Om HCML og snakkesalige robotter, også kaldet chatbots

Henning Christiansen
IMT, Roskilde Universitet
https://ruc.dk/~henning/, henning@ruc.dk

Resumé

Dette notat giver en introduktion til det formelle sprog HCML med hvilket, man kan bygge sine egne chatbots. Et script i HCML definerer at antal mønstre, som bruges til at matche en brugers input, med tilhørende specifikationer af, hvordan der genereres et svar fra chatbot'en. Scripts formuleret i HCML kan fortolkes af et program, aktuelt skrevet i Processing, der således fungerer som et interaktivt dialogprogram.

1 Indledning

En chatbot er et system, som implementerer dialoger mellem en computer og en (som oftest menneskelig) partner: De fleste chatbots er ikke baseret på avancerede datalingvistiske metoder, men på simpel mønstergenkendelse: Der foreligger et større eller mindre antal mønstre, hvor designeren har forsøgt at forudsige, hvilke udsagn og spørgsmål den menneskelige bruger typisk vil benytte i den givne sammenhæng, og til hvert mønster findes så en beskrivelse af svar, som så kan varieres mere eller mindre sofistikeret.

Chatbots er meget udbredte i grænseflader som hjælpefunktioner, tænk på Apples Siri, og hvad der dukker på forskellige hjemmesider.[1] Chatbots benyttes også i virtuelle verdener, hvor de typisk er forklædt som avatarer. Såkaldte question-answering systemer (søg efter emnet!) er ofte baseret på de samme teknikker, men typisk koblet med en mere avanceret videnbehandling.

Når man skal designe en chatbot, dvs. et sæt regler for spørgsmål og svar, kan man selvfølgelig programmere sådan en chatbot helt fra bunden i et eller andet programmeringssprog, men det er ikke særligt hensigtsmæssigt af flere grunde.

- Det er besværligt, det tager tid, og det er svært at modificere, hvis man får lyst til at tilføje nye mønstre og svar eller finpudse dem, man allerede har lavet.
- Programmeringssprog handler om noget andet end dialoger; de handler om tal, kontrolstrukturer, objekter og mange andre besynderlige ting, som vi ikke forbinder med normale sproglige dialoger.

[1] SAS og IKEA har tidligere haft deres »Anna« og »Eva«, men de er senere blevet fjerner. Man kan så spekulere på, hvorfor ...

- Det er ikke alle som er gode til at programmere, det er ikke alle som ønsker at blive gode til det — og hvorfor skulle de da også bruge tid på det, hvis de er mere interesseret i at designe meningsfulde dialoger.

Det er mere hensigtsmæssigt at benytte et sprog, som er specielt udviklet til at beskrive[2] dialoger med, dvs. forhåbentligt meningsfylde sekvenser af udsagn og spørgsmål; en sådan beskrivelse kaldes ofte et `script`. HCML er et eksempel på sådan et sprog, udviklet af Henning Christiansen i starten af 2017. Navnet er en forkortelse for *H*ot *C*hatbot *M*eta-*L*anguage.

Lige en kort bemærkning om terminologien før vi går igang. »Bot« bruges i visse kredse som en kort form af ordet »robot«, og både »bot«, »robot« og »chatbot« bruges ofte om konverserende programmer, evt. koblet sammen med en avatar, selvom det nok er at strække de sædvanlige definitioner af, hvad en robot er, temmelig langt.

Chatbots og HCML er taget med i kurset, fordi de giver en glimrende illustration af, hvor enkle midler, der skal til, for at man kan få et computerprogram til at give et indtryk afm at der er en eller anden form for »intelligens« bagved, eller i det mindste kompetencer, som minder lidt om, hvad vi forbinder med »intelligens«. En af de store mangler ved HCML og lignende sprog, er at de ikke er integreret med en egentlig vidensrepræsentation og mekanismer til at ræsonnere om en sådan viden.

HCML er kraftigt inspireret at et tilsvarende sprog kaldet AIML, udviklet år 2000 [2]. Scripts i AIML kan udføres på en fast server via hjemmesiden `http://www.pandorabots.com`, og udmærker sig ved ved at den kan tilgås via protokollen http. Dette gør, at man kan benytte AIML til chatbotter i andre systemer. Disse, såkaldte pandorabots, var eksempelvis meget populære i Second Life, dengang dette virtuelle-verdens-system var på mode. AIML er en forkortelse for »Artificial Intelligence Markup Language«, hvilken er en smule misvisende, da begrebet kunstig intelligens dækker over væsentligt mere og andet end, hvad der kan udtrykkes i AIML.

Det har ikke været muligt at finde en standalone-implementation af en AIML-fortolker (hjemmeside nævner et par stykker, men de fungerer ikke eller er ikke tilgængelige). Dette er hovedbegrundelse for, at din lærer har udviklet en ny variant kalde HCML og udviklet et system, så du kan køre dine HCML scripts på din egen computer. HCML medtager ikke alle faciliteter i AIML, men til gengæld er der tilføjet et simpelt databasebegreb, så man kan indbygge en vis mængde predefineret viden i sit script, f.eks. et varekatalog, en prisliste eller en beskrivelse af kongehusets indbyrdes familierelationer. Dette har manglet i AIML, version 1.0, men der foreligger en skitse af en version 2.0, som indeholder faciliteter af den art (der foreligger ikke en detaljeret sammenligning af HCML og AIML 2.0). Der vil være scripts som kan køre som både HCML og AIML, og der vil være scripts som kun kan køre i den ene af dem.

