DEPARTMENT OF TELEINFORMATICS

Master Thesis Project

Author:

Andrew A. Warrengton
andrew.warrengton@ewi.ericsson.se
E94_ada@e.kth.se

Oswaldo Perdomo
oswaldo@erlang.ericsson.se
E94_ope@e.kth.se

1999-10-25
ver. 3.0

Innehållsförteckning

Oswaldo Perdomo
e94_ope@e.kth.se

Andrew A. Warrengton
e94_ada@e.kth.se

Abstract

*This document contains the outcome of the final version of our thesis project: **Contact Broker System**.*

This Master Thesis was done at Ericsson Radio Systems AB, in Stockholm, Sweden. The examination was reported to KTH, Department of Teleinformatics.

The rapid advances of computer and communications technologies are causing an increased demand for new services where a user requires not only the exchange of data but they need also a support system too.

*The purpose of this master's thesis was to find and demonstrate a system solution for a **Contact Broker System** (CBS), to support the user to find persons that are responsible for a certain function in an organisation or search domain and provide the proper means of contacting the person.*

This work consists of two main parts. In the first part we wrote about the theory, which was relevant for our thesis and in the second part we introduced a system solution for "CBS", which has a search engine with a database. Search engine is implemented in Erlang and database is Mnesia an OTP application (Open Telecom Platform).

Oswaldo Perdomo
e94_ope@e.kth.se

Andrew A. Warrengton
e94_ada@e.kth.se

1 Inledning

1.1 Förord

Stockholm tar ledningen i Internetanvändandet. Enligt färsk statistik från FSI är antalet användare (maj 1999) uppe i 34 procent, jämfört med siffran för övriga Sverige som är 27 procent. Enligt marknadsinstitutet *Nielsen*, är motsvarande siffra för USA och Kanada 23 procent. Utnyttjandet av Internet är dubbelt så högt i Sverige jämfört med det europeiska genomsnittet. Det affärsrelaterade användandet överträffar både det akademiska och det fritidsrelaterade nyttjandet.

Användandet av elektronisk kommunikation, som t ex Internet och mobiltelefoni, är överraskande högt i Stockholm, i vissa fall det högsta i världen. Enligt Forskningsgruppen för Samhälls- och Informationsstudier (FSI) använder t ex 48 procent av stockholmarna mellan 16-79 år regelbundet mobiltelefoner och 34 procent använder regelbundet Internet för *surfing* och/eller elektronisk post.

Stockholms mycket utvecklade IT- och kommunikationsstruktur placerar Sverige på toppen av IT-listorna. I en nyligen genomförd undersökning som introducerades i år av *International Data Corporation* (IDC), och *World Times Inc*. där 55 länder ingick i undersökningsmaterialet, kom Sverige på andra plats efter USA. Indexet är baserat på ett flertal faktorer som är relaterade till såväl sociala strukturer som till informations- och datainfrastrukturer. Dessa faktorer representerar tillsammans ett lands IT-mognad.[21]

En förutsättning för att Sverige skall fortsätta med sin ledande position inom IT är att tekniken måste förädlas och justeras i takt med tiden. Nya tjänster ska komplettera de gamla och förbättringar skall ge bättre stöd till användaren oavsett vilka sorters apparater man använder.

Ett av de ledande företagen inom forskning och systemlösningar är Ericsson koncernen. Vårt examensarbete genomfördes vid Ericsson Radio Systems AB i Kista, Stockholm. Examinationen utfördes i IT-sektionen vid Kungliga Tekniska Högskolan, KTH.

Vårt examensarbete bestod av att hitta/ge förslag till ett system som ska ge stöd till användaren vid sökning av en specifik tjänst. I första skedet skulle man försöka hitta systemlösningen för fasta nätet och i andra momentet skulle man kunna ha tjänsten för den trådlösa nätet.

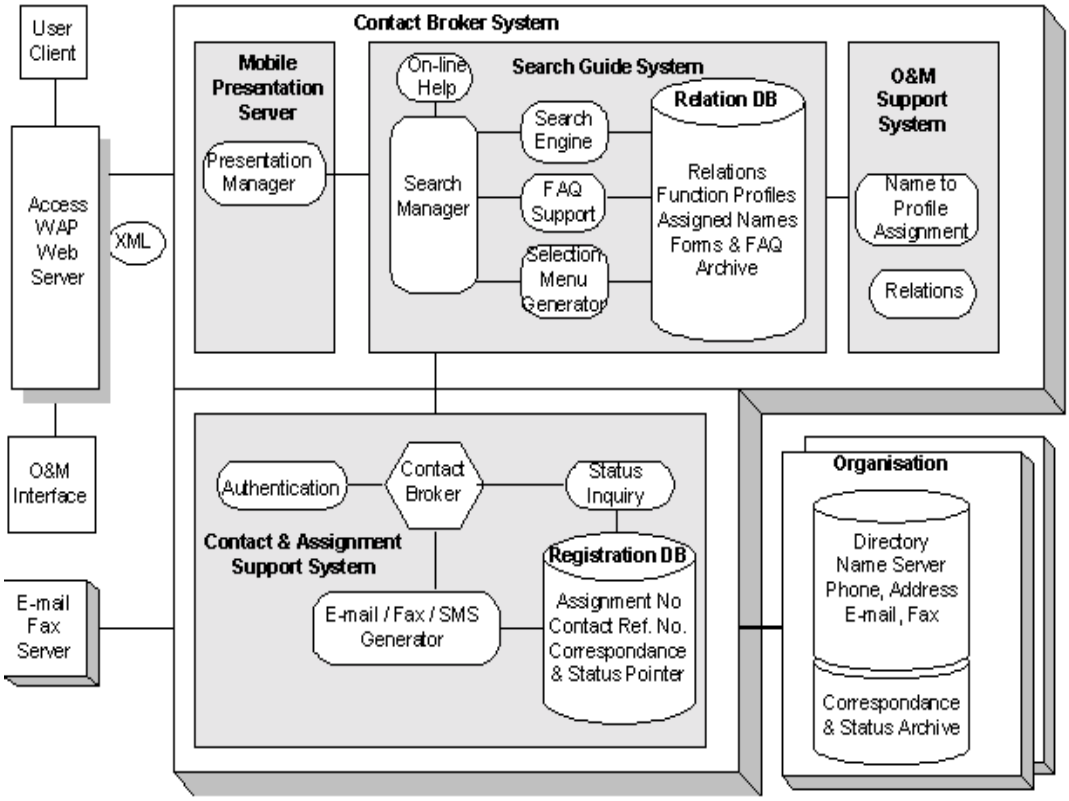
Rapporten innehåller två huvudmoment. I den första delen går vi igenom den teoretiska, för att senare kunna använda den för vår implementering i andra delen.

Oswaldo Perdomo
e94_ope@e.kth.se

Andrew A. Warrengton
e94_ada@e.kth.se

1.2 System arkitektur för CBS

Syftet med detta examensarbete var att utveckla en systemlösning för ett förmedlarsystem , ”Contact Broker System”. CBS:n skall göra det möjligt för en användare att kunna söka efter den person som ansvarar för en viss funktion (tjänst, osv.) inom en organisation. Vid sökning ska systemet föreslå en eller flera möjliga vägar att gå för att hitta den ansvariga personen för den sökta funktionen. När funktionen har hittats, ska systemet skicka ett formulär till användaren, vilken i sin tur fyller i formulären och skickar tillbaka dem till systemet. Systemet tilldelar arbetet till dem ansvarige person samtidigt som användaren informeras om detta och får ett referensnummer för uppföljning av ärendet. Se figur.1 här nedan.



Figur.1

Oswaldo Perdomo
e94_ope@e.kth.se
Andrew A. Warrengton
e94_ada@e.kth.se

Systemet skall byggas i tre delblock:

1. ***Search Engine,*** för fritext-sökning, som skall kunna ge stöd till mobilterminaler (mobiltelefoner) med begränsad textvisning. Denna del ingår i delblocket *Search Guide System* i figur.1.

2. ***Assignment Support System,*** som skall tilldela funktionsarbetet till en kontaktperson inom organisationen. Detta delblock presenteras i figur.1 med namnet *Contact & Assignment Support System*.

3. ***Contact Support System,*** som skall informera den valda kontaktpersonen genom att skicka E-mail eller fax med det ifyllda formuläret bifogat. Detta delblock presenteras i figur.1 med namnet *Contact & Assignment Support System.*

1.2.1 Avgränsningar

De delar som vi har inriktat oss på är själva sökmaskinen och att lägga upp en databas för att sökandet. Senare ska man kunna underhålla systemets databas med att lägga till nya tjänster eller ta bort de tjänster som inte är längre aktuella. Resultatet kan presenteras både i en web-browser och en mobiltelefondisplay. Se figur.1, det övre delblocket.

1.3 Tidigare arbete

Inom området systemlösning för olika tjänster i Internet är forskningen i full gång och nya avancerade sökmaskiner för fasta nätet är redan igång men någon fullständig sökmaskin för mobiltelefoner finns inte än.

Det arbete som ligger närmast till hands och är det senaste inom utvecklingen av Internet-användandet för fasta nätet, är ett projekt som heter *Stockholm business partner search* och detta projekt ingick i en global satsning som heter *The Global Bangemann Challenge* som sponsrades av både Stockholms stad och EU. Detta projekt har skapat en internetbaserad söktjänst där utländska företag kan söka affärspartners i Stockholmsregionen. Detta arbete fick ett pris under våren 1999.[22]

Det kan finnas sökmaskiner som är under utveckling och är baserade på XML men vi har inte hittat några relevanta sådana sökmaskiner. Däremot finns det inte någon sökmaskin för WAP-telefoner. Det kan finnas WAP-sökmaskiner under utveckling i andra företag men än så länge är resultatet inte känt för allmänheten.[23]

Skillnaden mellan vår sökmaskin och andra sökmaskiner är att man kan presentera sökresultatet på en mobiltelefon-display (sökning baserad på XML som senare översätts till WML) och att vi har använt OTP-plattformen som bas.

1.4 Metodval

För att genomföra det här examensarbetet har information från följande källor använts:

- Facklitteratur
- Internet
- Kurs i programmering

Den lösningsmetod som använts för att utföra examensarbetet har varit följande:

Först så har ämnesområdet överblickats genom insamling av information från ovanstående informationskällor. Därefter har informationen genomlästs och använts till att implementera systemet.

2 Erlang, systemarkitektur och systemprinciper

2.1 Introduktion

Erlang är ett funktionellt programmeringsspråk, konstruerat för att bygga feltoleranta distribuerade system. Erlang kan ta hand om flera processer som körs samtidigt, d.v.s. parallella processer. Erlang har dessutom andra konstruktioner som stödjer distribuerade system och feldetektering.

