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/Sensurdato[1]:	**21. juni 2010**

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

Bokmål (Eksamen utgjør 75% av sluttkarakteren.)

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 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 vektere. 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 vekter 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 *ga1* 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 *ga1* 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

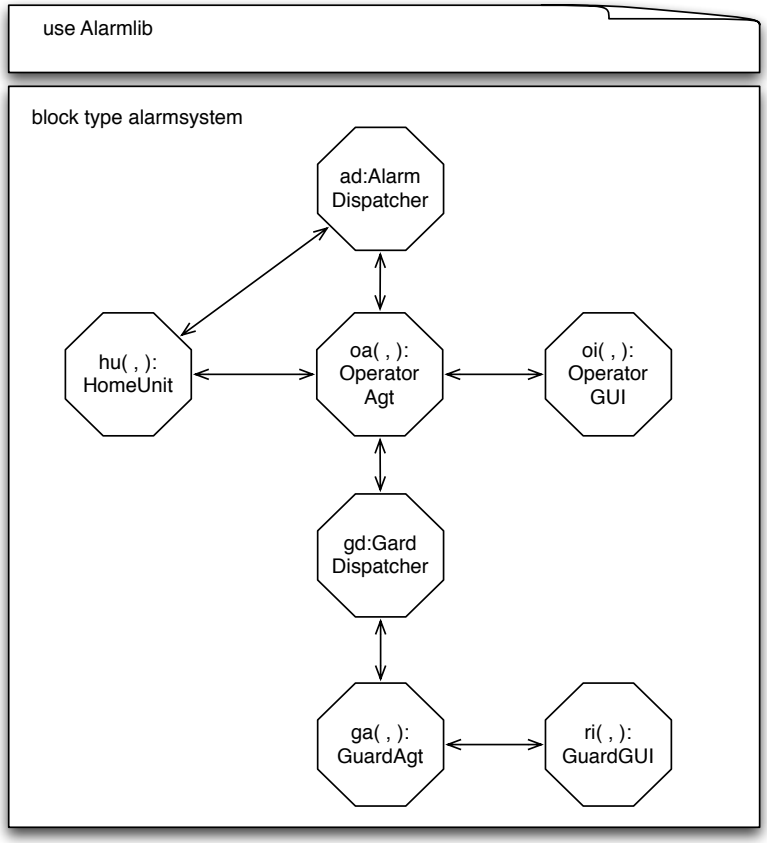
1. Er oppførselen til *GuardAgt* gitt i Figur 5 inputkonsistent? Påvis eventuelle inkonsistenser og forklar hva som må gjøres for at den skal bli input konsistent.
2. Hvilke deler av alarm systemet egner seg for fysisk distribusjon, og hvilke deler bør ligge på en felles server? Begrunn svarene.
3. Hva 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.

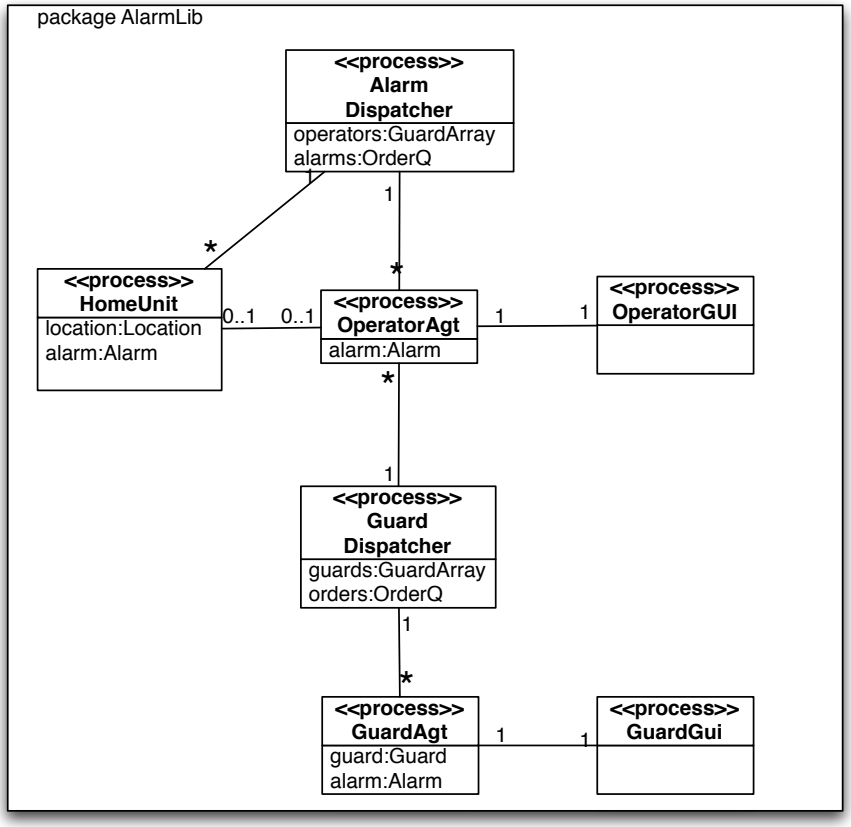
Question 1. (25%) Structure

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.



Using UML multiplicity, e.g. [*] in stead of SDL (,): -0,5
Missing package Use: -1
Including type references that should be in AlarmLib: -0,5
Using UML part structure, not SDL: -5
Missing multiplicity: -1
Missing instance names: -1
Not using process symbols: -1
Missing components (e.g. GUI): -0.5
Process graph in stead of block type: -10

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.

