# Hemtenta

Multimediadatabaser, tnm053, VT2004

David Kästel, davka237@student.liu.se

1. a) Man väljer någon form av spatiellt index. Lämpligen R-träd, då det kan innehålla multidimensionella data, i vårt fall koordinater och folkmängd. Trädrepresentationen i sin tur består av objekt med par av R, minsta möjliga omgivande rektangel och O, en pekare till objektet. Våra data sparas alltså i lövnoderna, vilka alla ligger på samma avstånd till rotnoden, men som framför allt ligger nära varandra spatiellt.
Detta ger stora träd med mer redundant data (eftersom rektanglar kan överlappa varandra), men det snabbar upp den spatiella sökningen och insättning samt borttagning implementeras lätt.

b) Utifrån de givna koordinaterna traverseras trädet tills rätt ort hittats. Syskonen till noden är ju som nämnts geografiska grannar. Om matchning med lagrad folkmängd mot eftersökt inte lyckats går man vidare till syskonen i subträdet för att se om de svarar mot kriteriet. Gör de inte det, går man upp ett steg i trädet och undersök de intilliggande underträden, med de närmaste först. Man fortsätter sedan att gå uppåt i underträdsstrukturen tills man funnit det som eftersöks.

2. a) Man delar in metadata för bilder i tre typer:
- innehållsoberoende: t.ex. en bilds storlek
- innehållsbeskrivande: som i sin tur delas in i domänberoende eller – oberoende där den senare kan handla om format, kompressionnivå
- innehållsberoende: egenskaper (se nedan)

Den sistnämnda torde vara av största intresse vid bildsökning. Olika egenskaper för en bild kan vara:
- *Färg och mättnad* = Lagring av respektive delfärgs (eller monokromatiskt) histogram.
- *Struktur* = Texturberäkning som kombinerat med histogram kan användas för "feature extraction". För matchning med andra bilder i sökning anges intressegrad av de homogena områdena. Kan lagras eller beräknas vid stunden för efterfrågan.
- *Form* = gränskoordinater lagras
- *Position* = geografiska system
- *Globala egenskaper* i en subjektiv "utseende"-egenskap

Att söka på en bilds egenskaper är svårare än enkel textsökning, så SQL-sökning blir blir olämplig. Istället är det vanligt att man försöker hitta en bild som är lik ett exempel i något eller flera av ovanstående hänseenden.

Dessa grupperas således till en egenskapsvektor (featurevector), vilka kan variera beroende på viktningen av användaren. Dels vid lagring av metadata för en viss bild, då en subjektiv bestämning, om vad som är viktigt för just den bilden, görs. Och dels när man sedan söker efter lagrat material utifrån en "ny bild". Då man återigen anger vikten av de olika egenskapskomponenterna, vilka bildar en vektor som jämförs med lagrade dito. För att snabba upp sökningen kan gruppering ske, se nedan.

b) Vid klustrering av data, i vårt fall, bilders metadata, tar man hänsyn till följande:
1. Man försöker hitta den eller de egenskaper som man vill att likheten ska mätas i, t.ex. färgvärden. Om man vid sökning bara tar hänsyn till en enda egenskap för jämförelse kallas algortimen monotetisk, annars polytetisk
2. Ofta har man fler dimensioner att bearbeta så gäller det att kunna jämföra dem på en unik skala. Man bestämmer då ett lämpligt sätt att normalisera värdena. Då undviker man att extremvärden får allt för stor betydelse. Detta med hjälp av standardavvikelsen.

3. Avgör om vi kan använda domänkunskapen till något.
4. Ta hänsyn till tillvägagångssätt om stora datamängder måste processesas.
5. För att bestämma klustren ser man helt enkelt på vilka punkter som ligger närmast varandra likt uppgift 3. Avståndet mellan punkterna beräknas praktiskt med hjälp av en vanlig k-medel-avståndsformel (se nedan).

