Förbättring av interoperabilitet med intelligenta utrymmen och ontologier

Jon von Weymarn
matr. 29087

Kandidatarbete
Handledare: André Kaustell
Faculty of Technology
Åbo Akademi University

Referat

Intelligenta apparater blir allt vanligare. Idag kan dessa apparaters fulla potential inte utnyttjas på grund av att mjukvara för detta inte ännu existerar. Detta arbete presenterar lösningar för att underlätta situationen genom att introducera intelligenta utrymmen. Dessa intelligenta utrymmen har en hub-apparat som styr informationsflödet samt innehar en gemensam databas. Trippler i RDF och ontologier introduceras för att representera och strukturera data.

Nyckelord: Interoperabilitet, Applikationsutveckling, Intelligenta utrymmen, Smartspace, Distribuerade beräkningar, Aktör-modellen, Ontologi, Resonering.

Innehåll

Kapitel 1

Inledning

Vår omgivning fylls idag mer och mer av olika typer av apparater. På samma gång som apparaternas antal ökar, ökar också deras beräkningskapacitet och därmed deras potential för vad de kan användas till. Till exempel, i hemmet har vi redan ett flertal effektiva apparater som bara har som uppgift att står för underhållningen i vardagsrummet. På arbetet har idag nästan alla en egen bärbar dator eller en intelligent telefon. Detta faktum, att vi är omringade av apparater och ett överflödigt utbud av data, har lett till problem för systemutvecklare. Problemen ligger i att utvecklingen i dessa apparater går så snabbt, samt att nya typer av apparater sätts på marknaden ständigt, vilket leder till att systemutvecklare inte hinner utveckla mjukvara som kunde utnyttja apparaternas fulla potential.

Ideala situationen vore att alla apparater sammanverkade med varandra samt att användare lätt kunde styra och utnyttja apparaterna i sin omgivning. Idag är detta inte möjligt på grund av ett antal orsaker. För det första, finns det ingen standard på hur apparater sammankopplas med andra apparater. För det andra, saknas det också en standard för hur data bland apparater skulle representeras. Dessutom är användarvänlighet ett område som det borde ske utveckling på för att närma sig den ideala situationen.

Denna kandidatavhandling adresserar de ovan nämnda problemen samt förslag till hur man kunde förbättra situationen ges där existerande teknologier utnyttjas. För att illustrera tekniker och lösningar kommer typiska användningsområden som existerar i hemmet att presenteras. Detta betyder dock inte att förlagen i detta arbete begränsar sig till hemmet utan de kan även användas som sådana i många andra områden.

Kapitel 2

Arkitektur för interoperabilitet

Tillvägagångssättet som presenteras i detta arbete baserar sig på en gammal idé. Gul A. Agha presenterade år 1985 en artikel som beskrev hur man kunde dela upp distribuerade datorberäkningar med att representera apparater som så kallade aktörer. Dessa simpla aktörers uppgift var att endast utföra de beräkningar de blivit allokerade och sedan rapportera resultat. På detta vis kunde flere apparaters kapacitet utnyttjas för att snabbare få resultat på problem som på den tiden ansågs som beräkningsmässigt tunga.

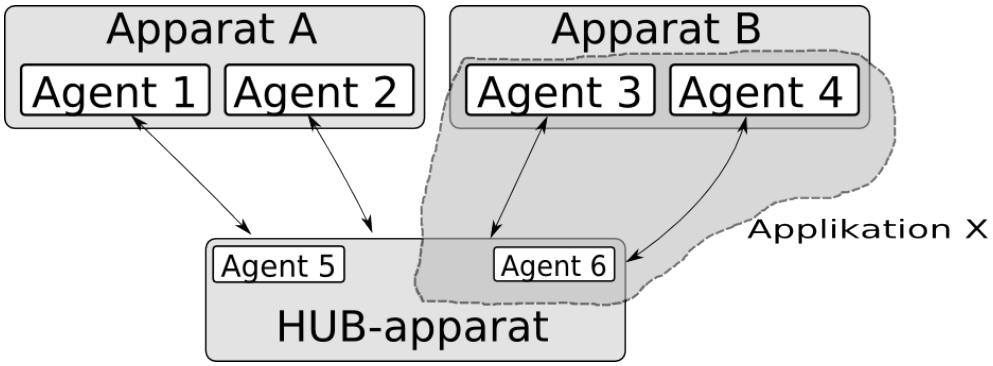
2.1 Agenter och applikationer

Lösningen i detta arbete utvidgar på Aghas idé på följande vis. Apparaters funktionalitet delas upp i agenter. En agent kan ses som ett litet program som kör på apparatens hårdvara och som har som uppgift att utföra en, och endast en, uppgift. Denna uppgift kan vara så simpel som att rapportera sensorvärde till mer komplexa uträckningar eller operationen. Till exempel en bastuugn skulle bestå av följande agenter:

- en agent för att styra om ugnen är på eller avstängd.
- en agent för att styra effekten i ugnen.