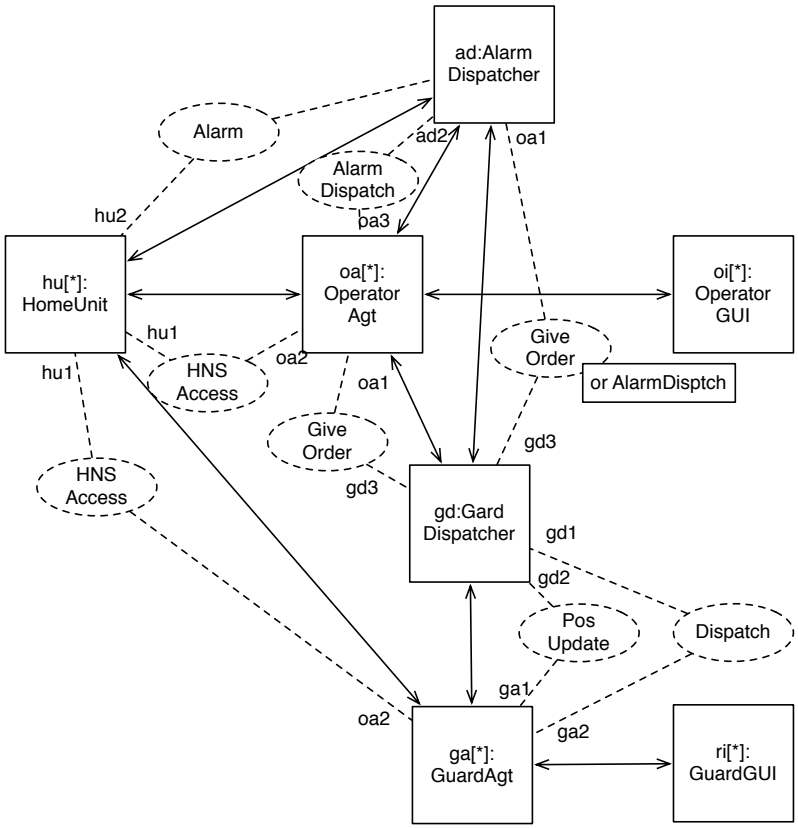


The main point here is to show the stereotyped classes with associations and attributes. Getting all the attributes right is not needed, but the attributes shown should be sensible and contain the most important ones.
Missing the type references, showing only data: -6
Missing given attributes such as Alarm: -0,5
Insensible attributes: - 0,5
No sterotypes: -0,5
No associations (and multiplicities): -2
No multiplicities: -1
Minor mistakes, e.g multiplicity mistake: -0,5
No attributes: -2
SDL type references, not UML: -5
Objects instead of types: -1.5

3. Explain the relationship between collaboration roles and classes. Use for instance *gd1*, *gd2*, *gd3* and the class *GuardDispatch* in Figure 1 as example.

Roles define partial class properties that are visible in the collaboration the role belongs to. A class has to be compatible with all its roles, i.e. behave according to the roles when the corresponding collaborations are active.
Lack of precision, but otherwise OK: -0.5 .. -1
Not stated that a Class can play many roles: -2
Not explaining roles but collaborations: -3

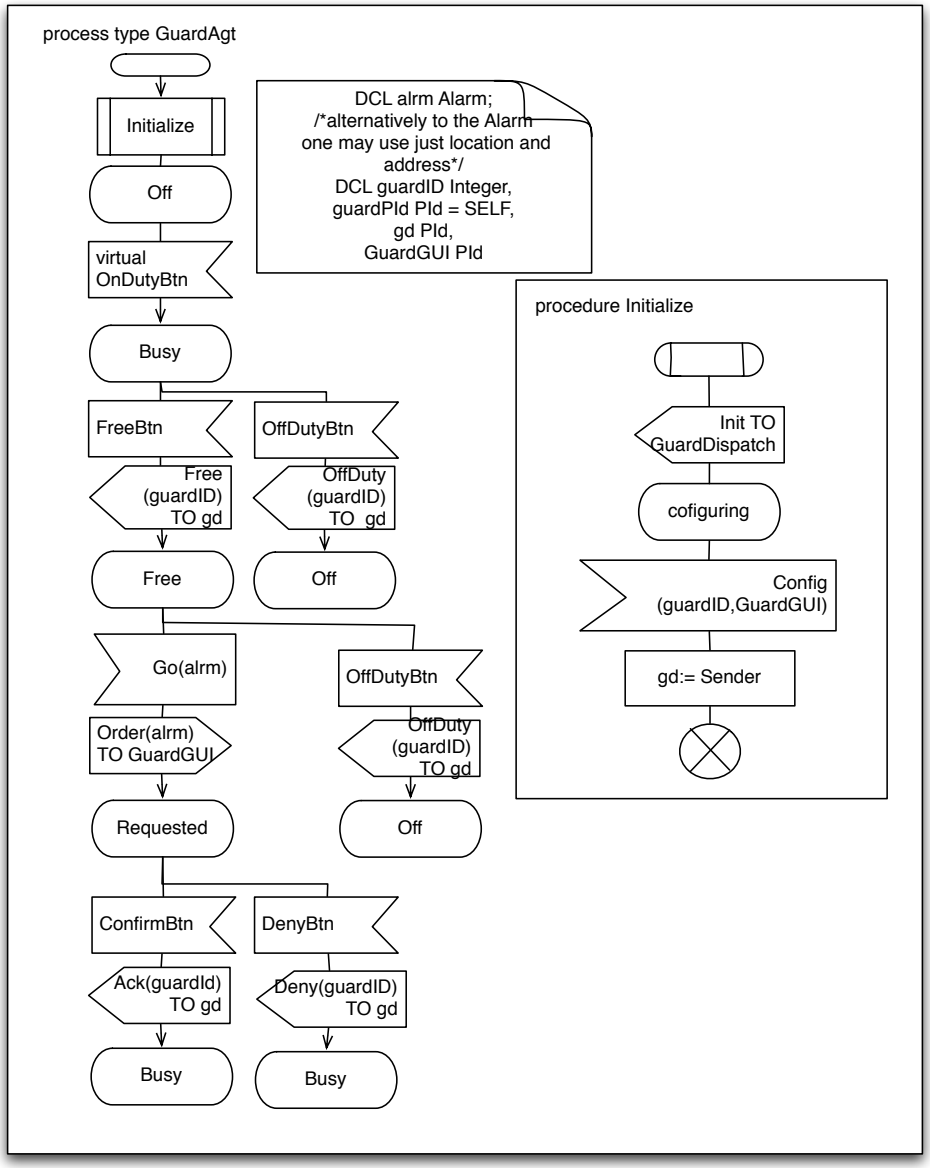
4. Assume that it shall be possible to operate the system without operators, using only guards, in periods with low traffic (for instance during nights). Propose changes to the system structure in Figure 1, including collaborations, and explain how it is supposed to work.