Klustreringsalgoritmen kan antingen vara hård eller ”fuzzy”, hopande eller delande, deterministiska eller stokastiska, ”singel link” eller ”komplett link”. Dessa skiljer sig för hur man delar upp sina punkter och vilka beslut som ska ligga till grund för dels indelning men även fastställande av tillhörighet (avstånd osv.)

3. Man kan dels se hur det datastrukturmässigt (programmering) är lättast att t.ex. söka element i de två givna schemana. Där har båda en enkel geometrisk uppdelning genom ett rakt streck. Ingen tydlig vinnare alltså! Så då väljer man att kolla på hur bra grupperade punkterna är gentemot sin mittpunkt. Genom att använda k-medelsteknik, med beräkning av euklidiskt avståndet, kan man avgöra bästa klusteruppdelningen.

Principen är att man beräknar var mitten (m, s.k. centroid) av varje kluster ligger och sedan tittar på elementens genomsnittliga avvikelse (a) från den punkten. Den totala genomsnittliga avvikelsen (A) för alla kluster i respektive schema avgör sedan hur bra klustreringen är.

i)

C1:

$$m = \frac{(8,4)_{p1} + (5,4)_{p2} + (2,4)_{p3}}{3} = (5,4)$$

$$a_1 = \sqrt{(8-5)^2 + (4-4)^2} = 3$$

$$a_2 = \sqrt{(5-5)^2 + (4-4)^2} = 0$$

$$a_3 = \sqrt{(2-5)^2 + (4-4)^2} = 3$$

$$a_{C1tot} = \frac{a_1 + a_2 + a_3}{3} = 2$$

C2:

$$m = \frac{(12,20)}{3} = (4,6.7)$$

$$a_{C2tot} = \frac{2.1 + 2.3 + 4.1}{3} = 2.8$$

Ger för *i*-schemat en genomsnittlig avvikelse A = (2+2.8)/2 = ***2.4***

ii)

C1:

$$m = \frac{(8,4)_{p1} + (8,6)_{p6}}{2} = (8,5)$$

$$a_{C1tot} = \frac{a_1 + a_6}{2} = 1$$

C2:

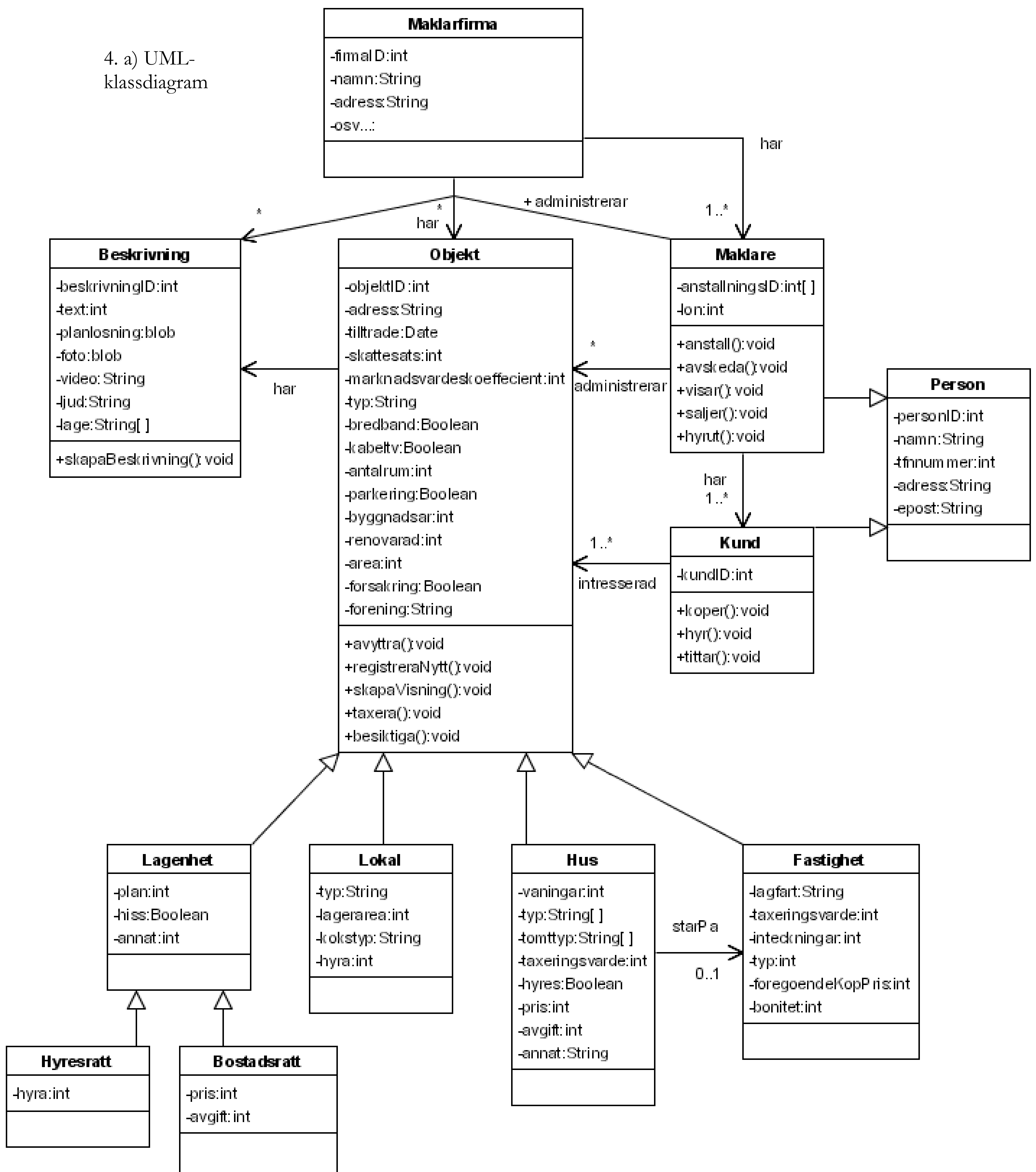
$$m = \frac{(5,4)_{p2} + (2,4)_{p3} + (2,6)_{p4} + (2,8)_{p5}}{4} = (2.75,5..5)$$

$$a_{C1tot} = \frac{a_2 + a_3 + a_4 + a_5}{4} = 2$$

Ger för *ii*-schemat en genomsnittlig avvikelse A = (1+2)/2 = ***1.5***

Svaret blir således att det andra klusterschemat är bättre eftersom den genomsnittliga avvikelsen från centroiden är mindre.

4. a) UML-klassdiagram




Stommen i mäklarfirmans verksamhet är Objekt-databasen. Denna innehåller en rad både intressanta mindre viktiga attribut. Det är snarare ut ibland instanserna, vilka ärver alla objektattribut, som man kan nämna några ord. Jag valde att dela upp de olika boendeformerna enligt ovan för att det någorlunda motsvarar de distinkta klasser man möter ute i verkliga livet bland mäklare och i andra bostadssammanhang.

Lokaler kan vara av företagstyp, restaurang, osv. Hus kan både hyras och köpas, liknar dock inte de villkor som gäller för lägenheter. Ett hus finns i olika former, ”typ”, radhus, villa osv. Vid köp måste fastigheten den står på beaktas, även om fastigheter i sig kan säljas utan hus på, likadant som hus kan stå på ”ofrigrund” (dvs. utan fastighetsbetäckning). Olika typer av inteckningar för en fastighet kan vara

servitut eller  nyttjanderätter. Bonitet står för är ett mått skogsbestånd. Alla dessa subbklasser kan kompletteras med ytterligare attribut.

Utan mängdbetäckning vid associationernas ändar betyder ”1”. Implementeringsmässig lösning för associationerna återfinns i schemat nedan.

b) Det mest väsentliga i hantering av ljud och bild/video att man måste synkronisera ljudet till videofilmen, så att ljudet startar när filmen startar. En möjlig lösning är att baka ihop ljud och video till samma fil, till exempel en mpeg-film.

