Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

LØSNINGSFORSLAG EKSAMENSOPPGAVE I TTM4115 – SYSTEMERING AV DISTRIBUERTE SANNTIDSSYSTEMER SOLUTION PROPOSAL EXAM TTM4115 ENGINEERING DISTRIBUTED REAL-TIME SYSTEMS

Contact person/Faglig kontakt under eksamen: Rolv Bræk

Phone/Tlf.: 415 44 605

Exam date/Eksamensdato:	01. juni 2012
Time/Eksamenstid:	09:00-13:00
Credits/Studiepoeng:	7,5 SP
Remedies/Tillatte hjelpemidler:	A: All written and handwritten examination support materials are permitted. All calculators are permitted A: Alle trykte og håndskrevne hjelpemidler tillatt. Alle kalkulatorer tillatt
Languages/Språkform:	
Antall sider bokmål:	1
Number of pages in English:	1
Tal på sider nynorsk:	0
Attachment/Antall sider vedlegg:	7
Results/Sensurdato[1]:	**22. juni 2012**

[1] Merk! Studentene må primært gjøre seg kjent med sensur ved å oppsøke sensuroppslagene.

Bokmål (Eksamen utgjør 75% av sluttkarakteren.)

Oppgavene referer seg til systemet som er beskrevet i vedlegg. Studer vedlegget først.

Oppgave 1. (30%) SDL

1. Definer *Carpool* systemet formelt som en SDL systemtype med *Parking Station Terminal* som en blokktype definert innenfor *Carpool* systemet. Ta med signaler på kanalene og signaldefinisjoner. (Ta bare med blokkene vist i Figur 1.)
2. Definer oppførselen til *Central Station* som en SDL prosessgraf (tilstandsdiagram) som tilfredsstiller alle *cs* rollene referert i Figur 2 og detaljert i Figur 3 og 4. Ta med datadeklarasjoner og operasjoner (du kan bruke Java eller pseudokode til dette).
3. Definer oppførselen til *ctr* rollen i *ReturnKey* kollaborasjonen som en SDL sammensatt tilstand (composite state) som tilfredsstiller sekvensdiagrammet i Figur 9. Ta med nødvendige datadeklarasjoner og operasjoner.
4. Anta at alle *ctr* rollene som spilles av *PSControl*, se Figur 5, er definert som sammensatte tilstander (composite states). Definer oppførselen til *PSControl* som en SDL prosessgraf (tilstandsdiagram) som benytter disse tilstandene.

Oppgave 2. (25%) Aktivitetsdiagram (Arctis form)

1. Konstruer en Arctis byggeblokk (building block) for *ctr* rollen i *ReturnKey* kollaborasjonen. Den skal tilfredsstille sekvensdiagrammet i Figur 9. (Samme rolle som i oppgave 1.3 over.) Anta at hver melding i sekvensdiagrammet representeres med en tilsvarende pin. Den indre aktivitetsflyten skal defineres.
2. Anta nå at hver av *ctr* rollene som spilles av *PSControl* er pre-definert som en Arctis byggeblokk med pins for hver av meldingene den mottar og sender. Konstruer en Arctis byggeblokk for *PSControl* som viser kontrollflyten mellom disse pre-definerte blokkene. Pinner (pins) som representerer meldinger til og fra de predefinerte blokkene skal vises men trenger ikke forbindes.
3. Forklar bruken av hendelse-mottak-aksjoner (receive event actions) til å fange opp hendelser fra brukergrensesnitt. Illustrer med et eksempel fra *Panel* der du antar at *Panel* er en GUI blokk.

Oppgave 3. (20%) Validering

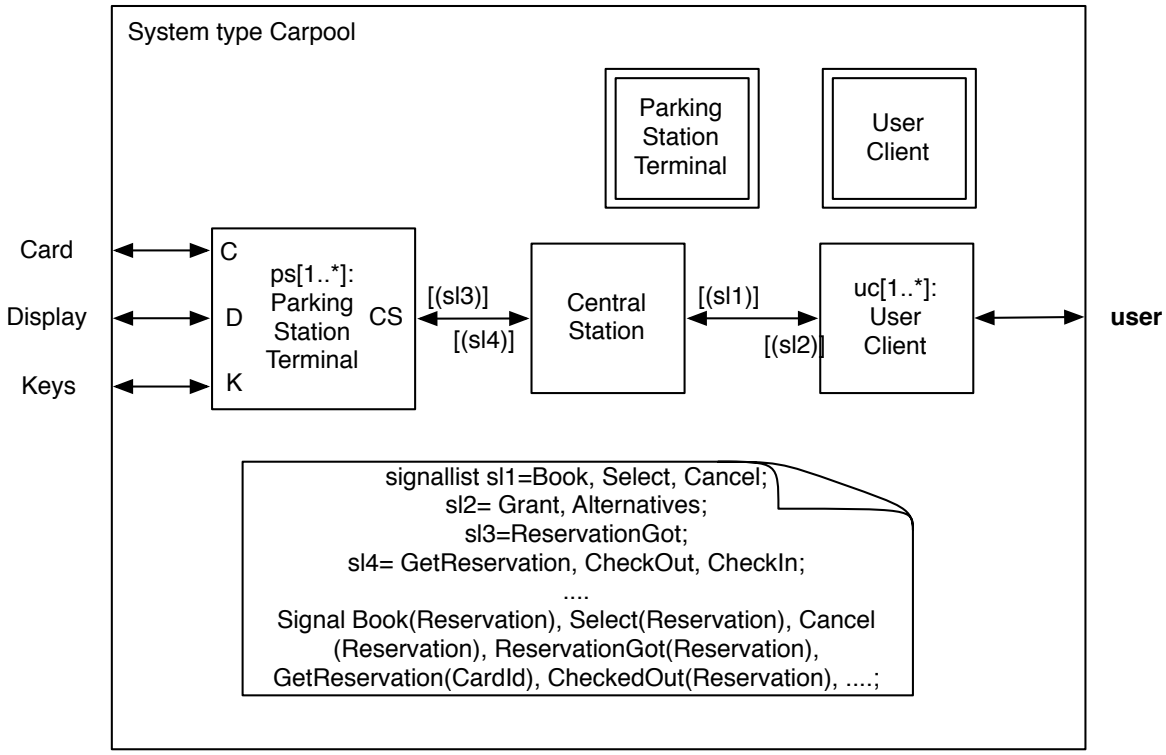
1. Forklar hva som kreves for at en SDL proessgraf (tilstandsdiagram) kan sies å tilfredsstille et sett med sekvensdiagram.
2. Forklar hva som menes med blandet initiativ (mixed initiative) og hvorfor det er viktig å kunne identifisere slike.
3. Forklar hva som menes med input konsistens og hvorfor det er viktig å sikre input konsistens.
4. Hva er formålet med ESM i Arctis?