Den videnskabelige litteratur levner ikke meget opmærksomhed på AIML eller Pandorabots, hvilket formentlig skyldes den dårlige dokumentation, men alligevel har Pandorabots flere gange vundet den prestigefyldte Loebner-pris for intelligente chatbots; se mere om det på `http://www.loebner.net/Prizef/loebner-prize.html`.

AIML, HCML og de fleste andre scriptsprog for chatbot'er er baseret på principper beskrevet afJ. Weizenbaums program Eliza, som er beskrevet i en artikel fra 1966 [3];. Weizenbaums mål

[2]Bemærk vor sprogbrug. Vi taler om at »beskrive« dialoger, i modsætning til at sige at man »programmerer« dem eller sin chatbot.

var bl.a. at vise, hvor nemt, det er at snyde en bruger til at tro, at den maskine vedkommende samtaler med, er et menneske. Weizenbaums primære eksempel var en parodi på en bestemt psykiatrisk metode, som går ud på at få patienten til at snakke om løst og fast, og psykiateren skal blot holde snakken gående uden at stille undersøgende spørgsmål. Hans artikel nævner eksempler, hvor folk faktisk hoppede på den. Der ligger vist også den sarkastisk kommentar nedenunder Weizenbaums artikel, at meget samtale mellem mennesker (herunder specielt »Rogerian« psykiatri) er ren tomgang.

Hvor Weizenbaums Eliza-program blev fodret med specifikationer formuleret i en afart af programmeringssprogetLISP, benytter HCML og AIML en tilsvarende repræsentation bygget i termer af XML. Idéen med at beskrive Eliza-agtige dialoger i termer af XML er en oplagt og indlysende ide, som giver en »moderne« syntaks, som gør det nemmere at vinde accept, og som gør det muligt at udnytte de myriader af værktøjer og teknikker, der understøtter XML. Mange flade tekst-editorer er udstyret med XML-syntakscheck.

Det må dog understreges, at der findes langt mere avancerede metoder til sproganalyse, vidershåndtering og dialoger, end man får indtryk af ved at læse om HCML og AIML. Disse metoder er dog i de fleste tilfælde svært tilgængelige og meget forudsætningskrævende at kunne udnytte, hvor HCML/AIML har den indlysende fordel, at man kan gå igang med det samme uden det mindste kendskab til lingvistik og datalingvistik. Jurafski og Martins bog fra 2009 [1] er en grundig og velskrevet grundbog om processering af talt og skrevet sprog.

I det følgende beskrives og eksemplificeres HCML og hvordan du kan bygge dit eget HCML-script og få det til at fungere som en chatbot vha. det program, som er tilgængeligt på `https://ruc.dk/~henning/hcml`.

2 Meget kort om XML

XML er en meget generel, tekstlig repræsentationsform for strukturerede og semistrukturerede data. Fordelen ved den tekstlige repræsentation er, at den er nem at opbevare på et hvilket som helst datamedie, den kan sendes over en vilkårlig transmissionslinje, og den er nem at pakke ind og ud. Der findes desuden databasesystemer som er baseret på XML. Nyere versioner af Microsoft Office-pakken benytter XML til intern repræsentation, og XML er officiel standard for dataudveksling i IT-opgaver for den danske stat.

Et XML-dokument er et tekstdokument, som overholder visse betingelser. Her er et eksempel på et XML-dokument.

```
<kursusnote aarstal="2019">
   <forfatter>
      <fornavn>Henning</fornavn>
      <efternavn>Christiansen</efternavn >
   </forfatter>
   <titel>Om HCML og snakkesalige robotter, også kaldet chatbots</titel>
   <abstract/>
</kursusnote>
```

Dokumentet består af »items« opdelt ved hjælp af såkaldte *tags* og *end-tags* (eller *closing tags*), som er indikeret med større-end- og mindre-end-tegn med en skråstreg til at indikere end-tags. Som eksempel skrives et tag »`forfatter`« som `<forfatter>` og det tilhørende end-tag som `</forfatter>`. Et XML-dokument kan også indeholde løbende tekst. Et tag skal matche med sit end-tag på samme måde som indlejrede paranteser, og til sammen kaldes et tag, dets end-tag og det der er indimellem, for et *item*. Når der ikke er noget indhold mellem tag og end-tag, kan de smelte sammen som i `<abstract/>` hvilket er en kort måde at skrive `<abstract></abstract>`. Et tag kan også have tilknyttet en eller flere *attributter* som i `<kursusnote aarstal="2019">`. Tag'et er »`kursusnote`«, attributten (s navn) er »`aarstal`« og dennes *værdi* er »`2019`«. Tegn som mellemrum og linieskift tæller ikke med, de tjener blot til at adskille ord, dvs. om der er et eller 10 mellemrum mellem to elementer er fuldstændigt ligegyldigt. Indrykningerne i dokumentet ovenfor tjener da også kun til overskuelighed og er teknisk set unødvendige.

Hvis man så lægger specifikke begrænsninger på, hvilke tags som er tilladte, hvilke attributter de kan tage og hvilke tags, som kan forekomme inde i hvilke, opnår vi hvad vi kan kalde en XML-dialekt, og HCML er et eksempel på en sådan.

HTML, som du måske kender fra websider,[3] er faktisk ikke en XML-dialekt, selv om det ligner til forveksling. Den lille detalje, som gør forskellen er at visse tags kan skrives uden end-tag, f.eks. `<br>`, som svarer til eksplicit indsættelse af et linieskift.