The simplest options are to add either the GiveOrder collaboration or the AlarmDispatch collaboration between the AlarmDispatcher and the GuardDispatcher. When no operators are on duty, the AlarmDispatcher simply forwards the alarm to the GuardDispatcher. It is then up to the selected Guard to call the customer and decide if it is necessary to go to the residence. If the Guard shall be able to access the HomeUnit, we need a link between GuardAgt and HomeUnit, and the GuardAgt must be able to play oa2 in the HNSAccess collaboration.
Collaborations missing in Figure or explanation: -2
Not re using existing collaborations: -1
Letting HomeUnit send alarm to GuardDispatcher is a possibility, but then the HomeUnit needs to be informed about availability, so this is not so good solution: -1
Missing HNS Access for GuardAgent: -0,5
Attaching HNSAccess to GuardDispatch is wrong since then GD will then need to sessions, which is better done by GA: -0,5 .. -1
Adding a new process can be OK, but not needed: -1
Low precision, unclear description:-0,5.. -1
Adding unnecessary allocators: -1,5

Question 2. (30%) SDL

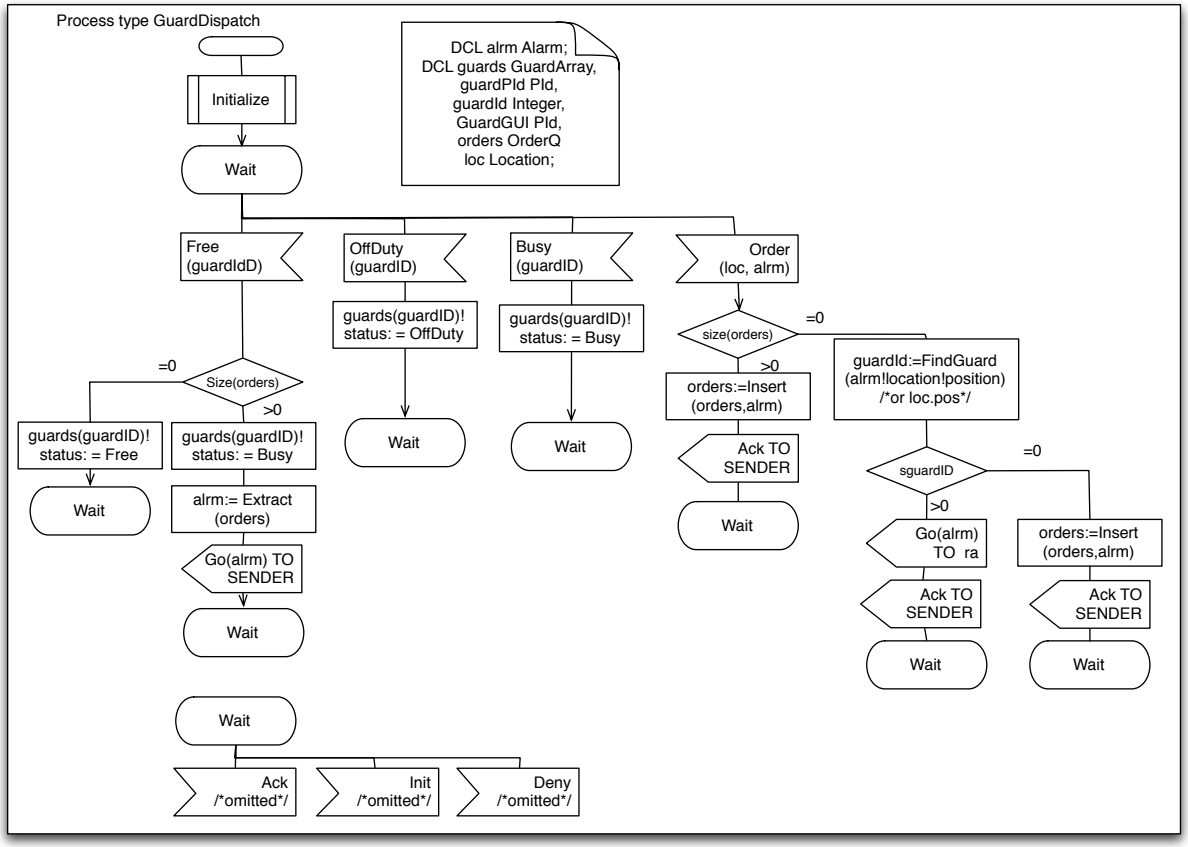
1. The process graph for *GuardAgt* given in Figure 5 is without local variables and signal parameters. Make the diagram complete with use of signal parameters and declaration of local variables (you may add the missing information to Figure 6 and include the figure in your answer). Define how *guardID* is initialized in the *Initialize* procedure.