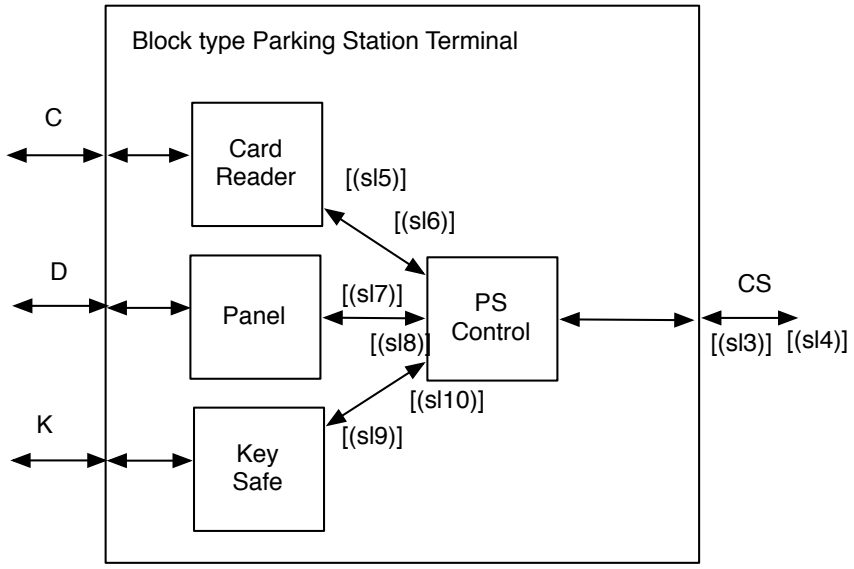
English (The exam counts 75% towards the final grade.)

The questions refer to the system described in the appendix. Study the appendix first.

Question 1. (30%) SDL

1. Define the *Carpool* system formally as an SDL System type with the *Parking Station Terminal* as a Block type defined within the scope of the *Carpool* system. Include signals on the channels and signal definitions. (Consider only the blocks shown in Figure 1.)

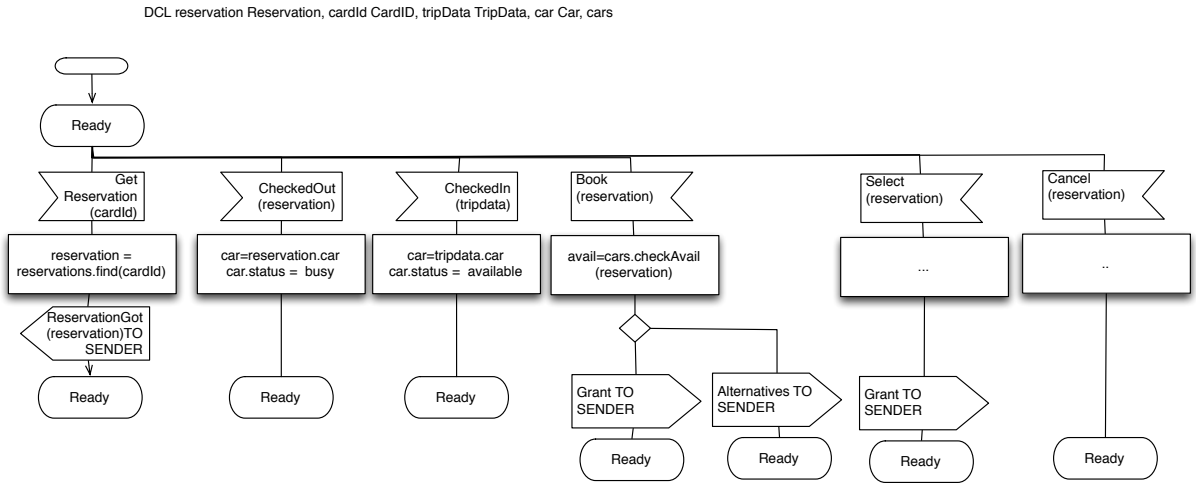




Some signals and signal definitions have been omitted. This is sufficient for full score. One may choose either to use 3 gates towards the user interface as here, or just one, but shall be done in a consistent way
PST not defined=-3

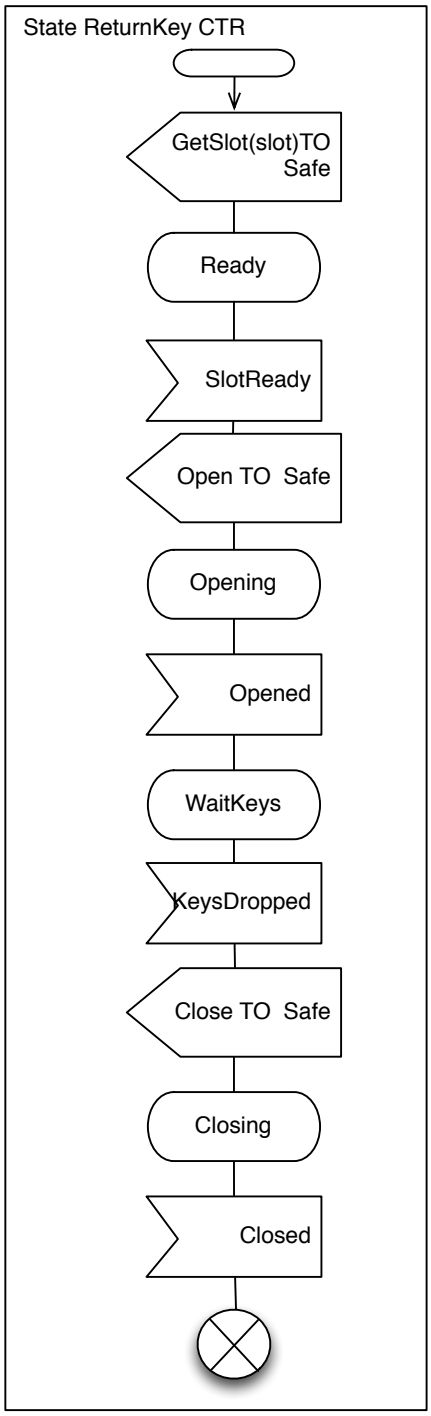
Gates not defined= -2
Gate references missing=-1.
Signals not defined=-1
Substantial SDL syntax errors, e.g two way signal lists=-1
Typereferneces missing =-1

2. Define the behaviour of the *Central Station* as an SDL process graph (state diagram) that satisfies all the *cs* roles referenced in Figure 2 and detailed in Figure 3 and 4. Include data declarations and operations (you may use Java or pseudo code for this).



