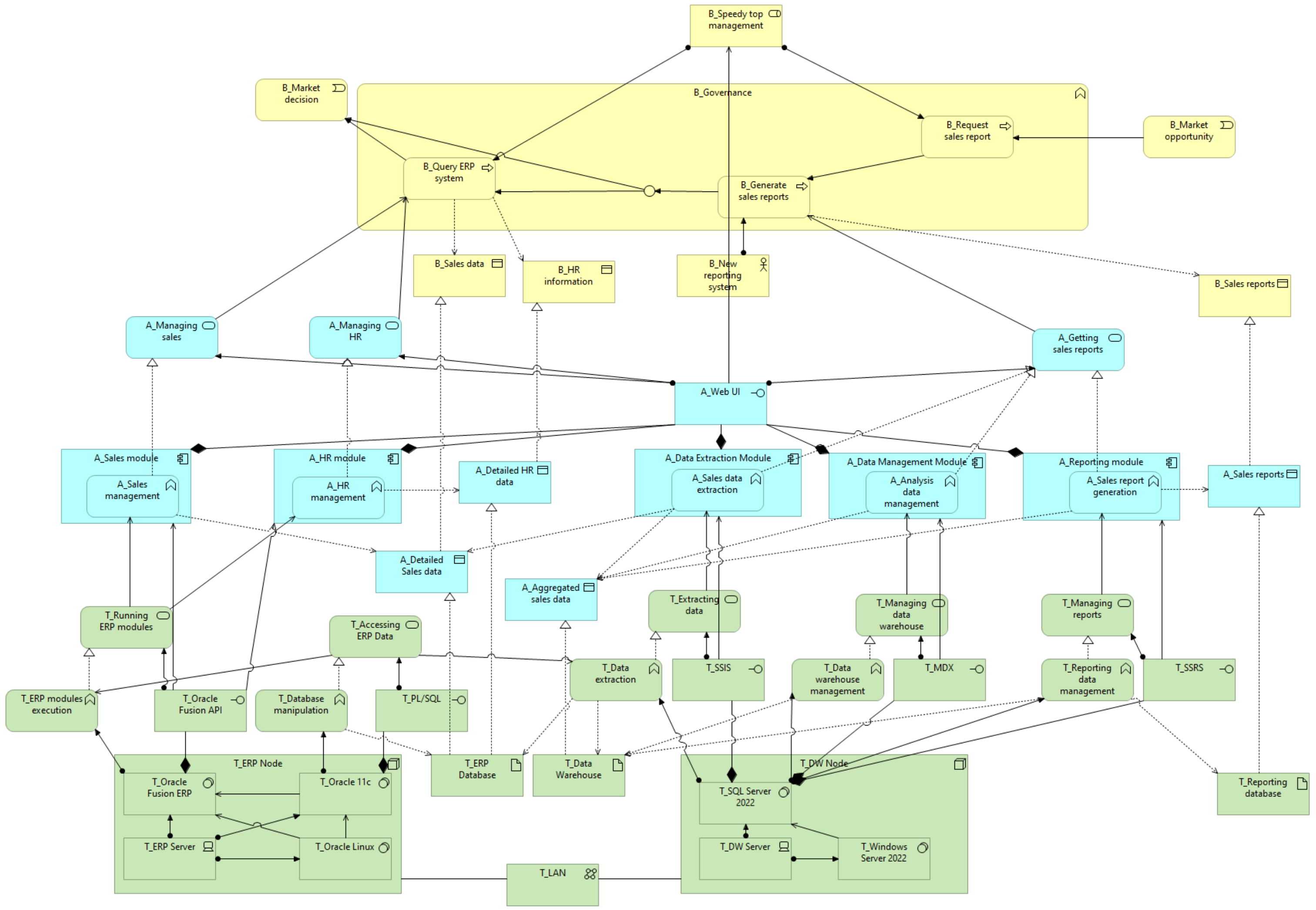
Exercise 1 – Business and Application layers



Exercise 1 – New infrastructure



Exercise 1 – Complete model



Exercise 2 – IC

IC is an insurance company which wants to offer a new insurance service for small objects (<2000$) managed completely online for reliable customers.

To achieve this, a customer who wants his assets to be insured has to provide its credentials and photo of the asset and its details (serial number, purchase date) to IC. To ensure that the customer is reliable and the asset inexpensive, IC will then check the customers credentials and past history and estimate the asset's price. If the checks succeed, a proposal will be generated. Otherwise, the request will be rejected.
To support the new service, IC needs to develop a new application with the following functionalities: proposal generation, price estimation, customer reliability check.

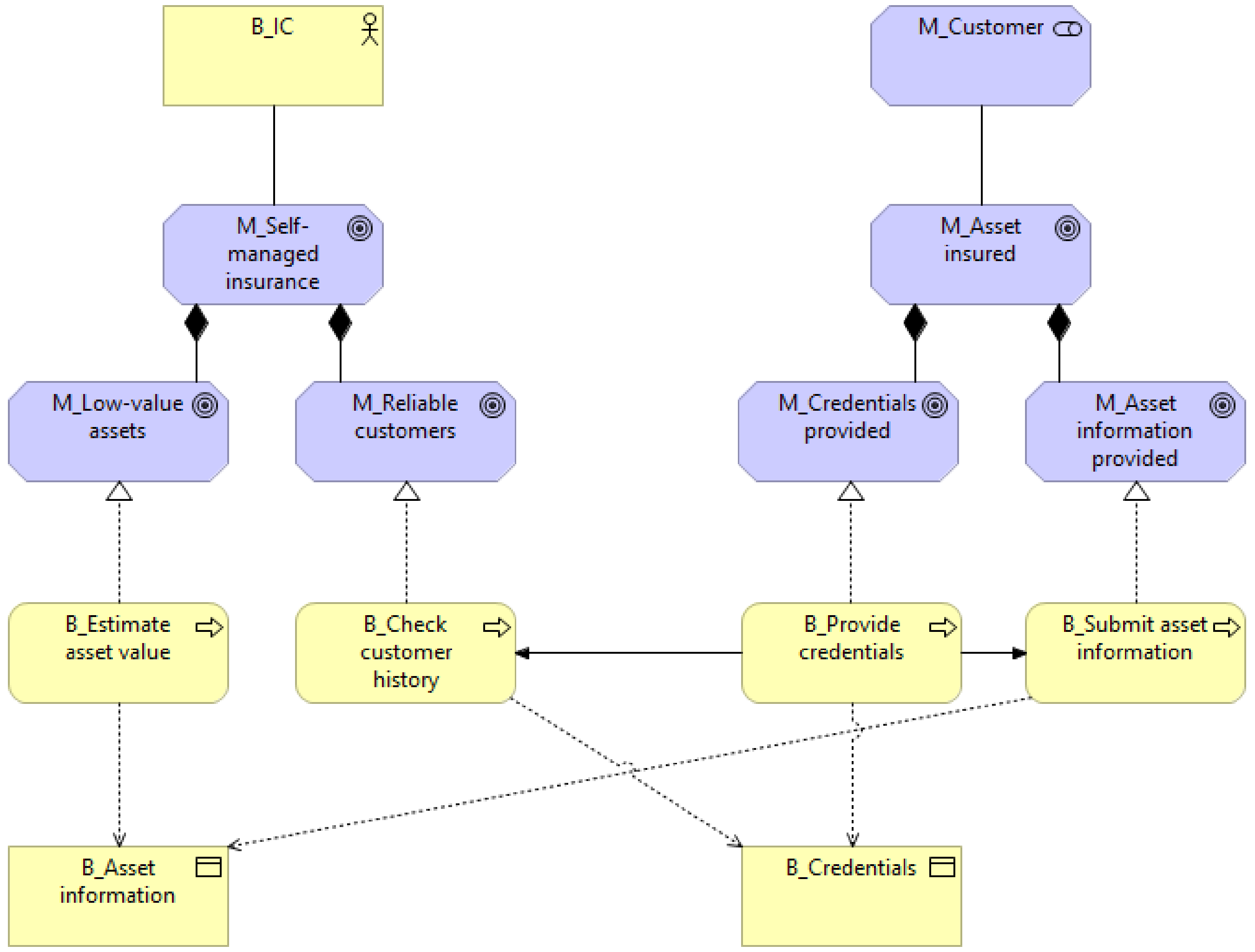
The new application will be implemented as a Java EE component running on Apache Tomcat. It will also rely on an operational database deployed on Oracle MySQL. Both the database and the application will reside on a dedicated server, which is connected to the corporate LAN.

To estimate an item price, an external service will be used.

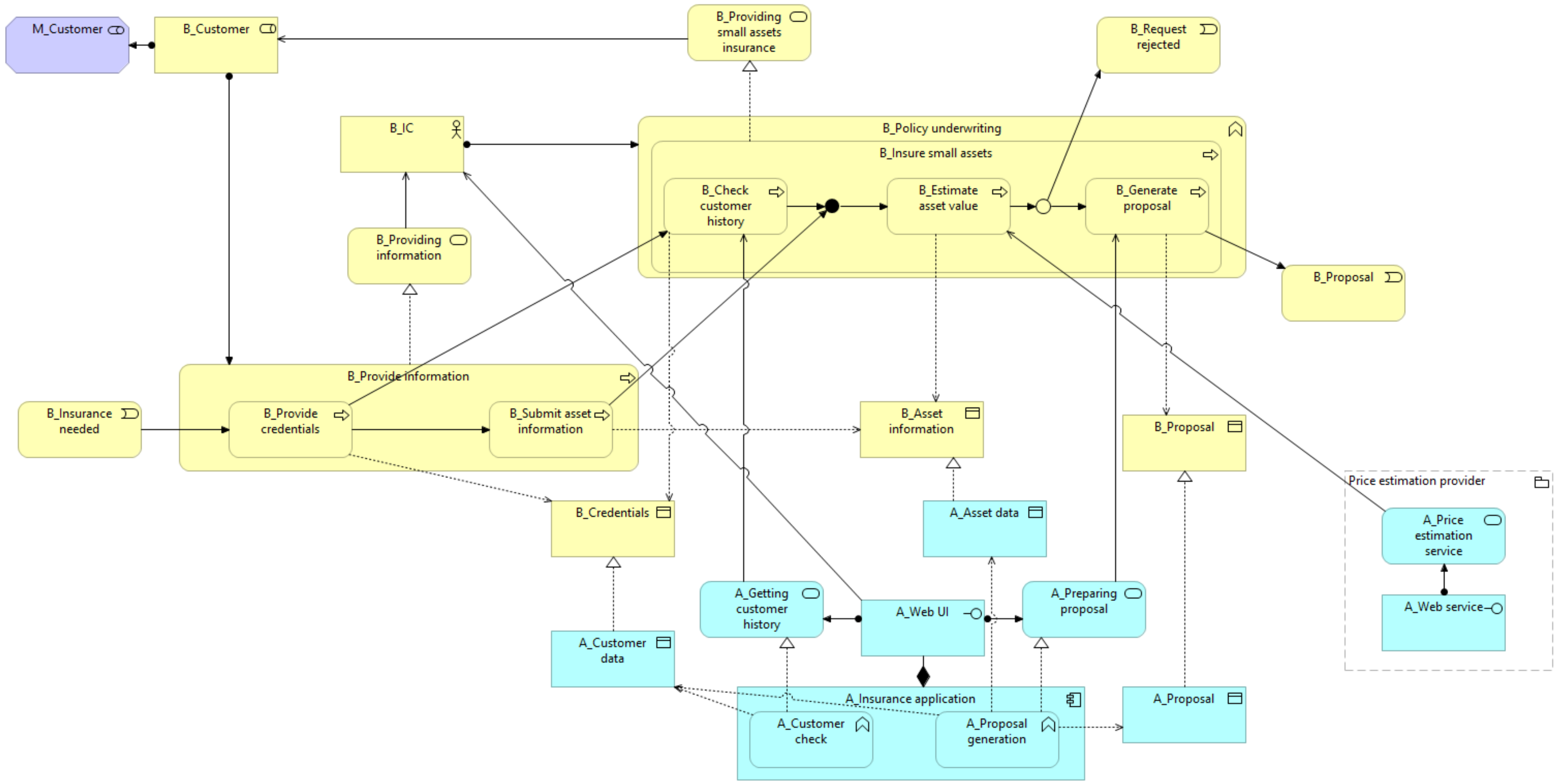
To check the customer reliability, the application needs data coming from the company's CRM, exposed by the CRM as a REST service and accessed through the corporate LAN.

The corporate LAN is separated from the public Internet by a firewall.
Model in ArchiMate the service provided by the company and its infrastructure.

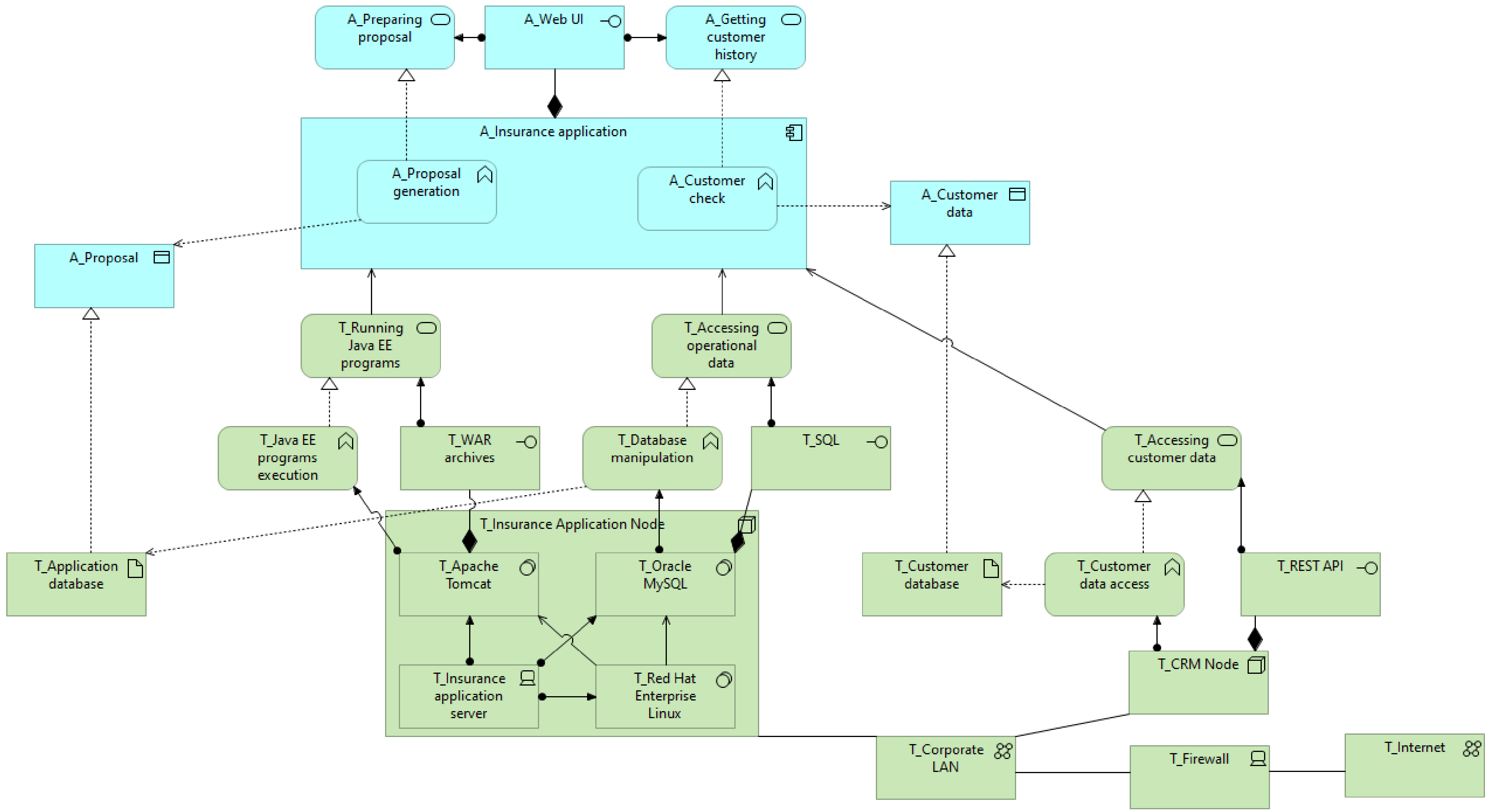
Exercise 2 – Motivation



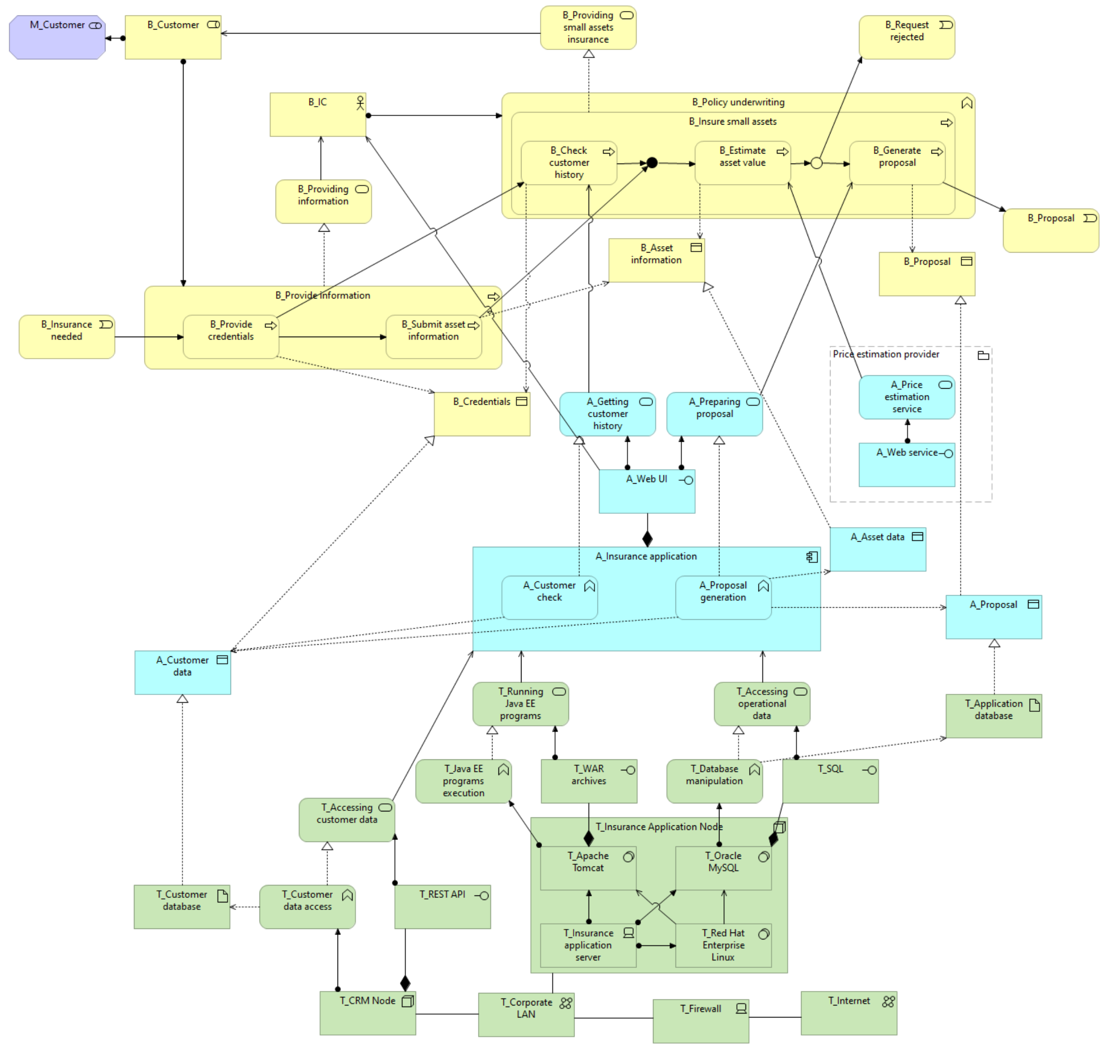
Exercise 2 – Business and Application layers



Exercise 2 – New infrastructure



Exercise 2 – Complete model



Exercise 3 – TEL

TEL is a telephony company interested in improving its customer experience. Currently, billing details are only accessible internally by TEL staff.