Mere behøver vi ikke vide om XML før vi fortsætter læsningen af denne note. Specielt interesserede kan læse alt om XML på `http://www.w3schools.com` og `http://www.w3.org`.

3 Tekniske forudsætninger

3.1 Processing

Du skal allerførst installere Processing på din computer, hvis du ikke allerede har det. Gå til `http://processing.org` og find ud af, hvordan du downloader den nyeste version. Du modtager så en fil, `Processing` med eller uden et efternavn. Placér den i din program-mappe eller hvad det nu hedder på din computer, og så er du kørende.

Advarsel: Det er vigtigt, du bruger en nyere version af Processing, minimum 3.5.2. Tidligere versioner har haft fejl i faciliteterne til håndtering af XML.

3.2 HCML-systemet

Dernæst skal du download HCML-systemet fra `https://ruc.dk/~henning/hcml`; der ligger en fil `HCML.zip`, som du pakker ud (hvis ikke det allerede er gjort automatisk), og læg mappen `HCML` et fornuftigt sted. For at gøre det nemt at starte systemet, anbefales det, at du går ind i mappen og skaber et alias/en genvej til filen `HCML.pde`; den kan du lægge på dig skrivebord eller placere den, hvor du plejer at have den slags.

[3] Hvis du vil se noget HTML-kode, så prøv at vælge »View source« (eller hvad det måtte hedde) i din web-browser, når du kigger på en ikke alt for dynamisk webside.

3.3 Skriv dine HCML-scripts med en flad teksteditor

Scripts i HCML skal skrives som flade tekstfiler, ellers duer de ikke. En flad tekstfil er én, som ikke indeholder formatteringer eller lignede; det duer slet ikke at skrive dit script i Word eller lignede og gemme filen, som du plejer. En flad fil indeholder en sekvens af tegn og ikke andet.

På Mac kan du bruge den editor, som hedder TextEdit, men det er vigtigt at du går ind i dens »Preferences« og klikker på »Plain text« (så vi slipper af med »Rich text«). Først når du har gjort det, kan du åbne et nyt dokument, som så vil være en flad tekstfil. (Der dukkede måske et dokument op, da du startede TextEdit, med en masse krimskrams som tydeligt viser, at det ikke er en flad fil – skynd dig at lukke det).

Historen er tilsvarende for Windows, her hedder editoren bare Notepad.

Der findes bedre editorer end de nævnte, og måske du har en allerede. Det er allerbedst, hvis din editor er forberedt for XML, så den hjælper dig med at få syntaksen rigtig.

OBS: Hvis du klippe-klistrer tekst fra dette notat over i dit eget script går der næsten med garanti kludder i de dobbelte anførselstegn, som bliver lavet om til nogle andre, som ligner til forveksling, og så brokker HCML-systemet sig mærkeligt. Erstat dem med de dobbelte anførselstegn, som du får gennem dit tastatur.

Advarsel: Din lærer er en lidt doven programmør og har ignoreret visse problemer omkring tegnsæt. Erfaringsmæssigt fungerer æ-ø-å korrekt på Mac, men en gang imellem gør de ikke på Windows.

3.4 At køre et HCML-script

Når du har dit script parat, kan du starte HCML gennem det alias/den genvej du lavede ovenfor. Så vil der i første omgang dukke et Processing-vindue op, som indeholder den programtekst, som implementerer systemet. Det kan du se bort fra, det mest interessante er de to knapper foroven til venstre. Du starter HCML ved at trykke på trekantsymbolet, og så dukker det interessante vindue op med overskriften »HCML Chatbots«. Følg derefter den korte instruktion på skærmen, om hvordan du loader dit script.

Du kan stoppe HCML ved at lukke dets vindue eller trykke på knappen med firkantsymbolet i Processing-vinduet.