Ett program skrivet i Erlang är uppbyggt av funktioner som är uppdelade i moduler. Funktioner startar processer som är de verkställande elementen i ett Erlangsystem. Processer kommunicerar genom att skicka och ta emot meddelande via portar, vilka själva uppför sig som processer. En inbyggd distributionsmekanism möjliggör för designern att skapa ett system vars processer kan köras på olika datorer. Erlang hanterar feldetektering och återhämtning i distribuerade system och har felåterhämtningsschema[2].

Erlang har en del karaktäristiska egenskaper[1], som:

- **Deklarativ syntax:** Erlang har en deklarativ syntax och är i stort sett fritt från sidoeffekter.

- **Samverkan:** Erlang har en processbaserad modell med asynkron meddelandepassning. Processerna i Erlang är *ligth-weigth,* d.v.s. processerna kräver lite minne. Att skapa eller ta bort en process samt att skicka meddelande mellan processer kräver inte så mycket prestanda.

- **Realtid:** Erlang är tänkt att användas i system med realtidprogrammering där svarstiden har en storleksordning av millisekunder.

- **Robust:** Säkerheten är ett kritiskt krav i ett system. Det finns tre konstruktioner i Erlang för detektering av *run-time error*. De kan användas för att programmera robusta applikationer.

- **Minneshantering:** Erlang är ett symboliskt programmeringsspråk med *garbage collection* i realtid (återvinner minne som inte längre kan åtkommas). Minne allokeras automatiskt när det behövs och tas bort när det inte används längre. De typiska programmeringsfel som är associerade med minneshantering kan inte uppkomma.

- **Distribution:** Erlang har inte något delat minne. Allt samspel mellan processer sker genom asynkron meddelandesändning. Det går enkelt att bygga ett distribuerat system i Erlang. Applikationer som är skrivna för en ensam processor kan utan några större svårigheter skickas till ett nätverk av flera processorer.

- **Integration:** Erlang kan använda program som är skrivna i andra programmeringsspråk. Främmande program kan m.h.a. ett interface, uppträda som ett program skrivet i Erlang för en programmerare, m.a.o. andra program anropas som vanliga Erlangprogram.

- **Kontinuerlig funktion:** Erlang tillåter att programkoden byts ut i ett system som körs och tillåter att nya och gamla versioner av programkoden körs samtidigt. Dessa är en klar fördel i ett *non-stop*-system.

Utvecklingsverktygen inkluderar flera möjligheter att skapa interface mellan applikationer som är skrivna i Erlang och andra programmeringsspråk. Dessa ingår:

- En *open-port* mekanism
- Ett *socket* bibliotek
- En C/C++ interface generator
- Ett *erl_interface* bibliotek som inkluderar funktioner för att skapa C-strukturer som beter sig som en Erlang nod.

Utvecklingsverktygen innehåller ett antal program för att utveckla och testa applikationer. Följande verktyg ingår:

- Editor
- Compiler
- Debugger
- C interface generator
- Application monitor
- Process manager
- Coverage analyzer
- Profiler
- Graphics interface
- ASN.1 compiler
- Cross-reference tool
- Parser generator

2.2 Erlangs applikationer

Ett Erlangsystem är uppbyggt av ett antal applikationer. Två av dessa applikationer är obligatoriska som heter *Kernel* och *stdlib*. Samtidigt finns det också ett antal valbara applikationer. Detta innebär att programmeraren kan välja de applikationer som är bäst lämpade för ett specifikt projekt och samtidigt förkasta de övriga[2]. Följande applikationer finns med i en distribution av ett Erlangsystem:

- Kernel
- Standard bibliotek (stdlib)
- Socket
- C/C++ interface generator (IG)
- Grafiskt system (GS)
- Erl_interface bibliotek
- Java interface (Jive)

I ett litet system körs bara två applikationer, de två obligatoriska, alltså *kernel* och *stdlib*.

2.2.1 Kernel (kärnan)

Kernel är alltid den första applikationen som startas. Den står för den låg-nivåservice som är nödvändig för att ett Erlangsystem ska kunna:

- Startas
- Delta i ett distribuerat system
- Handskas med felhantering
- Utföra IO-operationer

2.2.2 Standardbibliotek (stdlib)

Ett standardbibliotek innehåller ett stort antal återanvändningsbara moduler. Många av de här modulerna är speciellt anpassade för att kunna programmera flera parallella distribuerade system. Det finns moduler som kan lösa vanliga programmeringsuppgifter, till exempel *gen_server, gen_event* och *gen_fsm*.

2.2.3 Socket

Denna applikation innehåller två tjänster:

- Socket
- UDP (User Datagram Protocol)

2.2.4 C/C++ interface generator (IG)

Interfacegeneratorn är ett verktyg som används för att skapa ett interface mellan C och Erlang. Med interfacegeneratorn är det möjligt att transparent kunna nå C-språket, d.v.s. C-funktioner liknar Erlangsfunktioner på Erlang-sidan och en Erlangsfunktion ser ut som en C-funktion på C-sidan.

2.2.5 Erl_interface biblioteket

Erl_interface biblioteket innehåller funktioner som används för att kunna skapa ett interface mellan Erlang och andra programmeringsspråk. Särskilt innehåller det funktioner som stödjer kodningen av en datastruktur i Erlang till en sekvens av bytes, och avkodningen av en sekvens av bytes, från en datastruktur i Erlang till en *struct* i C.

2.2.6 Java-interface (Jive)

Java-interface möjliggör för en java-applet/applikation att kommunicera med en Erlangserver. Java är lämpligt för gränssnittet på klient-sidan, samtidigt som Erlang är lämpligt för programmeringen på server-sidan. Idén bakom *Jive* är att kunna integrera de här två programmerinsspråken. En java-applet/applikation och en Erlangserver kan samverka med hjälp av *Jive*.

2.2.7 Grafiskt system (GS)

Ett grafiskt system tillåter en användare att skapa grafiska objekt. Detta system fungerar på alla plattformar där Erlang är implementerat.

2.3 Erlangs ”Run-time system”

Erlangs *Run-time* system är uppbyggt av följande delar:

- Erlangs virtuella maskin
- Kernel
- Standardbibliotek

2.3.1 Erlangs virtuella maskin

Erlangs virtuella maskin körs på toppen av ett operativsystem. Erlangs *Run-time* system körs oftast som en enkel process i ett operativsystem. Dock, i de operativsystem som stödjer Erlang, är det möjligt att parallellt köra flera helt oberoende virtuella maskiner.

Följande stöd finns för Erlang program:

- Ett konsistent operativsysteminterface på alla plattformar.
- Minnesallokering och *garbage collection* i realtid.
- *Ligth-weigth*-samverkan och stöd för många simultana uppgifter.
- Transparent samarbete mellan alla datorer i systemet.
- Lokalisering av *run-time errors*.
- Övervakning av *run-time* kod när den laddas, när den flyttas och medan den länkas.

2.4 Kodladdningsstrategi

Programmeringskod laddas alltid relaterad till den aktuella sökvägen. Sökvägen hämtas från värdet som är angivet i *script*-filen (exempel: *start.script*), möjligen modifierad genom *path* i kommandoraden. Detta tillåter oss att köra ett system på flera olika sätt[4]:

- ***Interactive mode*:** Systemet laddar programkoden dynamiskt från den katalog som är specificerad i *path*-kommandot. Detta är det normala sättet att köra programmen.

- ***Embedded mode*:** Systemet laddar all sin programkod under dess uppstart. I speciella fall kan all programkod placeras i en separat katalog. Man skulle kunna kopiera alla filer i en viss katalog och därefter skapa en sökväg till denna katalog.

- ***Test mode*:** Test mode används normalt om man vill köra en ny testkod med en specifik utgåva av ett *embedded* system. Man vill ha alla bekvämligheter med ett interaktivt system med kodladdning på begäran och robustheten av ett *embedded* system.

2.5 Överblick av designprinciper i ett Erlangsystem

2.5.1 Introduktion

System eller kompletta produkter är uppbyggda av ett antal applikationer. Applikationer är designade för att vara svagt sammanfogade (*weakly coupled*) och det är ofta möjligt att bygga ett system genom att kombinera existerande applikationer med egna applikationer. Med andra ord kan vi sammanfoga en applikation avsedd för ett speciellt syfte med redan existerande applikationer.

Nya applikationer bör vara designade på så sätt att de är tillgängliga som fristående applikationer. Med denna princip kan man lägga nya applikationer till de existerande grundapplikationerna för att senare kunna använda dem i systemet.

Ett bra exempel på en redan existerande applikation är Mnesia. Mnesia har allt som krävs för att programmera en databasservice. Vidare kan man nämna ”Grafiskt System” för att bygga grafiska användarinterface.

Applikationer är specificerade med benämningen *resources*. *Resource* inkluderar moduler, registrerade namn, processer och saker som är beroende av andra applikationer.

Applikationer måste följa vissa regler och protokoll för att kunna presentera ett konsistent interface till Erlangsystem. De måste till exempel vara skrivna på så sätt, att man kan byta ut progamkod utan att stoppa systemet[3].

2.5.2 Behaviour

Det enklaste sättet att programmera en ny applikation är att använda sig av ett s.k. *behaviour*, som är inkluderat i systemet. *Behaviour* är ett slags formulering av ett konstruktionsmönster (*design pattern*) som kan

användas för att programmera vissa vanliga problem. Applikationer som är programmerade med ett *standard behaviour* följer automatiskt det nödvändiga protokollet.

Konkreta system kan programmeras genom att kombinera idéer och koden från ett flertal små *design patterns*.

Applikationer och processer som är implementerade med *behaviour* kommer att:

- Dela samma *behaviour*.
- Bli likformigt kontrollerade.

Principen är att dela upp programkoden till två moduler för en applikation/process:

- En *behaviour*-modul med den generiska delen av programkoden.
- En *callback*-modul med den specifika delen. Denna modul är implementerad av programmeraren.

Det finns ett antal *standard behaviour*, som är tillgängliga i standard biblioteket. De viktigaste är följande :

- ***Application*:** Denna *behaviour* definierar hur applikationen är implementerad och avslutad.

- ***Sup_bridge*:** Denna *behaviour* sammankopplar en process eller ett *subsystem*, till ett övervakarträd, även om den inte skapades med en övervakningsprincip i tankarna.

- ***Supervisor*:** Denna *behaviour* är en arbetar-/övervakarmodell för strukturering av feltolerant beräkningar och programmering av ett övervakarträd.

- ***Gen_server***: Denna *behaviour* används för att programmera klient-server processer.

- ***Gen_event*:** Denna *behaviour* används för programmering av *event-handling* mekanismer.

- ***Gen_fsm*:** Denna *behaviour* används för programmering av ändliga statusmaskiner.