Data about telephony usage is stored on a Linux server running IBM DB2 UDB database, and it can only be accessed through SQL queries. Conversely, billing information is stored and handled by a legacy transactional CICS application running on an IBM mainframe.

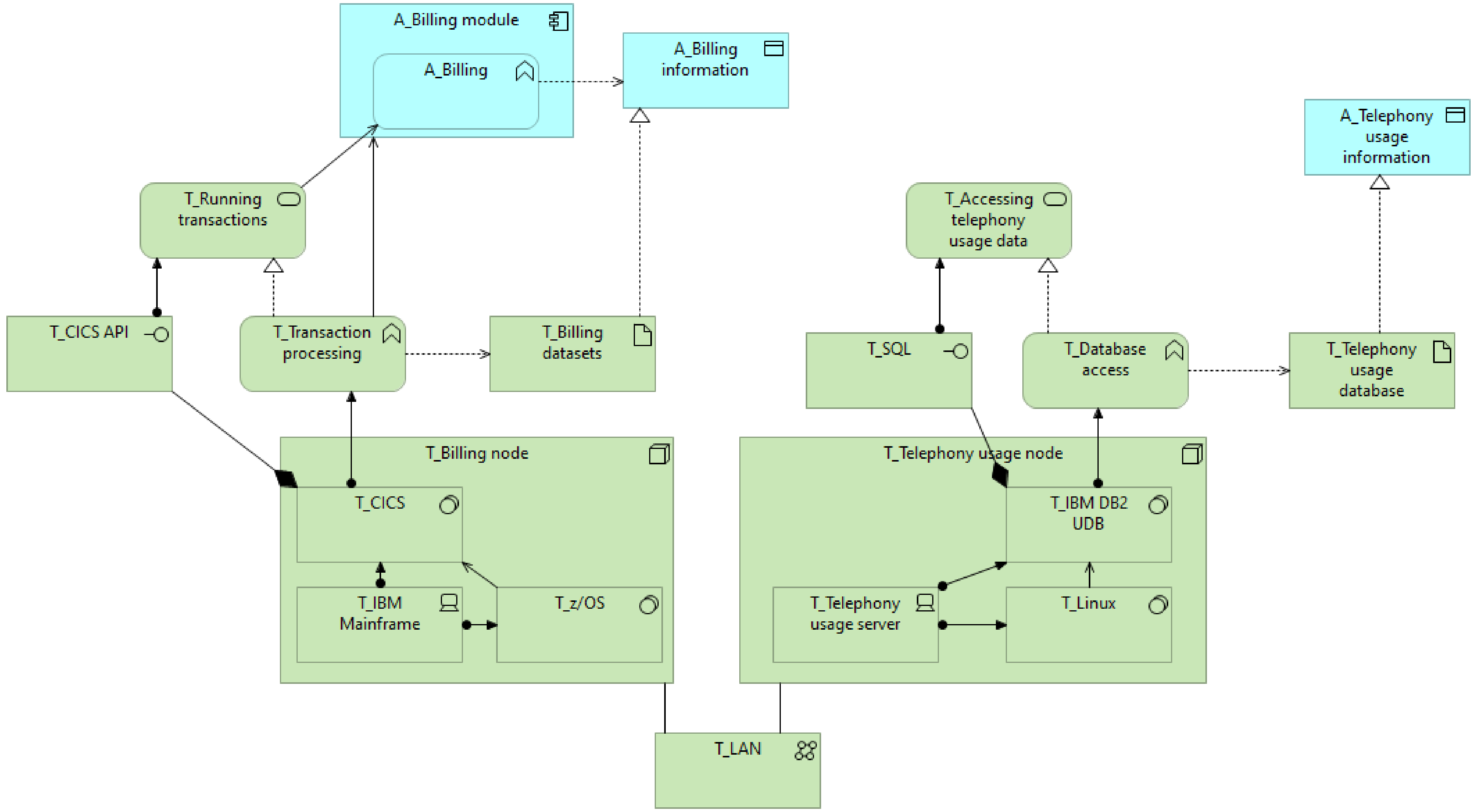
To improve its customer experience, TEL wants to develop a new web portal that can provide real-time billing details to users. In particular, the new portal will provide two main functionalities: inspect billing information and inspect usage information. The process enabled by the new portal will be organized as follows: the user logs in the portal, then he can select an item to inspect from the menu, and finally he can view the billing details in a dedicated page.

The new portal will be built for Microsoft SharePoint Online PaaS cloud service.

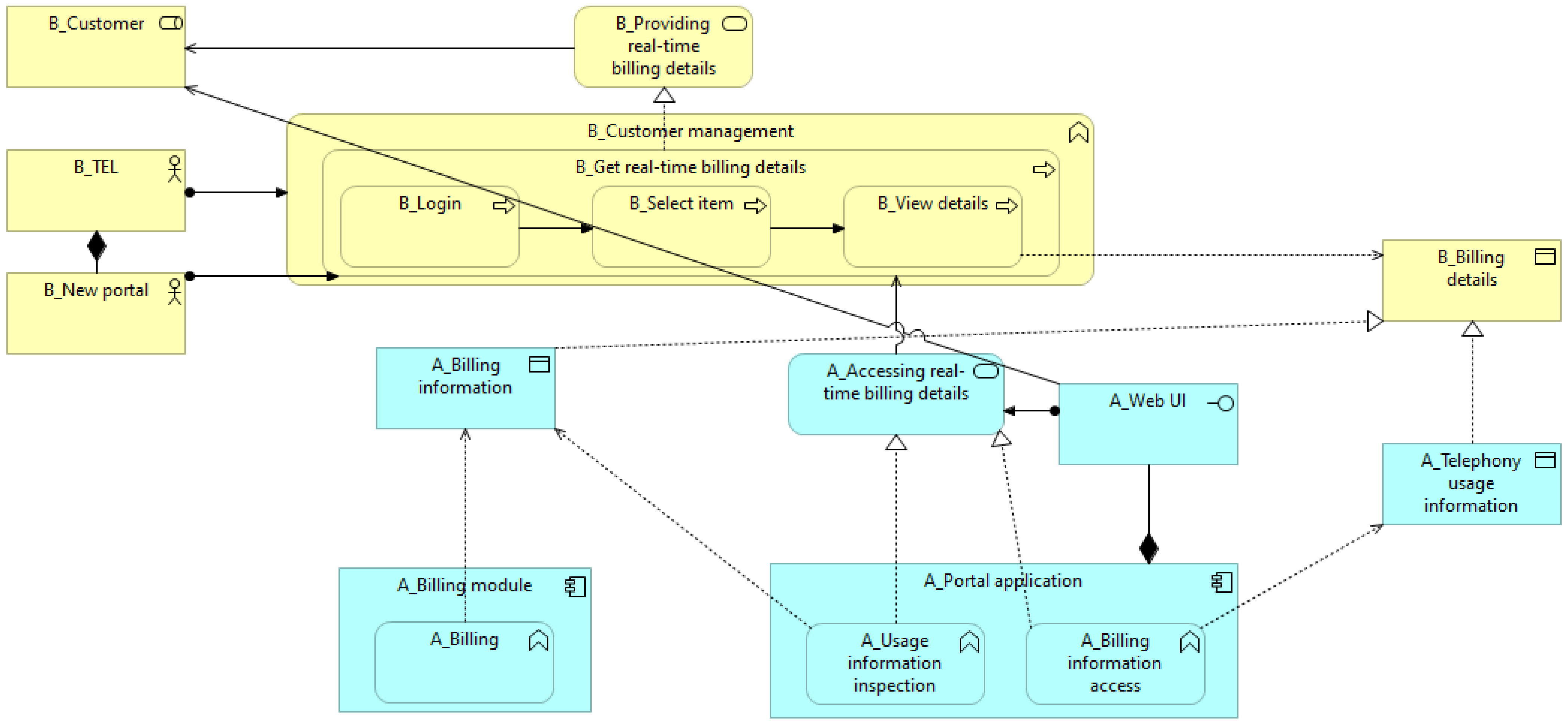
To access data from the existing infrastructure, a new node running Microsoft BizTalk Server 2020 middleware will also be introduced. BizTalk Server will offer a gateway service, making CICS applications and relational databases accessible through a standard REST interface.

An ArchiMate model of the existing system is enclosed in the next slide. Extend the model by adding the elements required for the new service.

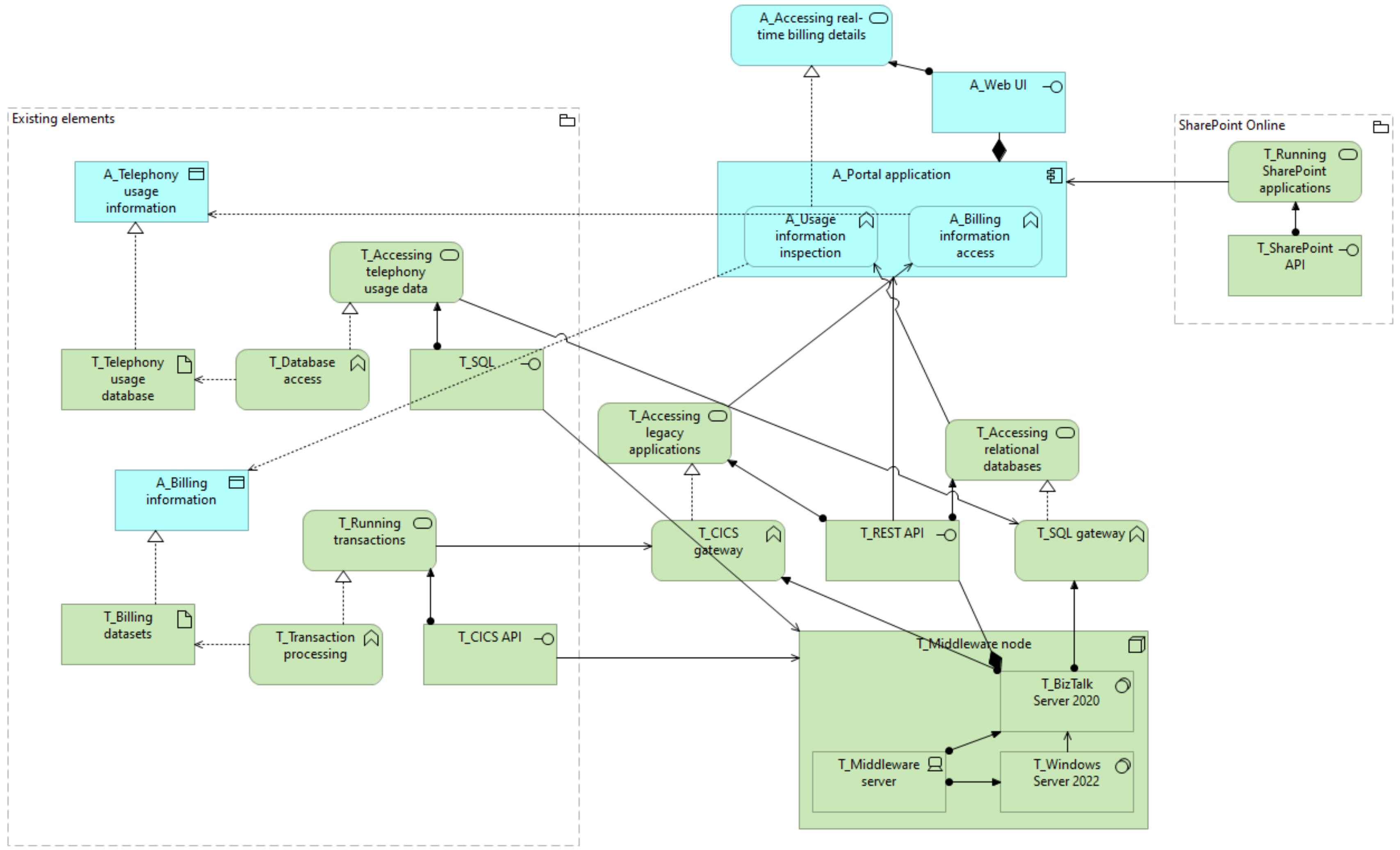
Exercise 3 – Existing infrastructure



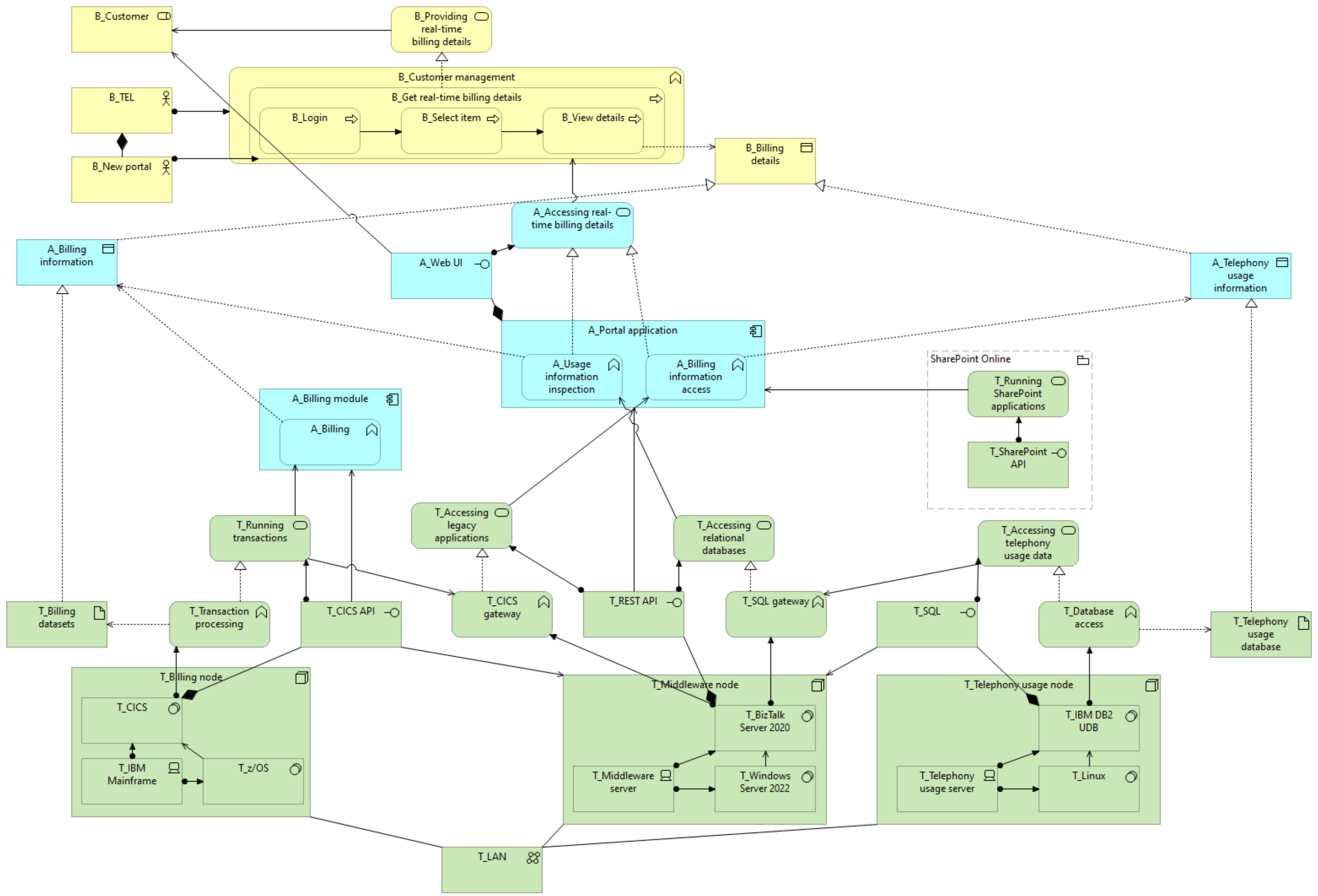
Exercise 3 – Business and Application layers


B_Customer
B_Providing real-time billing details
B_Customer management
B_TEL
B_Get real-time billing details
B_Login
B_Select item
B_View details
B_Billing details
B_New portal
A_Billing information
A_Accessing real-time billing details
A_Web UI
A_Telephony usage information
A_Billing module
A_Billing
A_Portal application
A_Usage information inspection
A_Billing information access