4 Oversigt over HCML og HCML-systemet

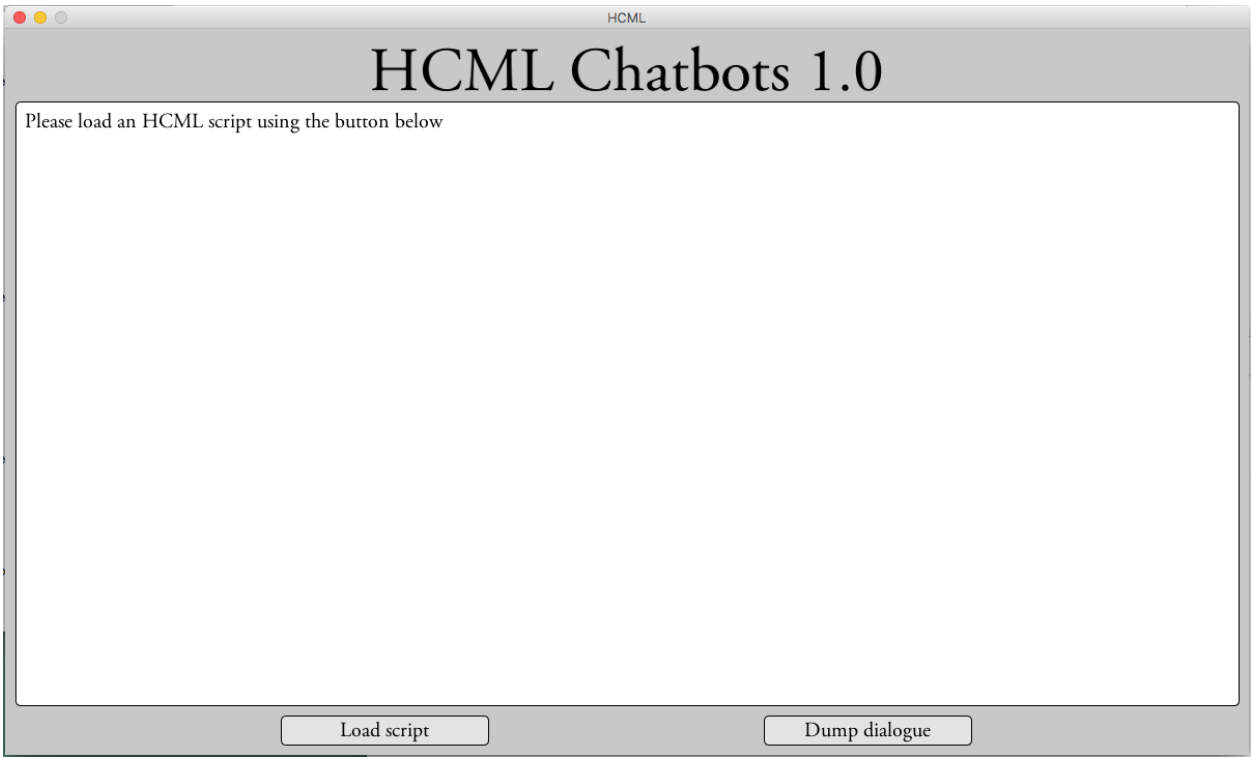
Dette er et eksempel på et HCML script, som vi antager ligger på filen `test.xml`.

```
<?xml version="1.0"?>
<hcml version="1.0">

<welcome bot_name="SuperBot">
     Jeg er en høflig og jovial superbot</welcome>

<category>
   <pattern>Godaw</pattern>
   <template>Godaw og velkommen til
   </template>
</category>

<category>
   <pattern>*</pattern>
   <template>
      <random>
         <li>Helt enig.</li>
         <li>Wof'det?</li>
         <li>Skål</li>
      </random>
   </template>
</category>

</hcml>
```

Det første item, `<welcome ...>`, bruges at give robotten et navn, her `SuperBot`, og en indledningsreplik.

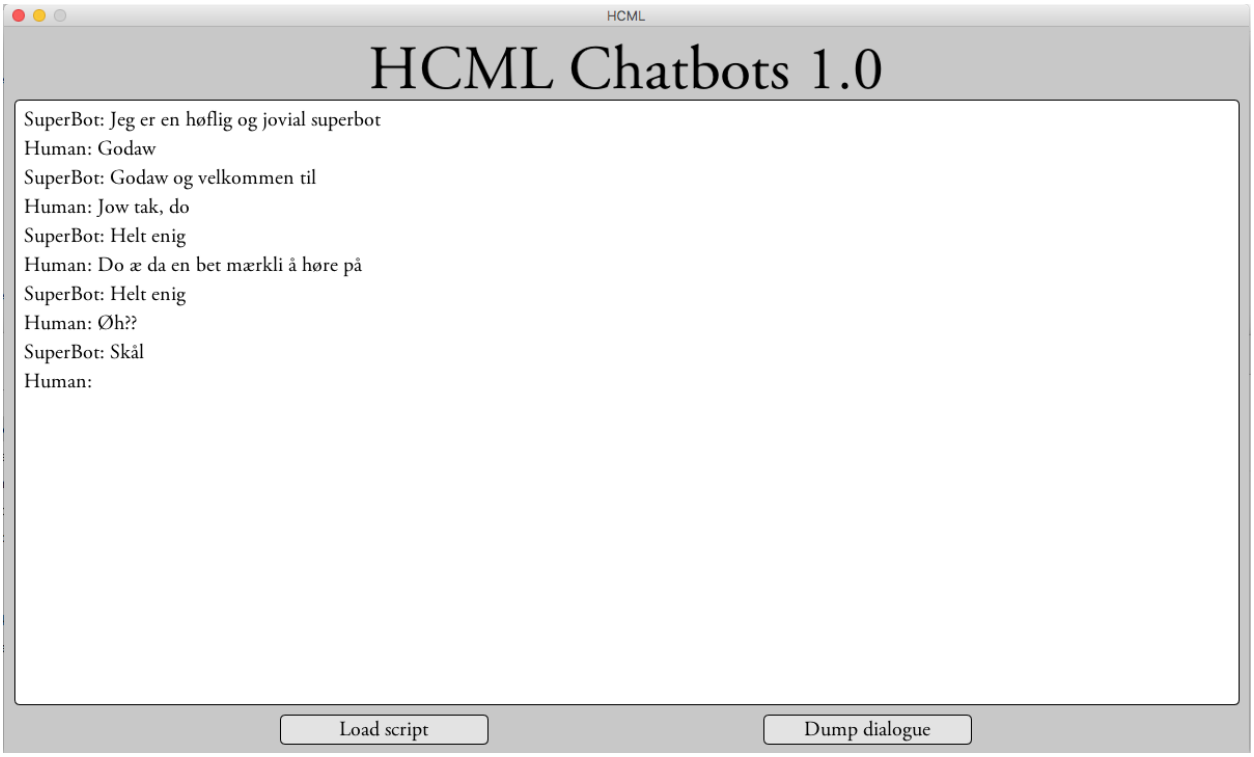
En `category` beskriver én type spørgsmål med tilhørende svar (vi skriver fremover »kategori« i den løbende tekst). Mere præcist, hvis en indtastning givet af brugeren matcher det givne `pattern`, så beskriver den tilhørende `template` svaret (eller de mulige svar). Den første kategori kommer i sving, hvis brugeren taster »`Godaw`«, og her vil systemet svare »`Godaw og velkommen til`«. Den anden kategoris pattern består af et såkaldt wildcard »`*`«, som i denne sammenhæng matcher hvad som helst, og det tilhørende svar vil så blive valgt tilfældigt blandt de tre nævnte alternativer. – De to første linjer skal bare være der og betyder ikke noget interessant, og der skal sluttes med `</hcml>`.

