Danmarks Tekniske Universitet

Udvikling af Framework til et Trust Management System

Kursus: Bachelor Projekt
Vejleder: Christian Damsgaard Jensen
Koordinator: Lars Staalhagen

Forfattere:
s154917 **Sebastian Busack Søberg**

16. maj 2017

Resumé

Projektet undersøger trust management systemer. Fokuset for opgaven er at lave en implementation af et dynamisk framework, der kan understøtte muligheden for at udskifte tillidsmodellen og risikomodellen uafhængigt af hinanden under den givne kontekst.

I rapporten udlægges de krav, der bliver stillet til tillidsmodellen og risikomodellen på baggrund af beslutningslogikken. Det kan i rapporten konkluderes at dette kan begrænses til et krav om, hvilke værdier der er mulige at returnere. På samme måde vises det med den konkrete implementation af et framework, at en løsning for at bibeholde det det generelle datalagre, så er det en mulighed, at løse dette gennem nedarvning af datamodellen.

Til sidst argumenteres det at på trods af at det er muligt at lave et framework med en kontekst, så ville et universelt framework uden kontekst, miste størstedelen af funktionaliteten i et trust management systemet, og derfor ville den ikke have den samme form for brugbarhed.

Indhold

Figurer

Tabeller

List of Code

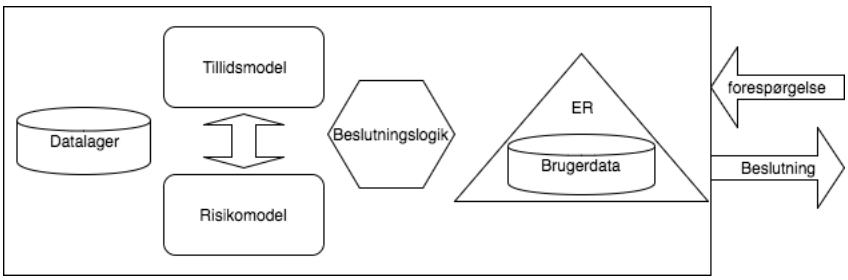
1 Introduktion

Den generelle verden bygger på et system af garantier, når man sker interaktioner mellem parter. For eksempel hvis man går ind i en butik og køber en vare, så er man i tilfælde af at varen er i stykker beskyttet af dansk lov i form af reklamationsret. På samme måde når man går op til kassen og betaler for varen med sit dankort, så er butikken garanteret af VISA, at regning bliver betalt. Disse garantier gør at der vil ske en rettergang på trods af dit personlige forhold til modparten i handlen. Dette er mulighed fordi modparten er en kendt person, der er underlagt de samme regler.
Når man laver transaktioner online, så kan det være svært at garantere de samme rettigheder. Dette skyldes, at man ikke nødvendigvis kender sin modparts identitet, eller det kan ske, at modparten ikke er under den samme retsinstans. Dette kan gøre rettergang en del sværere, derfor er man nød til at bruge en alternativ form sikkerhed, og dette kan være et tillidssystem. Tillid er et udtryk for hvorvidt man tror en given modpart handler forudsigeligt. Tillid er oftest brugt i et sammenhæng, hvor man har begrænset eller ingen viden om modpartens reelle identitet. Tillid bliver opbygget ved at få en historik med sin modpart. Hver gang man har lavet en succesfuld transaktion med sin modpart, så stiger tilliden til den modpart; på samme måde, hver gang man har lavet en dårlig transaktion med modparten, så falder tilliden. Dertil er en anden faktor tredjeparters rekommandationer. Hvis man har tre parter A,B og C, hvor B ønsker at øge A's tillid til sig, så kan B kontakte C og spørge efter en rekommandering. C vil så rekommandere B til A, og derved kan A's tillid til B stige.
Tillid bliver også tænkt som risikoen ved en transaktion. Ved høj tillid er risikoen forventet at være lav, hvorimod ved lav tillid, så er risikoen høj, da modparten ikke nødvendigvis har bevist sig at være konsistent. Det er ikke altid, at en tillidsværdi i sig selv kan være dækkende for en risikoen i en transaktion. Dette kunne være situation, hvor modparten kun indirekte er beslutningstageren om, hvorvidt transaktionen skal gå godt.
Et eksempel kunne være en grossist, der sælger såsæd på kredit til landmænd. I denne situation kan en beslutningsfaktor ud over en generel tillid være vejret. Dette er aktuelt, da en landmands mulighed for at tilbagebetale sin kredit kan være afhængig af en god høst. Så hvis der en dårlig vejrprognose for sommeren, så kunne det betyde en øget risiko ved en eventuel kredit.
I et moderne trust management system er det normalt, at både bruge en tillidsmodel samt en risikomodel, hvis vurdering for beregningen bliver lagt sammen i en vægtet ligning. Trust management systemer oftest består af hard-coded metoder og beregninger Sammenhænget mellem modellerne er ikke altid klare. Dette er en meget rigid implementation som ofte betyder, at der ikke er mulighed for en senere rekonfigurering.

I projektet vil der fokuseres på at udvikle en arkitektur for det generelle trust management system. Dertil vil der undersøges og specificeres, hvad de nødvendige krav til tillidsmodeller og risikomodeller er under en given kontekst. Dette gøres så der kan opnås en trust management arkitektur med et dynamisk design, hvor det muligt at udskifte modellerne uafhængigt af hinanden.

2 Teori

Figur 1 viser et simpelt trust management system indeholdende de basale komponenter.



Figur 1: Et simpelt trust management system.

Ved en forespørgelse til et trust management system vil systemets brugerdatabase først verificere brugeren. Denne deloperation er til for at sikre anonymitet, hvilket gør det muligt for brugere, at handle under pseudonymer. Efter brugerens identitet i systemet er fastlagt, er det op til beslutningslogikken, at bestemme, hvad svaret på forespørgelsen skal være. Dette gøres ved hjælp af tillidsmodellen og risikomodellen. Både tillidsmodellen og risikomodellen har data liggende i datalagret. Tillidsmodellen er generelt afhængig af data omkring den tidligere beregnet tillidsværdi, samt anbefalinger fra andre entiteter omkring brugeren. For en risikomodel er dataen afhængig af modellen.
Som det udtrykkes på figur 1, så er der kommunikation på tværs af tillidsmodellen og Risikomodellen. Dette er meget normal, og ofte er risiko anset som en del af tillidsmodellen, og de kan ofte dele data fra datalagret.

2.1 Tillidsmodeller

En tillidmodel består af en vurdering af, hvorvidt man tror modparten vil opfører sig som forventet. Dette er udtrykt ved en tillidsværdi. Tillidsværdien er beregnet på baggrund af historisk data over tidligere transaktioner med samme modpart.

De generelle funktioner i en tillidsmodel kan udtrykkes som følgende:

Beregning af tillid. Tillidsmodellen bestemmer på baggrund af den tilgængelige data, hvor stor tilliden er til en anden entitet. Denne tillid bliver så efterfølgende brugt til at beslutte, hvorvidt entitetens forespørgelse skal godkendes.

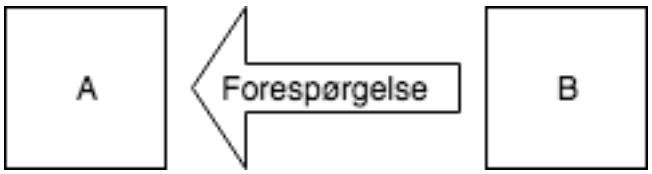
Updatering af tillid. Tillidsmodellens tillid til en anden entitet ændrer sig på baggrund af erfaringerne der laves med entiten i tidligere transaktioner. Efter en god transaktion, så vil tilliden til entiten øges og på samme måde efter en dårlig transaktion falder tilliden.

Anbefaling af entiteter. Tillidsmodeller bruger ofte andre entiters anbefalinger af entiter begge kender til at bestemme den endelige tillid.

Disse funktionaliteter er de generelle grundsten i en tillidsmodel, men der er mange varianter, hvor på der er mere eller mindre fokus på delelementerne i modellen. Dette kan være aktuelt i forehold til at modarbejde sikkerhedshuller i modellen, der kan forekomme i visse kontekster.

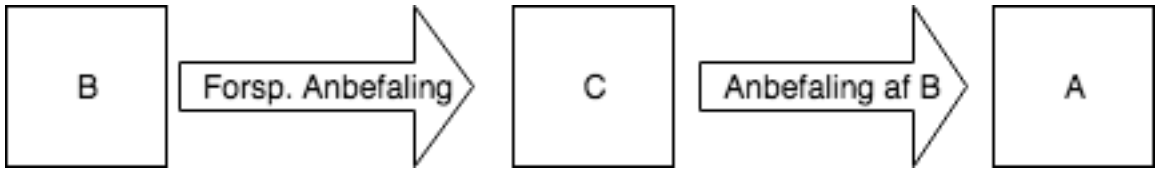
I et scenarie, som på figur 2 hvor B laver en forespørgelse til bruger A, så vil det være op til A at besluttte om han skal godkende forespørgelsen på baggrund af, hvordan B tidligere har opført sig under transaktioner. A's værdi for tilliden til B kaldes $t_{A,B}$. A tillidsværdi til B $t_{A,B}$

beregnes ud fra tidligere data fra transaktioner med B, og på baggrund af dem besluttes der, hvor stor tilliden til B er.
Tilliden mellem to entiteter er asymmetrisk, da A's tillid til B ikke fortæller noget om, hvor stor B's tillid til A er. Så hvis A kan acceptere en transaktion fra B på baggrund af $t_{A,B}$, så betyder det ikke at B kan acceptere en transaktion fra A på baggrund af sin egen tillidsværdi for A, $t_{B,A}$.



Figur 2: Bruger B laver en transaktionsforespørgelse til A

Hvis det viser sig, at tillidsværdien $t_{A,B}$ er for lav til B's ønskede handling, så kan den ofte i tillidsmodeller øges gennem anbefalinger. Så hvis B ønsker at øge sin tillidværdi hos A, så kan B forespørge en anbefaling fra bruger C, som i figur 3. I så fald B har en høj nok tillid hos C, vil C så anbefale B til A.



Figur 3: Bruger B forespørger en anbefaling fra bruger C til A

Ved brug af anbefaling vil man ofte i stedet for tillid udtrykke ry, $rep_{A,B}$. Med en anbefaling fra C kan man på simpel vis udtrykke $rep_{A,B}$ som gennemsnittet af entiteternes tillid til B:

$$rep_{A,B} = \frac{t_{A,B} + t_{C,B}}{2} \tag{1}$$

En eventuel beregningsmodel for $rep_{A,B}$ kunne så være en summering af alle anbefalinger. Dette kan i et system med n entiter udtrykkes som:

$$\frac{\sum_i^{n\setminus\{B\}} t_{i,B}}{n-1} = rep_{A,B} \tag{2}$$

Som det kan ses i ligningen er B fjernet fra sættet af anbefalinger, som der summeres over. Dette er på grund af den reflektive egenskab i tillidsmodeller, der gør at en entitet vil altid have fuld tillid til sig selv, og derfor kan man generelt ikke anbefale sig selv.
Denne simple model kan dog give det problem at, der ikke er taget højde for, hvorvidt anbefalingerne er troværdige. I en situation, hvor både B og C var utroværdige entiteter, kunne de aftale at give hinanden høje anbefalinger for at opnå bedre ry hos tredjeparter. En måde at udvide den ovenstående fremgangsmåde, for at undgå dette problem, ville være:

$$rep_{A,B} = \frac{t_{A,B} + \sum_i^{n\setminus\{A,B\}} t_{A,i} t_{i,B}}{n-1} \tag{3}$$

Nu vil i's tillidsværdi vægtes ind i A's beregning af $rep_{A,B}$. Tillid er reflektiv så i en situation som i figure 3, hvor C's tillid til B er brugt i beregningen af $rep_{A,B}$, så vil A's egen tillid til B altid have maksimum vægtning, da en tillidsmodel altid har fuld tillid til sigselv, hvorimod C's

vægt i det endelige ry er afhænging af, hvor stor A's tillid til C er. Det kan ses i ligningerne, at hvis $t_{A,C}$ samt $t_{C,B}$ har en høj værdi, så har det en positiv indflydelse på B's ry. På samme måde kan det ses, hvis $t_{A,C}t_{C,B}$ har en lav værdi, så vil det give en dårlig inflydelse til B. Denne transitive funktionalitet gør, at det kan give et dårligere ry at lave transaktioner med entiteter, der har et dårligt ry. Denne funktionalitet ville også kunne fjernes, ved blot at ignorere alle entiteter med en lav tillidsværdi.

$$e = \{\text{alle anbefalinger af B}\} \tag{4}$$

$$n = \{i \in e | t_{A,i} > minTillid\} \tag{5}$$

Når det kommer til de reelle beregninger i en tillidsmodel, så kan de variere alt efter, hvad der er prioriteret i modellen. Det kan være at det skal være svære at opnå tillid en det er at miste den, eller det kan være, at tillidshistoriken skal stagnere med tiden, så nyere opdateringer tæller for mere end ældre opdateringer.