Exercise 3 – New infrastructure



Exercise 3 – Complete model



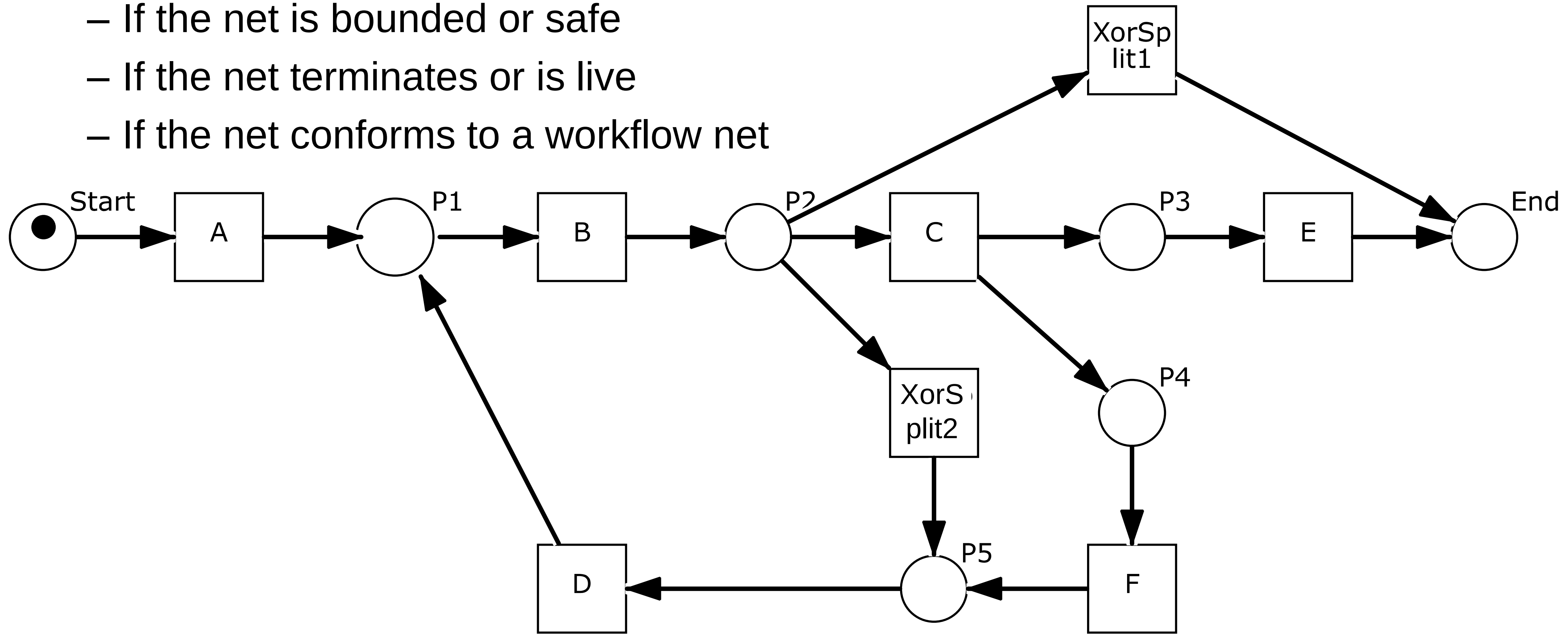
02291 System Integration

Behavioral Models with Petri Nets: Solutions to exercises

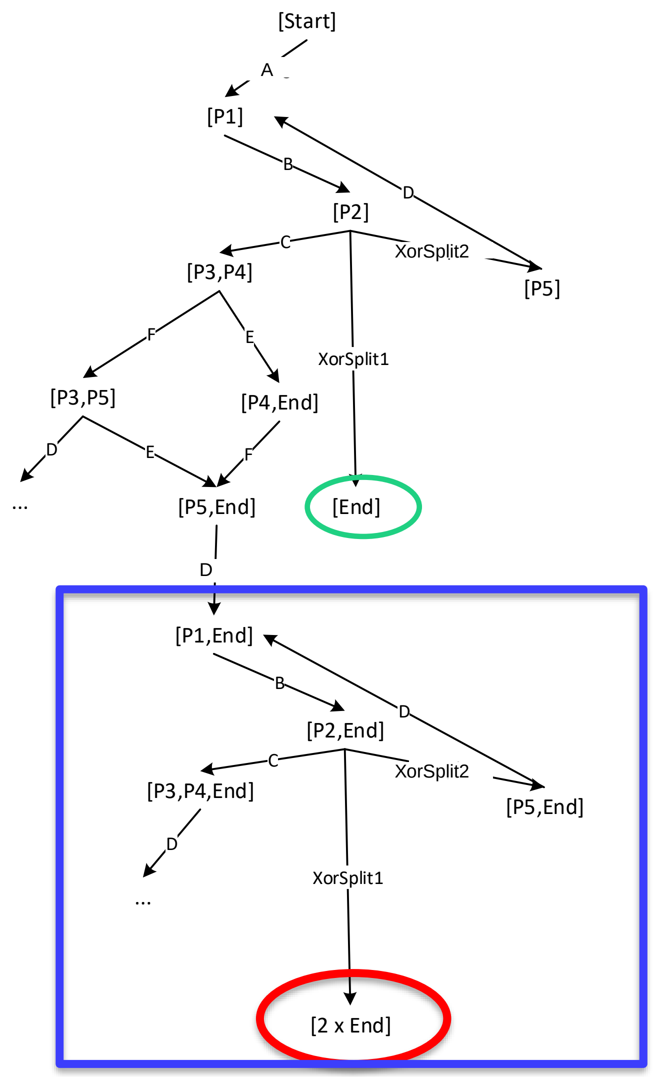
© Giovanni Meroni

Exercise 1

- Given the following Petri Net, compute:
  - P, T, F sets
  - The reachability graph
  - If the net is bounded or safe
  - If the net terminates or is live
  - If the net conforms to a workflow net



Exercise 1 – Reachability graph



- P = {Start, P1, P2, P3, P4, P5, End}
- T = {A, B, C, D, E, F, XorSplit1, XorSplit2}
- F = {(Start,A), (A,P1), (P1,B), (B,P2), (P2,XorSplit1), (P2,C), (P2,XorSplit2), (XorSplit1,End), (C,P3), (C,P4), (P3,E), (E,End), (P4,F), (F,P5), (P5,D), (D,P1), (XorSplit2,P5)}

Infinite reachability graph (tokens always grow)

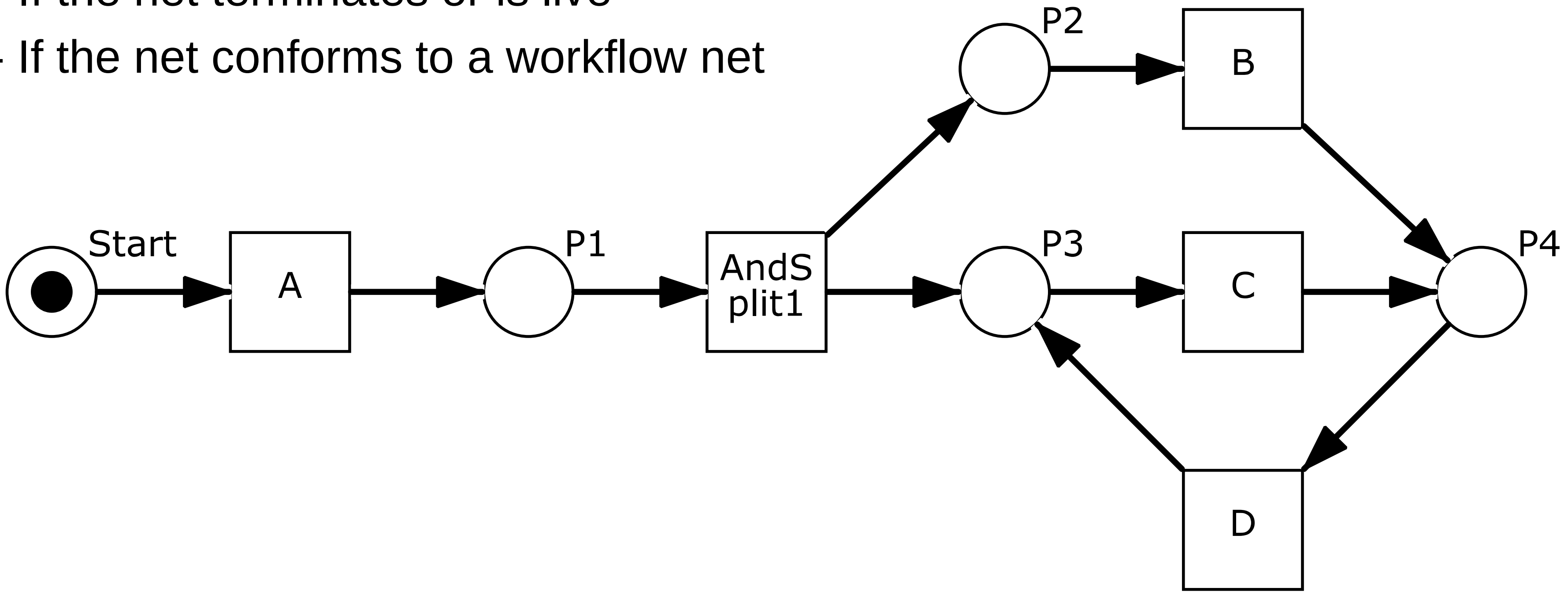
Not safe (not 1-bounded)

Not live (there are final markings)

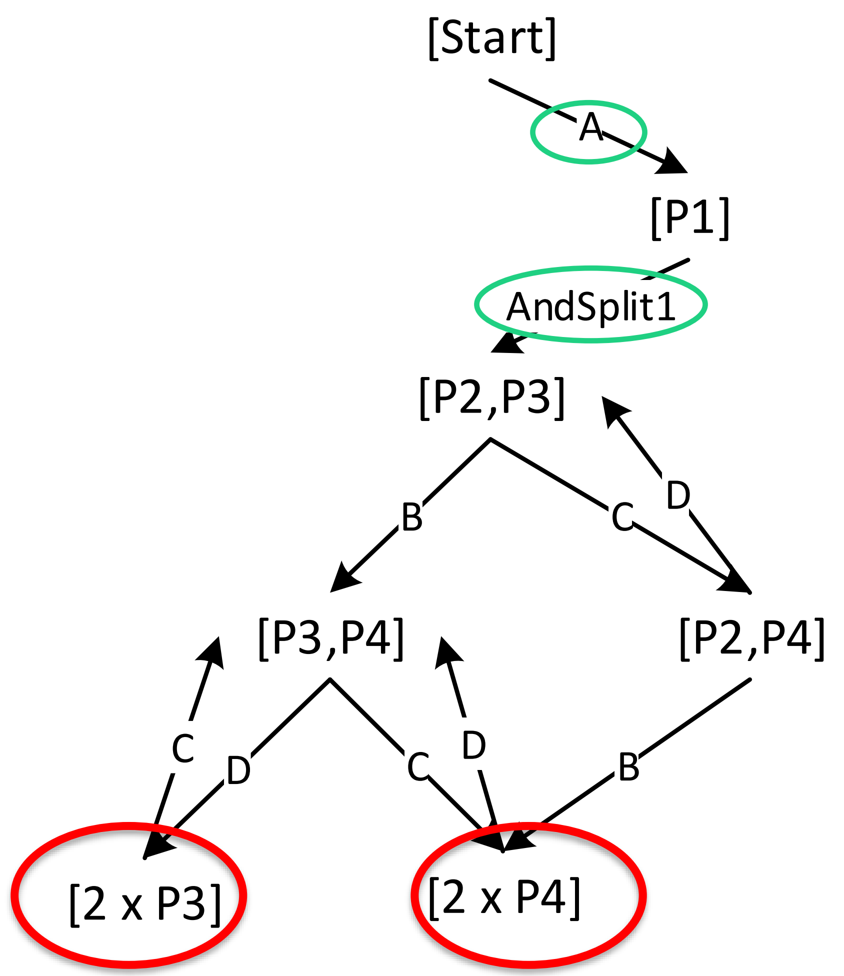
Net conforms to a workflow net: only one input (Start) and one output (End) place, every node is in a path from Start to End

Exercise 2

- Given the following Petri Net, compute:
  - P, T, F sets
  - The reachability graph
  - If the net is bounded or safe
  - If the net terminates or is live
  - If the net conforms to a workflow net



Exercise 2 – Reachability graph



- P = {Start, P1, P2, P3, P4}
- T = {A, AndSplit1, B, C, D}
- F = {(Start,A), (A,P1), (P1,AndSplit1), (AndSplit1,P2), (AndSplit1,P3), (P2,B) (B,P4), (P3,C), (C,P4), (P4,D), (D, P3)}

Not safe (2-bounded), cannot become sound

Not live (we cannot fire A or AndSplit1 more than once)

No final place, hence not terminating (and not conforming to a workflow net)!

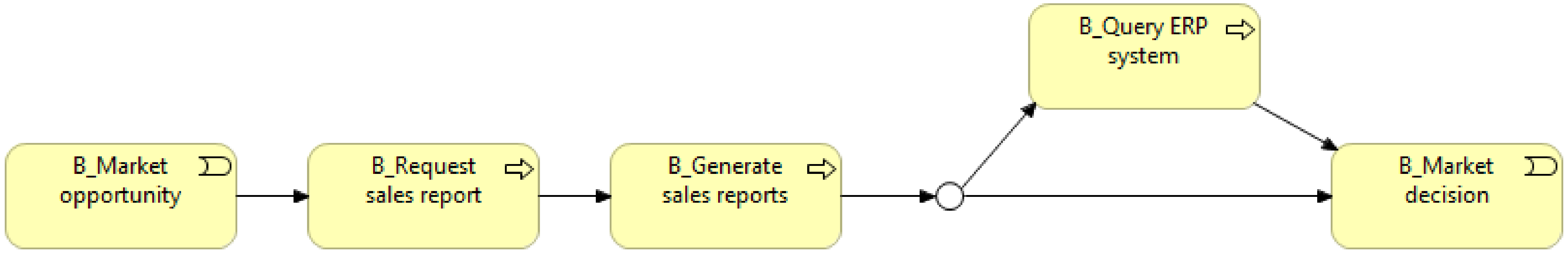
Exercise 3

Speedy is a delivery company that wants to create a new reporting system for the top management. After inspecting the sales reports, the top management may also need to query the existing ERP system, based on Oracle Fusion, to get detailed sales and HR information.

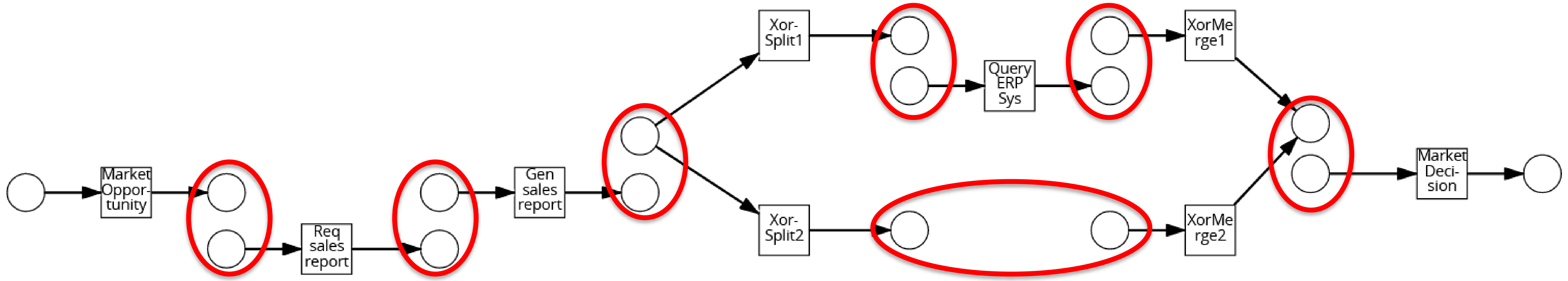
An ArchiMate model representing the process is enclosed below.

Starting from this model, create an equivalent Petri Net and try to answer the following questions:

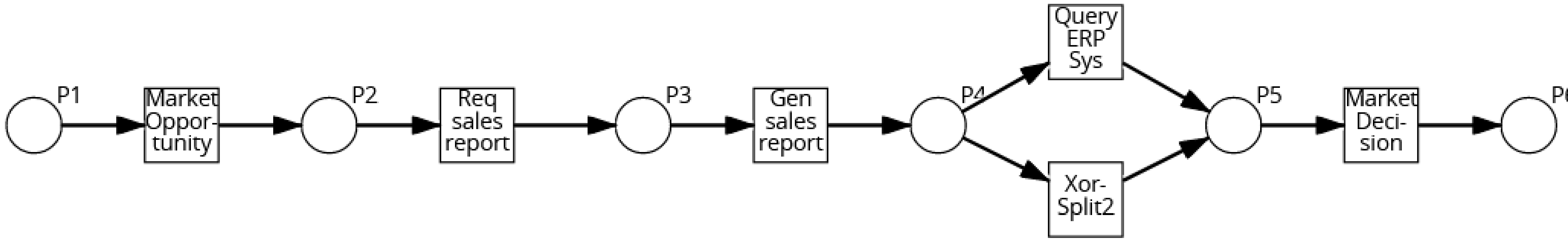
- What is the reachability graph of the net?
- Is the net sound?