Nu skal vi se, hvordan scriptet kan køres HCML-systemet. Når vi starter systemet, ser dets vindue i første omgang sådan ud.



Vi følger nu anvisningen, trykker »Load script« og der dukker en sædvanlig filmenu op, så vi kan navigere frem til og vælge vores fil `test.xml`. Hvis systemet ellers kan forstå filen, vil chatbotten præsentere sig, og den og brugeren skiftes så til at skrive en sætning.

Efter et stykke tid kan vindue se sådan ud.



4.1 Om brug af tastaturet

Du kan bruge piletasterne pil-op og pil-ned for at rulle frem og tilbage mellem tidligere indtastninger, og pil-frem og pil-tilbage til at flytte curseren frem og tilbage (du kan ikke flytte den med musen).

I den nuværende udgave er det ikke muligt at klippe eller klistre fra/til HCML-vinduet, men kommandoen »Dump dialogue« vil skrive teksten fra vinduet over i den nederste del af Processing-vinduet, og derfra kan du kopiere den på normal vis; dette er nyttigt, når du skal skrive en rapport.

Advarsel: Systemet kan gå i stå, hvis man holder en tast nede i længere tid. Det er kun set på en enkelt computer, og normalt skulle det blot repetere tegnet. Hvis det sker, må du stoppe og genstarte HCML-systemet.

4.2 Fejlmeddelelser

Hvis din script-fil ikke kan læses korrekt, vil systemet brokke sig. Det kan skyldes, at filen ikke overholder XML-standarden (start-tag og slut-tag skal matche osv.), eller at de tags, som er brugt, ikke stemmer overens med HCML-syntaksen.

Disse fejlmeddelelser vil blive skrevet i Processing-vinduet (det nederste, sorte delvindue). De er forsøgt gjort så forståelige som muligt, så man kan finde og rette fejlen. (En gang imellem skriver Processing af sig selv noget besynderligt vås i det vindue, som man bare kan ignorere).

5 Grundbegreber i HCML

Som vist i afsnit 4 indledes et HCML-dokument med en linje, som ikke siger noget særligt (men bare skal være der), og derefter følger den interessante del af dokumentet omsluttet af af strukturen `<hcml ··· > ··· </hcml>` .

```
<?xml version="1.0"?>
<hcml version=1.0">
   <category> ... </category>
   <category> ... </category>
    ...
   <database> ... </database>
</hcml>
```

Et HCML-script indeholder et sekvens af kategorier, som hver består af et `pattern` og en `template`. Som vist i afsnit 4 benyttes `pattern`-delen til at matche brugerens input og den tilhørende `template` bestemmer et svar fra chatbot'en. Desuden findes en databasekomponent, som vi ikke har set endnu, og den er forklaret i afsnit 5.5 nedenfor.

Som vi har set tidligere, ser en kategori overordnet således ud.

```
<category>
   <pattern>  ... </pattern>
   <template> ... </template>
</category>
```

En kategori kan desuden indeholde nul eller flere `that`-items, som forklares nedenfor i afsnit 5.4.

```
<category>
   <pattern>  ... </pattern>
   <that>     ... </that>
   ...
   <that>     ... </that>
   <template> ... </template>
</category>
```

5.1 Wildcards

Et `pattern` kan være et fast mønster, som kun matches af den samme eksakte tekst (vist i eksemplet i afsnit 4). Man kan opnå væsentlig mere fleksibilitet gennem såkaldte *wildcards*. Et wildcard kan skrives i et pattern som et stjerne eller asterisk, »`*`«. Her er et eksempel:

```
<pattern>Jeg elsker *</pattern>
```

I dette eksempel vil `pattern`et matche hvad som helst, der begynder med »`Jeg elsker`«. Den del, som er blevet matchet af wildcardet kan hentes frem igen i den tilhørende template ved hjælp af et `<star/>` tag. Vi demonstrerer dette i denne kategori:

```
<category>
   <pattern>Jeg elsker *</pattern>
   <template>Nå, så du elsker <star/>, fortæl mig lidt mere om <star/></template>
</category>
```

Det kan eksempelvis give anledning til denne dialog:

Human: Jeg elsker pandekager
Bot: Nå, så du elsker pandekager, fortæl mig lidt mere om pandekager.

Man kan have flere wildcards i samme pattern, og i såfald skal `star`-tag'et udstyres med et nummer på hvilket wildcard, man ønsker at referere til. For eksempel:

```
<category>
   <pattern>* elsker *</pattern>
   <template>Jamen, guuuud, er det sandt at <star index="1"/>
        æææælsker <star index="2"/>??
        Ved du om <star index="2"/> også æææælsker <star index="1"/>??
    </template>
</category>
```

Dialogen bliver nu eksempelvis:

Human: Klavs Krikke elsker Nora Malkeko
Bot: Jamen, guuuud, er det sandt at Klavs Krikke æææælsker Nora Malkeko?? Ved du om Nora Malkeko også æææælsker Klavs Krikke??