I et eksempel har vi en fiktiv markedsportal, der hedder Junglekøb.dk. Junglekøb gør det muligt for private såvel som professionelle sælgere, at sælge deres vare online. Junglekøb har ikke nogen direkte kontrol over varene, der bliver solgt gennem deres portal, de facilitere blot kontakten mellem køber og sælger, og de giver sælgerne muligheden for at vise deres varer. Junglekøb er en international side, der har købere og sælgere fra mange forskellige lande, derfor er det i tilfælde af en varer der ikke dukker efter et køb, svært at holde nogen parter ansvarlige. Det er i et sådan tilfælde, hvor en varer ikke dukker op, er svært at afgøre, hvorvidt varen forsvandt i posten, eller om sælgeren var uærlig og ikke sendte den. Dette kunne løses ved at kræve sporing på varene, når de sendes. Det er dog ikke altid ideelt, da den totale pris på varen ville stige drastisk, og i nogen tilfælde til et niveau, hvor forsendelsesprisen overstiger varens pris. Derfor ville det selvfølgelig være bedre, hvis man blot kunne stole på sælgeren. I denne situation kan man bruge en simpel tillidsmodel.
Da Junglekøb ikke har nogen direkte kontakt med sælgerne, så har de ikke nogen personlig holdning til tilliden af sælgerne. I stedet bygges tilliden kun på anbefalingerne fra købere, der har købt fra en given sælger. Hver sælger får en tillidsværdi, som starter på en lav værdi, og for hver varer, som en køber melder, at de har modtaget, opdateres sælgerens tillidsværdi. Sælgerens tillidsniveau kan så diktere, hvor dyre varer sælgeren kan må sælge uden at sporing på pakkerne er påkrævet.

Junglekøb har nu opstillet en simpel model. Men før denne model er funktionel, skal der tilføjes nogle beslutninger. Der skal vælges en strategi for tillidsopdatering, der forebygger potentielle sikkerhedshuller i deres kontekst. I en kontekst som Junglekøbs kunne en eventuel sikkerhedstrussel være on-off attacks. Et on-off attack ville i denne kontekst kunne være en sælger der på systematisk vis ikke sender pakker, og derved snyder, men er stadig i stand til at bibeholde en høj tillidsværdi. Dette kunne forebygges i en tillidsmodel, ved at bruge en sandsynlighedsbaseret tillidsmodel[1]. I en sandsynlighedsbaseret tillidsmodel er man også i stand til at bruge en spredningsværdi til at afsløre on-off attacks.

2.2 Risikomodeller

Da tillid alene ikke altid giver det bedste beslutningsgrundlag, bruger man også risikomodeller. Hvor en tillidsmodel bygger på, hvor konsistent en entitet har handlet over for andre entiteter, så bygger risikomodeller generelt på sekundære risikoaspekter. Dette kunne for eksempel i et online system være at et risikoaspekt var, hvilken sikkerhedsprotokol den anden entitet brugte. Dette er både brugbart i situationer, hvor man har interesse i at lave transaktioner med en utroværdig entitet, men også i situationer, hvor der kan forekomme en stor usikkerhed fra sekundære trusler.

Ved en utroværdig bruger, kan en risikomodel bruges til at bestemme, hvorvidt der er nok sikkerhed omkring transaktionen, så det faktum, at modparten måske har tænkt sig at snyde ikke er et problem.
En risikomodel opstilles ud fra en sikkerhedsanalyse af konteksten, og derfor er en risikomodel meget svære at generalisere. For eksempel kunne man i en kontekst have brugernes kreditværdighed. Denne risikomodel ville ikke give mening i en kontekst, hvor det handler om at sikre den bedste forbindelse i et ad hoc netværk[2]

Hos Junglekøb har man været ude for at sælgere ville sælge billige varer for at hæve deres tillidsværdi, for derefter så at snyde købere for meget dyrere varer. Dette kan løses ved at tilføje en risikomodel, som nu også bliver brugt når der skal besluttes, hvorvidt der må sendes varer uden sporing. Denne model kunne fungere ved at opbygge nogle krav baseret på tidligere forsendelsers værdi. Denne model kunne være baseret på følgende algoritme:

- De første 50 en sælger sælger, skal spores.
- Varer der er mere end 50% dyrere end gennemsnitsværdien på de tidligere solgte varer, skal spores.

Med denne risikomodel kan Junglekøb nu sikre købere mod sælgere, der har opbygget en tillidsværdi ved at sælge billige varer for så at snyde på meget dyrer varer.

De sikkerheder man er interesseret i at opnå kan ofte være afhængig af information fra en uafhængig tredjepart. Derfor er en overvejelse, at en potentiel pris for at undersøge risikoen kan overgå værdien af transaktionen.

2.3 Beslutningslogik

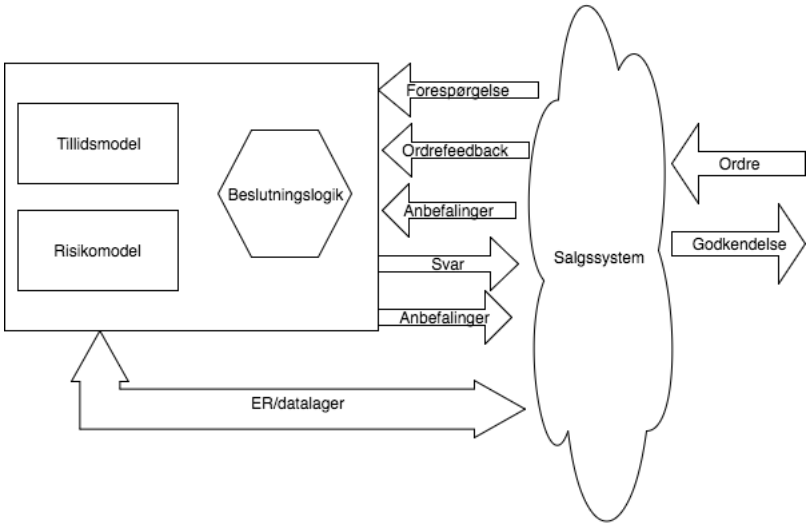
En endelig beslutning omkring en forespørgelse bliver afgjort i et trust management systems beslutningslogik. Beslutningslogikkens afgørelse bliver taget på baggrund af værdierne beregnet af henholdsvis tillidsmodellen og risikomodellen. Der er ikke en generel algoritme for, hvordan dette beregnes, men i stedet er det afhængigt af en given kontekst. For eksempel kan det kræves i algoritmen, at både tillidsværdien og risikoværdien er over en hvis værdi. Alternativt kan det være at gennemsnittet beregnes og ud fra den værdi afgøres beslutningen. I en kontekst, hvor der er en udgift i at beregne risikoen, er det muligt at konstruere en beslutningslogik, hvor at man mindsker denne udgift, ved først at undersøge tillidsværdien, og på baggrund af den derefter beslutte om, der skal bruges resurser på at beregne risikoen.

3 Case

Den fiktive virksomhed Korn og Fodder er en grossist, der sælger varer på kredit til landmænd. Dette sker gennem Korn og Fodders hjemmeside, som er en fuldautmatiseret købsportal, så fra bestilling er der ikke nogen videre kontrol eller godkendelse af landmandens ordre. Dette har givet noget problemer. Blandt andet har Korn og Fodder været ude for, at en landmand har lagt adskillige store ordre, for så at forsvinde efterfølgende uden at have betalt sine regninger. Korn og Fodder har også været ude for at en landmand har købt for stort ind så efterfølgende ikke været i stand til at betale sin regning på forfaldsdatoen. For Korn og Fodder betyder en handling som denne et stort tab i omsætningen.
Korn og fodder ønsker derfor et Trust Management System til deres hjemmeside, som kan være en foranstaltning mod sådan situationer i fremtiden.
Korn og Fodder er del af en større grossist forening, hvor de individuelle grossister har mange af de samme kunder. De andre grossister har intentioner om at implementere lignende trust management systemer, derfor skal det være muligt, at dele anbefalinger af kunderne.

4 Design

Et reelt Trust management system framework, ville skulle implementeres som en del af et eksisterende system. I casen er systemet en salgsportal, hvilket betyder, at der allerede eksisterer et bruger valideringssystem, samt en database med historikker over brugernes tidligere køb. Dette betyder at den mest effiktive løsning ville være, at give frameworket mulighed for at understøtte udefrakommende datalagring og brugervalidering. Dette ville reducerer systemet, til kun at indeholde elementerne Tillidsmodel, Risikomodel og Beslutningslogik, som på figur 4.



Figur 4: Visualisering af interaktionen mellem Trust management system frameworket og det aktuelle system.

Frameworket på figur 4 har ud over forbindelsen der ville stå for at sende data og brugervalidering frem og tilbage, tre funktionaliteter.

Forespørgelse frameworket underbygger hovedfunktionen, der er i konteksten at beslutte, hvorvidt et køb kan godkendes.

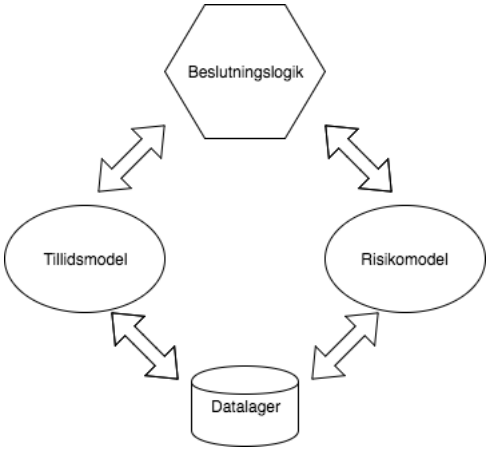
Ordrefeedback når det er muligt, skal der gives feedback på, hvorvidt en forespørgelse er gået godt.

Anbefalinger Der skal understøttes med muligheden for at tilføje anbefalinger fra andre grossister.

Konteksten for designet er at sikre, en hvis tillid til køber baseret på størrelsen af købet. På samme måde skal der sikres diversificering af investering ved at lave en begrænsning på krediteringer hos samme køber. Denne kontekst giver et basis for måden hvorpå beslutningslogikken implementeres.

4.1 Beslutningslogik

For at understøtte hovedmålet med et framework, hvor tillidsmodellen og risikomodellen er løst koblet, er vi nød til at sørge for, at der ikke er nogen direkte relation mellem tillidsmodellen og relationsmodellen. Dette betyder også at alle relationer mellem tillid og risiko kun kan forgå igennem beslutningslogikken. Systemet har den overordnet struktur som på figur 5



Figur 5: Overordnet sammenhæng mellem tillidsmodellen, risikomodellen og beslutningslogikken

Ved en forespørgelse skal der tage en beslutning om hvorvidt et køb kan godkendes. Dette skal ske på baggrund af købers tillidsværdi og risikoværdi.
I systemet er der tre risikoniveau'er henholdsvis lav, medium og høj. Dette bruges til variere, hvor stor betydning en transaktion har. Et køb på for et lille beløb betyder, at der kun er en lille kreditering, og derved en lille risiko ved manglende betaling eller lignende. En stor ordre eller en ordrer med udvidet kredit tid er derimod en meget større investering, og investeringer af den slags er man interesseret i at sprede ud, så man mindsker risikoen, hvis en køber skulle ske at ikke betale. På samme måde er man ikke interesseret i at tilbyde højrisiko krediteringer til ukendte købere.
Algoritmen i beslutningslogikken bruger en algoritme, der kræver at både tillidsværdien og risikoværdien har en minimumværdi, der er fastsat i tabel 1.

Risikonivau	Tillidsmodel	Risikomodel
Lav	0	0,5
Medium	0,5	0,5
Høj	0,8	0,8

Tabel 1: Grænseværdier for tillidsværdien og risikoværdien for at et køb bliver godkendt.

Som tabel 1 viser er minimumsværdien for lavrisiko køb 0. Dette er et valg, for at nye købere uden et ry kan købe varer. Man kunne alternativt hæve minimum tillidsværdien. Det ville betyde, at en køber ville være nød til at have anbefalinger, for at få købene godkendt.

For at købet bliver godkendt kræver det at både tillids- og risikoværdien er ved grænseværdien minimum. For at beslutningslogikkens algoritme fungere kræver det at værdierne, der returneres fra tillidsmodellen og risikomodellen ligger imellem 0 og 1.

4.2 Tillidsmodel

Tillidenmodellen beregnes med samme formel, som den i eksemplet i det tidligere afsnit på følgende måde:

$$t_{i,j} \in [0,1]n = \{\text{Anbefalinger af bruger B}\} \tag{6}$$

$$rep_{A,B} = \frac{t_{A,B} + \sum_i^n t_{A,i} t_{i,B}}{n} \tag{7}$$

Med anbefalinger har nye købere mulighed for at hurtigere opbygge tilliden, der er nødvendig for at få krediteringerne nødvendig for et større køb.

Tilliden i modellen opdateres efter et køb. Hvorvidt en regning er blevet betalt til tiden bestemmer om det bliver en positiv eller negativ opdatering. På samme måde er hvor stor en ændring det er afgjort af hvor stort et køb det er.

Risikoniveau	Betalt	Ikke Betalt
Lav	0,03	-0,075
Medium	0,05	-0,125
Høj	0,08	-0,2

Tabel 2: Ændring af tillid på baggrund af opdateringer

Som det kan ses i tabel 2 så stiger forøgelsen, og det samme gælder for reduceringerne.

4.3 Risikomodel

Risikomodellen bygger på en vurdering af historikken over tidligere køb. Ved gennemgangen af købene er der et fokus på diversificering af investering for grossisten. Grossisten er generelt ikke interesseret i at investere for mange penge i krediteringer hos den samme køber, da det i tilfælde af at køberen ikke er i stand til at betale sine regninger en større risiko.
Når risikoen beregnes bliver alle køberens tidligere køb delt ind i fire kategorier:

- ubetalte regninger, der er før forfaldsdatoen
- ubetalte regninger, der er efter forfaldsdatoen
- betalte regninger, der blev betalt før forfaldsdatoen
- betalte regninger, der blev betalt efter forfaldsdatoen

Når en Køber A's risikoværdi beregnes, bliver samtlige tidligere køb vurderet, for at tilskrive dem en værdi som i tabel 3.

Type	værditilskrivning
Ubetalt forfalden regning	0
Regning betalt efter forfaldsdato	0,5
Ubetalt ikke forfalden regning	0,75
Regning betalt til tiden	1

Tabel 3: Oversigt over værditilskrivninger til købers tidligere køb.