The main point here is to demonstrate understanding of declarations, use of signal parameters and initialization of identities.
In this solution we use the Alarm type, but we might also use the location and address as stand alone variables. We assume here that gd and GuardGUI are PId variables that we initialize. Alternatively to using gd as PId we might have used gd or GuardDispatch as name. It is assumed that GuardDispatch makes a new entry in GuardArray for each Init signal it receives from a newly started Guard, and then returns the guardID and the GuardGUI to the Guard, assuming that the GuardGui is known to the GuardDispatcher. Alternatively the GuardAgt could create the GuardGui and get the PId that way. We did not ask about the GuardGUI PId initialization so this may be omitted. One transition has been made virtual as partial answer to Q2.4.
Init OK explained, but no diagram to define the behaviour:-1,5
No init or meaningless init: -3.5
Init from GUI: -0,5-1
No DCL at all: -3
No parameter use at all: -3

Missing alarm parameter in Go: -1
No guardID parameter use: -1
guardId used as PID: -1

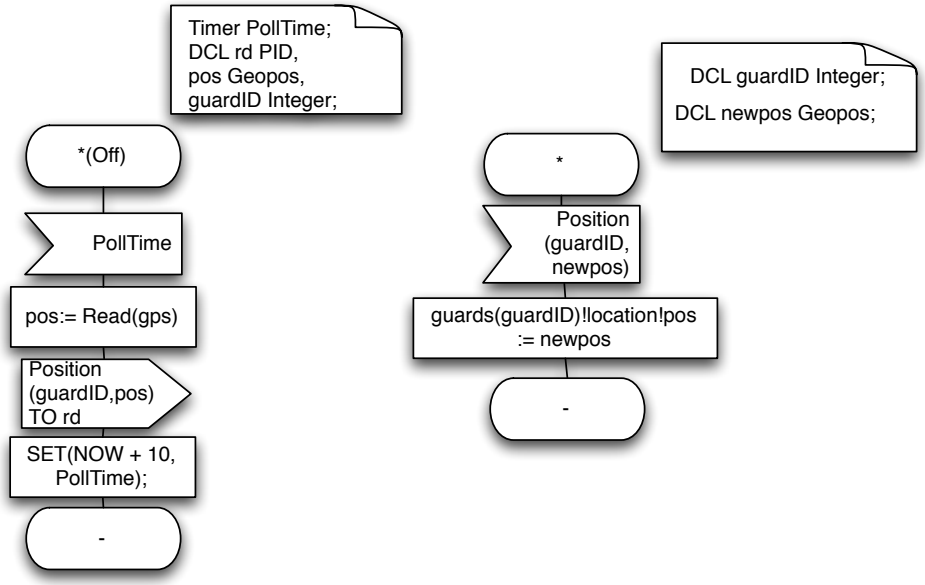
2. Given the SDL process graph for *GuardAgt,* and the collaboration *GiveOrder* in Figure 3, define the behavior of *GuardDispatcher* as an SDL process. The process has local variables of type *GuardArray* and *OrderQ,* see Figure 4. Declare the local variables and how they are used. Assume the data 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*.



The main point here is to demonstrate data usage, and allocator behavior. We have not spent much time on the SDL syntax, so deviations are OK as long as the expressions are precise. Using Java syntax is OK. Here we are only using the Alarm parameter in the Order, because the Alarm contains the location. It is OK to use both the Location and the Alarm of course.
Missing DCL: -1
Type errors in DCL or use: -1
Missing decisions: -1
Missing TO PId (for GA) in Go: -1
Serious SDL error: -1
FindGuard as signal: -1
No data use at all: -3
No queue operations: -0.5-1
Missing allocation: -1
Unprecise data operations:-0.5-1
Missing operations: -0,5-1

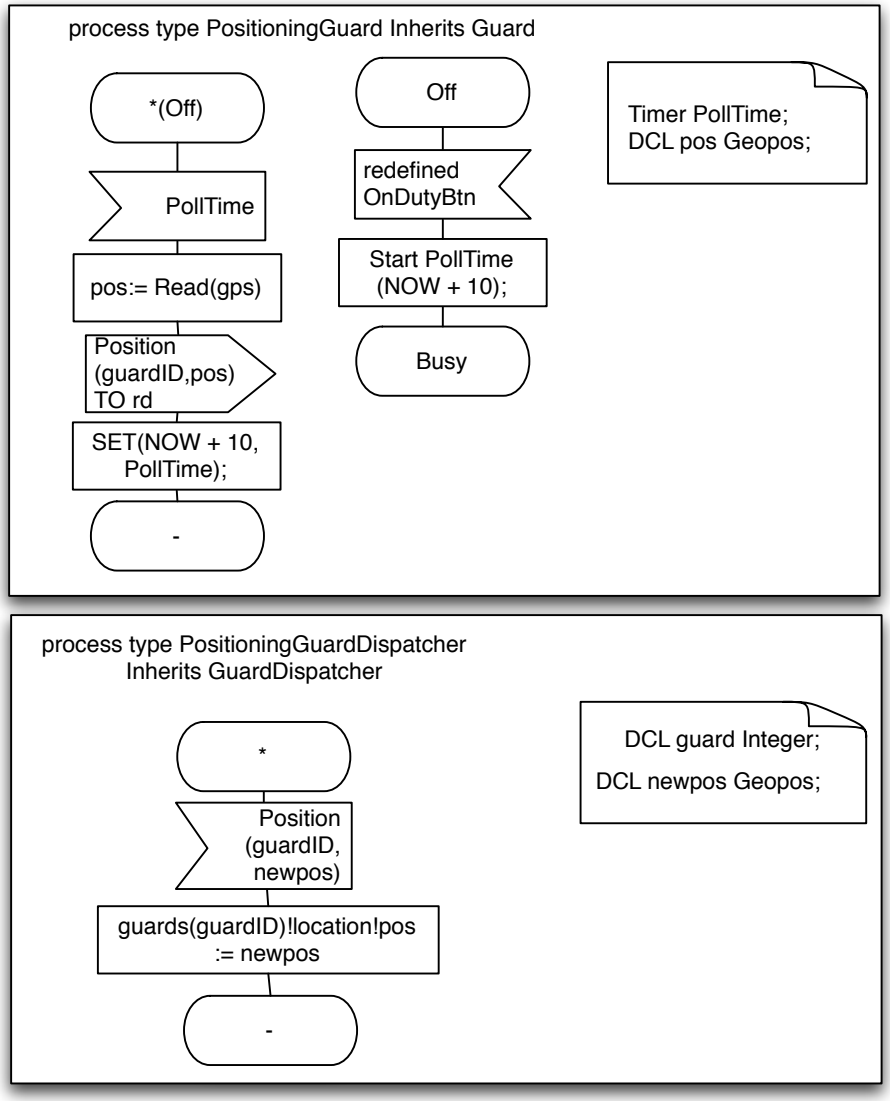
Receiving Ack and Deny in session specific states: -0.5 (because that would interfere with other sessions)