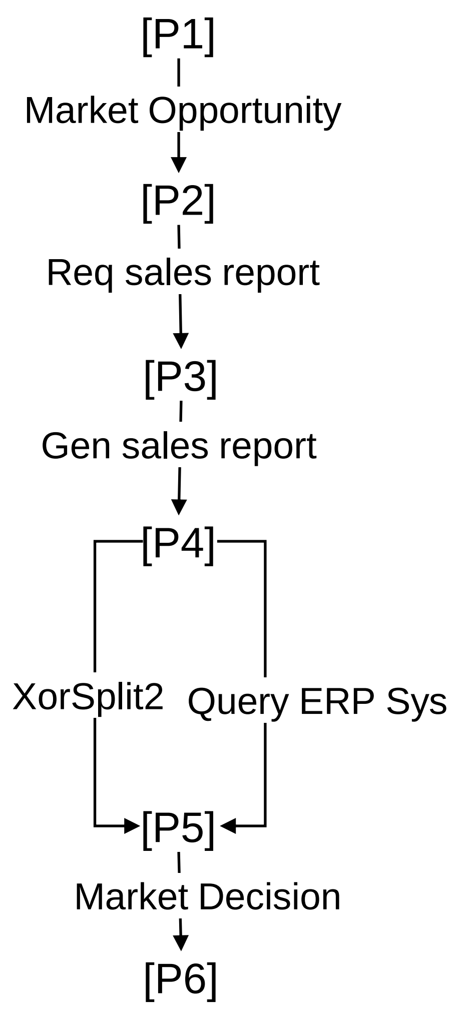
Exercise 3 – Petri Net



Exercise 3 – Optimized Petri Net



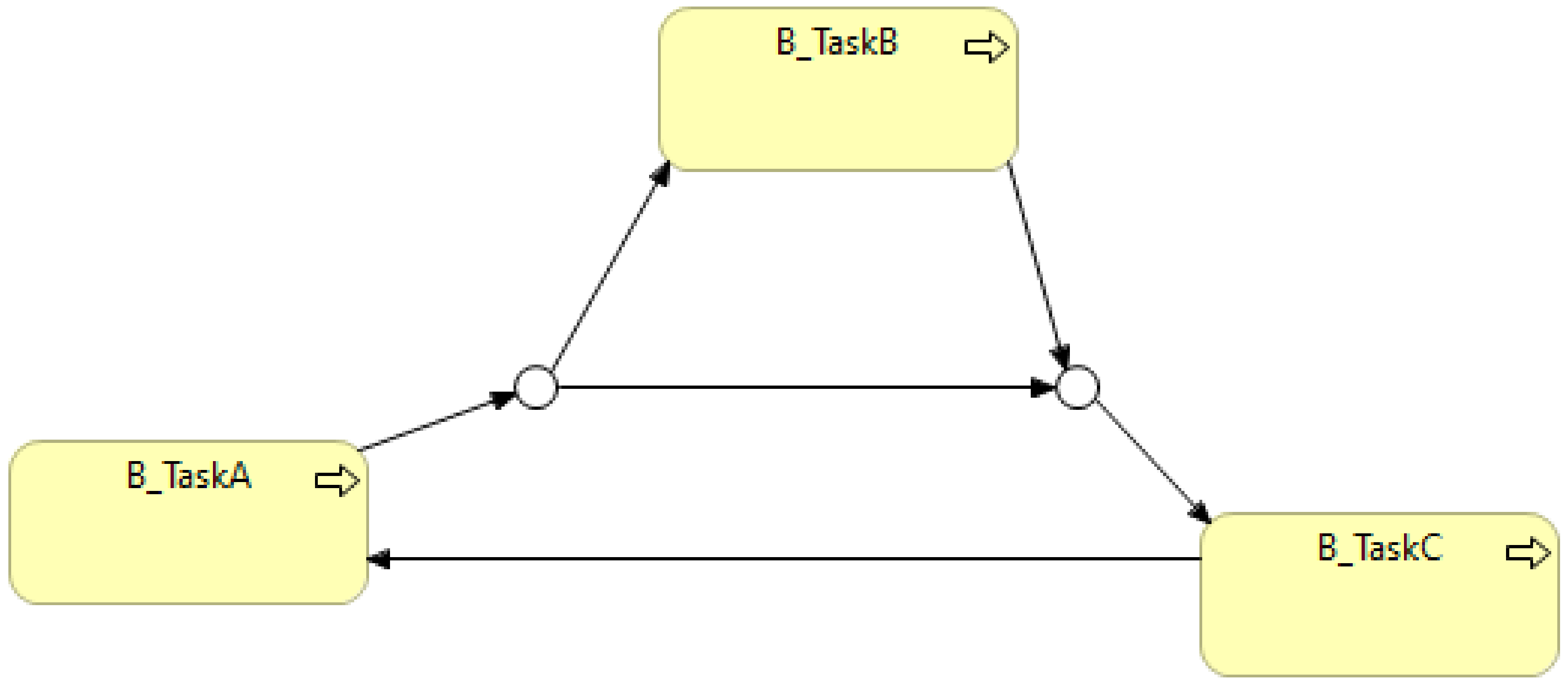
Exercise 3 – Reachability graph



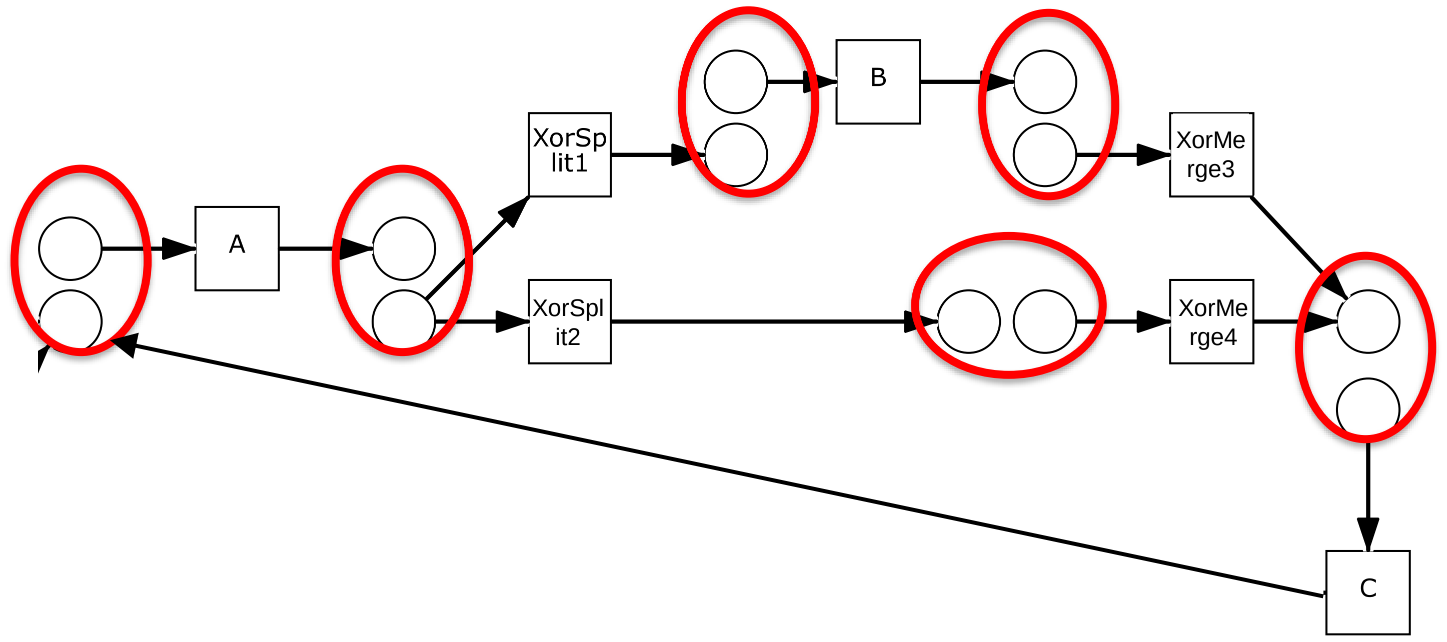
Net conforms to a workflow net: only one input (P1) and one output (P6) place, every node is in a path from P1 to P6

Exercise 4

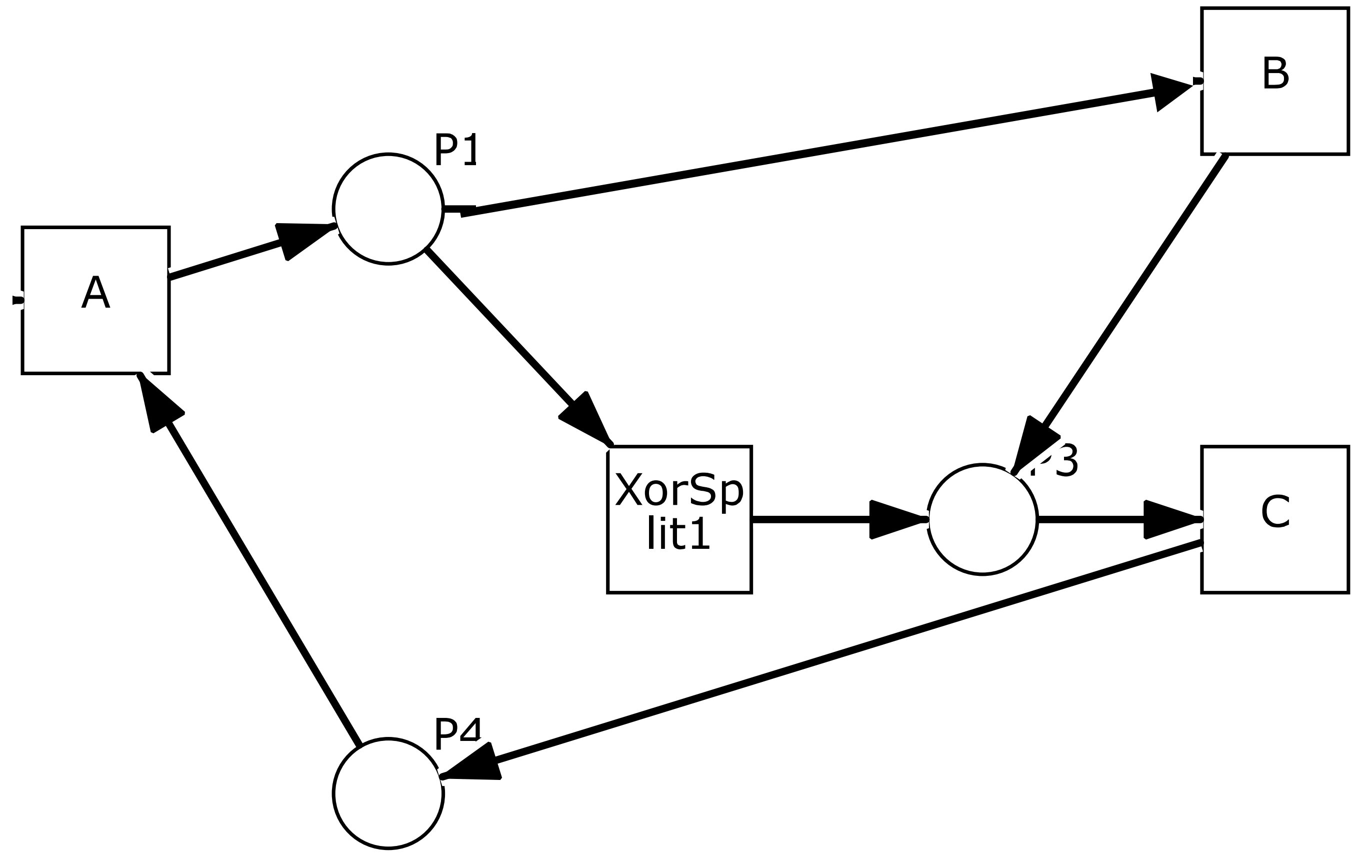
- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.



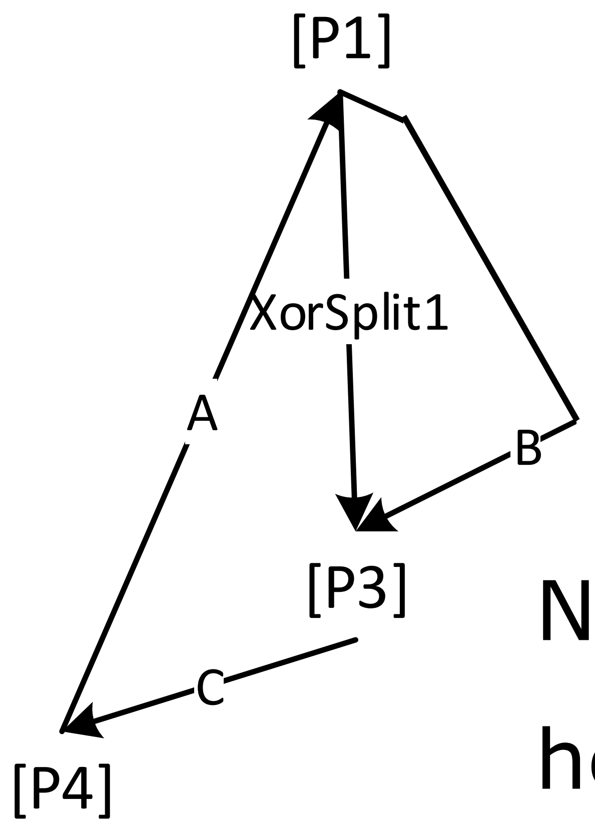
Exercise 4 – Petri Net



Exercise 4 – Optimized Petri Net

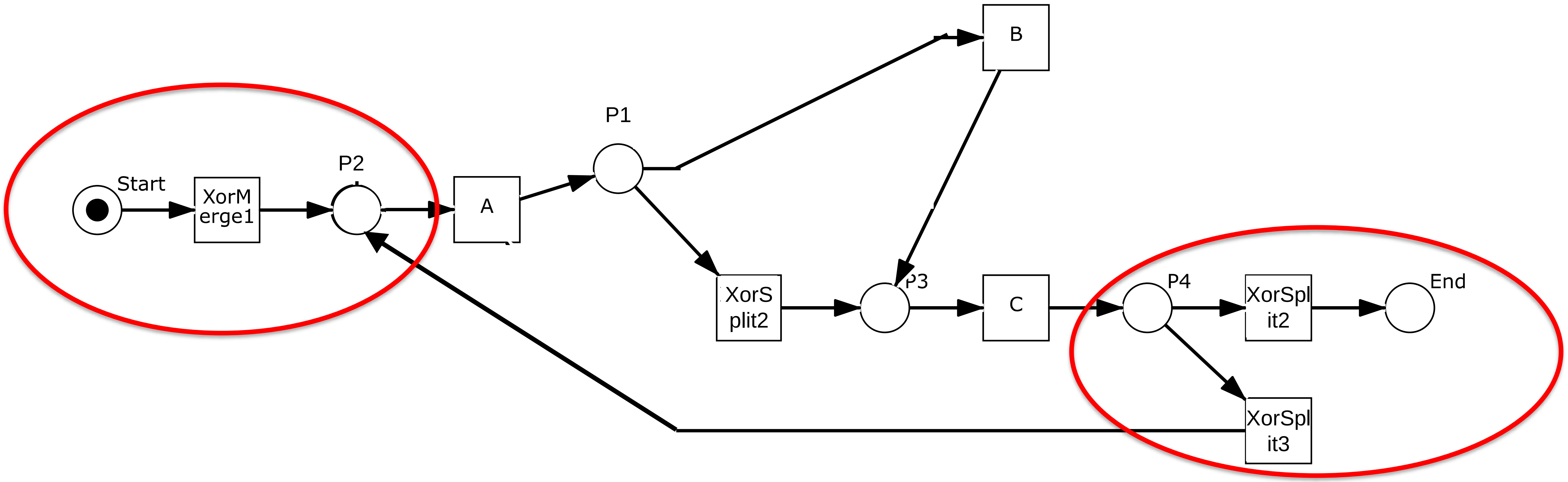


Exercise 4 – Reachability graph

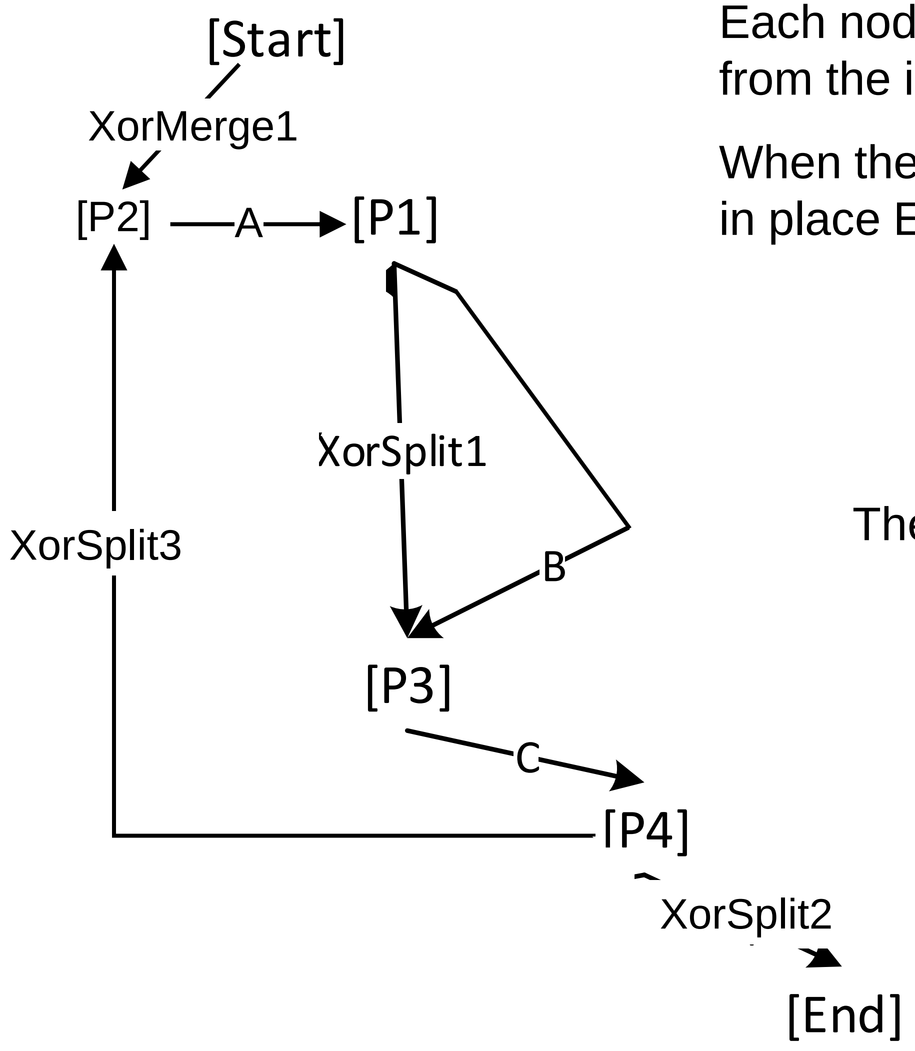


No initial and no final place, hence not sound!