For at beregne den endelige risikoværdi r_A vægtes hver af A's n købs værditilskrivning v_{Ai} med købets værdi p_Ai divideret med kundens totale købsværdi s_A Denne værdi divideres med n

$$r_A = \frac{\sum_i^n v_{Ai} p_{Ai} s_A}{n} \tag{8}$$

Hvis køber A ikke har nogen tidligere køb er risikoværdien 1. Som det så kan ses i tabel 3, at for hver ikke betalte køb, falder risikoværdien.

4.4 Datalager

Dataen der bliver brugt af henholdsvis tillidsmodellen og risikomodellen kan ikke nødvendigvis deles, men datamodellen er i dette system lavet til at kunne bruges til begge. Datamodellen er oprettet med en række generiske felter, som kan dække det mest generelle data.

Datamodel:	
Type	Data/Anbefaling/Tillidsværdi
Dato	Dato
Køber	relaterede bruger
Anbefaler	brugeren der laver en potentiel Anbefaling
Værdi	værdi [0,1]

Tabel 4: Standard felterne i trust management system frameworkets data

Tabel 4 viser de konkrete felter. I riskimodellen bruges data, af typen data, mens typerne anbefaling og tilidsværdi bruges af tillidsmodellen. Dato feltet giver muligheden for at understøtte modeller, hvor der for eksempel er en stagnering af data. Køber og Anbefaler er referencer til brugere i systemet. Anbefaler bliver kun brugt ved anbefalinger, hvor feltet giver en reference til den bruger, der har anbefalet brugeren. Værdi feltet indeholder en decimalværdi mellem nul og et. Ved tillidsværdi og anbefalings datatyperne beskriver værdifeltet tillidsværdien.

Da den konkrete risikomodels kreditdata ikke passer til den nuværende datamodel, udvides dette som i tabel 5.

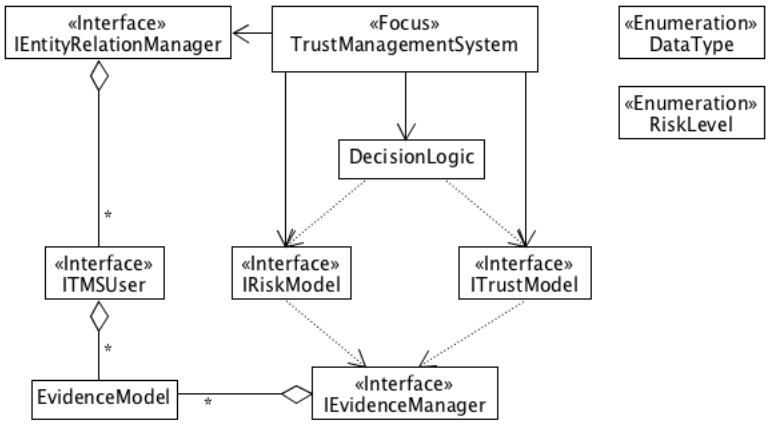
Datamodel:	
Type	Data
Dato	Dato for køb
Køber	relaterede bruger
Anbefaler	ubrugt
Værdi	pris
Betalingsdato	Dato for betaling
Kreditforlængelse	potentiel udvidet kredittid

Tabel 5: Felterne i trust management system frameworket udvidet kreditdata

Tabel 5 udvider datamodellen med en betalingsdato og en kreditforlængelse. Ved datatypen data, som bliver brugt i risikomodellen bliver dato brugt til at notere datoen for et købt, og værdi bliver brugt til at notere værdien af købet.

Ved en potentiel udskiftning af risikomodellen kan datamodellen så udvides med de nødvendige felter som i tabel 5.

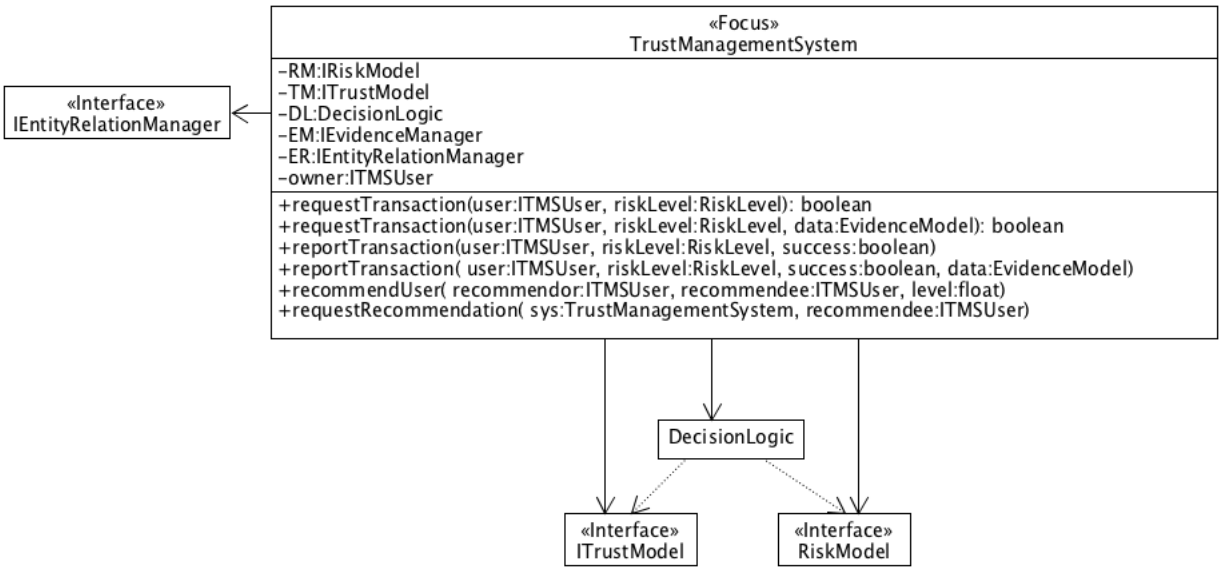
5 Implementering



Figur 6: UML diagram over trust management system frameworket.

For at understøtte muligheden for at skifte tillidsmodel og risikomodel, kan det ses på figur 6, at de er implementeret som interfaces, hvilket giver en løs kobling. På samme måde er brugerdatabasen og datalagret også implementeres som interfaces, for at give muligheden for at koble dem direkte på for eksempel en hjemmesides eksisterende brugerdatabase og datalager.
I den reelle implementering er **IEvidenceManager** og **IEntityRelationManager** på simpel vis i klasserne **EvidenceManager** og **EntityRelationManager**. Der er aldrig lavet en aktuel implementering af bruger interfacet, da det ikke har mere nødvendig funktionalitet end at returnere et string indeholdende en brugers brugernavn.

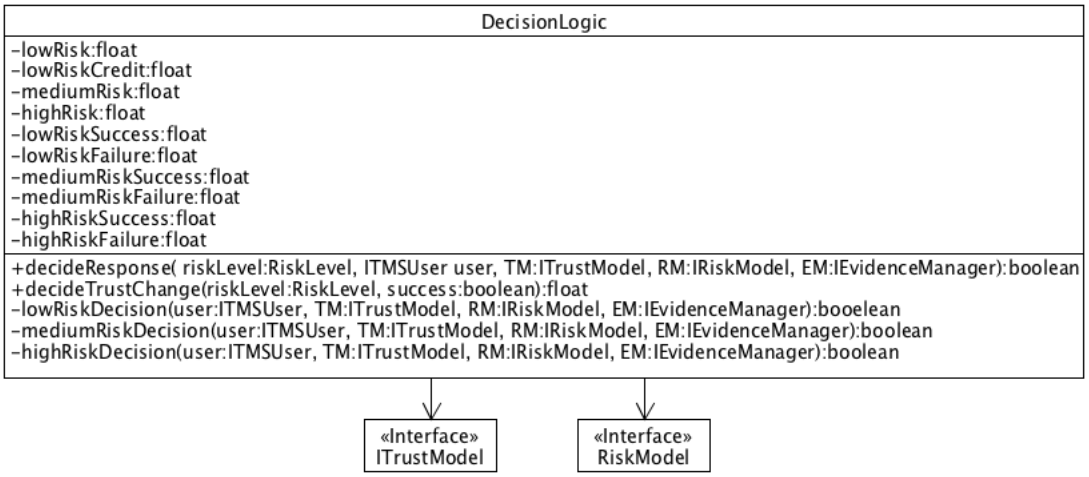
5.1 TrustManagementSystem.java



Figur 7: Udvidet UML diagram for TrustManagementSystem klassen.

I **TrustManagementSystem** implementeres alle systemets andre klasser igennem constructoren. Dette er i tilfælde af interfacene for at give muligheden for at implementere alternative realiseringer af de interfaces. For resten af klasserne er det for at give muligheden for at bedre kunne teste koden gennem mocks. **TrustManagementSystem** klassens hovedfunktion er at sende valideringsforespørgelser til realiseringen af **IEntityRelationManager** for derefter at videregive godkendelses forespørgelser til **DecisionLogic**, samt sende opdateringer med nyt data til de rette modeller.

5.2 DecisionLogic.java



Figur 8: Udvidet UML diagram for DecisionLogic klassen.

Som det kan ses på figur 8 har **DecisionLogic** kun to public metoder. Den ene håndtere svaret på en købsforespørgelse, og den anden håndtere ændringen der sker i tillidsmodellen efter et køb. Når der skal besluttes, hvorvidt en brugers tillids- og risikoværdier er tilstrækkelige til at godkende et køb, bruges parametret i kode 1, RiskLevel til at beslutte, hvilken af metoderne i kode 2

```
1  public boolean decideResponse(RiskLevel riskLevel,
       ITMSUser user, ITrustModel TM, IRiskModel RM,
       IEvidenceManager EM) {
2      switch(riskLevel){
3        case LOWRISK:
4          return lowRiskDecision(user, TM, RM, EM);
5        case MEDIUMRISK:
6          return mediumRiskDecision(user, TM, RM, EM);
7        case HIGHRISK:
8          return highRiskDecision(user, TM, RM, EM);
9      }
10     throw new UnsupportedOperationException();
11   }
```

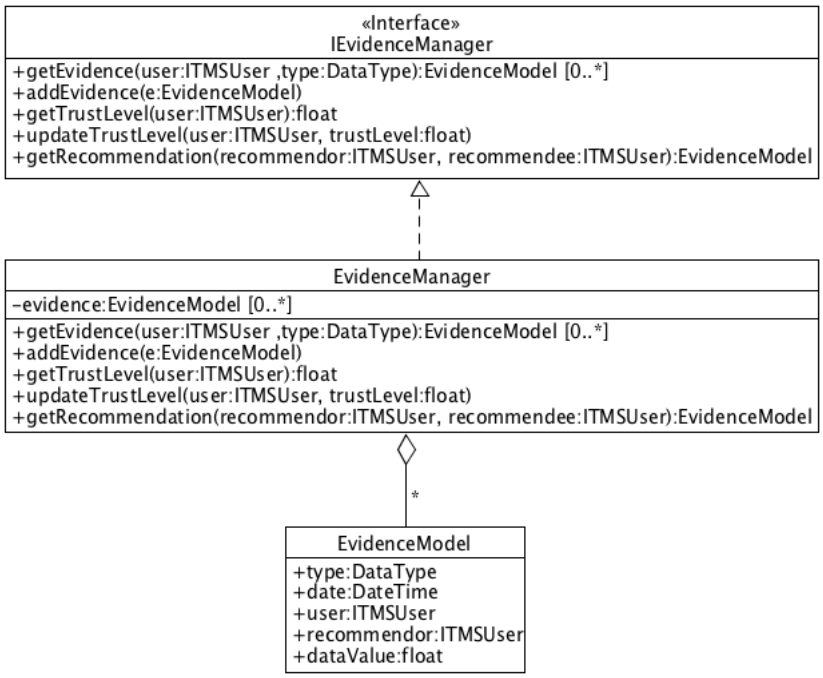
Code 1: decideResponse returnere en boolean beregnet på baggrund tillidsværdien og risikoværdien

De variablene i if sætningerne i kode 2 er værdierne opstillet i tabel 1. Ved begge public metoder modtages tillidsmodellen og risikomodellen som parametre. Dette giver en løs kobling, som ikke nødvendigvis er kritisk, men det giver som i **TrustManagemenSystem** klassen en mulighed for at teste klassen gennem mocks.

```
1  private boolean lowRiskDecision(ITMSUser user, ITrustModel
       TM, IRiskModel RM, IEvidenceManager EM){
2      if(TM.calculateTrust(user, EM) >= lowRisk && RM.
           calculateRisk(user, EM) >= lowRiskCredit){
3        return true;
4      }
5      return false;
6    }
7
8
9    private boolean mediumRiskDecision(ITMSUser user,
         ITrustModel TM, IRiskModel RM, IEvidenceManager EM){
10     if(TM.calculateTrust(user, EM) >= mediumRisk && RM.
           calculateRisk(user, EM) >= mediumRisk){
11       return true;
12     }
13     return false;
14   }
15
16
17   private boolean highRiskDecision(ITMSUser user,
         ITrustModel TM, IRiskModel RM, IEvidenceManager EM){
18     if(TM.calculateTrust(user, EM) >= highRisk && RM.
           calculateRisk(user, EM) >= highRisk){
19       return true;
20     }
21     return false;
22   }
```

Code 2: Metoder der håndtere de individuelle risikoniveauer

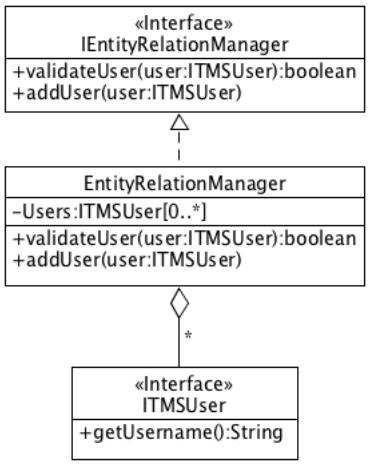
5.3 IEvidenceManager.java



Figur 9: UML diagram, der viser relationen mellem IEvidenceManager samt dens realisering og datamodel, EvidenceModel

Interfacet **IEvidenceManager** dikterer de nødvendige metoder, som systemet bruger med henhold til at hente data til beregninger af tillidsværdier og risikoværdier. Ideelt set ville **IEvidenceManager** realiseres af det kapsulerende systems datahåndtering. Som det kan ses på figur 9, så har
textbfIEvidenceManager metoden getEvidence. Denne metode er ment som en mere universel metode til hente data, så i realiseringen **EvidenceManager**, er getEvidence i stand til at hente alt gemt data. Interfacet tilbyder også mere specifikke metoder som getTrustLevel og getRecommendation, som er metoder ment til at mindske den nødvendige datamanipulering i tillidsmodellen, og derved gøre det nemmere at implementere beregningsalgoritmen.