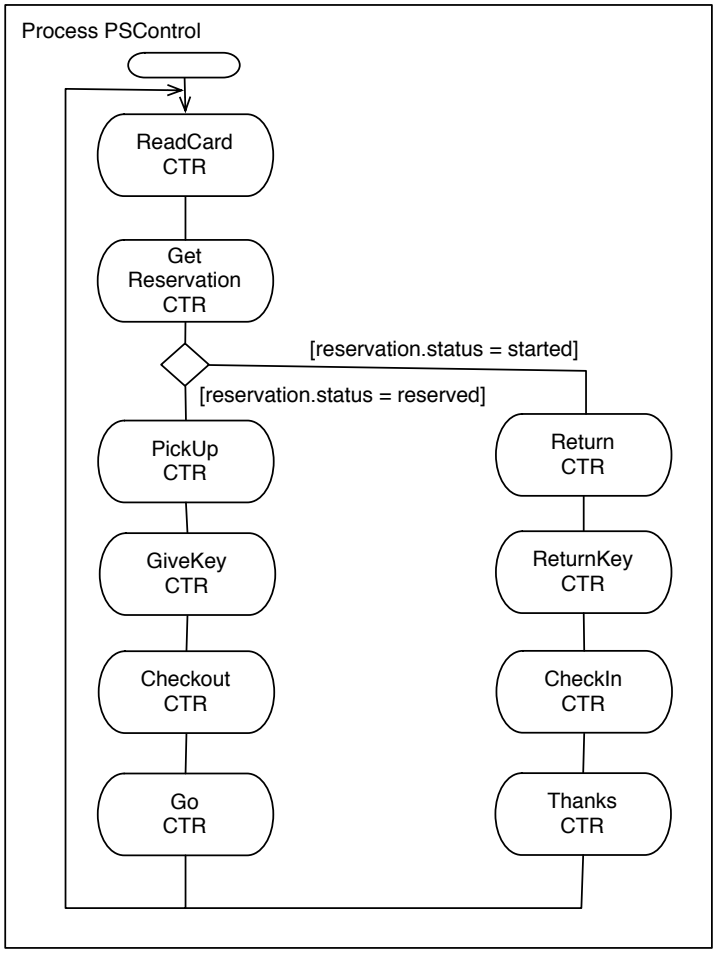
++add declarations and operations
Needs to be stateless in order to handle several customers in parallel.
Needs variables to hold reservation lists while waiting for option selection.
Stateful behaviour=-1,5
None in stead of decisions=-1,5
Missing or unclear actions=-1
No data dcl and no actions=-3
Missing data dcl=-1,5

3. Define the behaviour of the *ctr* role of the *ReturnKey* collaboration as an SDL composite state that satisfies the sequence diagram in Figure 9. Include the necessary data declarations and operations.



Note that the first sending should be done on the initial transition!
Using none or an unspecified signal to trigger the first sending=-1
Missing exit=-1,5
Minor design flaws=-1
Major SDL error=-1,5-2
Stateless=-1

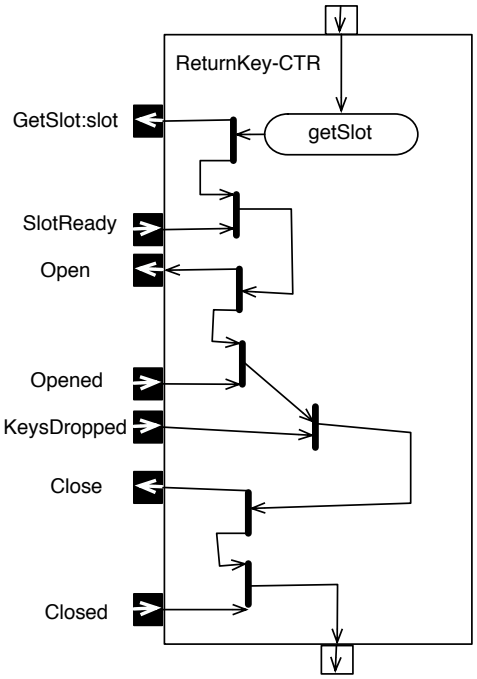
4. Assume that all the *ctr* roles played by the *PSControl*, see Figure 5, are defined as SDL composite states. Define the behaviour of the *PSControl* as an SDL process graph using these composite states.



Note that the diagram shall show the ordering of the roles!
No ordering defined=-5
Showing inputs that should be in states =-1
showing none=-1
not using the state defined in q1,3=-1
not using composite states=-2 (but otherwise the right flow)

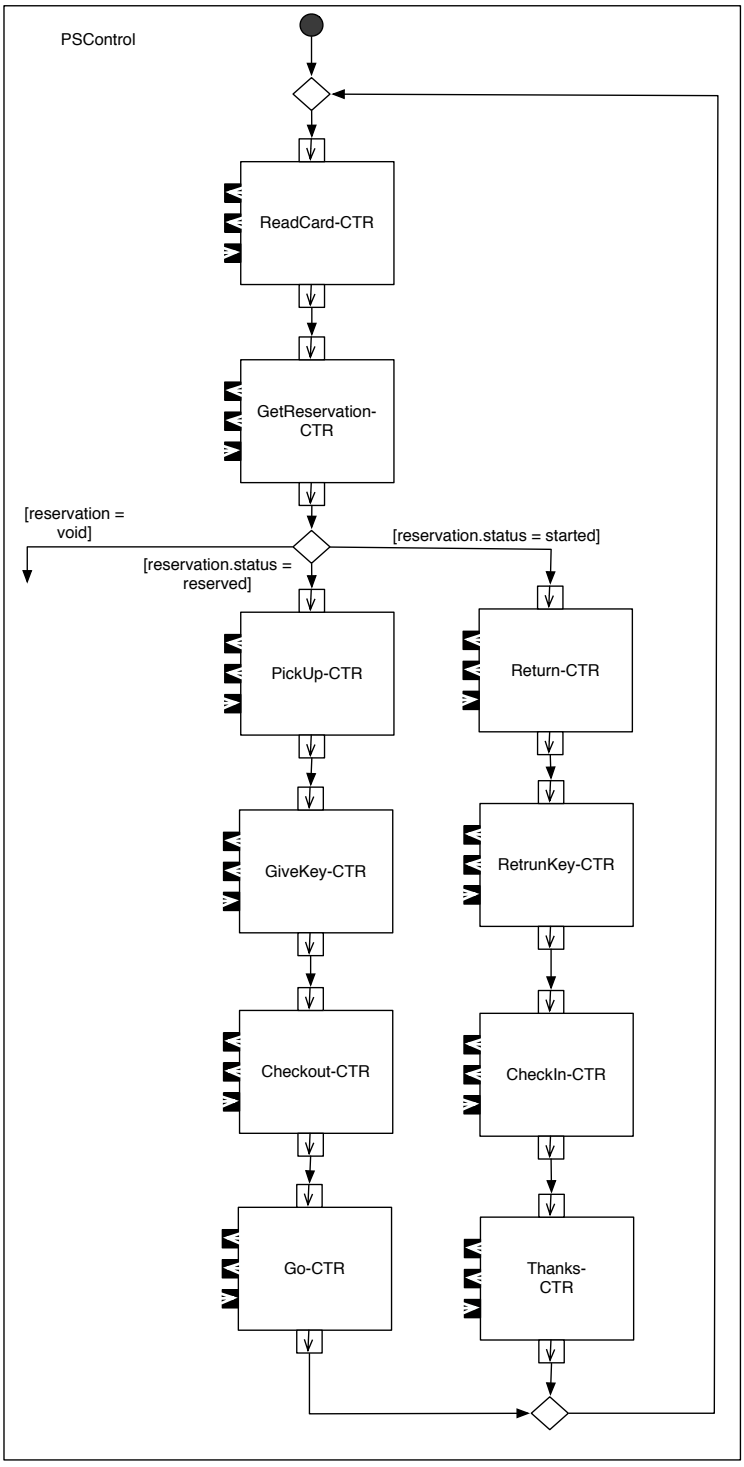
Question 2. (25%) Activity diagrams (Arctis style)

1. Design an Arctis building block for the *ctr* role of the *ReturnKey* collaboration. It shall satisfy the sequence diagram in Figure 9. (Same role as in question 1.3 above.) Assume that each message in the sequence diagram maps to a corresponding pin. The internal activity flow shall be defined.