Om en applikation är skriven utan hjälp av en *behaviour-modul*, bör programmeraren vara säker på att denna applikation följer de nödvändiga protokollkraven.

De två följande modulerna är till för att hjälpa programapplikationer som inte använder sig av *standard behaviour*.

- ***Sys*:** Denna modul tillhandahåller en uppsättning bibliotekfunktioner som följer ett standardsystemprotokoll. Funktioner i *Sys* kan användas till att skapa ett interface mellan en process och resterande delar av systemet.

- ***Supervisor_bridge*:** Denna modul möjliggör användandet av redan existerande processer med ett övervakarträd.

Både *sys* och *supervisor_bridge* är avsedda för mera speciella ändamål och kräver detaljerad kunskap om Erlangsystem. När den skrivna programkoden anropas genom en *standard behaviour*, kan en programmerare anropa funktioner från ett standardbibliotek. Biblioteket skaffar en innehållsrik och växande uppsättning av moduler som innehåller de mest använda standardfunktionerna.

Behaviour är implementerade som *callback*-moduler.
En *callback*-modul måste exportera ett antal specifika funktioner som senare anropas av systemet, när en *behavior*-process utförs.

Alla moduler som använder sig av *behaviour*, måste startas på följande sätt:

-module (xx).
-behaviour (yy).

Detta betyder att modul ”xx” har uppförandet ”yy”.
I följande deklaration har modulen *disk_alloc* uppförandet *gen_server*.

-module (disk_alloc).
-behaviour (gen_server).

Det är möjligt att skriva program som inte är baserade på applikation- och *behaviour*-mekanismerna. Detta kan ge effektivare program, dock på bekostnad av robustheten.

Programmerare kommer att finna att det är lättare att läsa och förstå programkod som är skriven av andra programmerare om de känner till applikationsarkitekturen och *standard behaviour*[3].

2.5.2.1 Exempel på behaviour

Behaviour är en slags automatik av konstruktionsmönster. Många applikationer följer ett liknande mönster i sin programstruktur. Detta medför att vissa applikationer kan implementeras m.h.a. en sorts program mall. Sådana program mallar finns för att programmera ett flertal olika applikationer, exempelvis tillståndsmaskiner och servrar.

Vi ska nedan studera *behaviour* för en *gen_server*. En *gen_server* erbjuder ett standardsätt att programmera en klient-server-applikation. Den är uppdelad i två delar, varav den ena är den generiska delen och den andra är den specifika delen.
Den generiska delen av servern innehåller funktioner för *debugging*, för hantering av avslutning av föräldraprocessen och för att presentera information om något fel med servern uppstår.
Den specifika delen av servern finns i en modul som kallas *callback*-modul. *Callback*-modulen innehåller gränssnitt mot klient-delen och mot serverns *callback*-funktioner (exempel: *handle_call/3*, se här nedan).
När den generiska delen av servern tar emot ett meddelande, anropas respektive callback-funktion[3].

Relationen mellan de generiska funktionerna och *callback*-funktionerna kan illustreras på följande sätt:

Callback-modul *gen_server*

gen_server:start ---------> startar en ny server
Module:init/1 <---------
Loopar

gen_server:call --------->
Module:handle_call/3 <-----

gen_server:cast --------->
Module:handle_cast/2 <------

gen_server:multi_call ------->
Module:handle_call/3 <-------

Module:handle_info/2 <----- andra mottagna meddelanden.

Module:terminate/2 <----- städa upp innan avslutningen.

Följande systemhändelser produceras av ett *gen_server behaviour*, vilka i sin tur hanteras av *sys*-modulen:

- {in, Msg}, när ett meddelande tas emot.
- {out, Msg, To}, när ett meddelande har skickats
- {noreply, State}, när man inte får något svar.

Nedan följer ett exempel på en kö-server. Servern har fyra interface-funktioner mot klientdelen, nämligen:

- *start/0*, som startar serverprocessen och registrerar den med namnet *queue*.
- *stop/0*, avslutar serverprocessen.
- *in/1*, som stoppar in ett element sist i kön.
- *out/0*, som tar bort första elementet i kön.

```
%%%----------------------------------------------------------------------
%%% File    : queue.erl
%%% Author  : we
%%% Purpose : exercise
%%% Created : 26 Apr 1999
%%%----------------------------------------------------------------------

-module(queue).

-behaviour(gen_server).

%% External exports
-export([start/0, in/1, out/0, stop/0]).

%% gen_server callbacks
-export([init/1, handle_call/3, handle_cast/2, handle_info/2,
terminate/2]).


%%%----------------------------------------------------------------------
%%% API
%%%----------------------------------------------------------------------

start() ->
   gen_server:start_link({local, queue}, queue, [], []).

in(Item) -> gen_server:call(queue, {in, Item}).

out() -> gen_server:call(queue, out).

stop() -> gen_server:call(queue, stop).


%%%----------------------------------------------------------------------
%%% Callback functions from gen_server
%%%----------------------------------------------------------------------

%%----------------------------------------------------------------------
%% Func: init/1
%% Returns: {ok, State}          |
```

```
%%       {ok, State, Timeout} |
%%       ignore               |
%%       {stop, Reason}
%%----------------------------------------------------------------------

init([]) ->
   {ok, {[],[]}}.


%%----------------------------------------------------------------------
%% Func: handle_call/3
%% Returns: {reply, Reply, State}          |
%%       {reply, Reply, State, Timeout} |
%%       {noreply, State}                |
%%       {noreply, State, Timeout}       |
%%       {stop, Reason, Reply, State}    | (terminate/2 is called)
%%       {stop, Reason, State}           (terminate/2 is called)
%%----------------------------------------------------------------------

handle_call({in, X}, From, {In, Out}) ->
   Reply = ok,
   {reply, Reply, {[X|In], Out}};

handle_call(out, From, Queue) ->
   {Reply, NewQueue} = out(Queue),
   {reply, Reply, NewQueue};

handle_call(stop, From, Queue) ->
   {stop, normal, ok, Queue}.

%%----------------------------------------------------------------------
%% Func: handle_cast/2
%% Returns: {noreply, State}          |
%%       {noreply, State, Timeout} |
%%       {stop, Reason, State}           (terminate/2 is called)
%%----------------------------------------------------------------------

handle_cast(Msg, State) ->
   {noreply, State}.
```

```
%%--------------------------------------------------------------------
%% Func: handle_info/2
%% Returns: {noreply, State}          |
%%          {noreply, State, Timeout} |
%%          {stop, Reason, State}            (terminate/2 is called)
%%--------------------------------------------------------------------

handle_info(Info, State) ->
   {noreply, State}.

%%--------------------------------------------------------------------
%% Func: terminate/2
%% Purpose: Shutdown the server
%% Returns: any (ignored by gen_server)
%%--------------------------------------------------------------------

terminate(Reason, State) ->
   ok.

%%%------------------------------------------------------------------
%%% Internal functions
%%%------------------------------------------------------------------

out({In, [H|Out]}) ->
   {{value, H}, {In, Out}};

out({[], []}) ->
   {empty, {[],[]}};

out({In, _}) ->
   out({[], lists:reverse(In)}).
```

Denna modul, queue-modulen, får en gen_servers behaviour genom raden:

```
        -behaviour(gen_server).
```

Detta innebär, bland annat, att *callback*-funktionerna från *gen_server* kan tas emot i den här modulen. När man anropar funktionen *”in (Item)”*, så anropar man egentligen *callback*-funktionen *”handle_call ({in, X}, From, {In, Out})”*. På liknande sätt anropas *”handle_call (out, From, Queue)”*

genom funktionen *”out ( )”*. *Callback*-funktionen *”handle_call (stop, From, Queue)”* anropas m.h.a. funktionen *”stop ( )”* och den i sin tur anropar *callback*-funktionen *”terminate (Reason, State)”*, som rensar upp minnet och avslutar serverprocessen.

En körning av kö-servern kan se ut på följande sätt:

```
1> queue:start().
{ok,<0.30.0>}
2> queue:in(oswaldo).
ok
3> queue:in(perdomo).
ok
4> queue:in(andrew).
ok
5> queue:in(warrengton).
ok
6> queue:in(1).
ok
7> queue:in(2).
ok
8> queue:out().
{value,oswaldo}
9> queue:out().
{value,perdomo}
10> queue:out().
{value,andrew}
11> queue:out().
{value,warrengton}
12> queue:out().
{value,1}
13> queue:out().
{value,2}
14> queue:out().
empty
15> queue:stop().
ok
```

2.5.3 Övervakningsträd

Det vanligaste sättet att programmera en Erlang-applikation är att starta med ett övervakningsträd. Ett övervakningsträd är ett hierarkiskt träd av processer som används för att programmera ett feltolerant system.

Högre noder i ett övervakningsträd kallas för ”Övervakare” *(supervisor)*. Övervakare kontrollerar noder i lägre nivå, vilka kallas för ”Arbetare” *(worker)*och upptäcker när ett fel uppstår i en lägre nod. En övervakare kan övervaka både en arbetare eller en annan övervakare.

Arbetarnoder utför beräkningarna. De utför själva arbetet och Övervakare kontrollerar status och startar om en Arbetarnod, om någonting har gått fel. Denna övervakningsprincip låter oss designa och programmera feltoleranta program. Man kan skapa ett övervakarträd genom att använda sig av ett *supervisor behaviour*[3].

2.5.3.1 Exempel på ett övervakarträd behaviour

Ett övervakarträd skapas från en startspecifikation. Modulen *xx* är ett exempel på *supervision behaviour*. Funktionen *”xx:init(Args)”* måste vara skriven på så sätt att den antingen ska returnera *{ok, SuperSpec}* eller *{error, what}*. Ett övervakarträd skapas genom anropet:

Supervisor:start_link(Name, xx, Args).

I funktionsanropet ovan, är *”Name”* namnet på övervakarträdet och *”Args”* ett argument som ges till *xx:init*.

```
-module(xx).
-behaviour(supervisor).
-export([init/1]).
```

```
init(_) ->
   {ok,{{one_for_all, MaxR, MaxT},
       [
        {log,  {log, start, [25]}, permanent, 5000,   worker,   [log]},
        {mysup, {mysup, start, []}, permanent, infinity, supervisor,
[mysup]},

        ...

        {ChildName, MFA,        Type,    Shutdown, Class,
Modules}
       ]}}.
```

3 OTP (Open Telecom Platform) applikationer

3.1 Introduktion

OTP är en utvecklingssystemsplattform för att bygga telekommunikationsapplikationer och ett kontrollsystem för att kunna köra dessa applikationer.

Det finns ett antal applikationer som ingår i OTP men de applikationer som är väsentliga för vårt arbete och som tas upp i kommande avsnitt är:

- *Mnesia*
- *Mnesia Session*
- *Mnemosyne*

3.2 Mnesia

3.2.1 Introduktion

Mnesia är ett distribuerat *DataBase Management System* (DBMS), lämpligt för telekommunikationsapplikationer och andra Erlangapplikationer som kräver kontinuerliga funktioner och uppvisningar av *soft*-realtid utrustningar[6].