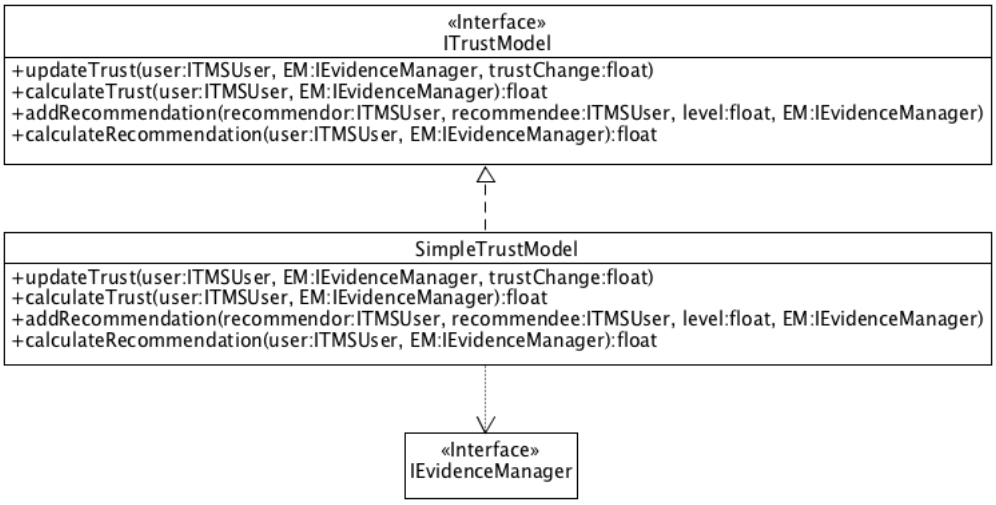
5.4 IEntityRelationManager.java



Figur 10: UML diagram, der viser relationene mellem IRelationmanager samt dens realisering og bruger interface

I en ideel implementering ville **IEntityRelationManager** være realiseret i det omkringliggende systems brugerdatabase, da det også ville give mulighed for bedre user authentication. Som det kan ses på UML'et i figur 10, så realiseres interfacet **IEntityRelationManager** af klassen **EntityRelationManager**, som i denne implementering blot er en simpel klasse, der indeholder et array med brugere.

5.5 ITrustModel.java



Figur 11: Udvidet UML diagram for TrustModel

ITrustModel dikterer de basale funktionaliteter, der understøttes af en tillidsmodel. I denne realisering implementeres **ITrustModel** af **SimpleTrustModel**, som det kan ses på figur 11. Når calculateTrust metoden kaldes på **SimpleTrustModel** beregnes tillidsværdien med algoritmen, som kan ses på kode 3

```
1     @Override
2   public float calculateTrust(ITMSUser user,
       IEvidenceManager EM) {
3     float trust = EM.getTrustLevel(user);
4     ArrayList<EvidenceModel> recommendations = EM.
         getEvidence(user, DataType.RECOMMENDATION);
5     int n = recommendations.size() + 1;
6
7     float sum = trust;
8
9     for (int i = 0; i < recommendations.size(); i++){
10      System.out.println("Sum: " + sum + ", n: " + n + ",
           Recommendation TrustLevel: " + EM.getTrustLevel(
           recommendations.get(i).recommendor));
11      sum += recommendations.get(i).dataValue*EM.
           getTrustLevel(recommendations.get(i).recommendor)
           ;
12    }
13
14    return sum/n;
15  }
```

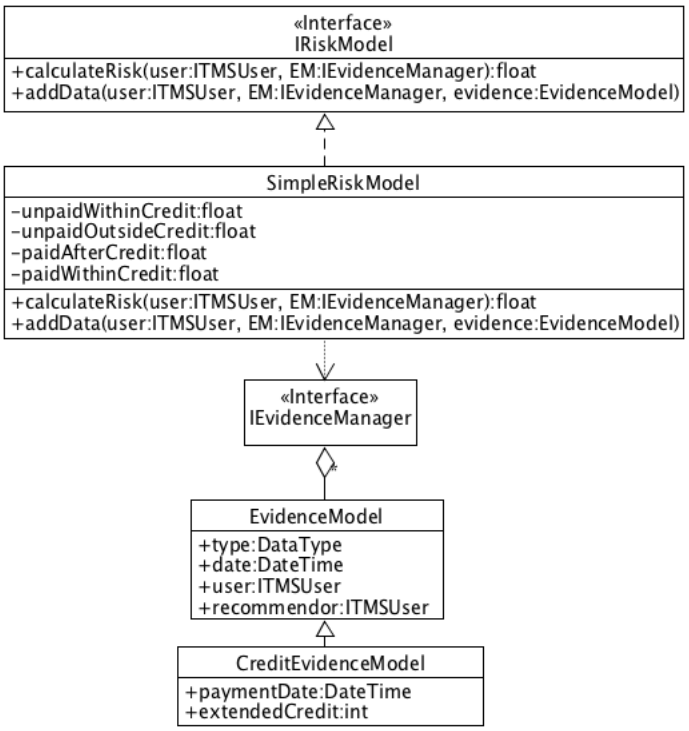
Code 3: Implementering af algoritmen til beregning af en købers tillidsværdi.

Som det kan ses i kode 3 er beregningen baseret på dataen der bliver hentet fra **IEvidence-Manager**. Ved forespørgelse på en anbefaling fra systemet, kan det ses i kode 4, at det i dette system blot er implementeret som systemets egen tillidsværdi for den givne bruger.

```
1     @Override
2   public float calculateRecommendation(ITMSUser user,
       IEvidenceManager EM) {
3     return calculateTrust(user,EM);
4   }
```

Code 4: Ved forespørgelse på en anbefaling returneres tillidsværdien for brugeren blot.

5.6 IRiskModel.java

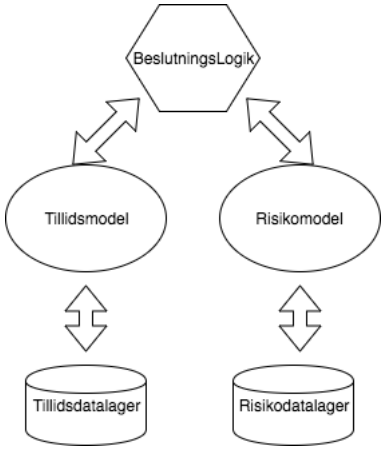


Figur 12: Udvidet UML diagram for RiskModel

Som det kan ses på figur 12, så har **IRiskModel** to metoder. **IRiskModel** har en metode til beregning af risiko og en metode til at tilføje potentielt data. I realiseringen **SimpleRiskModel** er addData metoden ikke implementeret, men i stedet kaster den en UnsupportedOperationException. Metoden er en del af **IRiskModel** for at give muligheden for at bruge modeller, det er baseret på extern data. I figur 12 kan det ses at klassen **CreditEvidenceModel** nedarver standard datamodellen **EvidenceModel**. Dette giver muligheden for at risikomodeller har ikke generiske datamodeller, hvilket giver flere muligheder for at understøtte modeller.

6 Diskussion

Universelle TMS'er
Den ovenstående implementation underbygger den ønskede funktionalitet, hvor det er muligt af udskifte tillidsmodellen og risikomodellen uafhængigt af hindanden. Med den interne relation, som vist på figur 5, kan det ses, at tillidsmodellen og risikomodellen stadig er afhængig af datalagret. Dette kunne eventuelt løses ved at introducere individuelle datalagre, som på figur 13



Figur 13: Alternativ opbygning med individuelle datalagre.

Denne opbygning ville give fordelen, at man kan lave mere præcise database integrationer, hvor datalagret kun indeholder de nødvendige funktionaliteter, og man ville ved risikomodellerne være i stand til at oprette mere passende datamodeller. På trods af at modellen i figur 13 ville give en klarere opdeling af modellerne, så ville det i reelle implementationer alligevel pege på den samme database, og da datamodellen kan ændre sig meget på baggrund af dataet nødvendig i risikomodellen, virker det som et mere solidt valg, at bruge en fælles datalager adgang og lave nedarvninger, hvis datamodellen ikke er dækkende.

Som vist på figur 5, så er beslutningslogikken samlingspunktet for modellerne, hvilket stiller spørgsmålet om, hvorvidt beslutningslogikken også kan gøres dynamisk. På det designmæssige niveau ville det være simpelt at implementere beslutningslogikken gennem et interface. Da beslutningslogikken er tæt bundet til konteksten, ville en ændring til en dynamisk implementering af beslutningslogik medføre at konteksten ikke længere ville være ligeså stærk defineret. Hvis man skulle introducere et mere universelt trust management system framework, hvor tillidsmodel, risikomodel, og beslutningslogik var dynamiske, ville tillidsmodellen og risikomodellen stadig være afhængige af beslutningslogikken, medmindre der forslås en anderledes opsætning af trust management systemer. I den nuværende opsætning ville den første komponent der skulle udvikles være beslutningslogikken, som efterfølgende ville diktere udviklingen af tillidsmodellen og risikomodellen, derved ender man alligevel med, hvad der svarer til et framework med en kontekst.
Derfor giver det ikke nødvendigvis mening at bestræbe sig på at skabe et mere universelt framework, da det også ville fjerne det meste af grundlaget for at have et framework.

7 Konklusion

Det implementerede trust management system framework præsterede at opnå en løs kobling af tillidsmodellen og risikomodellen. Tillidsmodeller og risikomodeller er i systemet uafhængige af hinanden, men der er nogle krav mellem beslutningslogikken og modellerne. Modellerne er nød til at overholde formatet for værdierne de beregner. Disse værdiformater fastlagt i beslutningslogikken, og i den konkrete implementering er kravet at det er en float værdi mellem 0 og 1.
Som i implementeringens kontekst, så har risikomodellen og tillidsmodellen ikke nødvendigvis ens data. Dette kan som vist i figur 12 løses ved at nedarve en generel datamodel, og derved udvide med de nødvendige tilføjede datafelter.
Med hensynet til beslutningslogikkens format og nedarvning i datalagret, kan det ses at et dynamisk framework kan skabes for en kontekst. Dette kunne være et system, der ville være simpelt at implementere i fungerende miljøer med en konkret kontekst, hvorimod som der blev diskuteret, så ville det være problematisk at udvide frameworket til at være universelt, da det i sidste ende ville fjerne konceptet i framework, ved ikke længere at tilbyde en simpel implementation af trust management.

Litteratur

[1] Ping Dong, Hongchao Wang, Hongke Zhang. Probability-based trust management model for distributed e-commerce. *2009 IEEE International Conference on Network Infrastructure and Digital Content*, pages 419–423, 2009.

[2] Baptiste Alcalde, Eric Dubois, Sjouke Mauw, et al. Towards a decision model based on trust and security risk management. *Conferences in Research and Practice in Information Technology Series*, 98:61–69, 2009.

[3] Cao Yonghui. Study of designing the trust management system. *Biotechnology: an Indian Journal*, 8(2):215–218, 2013.

[4] Soon Keow Chong, Jemal Abawajy. E-commerce trust management system reliability. *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining*, pages 13–18, 2012.

[5] Andrew G. West, Adam J. Aviv, Jian Chang, et al. Quantm: A quantitative trust management system. *The 2nd European Workshop on System Security*, pages 28–35, 2009.

[6] S. Thamarai Selvi, R. Kumar, K. Rajendar. Behavioral trust management system. *2008 International Conference on Grid Computing and Applications*, pages 147–152, 2008.

[7] Yong Wang, Ming Li, Ling Guo Cui, et al. Context-aware trust management system. *Beijing Ligong Daxue Xuebao/transaction of Beijing Institute of Technology*, 28(2):95–96+115.

[8] Yao Wang, Julita Vassileva. Bayesian network-based trust model. *IEEE/WIC International Conference on Web Intelligence*, 2003.

[9] Shashi Bhanwar, Seema Bawa,. Reputation enhancement in a trust management system.

[10] Hisham Salah, Mohamed Eltoweissy. Towards a personalized trust management system. *2012 International Conference on Innovations in Information Technology*, page 6207771, 2012.

[11] JM Seigneur, CD Jensen. Trading privacy for trust. *Lecture Notes in Computer Science*, 2995:93–107, 2004.

[12] Research and application of trust management system.

[13] Ryma Abassi, Sihem Guemara El Fatmi. Towards a generic trust management model. *2012 19th International Conference on Telecommunications*, page 6221228, 2012.

