Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

SOLUTION PROPOSAL TO EKSAMENSOPPGAVE I TTM4115 – SYSTEMERING AV DISTRIBUERTE SANNTIDSSYSTEMER

The solution proposal is given in English only!

Faglig kontakt under eksamen:	Rolv Bræk
Tlf.:	415 44 605
Eksamensdato:	28. mai 2008
Eksamenstid:	09:00-13:00
Studiepoeng:	7,5 SP
Tillatte hjelpemidler:	A: Alle trykte og håndskrevne hjelpemidler tillatt. Alle kalkulatorer tillatt
Språkform:	
Antall sider bokmål:	1
Antall sider nynorsk:	0
Antall sider engelsk:	1
Antall sider vedlegg:	4
Sensurdato[1]**:**	**26. juni 2007**

[1] Merk! Studentene må primært gjøre seg kjent med sensur ved å oppsøke sensuroppslagene. Evt. telefoner om sensur må rettes til sensurtelefonene. Eksamenskontoret vil ikke kunne svare på slike telefoner.

Bokmål (Eksamen utgjør 75% av sluttkarakteren.)

Oppgavene referer seg til systemene som er beskrevet i vedlegg. Studer vedlegget først. Løsningsforslaget finnes under den engelske delen

Oppgave 1. (25%) System design

1. Hvordan stemmer systemstrukturen beskrevet i Figure 1 og Figure 2 overens med reglene i SDL metoden? Hvilke regler er brukt.
2. Oppførselen til kollaborasjonen C*onsultation* er definert med MSC i Figure 4. Definer oppførselen til rollen *docpat* med SDL slik at den direkte tilsvarer oppførselen til *docpat* beskrevet i Figure 4.
3. Kontroller at rollen for *docpat* som du laget under punkt 2 er input konsistent. Gjør de endringene som eventuelt er nødvendige for at den skal bli input konsistent.
4. Hva er sammenhengen mellom roller i kollaborasjoner og klasser (*Classes*) i UML 2; hva betyr det at en rolle, som *docpat* er bundet til en klasse som *DoctorAgent*?

Oppgave 2. (25%) Oppførsel og endringer

1. Gjør kort rede for virkemåten til *RoleRequest* protokollen i *ActorFrame*.
2. I *Telemedicine* systemet skal vi bruke en variant av *RoleRequest*, kalt *AgentRequest,* se Figure 3 a) og Figure 5. Denne tar hensyn til at agentene er persistente og at forespørsler stilles i kø når det ikke er ledige agenter. Beskriv noen typiske forløp av oppførselen til *AgentRequest* med MSC.
3. Definer en SDL prosess for *agtAllo* rollen i *AgentRequest*. Anta at det finnes en datatype *Queue*, med operasjonene *insert*, *extract* og *length*, som kan benyttes.
4. Vi skal legge til funksjonalitet for at pasientene i stedet for å bruke pasientterminaler kan ringe inn via vanlige telefoner og snakke med en resepsjonist som gjør registreringen. Det skal være mulighet for at flere resepsjonister kan dele lasten. Foreslå en struktur og forklar hvordan registreringen blir gjort.

Oppgave 3. (25%) Diverse

Figure 6 viser to systemer *VM1* og *VM2* beskrevet med prosessalgebra, CCS.

1. Foreta ekspansjon av utrykket for *VM2 = IF || CM || TM*
2. Sammenlign uttrykkene for *VM1* og *VM 2*. Er de observasjonsekvivalente eller ikke? Begrunn.
3. Finn uttrykket for en omgivelse *E* for *VM1* som er slik at alle deler av *VM1* blir benyttet uten at det oppstår vranglås. Ekspander *E || VM1*.
4. Forklar hensikten med Tagger (Tags) i ASN.1, og gi et par eksempel på bruken.

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

The questions refer to the systems described in the appendix (Vedlegg). Study the appendix first. Each sub-question is given points 0-10.