We have here assumed that messages map to pins and flows as specified. Furthermore we have enforced the specified ordering by the flow. We have not defined an ESM, but it would not be wrong to do so. This is a case where just an ESM could be sufficient (a shallow block). Doing so would be OK!
Not enforcing ordering in flow or ESM=-3
Not separating streaming from other pins=-1
Using receive event actions=-1
Object types missing or wrong=-0.5
Only showing interface=+3

2. Assume that each *ctr* role played by the *PSControl* has been pre-defined as an Arctis building block, with pins corresponding to the messages it receives and sends. Design an Arctis building block for the *PSControl* showing the control flow among these pre-defined blocks. The message pins of the predefined blocks need not be connected, but shall be shown.



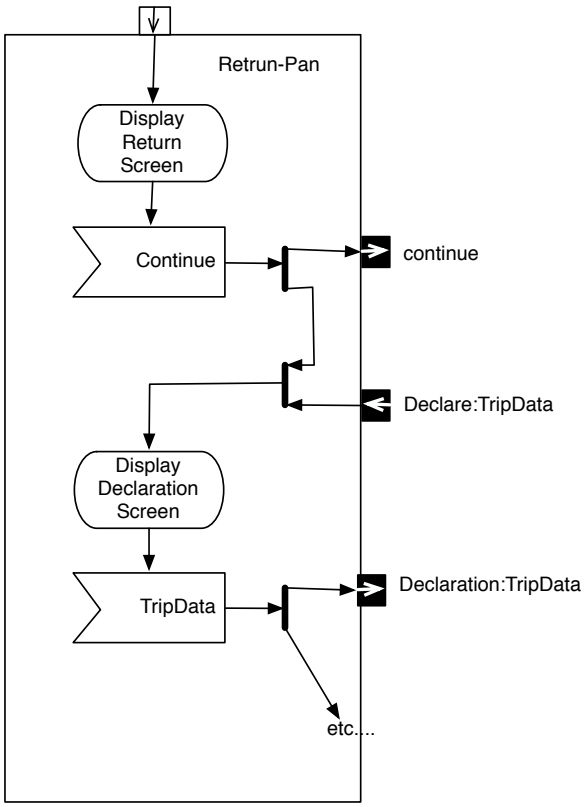
We have just indicated the presence of streaming pins for messages here. Giving exact pins was not the main point, but to show the flow. (specifying pins in detail is fine).

Here a student may do similar mistakes as on q1.4. We give credit for solutions that are consistent with their solution to 1.4 to avoid penalizing twice for same mistake.

Not using answer to q2.1=-1
Not representing the ordering if done so in q1.4=-3
... if not done so in q1.4=-1
Only streaming flows or control flows=-1

Combining blocks, e.g. just one for Safe=-1
Inventing new message-pins not specified=-1
Specifying message flows incorrectly=-0.5
Only PSControl context- not internal flow=+3

3. Explain the use of receive event actions to capture user interface events. Illustrate with a case from the *Panel* assuming that the *Panel* is a GUI block.



The principle is that the Java code for the GUI has listeners corresponding to buttons clicks. This is the normal technique for GUI programming using Java. When a listener is called due to a button click the event is conveyed to the Arctis model by a method called SendToBlock(event, ...). This triggers a receive event action in the Arctis diagram. In the Panel we will use a receive event action for each button that can be clicked in the current window.

SendToBlock not mentioned=-2
Receive event actions not mentioned=-1
Display action not mentioned=-1
No illustration=-2

Question 3. (20%) Validation

1. Explain what is required for an SDL process graph to satisfiy a set of sequence diagrams.

The SDL process graph shall contain all the event sequences specified for the corresponding instances in the MSCs. It may contain more sequences, but no less. In

the case where MSCs define interfaces/roles one may ignore events on other interfaces/roles in the process graph.
Not mentioning sequences/ordering or behavior=-3
Not mentioning that the process can do more=-2
Only saying that all roles must be considered=+3

2. Explain the concept of a mixed initiative and why it is important to recognize mixed initiatives.

   Mixed Initiatives is a situation where a process in a given state may receive inputs from two or more independent sources/interfaces. In a role behaviour it shows up as a state with both none and visible inputs. Important because this is where input inconsistency may occur.
   Missing why=-2
   Imprecise why=-1

3. Explain the concept of input consistency and why it is important to ensure input consistency.

   When two states of a role are connected by a none-transition (also called an invisible transition) the next state should accept at least the same visible inputs as the current state. If not, it is impossible to construct a process in the environment that can fully explore the role without causing unspecified receptions and problems that may follow from that. Thus, it is an indication that the process will not work correctly with any other process that try to explore its full behaviour. This problem only occur in connection with mixed initiatives!

   Not a correct definition, but a correct example=-2
   Not saying why it is important=-2
   Not mentioning unspecified reception as one reason=-1
   Imprecise why=-1

4. What is the purpose of an ESM in Arctis?

   The purpose is to:
   a. *Specify the ordering of tokens passed across the interfaces of a building block that the environment must respect and the building block will guarantee.*
   b. *Support validation that the internal flows are consistent with the ESM*
   c. *Replace the internal flows in the analysis of the enclosing flows.*
   d. *Support validation that external flows do not harm the ESM.*

   This has not been lectured in detail, but the students have used ESMs in their term assignment, and should be able to explain this to a certain degree.
   Mention that it provides states, but nothing else=+2
   Also mention restrictions=+2
   Explaining a above=+8.5
   Not mentioning b,c,d above=-1.5 total
   Not explaining ordering=-2
   Stating that it defines the external behaviour of a block=+5

Vedlegg/ Appendix

English only

Carpool

We shall here study a system for carpooling as defined in Figure 1 (using UML). The system manages a fleet of cars that can be rented by registered *Users* for shorter or longer periods. The cars are available at *Parking Stations* where there are special *Parking Station Terminals* that have a *Key Safe*, a *Card Reader* and a touch display *Panel*. *Users* book cars by accessing the *Central Station* from mobile *User Clients*. The *Central Station* manages reservations using the interface collaborations specified in Figure 2 and detailed in Figures 3 and 4. The *Central Station* keeps track of *available* and *busy* cars by means of the *Check Out* and *Check In* collaborations defined in Figure 4. Datatypes used in the system are defined in Figure 10.