Dessa kunde sedan kombineras till en applikation som användaren kunde utnyttja för att styra ugnen. Grundtanken, med agenterna, är att om en apparat har många funktioner delas dessa upp i flere små agenter.

Applikationen skulle alltså byggas upp genom att kombinera olika agenter som utför skilda funktioner och på så vis få en önskad funktionalitet som summan



Figur 2.1: Ett intelligent utrymme med en hub-apparat som är i kontakt med två andra apparater genom agenter.

av agenterna. Exemplet ovan, med bastuugnen, är som sådant inte speciellt komplicerat men då fallet utvidgas med att tillåta andra apparater att medverka blir applikationsmöjligheterna betydligt mer avancerade. Ett exempel är att utvidga fallet med en apparat som har en kalender (till exempel en mobiltelefon). Då kan man kombinera agenterna för att skapa en applikation som på basen av informationen i en kalender styr bastun. Vidare kunde applikationen utvidgas med en temperatursensor i bastun som skulle möjliggöra temperaturstyrning till ett en viss definierad referenstemperatur. Detta exempel undersöks i noggrannare detalj senare i detta kapitel.

2.2 Intelligenta utrymmen

För att möjliggöra att apparater, som i exemplet ovan, lättare kunde hitta varandra samt fungera sinsemellan görs en utvidgning av Aghas metod genom att introducera så kallade intelligenta omgivningar (Eng. smartspace). En intelligent omgivning är en virtuell representation beskrivande apparater som existerar i ett fysiskt gemensamt utrymme. Grundidén är att all information från apparaterna är tillgänglig för alla apparater inom utrymmet, och att applikationer kunde existera inom utrymmet som kan göra intelligenta beslut baserande på utrymmets apparater. I praktiken implementeras detta genom att introducera en hub-apparat som fungerar som en gemensam databas och informationsförmedlare. Att använda en arkitektur med en förmedlande apparat baserar sig på *Common Object Request Broker Architecture* (CORBA) standarden.

I figur 2.1 illustreras intelligenta utrymmens arkitektur. Som tidigare nämnts delas apparaters funktionalitet upp i agenter. Dessa agenter är begränsade till att antingen skriva till, eller söka information från, den genensamma hub-

apparaten. Hur en agent får information från hub-apparaten kan göras på två olika vis, nämligen som en vanlig förfrågan eller som en prenumeration:

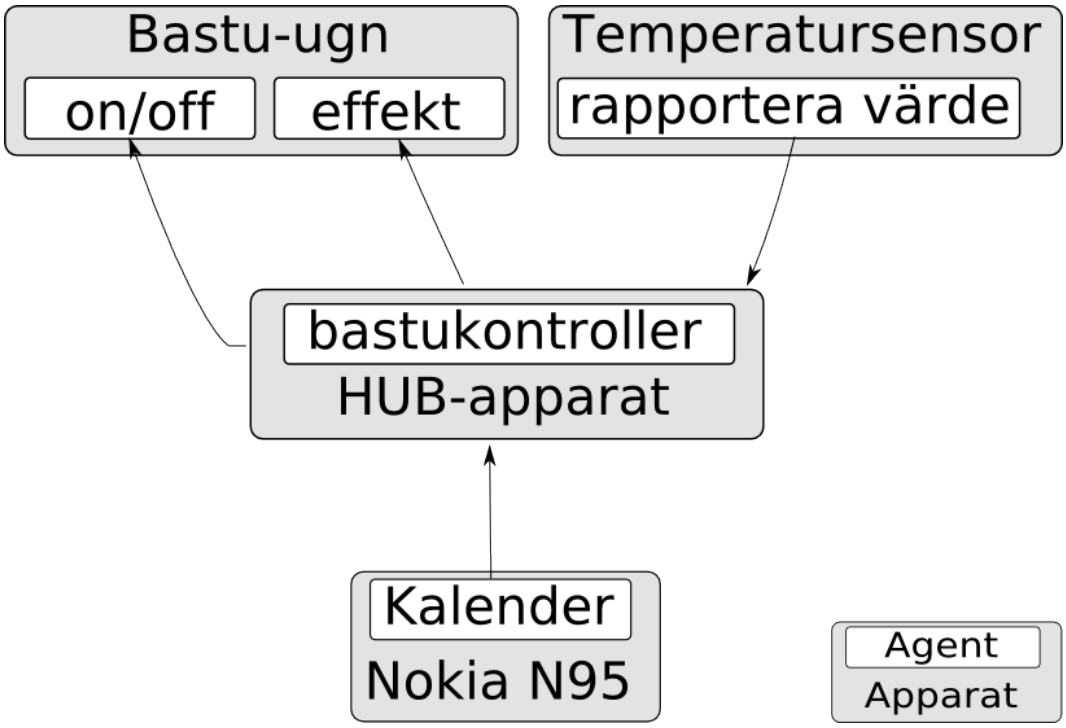
- I en vanlig *förfrågan* frågar agenten om data som den är intresserad av och hub-apparaten svarar med denna information.
- Då fallet är en *prenumeration* meddelar agenten hub-apparaten att den är intresserad av att följa med viss information i databasen. Om denna information ändrar meddelar hub-apparaten agenten om detta och skickar den nya informationen till agenten.