Mnesia är skrivet i Erlang och är tänkt att användas tillsammans med Erlangapplikationer.

3.2.2 Mnesia DataBase Management System, (DBMS)

3.2.2.1 Kännetecken

Mnesia innehåller följande kännetecken, vilka kombineras för att producera ett feltolerant, distribuerat databassystem skrivet i Erlang:

- Databasschemat kan dynamiskt omkonfigureras under *run-time*.
- Tabeller kan deklareras på så sätt att de har *location, replication* och *persistence.*
- Tabeller kan flyttas eller kopieras till flera noder för att förbättra feltoleransen.
- Tabell-lokalisering är transparent för programmeraren. Programmen adresserar ett tabellnamn och systemet håller själv reda på var den befinner sig.

- Databastransaktioner kan vara distribuerade. Ett stort antal av funktionerna kan anropas från en transaktion.

- Flera transaktioner kan köras samtidigt och de kan utföras fullt synkroniserade av ett databassystem. Mnesia garanterar att det aldrig förekommer två processer som manipulerar data samtidigt.

- Transaktioner kan ha egenskapen att kunna utföras på alla noder i ett system eller på inga noder alls.

3.2.2.2 Hjälpapplikationer

Mnemosyne och Mnesia Session används i samband med Mnesia för att producera speciella funktioner, vilka förbättrar tillämpningen av Mnesia.

Mnesia har två *query*-interface:

- *Mnemosyne*
- *Query list comprehensions*, som är ett Erlang språk konstruerat för databas-förfrågningar.

Nedan beskrivs de huvudsakliga finesserna med Mnesia Session och Mnemosyne i samband med användandet av Mnesia:

- ***Mnemosyne*** är ett *query*-språk, som kan vara kombinerat med Mnesia för att kunna uttrycka komplexa relationer mellan tabeller och tabelldefinitionsvyer.

- ***Mnemosyne*** har möjligheten att optimera *query*-kompilatorn för Mnesia DBMS som huvudsakligen gör DBMS mera effektiv.

- ***Mnemosyne*** kan användas som ett programmeringsspråk för Mnesia. Den inkluderar ett beteckningssätt, s.k. *list comprehensions* och kan användas till att göra komplexa databasförfrågningar över en mängd av tabeller.

- ***Mnesia Session*** är ett interface för Mnesia DBMS.

- ***Mnesia Session*** möjliggör tillgången till Mnesia DBMS från ett annat språk än Erlang.

3.2.2.3 När ska man använda sig av Mnesia

Vi kan använda Mnesia med följande typer av applikationer:

- Applikationer där man behöver datakopiering.

- Applikationer där man utför komplicerad sökning av data.

- Applikationer där man behöver använda atomiska transaktioner för att uppdatera flera arkiv samtidigt.

- Applikationer där man använder mjuk realtids-karaktäristik.

3.2.2.4 När ska man inte använda sig av Mnesia

Mnesia är inte riktigt lämpligt för följande applikationer:

- Program som behandlar text eller binära datafiler.

- Applikationer där endast uppslagning av lexikon behövs, vilket kan vara sparat på disk. Där kan man använda standardbiblioteket modul *dets*, vilket är en disk-baserad version av modulen *ets*.

- Applikationer som behöver diskloggningsmöjligheter, kan utnyttja modulen *disc_log* inställningen.

- Mnesia är inte lämpligt för ”hårda realtids system”.

3.2.3 Att bygga en Mnesia databas

3.2.3.1 Definition av schema

Inställning av ett Mnesia-system är beskrivet i schema. Schema är en speciell tabell, vilken innehåller upplysningar som t.ex. tabellnamn och varje tabells förvaringssätt, d.v.s. om en tabell borde vara sparad i RAM minne, på disk eller möjligtvis på båda.

Till skillnad från datatabeller, kan information som förvaras i schema tabellerna bara vara åtkomligt och ändras genom schemarelaterade funktioner.

Mnesia har olika funktioner för att definiera ett databasschema. Det är möjligt att flytta, ta bort eller omkonfigurera layouten av tabeller. En viktig aspekt av dessa funktioner är, att ett system kan ha tillgång till en tabell medan det blir omkonfigurerat. Till exempel är det möjligt att flytta på en tabell och samtidigt utföra inskrivningsoperationer till samma tabell. Denna finess är viktig för applikationer som behöver oavbruten betjäning[6].

3.2.3.1.1 Funktioner i Schema

Här beskrivs funktioner som är tillgängliga för schemahantering, vilka alla returnerar en tupel, *{atomic, ok}*, eller *{aborted, Reason}* om den misslyckas[6].

- ***mnesia:create_schema(NodeList):*** Denna funktion används för att starta ett nytt och tomt schema. Den är ett obligatoriskt krav innan Mnesia kan startas. Mnesia är ett riktigt distribuerat DBMS och schema är en ”systemtabell” som är kopierad till alla noder i ett Mnesiasystem. Funktionen kommer att misslyckas om ett schema redan finns på någon nod i *”NodeList”*. Denna funktion behöver

Mnesia för att läggas in i alla *”db_nodes”* förvarade i parametern *”NodeList”*. Applikationer anropar denna funktion bara en gång, eftersom det vanligtvis räcker med ett anrop för att starta en ny databas.

- ***mnesia:delete_schema(DiscNodeList):*** Denna funktion raderar allt gammalt schema från noder i *”DiscNodeList”*. Den avlägsnar även alla gamla tabeller med sina data-innehåll. Denna funktion kräver Mnesia för att läggas in i alla *”db_nodes”*.

- ***mnesia:delete_table(Tab):*** Denna funktion tar bort alla kopior av tabellen *”Tab”*, permanent.

- ***mnesia:move_table_copy(Tab, From, To):*** Denna funktion flyttar kopian av tabellen *”Tab”* från noden *”From”* till en annan nod *”To”*. Typen av tabellförvaring, *{Type}* är reserverad så, att om en tabell med förvaringssättet ”RAM” flyttas (eller kopieras) från en nod till en annan nod, då är den nya tabellen av samma typ, d.v.s. ”RAM”. Det är fortfarande fullt möjligt att både läsa och skriva i en tabell under dess flytt.

- ***mnesia:add_table_copy(Tab, Node, Type):*** Denna funktion skapar en kopia av tabellen *”Tab”* vid noden *”Node”*. Argument *”Type”* måste vara atomen *ram_copies*, *disc_copies*, eller *disc_only_copies*.

- ***mnesia:del_table_copy(Tab, Node):*** Denna funktion tar bort kopian av tabellen *”Tab”* vid noden *”Node”*. När den sista kopian av en tabell är flyttad raderas tabellen.

- ***transform_table(Tab, Fun, NewAttributeList):*** Denna funktion ändrar formaten av alla *records* i tabellen *”Tab”*. Den placerar argumentet *”Fun”* till alla *records* i tabellen *”Tab”*. *”Fun”* kommer att bli en funktion som tar en gammal typ av *record* och returnerar en ny *record* av den nya typen.

- ***change_table_copy_type(Tab, Node, ToType):*** Denna funktion ändrar förvaringssättet av en tabell. Till exempel, om en tabell är lagrad som ”RAM”, så ändras den till *disc_table* om detta förvarinssätt önskas.

3.2.3.2 Datamodell

Datamodellen i Mnesia är en utökad relationsdatamodell. Datan är organiserad som en samling tabeller och relationer mellan olika data-*record* kan vara modellerade som extra tabeller som beskriver den verkliga relationen. *Record* är representerade som Erlang tupel.

Objekt-identifierare, också kända som ”*oid*”, är uppbyggda av ett tabellnamn och en nyckel. Om vi t.ex. har en anställds *record* så är den representerad som en tupel, {anställd, 328724, Anders, man, 97104, {221, 015}}. Denna *record* har ett objekt-ID, (Oid) som är presenterat i en tupel, {anställd, 328724}.

Varje tabell är uppbyggd av *record*. Det första elementet i en tabell är ett *record*-namn och det andra elementet är en nyckel, vilken identifierar en speciell *record* i en tabell. Kombinationen av en tabells namn och nyckel är presenterad som en tupel, *{Tab, key}* s.k. ”Oid”.

Det som gör en Mnesia datamodell till en utökad relationsdatamodell är möjligheten att förvara godtyckliga Erlang termer i ett attributfält. Attributets värde kan till exempel vara en hel trädstruktur av ”Oid” som leder till andra termer i en annan tabell. Med andra ord fungerar den som en pekare. Detta typ av *record* är svår att modellera i den traditionella DBMS[6].

3.2.3.2.1 Record

Record är en datastruktur avsedd för att lagra ett bestämt antal data som har besläktade egenskaper. *Record* är densamma som *struct* i C och *record* i Pascal.

Record presenteras som en tupel och ordningen av alla element i denna tupel är strikt definierade. *Record* kan definieras som följande exempel:

-record (recordname, {company, phon, city}).

Som vi ser är denna *record* presenterad som en tupel: { *recordname, x, y, z}* och första elementet är alltid namnet på *record* och resterande element är fältet av *record*. Fördelen med *record* är att man kan referera till ett fält med hjälp av fältets namn och inte med fältets position.

Record kan vara skrivet både som en extern fil som sedan inkluderas i en modul eller som en intern programrad. I första fallet har filen ändelsen ”*.hrl*”, t.ex. ”*person.hrl*” .

3.2.3.2.2 Transaction

Transaction är ett viktigt verktyg när man vill konstruera ett pålitligt distribuerat system. *Transaction* är nyckeln till konstruktionen av ett feltolerant distribuerat system. *Transaction* är en mekanism, vilken tillåter oss att utföra en serie databas-operationer som ett funktionellt block. *Transaction* har s.k. *ACID* egenskaper:

- ***A**tomicity:* Varje ändring av programkod i DBMS resulterar i full effekt på alla noder i nätverket.
- ***C**onsistency: Transaction* lämnar alltid DBMS i ett logiskt läge.
- ***I**solation:* Olika *Transactions* som körs på olika noder i ett nätverk och har tillgång till samma dataobjekt och manipulation av dessa data påverkar inte varandra.
- ***D**urability:* Ändringar som görs i DBMS är garanterat permanenta efter att ett *Transaction* är avslutad.