A car is picked up and returned at a *Parking Station* according to the procedure define by the *Car Pickup and Return* activity referenced in Figure 5 and defined in Figure 6. This is an activity diagram that defines an ordering of sequence diagrams. It is essentially an Interaction Overview diagram (which is the UML version of an HMSC diagram). The roles participating in each sequence diagram is indicated by partitions in the action symbols in order to help reading the diagram. The referenced sequence diagrams are defined in Figures 4,7,8 and 9.

At the parking station the user inserts a card in the *Parking Station Terminal*. The card carries a *CardID* that is used to identify the *User* and to fetch the *User's* current *Reservation* from the *Central Station*. If there is a current *Reservation* with status *reserved*, see Figure 10, the terminal will assume that the user is there to pick up a car. It will then identify the car and the corresponding *Slot* in the *KeySafe* where the car keys are stored and make the keys available for the user in a drawer as indicated in the *GiveKeys* collaboration in Figure 9. If the status of the current *Reservation* is *started* the terminal will assume the user is there to return a car. The user is then asked to fill in *TripData* information about mileage, fuel level and parking lot. When this is done the *KeySafe* will open a drawer to receive the returned car keys. If the returned *Reservation* is *void*, the user may pick up a car directly if one happens to be available (not detailed here).

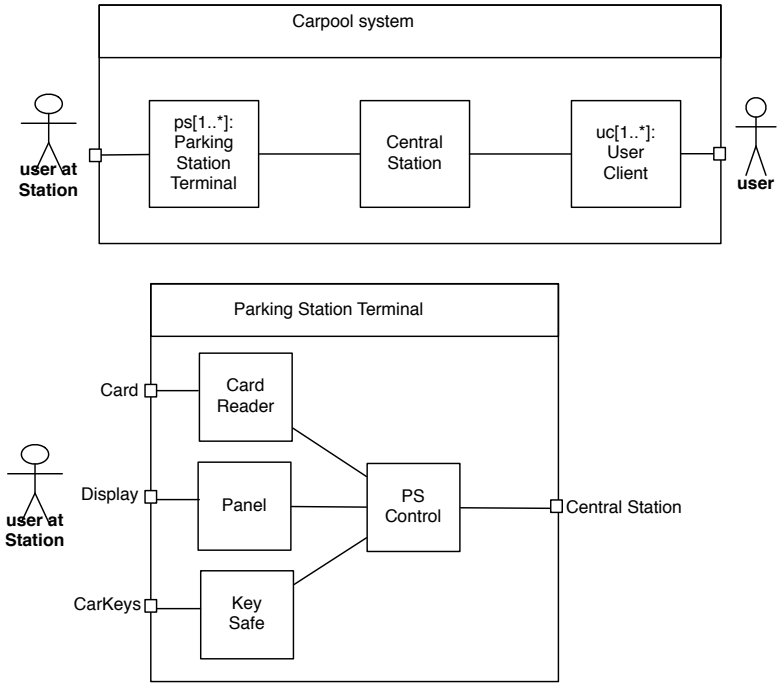


Figure 1 The Carpool system (UML notation)