Exercise 4 – Repaired process



Exercise 4 – Reachability graph of repaired process



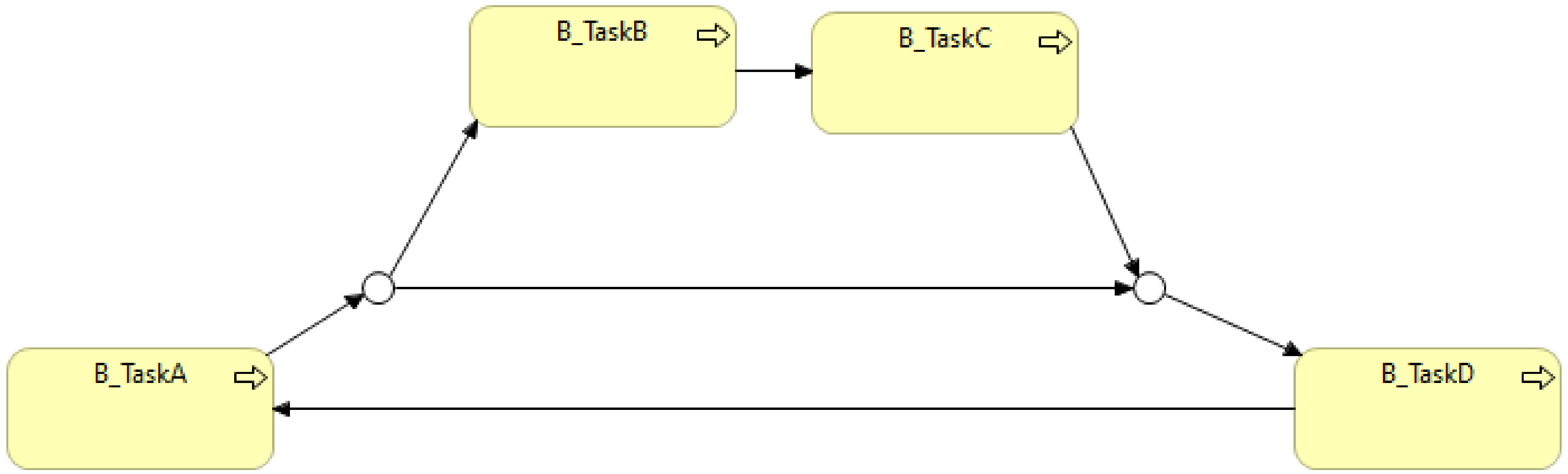
Each node in the process model is on a path from the initial node to the final node

When the procedure terminates, there is a token in place End and all the other places are empty

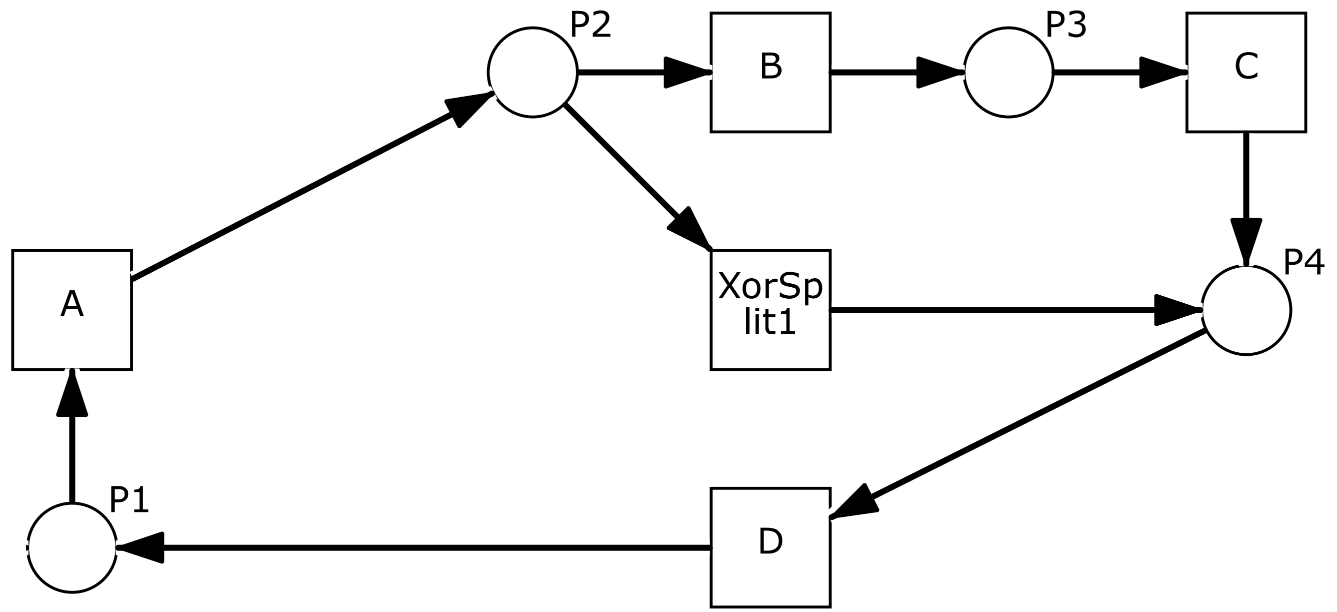
The process is now sound!

Exercise 5

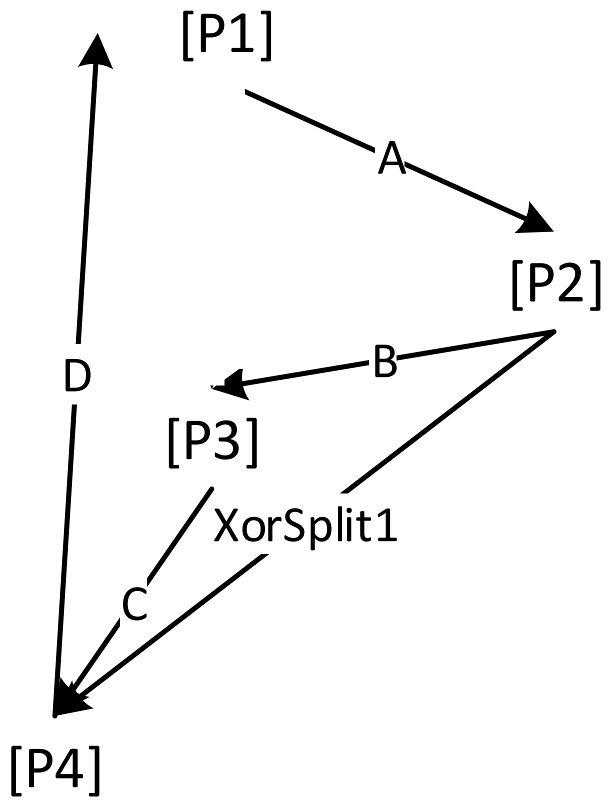
- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.



Exercise 5 – Optimized Petri Net



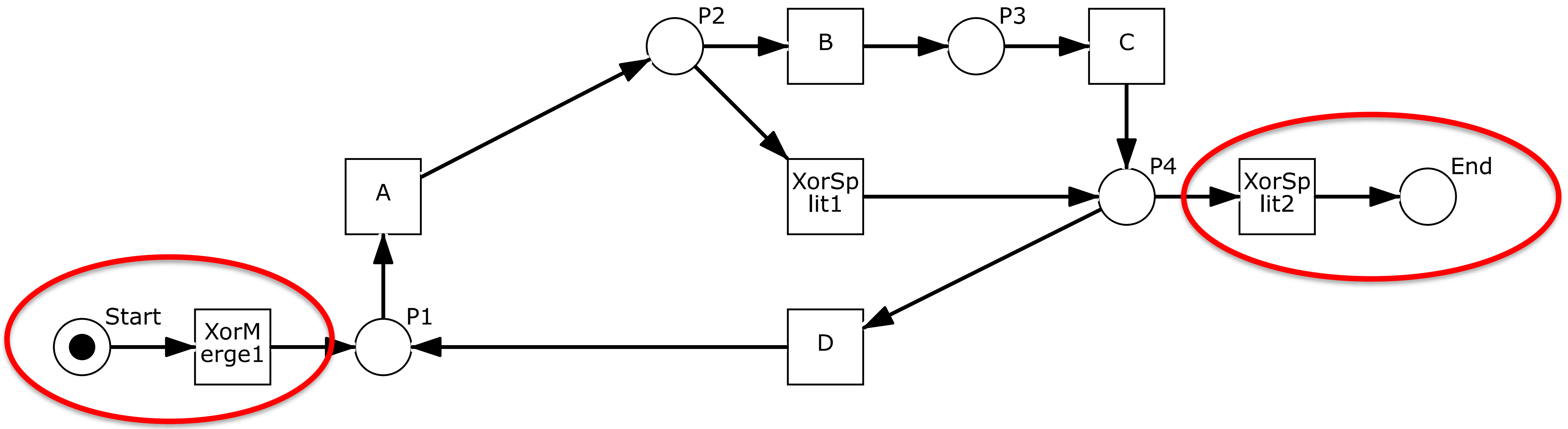
Exercise 5 – Reachability graph



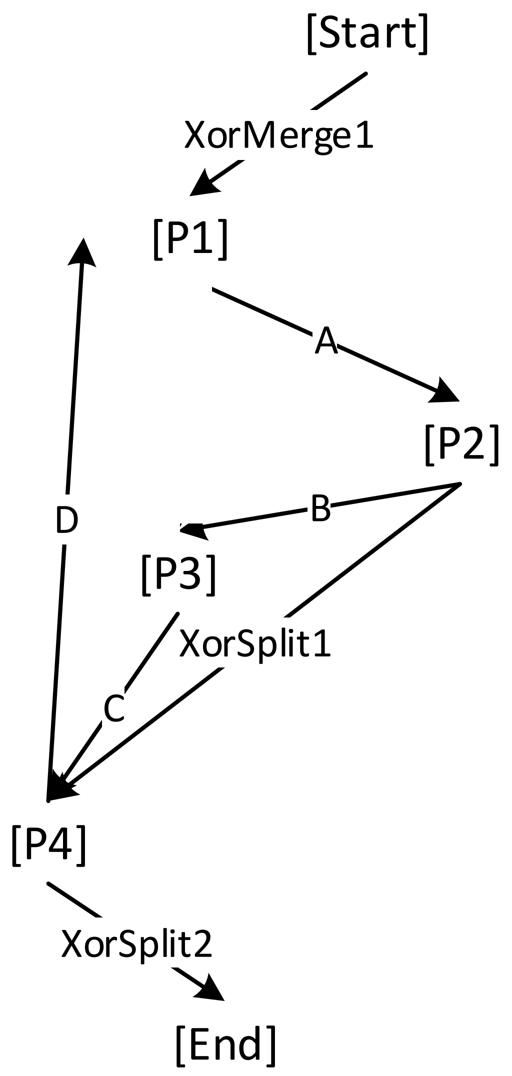
No initial and no final place,

hence not sound!

Exercise 5 – Repaired process



Exercise 5 – Reachability graph of repaired process



Each node in the process model is on a path from the initial node to the final node

When the procedure terminates, there is a token in place End and all the other places are empty

The process is now sound!

Declarative Process Modelling - Solutions Week 7

Hugo A. López

Exercises

Model the following patterns in a DCR-graph (use the simulator to verify if your model is right):

1. **Direct Precedence of a Task.** “Every time B occurs, it should be directly preceded by A.” If B occurs without a directly preceding A, the rule is violated. For instance, traces ⟨ACCAAC⟩ and ⟨ABCAAB⟩ comply to the rule, whereas ⟨ABACB⟩ violates the rule.

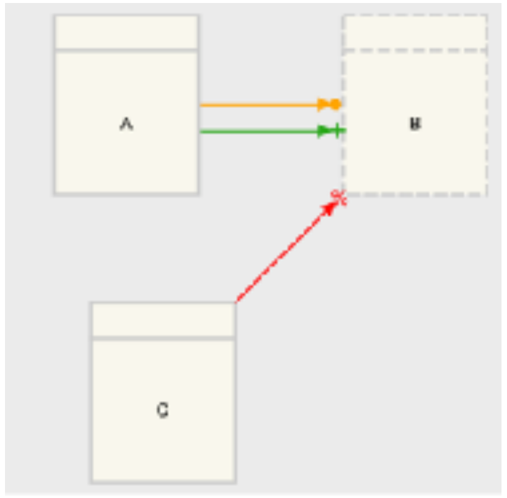
2. **Direct Precedence or Simultaneous Occurrences of Tasks.** “Task *A* must always be executed simultaneously or directly before task *B*.” (hint: consider A/B as non-atomic tasks)

3. **Bounded Existence of a Task:** Task A should be executed exactly k times.” If A occurs less than or more than k times, the rule is violated. For instance, for k = 2, the trace ⟨BCADBCAD⟩ complies to this rule and ⟨BCADBCAAD⟩ violates the rule.

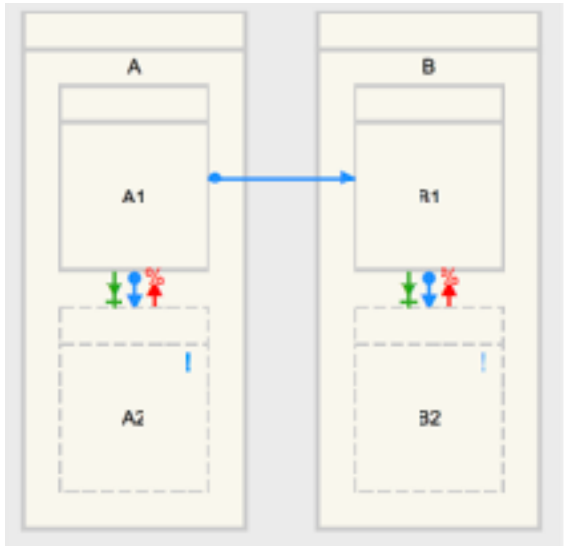
4. **Execution in Between.** “Task B should be performed not before task A has been executed, and not later than C.”

SOLUTIONS

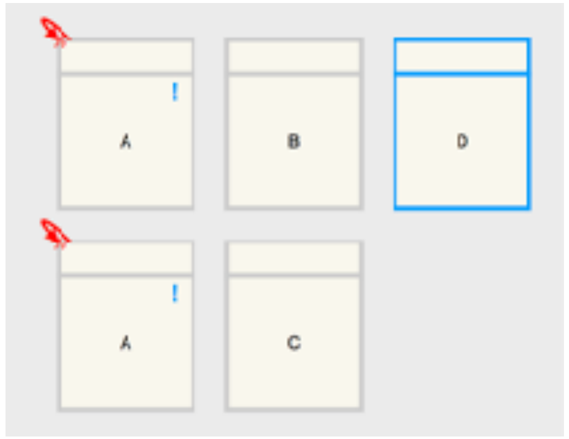
1.



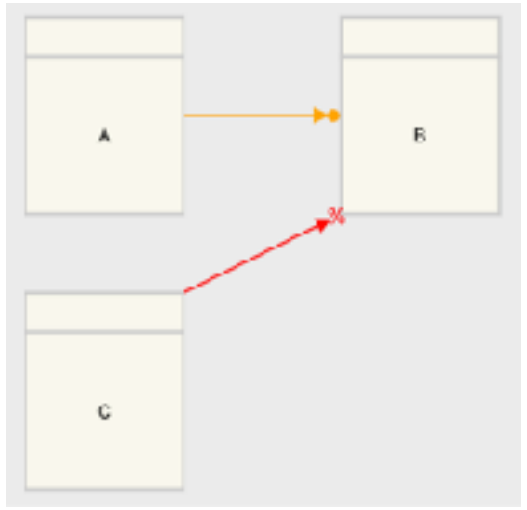
2.



3.

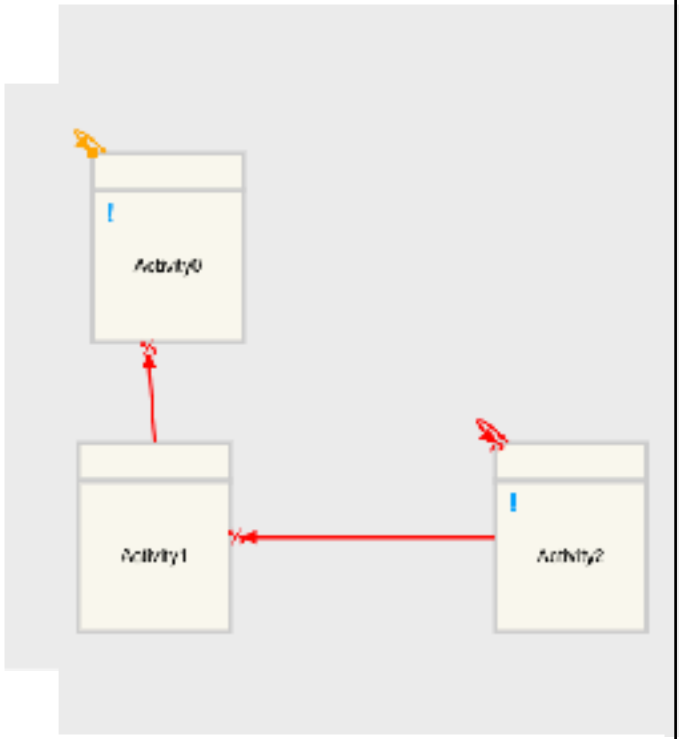


4.

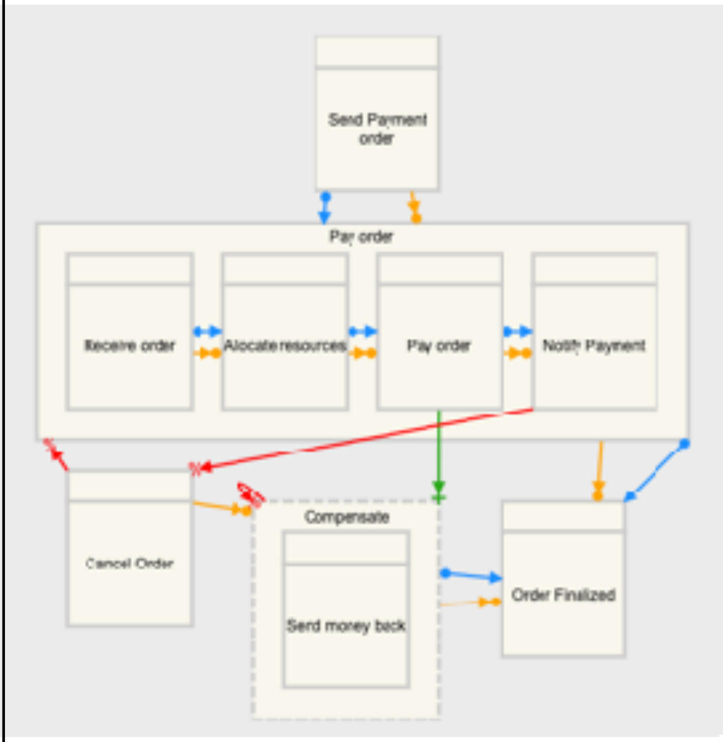


Exercises (Soundness)

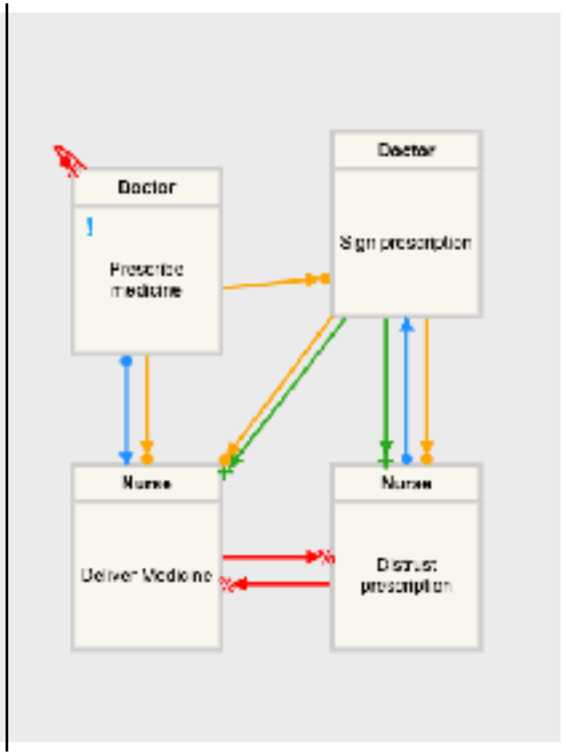
1. Discuss in pairs whether the following processes are sound:



A

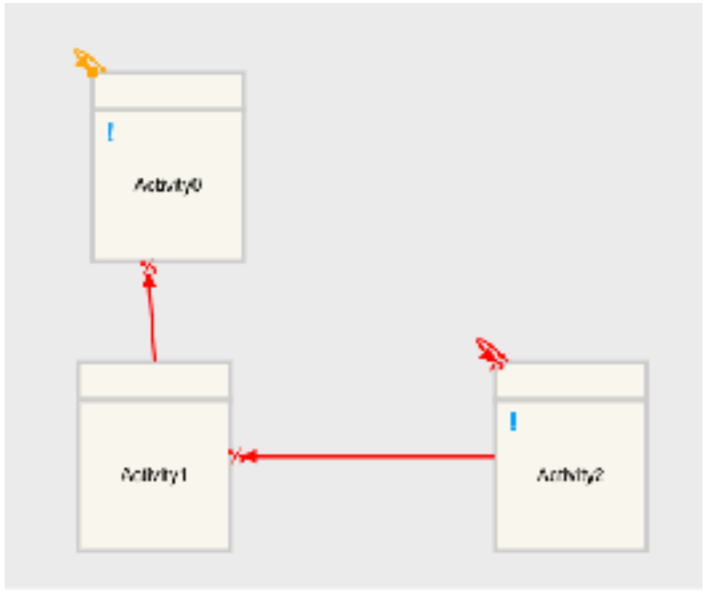


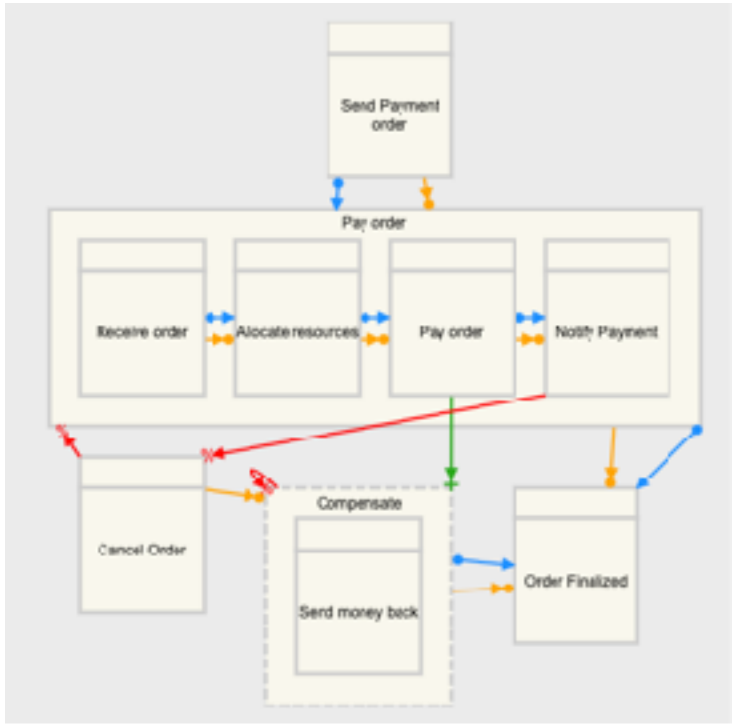
B

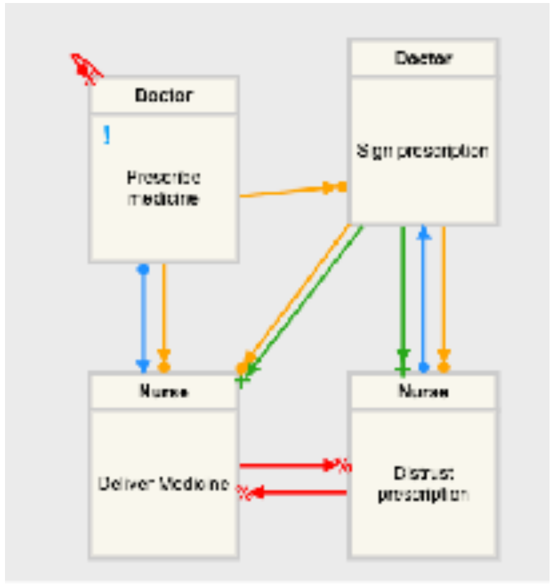


C

SOLUTIONS







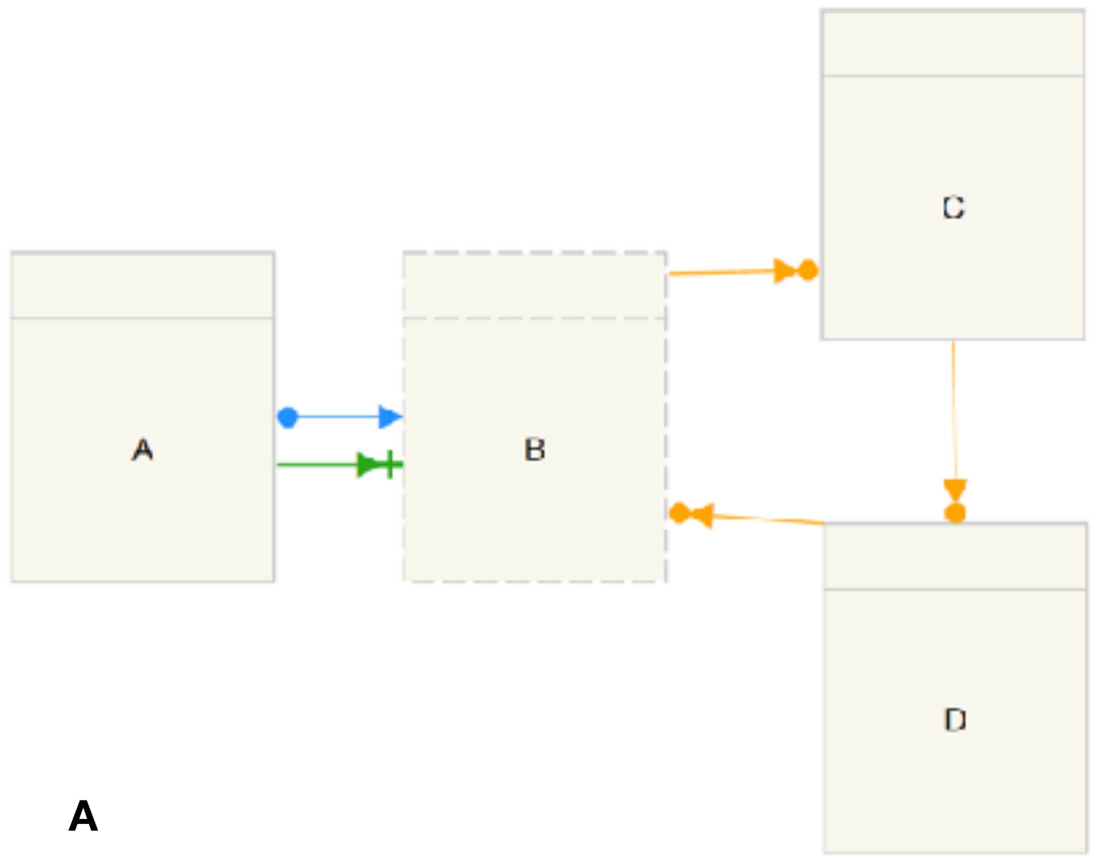
- Sound. The trace <*Activity1, Activity2*> is an accepting trace

- Sound. The empty trace is valid. Moreover, the execution of *Send Payment* order generates a **finite** chain of responses

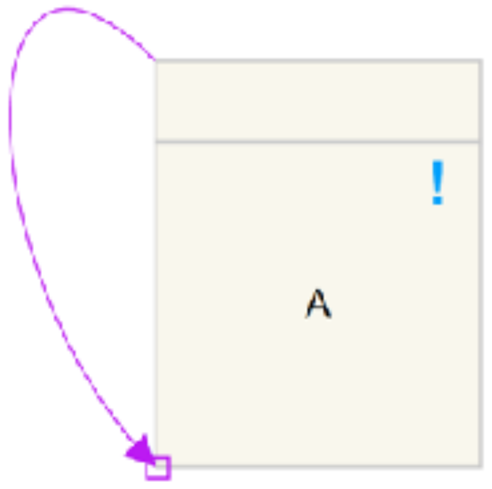
- Sound: The empty trace is not accepting, but executing <*Prescribe Medicine, Sign Prescription, Deliver Medicine*> will generate an accepting trace.

Exercise:

2.Argument whether the following processes are sound:

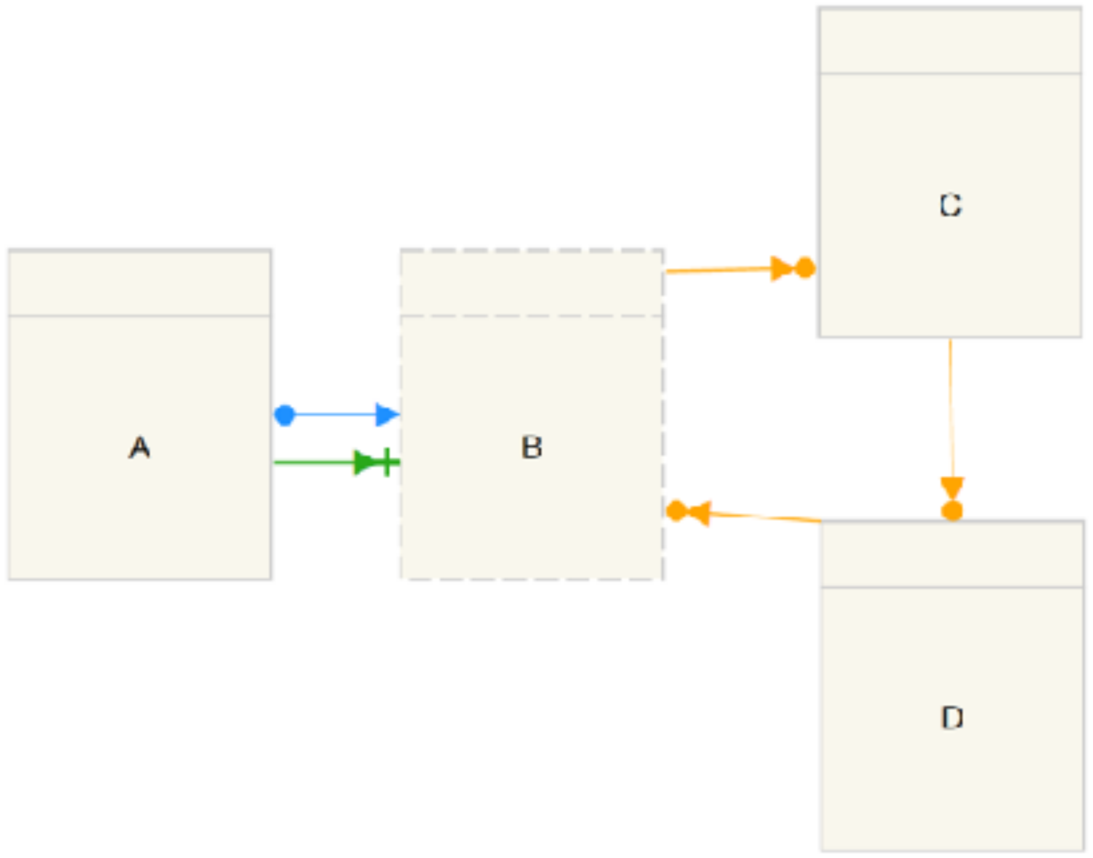


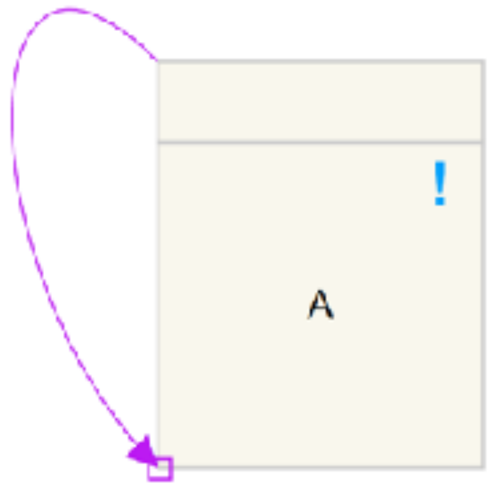
A



B

Solutions



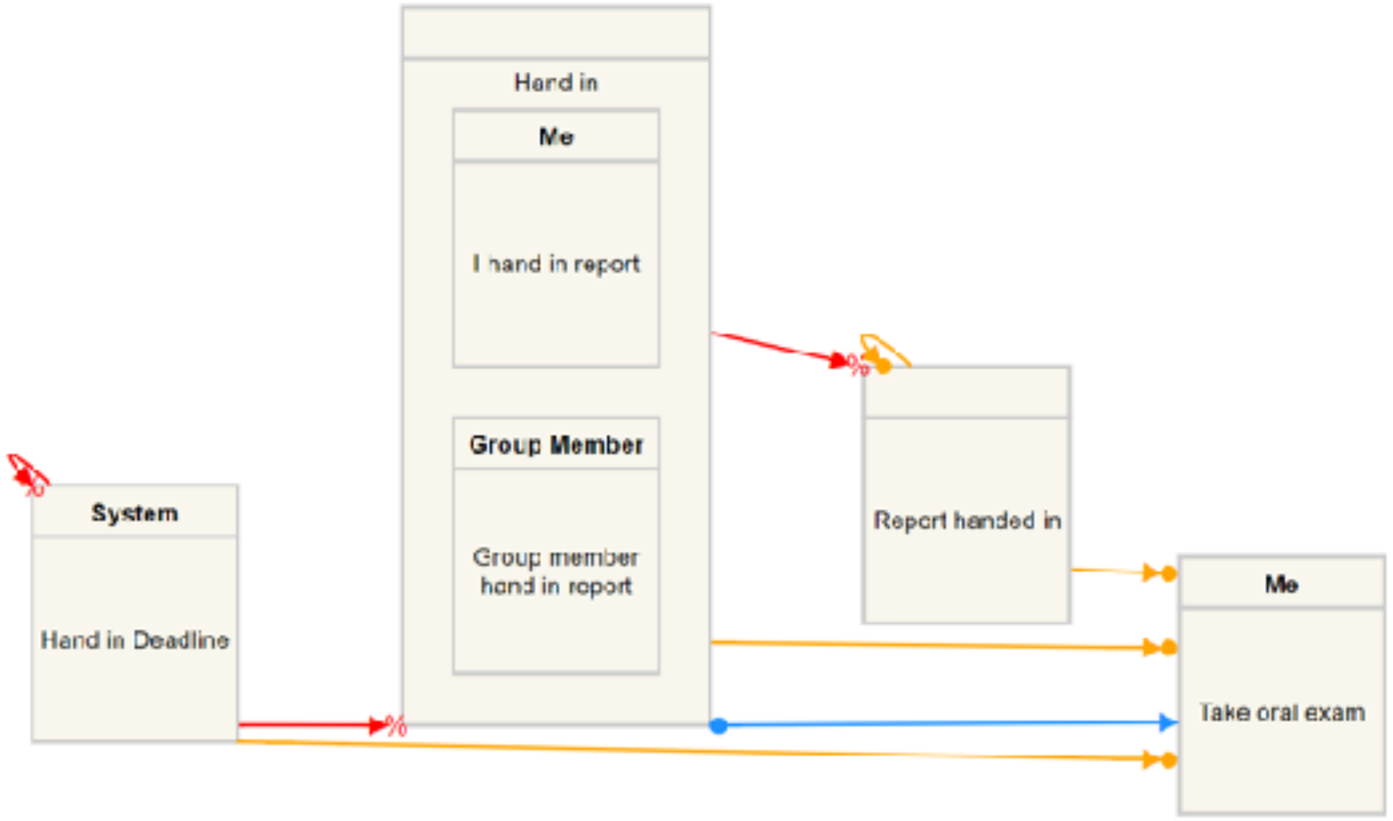


- Sound: if you execute *C* first, you can generate an accepting trace.
  But, if you were to start with *A*, you will generate a deadlocked process

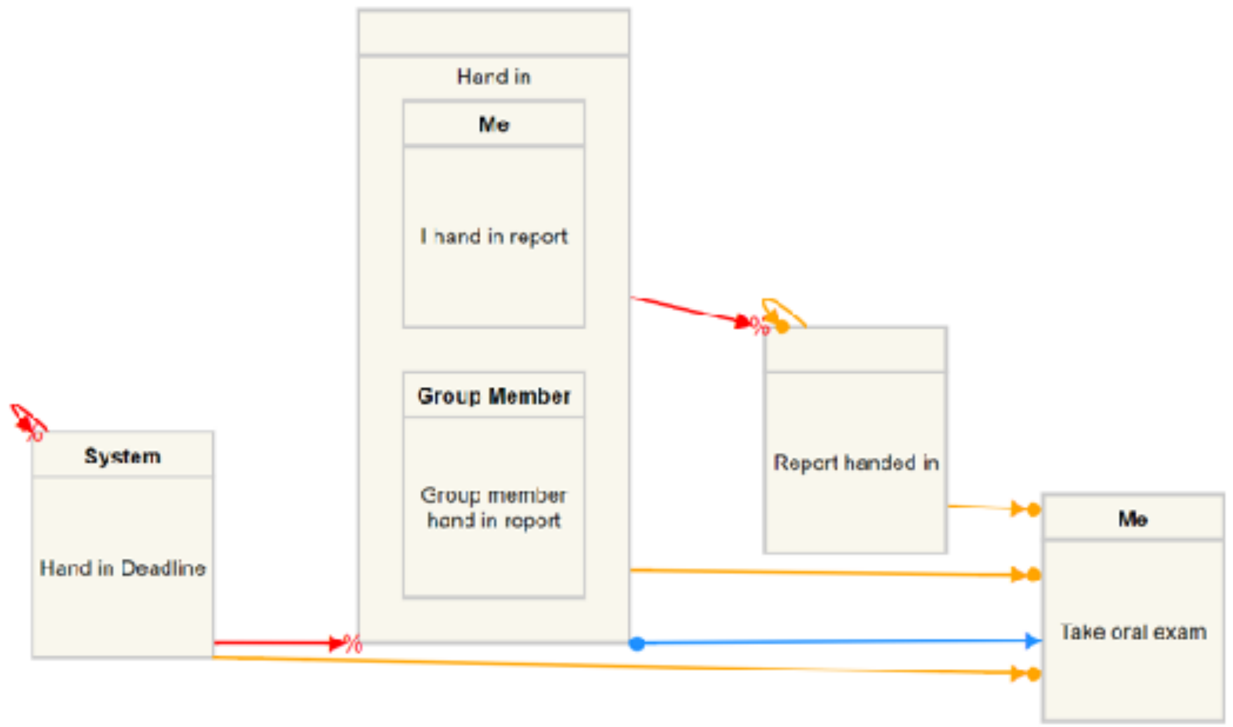
- Unsound: *A* can only be executed if the source of the milestone (*A* itself) is not pending, which is not the case.

Exercise

3. Argument whether this process is sound:



Solution



The process is sound: even if you have *Report handed in* with a cyclic dependency blocking *take oral exam*, this does not block the execution as any execution of the activities in the nesting *hand in* will exclude it.