Exemplet här nedan demonstrerar *ACID* egenskaper. Detta är programkoden för att höja lönen för en specifik anställd.

```
raise (Eno, Raise) ->
        F = fun () ->
                [E] = mnesia:read ({employee, Eno}),
```

```
            Salary = E#employee.salary + Raise,
            New = E#employee {salary = Salary},
            Mnesia:write (New)
        End,
Mensia:transaction (F).
```

3.2.3.3 Starta Mnesia

Före starten av Mnesia, måste man:

- Initiera ett tomt schema för alla närvarande noder.
- Erlangsystemet måste vara startat.
- Noder med disk-databas-schema måste vara definierade och implementerade med funktionen ”*create_schema(NodeList).*”.

När ett distribuerat system med två eller flera noder körs, så måste funktionen ”*mnesia:start ( ).*”, köras på alla närvarande noder. Detta kan vara en typisk del av en *boot script* i en ”inbäddad miljö”. I en ”test miljö” eller ”interaktiv miljö”, kan ”*mnesia:start ( ).*” användas från ett Erlangskal eller något annat program.

3.2.3.3.1 Ett databasexempel

Nedan följer ett exempel på en persondatabas. Modulen heter *”db_person”* och exporterar fem funktioner som interface mot användaren.

- *start (),* startar en ny mnesiatabell med namnet *”person”*. Fälten i tabellerna är beskrivna i *record* där det första fältet (*name*), blir nyckelordet:

  ```
  -record (person, {name, occupation, city}).
  ```

- *stop ()*, raderar databasen *”person”*.

- *mn_put (Name, Occupation, City)*, gör en ny *record* m.h.a. den interna funktionen *”fun_put(Name, Occupation, City)”*, som senare sparas i databasen genom en mnesiatransaktion. All kommunikation med Mnesia görs genom transaktioner.

- *mn_get (Name)*, läser av informationen om personen *”Name”*, om den finns med i databasen. Observera att nyckelordet blir det första fältet i *record*.

- *mn_delete (Name)*, tar bort record med nyckelordet *”Name”* ifrån databasen.

```
-module(db_person).
-export([ start/0,
          stop/0,
          mn_put/3,
          mn_get/1,
          mn_delete/1 ]).


-record(person, {name, occupation, city}).


start() ->
   case  mnesia:create_table( person,
                              [{attributes,
                                  record_info( fields, person ) }]) of
         {atomic, ok} ->
            mnesiaTable_person_started;
         {aborted, Reason} ->
            {error, Reason};
         Other ->
            Other
   end.
```

```
stop()->
   case mnesia:delete_table(person) of
         {atomic, ok} ->
            ok;
         {aborted, Reason} ->
            {error, Reason}
   end.

mn_put( Name, Occupation, City ) ->
   case mnesia:transaction( fun_put( Name, Occupation, City )) of
         {atomic, ok} ->
            ok;
         {aborted, Reason} ->
            {error, Reason};
         Other  ->
            {error, Other}
   end.


mn_get( Name ) ->
   case mnesia:transaction( fun_get( Name ))  of
         {atomic, [Val]} ->
            {ok, Val};
         {aborted, Reason} ->
            {error, Reason};
         Other  ->
            {error, Other}
   end.


mn_delete( Name ) ->
   case mnesia:transaction(fun_delete(Name))  of
         {atomic, ok} ->
            ok;
         {aborted, Reason} ->
            {error, Reason};
         Other  ->
            {error, Other}
   end.
```

```
%%--------------------
%% internal FUN:s
%%--------------------

fun_delete( Name ) ->
   fun()->
          mnesia:delete({ person, Name })
   end.

fun_get( Name ) ->
   fun() ->
          mnesia:read({person, Name})
   end.

fun_put( Name, Occupation, City ) ->
   fun() ->
          mnesia:write(#person{name       = Name,
                               occupation = Occupation,
                               city       = City})
   end.
```

Mnesia startas från applikationsmodulen, med kommandot:

application:start(mnesia).

Och avslutas med:

application:stop(mnesia).

En körning av persondatabasen kan se ut på följande sätt:

```
1> application:start(mnesia).
ok

2> db_person:start().
mnesiaTable_person_started
```

3> db_person:mn_put(tore, snickare, uppsala).
ok

4> db_person:mn_put(magnus, consultant,kista).
ok

5> db_person:mn_put(urban, maalare, stockholm).
ok

6> db_person:mn_get(magnus).
{ok,{person,magnus,consultant,kista}}

7> db_person:stop().
ok

8> application:stop(mnesia).
=INFO REPORT==== 12-Mar-1999::09:59:07 ===
application: mnesia
exited: stopped
type: temporary

3.3 Mnesia Session

3.3.1 Introduktion

Mnesia Session, är ett interface mot Mnesia *Database Management System (*Mnesia *DBMS)* och är en del av *Open Telecom Platform (OTP).* Mnesia Session gör det möjligt att komma åt Mnesia databasen från andra programmeringsspråk än Erlang. Mnesia Sessions interface är definierat i *Interface Definition Language (IDL)* och kan nås m.h.a. protokollen *Internet Inter ORB Protocol (IIOP)* och *erl_interface*[7].

3.3.2 Interface

IDL-modulen mot Mnesia består av två interface:

- ***Connector interface***, som startas statiskt när en Mnesia Session applikation startas.

- ***Session interface***, som startas dynamiskt vid en *request* från en Klient.

När en Mnesia Session klient behöver tillgång till Mnesia DBMS, lokaliseras först en *connector* i någon Erlang nod och startar en session där m.h.a. funktionen *connect* från *connector interface.*
Nedanför följer ett exempel på hur en session startas.

I Erlangs kommandofönster skrivs:

```
 1> application:start(mnesia_session).
ok
```

```
 2> Name = mnesia_connector.
mnesia_connector

3> Connector = erlang:whereis(Name).
<0.34.0>

4> Session = mnesia_connector:connect(Connector).
<0.35.0>

5> ok = mnesia_connector:disconnect(Connector, Session).
ok
```

När en Mnesia Session applikation startas, så registreras en Erlangprocess med namnet *mnesia_connector*.

När en session har startats kan den användas för att starta flera Mnesia funktioner. Dessa funktioner körs lokalt i noden där själva session befinner sig. De flesta av Mnesias funktioner har ett *location transparent* beteende[7].

3.3.2.1 Interface Definition Language (IDL)

IDL-specifikationen av Mnesia Session har två alternativ när den kompileras. Dessa kan hittas i *mnesia_session/include* biblioteket:

- *Mnesia_session.idl*
- *Mnesia_corba_session.idl*

Mnesia_session.idl filen måste vara kompilerade med IC (OTPs egen IDL-kompilatorn). Genererade filer använder distribuerade protokoll av Erlang (*erl_interface*) till att utföra kommunikation mellan klienter och servrar av *connector* och *session*[7].

3.4 Mnemosyne

3.4.1 Introduktion

Mnemosyne är det *query*-språk som Mnesia använder. *Query*-kompilatorn är optimerad för Mnesia DBMS och stödjer en enkel syntax för komplexa sökningar.

En databas förfrågning (*query databas*) används när en mer komplex operation än en enkel *key-value*-sökning krävs av databasen. En *query* kan hitta alla *records* i en tabell som uppfyller en viss egenskap. Exempelvis kan man tänka sig att man har en tabell med tillstånden av telefonabonnenternas linjer i en telefonväxel. En förfrågan i denna tabell kan vara: ”Vilka linjer är 'blockerade'?”

En förfråga kan också hitta *records* baserade på relationen till andra *records* från samma tabell eller från andra tabeller. Om tabellen i förra exemplet utökas med en tabell som ger en relation mellan abonnentens nummer och dess linjeidentifikation, då kan man ändra den gamla förfrågan och ställa frågan: ”Vilka abonnentnummer är 'blockerade'?” Detta kan besvaras genom att man först gör en förfråga som hittar alla blockerade linjer (i abonnenternas linjetabell) och sedan hittar de associerade abonnentsnumren i relationstabellen (m.h.a. relationen mellan abonnentsnummer och dess linjeidentifikation)[5].

3.4.2 Mnemosyne exempel

För att illustrera Mnemosyne *query*-språket, kan man tänka sig att programkod för tabellerna av abonnenternas linje och abonnenternas nummer från föregående exempel i introduktionen är följande:

```
record (subscriber, {snb,
                     cost_limit,
                     li}).
```

record (line, {li,
state}).

En förfrågning är deklarerade på följande sätt:

Query [<pattern> || <body>] end

<pattern> är en Erlang term utan någon funktionsanrop och *<body>* kan vara ett antal villkor som separeras med koma tecken. För bättre illustration se exempel nedan.

Förfrågan ”Vilka abonnentnummer är 'blockerade'?” kan också uttryckas ”Vilka abonnenter har linjer som befinner sig i tillståndet 'blockerade'?”. Denna förfrågan kan programmeras på följande sätt:

```
query
      [ S.snb ||                  % samlar alla abonnentnummer
        S <- table(subscriber),   % S är i abonnentnummers tabell
        L <- table(line),         % L är i linje tabellen
        L.state = blocked,        % denna linjes tillstånd är blockerat
        L.li = S.li               % L och S använder samma li (linje id.)
      ]
end
```

Exemplet ovan kan förstås bättre m.h.a. följande tabell:

Erlangs lista	[ ]
Sådant att	||
Är i	<-
Och	,

I föregående exempel, i det fallet där man bara har ett par abonnentnummer men en miljon av blockerade linjer, skulle det vara mycket mer effektivt att först hitta abonnenterna och sedan kontrollera om deras linjer är blockerade. I vilken ordning en *query* genomförs beror på hur stora tabellerna är. Exekveringsordning beror också på andra variabler som till exempel: nyckeln och andra distributionsvärden. Användaren behöver bara ange vad han/hon vill och både *query compiler* och *query evaluator* försöker avgöra den bästa exekveringsordningen. Följaktligen kan man ange sin förfrågan på ett läsbart sätt.

Observera även nedanstående exempel:

```
record (employee, {namn,
                   ans_nr,
                   sex,
    company}).

query
      [E.name || E <- table(employee),
                E.sex = female]
end.
```

Här efterfrågas en lista över namn på alla kvinnliga anställda i tabellen *”employee”*.

En viktigt sak, när man skriver en *query*-programkod är att inte glömma och inkludera anvisningen för kompilatorn att hur *query* ska behandlas. Detta görs genom att skriva nedanstående raden i början av den modul man har skrivit själva programkoden för *query*.

```
-include_lib (”mnemosyne / include / mnemosyne.hrl”).
```

4 Märkordspråk

4.1 Introduktion

Den expansiva tillväxten av *World Wide Web* (WWW) har till stor del berott på förmågan att på ett enkelt och billigt sätt kunna distribuera olika sorters dokument till ett allmänt (internationellt) auditorium. En förutsättning för det språng som utvecklingen av Internet har tagit är introduktionen av *HyperText Transport Protocol* (HTTP) och informationsformatet *HyperText Markup Language* (HTML)[25].