Problem i all mediahantering hur man ska spara datat. Som binära stora i objekt i databasen eller som sökvägar. Om man väljer det sistnämnda gäller det att man har en bra struktur, som helst även är automatiserat och självförklarande, eller väl dokumenterat. Håller man inte det i styr kan det exempelvis bli problem att veta vilket ljud som hör samman med vilken bild/videosekvens. Någon slags indexering kommer förr eller senare bli ett måste för att matcha informationen.

Sedan kan man beakta huruvida man ska låta användaren ladda ner hela filen eller om man ska kunna se eller lyssna till strömmande medier. Oavsett, kan även det ställa till med problem.

c) Som förklarat tidigare kan man ju utifrån de bilder som finns lagrade i beskrivningsdelen av mäklarfirman göra bildsökningar. Formen på fastigheten anges som sökkriterium. Antag att man från innan, med hjälp av färger i bilden, extraherat de strukturer som finns i bilden. Sedan har man manuellt valt ut den struktur som representerar konturen på huset. Kanske inte så enkelt, och knappast självklart att det ger önskvärda resultat. Men hus med liknande färg hittas säkerligen.

En idé är att komplettera informationen med geografisk information, såsom en karta. Då kan man enkelt mäta avstånd från fastigheter till vägar som kan tänkas störa, eller var det finns friluftsområden.