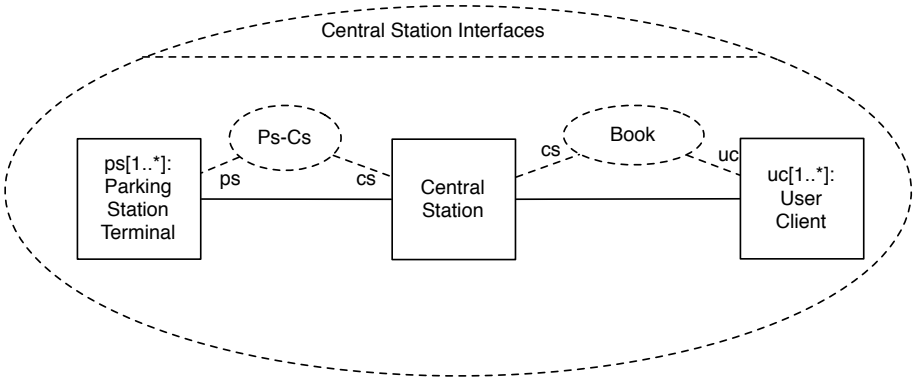


Figure 2 Central Station interfaces as collaborations

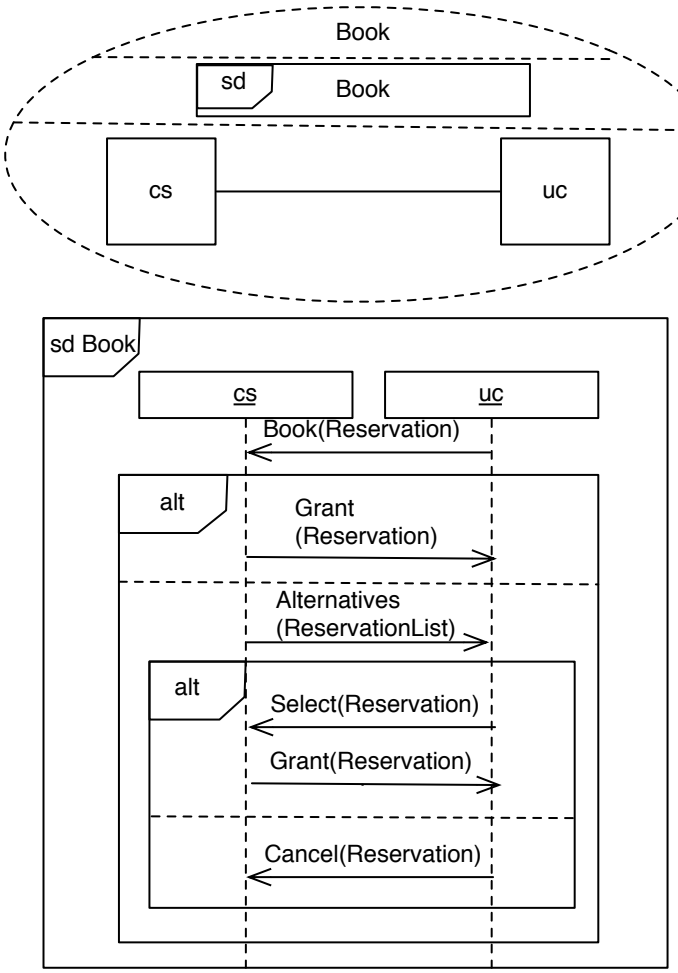


Figure 3 The Book collaboration

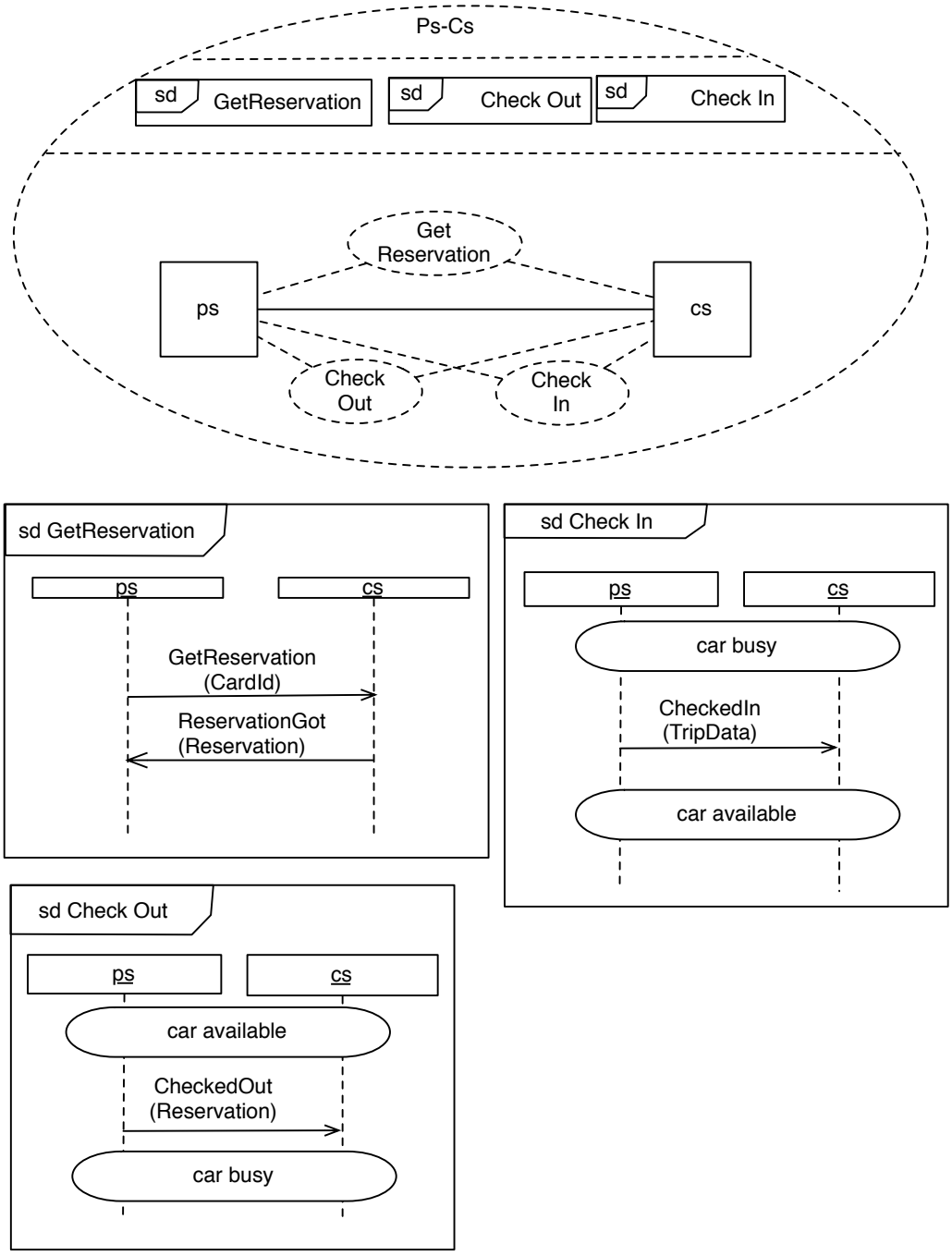


Figure 4 The Ps-Cs interface

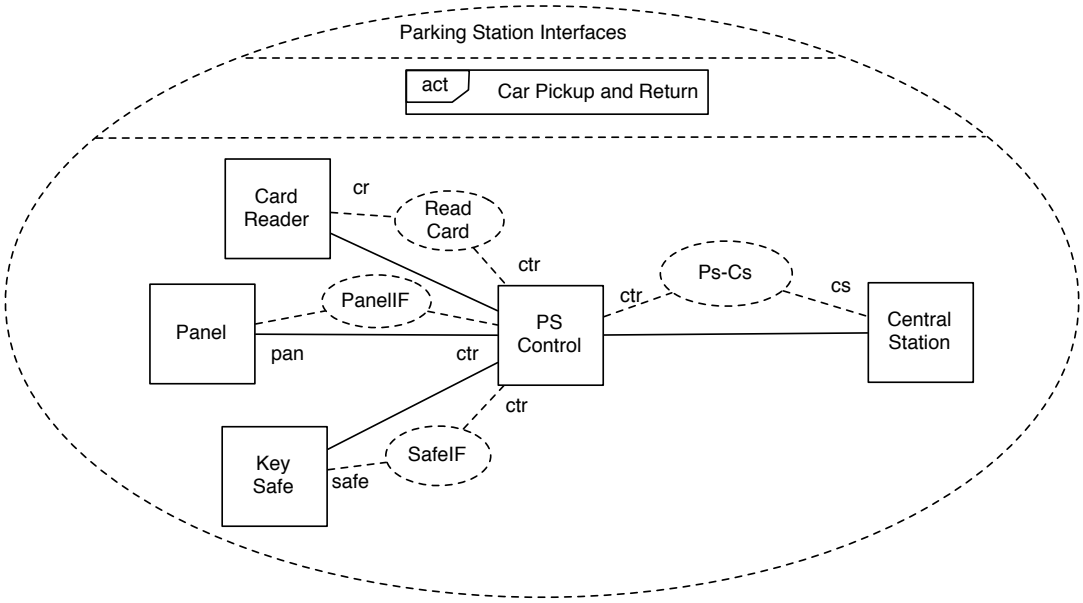