Question 1. (25%) System design

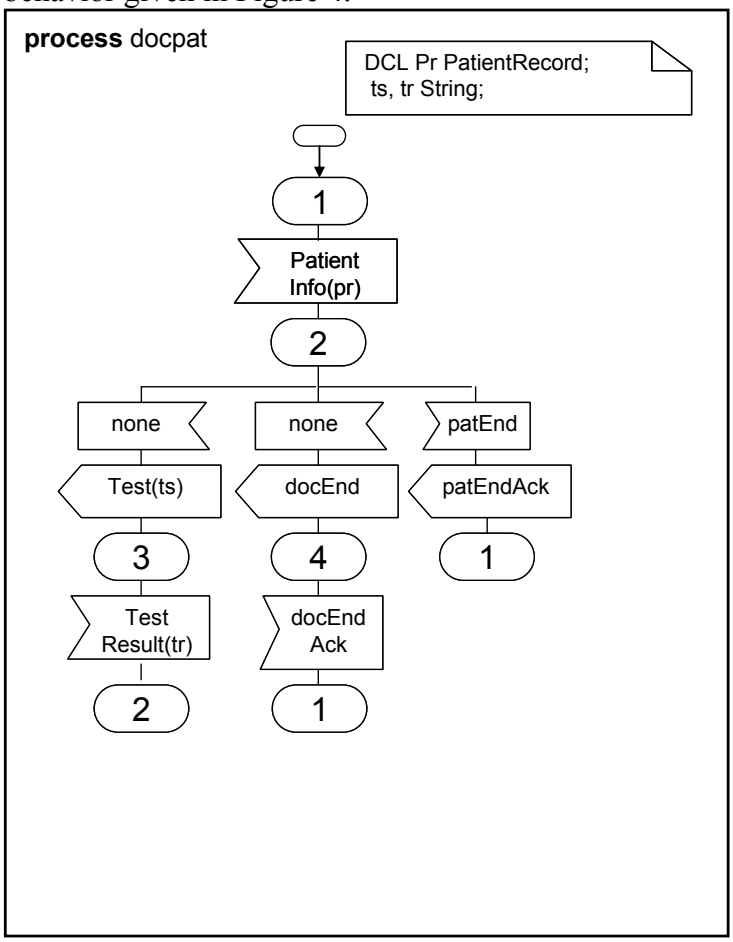
1. How does the system structure described in Figure 1 and Figure 2 comply with the rules of the SDL method? Which rules apply?

The system is consistent with these main rules:

- *the mirroring rule s-rule …*
- *the resource allocation rule s-rule …*

Other more detailed rules such as the block purposes apply too.
Mangler en av disse reglene -2. Nevne tilsvarende regler: f.eks omgivelser og paralellitet i stedet for speiling -1. Nevne regler som ikke gjelder: -1. Ingen regler max 4. Misforstått og gitt syntaxregler i stedet: max 4.

2. The behavior of the C*onsultation* collaboration is defined in Figure 4 using MSC. Define the behavior of the *docpat* role using SDL such that it directly corresponds to the *docpat* behavior given in Figure 4.



Missing data DCL: -0,5. Missing none input: -1. Test not optional: -1,5. SDL errors: 1-2. Stop in stead of looping is OK. Showing other roles in stead of None: -1. Stateless behavior

and/or allowing other sequences than specified: -1,5-2 Unspecified behavior: -1 Test guarded by decision: -0,5-1

3. Check that the *docpat* role you just designed (answer to point 2) is input consistent. Make the corrections necessary to make the role behavior input consistent.

Mixed initiatives occur in the state 2. One must then assure that the collision is properly handled. The none transitions are not input consistent, and therefore it is necessary to add reception of patEnd to state 3 and 4. In this case the goal is the same for both initiatives, and therefore they may be treated the same way with next state 1.
Not found inconsistency: -2-5. Not specified next state: -1. Saying there is no inconsistency because the solution of 1.2 is stateless is OK. Not corrected the diagram: -3 Found too many inconsistencies: -1, Adding none transition to correct inconsistency: -2

4. What is the relationship between collaboration roles and Classes in UML2; what does it mean that a role such as *docpat* is bound to a Class such as *DoctorAgent*?

Roles define properties that classes shall have. The class may have other properties in addition to the role properties. A class is said to be compliant with a role if it satisfies the properties. Role behaviors may be seen as projections of class/actor behaviors. An actor is compatible with a collaboration role if its projected role behavior contains the collaboration role behavior. This point must be judged mildly, as we have not spent much time on this. We have spent time on projections and input consistency so they should at least understand that collaboration roles can be used to model interface roles.
Nevne at roller modellerer egenskaper som klasser skal ha: +5- 8
Bare at de modellerer egenskaper: +3-5
At roller modellerer grensesnitt: + 3-5
Klargjøre at klasser kan spille en eller flere roller: +8
At disse kan være på forskjellige grensesnitt og i forskjellige kollaborasjoner:+9
Også nevne kompatibilitet: +2
At roller er anonyme objekter: +2