A Kildekode

Den overordnede struktur i kildekoden er:

- baseComponents
  - DecisionLogic.java
  - EntityRelationManager.java
  - EvidenceManager.java
  - TrustManagementSystem.java
- enums
  - DataType.java
  - RiskLevel.java
- exceptions
  - FunctionalityNotSupportedException.java
  - IllegalRecommendationException.java
  - UnknownEvidenceTypeException.java
  - UnknownUserException.java
- implementationExample
  - SimpleRiskModel.java
  - SimpleTrustModel.java
- interfaces
  - IEntityRelationManager.java
  - IEvidenceManager.java
  - IRiskModel.java
  - ITMSUser.java
  - ITrustModel.java
- models
  - CreditEvidenceModel.java
  - EvidenceModel.java

A.1 BaseComponents

A.1.1 DecisionLogic.java

```
1  package baseComponents;
2
3  import enums.RiskLevel;
4  import interfaces.IEvidenceManager;
5  import interfaces.IRiskModel;
6  import interfaces.ITMSUser;
7  import interfaces.ITrustModel;
8
9  public class DecisionLogic{
10
11   private float lowRisk = 0f;
12   private float mediumRisk = 0.5f;
13   private float highRisk = 0.8f;
14
15   private float lowRiskSuccess = 0.03f;
16   private float lowRiskFailure = -0.075f;
17   private float mediumRiskSuccess = 0.05f;
18   private float mediumRiskFailure = -0.125f;
19   private float highRiskSuccess = 0.08f;
20   private float highRiskFailure = -0.2f;
21
22
23   public boolean decideResponse(RiskLevel riskLevel,
        ITMSUser user, ITrustModel TM, IRiskModel RM,
        IEvidenceManager EM) {
24     switch(riskLevel){
25       case LOWRISK:
26         return lowRiskDecision(user, TM, RM, EM);
27       case MEDIUMRISK:
28         return mediumRiskDecision(user, TM, RM, EM);
29       case HIGHRISK:
30         return highRiskDecision(user, TM, RM, EM);
31     }
32     throw new UnsupportedOperationException();
33   }
34
35
36   private boolean lowRiskDecision(ITMSUser user,
        ITrustModel TM, IRiskModel RM, IEvidenceManager EM){
37     if(TM.calculateTrust(user, EM) >= lowRisk && RM.
          calculateRisk(user, EM) >= lowRisk){
38       return true;
39     }
40     return false;
41   }
42
43
```

```
44    private boolean mediumRiskDecision(ITMSUser user,
          ITrustModel TM, IRiskModel RM, IEvidenceManager EM){
45      if(TM.calculateTrust(user, EM) >= mediumRisk && RM.
            calculateRisk(user, EM) >= mediumRisk){
46        return true;
47      }
48      return false;
49    }
50
51
52    private boolean highRiskDecision(ITMSUser user,
          ITrustModel TM, IRiskModel RM, IEvidenceManager EM){
53      if(TM.calculateTrust(user, EM) >= highRisk && RM.
            calculateRisk(user, EM) >= highRisk){
54        return true;
55      }
56      return false;
57    }
58
59
60    public float decideTrustChange(RiskLevel riskLevel,
          boolean success) {
61      switch(riskLevel){
62        case LOWRISK:
63          if(success){
64            return lowRiskSuccess;       //0.03
65          } else {
66            return lowRiskFailure;     //-0.075
67          }
68        case MEDIUMRISK:
69          if(success){
70            return mediumRiskSuccess;      //0.05
71          } else {
72            return mediumRiskFailure;    //-0.125
73          }
74        case HIGHRISK:
75          if(success){
76            return highRiskSuccess;      //0.08
77          } else {
78            return highRiskFailure;      //-0.2
79          }
80      }
81      throw new UnsupportedOperationException();
82    }
83
84  }
```

A.1.2 EntityRelationManager.java

```
1  package implementationExample;
2
3  import java.util.ArrayList;
4
5  import interfaces.IEntityRelationManager;
6  import interfaces.ITMSUser;
7
8  public class EntityRelationManager implements
       IEntityRelationManager {
9
10
11   private ArrayList<ITMSUser> Users = new ArrayList<
         ITMSUser>();
12
13   public boolean validateUser(ITMSUser user){
14     for (int i = 0; i < Users.size(); i++){
15       if((Users.get(i)).getUsername() == user.getUsername
             () ){
16         return true;
17       }
18     }
19     return false;
20   }
21
22   public void addUser(ITMSUser user){
23     Users.add(user);
24   }
25
26 }
```

A.1.3 EvidenceManager.java

```
1  package implementationExample;
2
3  import java.util.ArrayList;
4
5  import enums.DataType;
6  import interfaces.IEvidenceManager;
7  import interfaces.ITMSUser;
8  import models.EvidenceModel;
9
10 public class EvidenceManager implements IEvidenceManager {
11
12   private ArrayList<EvidenceModel> evidence;
13
```

```
14  public EvidenceManager(){
15    this.evidence = new ArrayList<EvidenceModel>();
16  }
17
18  public ArrayList<EvidenceModel> getEvidence(ITMSUser
       user, DataType type){
19    ArrayList<EvidenceModel> result = new ArrayList<
         EvidenceModel>();
20
21    for (int i = 0; i < evidence.size(); i++){
22      if(evidence.get(i).user.getUsername() == user.
           getUsername() && evidence.get(i).type == type){
23        result.add(evidence.get(i));
24      }
25    }
26
27    return result;
28  }
29
30  public void addEvidence(EvidenceModel e){
31    evidence.add(e);
32  }
33
34  public float getTrustLevel(ITMSUser user){
35    ArrayList<EvidenceModel> eList = getEvidence(user,
         DataType.TRUSTLEVEL);
36    float result = 0;
37    if (!eList.isEmpty()){
38      result = eList.get(0).dataValue;
39    }
40    return result;
41  }
42
43  public void updateTrustLevel(ITMSUser user, float
       trustLevel){
44    getEvidence(user, DataType.TRUSTLEVEL).get(0).
         dataValue = trustLevel;
45  }
46
47  public EvidenceModel getRecommendation(ITMSUser
       recommendor, ITMSUser recommendee){
48    ArrayList<EvidenceModel> evidence = getEvidence(
         recommendee, DataType.RECOMMENDATION);
49    for (int i = 0; i < evidence.size(); i++){
50      if(evidence.get(i).recommendor.getUsername() ==
           recommendor.getUsername()){
51        return evidence.get(i);
52      }
53    }
54    return null;
```

```
55    }
56 }
```

A.1.4 TrustManagementSystem.java

```
1  package baseComponents;
2
3  import enums.RiskLevel;
4  import exceptions.FunctionalityNotSupportedException;
5  import exceptions.IllegalRecommendationException;
6  import exceptions.UnknownUserException;
7  import interfaces.IEntityRelationManager;
8  import interfaces.IEvidenceManager;
9  import interfaces.IRiskModel;
10 import interfaces.ITMSUser;
11 import interfaces.ITrustModel;
12 import models.EvidenceModel;
13
14 public class TrustManagementSystem {
15
16   private IRiskModel RM;
17   private ITrustModel TM;
18   private DecisionLogic DL;
19   private IEvidenceManager EM;
20   private IEntityRelationManager ER;
21   private ITMSUser owner;
22
23
24   public TrustManagementSystem(IRiskModel RM, ITrustModel
        TM, DecisionLogic DL,
25       IEvidenceManager EM, IEntityRelationManager ER,
             ITMSUser owner){
26     this.TM = TM;
27     this.RM = RM;
28     this.DL = DL;
29     this.EM = EM;
30     this.ER = ER;
31     this.owner = owner;
32   }
33
34
35   /*
36    * If the user doesn't exist, then it will be added.
37    */
38
39   public boolean requestTransaction(ITMSUser user,
        RiskLevel riskLevel) throws
        FunctionalityNotSupportedException{
```

```
40      if(!ER.validateUser(user)){
41        ER.addUser(user);
42      }
43
44      return DL.decideResponse(riskLevel, user, TM, RM, EM);
45    }
46
47
48    public boolean requestTransaction(ITMSUser user,
          RiskLevel riskLevel, EvidenceModel data) throws
          FunctionalityNotSupportedException{
49      if(!ER.validateUser(user)){
50        ER.addUser(user);
51      }
52
53      RM.addData(user, EM, data);
54      return requestTransaction(user, riskLevel);
55    }
56
57
58    public void reportTransaction(ITMSUser user, RiskLevel
          riskLevel, boolean success){
59      if(!ER.validateUser(user)){
60        ER.addUser(user);
61      }
62
63      TM.updateTrust(user, EM, DL.decideTrustChange(
            riskLevel, success));
64    }
65
66
67    public void reportTransaction(ITMSUser user, RiskLevel
          riskLevel, boolean success, EvidenceModel data){
68      if(!ER.validateUser(user)){
69        ER.addUser(user);
70      }
71      RM.addData(user, EM, data);
72
73      reportTransaction(user, riskLevel, success);
74    }
75
76
77
78    /*
79     * This method will add the recommendation of the
           recommendor about the recommendee to the
80     * Trust Model.
81     * If the recommendor is unknown, it will throw a
           UnknownUserException.
```

```
82      * If the recommendee is unknown, it will be added to
            the EntityRelationManager.
83      */
84    public void recommendUser(ITMSUser recommendor, ITMSUser
          recommendee, float level) throws
        UnknownUserException,
        FunctionalityNotSupportedException,
        IllegalRecommendationException{
85      if(recommendor.getUsername() == recommendee.
            getUsername()){
86        throw new IllegalRecommendationException("It's is
            not possible for a user to recommend themselves")
            ;
87      }
88
89      if(!ER.validateUser(recommendor)){
90        throw new UnknownUserException(recommendor.
            getUsername());
91      }
92
93      if(!ER.validateUser(recommendee)){
94        ER.addUser(recommendee);
95      }
96
97      TM.addRecommendation(recommendor, recommendee, level,
            EM);
98    }
99
100
101
102   public void requestRecommendation(TrustManagementSystem
          sys, ITMSUser recommendee) throws
        UnknownUserException,
        FunctionalityNotSupportedException,
        IllegalRecommendationException{
103     if(!ER.validateUser(recommendee)){
104       throw new UnknownUserException(recommendee.
            getUsername());
105     }
106     float trustValue = TM.calculateTrust(recommendee, EM);
107     sys.recommendUser(owner, recommendee, trustValue);
108   }
109
110
111 }
```

A.2 Enums

A.2.1 DataType.java

```
1 package enums;
2
3 public enum DataType {
4   RECOMMENDATION, TRUSTLEVEL, DATA
5 }
```

A.2.2 RiskLevel.java

```
1 package enums;
2
3 public enum RiskLevel {
4   LOWRISK,MEDIUMRISK,HIGHRISK
5 }
```

A.3 Exceptions

A.3.1 FunctionalityNotSupportedException.java

```
1  package exceptions;
2
3  public class FunctionalityNotSupportedException extends
       Exception {
4
5    /**
6     *
7     */
8    private static final long serialVersionUID =
         7193254113664121768L;
9
10   public FunctionalityNotSupportedException(String message
         ){
11     super(message);
12   }
13
14 }
```

A.3.2 IllegalRecommendationException.java

```
1  package exceptions;
2
3  public class IllegalRecommendationException extends
       Exception{
4    /**
5     *
6     */
7    private static final long serialVersionUID =
       4892046857758870060L;
8
9    public IllegalRecommendationException(String message){
10     super(message);
11   }
12 }
```

A.3.3 UnknownEvidenceTypeException.java

```
1  package exceptions;
2
3  public class UnknownEvidenceTypeException extends
       Exception {
4
5    /**
6     *
7     */
8    private static final long serialVersionUID =
       8206663195157291888L;
9
10   public UnknownEvidenceTypeException(String message){
11     super(message);
12   }
13 }
```

A.3.4 UnknownUserException.java

```
1  package exceptions;
2
3  public class UnknownUserException extends Exception {
4    /**
5     *
6     */
7    private static final long serialVersionUID =
       2592790859798989 47L;
8
```

```
9   public UnknownUserException(String message){
10    super("Unknown user: " + message);
11  }
12 }
```

A.4 ImplementationExample

A.4.1 SimpleRiskModel.java

```
1  package implementationExample;
2
3  import java.util.ArrayList;
4  import java.util.Date;
5
6  import org.joda.time.DateTime;
7  import org.joda.time.Days;
8
9  import enums.DataType;
10 import interfaces.IEvidenceManager;
11 import interfaces.IRiskModel;
12 import interfaces.ITMSUser;
13 import models.CreditEvidenceModel;
14 import models.EvidenceModel;
15
16 public class SimpleRiskModel implements IRiskModel{
17
18   private float unpaidWithinCredit = 0.75f;
19   private float unpaidOutsideCredit = 0;
20   private float paidAfterCredit = 0.5f;
21   private float paidWithinCredit = 1;
22
23
24   /*
25    * The risk is calculated as a number between 0 and 1
        weighted by purchase size and payment date.
26    * purchases that are unpaid but within the creditwindow
        are denoted by     unpaidWithinCredit
27    * purchases that are unpaid and past the credit window
        are denoted by      unpaidOutsideCredit
28    * purchases that are paid but after the credit window
        are denoted by       unpaidAfterCredit
29    * purchases that are paid within the credit window are
        denoted by        paidWithinCredit
30    */
31   @Override
32   public float calculateRisk(ITMSUser user,
       IEvidenceManager EM) {
```

```
33      ArrayList<EvidenceModel> data = EM.getEvidence(user,
            DataType.DATA);
34      if(data.isEmpty()){
35        return 1;
36      }
37      ArrayList<CreditEvidenceModel> creditEvidence = new
            ArrayList<CreditEvidenceModel>();
38      for (int i = 0; i < data.size(); i++){
39        if(data.get(i) instanceof CreditEvidenceModel){
40          creditEvidence.add((CreditEvidenceModel) data.get(
                i));
41        }
42      }
43      float totalPurchaseAmount = 0;
44      for(int i = 0; i < creditEvidence.size(); i++){
45        totalPurchaseAmount += creditEvidence.get(i).
              dataValue;
46      }
47      float sum = 0;
48      for(int i = 0; i < creditEvidence.size(); i++){
49        float weight = 0;
50        CreditEvidenceModel c = creditEvidence.get(i);
51        int creditWindow = 30 + c.extendedCredit;
52        DateTime purchaseDate = c.date;
53        DateTime paymentDate = c.paymentDate;
54
55        //Denotations of the previous purchases.
56        if(paymentDate == null){
57          int daysPassed = Days.daysBetween(purchaseDate,
                new DateTime(new Date())).getDays();
58          if(daysPassed < creditWindow){
59            weight = unpaidWithinCredit;
60          } else {
61            weight = unpaidOutsideCredit;
62          }
63        } else {
64          int daysPassed = Days.daysBetween(purchaseDate,
                paymentDate).getDays();
65          if (daysPassed < creditWindow){
66            weight = paidWithinCredit;
67          } else {
68            weight = paidAfterCredit;
69          }
70        }
71        sum += weight*(c.dataValue/totalPurchaseAmount);
72
73      }
74
75      return sum/creditEvidence.size();
76    }
```

```
77
78    @Override
79    public void addData(ITMSUser user, IEvidenceManager EM,
          EvidenceModel evidence) {
80      throw new UnsupportedOperationException();
81    }
82
83  }
```

