

Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk



**EKSAMENSOPPGAVE I TTM4115 – SYSTEMERING AV
DISTRIBUERTE SANNTIDSSYSTEMER
EXAM TTM4115 ENGINEERING DISTRIBUTED REAL-TIME
SYSTEMS**

Contact person/Faglig kontakt under eksamen:	Rolv Bræk
Phone/Tlf.:	415 44 605
Exam date/Eksamensdato:	29. mai 2010
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
Antall sider nynorsk:	1
Attachment/Antall sider vedlegg:	6
Results/Sensurdato¹:	21. juni 2010

¹ Merk! Studentene må primært gjøre seg kjent med sensur ved å oppsøke sensuoppslagene.

Bokmål (Eksamen utgjør 75% av slutt karakteren.)

Noen av oppgavene referer seg til systemet som er beskrevet i vedlegg. Studer vedlegget først.

Oppgave 1. (25%) Struktur

1. Anta at alle delene (boksene) i Figur 1 skal være SDL prosesser. Gjør Figur 1 om til et mest mulig tilsvarende SDL blokk type diagram. Anta at alle typene er definert utenfor blokk typen i Package *AlarmLib*. Signaler på kanalene kan utelates.
2. Vis typereferansene som inngår i *AlarmLib* som UML klasser. Vi antar her bruk av SDL2000 eller nyere SDL versjoner. Angi lokale variabler som attributter til klassene og vis assosiasjoner mellom klassene med multiplisitet på endepunktene.
3. Forklar forholdet mellom kollaborasjonsroller og klasser. Bruk eventuelt *gd1*, *gd2* og *gd3* og klassen *GuardDispatch* i Figur 1 som eksempel.
4. Anta at det i perioder med lav trafikk (natt for eksempel) skal være mulig å kjøre systemet uten operatører og bare med vektore. Foreslå endringer i systemstrukturen i Figur 1, inkludert kollaborasjoner, og forklar virkemåten.

Oppgave 2. (30%) SDL

1. Prosessgrafen for *GuardAgt* gitt i Figur 5 mangler lokale variabler og signalparametre. Kompletter diagrammet med bruk av signalparametre og erklæring av lokale variabler (tilføyelsene kan gjøres på Figur 6 som rives ut og legges ved besvarelsen). Definer hvordan *guardID* initialiseres i *Initialize* prosedyren.
2. Med utgangspunkt i SDL prosessgrafen for *GuardAgt*, og kollaborasjonen *GiveOrder* i Figur 3, definer oppførselen til *GuardDispatcher* som en SDL prosess. Prosessen skal ha variabler av typen *GuardArray* og *OrderQ*, se Figur 4. Deklarer de lokale variabelene og vis hvordan de brukes. Vi antar at data typene er definert i *AlarmLib* og at operasjonen *FindGuard(GeoPos)* returnerer identiteten på nærmeste ledige vektor eller null dersom ingen er ledig. Det er tilstrekkelig å definere transisjoner for signalene *Order* og *Free*.
3. Definer oppførselen til de to rollene *gd2* og *gal* i *PosUpdate*, vist i Figur 2, som fragment av SDL prosessgrafer. Deklarasjon og bruk av lokale variable og timere skal medtas.
4. Vis hvordan fragmentet for *gal* ved hjelp av arv kan legges til i *GuardAgt* slik at det kan utføres i enhver tilstand der *GuardAgent* er i tjeneste. Gjør om nødvendig, enkelte transisjoner i Figur 6 virtuelle (og legg ved figuren).

Oppgave 3. (20%) Diverse

1. Er oppførselen til *GuardAgt* gitt i Figur 5 inputkonsistent? Påvis eventuelle inkonsistenser og forklar hva som må gjøres for at den skal bli input konsistent.
2. Hvilke deler av alarm systemet egner seg for fysisk distribusjon, og hvilke deler bør ligge på en felles server? Begrunn svarene.
3. Hva er de viktigste fordelene man oppnår ved å bruke asynkrone meldinger som i SDL, fremfor synkrone metodekall som i Java?
4. Hvilke mekanismer på toppen av standard Java må til for å realisere et distribuert SDL system som Alarmsystemet? Begrunn kort.

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

Some of the questions refer to the system described in the appendix. Study the appendix first.