Oväntade och dolda egenskaper för ett hus, såsom någon form av ohyra, fuktskador och dylikt kommer aldrig systemet till handa.

```
d) <?xml version="1.0" encoding="utf-8" ?>
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- Nedan fooljer alla klasser fraan UML-klassdiagrammets klasser, haar
som element. Utookning med nya element, jaamfort med UML:n, har gjorts
foor att koppla samman klasserna, se nedan -->

  <xs:element name="Maklarfirma">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="firmaID" type="xs:int" />
          <xs:element name="namn" type="xs:string" />
          <xs:element name="adress" type="xs:string" />
          <!-- osv... se UML-diagram -->
        </xs:sequence>
    </xs:complexType>
        <xs:unique name="firma">
          <xs:selector xpath="Maklarfirma"/>
          <xs:field xpath="firmaID"/>
        </xs:unique>
        <!-- unique-constraint saetts i komplett (inte haer) schema foor
        lla element med ett ID-subelement (id-attribut i HTML)  -->
  </xs:element>

  <xs:element name="Objekt" type="objektattribut"/>
  <xs:element name="Lagenhet" type="lagenhetsattribut"/>
  <xs:element name="Hyresratt" type="hyresrattsattribut"/>
  <xs:element name="Bostadsratt" type="bostadsrattsattribut"/>
  <xs:element name="Hus" type="husattribut"/>
  <xs:element name="Fastighet" type="fastighetsattribut"/>
```

```xml
  <xs:complexType name="objektattribut">
    <xs:sequence>
       <xs:element name="objektID" type="xs:int" />
       <xs:element name="adress" type="xs:string" />
       <xs:element name="tilltrade" type="xs:date" />
       <!-- osv... se UML-diagram -->
    </xs:sequence>
  </xs:complexType>

  <!-- arv, extends modelleras i XML med hjaelp av ComplexCOntent och
extensions -->
  <xs:complexType name="lagenhetsattribut">
    <xs:complexContent>
      <xs:extension base="objektattribut">
        <xs:sequence>
          <xs:element name="plan" type="xs:int"/>
          <xs:element name="hiss" type="xs:boolean"/>
          <xs:element name="annat" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="hyresrattsattribut">
    <xs:complexContent>
      <xs:extension base="lagenhetsattribut">
        <xs:sequence>
          <xs:element name="hyra" type="xs:int"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- likadant foor bostadsrattsattribut -->

  <xs:complexType name="husattribut">
    <xs:complexContent>
      <xs:extension base="objektattribut">
        <xs:sequence>
          <xs:element name="vaningar" type="xs:int"/>
          <xs:element name="typ" type="xs:string"/>
          <xs:element name="tomttyp" type="xs:string"/>
           <!-- osv... se UML-diagram -->
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

 <!-- likadant foor fastighetsattribut, se UML -->

  <xs:element name="Beskrivning">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="beskrivning" type="xs:int" />
          <xs:element name="planlosning" type="xs:hexBinary" />
          <xs:element name="foto" type="xs:hexBinary" />
          <!-- osv... se UML -->
        </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Person" type="personattribut"/>
  <xs:element name="Maklare" type="maklareattribut"/>
  <xs:element name="Kund" type="kundattribut"/>
```

```
  <xs:complexType name="personattribut">
    <xs:sequence>
       <xs:element name="personID" type="xs:int" />
       <xs:element name="namn" type="xs:string" />
       <!-- osv... se UML-diagram -->
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="maklareattribut">
    <xs:complexContent>
      <xs:extension base="personattribut">
        <xs:sequence>
          <xs:element name="anstallningsID" type="xs:int"/>
          <xs:element name="lon" type="xs:int"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="kundattribut">
    <xs:complexContent>
      <xs:extension base="personattribut">
        <xs:sequence>
          <xs:element name="kundID" type="xs:int"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

 <!-- Nycklar -->
 <xs:element name = "Nycklar">
       <xs:key name="firmaKey">
           <xs:selector xpath = "Maklarfirma"/>
           <xs:field xpath = "firmaID"/>
       </xs:key>

       <xs:key name="objektKey">
           <xs:selector xpath = "Objekt"/>
           <xs:field xpath = "objektID"/>
       </xs:key>

        <xs:key name="maklareKey">
           <xs:selector xpath = "Maklare"/>
           <xs:field xpath = "anstallningsID"/>
       </xs:key>

       <xs:key name="kundKey">
           <xs:selector xpath = "Kund"/>
           <xs:field xpath = "kundID"/>
       </xs:key>

      <!-- se foorklaring till denna bindning nedan vid element
"firmaTillObjekt1" -->
      <xs:keyref name="firmaTillObjekt1" refer="firmaKey">
           <xs:selector xpath = "firmaTillObjekt"/>
           <xs:field xpath = "firmaID"/>
       </xs:keyref>

       <xs:keyref name="firmaTillObjekt2" refer="objektKey">
           <xs:selector xpath = "firmaTillObjekt"/>
           <xs:field xpath = "objektID"/>
       </xs:keyref>
```

```
      <xs:keyref name="maklareHarKund1" refer="maklareKey">
           <xs:selector xpath = "maklareHarKund"/>
           <xs:field xpath = "anstallningsID"/>
       </xs:keyref>

       <xs:keyref name="maklareHarKund2" refer="kundKey">
           <xs:selector xpath = "maklareHarKund"/>
           <xs:field xpath = "kundID"/>
       </xs:keyref>

       <!-- OSV.....-->
  </xs:element>

  <!-- Foor alla bindnigar i UML-schemat (har, administrerar, staarPaa,
osv...) kommer bindas ihop i XML med referenstabeller likt vanlig RDMS.
Haer kommer ett exempel -->
  <xs:element name="firmaTillObjekt">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="firmaID" type="xs:int" />
          <xs:element name="objektID" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Haer kommer ett till -->
  <xs:element name="maklareHarKund">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="anstallningsID" type="xs:int" />
          <xs:element name="kundID" type="xs:int" />
        </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- OSV.....-->
</xs:schema>
```

5. Med det knappa bakgrundsmaterialet – en rejäl kravspecifikation brukar ju stå som grund för liknande modelleringsprojekt – så är man tvungen att göra vissa antaganden.