A.4.2 SimpleTrustModel.java

```
1   package implementationExample;
2
3   import java.util.ArrayList;
4   import java.util.Date;
5
6   import org.joda.time.DateTime;
7
8   import enums.DataType;
9   import interfaces.IEvidenceManager;
10  import interfaces.ITMSUser;
11  import interfaces.ITrustModel;
12  import models.EvidenceModel;
13
14  public class SimpleTrustModel implements ITrustModel {
15
16    /*
17     * The model update by adding the value of the
           trustLevel to a maximum
18     * of 3. The cap is in order to disable the possibility
           of saving up to
19     * make future bad transaction without a consequence.
20     */
21    @Override
22    public void updateTrust(ITMSUser user, IEvidenceManager
          EM, float trustChange) {
23      EvidenceModel evidence = EM.getEvidence(user, DataType
            .TRUSTLEVEL).get(0);
24      EM.updateTrustLevel(user, Math.min(evidence.dataValue+
            trustChange, 1));
25    }
26
27    /*
28     * Trust is calculated as (3*T+sum_i=1^(n\{owner,trustee
           })w_i*L_i)/n-1
29     * where T is the old trust value, L is the level of
           trust from use i and w is the weigth
30     * the owner prescribes to i's recommendation.
```

```
31      */
32     @Override
33     public float calculateTrust(ITMSUser user,
           IEvidenceManager EM) {
34       float trust = EM.getTrustLevel(user);
35       ArrayList<EvidenceModel> recommendations = EM.
             getEvidence(user, DataType.RECOMMENDATION);
36       int n = recommendations.size() + 1;
37
38       float sum = trust;
39
40       for (int i = 0; i < recommendations.size(); i++){
41         System.out.println("Sum: " + sum + ", n: " + n + ",
             Recommendation TrustLevel: " + EM.getTrustLevel(
             recommendations.get(i).recommendor));
42         sum += recommendations.get(i).dataValue*EM.
             getTrustLevel(recommendations.get(i).recommendor)
             ;
43       }
44
45       return sum/n;
46     }
47
48
49     @Override
50     public void addRecommendation(ITMSUser recommendor,
           ITMSUser recommendee, float level, IEvidenceManager
           EM) {
51       EvidenceModel recommendation = EM.getRecommendation(
             recommendor, recommendee);
52       if( recommendation == null){
53         recommendation = new EvidenceModel();
54         recommendation.type = DataType.RECOMMENDATION;
55         recommendation.date = new DateTime(new Date());
56         recommendation.user = recommendee;
57         recommendation.recommendor = recommendor;
58         recommendation.dataValue = Math.min(level, 1);
59         EM.addEvidence(recommendation);
60       } else {
61         recommendation.dataValue = Math.min(level, 1);
62         recommendation.date = new DateTime(new Date());
63       }
64     }
65
66     /*
67      * In this model recommendation level is equal to the
            trust level.
68      */
69     @Override
```

```
70    public float calculateRecommendation(ITMSUser user,
          IEvidenceManager EM) {
71      return calculateTrust(user,EM);
72    }
73
74  }
```

A.5 Interfaces

A.5.1 IEntityRelationManager.java

```
1  package interfaces;
2
3  public interface IEntityRelationManager {
4
5    public boolean validateUser(ITMSUser user);
6
7    public void addUser(ITMSUser user);
8
9  }
```

A.5.2 IEvidenceManager.java

```
1  package interfaces;
2
3  import java.util.ArrayList;
4
5  import enums.DataType;
6  import models.EvidenceModel;
7
8  public interface IEvidenceManager{
9
10   public ArrayList<EvidenceModel> getEvidence(ITMSUser
         user, DataType type);
11
12   public void addEvidence(EvidenceModel e);
13
14   public float getTrustLevel(ITMSUser user);
15
16   public void updateTrustLevel(ITMSUser user, float
         trustLevel);
17
18   public EvidenceModel getRecommendation(ITMSUser
         recommendor, ITMSUser recommendee);
19 }
```

A.5.3 IRiskModel.java

```
1  package interfaces;
2
3  import models.EvidenceModel;
4
5  public interface IRiskModel {
6    public float calculateRisk(ITMSUser user,
       IEvidenceManager EM);
7
8    public void addData(ITMSUser user, IEvidenceManager EM,
       EvidenceModel evidence);
9
10
11 }
```

A.5.4 ITMSUser.java

```
1  package interfaces;
2
3  public interface ITMSUser {
4
5    public String getUsername();
6
7  }
```

A.5.5 ITrustModel.java

```
1  package interfaces;
2
3  public interface ITrustModel {
4    public void updateTrust(ITMSUser user, IEvidenceManager
       EM, float trustChange);
5
6    public float calculateTrust(ITMSUser user,
       IEvidenceManager EM);
7
8    public void addRecommendation(ITMSUser recommendor,
       ITMSUser recommendee, float level, IEvidenceManager
       EM);
```

```
9
10    public float calculateRecommendation(ITMSUser user,
          IEvidenceManager EM);
11
12
13  }
```

A.6 Models

A.6.1 CreditEvidenceModel.java

```
1  package models;
2
3  import org.joda.time.DateTime;
4
5
6  public class CreditEvidenceModel extends EvidenceModel{
7    public DateTime paymentDate;
8    public int extendedCredit;
9  }
```

A.6.2 EvidenceModel.java

```
1  package models;
2
3  import org.joda.time.DateTime;
4
5  import enums.DataType;
6  import interfaces.ITMSUser;
7
8  public class EvidenceModel {
9
10   public DataType type;
11   public DateTime date;
12   public ITMSUser user;
13   public ITMSUser recommendor;
14   public float dataValue;
15 }
```

B Testkode

B.1 FullSystemTest.java

```
1  package baseComponents;
2
3  import static org.junit.Assert.*;
4  import static org.mockito.Mockito.mock;
5  import static org.mockito.Mockito.verify;
6  import static org.mockito.Mockito.when;
7
8  import java.util.Date;
9
10 import org.joda.time.DateTime;
11 import org.junit.Before;
12 import org.junit.Test;
13 import org.junit.runner.RunWith;
14 import org.mockito.junit.MockitoJUnitRunner;
15
16 import baseComponents.TrustManagementSystem;
17 import enums.*;
18 import exceptions.FunctionalityNotSupportedException;
19 import exceptions.IllegalRecommendationException;
20 import exceptions.UnknownUserException;
21 import implementationExample.*;
22 import interfaces.*;
23 import models.*;
24
25 @RunWith(MockitoJUnitRunner.class)
26 public class FullSystemTest {
27
28   public TrustManagementSystem TMS;
29   public DecisionLogic DL;
30   public IEntityRelationManager ER;
31   public IEvidenceManager EM;
32   public IRiskModel RM;
33   public ITrustModel TM;
34   public ITMSUser mockedOwner;
35   public ITMSUser mockedUser;
36
37   @Before
38   public void setup(){
39     DL = new DecisionLogic();
40     ER = new EntityRelationManager();
41     EM = new EvidenceManager();
42     RM = new SimpleRiskModel();
43     TM = new SimpleTrustModel();
44     mockedOwner = mock(ITMSUser.class);
45
```

```
46      TMS = new TrustManagementSystem(RM, TM, DL, EM, ER,
            mockedOwner);
47
48      mockedUser = mock(ITMSUser.class);
49      when(mockedUser.getUsername()).thenReturn("B");
50    }
51
52    @Test
53    public void calculateTrustHighTrustNoRecommendationsTest
          () throws FunctionalityNotSupportedException {
54      EvidenceModel e = new EvidenceModel();
55      e.dataValue = 0.5f;
56      e.type = DataType.TRUSTLEVEL;
57      e.user = mockedUser;
58      ER.addUser(mockedUser);
59      EM.addEvidence(e);
60
61      assertEquals(true,TMS.requestTransaction(mockedUser,
            RiskLevel.MEDIUMRISK));
62    }
63
64    @Test
65    public void
          calculateTrustHighTrustWithHighRecommendationsTest()
          throws FunctionalityNotSupportedException{
66      EvidenceModel e1 = new EvidenceModel();
67      e1.dataValue = 1f;
68      e1.type = DataType.TRUSTLEVEL;
69      e1.user = mockedUser;
70      ER.addUser(mockedUser);
71      EM.addEvidence(e1);
72
73      ITMSUser mockedRecommendor = mock(ITMSUser.class);
74      when(mockedRecommendor.getUsername()).thenReturn("C");
75      EvidenceModel e2 = new EvidenceModel();
76      e2.dataValue = 1f;
77      e2.type = DataType.RECOMMENDATION;
78      e2.user = mockedUser;
79      e2.recommendor = mockedRecommendor;
80      ER.addUser(mockedRecommendor);
81      EM.addEvidence(e2);
82
83      EvidenceModel e3 = new EvidenceModel();
84      e3.dataValue = 1f;
85      e3.type = DataType.TRUSTLEVEL;
86      e3.user = mockedRecommendor;
87      EM.addEvidence(e3);
88
89
```

```
90      assertEquals(true, TMS.requestTransaction(mockedUser,
            RiskLevel.HIGHRISK));
91    }
92
93
94    @Test
95    public void
          calculateTrustHighTrustWithLowRecommendationsTest()
          throws FunctionalityNotSupportedException{
96      EvidenceModel e1 = new EvidenceModel();
97      e1.dataValue = 1f;
98      e1.type = DataType.TRUSTLEVEL;
99      e1.user = mockedUser;
100     ER.addUser(mockedUser);
101     EM.addEvidence(e1);
102
103     ITMSUser mockedRecommendor = mock(ITMSUser.class);
104     when(mockedRecommendor.getUsername()).thenReturn("C");
105     EvidenceModel e2 = new EvidenceModel();
106     e2.dataValue = 1f;
107     e2.type = DataType.RECOMMENDATION;
108     e2.user = mockedUser;
109     e2.recommendor = mockedRecommendor;
110     ER.addUser(mockedRecommendor);
111     EM.addEvidence(e2);
112
113     EvidenceModel e3 = new EvidenceModel();
114     e3.dataValue = 0f;
115     e3.type = DataType.TRUSTLEVEL;
116     e3.user = mockedRecommendor;
117     EM.addEvidence(e3);
118
119
120     assertEquals(false, TMS.requestTransaction(mockedUser,
            RiskLevel.HIGHRISK));
121   }
122
123
124   @Test
125   public void calculateTrustWithOnlyRiskUnpaid() throws
          FunctionalityNotSupportedException{
126     CreditEvidenceModel e = new CreditEvidenceModel();
127     e.date = new DateTime(new Date()).minusDays(35);
128     e.dataValue = 200;
129     e.type = DataType.DATA;
130     e.paymentDate = null;
131     e.user = mockedUser;
132
133     ER.addUser(mockedUser);
134     EM.addEvidence(e);
```

```
135
136     assertEquals(true, TMS.requestTransaction(mockedUser,
           RiskLevel.LOWRISK));
137
138     e.date = new DateTime(new Date()).minusDays(10);
139
140     assertEquals(true, TMS.requestTransaction(mockedUser,
           RiskLevel.LOWRISK));
141     assertEquals(false, TMS.requestTransaction(mockedUser,
            RiskLevel.MEDIUMRISK));
142
143     e.paymentDate = new DateTime(new Date());
144     assertEquals(false, TMS.requestTransaction(mockedUser,
            RiskLevel.MEDIUMRISK));
145   }
146
147
148   @Test
149   public void reportTransactionTest(){
150     EvidenceModel e = new EvidenceModel();
151     e.dataValue = 1f;
152     e.type = DataType.TRUSTLEVEL;
153     e.user = mockedUser;
154     ER.addUser(mockedUser);
155     EM.addEvidence(e);
156
157     TMS.reportTransaction(mockedUser, RiskLevel.HIGHRISK,
           false);
158
159     assertEquals(0.8, e.dataValue, 0.01);
160   }
161
162
163   @Test
164   public void recommendUserBadRecommendationTest() throws
          FunctionalityNotSupportedException,
          UnknownUserException, IllegalRecommendationException{
165     EvidenceModel e1 = new EvidenceModel();
166     e1.dataValue = 1f;
167     e1.type = DataType.TRUSTLEVEL;
168     e1.user = mockedUser;
169     ER.addUser(mockedUser);
170     EM.addEvidence(e1);
171
172     ITMSUser mockedRecommendor = mock(ITMSUser.class);
173     when(mockedRecommendor.getUsername()).thenReturn("C");
174     ER.addUser(mockedRecommendor);
175
176     EvidenceModel e3 = new EvidenceModel();
177     e3.dataValue = 0f;
```

```
178     e3.type = DataType.TRUSTLEVEL;
179     e3.user = mockedRecommendor;
180     EM.addEvidence(e3);
181
182     assertEquals(true, TMS.requestTransaction(mockedUser,
            RiskLevel.HIGHRISK));
183     TMS.recommendUser(mockedRecommendor, mockedUser, 1f);
184
185     assertEquals( false, TMS.requestTransaction(mockedUser
            , RiskLevel.HIGHRISK));
186   }
187
188
189   @Test
190   public void requestRecommendationTest() throws
          UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
191     TrustManagementSystem sys = mock(TrustManagementSystem
            .class);
192
193     EvidenceModel e = new EvidenceModel();
194     e.dataValue = 1f;
195     e.type = DataType.TRUSTLEVEL;
196     e.user = mockedUser;
197     ER.addUser(mockedUser);
198     EM.addEvidence(e);
199
200     TMS.requestRecommendation(sys, mockedUser);
201
202     verify(sys).recommendUser(mockedOwner, mockedUser, 1);
203   }
204
205 }
```