02291 System Integration

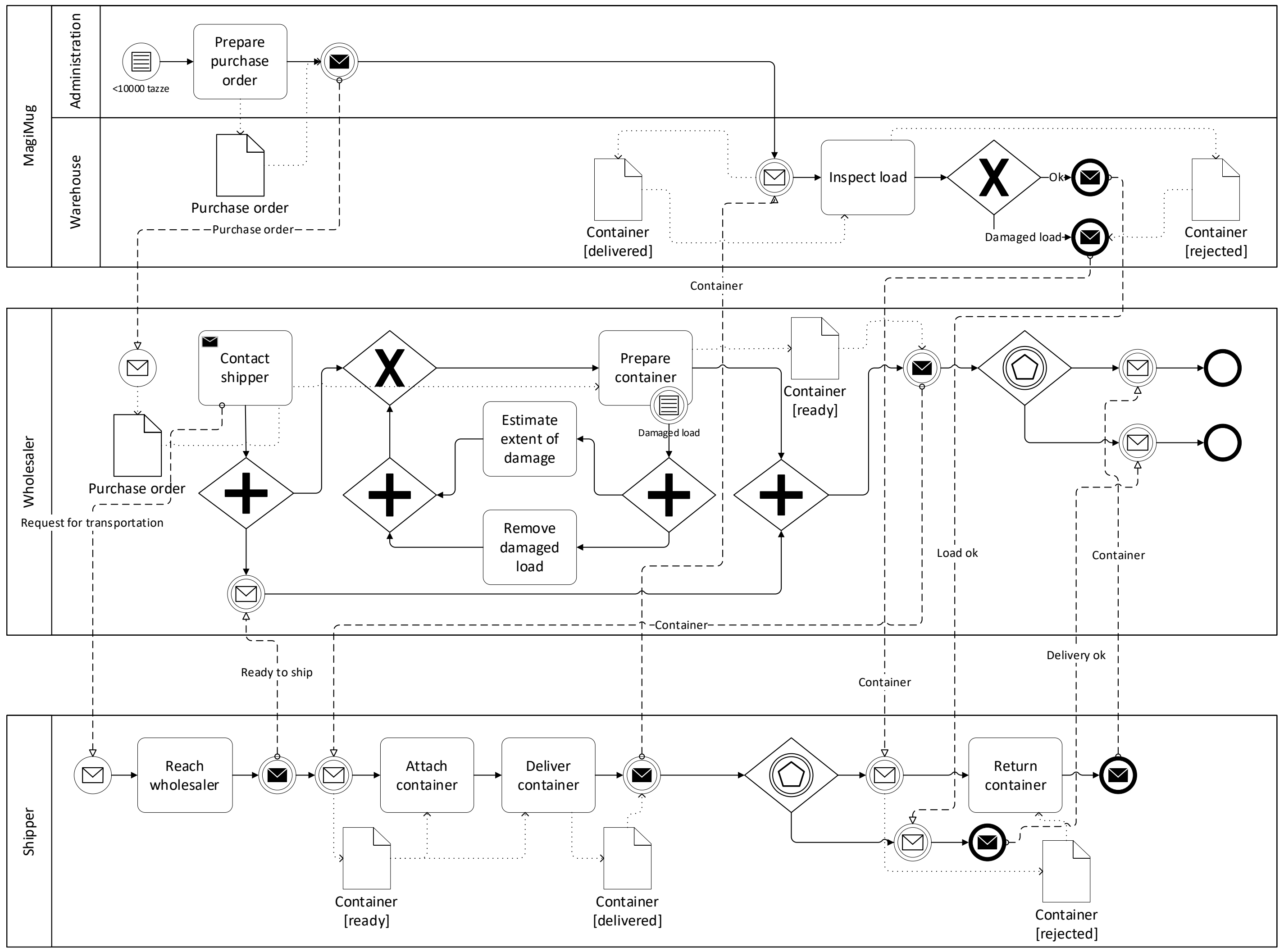
Introduction to BPMN: Solutions to exercises

Exercise 1 – MagiMug

MagiMug is a manufacturer of promotional mugs, which are sold to companies, universities, hotels, etc. In particular, MagiMug buys white ceramic mugs from a wholesaler and screen prints on them a picture provided by the customer.

When the number of cups in stock is less than 10000 units, MagiMug's administration prepares a purchase order and sends it to its wholesaler. Once the order has been received, the wholesaler contacts a shipper. Then, while waiting for the shipper to reach its premises, the wholesaler prepares a container to be shipped to MagiMug. If, while preparing the container, part of the load gets damaged, the activity is stopped, and the extent of the damage is estimated. At the same time, the damaged load is removed from the container. Once these two activities are complete, the container preparation is repeated. Once the container is ready and the shipper has reached the wholesaler's premises, the container is given to the shipper, who attaches it to his/her truck, and delivers it to MagiMug. Once the container is received, MagiMug's warehouse workers check the integrity of the load. If everything is in order, this is reported to the shipper, who in turn notifies the wholesaler of the success of the activity, and the process ends. If, on the other hand, part of the load is damaged, the shipper takes the container back to the wholesaler and the process ends."

Exercise 1 – BPMN collaboration diagram

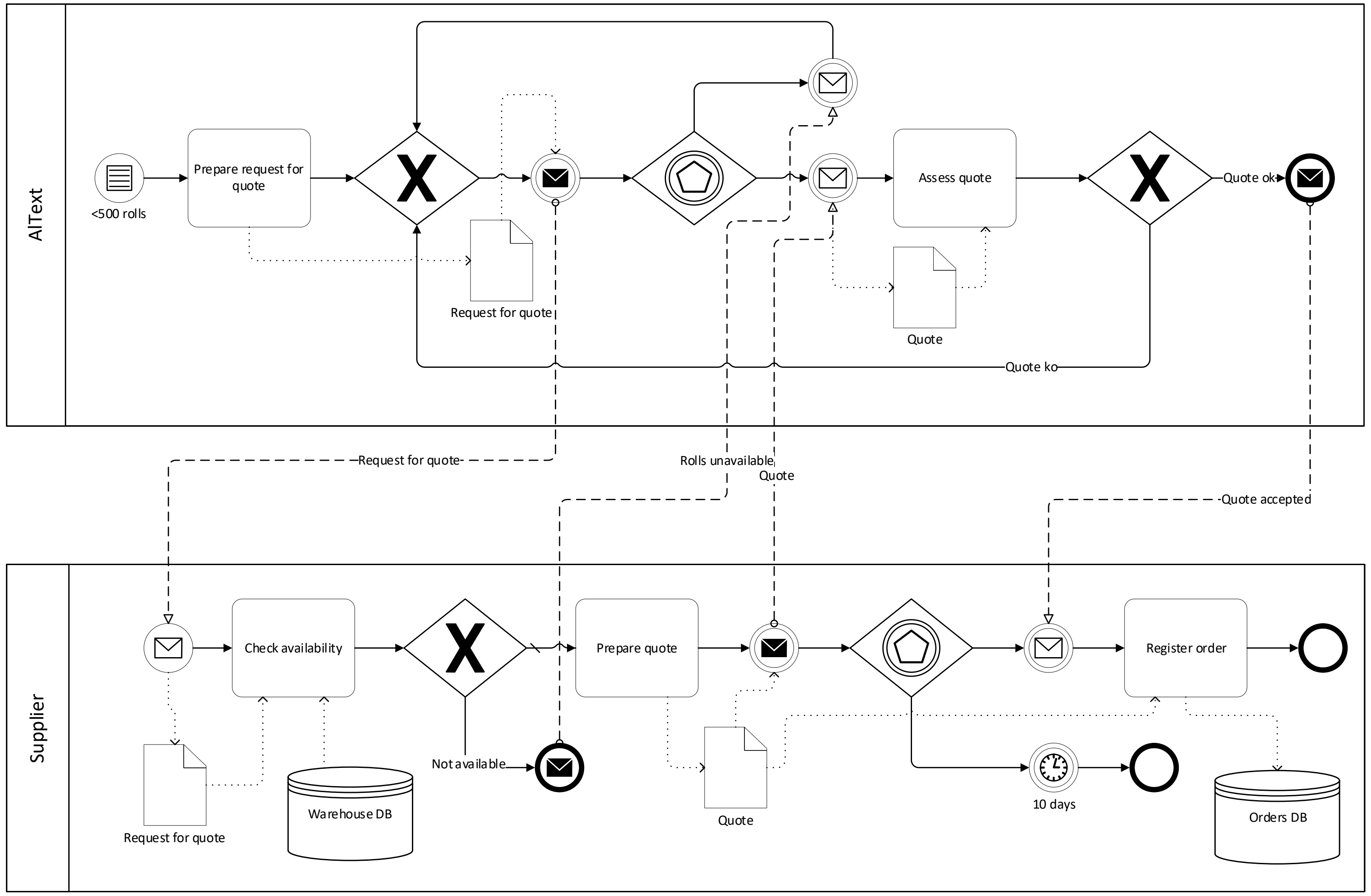


Exercise 2 – AlText

AlText is a publisher of printed academic textbooks and scientific journals.

When the paper rolls that AlText has in stock fall below 500 rolls, a request for a quote for 2000 rolls is prepared and sent to one of its supplier. The supplier, upon reception of the request, verifies if it has enough paper rolls in its warehouses. If the supplier does not have enough rolls, it terminates its process by informing AlText. AlText then sends the request for a quote again to another supplier. If a supplier has enough rolls available, it prepares a quote, which is sent to AlText. If, after assessing the quote, AlText is satisfied with it, AlText notifies the supplier that the quote has been accepted, the supplier registers the order, and the process ends on both sides. If AlText is not satisfied with the quote, it sends the request for a quote again to another supplier. If a supplier does not receive the notification within 10 days since the quote was sent, it ends the process.

Exercise 2 – Main process



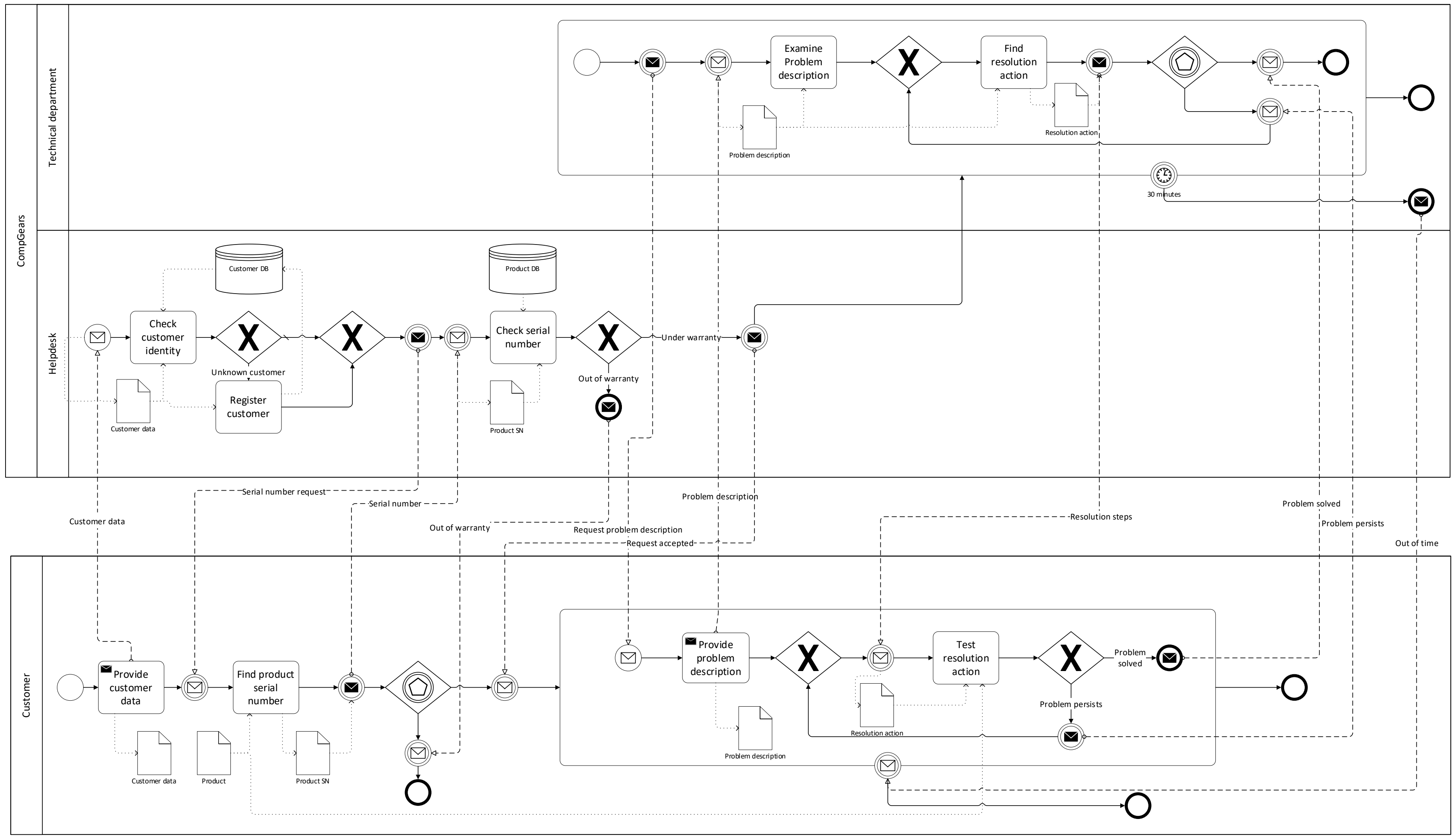
Exercise 3 - CompGears

CompGears, a consumer electronics company, wants to model its helpdesk's technical support process.

The process begins when a customer contacts the company's helpdesk with a technical support request, communicating his/her identity. First, the helpdesk operator checks whether the customer is already present in the system and, if not, (s)he registers the customer. Then, the helpdesk operator asks the customer to provide the serial number of the product. Once this information is received, the helpdesk operator checks if that product is still under warranty. If not, the helpdesk operator ends the process by notifying the customer that the product is out of warranty. Otherwise, (s)he notifies the user that the request has been accepted and forwards it to an operator from the company's in-house technical department.

First, the technical operator asks the customer to describe the problem. Once this information is received, (s)he examines the symptoms and then looks for a possible resolution action, which is provided to the customer. The customer then verifies whether that resolution action solves his problem and communicates the outcome to the technical operator. If the resolution action is successful, the process ends on both sides. If not, the technical operator searches again for a resolution action. This is repeated until the problem is resolved. If the technical operator is busy for more than 30 minutes on a specific support request, the process ends by inviting the user to repeat the entire procedure again.

Exercise 3 – BPMN collaboration diagram



02291 System Integration

Introduction to DMN: Solutions to exercises

© Giovanni Meroni

Exercise 1

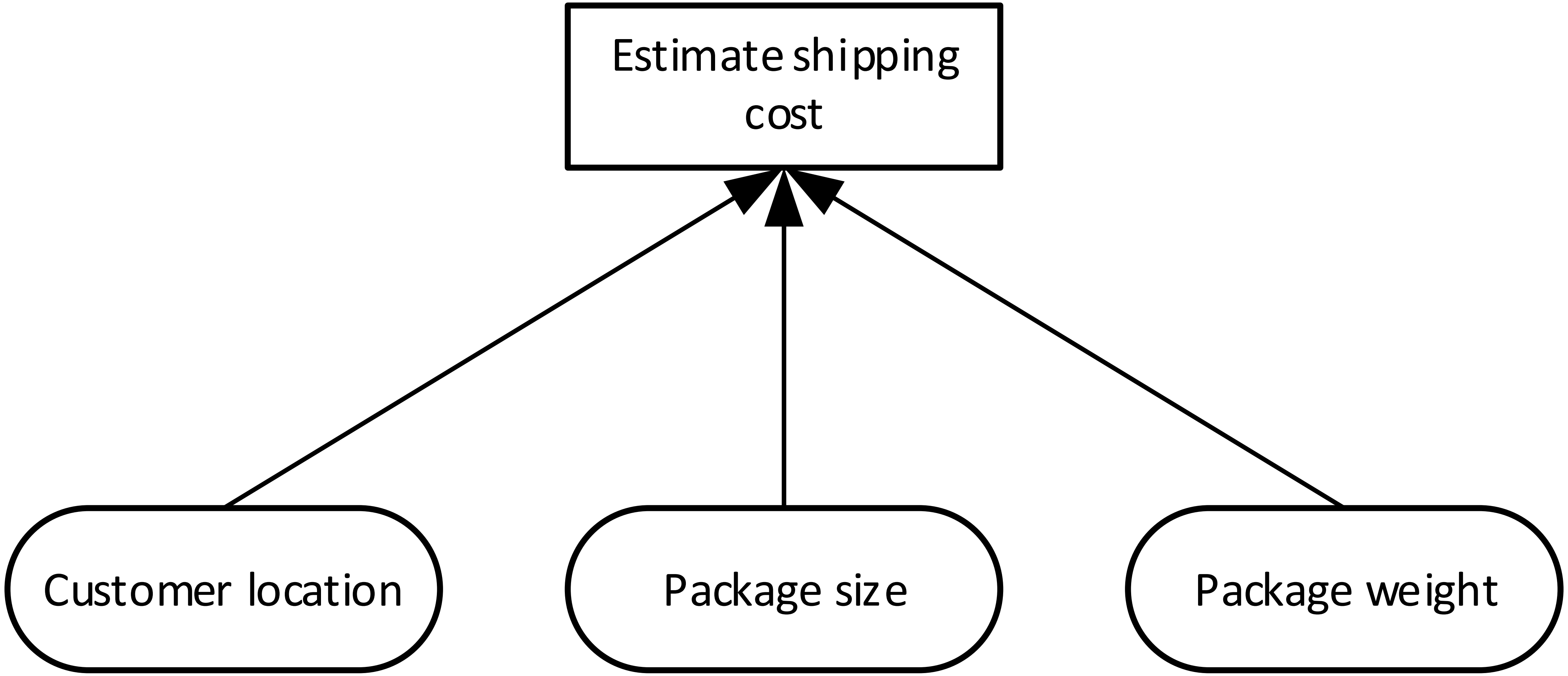
The decision task “Estimate shipping cost” of Example 3 is organized as it follows:

- If the customer is in EMEA, the weight of the package is lower than 10 Kg, and the volume of the package is smaller than 10 cm3, then the cost amounts to 100 DKK. Otherwise, the cost is 300 DKK.
- If the customer is in America or APAC, the weight of the package is lower than 5 Kg, and the volume of the package is smaller than 7 cm3, then the cost amounts to 200 DKK. If the weight of the package is greater than 20 Kg, and the volume of the package is bigger than 30 cm3, then the cost amounts to 2000 DKK. Otherwise, the cost amounts to 1000 DKK.

Implement this task with DMN:

- Draw the DMN diagram of this task
- Implement the decision with a decision table that uses the First (F) hit policy.
- Modify that decision table to use the Any (A) hit policy instead.
- Modify that decision table to use the Unique (U) hit policy instead.