Genom att sedan kombinera ett antal agenter kan en applikation skapas som har önskad funktionalitet. Denna arkitektur möjliggör en abstraktion av den traditionella synen på en applikation. Vanligen ses en applikation som något som kör på en specifik apparat som möjligen söker data från andra apparater. Då denna arkitektur används är det inte längre så relevant var applikationen rent fysiskt kör. I figur 2.1 har agenterna 3,4 och 6 kombinerats för att skapa applikation X.

Genom att använda denna arkitektur med en central förmedlare elimineras en mängd problem som skulle uppstå vid sammankoppling av olika apparater. För att två apparater skall kunna sammanverka traditionellt måste de först hitta varandra och sedan använda ett gemensamt protokoll för att öppna en kanal för dataöverföring. Då hub-apparaten används behöver apparaterna inte oroa sig med dessa faktorer utan det räcker med att enstaka agenter lyckas skriva till, eller läsa från, den centrala hub-apparaten. Dessutom möjliggör arkitekturen att många olika typer av apparater kan användas tillsammans. Till exempel som i fallet med bastun där en mobiltelefon, med minne och beräkningskapacitet, medverkade i ett system tillsammans med en primitiv sensor som endast kan rapportera sitt eget värde.

En annan stor fördel kommer ifrån användnigen av agentuppdelningen. Då alla funktioner är uppdelade i egna separata agenter ökar modulariteten märkbart. För att skapa en ny applikation, med likande funktionalitet som en tidigare applikation, kan ofta gamla agenter användas och därmed minskar behovet att skapa ny kod.

Lösningar för att adressera interoperabilitet mellan små apparater existerar idag men ingen av dessa har blivit till en standard. Detta beror på att de lösningar som används idag ofta är domänspecifika och endast fungerar för ett begränsat antal apparater. Ett annat problem idag är att de lösningar som används inte lätt går att utvidga med nya apparater. Därför är det dyrt att med dagens lösningar bygga upp ett system där många apparater sammanverkar.



Figur 2.2: Intelligent utrymme för bastukontollerapplikation.

2.2.1 Praktiskt exempel med bastukontroller

För att illustrera och summera de ovan diskuterade koncepten behandlas bastuexemplet här i närmare detalj. Figur 2.2 visar en överblick av intelligenta utrymmet samt vilka apparater och agenter som krävs för att fallet med bastustyrningen är möjligt.

Totalt består intelligenta utrymmet av 4 apparater:

- Bastu-ugn.
- Temperatursensor.
- Mobiltelefon med kalender (Nokia N95 i detta fall).
- HUB-apparaten med gemensam databas som styr och förmedlar data.

Agenterna som krävs för att en bastukontrollerapplikation är möjlig är:

- En agent för att skriva sensorns värde till hub-apparatens gemensamma databas.
- En agent för att skriva mobiltelefonens kalenderdata till hub-apparatens gemensamma databas.

- En agent för att läsa från hub-apparatens gemensamma databas om bastuugnen skall vara på eller av.
- En agent för att läsa från hub-apparatens gemensamma databas hur hög effekt bastuugnen skall vara inställd på.
- En agent inom hub-apparaten som styr bastuugnen.