B.2 TrustManagementSystemTest.java

```
1  package baseComponents;
2
3  import static org.mockito.ArgumentMatchers.any;
4  import static org.mockito.ArgumentMatchers.anyFloat;
5  import static org.mockito.Mockito.mock;
6  import static org.mockito.Mockito.never;
7  import static org.mockito.Mockito.verify;
8  import static org.mockito.Mockito.when;
9
10 import org.junit.Before;
11 import org.junit.Test;
```

```
12 import org.junit.runner.RunWith;
13 import org.mockito.MockitoAnnotations;
14 import org.mockito.junit.MockitoJUnitRunner;
15
16 import baseComponents.TrustManagementSystem;
17 import enums.RiskLevel;
18 import exceptions.FunctionalityNotSupportedException;
19 import exceptions.IllegalRecommendationException;
20 import exceptions.UnknownUserException;
21 import interfaces.IEntityRelationManager;
22 import interfaces.IEvidenceManager;
23 import interfaces.IRiskModel;
24 import interfaces.ITMSUser;
25 import interfaces.ITrustModel;
26 import models.CreditEvidenceModel;
27
28 @RunWith(MockitoJUnitRunner.class)
29 public class TrustManagementSystemTest {
30
31   private IRiskModel mockedRM;
32   private ITrustModel mockedTM;
33   private DecisionLogic mockedDL;
34   private IEvidenceManager mockedEM;
35   private IEntityRelationManager mockedER;
36   private ITMSUser mockedOwner;
37
38   @Before
39   public void setup(){
40     this.mockedRM = mock(IRiskModel.class);
41     this.mockedTM = mock(ITrustModel.class);
42     this.mockedDL = mock(DecisionLogic.class);
43     this.mockedEM = mock(IEvidenceManager.class);
44     this.mockedER = mock(IEntityRelationManager.class);
45     this.mockedOwner = mock(ITMSUser.class);
46
47
48   }
49
50   /*
51    *
52    * requestTransaction Tests
53    *
54    */
55
56
57   @Test
58   public void requestTransactionWithKnownUserTest() throws
         FunctionalityNotSupportedException {
59     //Setup
60     ITMSUser mockedUser = mock(ITMSUser.class);
```

```
61      when(mockedER.validateUser(any(ITMSUser.class))).
            thenReturn(true);
62      MockitoAnnotations.initMocks(this);
63
64      TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
65
66      //The method call
67      sys.requestTransaction(mockedUser, RiskLevel.LOWRISK);
68
69
70      //Verification that there is no call to setup a new
            user in the ER.
71      verify(mockedER,never()).addUser(any());
72    }
73
74
75    @Test
76    public void requestTransactionWithUnknownUserTest()
         throws FunctionalityNotSupportedException{
77      ITMSUser mockedUser = mock(ITMSUser.class);
78      when(mockedER.validateUser(any(ITMSUser.class))).
            thenReturn(false);
79
80      TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
81
82      //The method call
83      sys.requestTransaction(mockedUser, RiskLevel.LOWRISK);
84
85      //Verification that there is a call to add a new user
            to the ER.
86      verify(mockedER).addUser(any());
87    }
88
89    /*
90     *
91     * reportTransaction Tests
92     *
93     */
94
95    @Test
96    public void reportTransactionWithKnownUser(){
97      ITMSUser mockedUser = mock(ITMSUser.class);
98      CreditEvidenceModel mockedEvidence= mock(
            CreditEvidenceModel.class);
99
```

```
100     when(mockedER.validateUser(mockedUser)).thenReturn(
            true);
101     when(mockedDL.decideTrustChange(RiskLevel.LOWRISK,
            true)).thenReturn((float)0.2);
102
103     TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
104
105     //The method call
106     sys.reportTransaction(mockedUser, RiskLevel.LOWRISK,
            true, mockedEvidence);
107
108     verify(mockedRM).addData(mockedUser, mockedEM,
            mockedEvidence);
109     verify(mockedER, never()).addUser(mockedUser);
110     verify(mockedTM).updateTrust(mockedUser, mockedEM, (
            float) 0.2);
111   }
112
113   @Test
114   public void reportTransactionWithUnknownUser(){
115     ITMSUser mockedUser = mock(ITMSUser.class);
116     CreditEvidenceModel mockedEvidence= mock(
            CreditEvidenceModel.class);
117
118     when(mockedER.validateUser(mockedUser)).thenReturn(
            true).thenReturn(false);
119     when(mockedDL.decideTrustChange(RiskLevel.LOWRISK,
            true)).thenReturn((float)-0.2);
120
121     TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
122
123     //The method call
124     sys.reportTransaction(mockedUser, RiskLevel.LOWRISK,
            true, mockedEvidence);
125
126     verify(mockedRM).addData(mockedUser, mockedEM,
            mockedEvidence);
127     verify(mockedER).addUser(mockedUser);
128     verify(mockedTM).updateTrust(mockedUser, mockedEM, (
            float)-0.2);
129   }
130
131
132
133   /*
134    *
```

```
135     * recommendUser Tests
136     *
137     */
138
139   @Test
140   public void recommendUserTest() throws
          UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
141     ITMSUser mockedRecommendor = mock(ITMSUser.class);
142     ITMSUser mockedRecommendee = mock(ITMSUser.class);
143     when(mockedER.validateUser(mockedRecommendor)).
          thenReturn(true);
144     when(mockedER.validateUser(mockedRecommendee)).
          thenReturn(true);
145     when(mockedRecommendor.getUsername()).thenReturn("A");
146     when(mockedRecommendee.getUsername()).thenReturn("B");
147
148     TrustManagementSystem sys = new TrustManagementSystem(
          mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
          mockedOwner);
149
150     sys.recommendUser(mockedRecommendor, mockedRecommendee
          , 1);
151
152     verify(mockedER, never()).addUser(any());
153     verify(mockedTM).addRecommendation(mockedRecommendor,
          mockedRecommendee, 1, mockedEM);
154     //verify(mockedTM).addRecommendation( any(), any(),
          any(), any());
155   }
156
157
158   @Test(expected=UnknownUserException.class)
159   public void recommendUserWithUnknownRecommendorTest()
          throws UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
160     ITMSUser mockedRecommendor = mock(ITMSUser.class);
161     ITMSUser mockedRecommendee = mock(ITMSUser.class);
162     when(mockedER.validateUser(mockedRecommendor)).
          thenReturn(false);
163     when(mockedRecommendor.getUsername()).thenReturn("A");
164     when(mockedRecommendee.getUsername()).thenReturn("B");
165
166     TrustManagementSystem sys = new TrustManagementSystem(
          mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
          mockedOwner);
167     sys.recommendUser(mockedRecommendor, mockedRecommendee
          , 1);
```

```
168   }
169
170
171   @Test
172   public void recommendUserWithUnknownRecommendeeTest()
          throws UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
173     ITMSUser mockedRecommendor = mock(ITMSUser.class);
174     ITMSUser mockedRecommendee = mock(ITMSUser.class);
175     when(mockedER.validateUser(mockedRecommendor)).
            thenReturn(true);
176     when(mockedER.validateUser(mockedRecommendee)).
            thenReturn(false);
177     when(mockedRecommendor.getUsername()).thenReturn("A");
178     when(mockedRecommendee.getUsername()).thenReturn("B");
179
180     TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
181     sys.recommendUser(mockedRecommendor, mockedRecommendee
            , 1);
182
183     verify(mockedER).addUser(mockedRecommendee);
184   }
185
186
187
188   /*
189    *
190    * requestRecommendation Tests
191    *
192    */
193
194
195   @Test
196   public void requestRecommendationTest() throws
          UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
197     ITMSUser mockedRecommendee = mock(ITMSUser.class);
198     TrustManagementSystem sys2 = mock(
            TrustManagementSystem.class);
199     when(mockedER.validateUser(mockedRecommendee)).
            thenReturn(true);
200
201
202     TrustManagementSystem sys = new TrustManagementSystem(
            mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
            mockedOwner);
```

```
203     sys.requestRecommendation(sys2, mockedRecommendee);
204
205     verify(sys2).recommendUser(any(), any(), anyFloat());
206   }
207
208
209   @Test(expected=UnknownUserException.class)
210   public void requestRecommendationWithUnknownUserTest()
          throws UnknownUserException,
          FunctionalityNotSupportedException,
          IllegalRecommendationException{
211     ITMSUser mockedRecommendee = mock(ITMSUser.class);
212     TrustManagementSystem sys2 = mock(
          TrustManagementSystem.class);
213     when(mockedER.validateUser(mockedRecommendee)).
          thenReturn(false);
214
215     TrustManagementSystem sys = new TrustManagementSystem(
          mockedRM, mockedTM, mockedDL, mockedEM, mockedER,
          mockedOwner);
216     sys.requestRecommendation(sys2, mockedRecommendee);
217   }
218
219
220 }
```

B.3 EntityRelationManagerTest.java

```
1  package baseComponents;
2
3  import static org.junit.Assert.assertEquals;
4  import static org.mockito.Mockito.mock;
5
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8  import org.mockito.junit.MockitoJUnitRunner;
9
10 import baseComponents.EntityRelationManager;
11 import interfaces.ITMSUser;
12
13
14 @RunWith(MockitoJUnitRunner.class)
15 public class EntityRelationManagerTest {
16
17   @Test
18   public void entityRelationManagerTest(){
19     EntityRelationManager ER = new EntityRelationManager()
          ;
```

```
20      ITMSUser mockedUser = mock(ITMSUser.class);
21
22      ER.addUser(mockedUser);
23      assertEquals(true, ER.validateUser(mockedUser));
24
25    }
26
27  }
```

B.4 EvidenceManagerTest.java

```
1   package baseComponents;
2
3   import static org.junit.Assert.assertEquals;
4   import static org.mockito.Mockito.mock;
5
6   import org.junit.Before;
7   import org.junit.Test;
8   import org.junit.runner.RunWith;
9   import org.mockito.junit.MockitoJUnitRunner;
10
11  import enums.DataType;
12  import baseComponents.EvidenceManager;
13  import interfaces.ITMSUser;
14  import models.EvidenceModel;
15
16
17  @RunWith(MockitoJUnitRunner.class)
18  public class EvidenceManagerTest {
19    public EvidenceManager EM;
20    public ITMSUser mockedUser;
21    public EvidenceModel mockedEvidence;
22
23    @Before
24    public void setup(){
25      EM = new EvidenceManager();
26      mockedEvidence = mock(EvidenceModel.class);
27      mockedUser = mock(ITMSUser.class);
28      mockedEvidence.user = mockedUser;
29      mockedEvidence.type = DataType.TRUSTLEVEL;
30      mockedEvidence.dataValue = 1;
31      EM.addEvidence(mockedEvidence);
32    }
33
34    @Test
35    public void AddGetEvidenceTest(){
36      assertEquals(mockedEvidence, EM.getEvidence(mockedUser
          , DataType.TRUSTLEVEL).get(0));
```

```
37    }
38
39    @Test
40    public void getTrustLevelTest(){
41      assertEquals(1, EM.getTrustLevel(mockedUser), 0.01);
42
43
44    }
45
46    @Test
47    public void updateTrustLevelTest(){
48      assertEquals(1,EM.getTrustLevel(mockedUser), 0.01);
49      EM.updateTrustLevel(mockedUser, 0);
50      assertEquals(0, EM.getTrustLevel(mockedUser), 0.01);
51    }
52
53    @Test
54    public void getRecommendationTest(){
55      mockedEvidence.type = DataType.RECOMMENDATION;
56      ITMSUser mockedRecommendor = mock(ITMSUser.class);
57      mockedEvidence.recommendor = mockedRecommendor;
58      assertEquals(mockedEvidence, EM.getRecommendation(
          mockedUser, mockedRecommendor));
59    }
60
61
62  }
```