Figure 5 Parking Station Interfaces as collaborations

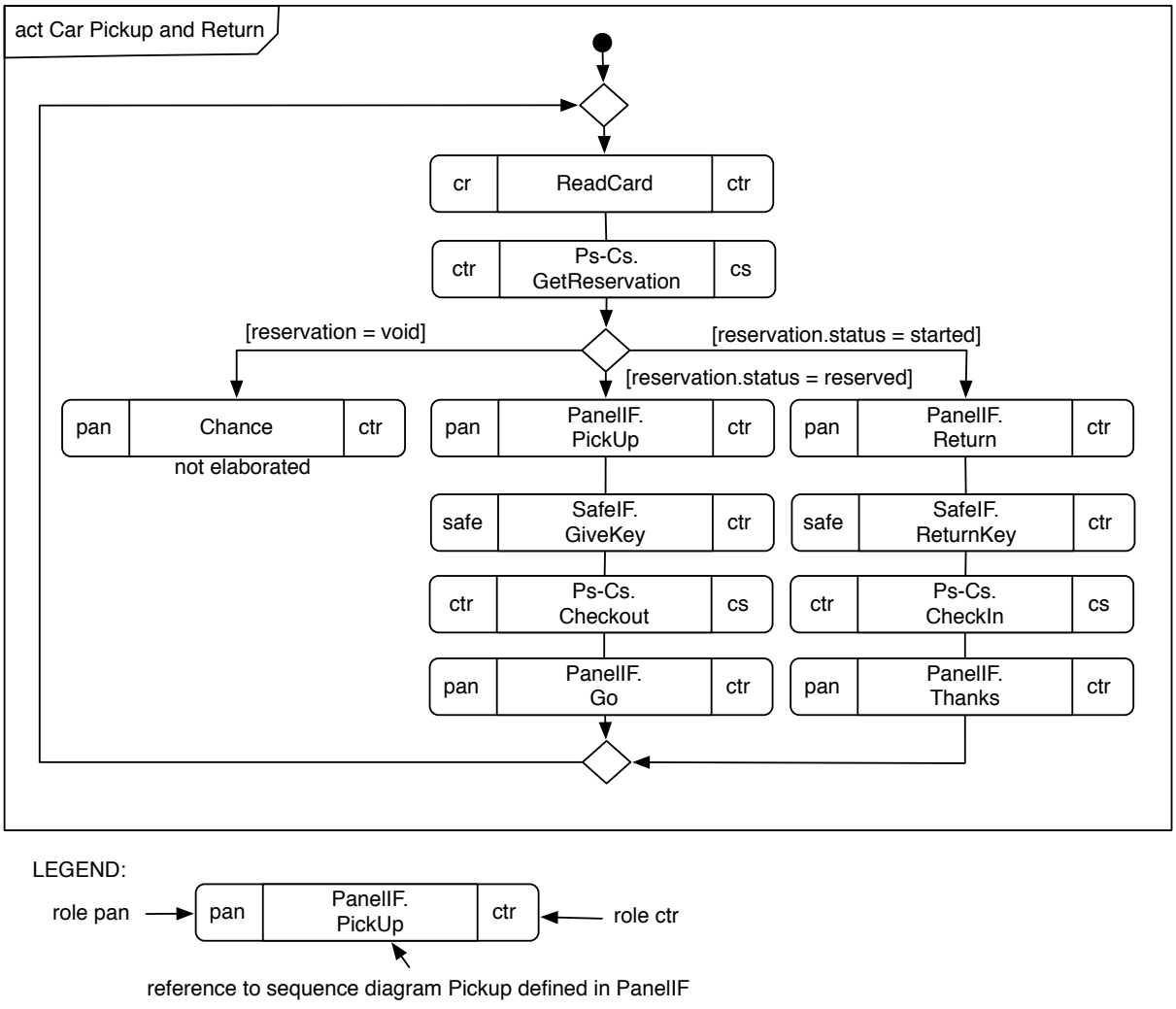


Figure 6 The Car Pickup and Return behaviour. The diagram is an Activity Diagram used like an Interaction Overview Diagram (UML) or HMSC (MSC). Actions refer to sequence diagrams having the roles indicated.