Som der er vist i afsnit 4, er det en god ide at afslutte et script med en kategori som indeholder `<pattern>*</pattern>`, således at chatbot'en aldrig løber tør for svarmuligheder.

5.2 Lidt mere om matching af input

Brugerens input forsøges matchet af de patterns, som er i HCML-scriptet i den rækkefølge de står. Når det første match er fundet, aktiveres den tilsvarende template, så hvis man i eksemplerne ovenfor ønsker en særbehandling af »`Jeg elsker *`« fremfor den generelle »`* elsker *`«, skal man skrive dem i den rækkefølge (dvs. »`Jeg elsker *`« først).

Matchingen med eksplicitte ord i pattern tager ikke hensyn til store og små bogstaver. Dog bevares store/små bogstaver i de sekvenser som sendes videre i `<star/>`. I machingen filtreres specialtegn fra, så eksempelvis »`Klavs Krikke elsker Nora Malkeko`« og »`Klavs Krikke elsker Nora Malkeko!!!!`« giver samme resultat.

5.3 Tilfældighed i dannelse af svar

Når du fører en dialog med et andet menneske, er en af de egenskaber du vil forvente, at vedkommende ikke svarer det samme hver gang, selv på det samme spørgsmål. Det kan man til en

vis grad simulere ved HCMLs <random> tag til at vælge mellem alternative svar i sine templates. Her følger et eksempel.

```
<category>
   <pattern>* elsker *</pattern>
   <template>
      <random>
         <li>Jamen, guuuud, er det sandt at <star index="1"/>
             æææælsker <star index="2"/>??
             Ved du om <star index="2"/> også æææælsker <star index="1"/>??
         </li>
         <li>Nå.
         </li>
         <li>Hvad med dig, elsker du også <star index="2"/>?
         </li>
         <li>Ja, det overrasker ingen, når man kender <star index="1"/>.
         </li>
         <li>Kan vi ikke snakke om noget andet?
         </li>
      </random>
   </template>
```

Såfremt det givne pattern er blevet matchet, vil robotten vælge et af 5 mulige svar ved at benytte pseudo-tilfældige tal (beregnet ved en standardalgoritme til den slags). Man har ikke mulighed for at påvirke sandsynlighederne, valget er altid uniformt. Hvis man f.eks. vil give præference til »Kan vi ikke snakke om noget andet?« kan man gentage den mulighed; så hvis man f.eks. tilføjer tre kopier mere vil robotten svare »Kan vi ikke snakke om noget andet?« i 50% af tilfældene.

5.4 Matching betinget af bot'ens forrige svar

Normalt er valg af template bestemt ved matching af det tilhørende pattern, men man kan udvide dette ved at betinge på robottens seneste svar. Hvis en kategori kun må komme i anvendelse umiddelbart efter at robotten har svaret med en tekst *t*, angiver man denne som <that>*t*</that> mellem pattern og template.[4] Her er et eksempel, som ville passe naturligt ind i et script, som indeholdt den kategori, der er vist i afsnit 5.3:

[4]Dette tag er taget over fra AIML; der synes ikke at være en god grund til at tag'et for det forrige svar hedder <that>. Det kunne lige så godt havde heddet <børge>, men nu hedder det altså <that>.

```
<category>
   <pattern>* elsker *</pattern>
   <that>Kan vi ikke snakke om noget andet?</that>
   <template>Bliv nu ikke ved med det sladder.
             Fortæl mig i stedet, hvad du lavede i går.
   </template>
</category>
```

Meningen er, at hvis robotten netop har svaret med teksten »Kan vi ikke snakke om noget andet?« og at brugerens input indeholder ordet »elsker« (med noget på hver side), så vil den viste template blive aktiveret, ellers søges efter næste. Som antydet i dette eksempel, kan man bruge det til at simulere at robottens humør skifter gennem dialogen.

En kategori kan have flere `<that>` dele, og så vil kategorien kunne anvende, bare én af disse svarer til bot'ens foregående replik. Det kan ydermere refereres til wildcards inde i `<that> ··· </that>`. Eksempel:

```
<category>
   <pattern>Blabla *</pattern>
   <that>Jeg gider ikke snakke mere om <star/></that>
   <that>Du sladrer hele tiden og så igen</that>
   <template>Hold dog kæft med det snak om <star/>
   </template>
</category>
```

5.5 Databasen og hvordan man bruger den

Når et script kører, har man mulighed for løbende at gemme informationer og hente dem frem igen, og man kan evt. angive noget indhold i databasen fra starten.