B.5 DecisionLogicTest.java

```
1  package baseComponents;
2
3  import static org.junit.Assert.assertEquals;
4
5  import org.junit.Before;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
8  import org.mockito.junit.MockitoJUnitRunner;
9  import static org.mockito.Mockito.mock;
10 import static org.mockito.Mockito.when;
11
12 import enums.RiskLevel;
13 import interfaces.IEvidenceManager;
14 import interfaces.IRiskModel;
15 import interfaces.ITMSUser;
16 import interfaces.ITrustModel;
17 import baseComponents.DecisionLogic;
18
```

```
19 @RunWith(MockitoJUnitRunner.class)
20 public class DecisionLogicTest {
21
22   public DecisionLogic DL;
23   public ITMSUser mockedUser;
24   public IRiskModel mockedRM;
25   public ITrustModel mockedTM;
26   public IEvidenceManager mockedEM;
27
28   @Before
29   public void setup(){
30     DL = new DecisionLogic();
31     mockedUser = mock(ITMSUser.class);
32     mockedRM = mock(IRiskModel.class);
33     mockedTM = mock(ITrustModel.class);
34     mockedEM = mock(IEvidenceManager.class);
35
36   }
37
38
39   @Test
40   public void decideResponseTest(){
41     when(mockedTM.calculateTrust(mockedUser, mockedEM)).
          thenReturn(0.3f);
42     when(mockedRM.calculateRisk(mockedUser, mockedEM)).
          thenReturn(0.3f);
43     assertEquals( true, DL.decideResponse(RiskLevel.
          LOWRISK, mockedUser, mockedTM, mockedRM, mockedEM))
          ;
44
45     when(mockedTM.calculateTrust(mockedUser, mockedEM)).
          thenReturn(0.3f);
46     when(mockedRM.calculateRisk(mockedUser, mockedEM)).
          thenReturn(0f);
47     assertEquals( true, DL.decideResponse(RiskLevel.
          LOWRISK, mockedUser, mockedTM, mockedRM, mockedEM))
          ;
48
49     when(mockedTM.calculateTrust(mockedUser, mockedEM)).
          thenReturn(0f);
50     assertEquals( true, DL.decideResponse(RiskLevel.
          LOWRISK, mockedUser, mockedTM, mockedRM, mockedEM))
          ;
51
52     when(mockedTM.calculateTrust(mockedUser, mockedEM)).
          thenReturn(0.5f);
53     when(mockedRM.calculateRisk(mockedUser, mockedEM)).
          thenReturn(0.5f);
54     assertEquals( true, DL.decideResponse(RiskLevel.
          MEDIUMRISK, mockedUser, mockedTM, mockedRM,
```

```
        mockedEM));
55
56      when(mockedTM.calculateTrust(mockedUser, mockedEM)).
           thenReturn(0.5f);
57      when(mockedRM.calculateRisk(mockedUser, mockedEM)).
           thenReturn(0f);
58      assertEquals( false, DL.decideResponse(RiskLevel.
           MEDIUMRISK, mockedUser, mockedTM, mockedRM,
           mockedEM));
59
60      when(mockedTM.calculateTrust(mockedUser, mockedEM)).
           thenReturn(0f);
61      assertEquals( false, DL.decideResponse(RiskLevel.
           MEDIUMRISK, mockedUser, mockedTM, mockedRM,
           mockedEM));
62
63      when(mockedTM.calculateTrust(mockedUser, mockedEM)).
           thenReturn(0.8f);
64      when(mockedRM.calculateRisk(mockedUser, mockedEM)).
           thenReturn(0.8f);
65      assertEquals( true, DL.decideResponse(RiskLevel.
           HIGHRISK, mockedUser, mockedTM, mockedRM, mockedEM)
           );
66
67      when(mockedTM.calculateTrust(mockedUser, mockedEM)).
           thenReturn(0.8f);
68      when(mockedRM.calculateRisk(mockedUser, mockedEM)).
           thenReturn(0f);
69      assertEquals( false, DL.decideResponse(RiskLevel.
           HIGHRISK, mockedUser, mockedTM, mockedRM, mockedEM)
           );
70
71      when(mockedTM.calculateTrust(mockedUser, mockedEM)).
           thenReturn(0f);
72      assertEquals( false, DL.decideResponse(RiskLevel.
           HIGHRISK, mockedUser, mockedTM, mockedRM, mockedEM)
           );
73    }
74
75    @Test
76    public void decideTrustChangeTest(){
77      assertEquals( DL.decideTrustChange(RiskLevel.LOWRISK,
           true), 0.03f, 0.001f);
78      assertEquals( DL.decideTrustChange(RiskLevel.LOWRISK,
           false), -0.075f, 0.001f);
79      assertEquals( DL.decideTrustChange(RiskLevel.
           MEDIUMRISK, true), 0.05f, 0.001f);
80      assertEquals( DL.decideTrustChange(RiskLevel.
           MEDIUMRISK, false), -0.125f, 0.001f);
```

```
81      assertEquals( DL.decideTrustChange(RiskLevel.HIGHRISK,
            true), 0.08f, 0.001f);
82      assertEquals( DL.decideTrustChange(RiskLevel.HIGHRISK,
            false), -0.2f, 0.001f);
83    }
84
85  }
```

B.6 SimpleRiskModel.java

```
1   package implementationExamples;
2
3   import static org.junit.Assert.assertEquals;
4   import static org.mockito.Mockito.mock;
5   import static org.mockito.Mockito.when;
6
7   import java.util.ArrayList;
8   import java.util.Date;
9
10  import org.joda.time.DateTime;
11  import org.junit.Before;
12  import org.junit.Test;
13  import org.junit.runner.RunWith;
14  import org.mockito.junit.MockitoJUnitRunner;
15
16  import enums.DataType;
17  import implementationExample.SimpleRiskModel;
18  import interfaces.IEvidenceManager;
19  import interfaces.ITMSUser;
20  import models.CreditEvidenceModel;
21  import models.EvidenceModel;
22
23  @RunWith(MockitoJUnitRunner.class)
24  public class SimpleRiskModelTest {
25
26    public SimpleRiskModel RM;
27    public IEvidenceManager mockedEM;
28    public ITMSUser mockedUser;
29    public ArrayList<EvidenceModel> evidence;
30
31
32    @Before
33    public void setup(){
34      RM = new SimpleRiskModel();
35      mockedEM = mock(IEvidenceManager.class);
36      mockedUser = mock(ITMSUser.class);
37      evidence = new ArrayList<EvidenceModel>();
```

```
38      when(mockedEM.getEvidence(mockedUser, DataType.DATA)).
            thenReturn(evidence);
39    }
40
41
42    @Test
43    public void calculateRiskTest(){
44      CreditEvidenceModel e1 = mock(CreditEvidenceModel.
            class);
45      e1.dataValue = 1;
46      e1.date = new DateTime(new Date()).minusDays(30);
47      e1.user = mockedUser;
48      e1.paymentDate = new DateTime(new Date()).minusDays
            (25);
49      e1.extendedCredit = 0;
50      evidence.add(e1);
51
52      assertEquals(1f, RM.calculateRisk(mockedUser, mockedEM
            ), 0.01);
53
54      CreditEvidenceModel e2 = mock(CreditEvidenceModel.
            class);
55      e2.dataValue = 1;
56      e2.date = new DateTime(new Date()).minusDays(31);
57      e2.user = mockedUser;
58      e2.extendedCredit = 0;
59      evidence.add(e2);
60
61      assertEquals(0.25f, RM.calculateRisk(mockedUser,
            mockedEM), 0.01);
62    }
63
64
65    @Test
66    public void calculateRiskUnpaidWithinCreditTest(){
67      CreditEvidenceModel e1 = mock(CreditEvidenceModel.
            class);
68      e1.dataValue = 1;
69      e1.date = new DateTime(new Date()).minusDays(10);
70      e1.user = mockedUser;
71      e1.extendedCredit = 0;
72      evidence.add(e1);
73
74      assertEquals(0.75f, RM.calculateRisk(mockedUser,
            mockedEM), 0.01);
75    }
76
77
78    @Test
79    public void calculateRiskPaidAfterCreditTest(){
```

```
80      CreditEvidenceModel e1 = mock(CreditEvidenceModel.
            class);
81      e1.dataValue = 1;
82      e1.date = new DateTime(new Date()).minusDays(32);
83      e1.paymentDate = new DateTime(new Date());
84      e1.user = mockedUser;
85      e1.extendedCredit = 0;
86      evidence.add(e1);
87
88      assertEquals(0.5f, RM.calculateRisk(mockedUser,
            mockedEM), 0.01);
89    }
90
91
92  }
```

B.7 SimpleTrustModelTest.java

```
1   package implementationExamples;
2
3   import static org.junit.Assert.assertEquals;
4   import static org.mockito.Mockito.mock;
5   import static org.mockito.Mockito.verify;
6   import static org.mockito.Mockito.when;
7
8   import java.util.ArrayList;
9
10  import org.junit.Before;
11  import org.junit.Test;
12  import org.junit.runner.RunWith;
13  import org.mockito.junit.MockitoJUnitRunner;
14
15  import enums.DataType;
16  import implementationExample.SimpleTrustModel;
17  import interfaces.IEvidenceManager;
18  import interfaces.ITMSUser;
19  import models.EvidenceModel;
20
21  @RunWith(MockitoJUnitRunner.class)
22  public class SimpleTrustModelTest {
23
24    public IEvidenceManager mockedEM;
25    public ITMSUser mockedUser;
26    public SimpleTrustModel TM;
27
28
29    /*
30     *
```

```
31      * Setup being run before each test
32      *
33      */
34     @Before
35     public void setup(){
36       mockedEM = mock(IEvidenceManager.class);
37       mockedUser = mock(ITMSUser.class);
38       TM = new SimpleTrustModel();
39       ArrayList<EvidenceModel> evidence = new ArrayList<
             EvidenceModel>();
40       EvidenceModel e = mock(EvidenceModel.class);
41
42       e.user = mock(ITMSUser.class);
43       e.dataValue = 0;
44       e.type = DataType.TRUSTLEVEL;
45
46
47       evidence.add(e);
48       when(mockedEM.getEvidence(mockedUser, DataType.
             TRUSTLEVEL)).thenReturn(evidence);
49     }
50
51
52     /*
53      *
54      *  updateTrust tests
55      *
56      */
57     @Test
58     public void updateTrustTest(){
59       TM.updateTrust(mockedUser, mockedEM, 3);
60       verify(mockedEM).updateTrustLevel(mockedUser, 1);
61     }
62
63     @Test
64     public void updateTrustWithTooHighValueTest(){
65       TM.updateTrust(mockedUser, mockedEM, 30);
66       verify(mockedEM).updateTrustLevel(mockedUser, 1);
67     }
68
69
70     /*
71      *
72      * calculateTrust tests
73      *
74      */
75     @Test
76     public void calculateTrustWithNoRecommendationsTest(){
77       ArrayList<EvidenceModel> noRecommendations = new
             ArrayList<EvidenceModel>();
```

```
78      when(mockedEM.getEvidence(mockedUser,DataType.
           RECOMMENDATION)).thenReturn(noRecommendations);
79      when(mockedEM.getTrustLevel(mockedUser)).thenReturn(1f
           );
80
81      assertEquals(1, TM.calculateTrust(mockedUser, mockedEM
           ),0.01);
82
83    }
84
85
86    @Test
87    public void
         calculateTrustWithOneRecommendationFromHighTrustRecommendorTest
         (){
88      ArrayList<EvidenceModel> recommendations = new
           ArrayList<EvidenceModel>();
89      ITMSUser mockedRecommendor = mock(ITMSUser.class);
90      EvidenceModel r = new EvidenceModel();
91      r.dataValue = 1f;
92      r.recommendor = mockedRecommendor;
93      r.user = mockedUser;
94      r.type = DataType.RECOMMENDATION;
95      when(mockedEM.getEvidence(mockedUser,DataType.
           RECOMMENDATION)).thenReturn(recommendations);
96      when(mockedEM.getTrustLevel(mockedUser)).thenReturn(1f
           );
97      assertEquals(1, TM.calculateTrust(mockedUser, mockedEM
           ), 0.01);
98    }
99
100
101   @Test
102   public void
         calculateTrustWithOneRecommendationLowTrustRecommendorTest
         (){
103     ArrayList<EvidenceModel> recommendations = new
           ArrayList<EvidenceModel>();
104     ITMSUser mockedRecommendor = mock(ITMSUser.class);
105     EvidenceModel r = mock(EvidenceModel.class);
106     r.dataValue = 1f;
107     r.recommendor = mockedRecommendor;
108     r.user = mockedUser;
109     r.type = DataType.RECOMMENDATION;
110     recommendations.add(r);
111
112     when(mockedEM.getEvidence(mockedUser,DataType.
           RECOMMENDATION)).thenReturn(recommendations);
113     when(mockedEM.getTrustLevel(mockedUser)).thenReturn(1f
           );
```

```
114     when(mockedEM.getTrustLevel(mockedRecommendor)).
            thenReturn(0f);
115     assertEquals(0.5f, TM.calculateTrust(mockedUser,
            mockedEM), 0.01);
116   }
117
118
119   @Test
120   public void
          calculateTrustWithOneRecommendationMediumTrustRecommendorTest
          (){
121     ArrayList<EvidenceModel> recommendations = new
            ArrayList<EvidenceModel>();
122     ITMSUser mockedRecommendor = mock(ITMSUser.class);
123     EvidenceModel r = mock(EvidenceModel.class);
124     r.dataValue = 1f;
125     r.recommendor = mockedRecommendor;
126     r.user = mockedUser;
127     r.type = DataType.RECOMMENDATION;
128     recommendations.add(r);
129
130     when(mockedEM.getEvidence(mockedUser,DataType.
            RECOMMENDATION)).thenReturn(recommendations);
131     when(mockedEM.getTrustLevel(mockedUser)).thenReturn(1f
            );
132     when(mockedEM.getTrustLevel(mockedRecommendor)).
            thenReturn(0.5f);
133     assertEquals(0.75, TM.calculateTrust(mockedUser,
            mockedEM), 0.01);
134   }
135
136
137   /*
138    *
139    * addRecommendation Tests
140    *
141    */
142   @Test
143   public void
          addRecommendationWithAPriorRecommendationTest(){
144     ITMSUser mockedRecommendor = mock(ITMSUser.class);
145     EvidenceModel r = mock(EvidenceModel.class);
146     r.dataValue = 1f;
147     r.recommendor = mockedRecommendor;
148     r.user = mockedUser;
149     r.type = DataType.RECOMMENDATION;
150     when(mockedEM.getRecommendation(mockedRecommendor,
            mockedUser)).thenReturn(r);
151
```

```
152     TM.addRecommendation(mockedRecommendor, mockedUser, 1,
            mockedEM);
153     assertEquals(r.dataValue, 1,0.01);
154   }
155 }
```