Under senare tid har Web-dokumenten blivit stora och mycket mer komplexa. Följaktligen har man börjat känna av de olika begränsningar som HTML innehåller. Dessa mer avancerade tillämpningar som man nu vill skapa ställer högre krav på att kunna hantera information på ett strukturerat sätt. Ett språk utformat för att strukturera och beskriva information finns nu tillgängligt. W3C (*World Wide Web Consortium*, ca. 225 medlemsorganisationer, arbetar med att ta fram gemensamma protokoll för WWW) har utvecklat *eXtensible Markup Language* (XML) för de applikationer som kräver den ovan beskrivna funktionaliteten[17].

För att visa ett XML-dokument behövs inga komplicerade verktyg. Det går bra att läsa det i en vanlig HTML-browser om man bara ändrar filändelsen på dokumentet till ”html”. Vad som händer är att browsern skalar av alla taggar som den inte förstår och fortsätter sedan med det som står innanför. Det betyder att XML-dokumenten normalt visas som en enda löpande text. Om det däremot skulle finnas någon tagg som använder ett namn som finns specificerat i HTML-specifikationen kommer det att behandlas som det skulle ha gjorts i HTML. För att få en bättre presentation av dokumentet använder man en mall som beskriver hur dokumentet skall konverteras till lämpligt format, HTML.

4.2 HTML

4.2.1 Vad är HTML?

För att kunna publicera information för global distribution behövs ett allmänt accepterat språk. Det språk som används av *World Wide Web* för att publicera dokument är *HyperText Markup Language* (HTML).

HTML ger oss förmågan att kunna:

- Publicera *online*-dokument med *head*, text, tabeller, listor, bilder, etc.
- Hämta *online*-information via hypertextlänkar.
- Designa formulär för att få tillgång till olika tjänster, för att söka information, göra reservationer, beställa varor, etc.
- Inkludera olika applikationer i själva dokumentet (ljud, video, etc.).

4.2.2 Varför blev HTML en succé?

Användningen av HTML som är oberoende av leverantör och allmänt tillgängligt har lett till att man som läsare och användare kan ta del av en mängd information oavsett var den publiceras i världen, vilken webläsare man använder och vilken dator man har.

4.2.3 Vad är problemet med HTML?

Problemet med HTML består till stora delar på att man bara har tillgång till en begränsad mängd av fixerade märkord (*tags*) som ibland inte räcker till för att åstadkomma mer avancerade presentationer och tillämpningar. Ska man skriva om en adressbok vill man ha märkord som till exempel:

<ADRESS>, <FORNAMN>, <EFTERNAMN>, <GATA>, <NR>, <TFNNUMMER>, etc.

Att utöka antalet märkord i HTML är inte möjligt. Detta har lett till att många leverantörer av HTML-verktyg har valt att utöka HTML med sina egna märkord. Man ser ofta websidor med följande hänvisning: ”Denna sida visas bäst i Internet Explorer” eller ”Denna sida visas bäst i Netscape”. Ett annat problem med HTML är att strukturen som definieras i HTML-specifikationen är mycket bristfällig. Detta medför att man kan skriva syntaktiskt korrekt HTML som strukturellt sätt är ganska egendomligt. De flesta märkord får förekomma i vilken ordning som helst (<H3> kan förekomma ett <H1> märkord). Detta orsakar att det är svårt att skapa program som bearbetar informationen med automatik[25].

4.3 XML

4.3.1 Vad är XML?

eXtensible Markup Language, XML är en specifikation som beskriver en syntax för hur man kan skapa märkords uppsättningar som i sin tur kan användas för att beskriva och strukturera information eller data. I XML har man tagit bort det som i *SGML (Standard Generalized Markup Language*) har varit komplext, både att förstå och att bygga programvaror för. Det viktigaste hos SGML, flexibiliteten (att själv kunna bestämma märkorden och struktur) finns kvar i XML.

XML är det språk som nästa generation av web-tillämpningar baseras på. Det är ett språk som kompletterar HTML och gör det möjligt att skapa nya avancerade web-tillämpningar som databaspublicering, intelligenta agenter, elektronisk handel och kanaltjänster.

XML är skapat för att underlätta användningen av SGML på nätet. Det ska vara lätt att definiera egna dokument, lätt att underhålla, skicka och dela dem över nätet. Målet är att det ska vara lika lätt att använda XML som HTML.

Vid första anblicken av källkoden i ett XML-dokument ser det ut som ett HTML-dokument. Skälet är att de kommer från samma familj av märkord-språk. Båda språken är en delmängd av SGML, vilket är ett stort språk som försöker omfatta allt som kan vara av intresse i ett märkord-språk. Namnet XML, eX*tensible Markup Language*, är faktiskt lite missvisande eftersom det inte är ett definierat språk utan ett metaspråk som möjliggör konstruktion av ett eget märkord-språk.

XML är inte en ersättare till alla sorters HTML-dokument. Vanliga hemsidor som presenterar information som inte ska förändras ofta kommer nog alltid att skrivas i HTML. XML lämpar sig bättre vid de tillfällen där man är mer intresserad av data istället för presentationen eller när man har dynamiska dokument som skall byggas upp vid presentationen[26].

Nedanför kommer ett enkelt exempel på hur XML ser ut:

```
<?XML version="1.0"?>
<conversation>
<greeting>Hello, world!</greeting>
<response>Stop the planet, I want to get off!</response>
</conversation>
```

Först måste det stå vilken version av XML som används. I en applikation kan man sedan välja ut texten som är märkt med <greeting> och som resultat då få *”Hello, world!”*. Reglerna som bestämmer hur dokumentet ska byggas upp beskrivs i en *Document Type Definition* (DTD) och kan lagras antingen direkt i dokumentet eller på en fil som lämpligen pekas ut med en *Universal Resource Locator* (URL). Reglerna för XML-koden ovan skulle bli:

```
<!ELEMENT conversation (greeting, response)>
<!ELEMENT greeting (#PCDATA)>
<!ELEMENT response (#PCDATA)>
```

”!ELEMENT conversation” betyder att detta är en regel för en *tag* som heter *conversation*. Sedan kommer en parentes som visar vilka typer av *taggar* som tillåts innanför *conversation-taggen*. #PCDATA är ett fördefinierat värde som betyder löpande text.

XML-namespace är en samling av namn, som är identifierade av en URI-reference, vilka används i XML-dokument som elementtyper och attributnamn.

Det finns inget krav på att man måste deklarera en DTD i ett XML-dokument. De applikationer som använder sig av XML-dokumentet utan en DTD angiven kan dock inte använda en parser för att kontrollera att de är korrekta, utan de måste kontrollera det själva, alternativt lita på att dokumentet är korrekt beskrivet. Vid parsning av ett XML-dokument kontrolleras att det är välformat (*well formed*) och giltigt (*valid*). Välbildat betyder att det måste finnas lika många starttaggar som sluttaggar och att de kommer i rätt ordning. För att kontrollera om ett dokument är giltigt krävs att man använder sig av en DTD.

4.3.2 Beskrivning av DTD

Document Type Definition, DTD, består av en uppsättning regler i SGML/XML som talar om vilka märkord som får förekomma och i vilken ordning de får förekomma. Först i alla DTD:er skall det stå vilken version av XML och vilken kodningstyp som används.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

betyder att version 1.0 av XML används och att teckenuppsättningen är ”latin1”, d.v.s. en teckenuppsättning för västeuropeiska språk. Ibland utelämnas informationen om vilken teckenuppsättning som skall användas och då beror det på vilket språk, *parsern* har som *default*.

För att kontrollera om ett dokument är giltigt krävs att man har en DTD som beskriver den giltiga syntaxen för ett XML-dokument. Den bestämmer vilka element som kan kombineras med andra element. En DTD kan antingen ligga först i ett XML-dokument eller som en extern fil som läses in av *parsern* separat. Det finns inget krav på att ett XML-dokument ska innehålla en DTD, men utan DTD:n försvinner kontrollen att det är ett giltigt dokument. När man deklarerar DTD:n externt lägger man en rad i början av XML-dokumentet som ser ut så här:

```
<!DOCTYPE conversation PUBLIC
"http://www.e.kth.se/~e94_ope/DTD/HelloWorld.dtd">

<?XML version="1.0"?>
<conversation>
<greeting>Hello, world!</greeting>
<response>Stop the planet, I want to get off!</response>
</conversation>
```

Det är raden som börjar med *DOCTYPE* som är den intressanta för deklarationen. På den raden står också vad rotnoden i dokumentet heter, i det här fallet *conversation.* Sedan kommer texten *PUBLIC* som betyder att nästa element är en allmänt åtkomlig resurs, som pekas ut med en URL, därefter en hänvisning till var DTD:n finns.

4.3.2.1 Deklaration av element

För att bestämma vilka taggar som får finnas innanför en tagg gör man en deklaration av elementet i en DTD. För att kunna skriva in löpande text mellan taggarna måste man ange att en av undernoderna skall vara av typen #PCDATA. Nedanför kommer ett exempel på ett element med flera undernoder som tillåter löpande text:

```
<!ELEMENT person ( #PCDATA | namn | adress )*>
<!ELEMENT namn ( #PCDATA )>
<!ELEMENT adress ( #PCDATA )>
```

Efter *”<!ELEMENT”* kommer namnet på elementet man vill deklarera, därefter kommer vilka undernoder som tillåts. Stjärnan betyder att man tillåter flera undernoder av de typer som finns innanför parentesen.

XML-koden kan se ut så här:

```
<person>Detta är en beskrivning av<namn>Oswald
Valdo</namn>som bor på
<adress>Mimmervägen i Solna</adress></person>
```

4.3.2.2 Deklaration av attribut

Ett attribut är en beskrivning angiven till ett element. Attributen definieras i elementet som en attributlist. Det enda attribut som alltid finns med är ID-attributet och det finns även om man inte tilldelar det själv. ID-attributet måste vara unikt för varje nod i dokumentet.

De olika attributtyperna är:

- ID, unik identifikation
- IDREF, måste matcha ett ID i dokumentet

- IDREFS, flera stycken IDREF
- CDATA, en textsträng
- NMTOKEN, ett ord med de vanligaste tecknen i, inga blanka (WHITESPACE) tillåts
- NMTOKENS, flera NMTOKEN
- ENTITY, en tidigare definierad med icke parsad entitet (annat dokument, bild eller vad som helst)
- ENTITIES, flera ENTITY
- Enum, det finns en lista över giltiga val, består av ”textord”

Vidare finns det tre olika krav på värdena för attributen:

- #REQUIRED, man måste ange ett värde på attributet
- #IMPLIED, om man inte anger ett värde antas attributet ha sitt defaultvärde
- #FIXED, attributet har ett fördefinierat värde i DTD:n som inte får ändras.