Den sista agenten, som kör inom hub-apparaten, är i detta fall den som bestämmer om bastun är på eller inte. Detta gör den genom att läsa kalenderinformationen från databasen och söka efter en anmärkning där användaren sparat en önskad temperatur som en kalenderhändelse. Sedan då en sådan kalenderhändelse är aktuell kan den be temperatursensorn uppdatera sitt sensorvärde. Detta kan till exempel implementeras så att sensoragenten prenumererar på en bit data som kontrolleragenten ändrar på då den önskar uppdatering i sensorvärdet. Sedan om detta sensorvärde är under användarens önskade värde justerar kontrolleragenten bastuns av/på värde samt dess effektvärde i databasen. De motsvarande agenterna för denna data inom bastuugnen prenumererar på denna information och får, då informationen ändrat, ett meddelande om detta varefter de kan sätta ugnen i läget som sparats i databasen. Efter ett bestämt tidsintervall ber kontrolleragenten igen om ett nytt sensorvärde för att se om bastuns verkliga temperatur närmar sig det önskade referensvärdet. Enligt behov ändrar logiken inom kontrolleragenten av- och påvärdet samt effektvärdet i databasen för att bastun skall nå den önskade temperaturen.

Arkitekturen i detta arbete har ett antal fördelar varav en av de största är hur denna abstraktion underlättar systemutvecklarens arbete. Dock skapar den även nya utmaningar men dessa kommer inte att behandlas i detta arbete. Nästa kapitel behandlar hur data inom ett intelligent utrymme kan representeras med hjälp av ett antal teknologier för att uppnå en gemensam men dynamisk representation av informationen i intelligenta utrymmet som dessutom är lätt att förstå.

Kapitel 3

Datarepresetation

Datorer talar ett språk av ettor och nollor medan vi människor använder ord och meningar för att kommunicera. För att möjliggöra att olika typer av apparater, från olika tillverkare, med olika bakgrund, skall kunna sammanverka behövs det en standard som är lätt att förstå. *Resource Description Framework* (RDF) är en standard som kan användas för detta problem.

3.1 Uppdelning av data i trippler

RDF är en av de grundläggande teknikerna inom den semantiska webben. Standarden skapades ursprungligen för att uttrycka data om data dvs. metadata, men har idag utvecklas till en generell metod för att uttrycka all typ av information. Tanken bakom RDF är att skapa meningar, eller påståenden, som människan lätt kan förstå. Detta görs med hjälp av tripplar i formen *subjekt - predikat - objekt*.

Subjektet i en trippel fastställer resursen som trippeln besktiver (till exempel personen "*Kim*"). *Predikatet* beskriver en egenskap eller relation av subjektet som vi beskriver (till exempel "*är far till*"). Sist beskriver *objektet* värdet för relationen (t.ex. "*Erik*"). Med denna trippel har vi alltså beskrivit en enkel familjerelation mellan två personer (*Kim - är far till - Erik*). Objektet i en trippel kan också vara ett numeriskt värde som ses i exemplet i figur 3.1. Här beskriver en trippel värdet från en temperatursensor.

Figur 3.1: Trippel som beskriver temperaturvärdet från en sensor.

Fördelen i att använda RDF för att beskriva data är att både en människa och en dator kan förstå informationen. Dessutom är det ett steg mot ett mer naturlig språk för kommunikation mellan människa och maskin. Hur denna data sedan i praktiken sparas förklaras i nästa sektion.

3.2 XML

XML står för *eXtensible Markup Language* vilket är ett universellt märkspråk utvecklat för semantiska webben. Tekniken kan användas för att beskriva universell data som innehåller en viss struktur. I praktiken görs detta genom att dela upp bitar av information i så kallade element. Ett element i XML består av en start- och en sluttagg. Emellan dessa tagg finns data som elementet skall behandla samt möjligen attribut för att närmare definiera informationen. Taggarna är inramade i vinkelparenteser (<, >) och sluttaggen innehåller ett snedstreck (/).

Två exempeltripplar kan ses i listning 3.1.