5.5.1 Simple variabler i databasen

Disse kan benyttes ved de to tags `<set ···>` og `<get ···>`. Som navn på en variabel kan man vælge hvad som helst.[5] Vi viser dem ved et eksempel. Hvis vi ønsker f.eks. at sætte variablen (eller prædikatet) »`elskerSubjekt`« til »`Klavs Krikke`« kan vi i vores template skrive sådan her.

```
<set name ="elskerSubjekt">Klavs Krikke</set>
```

Når `<set ···>` optræder i en template som aktiveres, vil dets indhold indgå i det svar, som genereres; eksempel følger om et øjeblik. Værdien af variables »`elskerSubjekt`« kan hentes ved følgende konstruktion, mere præcist ved at dens indholdindsættes i det genererede svar.

```
<get name ="elskerSubjekt"/>
```

Det er oplagt at benytte `</star>` i forbindelse med `<set ···>`, som det fremgår af denne kategori:

[5] I AIML omtales sådanne variabler som »predicates«.

```
<category>
   <pattern>* elsker *</pattern>
   <template>Jamen, guuuud, er det sandt at
               <set name="elskerSubjekt"><star index="1"/></set>
            æææælsker
               <set name="elskerObjekt"><star index="2"/></set>??
         Ved du om
               <get name="elskerObjekt"/>
            også æææælsker
               <get name="elskerSubjekt"/>??
   </template>
</category>
```

Dette eksempel viste ikke andet end hvad vi kunne opnå ved brug af `<star ···/>`, men det interessante ligger i at systemet husker variablernes værdier fremover, så andre aktive templates kan benytte dem. Dette er vist i det følgende eksempel.

```
<category>
   <pattern>Har du noget frisk sladder?</pattern>
   <template>Sig det ikke til nogen, men jeg har hørt at
               <get name="elskerSubjekt"/>
            vistnok elsker
               <get name="elskerObjekt"/>
   </template>
</category>
```

De to kategorier kan tilsammen give anledning til denne her dialog.

Human: Klavs Krikke elsker Nora Malkeko
Bot: Jamen, guuuud, er det sandt at Klavs Krikke æææælsker Nora Malkeko?? Ved du om Nora Malkeko også æææælsker Klavs Krikke??
Human: Har du noget frisk sladder?
Bot: Sig det ikke til nogen, men jeg har hørt at Klavs Krikke vistnok elsker Nora Malkeko.

Konstruktionerne `<set ···>` og `<get ···>` kan bruges til at illudere, at robotten husker eller lærer noget mens dialogen kører, men der er kraftige begrænsninger. I eksemplerne ovenfor vil systemet kun huske værdierne »`Klavs Krikke`« og »`Nora Malkeko`« indtil der kommer en ny sætning fra brugeren, som aktiverer samme kategori, f.eks. »`Mary Moseand elsker Joakim von And`«. Har man brug at håndtere den slags, kan man bruge de relationer, som er beskrevet nedenfor.

Der findes en konstruktion `<think>···</think>` som man kan bruge til at skjule dele af output, og som kan være relevant i forbindelse med `<set ···>`, når man ikke er interesseret i at få værdien vist frem i svaret. Eksempel:

```
<category>
   <pattern>* elsker *</pattern>
   <template>Nåh, og hvad så?
       <think>
              <set name="elskerSubjekt"><star index="1"/></set>
              <set name="elskerObjekt"><star index="2"/></set>
       </think>
   </template>
</category>
```

Her vil system give det indtryk, at det ikke er interesseret i oplysningen, men husker den alligevel til senere brug.

5.5.2 Relationer

Udover de simple variabler, kan vi benytte relationer mellem ting. Det kan f.eks. bruges, hvis vi har en kompliceret samtale, som sladrer om mange forskellige personer, som elsker hinanden, kan vi opbevare viden om dette i en relation. Her benyttes `<set2>`, `<get2>` og `<get2rev>`. Eksempel:

```
<category>
<template>
  ...
  <set2 name="elsker">
      <key>Klavs Krikke</key> <value>Nora Malkeko</value>
  </set2>
  ...
</template>
```

Hvis vi senere skal bruge navnet på den, Klavs Krikke elsker, kan vi gøre det sådan her.

```
<category>
<template>
  ...
  <get2 name="elsker">
      Klavs Krikke
  </get2>
  ...
</template>
```

Bemærk, at der *ikke* skal stå `<key>...</key>` her.

I den nuværende udgave af HCML kan der kun huskes én `<value>` til hver `<key>`. Hvis vi udfører den følgende template, vil alt om Klavs Krikkes tidligere relation til Nora Malkeko være totalt glemt.

```
<category>
<template>
  ...
  <set2 name="elsker">
      <key>Klavs Krikke</key> <value>Mystiske Maren</value>
  </set2>
  ...
</template>
```

Den sidste facilitet `<get2rev>` benyttes, når vi kender en `<value>` og ønsker at finde den tilhørende `<key>`. Hvis ikke `elsker`-relationen er blevet opdatere siden det foregående eksempel, vil følgende template give anledning til et svar, som refererer til Klavs Krikke.

```
<category>
<template>
  ...
  <get2rev name="elsker">
      Mystiske Maren
  </get2rev>
  ...
</template>
```

I tilfælde, hvor der er flere, mulige svar, f.eks. hvis der er flere som elsker Mystiske Maren, vil systemet vælge en tilfældig.