4.3.3 RDF

4.3.3.1 Vad är RDF?

Resource Description Framework, RDF är en ram för beskrivning av metadata (metadata är maskinbegriplig information om web-*resources* m.m.). RDF är en tillämpning av XML, specifikt för metadatahantering. RDF använder XML för sin syntax överföring, men erbjuder mycket mer än XML. RDF infogar en formell matematisk modell vilken definierar relationer mellan olika *Resources*[18].

RDF är noggrant konstruerad och har följande kännetecken[18]:

- RDF är uppdelad i tre delar: *Resource, PropertyType, Value.*

- Oberoende, eftersom en *PropertyType* är en *Resource.* Detta innebär att *PropertyType* kan ha sina egna egenskaper och kan hittas och bli manipulerad likt vilken *Resource* som helst.

- *Value* kan vara en *Resource,* de flesta websidorna kommer att ha egenskapen ”hemsidor” vilken pekar på hemsidan av deras *site*.

- *Property* kan vara en *Resource,* den kan ha egenskaper också.

4.3.3.2 Basic RDF modell

Grunden för RDF är en modell för representation av namngiven *Property* och värdet (*Value*) av *Property*.

Datamodell för RDF är ett *syntax-neutralt* sätt för representationen av RDF-uttrycket. Datamodellsrepresentation används för att uppskatta likvärdigheten i betydelsen (*meaning*). Två RDF-uttryck är likvärdiga om och endast om deras datamodellsrepresentation är densamma. Denna definition av likvärdighet tillåter några syntaktiska variationer i uttrycket utan att förändra ”betydelsen”. Basic datamodell består av tre delar:

- ***Resource:*** Alla saker som beskrivs av RDF-uttryck kallas för *Resources*. En *Resource* kan vara en fullständig websida, t.ex. ett HTML-dokument (”http://w3.org/exempel.html”) eller bara en del av en websida, t.ex. ett specifikt HTML- eller XML-element inuti en dokument-*resource*. En *Resource* är någonting som har en *Uniform Resource Identifiers* (URI), denna inkluderar hela världens websidor, såväl som enskilda element av ett XML-dokument.

- ***Property:*** En *Property* är en kombination av *Resource, PropertyTyp* och värde *(value).* Ett exempel skulle vara: ”Författaren av http://www.textuality.com/RDF/varför.html är Anna”. *Value* kan vara en sträng som t.ex. ”Anna” i det tidigare exemplet eller så kan det

vara en *Resource* liksom ”hemsidan av http://www.textuality.com/RDF/varför.html är http://www.textuality.com.”

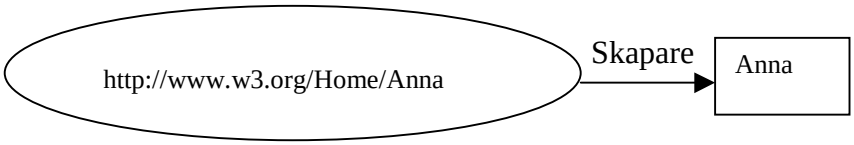
- ***Statement**:* En specifik *Resource* tillsammans med en namngiven *Property* plus värdet av *Property* för den specifika *Resource* är en RDF-*Statement*. De tre individuella delarna av ett *Statement* kallas för *Subject*, *predicate* och *object*.

4.3.3.3 RDF exempel

Resource identifieras genom ”*Resource identifier*”. En ”*Resource identifier*” är en URI plus en valfri *”anchor id”*. Meningen: ”Anna är skapare av *resource* http://www.w3.org/Home/Anna”, kan delas upp i följande delar:

1. *Subject* (Resource): http://www.w3.org/Home/Anna
2. *Predicate* (Property): skapare
3. *Object* (Literal): ”Anna”

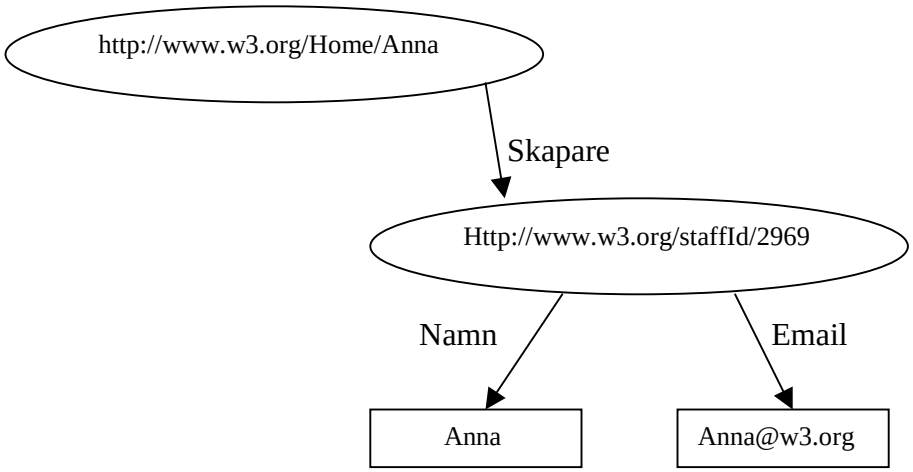
Dessa kan illustreras grafiskt med hjälp av s.k. *node* och *arc*.
Figur.2 här nedan innehåller två noder som sammankopplas med en enkel *arc*. Den ena noden (oval) innehåller en URI och den andra noden (rektangulär) innehåller text och den *arc* som sammankopplar dessa två noder har beteckningen ”skapare”. Notera att riktningen på pilen är viktig. En *arc* startar alltid från ett *subject* och pekar på ett *object* av ett *statement*.



Figur.2

Om man nu vill utveckla detta exempel med en unik identifierare och mera information om skaparen så kan man tänka på följande beskrivning och texten kan illustreras grafiskt med en RDF-modell:

”En hänvisning till en anställd med anställningsnummer 2969, är Anna som har email adressen Anna@w3.org. En *Resource* ”http://www.ww3.org/Home/Anna” är skapad av denna individ.”



Figur.3

Figur.3 har fyra noder och tre *arcs*. Två av noderna innehåller strängarna ”Anna” och ”Anna@w3.org” m.a.o. innehåller de här två noderna värdet av ”*name-Property*” och ”*Email-Property*”. En enkel *arc* med beteckningen ”skapare” sammankopplar noden ”http://www.ww3.org/Home/Anna” med den andra noden ”http://www.w3.org/staffId/2969”. Två *arc* med benämningen ”Namn” och ”Email” går från den sistnämnda noden till motsvarande noder med textsträngar.

4.3.3.4 Exempel på RDF med ett gemensamt värde

En *Resource* kan vara värdet på mer än bara en *Property*. Den kan vara *object* till mer än bara ett *statement* och därför kan den bli pekad av flera *arcs*. Till exempel kan en websida vara gemensam för flera dokument. Ett specifikt exempel är en samling av en författares verk, som är sorterade med tanke på publiceringens datum och även sorterade i alfabetisk ordning som visas här nedan:

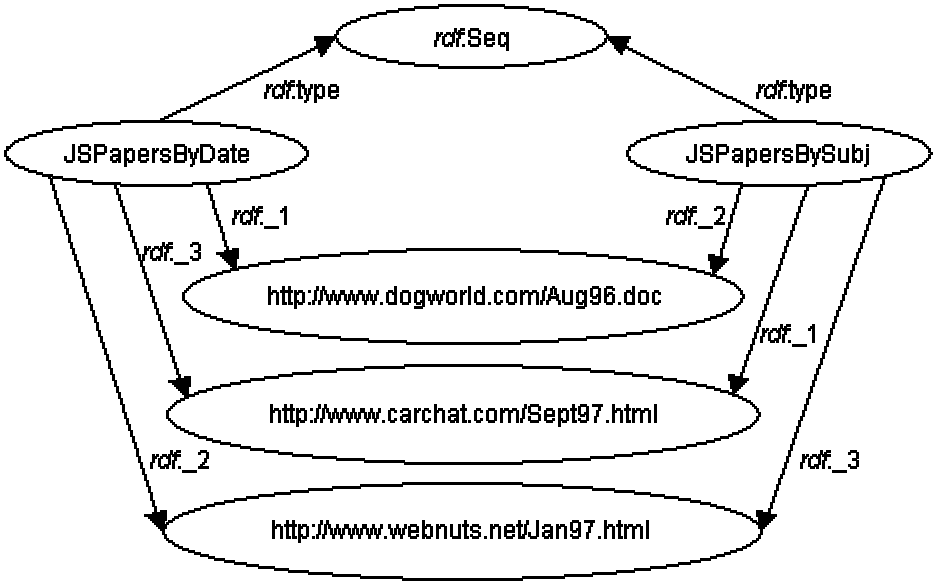
```
<RDF xmlns="http://w3.org/TR/1999/PR-rdf-syntax-19990105#">

<Seq ID="JSPapersByDate">
    <li resource="http://www.dogworld.com/Aug96.doc"/>
    <li resource="http://www.webnuts.net/Jan97.html"/>
    <li resource="http://www.carchat.com/Sept97.html"/>
  </Seq>

  <Seq ID="JSPapersBySubj">
    <li resource="http://www.carchat.com/Sept97.html"/>
    <li resource="http://www.dogworld.com/Aug96.doc"/>
    <li resource="http://www.webnuts.net/Jan97.html"/>
  </Seq>

</RDF>
```

Texten från föregående sida kan illustreras grafiskt med följande figur:



Figur.4

Figur.4 innehåller sex noder och åtta *arc*. Två av noderna motsvarar två värdesekvenser (JSPapersByDate och JSPapersBySubj) som pekar till noden rdf:Seq[18].

4.3.3.5 Varför inte bara använda sig av XML?

XML tillåter oss att uppfinna *taggar* som innehåller både datatext och andra *taggar*. XML har en inbyggd åtskillnad mellan elementtyper till exempel IMG elementtypen i HTML och element. XML faller isär på konstruktionsdesignens skalbarhet. XML har två problem:

1. Den ordning i vilken elementen framträder i ett XML-dokument är viktig och ofta meningsfull. Denna kan tyckas vara starkt onaturlig i metadatanvärld. Till exempel, vem bryr sig om att regissörens namn eller filmens titel kommer först i dokumentet eller sist, bara dessa är tillgängliga för att kunna slås upp.

2. XML tillåter konstruktioner som:
   <Description> Värdet av denna *Property* innehåller text, blandade med en *"child Property"* som dess temperatur (<Temp>48</Temp>) </Description>.
   När man presenterar en generell XML-dokument i ett dataminne, då får man en konstig datastruktur som innehåller träd, figurer och teckensträngar. Dessa är svåra att hantera.

Men trots alla XMLs brister så är XML en absolut nödvändigdel av lösningen till RDFs konstruktion.