Listing 3.1: Tripplar i XML.

```
1  <rdf:Exempel1>
2      <Subjekt>Temp_Sensor_001</Subjekt>
3      <Predikat:grape>harTemperatur</Predikat>
4      <Objekt>50</Objekt>
5  </rdf:Exempel1>
6
7  <rdf:Exempel2>
8      <Subjekt>Kim</Subjekt>
9      <Predikat:grape>ärFarTill</Predikat>
10     <Objekt>Erik</Objekt>
11 </rdf:Exempel2>
```

Kapitel 4

Ontologi

Att använda RDF standarden för att representera data kan ses som ett steg mot ett mer naturlig språk i den mån att information i trippelform är lättare att förstå. Dock genom att endast använda RDF finns det ingen struktur eller några regler för vad som kan beskrivas. Till exempel är det fullt möjligt att i RDF kombinera de ovan nämnda trippelexemplen för att skapa en trippel *Kim - är far till - 50*. I vissa applikationer kan det behövas att man kan göra sådana påståenden men oftast vill man begränsa data så att den endast beskriver möjlig logisk information. För att uppnå detta kan RDF kombineras med så kallade ontologier.

4.1 Vad är en ontologi?

En ontologi är ett sätt att begränsa och strukturera information inom en viss domän. Detta görs genom ett antal koncept som tillsammans skapar en modell av hurdana objekt och egenskaper som existerar inom domänen. I praktiken görs detta med hjälp av ett ontologispråk och i detta arbete används *Web Ontology Language* (OWL) ontologispråket. Att använda OWL i detta arbete är ett naturligt val på grund av att det anses vara en av de fundamentala teknologierna för semantiska webben och dessutom är det skapat för att bygga på RDF standarden. OWL är godkänt av *World Wide Web Consortium* (W3C) som är en internationell organisation för standarder inom webbteknologi.

4.1.1 Byggstenarna av en ontologi

För att beskriva närmare vad en ontologi är, och vad man kan uppnå med en, behandlas dess byggstenar nedan. Dessa är *klasser, individer, relationer,*

egenskaper och *restriktioner*.

Klass

Det viktigaste konceptet inom ontologier är klasser. En klass kan beskrivas som en abstrakt mekanism för att gruppera resurser som har liknande egenskaper. Som ett exempel på en klass kan klassen *Bil* nämnas vilken kan användas för att beskriva olika bilar. För att utvidga på detta kan klasser delas upp i en hierarki med klasser ovan, och under den klassen som beskrivs. Subklasser är klasser som placeras under i hierarkin som beskriver objektet med mer detalj. Exemplet ovan med klassen *Bil* kan utvidgas med subklasserna *Sedan*, *Cabriolet* och *Hatchback*. Superklasser däremot är klasser som finns ovanför och beskriver objektet mer abstrakt. För klassen *Bil* kunde ett exempel vara klassen *Fordon*. Klassnamn börjar alltid med stor bokstav.

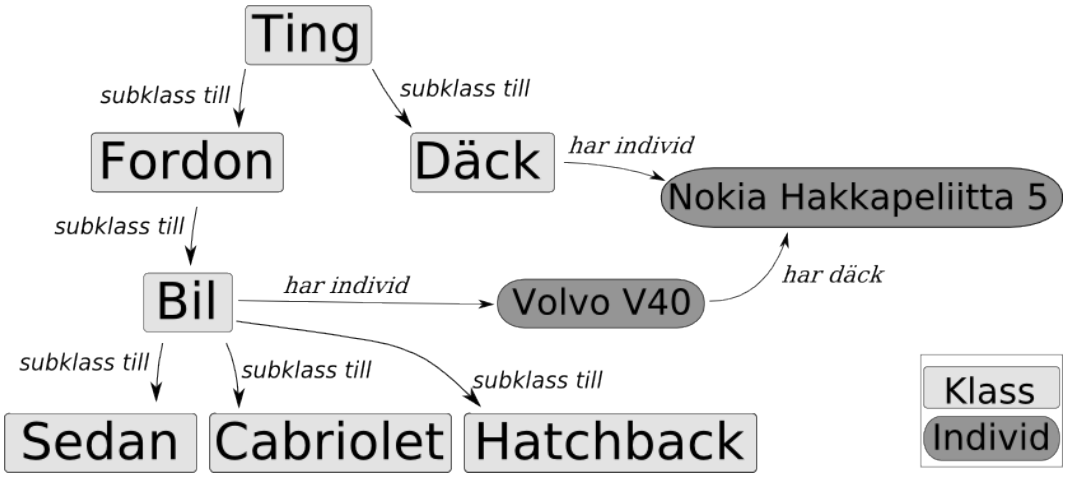
Ett av kraven på en ontologi är att klasshierarkin alltid börjar från samma klass. Denna klass är specialklassen *Ting* vilken är superklass till allt inom ontologin. Detta behövs för att man skall kunna återvända till en och samma rot från alla delar av ontologin.