Handlar det exempelvis om ett kompetent fotogalleri (dvs. med viss dynamik) för Herr Någon, kan en renodlad JSP-lösning vara enklast. Personligen hade jag i så fall föredragit någon av de konkurrerande teknikerna, ASP eller främst PHP. Inte minst med tanke på att fler kommersiella webbhotell stöder dessa framför JSP, men även p.g.a. en viss rädsla för komplexiteten kring ”allt det andra med JSP”…

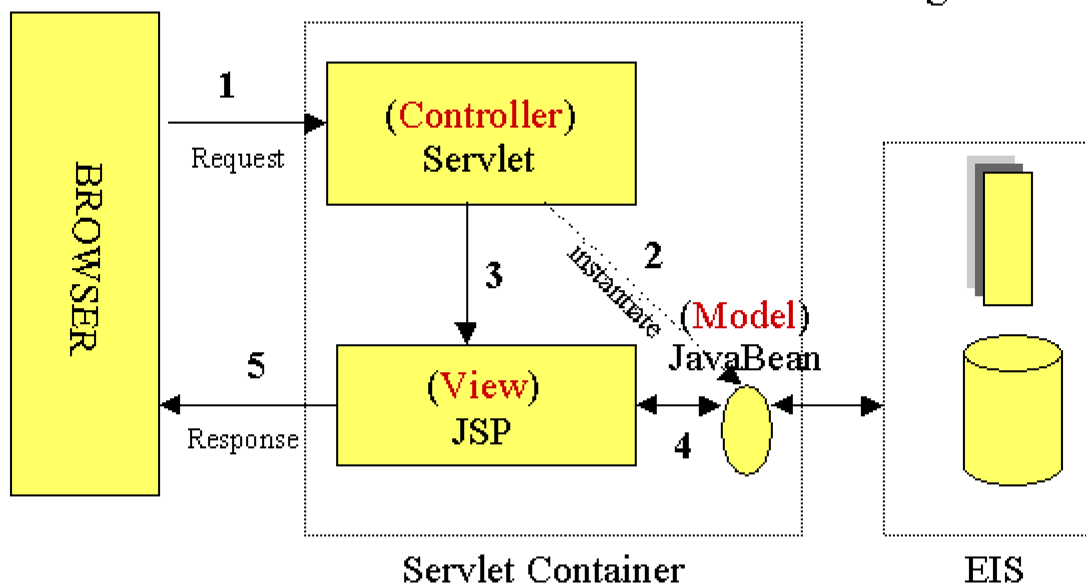
Just de andra bitarna är de intressanta för att lösa denna uppgift. Vi får helt enkelt anta att det rör sig om ett större projekt där helhetslösningen skall innefatta:

- många användare
- användare med vitt skilda behov, därmed helt dynamisk

men även med hög repetitivet

- skalbarhet och plattformsoberoende
- objektorientering, återanvändning av ”kod”, funktioner
- lågkostnadskrav

Vi hamnar på en helt annan komplexitetsnivå. Därför vill vi dra nytta av hela programspråket Javas potential. Detta kan åstadkommas endast med inblandning av servlets och till viss del även med s.k. Javabönor.
Ovan tre nämnda javatekniker, plus såklart clientsidan och applikations- eller databaskopplingen, finns alla representerade i Model-View-Controller-tekniken (se figur nedan).




Detta är en serversideimplementation där alla komponenter har sina respektive uppgifter. Den huvudsakliga avgränsningen görs mellan presentations- och kontrollkomponenter. Presentation (View) sker oftast med JSP (ibland servlets) som genererar HTML eller XML-innehåll som skickas för rendering av användargränssnitt av nätbläddraren. Här kommer alltså svaret på frågan om HTML:s betydelse. Det används fortfarande som layoutspråk för visning av text, grafik, bilder m.m. Tekniken tillhandahar även formulär för vidarebefordring via http till server: Grunden för nästan all interaktivitet i ett databasdrivet system som detta.