Exercise 1 – DMN diagram



Exercise 1 – First decision table

Estimate shipping cost (F)	Customer location {EMEA, APAC, America}	Package weight [0,50]	Package volume [0,100]	Shipping cost [0,1000] 1000>0
1	EMEA	< 10	< 10	**100**
2	EMEA	-	-	**300**
3	-	< 5	< 7	**200**
4	-	> 20	> 30	**2000**
5	-	-	-	**1000**

Exercise 1 – Any decision table

Estimate shipping cost (A)	Customer location {EMEA, APAC, America}	Package weight [0,50]	Package volume [0,100]	Shipping cost [0,1000] 1000>0
1	EMEA	< 10	< 10	**100**
2	EMEA	>= 10	-	**300**
3	EMEA	-	>= 10	**300**
4	APAC	< 5	< 7	**200**
5	APAC	> 20	> 30	**2000**
6	APAC	>= 5	<= 30	**1000**
7	APAC	<= 20	>= 7	**1000**
8	America	< 5	< 7	**200**
9	America	> 20	> 30	**2000**
10	America	>= 5	<= 30	**1000**
11	America	<= 20	>= 7	**1000**

Exercise 1 – Unique decision table

Estimate shipping cost (U)	Customer location {EMEA, APAC, America}	Package weight [0,50]	Package volume [0,100]	Shipping cost [0,1000] 1000>0
1	EMEA	< 10	< 10	**100**
2	EMEA	>= 10	< 10	**300**
3	EMEA	< 10	>= 10	**300**
4	EMEA	>= 10	>= 10	**300**
5	APAC	< 5	< 7	**200**
6	APAC	> 20	> 30	**2000**
7	APAC	>= 5	<= 30	**1000**
8	APAC	<= 20	> 30	**1000**
9	APAC	< 5	[7,30]	**1000**
10	America	< 5	< 7	**200**
11	America	> 20	> 30	**2000**
12	America	>= 5	<= 30	**1000**
13	America	<= 20	> 30	**1000**
14	America	< 5	[7,30]	**1000**

Exercise 2

- Given the following decision table:

Determine discount (?)	Fidelity level {gold, silver, bronze}	Items bought [0,100]	Total value [0,100000]	Discount [0,30] 30>0
1	Gold	> 2	-	**30**
2	Gold	-	-	**10**
3	Silver	-	> 10000	**20**
4	Silver	> 2	-	**10**
5	Bronze	> 2	> 1000	**10**

- Which single hit policies would not cause any violation?
- Would all the previously identified hit policies lead to the same outcome?
- Is the table complete?

Exercise 2

- Given the following decision table:

Determine discount (?)	Fidelity level {gold, silver, bronze}	Items bought [0,100]	Total value [0,100000]	Discount [0,30] 30>0
1	Gold	> 2	-	**30**
2	Gold	-	-	**10**
3	Silver	-	> 10000	**20**
4	Silver	> 2	-	**10**
5	Bronze	> 2	> 1000	**10**

- Which single hit policies would not cause any violation? **First and Priority**
  - Rules 1 and 2, 3 and 4 could hold simultaneously (thus violating Unique) and would give different results (thus violating Any)

Exercise 2

- Given the following decision table:

Determine discount (?)	Fidelity level {gold, silver, bronze}	Items bought [0,100]	Total value [0,100000]	Discount [0,30] 30>0
1	Gold	> 2	-	**30**
2	Gold	-	-	**10**
3	Silver	-	> 10000	**20**
4	Silver	> 2	-	**10**
5	Bronze	> 2	> 1000	**10**

- Would all the previously identified hit policies lead to the same outcome? **Yes**
  – When 1 and 2 hold, First would select 1, giving 30% discount. Priority would also select 1 since 10 has lower priority than 30 (i.e., 10 < 30).
  – When 3 and 4 hold, First would select 3, giving 20% discount. Priority would Priority would also select 3 since 10 has lower priority than 20 (i.e., 10 < 20).

Exercise 2

- Given the following decision table:

Determine discount (?)	Fidelity level {gold, silver, bronze}	Items bought [0,100]	Total value [0,100000]	Discount [0,30] 30>0
1	Gold	> 2	-	**30**
2	Gold	-	-	**10**
3	Silver	-	> 10000	**20**
4	Silver	> 2	-	**10**
5	Bronze	> 2	> 1000	**10**

- Is the table complete? **No**
  - – Discount is not specified for Silver customers with 2 or less items bought and 10000 DKK or less total value.
  - – Discount is not specified for Bronze customers with 2 or less items bought or 1000 DKK or less total value.

Information Processing Letters 21 (1985) 181-185
North-Holland 7 October 1985

DEFINING LIVENESS *

Bowen ALPERN and Fred B. SCHNEIDER

Department of Computer Science, Cornell University, 405 Upson Hall, Ithaca, NY 14853, U.S.A.

Communicated by David Gries
Received 5 November 1984
Revised 20 February 1985

A formal definition for liveness properties is proposed. It is argued that this definition captures the intuition that liveness properties stipulate that 'something good' eventually happens during execution. A topological characterization of safety and liveness is given. Every property is shown to be the intersection of a safety property and a liveness property.

Keywords: Liveness, absolute liveness, uniform liveness, safety, property, topology, concurrency, semantics

1. Introduction

An execution of a concurrent program can be viewed as an infinite sequence of states:

$\sigma = s_0 s_1 \ldots$.

Each state after s_0 results from executing a single atomic action in the preceding state. (For a terminating execution, an infinite sequence is obtained by repeating the final state.) A *property* is a set of such sequences. Since a program also defines a set of sequences of states, we say that a property *holds* for a program if the set of sequences defined by the program is contained in the property.

It is useful to distinguish two classes of properties, since they are proved using different techniques. A proof that a program satisfies a *safety property* rests on an invariance argument [4], while a proof that a program satisfies a *liveness property* depends on a well-foundedness argument [6,7]. Safety and liveness were first described in [2]. The defining characteristic of safety properties was recently formalized in [3]. This paper gives a formal characterization of liveness properties and shows that all properties are the intersection of safety and liveness properties.

2. Safety properties

Informally, a safety property stipulates that some 'bad thing' does not happen during execution [2]. Examples of safety properties include mutual exclusion, deadlock freedom, partial correctness, and first-come-first-serve. In *mutual exclusion*, the proscribed 'bad thing' is two processes executing in critical sections at the same time. In *deadlock freedom* it is deadlock. In *partial correctness*, it is terminating in a state not satisfying the postcondition after having been started in a state that satisfies the precondition. Finally, in *first-come-first-serve*, which states that requests are serviced in the order they are made, the 'bad thing' is servicing a request that was made after one not yet serviced.

* This work was supported, in part, by NSF Grant DCR-8320274. F.B. Schneider was also supported by an IBM Faculty Development Award.

We now formalize safety.[1] Let S be the set of program states, S^ω the set of infinite sequences of program states, and S^* the set of finite sequences of program states. An execution of any program can be modeled as a member of S^ω. We call elements of S^ω *executions* and elements of S^* *partial executions* and write $\sigma \models P$ when execution σ is in property P. Finally, let σ_i denote the partial execution consisting of the first i states in σ.

For P to be a safety property, if P does not hold for an execution, then at some point some 'bad thing' must happen. Such a 'bad thing' must be irremediable because a safety property states that the 'bad thing' never happens during execution. Thus, P is a *safety property* if and only if the following holds.

Safety

$$(\forall \sigma : \sigma \in S^\omega : \\ \sigma \not\models P \Rightarrow (\exists i : 0 \leqslant i : (\forall \beta : \beta \in S^\omega : \sigma_i \beta \not\models P))).$$

There are two things to notice about this definition. First, the definition does not restrict a 'bad thing' except to require that it be *discrete*—if the 'bad thing' happens during an execution, then there is an identifiable point at which it happens. Second, a safety property can never require that something happens sometime, as opposed to always. Thus, the definition merely stipulates that a safety property unconditionally prohibits a 'bad thing' from occurring and if it does occur, there is an identifiable point at which this can be recognized.

3. Liveness properties

Informally, a liveness property stipulates that a 'good thing' happens during execution [2]. Examples of liveness properties include starvation freedom, termination, and guaranteed service. In *starvation freedom*, which states that a process makes progress infinitely often, the 'good thing' is making progress. In *termination*, which asserts that a program does not run forever, the 'good thing' is completion of the final instruction. Finally, in *guaranteed service*,[2] which states that every request for service is satisfied eventually, the 'good thing' is receiving service.

The thing to observe about a liveness property is that no partial execution is irremediable: it always remains possible for the required 'good thing' to occur in the future.[3] We take this to be the defining characteristic of liveness since if some partial execution were irremediable, then it would be a 'bad thing'; liveness properties cannot proscribe a 'bad thing', they can only prescribe a 'good thing'.

We now formalize liveness. A partial execution α is *live* for a property P if and only if there is a sequence of states β such that $\alpha\beta \models P$. A *liveness property* is one for which every partial execution is live. Thus, P is a liveness property if and only if the following holds.

Liveness

$$(\forall \alpha : \alpha \in S^* : (\exists \beta : \beta \in S^\omega : \alpha\beta \models P)).$$

Again, there are two things to notice about this definition. First, the definition does not restrict what a 'good thing' can be; it does not even require that the 'good thing' be discrete. In starvation freedom the 'good thing'—progress—is an infinite collection of discrete events. In this way, 'good things' are fundamentally different from 'bad things'. Second, a liveness property cannot stipulate that some 'good thing' *always* happens, only that it eventually happens.

We believe that no definition of liveness can be more permissive than the one given above. Suppose, by way of contradiction, that P is a liveness property that does not satisfy our definition. There

[1] This formalization differs slightly from the one proposed in [3]. Under Lamport's assumption that properties are preserved under finite repetition of individual states ('stuttering'), both definitions are equivalent [1]. A property that is not invariant with respect to stuttering is "the value of x differs in any two successive states". We believe this should be considered a safety property, and it meets our definition but not Lamport's.

[2] This is called *responsiveness* in [5].

[3] "While there's life there's hope."—*Cicero.*

must be some partial execution α such that

$$(\forall\beta : \beta \in S^{\omega} : \alpha\beta \nvDash P).$$

Clearly, α is a 'bad thing' proscribed by P. Thus, P is in part a safety property and not a liveness property, as was assumed.

Obviously, definitions for liveness more restrictive than ours are possible. One candidate we have investigated is the following.

Uniform Liveness

$$(\exists\beta : \beta \in S^{\omega} : (\forall\alpha : \alpha \in S^* : \alpha\beta \vDash P)).$$

P is a uniform-liveness property if and only if there is a single execution (β) that can be appended to every partial execution (α) so that the resulting sequence is in P. Another definition has been proposed by Sistla [10].

Absolute Liveness

$$(\exists\gamma : \gamma \in S^{\omega} : \gamma \vDash P)$$
$$\wedge (\forall\beta : \beta \in S^{\omega} : \beta \vDash P \Rightarrow (\forall\alpha : \alpha \in S^* : \alpha\beta \vDash P)).$$

P is an absolute-liveness property if and only if it is nonempty and any execution (β) in P can be appended to any partial execution (α) to obtain a sequence in P.

It is instructive to contrast these formal definitions. L is a liveness property if any partial execution α can be extended by some execution β so that $\alpha\beta$ is in L—the choice of β may depend of α. U is a uniform-liveness property if there is a single execution β that extends all partial executions α such that $\alpha\beta$ is in U. And, A is an absolute-liveness property if it is nonempty and any execution β in A can be used to extend all partial executions α. Any absolute-liveness property is a uniform-liveness property and any uniform-liveness property is a liveness property.

While absolute liveness characterizes an interesting class of properties, we do not believe it includes all properties that should be considered liveness. Any *leads-to* property (e.g., guaranteed service) is not an absolute-liveness property. Such properties[4] are characterized as follows.

Leads-to. *Any occurrence of an event of type* E_1 *is eventually followed by an occurrence of an event of type* E_2.

When E_2 is satisfiable, such properties are liveness properties—E_2 is the prescribed 'good thing' [2]. To see that a leads-to property is not an absolute-liveness property, consider an execution β in which no event of type E_1 or E_2 happens. Leads-to holds on β. However, appending β to a partial execution consisting of a single event of type E_1 yields an execution that does not satisfy the property.

We also believe that uniform liveness does not correctly capture the intuition for liveness. An example of a liveness property that is not a uniform-liveness property is characterized as follows.

Predictive. *If* A *initially holds, then after some partial execution* B *always holds; otherwise, after some partial execution* B *never holds.*

We believe this to be a liveness property, because it requires some 'good thing' (either 'always B' or 'always $\neg$B') to happen eventually. It is not a uniform-liveness property since there is no *single* sequence that can succesfully extend all partial executions.

4. Other properties

Many properties are neither safety nor liveness. For example, any property characterized as follows:[5]

Until. *Eventually an event of type* E_2 *will happen and all preceding events are of type* E_1.

is the intersection of a safety property and a liveness property. The safety property is "'$\neg E_1$ before E_2' does not happen' and the liveness property is 'E_2 eventually happens'. Total correctness is also the intersection of a safety property (partial correctness) and a liveness property (termination). In fact, every property is the intersection of a

[4] These are the *eventuality properties* of Manna and Pnueli [5].

[5] In temporal logic this property is denoted by $E_1 \mathcal{U} E_2$.

safety property and a liveness property. The proof of this (below) depends on a topological characterization of safety and liveness proposed by Plotkin[6] [9], who was motivated by Smyth [11].

There is a natural topology of S^{ω} in which safety properties are exactly the closed sets, and liveness properties (as defined above) are exactly the dense sets. The basic open sets of this topology are the sets of all executions that share a common prefix. As usual, an open set is the union of basic open sets, a closed set is the complement of an open set, and a dense set is one that intersects every nonempty open set. It is now possible to prove the following.

Theorem 1. *Every property* P *is the intersection of a safety property and a liveness property.*

Proof. Let $\bar{P}$ be the smallest safety property containing P and let L be $\neg(\bar{P}-P)$. Then,

$$\begin{aligned} L \cap \bar{P} &= \neg(\bar{P}-P)\cap\bar{P} \\ &= (\neg\bar{P} \cup P)\cap\bar{P} \\ &= (\neg\bar{P}\cap\bar{P}) \cup (P\cap\bar{P}) \\ &= P\cap\bar{P} = P. \end{aligned}$$

It remains to show that L is dense, and hence a liveness property. By way of contradiction, suppose there is a nonempty open set O contained in $\neg L$ and thus L is not dense. Then, $O \subseteq (\bar{P}-P)$. Consequently, $P \subseteq (\bar{P}-O)$. The intersection of two closed sets is closed, so $\bar{P}-O$ is closed and thus a safety property. This contradicts the hypothesis that $\bar{P}$ is the smallest safety property containing P. □

An obvious corollary of this is the following.

Corollary 1.1. *If a notation Σ for expressing properties is closed under complement, intersection, and topological closure, then any Σ-expressible property is the intersection of a Σ-expressible safety property and a Σ-expressible liveness property.*

[6] Plotkin nevertheless is unhappy with our definition of liveness because it is not closed under intersection.

Thus, in order to establish that every property P expressible in a temporal logic can be given as the conjunction of a safety property and a liveness property expressed in the logic, is suffices to show that the smallest safety property containing P is also expressible in the logic.

Plotkin has shown that any property that can be expressed in temporal logic can be written as the conjunction of two temporal logic expressible liveness properties [8]. In fact, a more general result can be proven.

Theorem 2. *If* $|S| > 1$, *then any property* P *is the intersection of two liveness properties.*

Proof. By hypothesis, there are two states a and b in S. Let L_a (respectively L_b) be the set of all executions with tails that are an infinite sequence of a's (respectively b's). Both L_a and L_b are liveness properties and $L_a \cap L_b = \Phi$. Now,

$$\begin{aligned} &(P \cup L_a)\cap(P\cup L_b) \\ &= (P\cap P)\cup(P\cap L_a)\cup(P\cap L_b)\cup(L_a\cap L_b) \\ &= P. \end{aligned}$$

The union of any set and a dense set is dense, so $P \cup L_a$ and $P \cup L_b$ are liveness properties and the theorem is proven. □

As before, there is an obvious corollary.

Corollary 2.1. *If a notation Σ for expressing properties is closed under intersection and there exist Σ-expressible liveness properties with empty intersection, then any Σ-expressible property is the intersection of two Σ-expressible liveness properties.*

Topology also provides a convenient framework for investigating the closure of safety and liveness under boolean operations. Safety properties (closed sets) are closed under union and intersection. Liveness properties (dense sets) are closed only under union. Neither is closed under complement. Finally, the only property that is both safety and liveness is S^{ω} itself.

5. Conclusion

A formal definition of liveness should be as general as possible without doing violence to the intuition. We have argued that no definition that is less restrictive than ours corresponds to this intuition. Any argument for a more restrictive formal definition should include an example of a property that meets our definition, but that does not seem to be a liveness property. It appears to us that any such definition will unduly restrict what a 'good thing' can be, but we cannot prove this.

It seems naive to hope for a proof that a formal definition for liveness (or safety) is correct, because 'good things' and 'bad things' are not well-defined concepts. However, the simple topological characterization of the definitions suggests that they do indeed capture fundamental distinctions.

Acknowledgment

We wish to thank A. Demers, D. Gries, M. Krentel and E. Mallison for comments on an early draft of this paper. Comments from G. Plotkin on two separate occasions forced us to understand the consequences of our liveness definition and are gratefully acknowledged. Thanks are also due to J. Hook, L. Lamport, A. Pnueli, and A.P. Sistla for helpful discussions and encouragement concerning this work.

References

[1] B. Alpern, F.B. Schneider and A.J. Demers, A note on safety without stuttering, In preparation, 1985.

[2] L. Lamport, Proving the correctness of multiprocess programs, IEEE Trans. Software Engineering SE-3 (2) (1977) 125–143.

[3] L. Lamport, Basic concepts, in: Advanced Course on Distributed Systems—Methods and Tools for Specification, Lecture Notes in Computer Science 190 (Springer, Berlin–Heidelberg, 1984).

[4] L. Lamport and F.B. Schneider, The 'Hoare Logic' of CSP and all that, ACM Trans. Programming Languages and Systems 6 (2) (1984) 281–296.

[5] Z. Manna and A. Pnueli, Temporal verification of concurrent programs: The temporal framework for concurrent programs, in: R.S. Boyer and J.S. Moore, eds., The Correctness Problem in Computer Science (Academic Press, London, 1981).

[6] Z. Manna and A. Pnueli, Verification of concurrent programs: Proving eventualities by well-founded ranking, Tech. Rept. STAN-CS-82-915, Dept. of Computer Science, Stanford Univ., 1982.

[7] S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. Programming Languages and Systems 4 (3) (1982) 455–495.

[8] G. Plotkin, Private communication, 1983.

[9] G. Plotkin, Private communication, 1984.

[10] A.P. Sistla, Characterization of safety and liveness properties in temporal logic, Extended abstract, Univ. of Massachusetts at Amherst, MA, 1984.

[11] M.B. Smyth, Power domains and predicate transformers: A topological view, in: Proc. ICALP '83, Lecture Notes in Computer Science 154 (Springer, Berlin–New York, 1983) 662–675.