Question 2. (25%) Behavior and changes

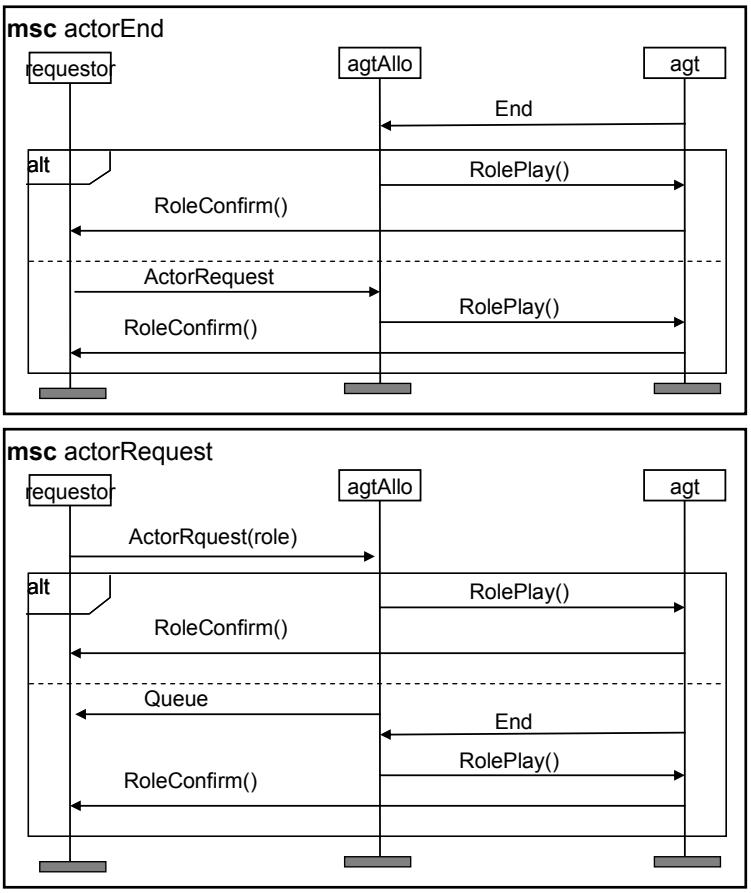
1. Explain briefly how the *RoleRequest* protocol in *ActorFrame* works.

The RoleRequest collaboration (protocol) is used to request and establish dynamic links between actors. The requesting actor issues a RoleRequest message to a requested actor indicating the requested role. The requested actor may respond in two ways:

2. *If it is able to play the role; it creates an inner actor that can play the role and initiates the role by sending a RolePlay message to it. The inner actor then responds back to the requestor by sending a RoleConfirm message.*
3. *If it is not able to play the requested role it simply responds by sending a RoleDenied message.*
4. *When an actor is finished it sends roleRelease and tops itself.*

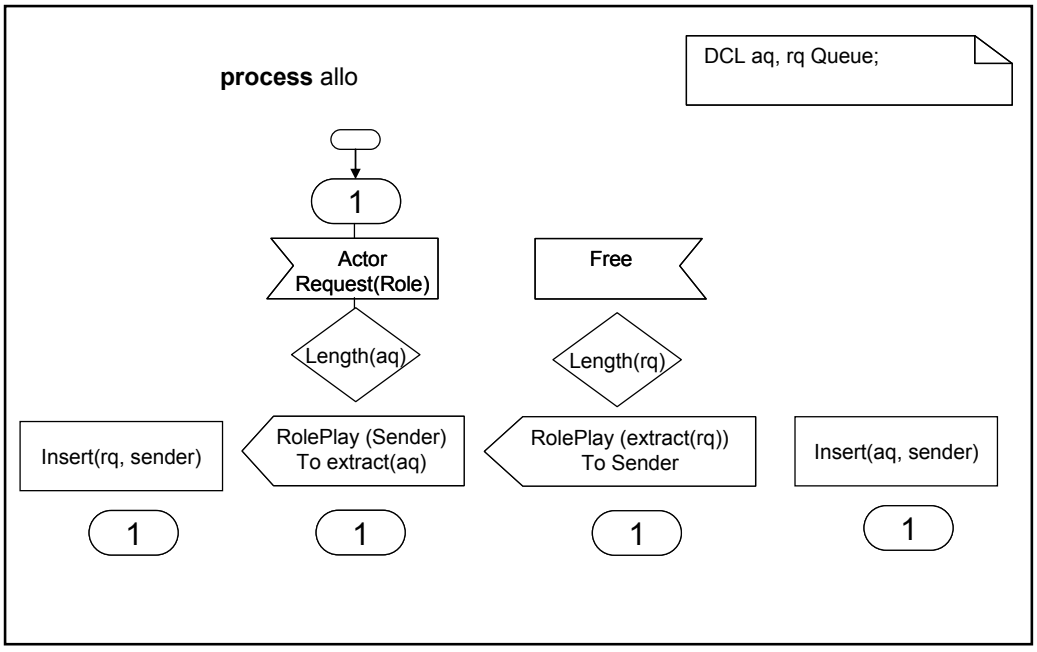
Failing to mention roleRelease: -0,5. Failing to mention other messages: -0,5-1. Saying that requests are queued in stead of denied is OK since this is said on a foil. Failing to mention creation of inner roles: -1. A good description without mentioning signals: 7,5 max. Just giving the overall purpose; to create links and invoke roles: +5 max