3. Define the behavior of the two roles *gd2* and *ga1* in *PosUpdate,* shown in Figure 2, as fragments of SDL process graphs. Declaration and use of local variables and timers shall be included.



The main point here is use of timers, and to map MSC to SDL. The particular state names are not so important here (but will be in Q2.4)Some of the DCL given here belongs to the enclosing process.
Missing gd2 entierly: -3
Missing timer completely: -1,5
Using None signals for timers: -1,5
Using decisions for timers: -1,5
Missing start timer: -1
Missing DCL: -1
Imprecise data use: -0,5
No position read from GPS: -0,5

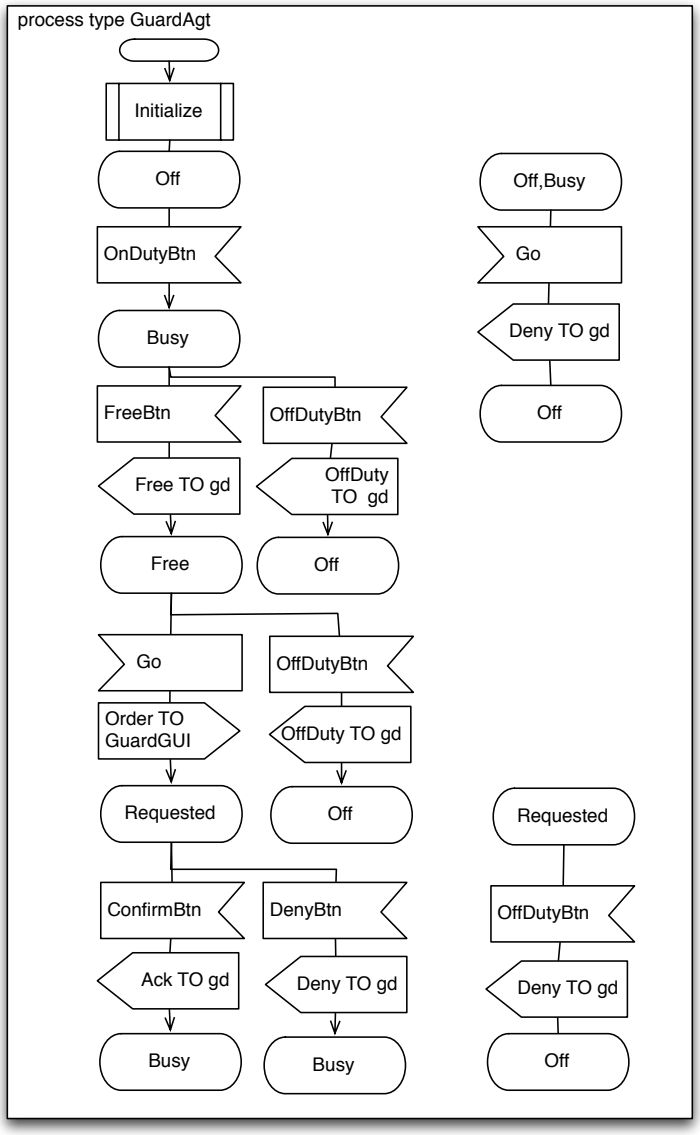
4. Show how the fragment for *ga1* can be added to *GuardAgt* using inheritance so that it can be executed in every state where a *GuardAgent* in on duty. If necessary, make some transitions in Figure 6 virtual (and enclose the figure).



The main point here is inheritance and virtuality. The variables of the supertype should be used where possible, hence we have omitted some that were declared in Q2.3. Note the redefined transition to start the timer. We might also have added reset timer where the guard goes off duty, but we assume simply that the timer will be discarded here.
Inheriting the wrong way: -2
No inheritance, but otherwise OK: -5
Copying the supertype: -2
Using Redefine in stead of Inherit: -2
Minor heading errors, such as omitting "type": -1
Working only in a few of the specified states: -2

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.