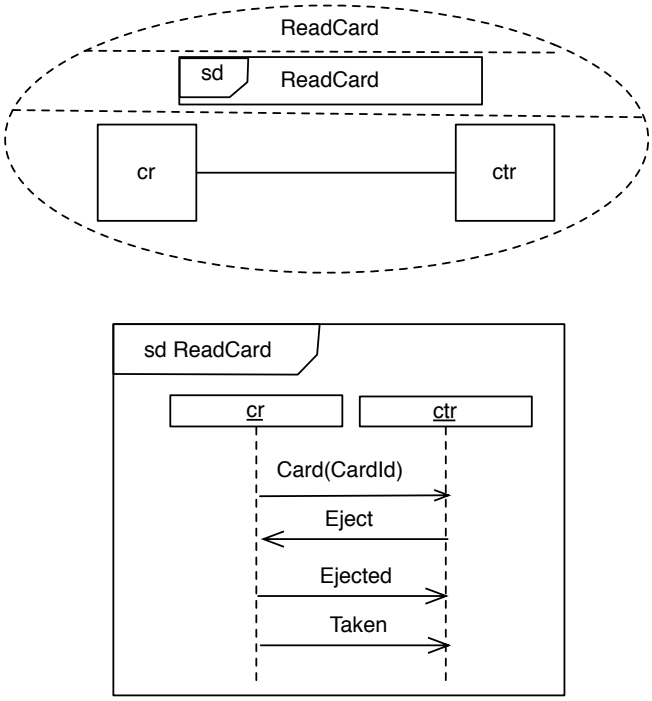


Figure 7 The ReadCard Collaboration

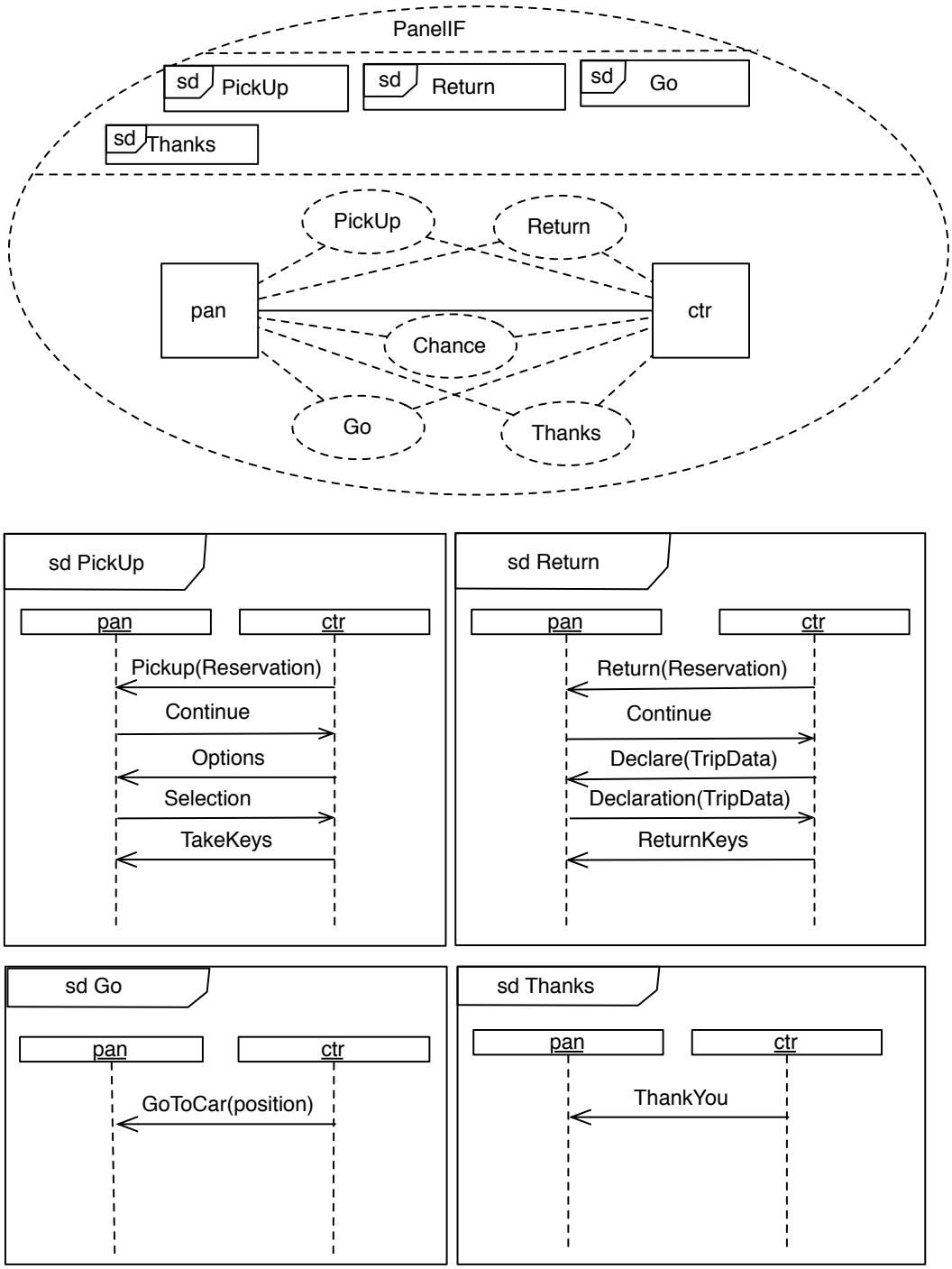


Figure 8 The PanelIF collaboration

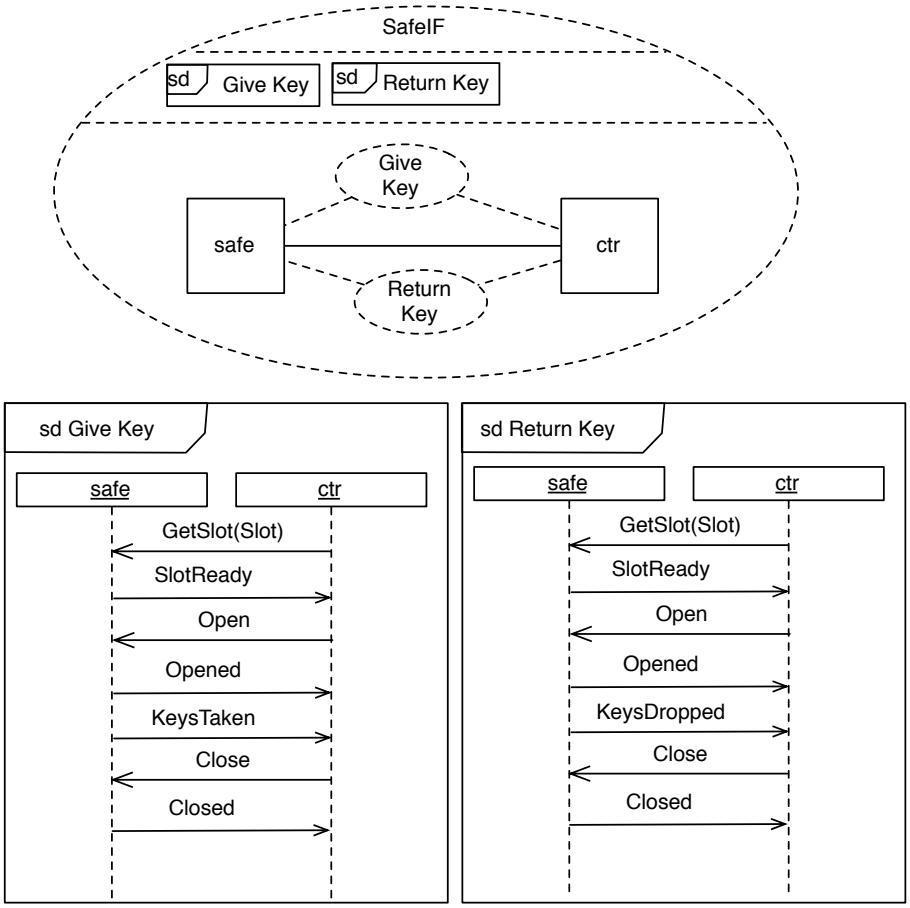


Figure 9 The SafeIF collaboration

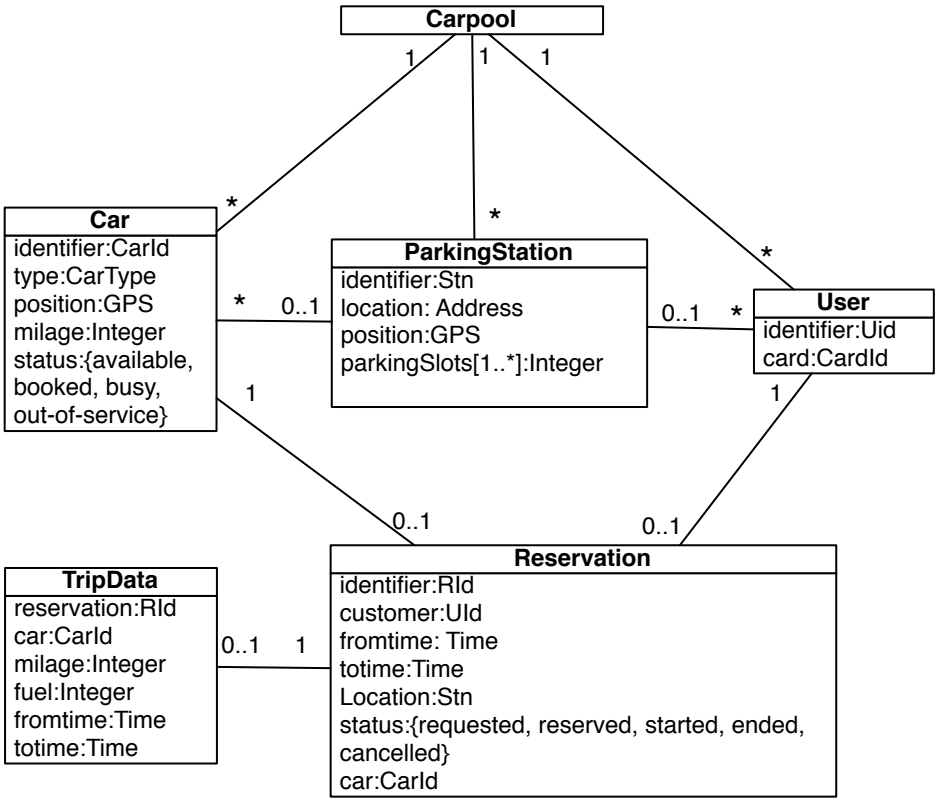


Figure 10 Data types in the Carpool system