5. In the *Telemedicine* system we use a variant of *RoleRequest*, called *AgentRequest,* see Figure 3 a) and Figure 5. It takes into account that the agents are persistent and that requests shall be queued when no agents are available. Describe some typical cases of *AgentRequest* behavior using MSC.



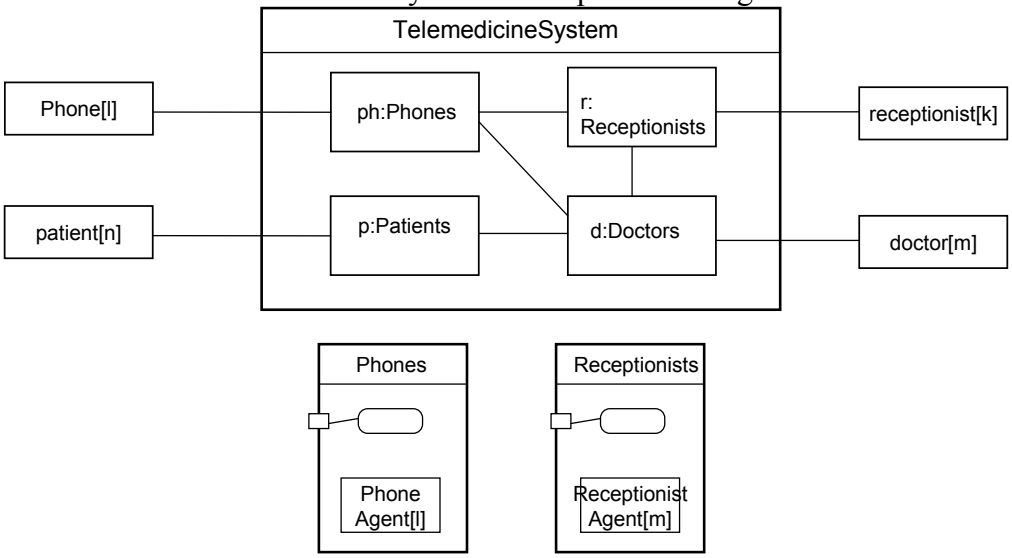
Other signal names are OK. We can accept that the student uses actor names and not role names. Allocator not properly used, i.e. asking the agent for avaiability in stead of letting the agent notify the allo when available: -2 Not showing the queue case:-1. Not showing how the queue case ends: -1. Not showing RoleRelease at all: -1. MSC errors: 0,5-1

6. Define an SDL process corresponding to the *agtAllo* role of *AgentRequest*. You may assume a datatype *Queue*, with the operations *insert*, *extract* and *length* that may be used.



Not consistent with MSC: -2. Incorrect queue treatment: -1-1,5 SDL errors: -0,5-1 Incomplete behavior: -1-3. Missing dcl: -1 Missing data operations: -1 Logical errors: -1No doctor queue: 0,5-1Can only handle one request at the time- 1,5. Only one request and one doctor: -2

7. We shall now add functionality that enables patients without patient terminals to call in using ordinary telephones and talk to a (human) receptionist that takes care of the registration. It shall be possible to distribute the load among several receptionists. Propose a structural addition to the system and explain how registration is done.