It is not input consistent. In the ga2 role, the interface to the GUI is hidden, and the transition from state Free to state Off is invisible and not input consistent. One must be prepared to receive Go in the Off state to make it input consistent. Adding a transition triggered by Go in the Off state, makes the invisible transition from Off to Busy also inconsistent. The solution is to handle Go in both the Off and Busy state.
In the role towards the GUI, the Go transition is invisible, and also inconsistent. The remedy is to add a transition triggered by OffDuty to the Requested state.
Correct answer without any explanation or fix: 5
Only identifying mixed initiatives, not inconsistencies: 3
Not finding one of the primary errors, e.g. Go in Off: -2
Unclear explanation: -1
Not finding or handling secondary inconsistency, i.e. Go in Busy: -0,5
Finding the wrong inconsistencies: -1
Not correcting the inconsistencies: - 1

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.

First of all the HomeUnits will naturally be situated in the residences and thus be distributed, in the same manner as the residences. Secondly the GuardAgents and the GuardGUI should move around with the Guards, and therefore be distributed on mobile devices the Guards may take along. As a minimum the GuardGUI should be local to the guard's device. The singleton processes the AlarmDispatcher and the GuardDispatcher are shared and central to the system, not needing to move or being distributed so they should be on a shared server or possibly one server each, if load demands so. Each pair of operatorAgents and operatorGUI should primarily be on the same device. They may run on a central server, or they may run on devices distributed to the operator locations. They may optionally be separated, but this is not considered as good.
Not distributing the HU and the GA: -2
Separating GUI from Agent: -1
Not centralizing the AD and the GD: -2
Wrong explanation:-2
Omitting to mention objects: -0.5
Failing to mention that OA can be either on server or distributed: -0,5
Sensible general observations, but not identifying any possible deployments:3

3. What are the main advantages of using asynchronous messages as in SDL over synchronous method calls as in Java?

The main advantages are

- *Message sending does not block the sender, so the sender may proceed immediately. This means more elastic coupling.*
- *Messages can easily be distributed, while method calls need additional mechanisms such as stubs and skeletons and are mapped to messages for transfer.*
- *Messages can support general interaction patterns while method invocations are restricted to three structured communication patterns where the sender can remain blocked until the invocation returns.*
- *Messages allow more flexible scheduling than the blocking behavior of method calls and this will in many cases lead to less overhead and provide better performance in a distributed real-time system.*

Failing to mention generality:-2
Failing to mention Distribution: -2
Failing to mention scheduling flexibility/efficiency/performance: -1

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.

One needs to support the basic SDL mechanisms not directly available in Java:

- *Timers (1 point)*
- *Message sending and message reception, i.e. a message system interface. (2 points)*
- *Message addressing and routing mechanisms. (1 point)*
- *State machine behavior. (1 point)*
- *Inheritance mechanisms for behavior. (0.5)*
- *Process scheduling. (0.5)*

One way to provide this is by a runtime system such as ActorFrame. In addition one normally needs an operating system.
Only mentioning the need for HW, operating system and communication networks: max 3.
Only mentioning runtime system/framework such as ActorFrame: max 4

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 versjoner. 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 vektarar. Foreslå endringar i systemstrukturen i Figur 1, inkludert kollaborasjoner, 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 leggast 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 vekter 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 *ga1* i *PosUpdate,* vist i Figur 2, som fragment av SDL prosessgrafar med tilhørande deklarasjonar og bruk av variable og timarar.
4. Vis korleis fragmentet for *ga1* ved hjelp av arv kan leggast 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? Grunngjev 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 grunngjeving!

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 inntrer 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* – markere 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.