Individer

Individer (eller objekt) kallas de unika medlemmarna till någon specifik klass. Klasser kan ha ett definierat antal individer som hör till den och som kan användas för att definiera relationer (beskrivs nedan) mellan flera olika klasser. Ett exempel på en individ till klassen *Bil* är *Volvo V40*. I praktiken skapar detta en trippel *Bil - har individen - Volvo V40*.

Relationer

För att beskriva relationer definieras en ny klass som kallas *Däck*. Denna klass har bland annat en individ *Nokia Hakkapeliitta 5*. Vi kan då med hjälp av relationer länka denna individ tillsammans med individen *Volvo V40*. Detta görs i praktiken med en trippel som till exempel *Volvo V40 - har däck - Nokia Hakkapeliitta 5*. Ovan beskrevs sub- och superklasser som också är exempel på relationer. Dessa ger upphov till tripplar som *Bil - är subklass till - Fordon*.



Figur 4.1: Ett exempel på en ontologi beskrivande bilar och däck.

Egenskaper

Egenskaper ger oss mer information om individen som vi vill beskriva. Dessa är alltid det mellersta elementet (predikatet) i en trippel. Egenskaper delas upp i två typer, nämligen *datatypsegenskaper* och *objektegenskaper*, beroende på vad de beskriver. Relationerna ovan (*är subklass till, har individen, har däck* och *är far till* i sektion 3.1) är alla objektegenskaper. Dessa är alltså egenskaper som linkar två individer till varandra med någon gemensam faktor. Datatypsegenskaper däremot används för att länka individer till värden tillhörande någon datatyp. I figur 3.1 ses ett exempel där egenskapen *har temperatur* länkas till ett siffervärde (*50*).

Då man ser noggrannare på de egenskaper som används hittills i detta arbete ser man att de har en riktnig, dvs. att de endast går att använda åt ett håll. Därför introduceras *invers egenskapen* som måste existera för varje egenskap. Invers egenskapen beskriver egenskapen så att den kan användas åt andra hållet. Till exempel *är far till* egenskapen har invers egenskapen *är barn till*. Då kan trippeln som introducerades i sektion 3.1 skrivas om med invers egenskapen som *Erik - är barn till - Kim*.

I figur 4.1 finns de hittills beskrivna delarna illustrerade. Samma information kan beskrivas med hjälp av trippler så att varje pil är predikatet, rutan före subjektet och rutan efter objektet. Invers relationer som *superklass till* har lämnats bort från figuren.

Figur 4.2: Domän och räckvidd för egenskapen *har däck*

Restriktioner

Den sista komponenten är även den komponenten som ger ontologierna en stor del av dess styrka. Med hjälp av dessa kan man definiera restriktioner som gör att tidigare trippeln, *Kim - är far till - 50*, inte godkänns. Det finns många olika typer av restriktioner men i detta arbete kommer endast de viktigaste att presenteras. Dessa är *domän*, *räckvidd* och *kardinalitet*.

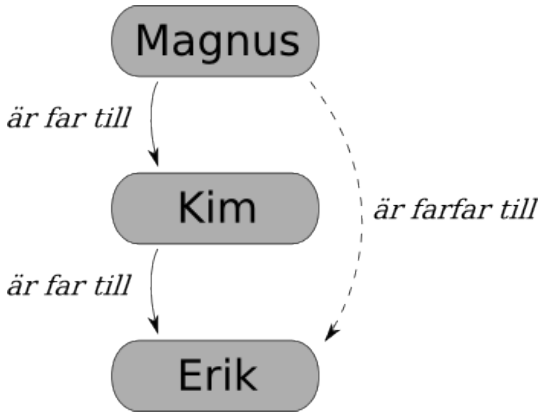
Domänen är en restriktion som kan sättas på en egenskap. Denna restriktion begränsar vad en egenskap beskriver. Tidigare definierade vi trippeln *Volvo V40 - har däck - Nokia Hakkapeliitta 5*. Denna trippel kunde ge upphov till domänrestriktionen för *har däck* att endast beskriva individer av klassen *Bil*. Det vill säga att egenskapen *har däck* har klassen *Bil* som domän. Detta illustreras i figur 4.2.

Räckvidden för en egenskap är en motsvarande restriktion som domänen men den begränsar, åt andra hållet, till hurdana individer en egenskap kan länkas. För det tidigare exemplet betyder detta att *har däck* egenskapen länkar klassen *Bil* till *Däck*, det vill säga räckvidden för *har däck* är klassen *Däck*. Detta illustreras i figur 4.2.