Question 1. (25%) Structure

1. Assume that all the parts (boxes) in Figure 1 shall be SDL processes. Make an SDL block type diagram that corresponds as closely as possible to Figure 1. Assume that all types are defined outside the block type in Package *AlarmLib*. Signals on the channels need not be defined.
2. Show the type references in *AlarmLib* using UML classes. We assume here that we use SDL 2000 or newer SDL versions. Define local variables as class attributes and show the associations among the classes indicating the multiplicities on association endpoints.
3. Explain the relationship between collaboration roles and classes. Use for instance *gd1*, *gd2*, *gd3* and the class *GuardDispatch* in Figure 1 as example.
4. Assume that it shall be possible to operate the system without operators, using only guards, in periods with low traffic (for instance during nights). Propose changes to the system structure in Figure 1, including collaborations, and explain how it is supposed to work.

Question 2. (30%) SDL

1. The process graph for *GuardAgt* given in Figure 5 is without local variables and signal parameters. Make the diagram complete with use of signal parameters and declaration of local variables (you may add the missing information to Figure 6 and include the figure in your answer). Define how *guardID* is initialized in the *Initialize* procedure.
2. Given the SDL process graph for *GuardAgt*, and the collaboration *GiveOrder* in Figure 3, define the behavior of *GuardDispatcher* as an SDL process. The process has local variables of type *GuardArray* and *OrderQ*, see Figure 4. Declare the local variables and how they are used. Assume the data types are defined in *AlarmLib*, and that the operation *FindGuard(GeoPos)* returns the identity of the nearest free guard or null if no guard is free. It is sufficient to define transitions for the signals *Order* and *Free*.
3. Define the behavior of the two roles *gd2* and *gal* in *PosUpdate*, shown in Figure 2, as fragments of SDL process graphs. Declaration and use of local variables and timers shall be included.
4. Show how the fragment for *gal* can be added to *GuardAgt* using inheritance so that it can be executed in every state where a *GuardAgent* is on duty. If necessary, make some transitions in Figure 6 virtual (and enclose the figure).

Question 3. (20%) Miscellaneous

1. Is the behaviour of *GuardAgt* given in Figure 5 input consistent? Identify inconsistencies, if any, and explain how they may be corrected.
2. Which parts of the alarm system are suited for physical distribution, and which parts are best deployed on a shared server? Justify your answers.
3. What are the main advantages of using asynchronous messages as in SDL over synchronous method calls as in Java?
4. Which mechanisms on top of standard Java are needed to realize a distributed SDL system such as the Alarm system? Give a short justification.

Nynorsk (Eksamen utgjør 75% av sluttkarakteren.)

Nokre av oppgåvene referer seg til systemet som er skildra i vedlegg. Studer vedlegget fyst.

Oppgåve 1. (25%) Struktur

1. Anta at alle delane (boksane) i Figur 1 skal være SDL prosesser. Gjør Figur 1 om til eit mest mogleg tilsvarande SDL blokk type diagram. Anta at alle typane er definert utanfor blokk typen i Package *AlarmLib*. Signal på kanalane kan utelatast.
2. Vis typereferansane som inngår i *AlarmLib* som UML klasser. Vi antar her bruk av SDL2000 eller nyare SDL versjonar. Angi lokale variablar som attributt til klassene og vis assosiasjonar mellom klassene med multiplisitet på endepunkta.
3. Forklar forholdet mellom kollaborasjonsroller og klasser. Bruk eventuelt *gd1*, *gd2* *gd3* og klassen *GuardDispatch* i Figur 1 som døme.
4. Anta at det i periodar med log trafikk (natt til dømes) skal være mogleg å kjøre systemet utan operatørar og bare med vektorar. Foreslå endringar i systemstrukturen i Figur 1, inkludert kollaborasjonar, og forklar virkemåten.

Oppgåve 2. (30%) SDL

1. Prosessgrafen for *GuardAgt* gitt i Figur 5 manglar lokale variablar og signalparametre. Kompletter diagrammet med bruk av signalparametre og erklæring av lokale variablar (kompletteringa kan gjerast på Figur 6 som rivs ut og leggst ved svaret). Definer korleis *guardID* initialiseres i *Initialize* prosedyren.
2. Med utgangspunkt i SDL prosessen for *GuardAgt*, og kollaborasjonen *GiveOrder* i Figur 3, definer oppførselen til *GuardDispatcher* som en SDL prosess. Prosessen skal ha lokale variablar av typen *GuardArray* og *OrderQ*, sjå Figur 4. Deklarer de lokale variablane og vis korleis de brukast. Vi antar at data typane er definert i *AlarmLib*, og at operasjonen *FindGuard(GeoPos)* returnerer identiteten på næraste ledige vektorer eller null dersom ingen er ledig. Det er tilstrekkelig å definere transisjonar for signala *Order* og *Free*.
3. Definer oppførselen til de to rollene *gd2* og *gal* i *PosUpdate*, vist i Figur 2, som fragment av SDL prosessgrafar med tilhørende deklarasjonar og bruk av variable og timarar.
4. Vis korleis fragmentet for *gal* ved hjelp av arv kan leggst til i *GuardAgt* slik at det kan utføres i einkvar tilstand der *GuardAgent* er i teneste. Gjer, om naudsynt, nokre av transisjonane i Figur 6 virtuelle (og legg ved figuren).