Vidare hanterar kontrollkomponenterna (Control) dessa http-förfrågningar. Denna del kan bestå av antingen en servlet eller en JSP-sida. I fallet JSP kompileras förfrågan i en JSP-motor som i sig är en specialiserad servlet som är under hela servlet-kontainerns kontroll. Filen blir således en ”JSP-servlet”. När klassfilen sedan är laddad sker all kommunikation (fråga/svar) via metoden _jspService().

I fallet servlet kompileras en källkodsfil, om den inte redan finns, till klassfil som i JSP-fallet. Denna klassfil kommer kunna återanvändas, vilket snabbar upp för efterföljande användare. Samma classfil som för JSP-fallet genereras och man anropar någon JSP-fil, eller eventuellt annan servlet, för presentation. Faktum är att vi återkommer till inledningen av frågan. Man kunde lika gärna löst det med JSP, inte minst med tanke på att JSP i sig är enklare att koda. Men även p.g.a. att endast små förändringar av användargränsnittet kräver omkompilering av servlet:en, och man tappar återanvändbarheten. Detta problem återfinns inte för JSP som redan från början är starkt kopplat (inbäddat i html) till presentationen. (Kompileras iofs. alltid)
MEN servlets kan mer:

- körs inte i separata processer
- hanterar persistens
- en enda instans svarar för alla förfrågningar samtidigt
- erbjuder färdiga moduler för sessionshanering (med hjälp av cookies), äkthetsbevisning, kommunikation med applets och applikationer.

Och här kommer vi till M:et i vår akronym MVC. För att ytterligare bryta ner programmeringsprojektet kan s.k. bönor användas. Man har helt enkelt ”outsourcat”, för att använda ett modeord, vissa av beräkningstunga och programmeringsintensiva delar i förkompilerade objekt, ”Models”. På begäran har dessa skapats av Control-delen för att exempelvis prata med en databas. Är det i ett inmatningsskede för vår multimediaapplikation kan det handla om att exempelvis koda (komprimera) en bild.

En annan böna kan sedan accessas eller uppdateras av presentationslagret för att exempelvis visa upp denna bild, omskalad som tumnagel. Dessa bönor effekteviserar på så sätt att man återanvänder dem gång på gång och de behöver inte omkompileras förän total systemförändring skall ske.

Hela denna struktur möjliggör klara uppdelningar av arbetsbördan i en projektgrupp. Distingerar slutanvändar-kunniga programmerare från ”algoritmtokar” osv. Gör även hanteringen av dataflödet och kopplingarna mot databasen mer strategisk och överskådlig, och systemet blir säkerligen mer säkert.

6. a) **CGI, Common Gateway Interface**

Som hela den här frågan handlar den här frågan om skapandet av webbapplikationer där innehållet och utseende kan förändras genom inmatning av användaren eller p.g.a. andra faktorer. Det handlar alltså om det som uttrycks lite slarvigt: ”dynamiska webbsidor”, likt fråga 5. En teknik som tidigt togs i bruk för detta ändamål var CGI.

Det är ett program som körs i realtid på serversidan och matar ut i många fall vanlig statisk HTML. Färdigkompilerade applikationer för att lösa i princip alla tänkbara uppgifter, såsom koppling till data bas. Man kan använda språk som t.ex. C/C++, Fortran, PERL, TCL, godtycklig Unix shell, Visual Basic, AppleScript.

*Fördelar:*
- välutbredd teknik och stöd för många språk och med alla typer av servrar
- relativt lättprogrammerat (beroende på språk)
- flexibelt

*Nackdelar:*
- lägger stor börda på servern och varje function som inkluderas I körningen av cgi ger en ny uppgift (tråd) för servern.
- kan bli långsamt.
- lite dålig felhantering, I fallet formulär behövs “räddning” av ifyllda värden implementeras.

b) **Java Servlets och JSP.**

För beskrivning av teknik och till vissa delar svar på frågan se fråga 5.