Kardinalitet är en begränsning som säger hur många instanser något kan länkas till. Det är vanligt att man vill försäkra sig om att någon information finns. Till exempel måste varje person ha ett socialskyddssignum. I detta fall är kardinaliteten *ett*. En person måste också ha ett förnamn men det är möjligt att ha flere. I detta fall är kardinaliteten *minst ett*. Tidigare beskrevs bilar och dess tillhörande däck. I detta fall är det naturligt att bestämma en restriktion att egenskapen *har däck* skall ha kardinalitet *fyra*. Detta resulterar i att varje instans av klassen *Bil* måste länkas till fyra instanser av klassen *Däck* med egenskapen *har däck*.

4.2 Resonering

Förutom att man med en ontologis regler kan strukturera och begränsa informationen som den innehåller finns det även andra fördelar i att använda dem. En av dessa fördelar är att man kan koppla ihop en agent som utför resonering

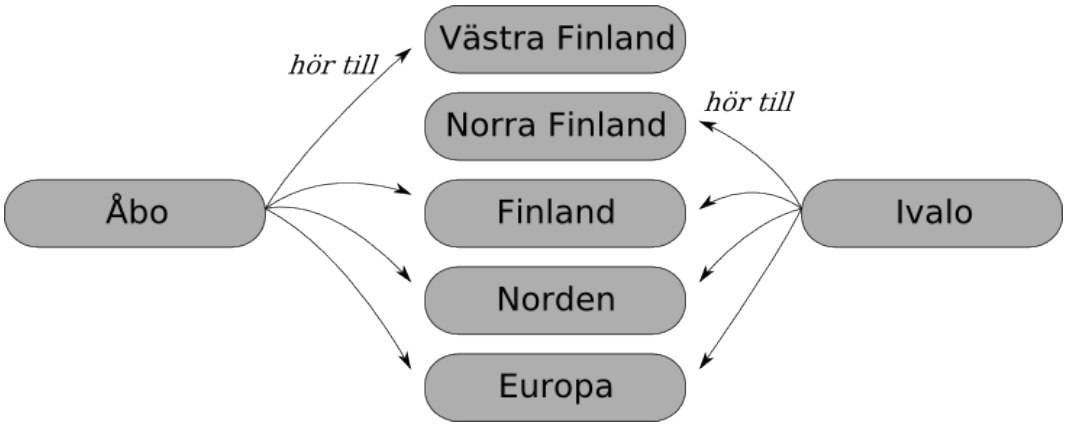


Figur 4.3: Exempel på resonering. Om en person är länkad till en annan person via två *är far till* egenskaper är dessa även länkade direkt med *är farfar till* egenskapen. Detta är information som en resonerare kan hitta och lägga till i databasen.

på informationen. Resonering betyder att en agent med artificiell intelligens behandlar informationen eller övervakar att man inte bryter mot ontologins regler. Dessutom kan en resonerare, i en bra ontologi, utvidga informationen med nya slutsatser som den dragit på basen av ontologins uppbyggnad.

För att illustrera resonering utvidgas exemplet med trippeln *Kim - är far till - Erik* med en ny trippel *Magnus - är far till - Kim*. Om man i ontologins regler definierar den speciella relationen som uppstår om två personer länkas till varandra med två *är far till* steg kan resoneraren hitta detta och lägga till den nya trippeln *Magnus - är farfar till - Erik*. Detta illustreras i figur 4.3 där den streckade linjen är information som resoneraren har funnit och lagt till.

Ett annat exempel som har större praktisk betydelse är då man utför sökningar på informationen inom ontologin. Om Kim och Erik var instanser i en större ontologi som innehöll information om deras hemstad kunde en resonerare möjliggöra ett antal sökningar som annars inte lyckades. Vi antar att Kim bor i Åbo och Erik i Ivalo. Ontologin i fråga innehåller geografisk information och vet vilka städer som hör till Finland. Då kunde en sökning göras på personer i Finlandsom gav både Kim och Erik som reslutat fastän de inte definierat Finland i informationen om dem. Detta är möjligt för att ontologin vet att till exempel Åbo hör till Finland, Västra Finland, Norden och Europa. Detta illustreras i figur 4.4.



Figur 4.4: Exempel på resonering. Ontologin innehåller information om två städer och vart de tillhör.