Oppgåve 3. (20%) Diverse

1. Er oppførselen til *GuardAgt* gitt i Figur 5 inputkonsistent? Påvis eventuelle inkonsistensar og forklar kva som må gjerast for at den skal bli input konsistent.
2. Kva deler av alarm systemet egner seg for fysisk distribusjon, og kva deler bør ligge på en felles server? Grunngeiv svara.
3. Kva er de viktigaste fordelane man oppnår ved å bruke asynkrone meldingar som i SDL, framfor synkrone metodekall som i Java?
4. Kva mekanismar på toppen av standard Java må til for å realisere et distribuert SDL system som Alarmsystemet? Gje ei stutt grunngeving!

Vedlegg/ Appendix

Bokmål

Et alarmsystem

Vi skal her studere et system som tilbyr alarmtjenester til boliger som vist i Figur 1. I hver bolig finnes det en *HomeUnit* med tilknyttede sensorer for innbrudd og brann (sensorene er utelatt i denne oppgaven). Når en alarm inntreffer sender *HomeUnit* en melding med informasjon om type alarm og utløsende sensor til *AlarmDispatcher*. Derfra rutes meldingen videre til en ledig *OperatorAgt*. Dersom ingen operatør er ledig settes meldingen i kø inntil en operatør blir ledig. Operatøren vurderer alarmen og tar beslutning om eventuell utrykning (etter å ha forsøkt å ringe til en oppgitt kontaktperson). Operatøren kan kommunisere med *HomeUnit* ved behov. Ved utrykning sendes meldingen *Order(Location, Alarm)*, fra *OperatorAgt* til *GuardDispatcher*. Denne sørger for at nærmeste ledige vekter (*Guard*) dirigeres til boligen. Vekterene er utstyrt med mobile enheter med GPS og et grafisk brukergrensesnitt hvor det bl.a. er felter for kart og meldinger samt følgende knapper:

- *OnDutyBtn* – markerer at vekter er i tjeneste
- *OffDutyBtn* – markerer at vekter ikke er i tjeneste
- *FreeBtn* – markerer at vekter er ledig for oppdrag
- *ConfirmBtn* – markerer at vekter aksepterer et oppdrag
- *DenyBtn* – markerer at vekter avviser et oppdrag

Når vekteren er i tjeneste (*OnDuty*) vil vekterens geografiske posisjonen sendes inn til *GuardDispatcher* med jevne intervall, slik at systemet vet hvor alle vektere i tjeneste befinner seg.

Figur 2 viser kollaborasjonen *PosUpdate* for periodisk oppdatering av posisjon. Figur 3 viser kollaborasjonen *GiveOrder*. Figur 4 viser de viktigste datatypene i form av UML klasser med attributter og operasjoner. Hver vekter har en *guardID* som brukes som indeks i *GuardArray*. Denne skal initialiseres ved oppstart. Figur 5 viser et SDL diagram for *GuardAgt* der posisjonsoppdateringene er utelatt.

English

An alarm system

We shall here study a system that provides alarm service to residences as described in Figure 1. Each residence has a *HomeUnit* with burglar and fire sensors attached (the sensors are not considered in the questions). When an alarm occurs the *HomeUnit* will send a message containing information about the type of alarm and the sensor giving the alarm to the *AlarmDispatcher*. The *AlarmDispatcher* will forward the message to a free *OperatorAgent*. If no *OperatorAgent* is free, the message will be queued until one becomes free. The operator will then assess the alarm and determine if a guard should be sent to the residence (after first trying to call a contact person on the phone). The operator may communicate with the *HomeUnit* upon need. If a guard shall be dispatched the message *Order(Location, Alarm)* is sent from the *OperatorAgent* to the *GuardDispatcher*, which orders the nearest free guard to go to the residence. The guards are equipped with mobile units with GPS and a graphical user interface displaying maps and messages and the following buttons:

- *OnDutyBtn* – marking the guard as on duty
- *OffDutyBtn* – marking the guard as off duty
- *FreeBtn* – marking the guard as free to take orders
- *ConfirmBtn* – marking that an order is accepted
- *DenyBtn* – marking that an order is denied

Whenever a guard is on duty, the guard's geographical position will be sent to the *GuardDispatcher* at regular intervals, so that the system knows where all guards on duty are positioned.