Patients ring in using normal phones. Each phone is handled by a PhoneAgt which seeks to establis contact with a receptionist. This is done using the AgentRequest collaboration with Receptionistgents playing et hagt role. In this way phones get connected to receptionists using a fair queue discipline for requests and free receptionists. The receptionist will then register the patient information. There are two options for connecting to a doctor:

1. *the ReceptionistAgent issues a request to Doctors on behalf of the PhoneAgent using a slightly modified AgentRequest protocol, e.g AgentRequestByProxy.*

2. *the ReceptionistAgent sends the patient information to the PhoneAgent which issues a normal AgentRequest to doctors.*

No representation of receptionists and their terminals: - 3.No patient phones: -1 Allocating receptionists to phones not treated:-2 Not used AgentRequest to handle this: -1. Only structure, no explanation: max 5. Only explanation, no structure: max 5. How the new is connected to the old is not shown: -1-3. No explanation of how patients are allocated to doctor agents: -1

Question 3. (25%) Miscellaneous
Figure 6 describes two systems *VM1* and *VM2* using processalgebra, CCS.

1. Expand the expression for *VM 2= IF || CM || TM*

IF = money; ((coffee; mc'; c; drink'; IF) + (tea; mt'; t; drink'; IF))
CM = mc; c'; CM
TM = mt; t'; TM

VM 2= IF || CM || TM =
money; ((coffee; τ; τ; drink'; IF||CM||TM) + (tea; τ; τ; drink' IF||CM||TM)

If nothing is correct, but they have done an attempt to do something, e.g. claims initial deadlock: +3max; If some expansion is correct, but much is incorrect: +7 max; Including terms for hidden actions: -2; Omitting the first visible actions: -2. Minor errors like omitting end terms: -0,5-1 Not complete, but correct:-1

2. Compare the expressions for *VM1* and *VM2.* Are they observation equivalent or not? Justify.

VM1 = money; ((coffee; τ; drink'; VM1) + (tea; τ; drink'; VM1))

The difference between VM1 and VM2 is just one τ–transition. Under observation equivalence one may ignore sequences of τ–transitions, and therefore they are observation equivalent. One of the τ–laws can also be used in this case: τ; B ≈ B

Correct answer but wrong reasoning: -2 Good reasoning wrong answer: -1 Just generally about obs.eq.: max3. Answering as if observation eq. is the same as strong bisimulation (i.e. confusing the two): -2

3. Find the expression for an environment *E* of *VM1* such that all parts of *VM1* are explored without any deadlock. Expand the expression *E || VM1*.

We are looking for an environment E that offers actions that are complementary to those in VM1, i.e.: E = money'; ((coffee'; drink; E) + (tea'; drink; E))

Omitting the E at the end: -1, giving just a sum or similar almost correct solution: -1

E||VM! = τ;((τ; τ; E ||VM1) + (τ; τ; E||VM1))

Totally wrong expansion, no expansion: -3Minor expansion errors:-1

4. Explain the purpose of Tags in ASN.1, and give a couple of examples of their use.

Tags are used to control the tags used in the encoding/decoding. Most tags are automatically generated, but the user may specify tags when needed to avoid ambiguity or for other reasons. Wrong/no explanation why tags are needed: -2. No correct example: -3. No mention of encoding/decoding/transfer: -1

Vedlegg/ Appendix

Bokmål

Systemet som inngår i oppgavene 1 og 2: Telemedicine
I oppgavene 1 og 2 ser vi på funksjonaliteten til et system for telemedisin, vist med UML i Figure 1.

Pasienter kommuniserer med systemet via pasientterminaler, representert som *patient[n]* i Figure 1. Pasientterminalene har utstyr for å utføre enkle tester på pasienten. Doktorer kommuniserer via legeterminaler representert som *doctor[m]* i Figure 1. I systemet er det en agent, *Patients*, som har en indre agent for hver pasient, *PatientAgent,* samt oppførsel for å administrere pasientene. Videre er det en agent, *Doctors*, som har en indre agent for hver doktor, *DoctorAgent,* samt oppførsel for å administrere doktorene. Figure 2 viser hvordan disse agentene alternativt kan representeres som SDL200 blokktypene *Patients* og *Doctors*. Vi antar at pasientagenter og doktoragenter er persistente objekter som eksisterer så lenge en pasient/doktor er registrert i systemet. De skapes altså ikke dynamisk for hver pålogging eller sesjon.

Vi tenker oss at pasientene lider av sykdommer som krever hyppige konsultasjoner med en lege. Når en pasient ønsker kontakt med en lege sender dens *PatientAgent* en forespørsel til *Doctors* i henhold til en kollaborasjon kalt *AgentRequest.* Dersom ingen doktor er ledig settes forespørselen i kø inntil en doktor blir ledig. Når en doktor er ledig for nye pasienter, vil dens *DoctorAgent* signalisere dette til sin *Doctors* agent i henhold til *AgentRequest* kollaborasjonen.