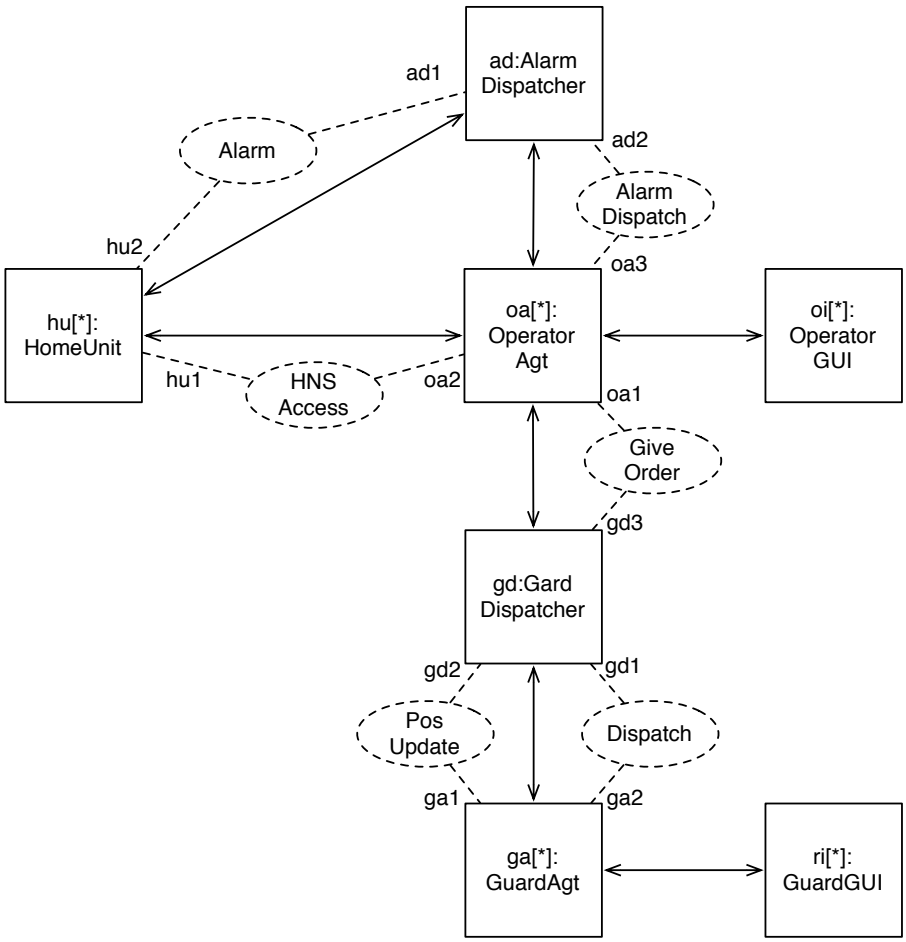


Figure 1 The part structure of the alarm system

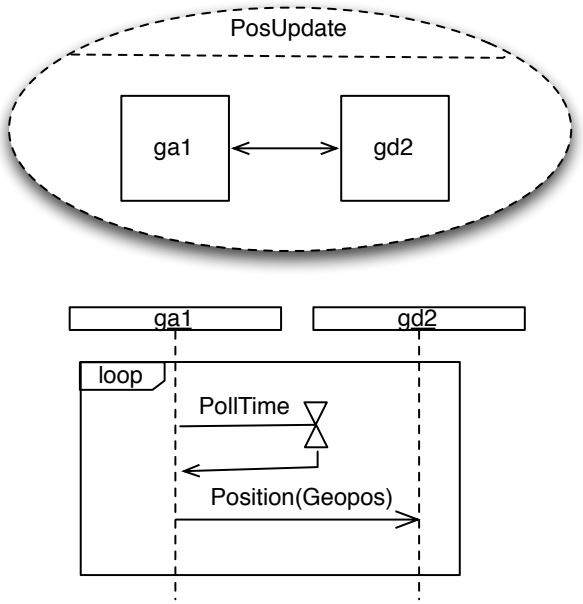


Figure 2 The PosUpdate collaboration

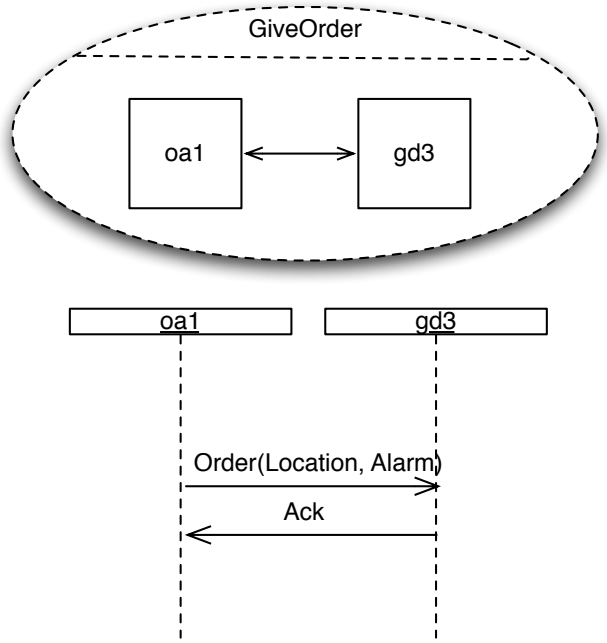


Figure 3 The GiveOrder collaboration

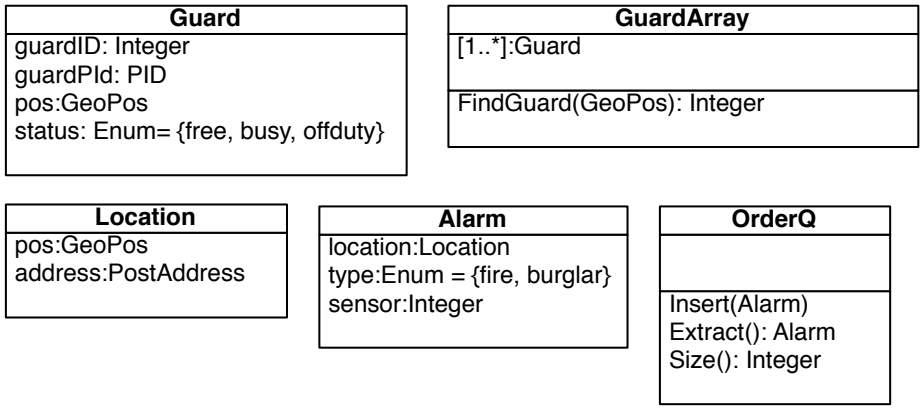