Figure 2 shows the *PosUpdate* collaboration for periodic position updates. Figure 3 shows the *GiveOrder* Collaboration. Figure 4 shows the most important data types in the form of UML classes with attributes and operations. Each guard has a *guardId* that is used as index in *GuardArray*. The *guardId* shall be initialized upon startup. Figure 5 gives an SDL diagram for the *GuardAgt* with position updates omitted.

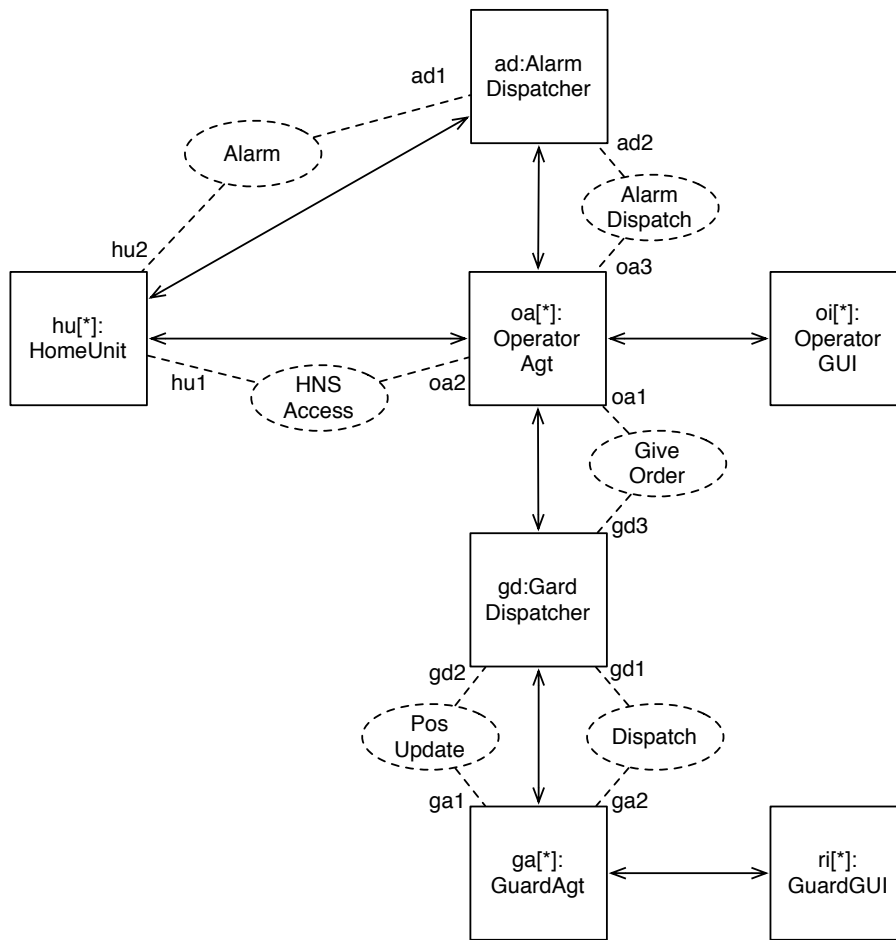


Figure 1 The part structure of the alarm system

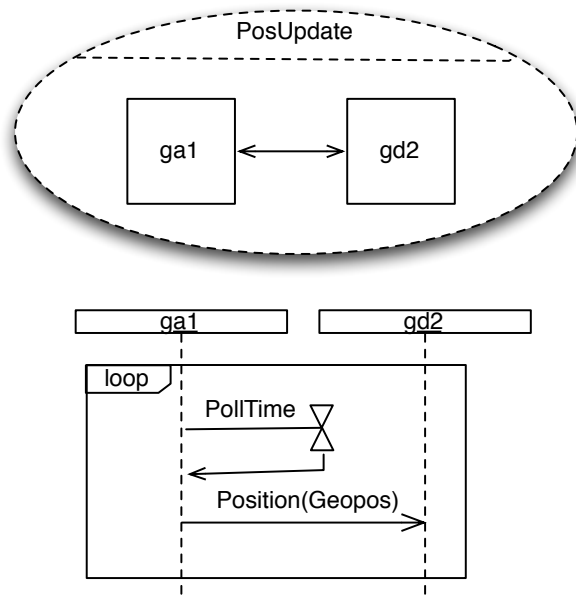


Figure 2 The PosUpdate collaboration

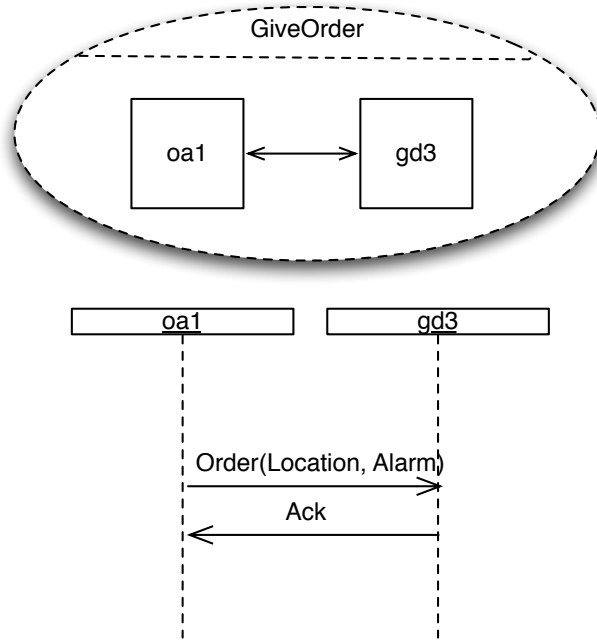


Figure 3 The GiveOrder collaboration