Kollaborasjonene som brukes til pålogging, *Logon*, og til å opprette sesjoner mellom pasienter og doktorer, *AgentRequest*, er vist i Figure 3 a). Kollaborasjonene som foregår under en sesjon mellom lege og pasient er vist i Figure 3 b). Oppførselen til kollaborasjonene C*onsultation* og *DoctorInterface* er gitt i Figure 4.

English

In questions 1 and 2 we consider the functionality of a telemedicine system described using UML in Figure 1.

Patients communicate with the system using patient terminals represented by *patient[n]* in Figure 1. The patient terminals have equipment to carry out simple tests on the patient. Doctors communicate with the system using doctor terminals represented by *doctor[m]* in Figure 1. The system has an agent, *Patients*, that has an inner agent for each patient, *PatientAgent,* as well as behavior to manage the patients. Another agent, *Doctors*, has an inner agent for each doctor, *DoctorAgent* as well as behavior to manage the doctors. Figure 2 depicts these agents alternatively as SDL 2000 block types *Patients* and *Doctors*. We assume

that patient agents and doctor agents are persistent objects that exist as long as a patient and doctor are registered in the system. They are not created dynamically for each Logon and session.

We assume that a patient is being treated for an illness that requires frequent consultations with a doctor. When a patient wants to contact a doctor its *patientAgent* will issue a request to *Doctors* according to a collaboration called *AgentRequest*. If there are no free doctors, the request is queued until a doctor becomes available. When a doctor is free to accept a new patient its *DoctorAgent* will signal this to the *Doctors* agent according to the *AgentRequest* collaboration.

The *Logon* collaborations as well as the *AgentRequest* collaboration is shown in Figure 3 a). The collaborations taking place during a session are shown in Figure 3 b). The behavior of collaborations *Consultation* and *DoctorInterface* is given in Figure 4.