Kapitel 5

Slutsatser

I denna kandidatavhandling har ett antal tekniker presenterats som tillsammans har potentialen att möjliggöra interaktion mellan apparater på ett nytt sätt. Uppdelningen av apparaters funktionalitet i agenter resulterar i en modularitet som kunde underlätta systemutvecklares arbete då olika typer av applikationer skall skapas. Att introducera en hub-apparat med en central gemensam databas betyder att apparaterna inte behöver kunna sammankoppla sig till varandra utan det räcker att de kan nå hub-apparaten. Dessutom betyder denna uppdelning även att alla apparaters närvaro inte är kritisk för att en applikationen skall kunna köra. Om en applikations kontakt med hub-apparaten bryts för en kortare tid behöver applikationen inte ens bli medveten om detta. Förutom detta faktum enablerar arkitekturen också att apparater utan nån som hellst beräkningskapacitet kan medverka direkt i en applikation.

Exemplet med bastukontrollern är ett typiskt exempel på vad denna arkitektur kunde användas till i praktiken men det beskriver endast en liten del av arkitekturens potential. Detta exempel kunde till exempel utvidgas med en agent som hade kontakt med webben för att skapa en applikation där bastun styrs över internet.

Tripplerna i formen av korta påståenden gör att informationen som en dator behandlar blir lättläst för en människa, speciellt om den presenteras grafisk. Fördelen som kommer från detta är att det blir lättare för en systemutvecklare att arbeta, samt att processen för en mindre teknisk person att lära sig systemet blir kortare. Idag då allt mer av apparaters applikationer skapas av användarna är detta väldigt viktigt. För att användaren lätt skall kunna skapa bra applikationer måste de förses med bra verktyg men det är också viktigt att informationen som apparaten behandlar är lätt att förstå.

Ontologier kan också hjälpa i denna bemärkelse. Om stukturen och reglerna för

informationen är bra definierad undelättas systemutvecklarens arbete genom att det blir lättare att undvika misstag. Dessutom möjliggör ontologier att man lätt kan sammankoppla olika typer av databaser genom att beskriva vad de olika databaserna har gemensamt. På samma vis, genom att definiera likheter, kan databaser i olika språk sammankopplas. Resoneringen däremot möjliggör att man genom att utvidga ontologin kan resonera till ny information för att utvidga databasen.

Som slutsats kan det konstateras att tekniker och arkitekturen som beskrivits i detta arbete har en stor potential att skapa en värld där vi kan nå information och styra alla apparater omkring oss. För att detta skall ske behövs dock en hel del utveckling inom området. Bland annat måste utvecklig ske inom hur informationens säkerhet i ett intelligent utrymme garanteras. Som arkitekturen beskrivits i detta arbete kan alla apparater nå alla apparaters data vilket i praktiken endast är acceptabelt i väldigt små och begränsade användningsområden. Dessutom för att exempel som beskrivits i detta arbete skall vara möjliga måste apparaters tillverkare öppna sina apparater för att möjliggöra att agenter kan utvecklas för dem. Med andra ord fungerar lösningarna i detta arbete bra i teorin, men före de kan användas i praktiken måste mycket utveckling ske.

Litteraturförteckning

[1] O. Lassila, "Enabling semantic web programming by integrating rdf and common lisp," in *The First Semantic Web Working Symposium*. Citeseer, 2001, pp. 403–410.

[2] I. Oliver and J. Honkola, "Personal semantic web through a space based computing environment," *arXiv*, vol. cs.NI, Aug 2008. [Online]. Available: http://arxiv.org/abs/0808.1455v1

[3] F. van Harmelen and D. L. McGuinness, "OWL web ontology language overview," *W3C recommendation*, Feb. 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[4] D. L. McGuinness, C. Welty, and M. K. Smith, "OWL web ontology language guide," *W3C recommendation*, Feb. 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[5] P. F. Patel-Schneider, I. Horrocks, and P. Hayes, "OWL web ontology language semantics and abstract syntax," W3C, W3C Recommendation, Feb. 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[6] J. J. Carroll and G. Klyne, "Resource description framework (RDF): Concepts and abstract syntax," *W3C recommendation*, Feb. 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[7] N. Noy and D. McGuinness, "Ontology development 101: A guide to creating your first ontology," *Knowledge Systems Laboratory*, pp. 01–05, 2001.

[8] G. Agha, "Actors: A model of concurrent computation in distributed systems," vol. AITR-844, Oct 1985.

[9] H. Baker, "Actor systems for real-time computation," vol. MIT-LCS-TR-197, Jan 1978.