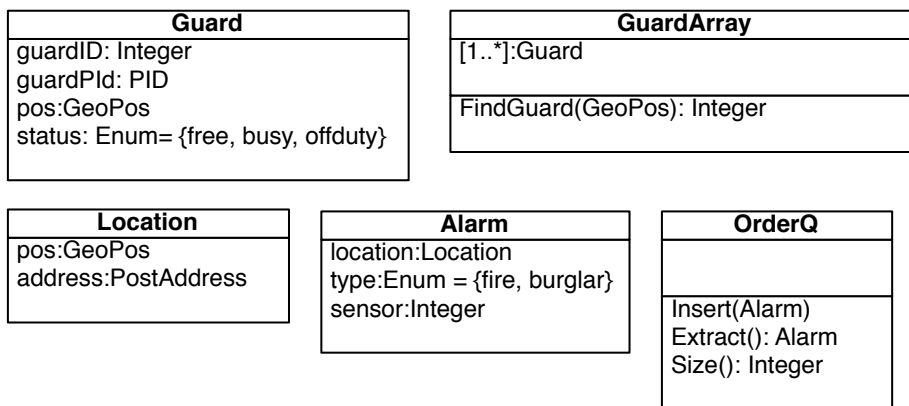


Figure 4 Important data types represented as UML Classes

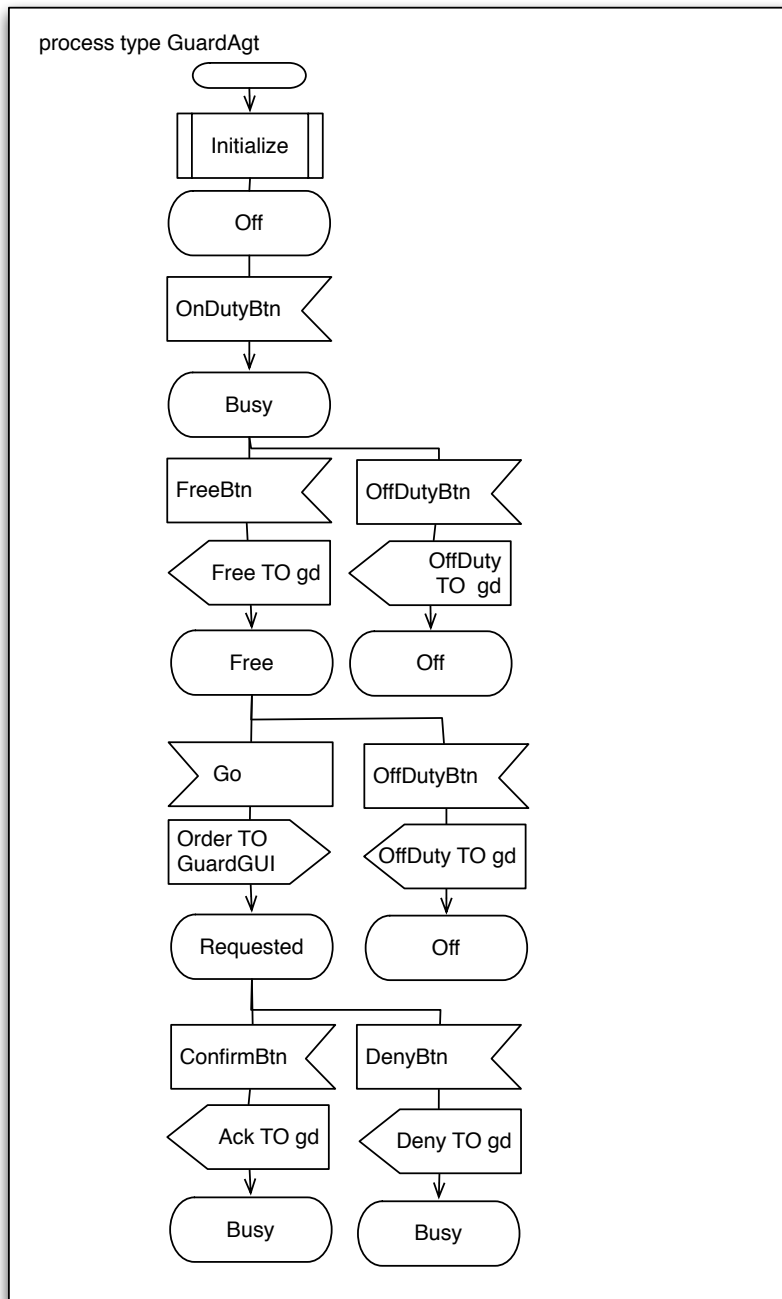


Figure 5 Process type GuardAgt, position Update not included.

Subject: TTM4115 Dato: 29.09.2010

Candidate number:

process type GuardAgt

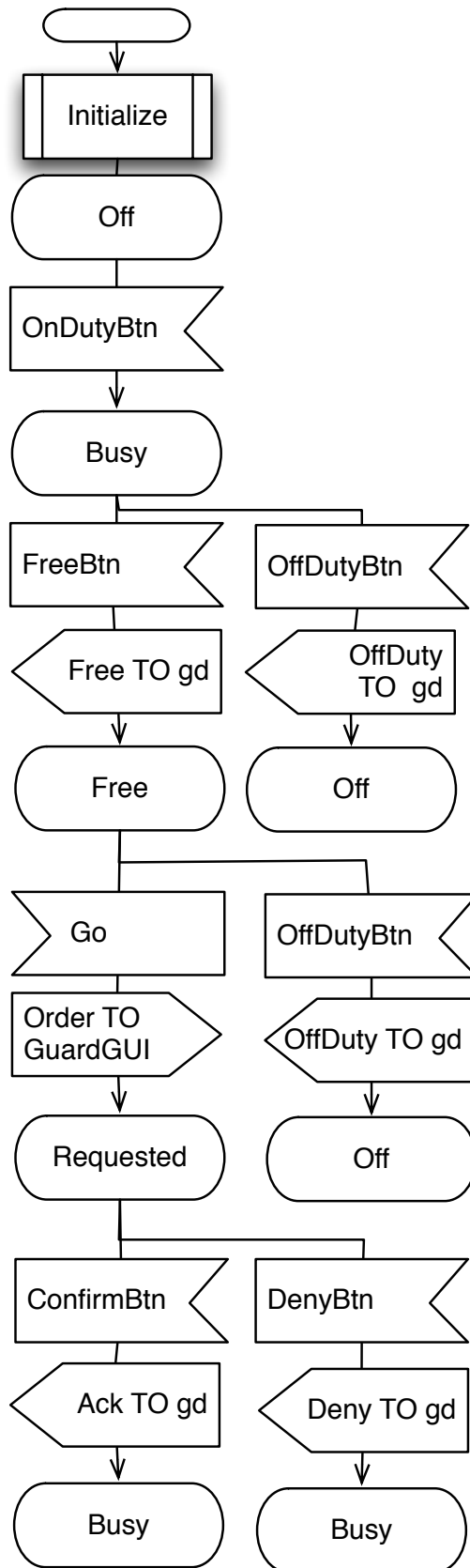


Figure 6 Copy to be completed and handed in