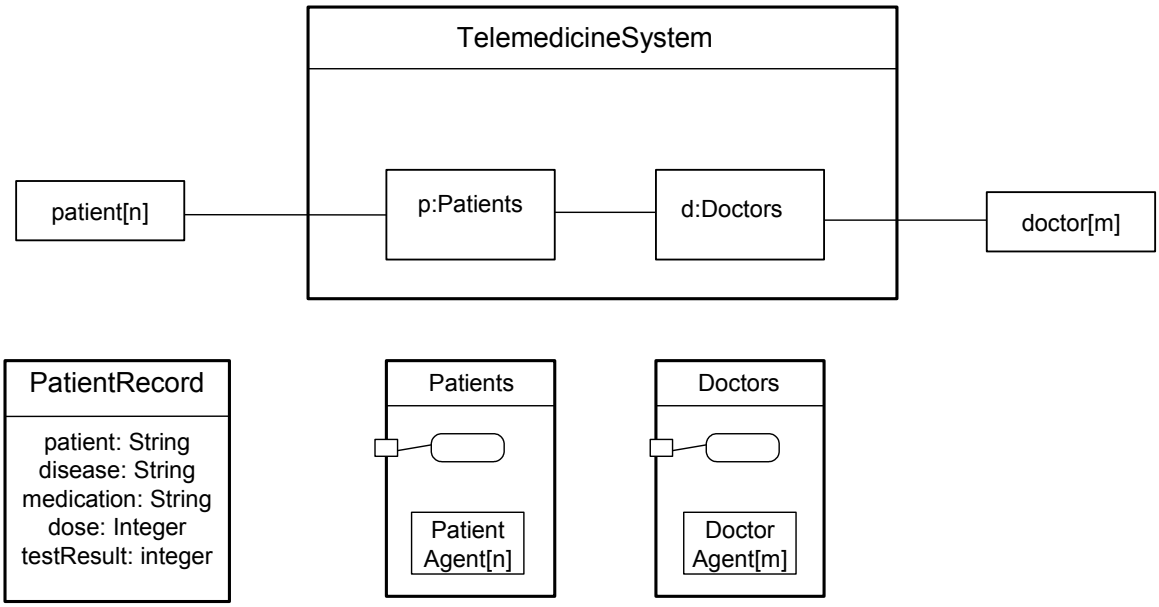


Figure 1 The Telemedicine system

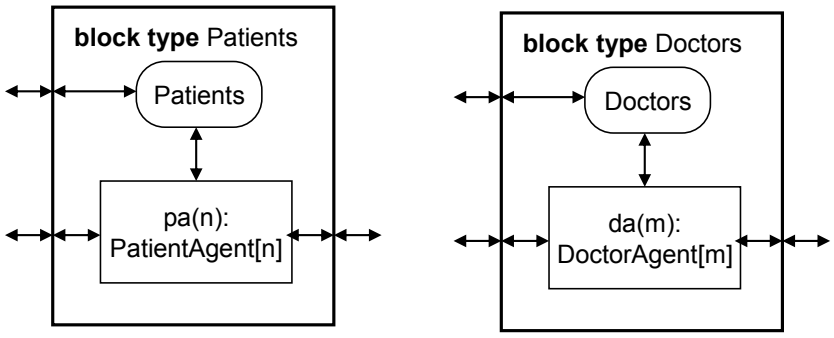
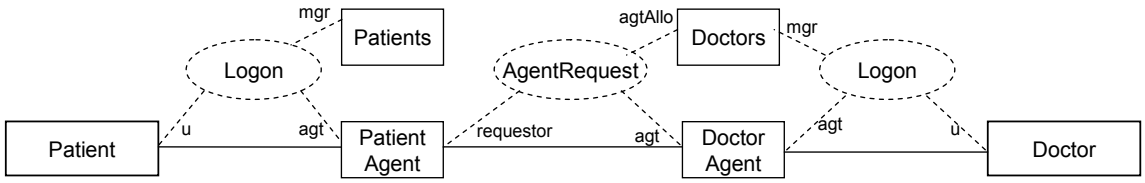
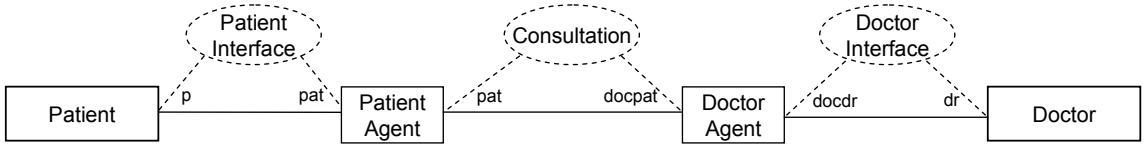


Figure 2 Alternative description of Patients and Doctors as SDL 2000 block types



a) Collaborations for logging on and for setting up sessions between patients and doctors