*Fördelar:*
- körs inte i separata processer
- hanterar persistens
- en enda instans svarar för alla förfrågningar samtidigt
- erbjuder färdiga moduler för sessionshanering (med hjälp av cookies), äkthetsbevisning, kommunikation med applets och applikationer.
- Full access to all Java APIs
- Plattformsoberoende

*Nackdelar:*
- Programmatic HTML output is hard to maintain: out.println(“<html>…”);
Requires manual compilation.
Low-level; harder to work with (similar to ISAPI DLL)
Better to use JSP for building HTML display output.

c) **Socketar**

Socketar möjliggör kommunikation mellan processer/program i ett nätverk eller internt mellan olika processer på samma dator. Det finns olika typer av socketer beroende på användningsområde, de man brukar prata om kallas för connnection-based (klient/server-modell) och connectionless (internt mellan kommunicerande processer).

Även om de olika typerna skiljer sig åt i metod så är tanken densamma. Att kunna få program skriva i olika språk att effektivt kunna kommunicera med varandra, utan påverkan av http-lagrets, och port 80, begränsningar.

*Fördelar:*
- Många servrar körs på unix, där det är socketskommandon är lättåtkomliga . Men det ger även möjlighet att på ett smidigt och uniformt sätt kommunicera mellan maskiner av olika typer.
- programspråksoberoende och även enkelt att få program som är skrivna i olika programspråk att kommunicera
- bättre möjligheter, ej bunden till port 80 och webbläsaren
- enkelt att distribuera uppgifter från servern till klienterna. Kan användas vid till exempel en renderingsfarm.

*Nackdelar:*
- Kräver att både server och klient arbetar under samma förutsättningar, och att de alltid jobbar med samma version av lösningar.
- Kan bli svårt att få en överblick över ett stort system då allt inte finns samlat centralt på en server utan utspritt över flera servrar och klienter.
- Till viss del abstrakt och svårbegripligt

d) **CORBA**
CORBA är ett försök till standardisering av objekttekniker, vilket leds av gruppen Object Managmenet Group (OMG). Tanken är att skapa specifikationer för att få fram möjligheter till objektorienterad applikationsfunktionalitet och distribution oberoende av plattform, operativsystem, programmeringsspråk, nätverk och protokoll. CORBA är en komplett Server-Klient modell och för att kontrollerat kunna sköta kommunikationen mellan dessa så går alla förfrågningar via en mäklare, Object Request Broker(ORB). Orben agerar som en ”Object Bus” över vilken varje CORBA-objekt interagerar transparent med andra CORBA objekt som kan vara lokalt placerade eller på nätverk.

CORBA är alltså ett exempel på en middleware - en programvara som fogar samman andra programvaror. Detta gör att programmerare kan koncentrera sig på sitt program mer än hur de ska lösa kommunikation eller dataåtkomst.

*Fördelar:*
- ett språk- och plattformsoberoende (så länge en CORBA ORB är installerad) objektorienterat system för distribution
- kostnadseffektivt
- binder inte utvecklingen till en viss leverantör
- Finns bindningar till flera programspråk, C++, ADA, Smalltalk, Java, C, COBOL, ...

*Nackdelar:*
- Svårare än många andra liknande tekniker att implementera presentationslagret på ett smidigt sätt.
- relativt komplicerat

e) **Java Web Smart**
Fristående applikation som laddas hem till klienten som likt många tekniker ovan gör anspråk på att vara plattformsoberoende. Applikationen kan göras godtyckligt stor och ger relativt stor frihet i design. Den packas ihop och kommer som en jar-fil. Kravet är Java2. JWS använder XML för att beskriva det som ska laddas ner. Om aktuell version på servern redan finns hos klienten laddas inget ner utan den redan nerladdade versionen startas. Annars laddas den nya versionen ner först.

*Fördelar:*
- automatisk uppdatering eller återanvändning
- självförklarande dokumentstruktur som XML är bra för ex. lång livslängd

*Nackdelar:*
- Java Virual machine krävs