5.5.3 At initialisere en database i HCML-scriptet

Simple variabler og relationer kan initialiseres i scriptet, som vis her.

```
<?xml version="1.0"?>
<hcml version="0.0">

  <welcome bot_name="Bla-bla"> Jeg er en rigtig sladrebot</welcome>

  <category>
   ...
  </category>

  ...
  <database>
    <set2 name="elsker">
      <key>Klaus Krikke</key> <value>Nora Malkeko</value>
    </set2>
    <set2 name="elsker">
      <key>Nora Malkeko</key> <value>Klaus Krikke</value>
    </set2>
    <set2 name="elsker">
      <key>Viola Vrinsk</key> <value>Klaus Krikke</value>
    </set2>
    <set2 name="elsker">
      <key>Mary Moseand</key> <value>Joakim von And</value>
    </set2>
  </database>
</hcml>
```

5.6 Rekursiv transformation af input

Med det særlige tag `<srai>`,[6] giver HCML mulighed for at omformulere brugerens input over flere omgange. Her følger et eksempel:

```
<category>
   <pattern>Jeg er vild med *</pattern>
   <template><srai>Jeg elsker <star/></srai>
   </template>
</category>
```

Hvis brugeren afgiver input'et »`Jeg er vild med pandekager`«, vil denne kategori lave det om til »`Jeg elsker pandekager`«, som derefter vil blive forsøgt matchet med de øvrige patterns i vores scripts. Processen vil gentage sig, hvis nu det nye pattern, som matches også indeholder `<srai>`. Vi kan f.eks. tilføje denne kategori:

[6] Betegnelsen »srai« er overtaget fra AIML; hvad der er en forkortelse for, er i skrivende stund en gåde.

```
<category>
   <pattern>Undertegnede *</pattern>
   <template><srai>Jeg<star/></srai>
   </template>
</category>
```

Hvis brugeren nu afgiver input'et »`Undertegnede er vild med pandekager`«, bliver det først lavet om til »`Jeg er vild med pandekager`« og derefter »`Jeg elsker pandekager`«, og processen fortsætter (og rammer nok en af de kategorier, vi har om »elsker«).

Fordelen ved at bruge `<srai>` er, at vi kan

- få vores robot til at genkende et stort udvalg af alternative måder at formulere samme udsagn på, og
- de kategorier som indeholder den egentlige behandling af et udsagn kan genbruges, dvs. den måde, de endelige svar genereres på skal kun skrives én gang..

Vær opmærksom på, at en regel som den for »`undertegnede`« ovenfor ikke fungere som en generel oversætter af ordet til »`jeg`«. Dertil behøver vi tre kategorier, som indeholde følgende patterns.

```
<pattern>* undertegnede</pattern>

<pattern>undertegnede *</pattern>

<pattern>* undertegnede *</pattern>
```

Man kan nå langt med brug af `<srai>`, men metoden er ikke skudsikker, da den ikke indholder grammatikregler men kun flade mønstre. Ved match med de viste patterns vil man f.eks. ikke kunne afgøre om »`undertegnede`« nu skal oversættes til »`jeg`« eller »`mig`«.

Det er muligt at benytte flere `<srai>` flere gange i samme template.

```
<category>
<template>
   bla bla
   <srai>blop <star index="1"/></srai>
   <srai>blip <star index="2"/></srai>
   mere bla bla
</template>
```

I dette eksempel bliver svaret af formet »`bla bla ... mere bla bla`«, hvor prikkerne står for sammensætningen af, hvad de to `srai`'er hver især kommer tilbage med.

Man kan også bruge princippet med rekursiv transformation til at afbilde teksterne over i fiktive udtryksformer, som ikke har noget med naturligt sprog at gøre, og som man gætter på at en bruger aldrig kunne finde på at skrive selv. Her er et meget enkelt eksempel.

```
<category>
   <pattern>Jeg hedder *</pattern>
   <template><srai>F117NAVN<star/></srai>
   </template>
</category>

<category>
   <pattern>Mit navn er *</pattern>
   <template><srai>F117NAVN<star/></srai>
   </template>
</category>

<category>
   <pattern>De kalder mig *</pattern>
   <template><srai>F117NAVN<star/></srai>
   </template>
</category>

<category>
   <pattern>F117NAVN *</pattern>
   <template>Hej <set name="brugernavn"><star/></set>!
   </template>
</category>
```

Hvis man har et meget komplekst script med mange sproglige variationer, kan dette være en metode til at strukturere det på.

Advarsel: Det er muligt med `srai` at konstruere scripts, som kan gå i uendelig løkke. HCML-systemet vil gog stoppe dem automatisk efter 77 iterationer og give en fejlmeddelelse.

5.7 Betingede udtryk i templates

Der findes forskellige måder, man kan lade det genererede svar afhænge af værdier af prædikater. Her er et eksempel, som burde være mere eller mindre selvforklarende.

```
<category>
   <pattern>Lad os tale om *</pattern>
   <template><think><set name="emne"><star/></set></think>
       Nå ja, ham (hende) der siger
       <condition name = "emne">
          <li  value="Klavs Krikke"> vrinsk!</li>
          <li  value="Nora Malkeko"> muh muh!</li>
          <li  value="Anders And"> rapraprapraprapperaaaap!</li>
          <li  value="defaultListItem"> blabla!</li>
       </condition>
    </template>
</category>
```

Det kan eksempelvis give anledning til denne dialog:

Human: Lad os tale om Anders And
Bot: Nå ja, ham (hende) der siger rapraprapraprapperaaaap!
Human: Lad os tale om Fedtmule
Bot: Nå ja, ham (hende) der siger blabla!

Eksemplet kan udvides, så teksten »`ham (hende)`« kun kommer frem, når vi rammer ned i sidste alternativ (»`defaultListItem`«), og at der for »`Anders And`« og »`Klavs Krikke`« skrives »`ham`« og »`hende`« for »`Nora Malkeko`«.

Der findes ikke en særskilt notation for at referere til relationer i betingede udtryk, så her man man have besværet først at hale informationen ud af relationen og putte den over i en simpel variabel.

Litteratur

[1] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd edition*. Pearson International Edition, 2009.

[2] Richard S. Wallace. Aiml overview. `http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html`.

[3] Joseph Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966.