Figure 4 Important data types represented as UML Classes

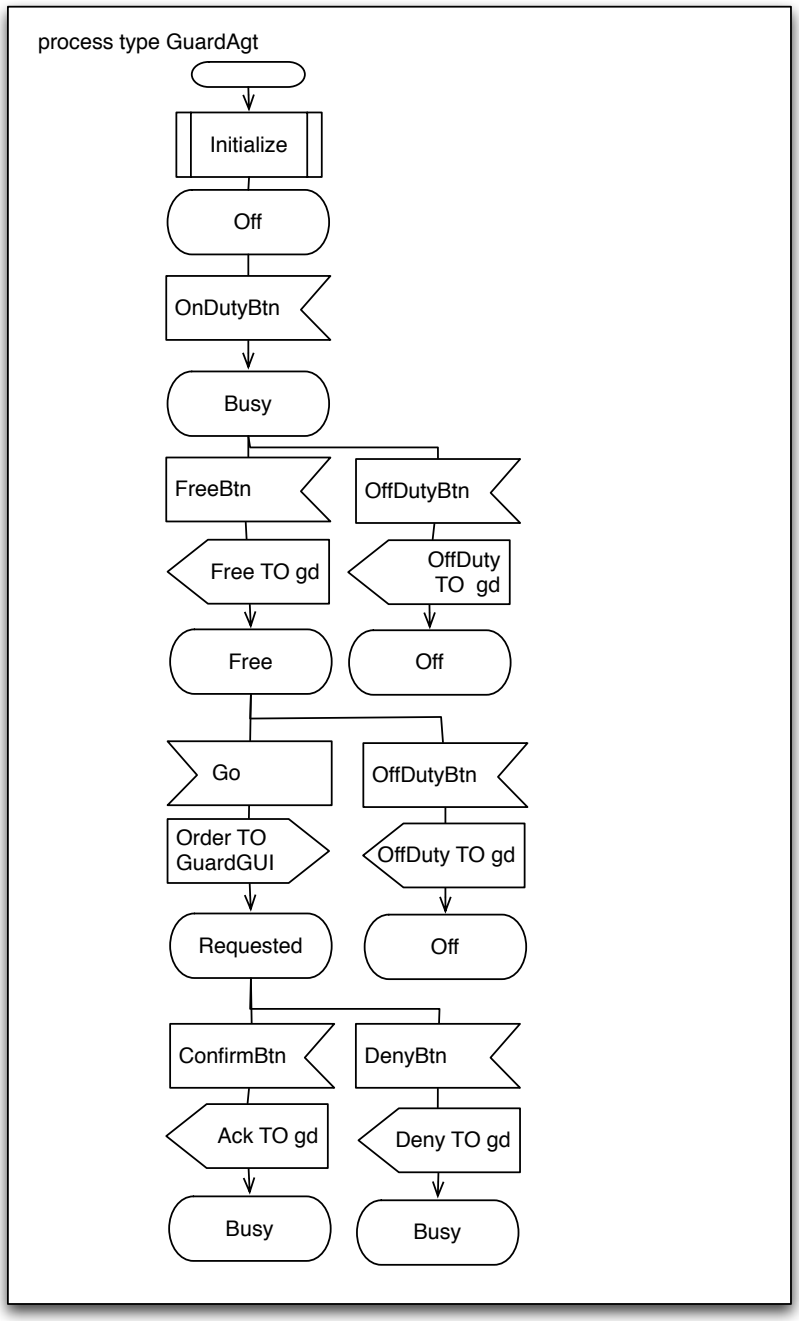


Figure 5 Process type GuardAgt, position Update not included.

Subject: TTM4115 Dato: 29.09.2010 Candidate number:

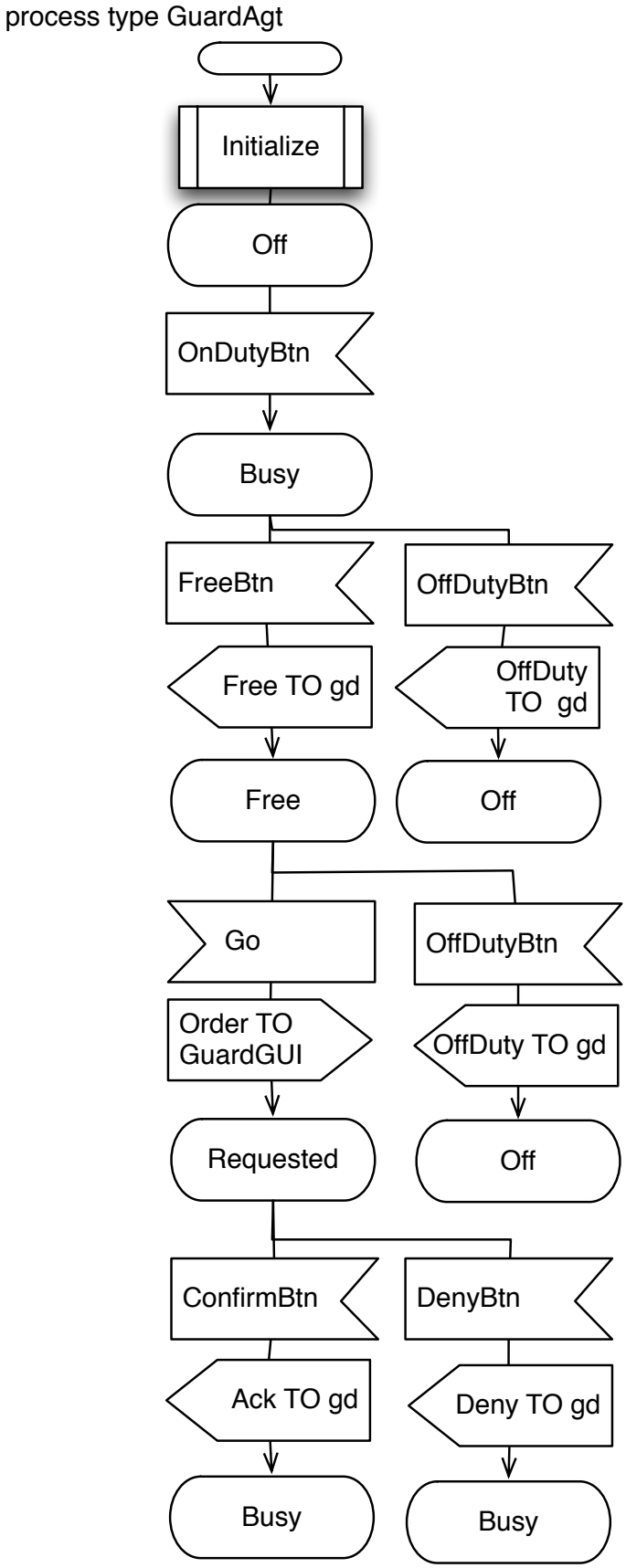


Figure 6 Copy to be completed and handed in