b) Collaborations in an established session between a patient and a doctor

Figure 3 Collaborations

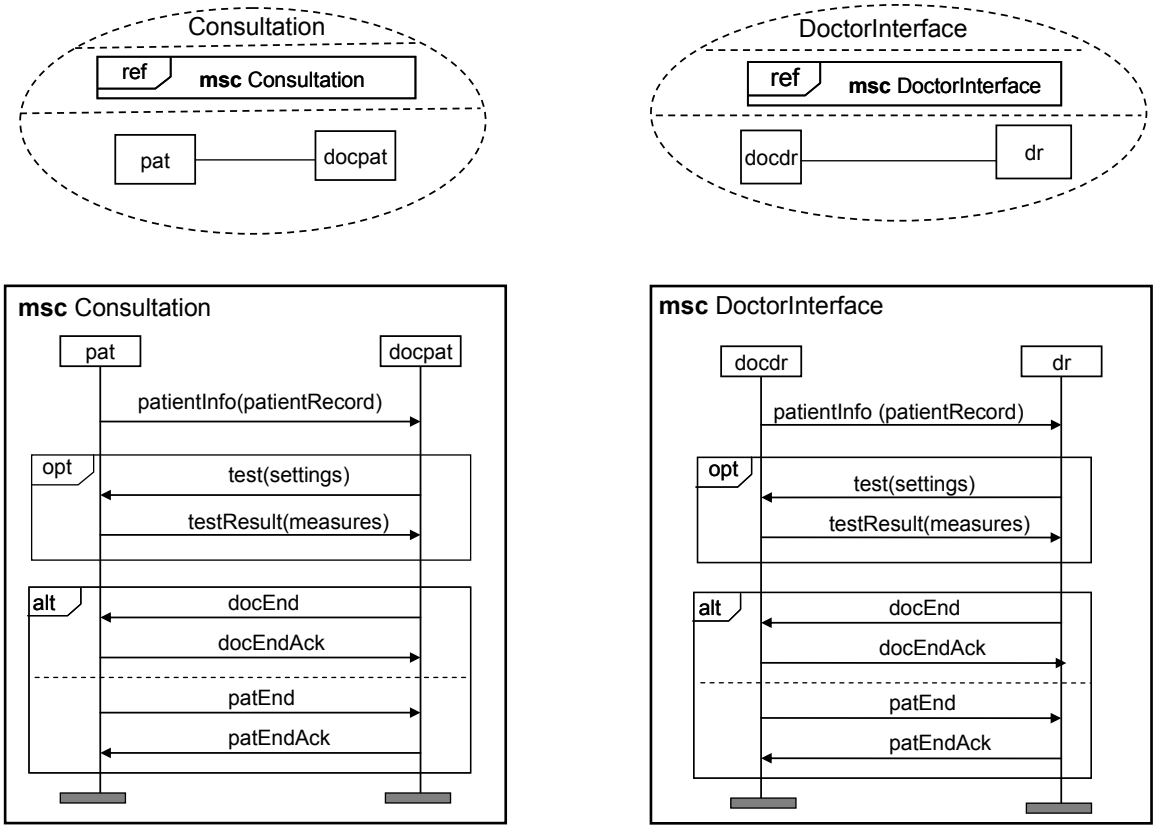


Figure 4 The *Consultation* and the D*octorInterface* collaboration

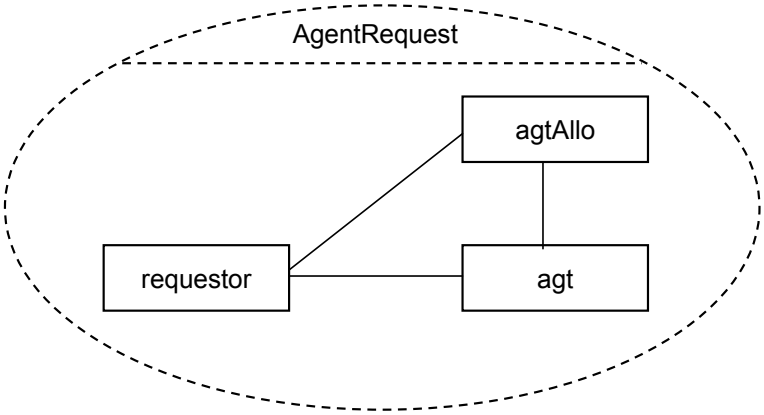


Figure 5 The *agentRequest* collaboration structure

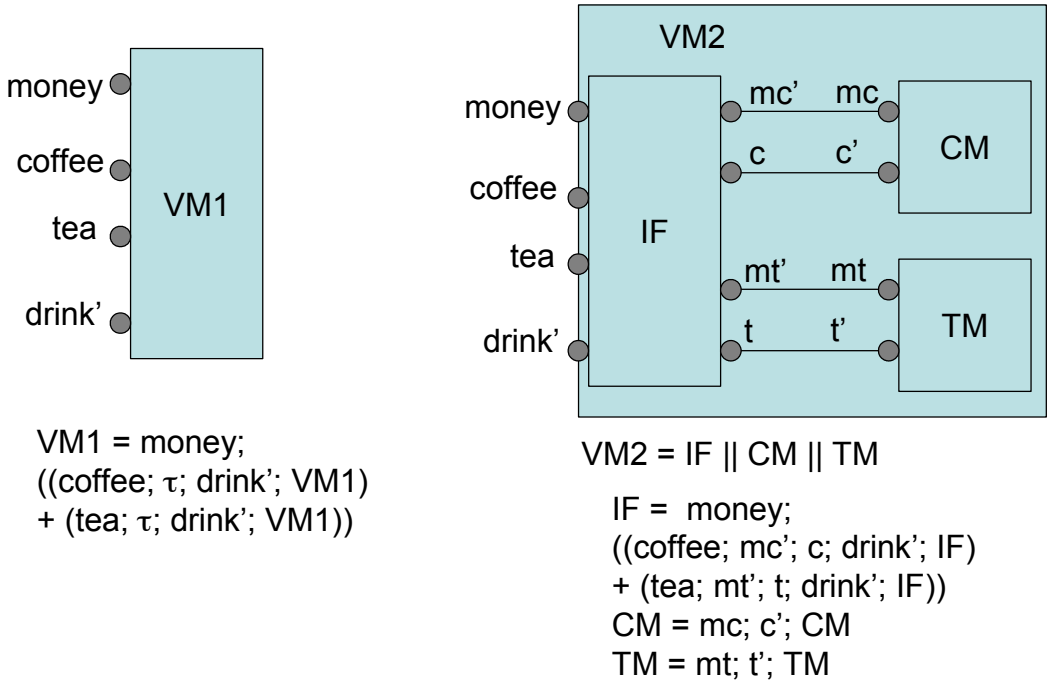


Figure 6 Vending Machines VM1 and VM2