4.3.3.6 Varför ska man vara intresserad av RDF?

Svaret på frågan kan summeras med att RDF innehåller följande förbättringar:

- Enkel manövrering av metadata.
- Maskinbegriplig semantik för metadata.
- Bättre precision med *resource* sökning än vid heltextsökning.

4.4 Skillnader mellan XML & HTML

Skillnaden mellan XML och HTML är att XML är ett metaspråk för definiering av nya språk medan HTML är ett fullständigt språk. I HTML lagras text och layout tillsammans som en enhet färdigt för publicering. HTML används för presentation av dokument främst på Internet. Eftersom HTML bakar ihop innehåll och layout är det svårare att återanvända material som en gång publicerats om man inte sparar dokumentet utan layoutinformation. Rent syntaxmässigt finns det också ett par undantag som skiljer XML ifrån HTML, t.ex. att man alltid måste avsluta taggar i XML. I vissa markeringar är det dock mest naturligt om det inte finns någon sluttagg, t.ex. bilder. Markeringen *”<IMG SRC="picture.gif">”* betyder i HTML, att det ska finnas en bild som heter *”picture.gif”* här. XML har löst kravet på att inte innehålla oavslutade taggar genom att placera ett snedstreck i slutet av taggen så att det istället står *”<IMG SRC="picture.gif"/>”*[17].

4.4.1 Ett generellt exempel på skillnaden mellan XML & HTML

I nya web-tillämpningar krävs ofta att data behöver utbytas och tolkas mellan två program, till exempel mellan två servrar eller mellan en klient och en server. Det är då HTMLs begränsningar blir tydliga. I ett HTML-dokument beskrivs bara hur data ska presenteras för användaren men inte vad det är för typ av data dokumentet innehåller. Därför är det svårt att skriva program som processar HTML-dokument på ett intelligent sätt. Ta som ett enkelt exempel orderdatum. I ett HTML-dokument skulle det stå:

<B>19990625</B>

Ett program som läser filen och letar efter orderdatum får problem. Det enda man vet är att det som står mellan <B>-taggarna ska presenteras i fetstil. I det här fallet kan det vara ett datum, men det skulle också kunna vara t.ex. ett produktnummer. Det går heller inte att utgå från att allt som

står mellan två <B>-taggar är orderdatum eftersom det finns massor av andra data i HTML-filen som är omgärdade av <B>-taggar.
Anta att det istället stod:

<orderdatum>19980326</orderdatum>

Då skulle det vara en enkel sak att skriva ett program som letar efter <orderdatum>-taggar och sedan extrahera data från filen.
Tyvärr tillåter inte HTML att man lägger in sådana egna taggar utan uppsättningen HTML-taggar är förutbestämd och kan inte ändras av en programmerare.

XML är det språk som ska ta webben till nästa nivå. Poängen med XML är att det tillåter att egna märkord definieras. Det gör det möjligt att i web-dokument uttrycka mer än bara hur innehållet ska visas i en bläddrare. Det blir till exempel möjligt att beskriva strukturerade data som poster från en databas. XML-*taggar* används då för att märka upp vad som är ”Id”, ”namn”, ”ålder”, ”lön” etc[17].

4.5 XSL

4.5.1 Vad är XSL?

eXtensible Stylesheet Language, XSL är ett s*tylesheet*-språk som anger en mall för att omvandla en XML-fil till HTML, och på så vis göra ett XML-dokument presentabelt i en webbrowser. Ett XSL-dokument är skrivet i XML och fungerar på så sätt att man skriver ett antal regler som har två komponenter. En som beskriver indata och en som beskriver utdata. Indata gör att XSL-dokumentet kan bestämma vilken regel som skall användas och sedan tillämpa det som står i utdata på det valda stycket i XML-dokumentet[13].

4.5.2 Exempel på en regel I XSL

```
<rule>
<element type="Meny">
<target-element type="Lunch" />
</element>


<TR>
<TD WIDTH="150" ALIGN="LEFT" VALIGN="TOP">
<eval>getAttribute("Typ")</eval>
</TD>


<TD ALIGN="LEFT" VALIGN="TOP">
<children/>
</TD>
</TR>
</rule>
```

Markeringen (*taggen*) <rule> visar att det kommer en regel. Inuti regeln finns två delar där den första delen är ett mönster som visar på vilka element regeln skall tillämpas och den andra delen vad som ska genereras. Mönstret börjar med *”<element type="Meny">”* som betyder att den först ska hitta ett element som heter *Meny*. Sedan står det *”<target-element type="Lunch" />”*, vilket betyder att under *Meny* ska det finnas ett element som heter *Lunch*. Ordet *target* specificerar att det är för det här elementet som regeln gäller. Snedstrecket i slutet säger att det inte kommer en sluttagg på det här elementet. Mönstret avslutas med en sluttagg för elementet *Meny*.

Sedan kommer det som ska genereras. Det är vanlig HTML-kod förutom raden *”<eval>getAttribute("Typ")</eval>”* som är ett *script*. Scriptet som körs är ett kommando som plockar ut värdet ur attributet *Typ*. En annan rad som inte är vanlig HTML-kod, är *”<children/>”*. Det betyder att regler för underliggande element ska appliceras och svaret ska presenteras där[14].

Exempel på en bit av ett XML-dokument som XSL-regeln kan appliceras på:

<Meny>
<Lunch Typ = "Miljövänlig:">Pannkakor och kycklingsoppa</Lunch>
<Lunch Typ = "Kött:">Köttbullar med stuvade makaroner</Lunch>
</Meny>

Resultatet blir:

Miljövänlig: Pannkakor och kycklingsoppa
Kött: Köttbullar med stuvade makaroner

XSL refereras från ett XML-dokument genom att tillägga följande rad:

<?xml-stylesheet href = "doc.xsl" type = "text/xsl" ?>

Där *”href”* pekar ut en URL med XSL-filen.

Ett XSL-*stylesheet* kan importera andra XSL-*stylesheet*, genom att använda ett *”xsl:import”* element med *”href”* attribut:

```
<xsl:import href = "doc.xsl"/>
```

Alla element *”xsl:import”* måste anges i början av *stylesheet*. Varje *”xsl:import”* har en *”href”* attribut vars värde är URI:n av det *stylesheet* som ska importeras.

5 WAP

5.1 Introduktion

Wireless Application Protocol, WAP är resultatet av arbete från flera industrier för att kunna standardisera och specificera en utvecklingsapplikation över trådlösa nätet. WAP är ett protokoll eller regelverk för överföring av internetinnehåll och andra avancerade telefonitjänster till digitala mobila terminaler som t.ex. GSM-mobiltelefoner. Med andra ord är WAP en teknik som gör det möjligt att ”surfa” på Internet från sin mobiltelefon. En av förutsättningarna är att hemsidan är optimerad för tjänsten. Kort beskrivet använder WAP ett märkordspråk s.k. WML, som i princip motsvarar WWWs HTML och som baseras på XML. Det är således en ganska enkel process att göra internetdata tillgängliga för WAP-telefoner. Märkordspråket, WML kommer att gås igenom detaljerat i avsnitt 5.5.

WAP ska kunna arbeta mot en mobiltelefon (WAP-telefon) som har ett fönster på bara några få punkter och lite CPU-resurser och även mot en PDA (Personal Digital Assistant) som kan ha ett betydligt större fönster och avsevärt mycket mer CPU-kraft och minne än mobiltelefonen[19].

5.2 Översikt

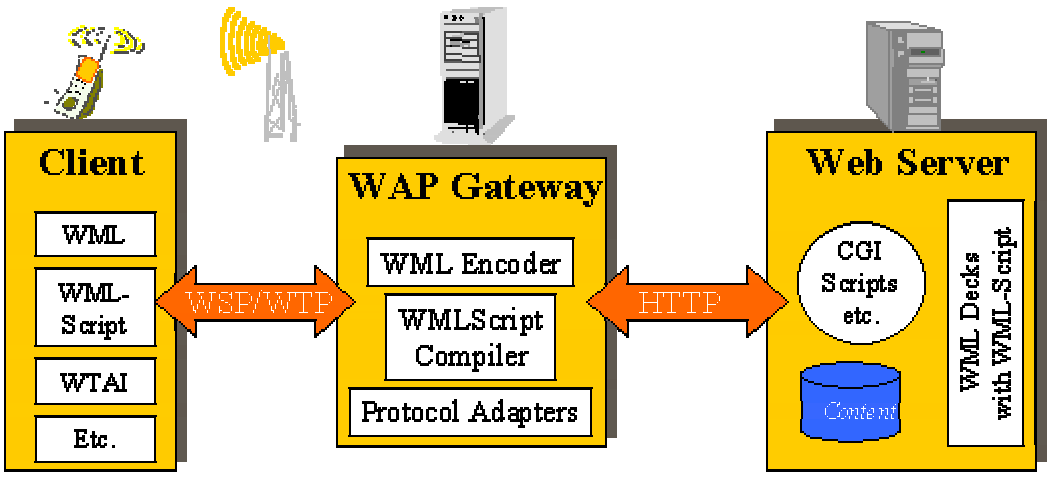
WAP är en industrispecifikation under utveckling för att skapa applikationer över trådlösa nät. WAP är inte knutet till någon speciell mobiltelefonistandard, utan ska fungera oavsett bärare.

En förutsättning för Internets snabba spridning var protokollet TCP/IP, som gör att datorer med olika system kan kommunicera med varandra via Internet. Tillverkare av program och datorer kunde gratis använda TCP/IP. Fler datoranvändare fick tillgång till Internet, som fick allt större spridning och ökad efterfrågan. WAP är också en öppen plattform som alla kan använda för att skapa nya tjänster. Dessutom är WAP oberoende av den underliggande teknologin och fungerar lika bra med SMS- och GSM-data som med circuit switched-data, paketdata, o.s.v. Man kan använda den teknologi som är bäst till applikationen och utveckla tjänsterna i takt med behovet.

WAP är en bro mellan Internet och trådlösa apparater som skapas för att snabbt kunna implementera effektiva applikationer för det trådlösa nätet. All information som kan visas upp i en web-browser med hjälp av HTML kan inte visas i ett begränsat telefonfönster, utan måste formateras om, så att bara den relevanta informationen skickas över till telefonen samt några utvalda bilder. För att kunna åtstakomma detta har man utvecklat ett programmeringsspråk, WML. WML baseras på XML (*eXtensible Markup Language)*, vilken är en specifikation som beskriver en syntax för hur man kan skapa märkordsuppsättningar av ”taggar” som i sin tur kan användas för att beskriva och strukturera information eller data.

XML är det språk som nästa generation av web-tillämpningar baseras på. Det är ett språk som kompletterar HTML och gör det möjligt att skapa nya avancerade web-tillämpningar som databaspublicering, intelligenta agenter, elektronisk handel och kanaltjänster[19].

Här nedan illustreras hur uppkoppling med hjälp av en WAP-telefon går till.

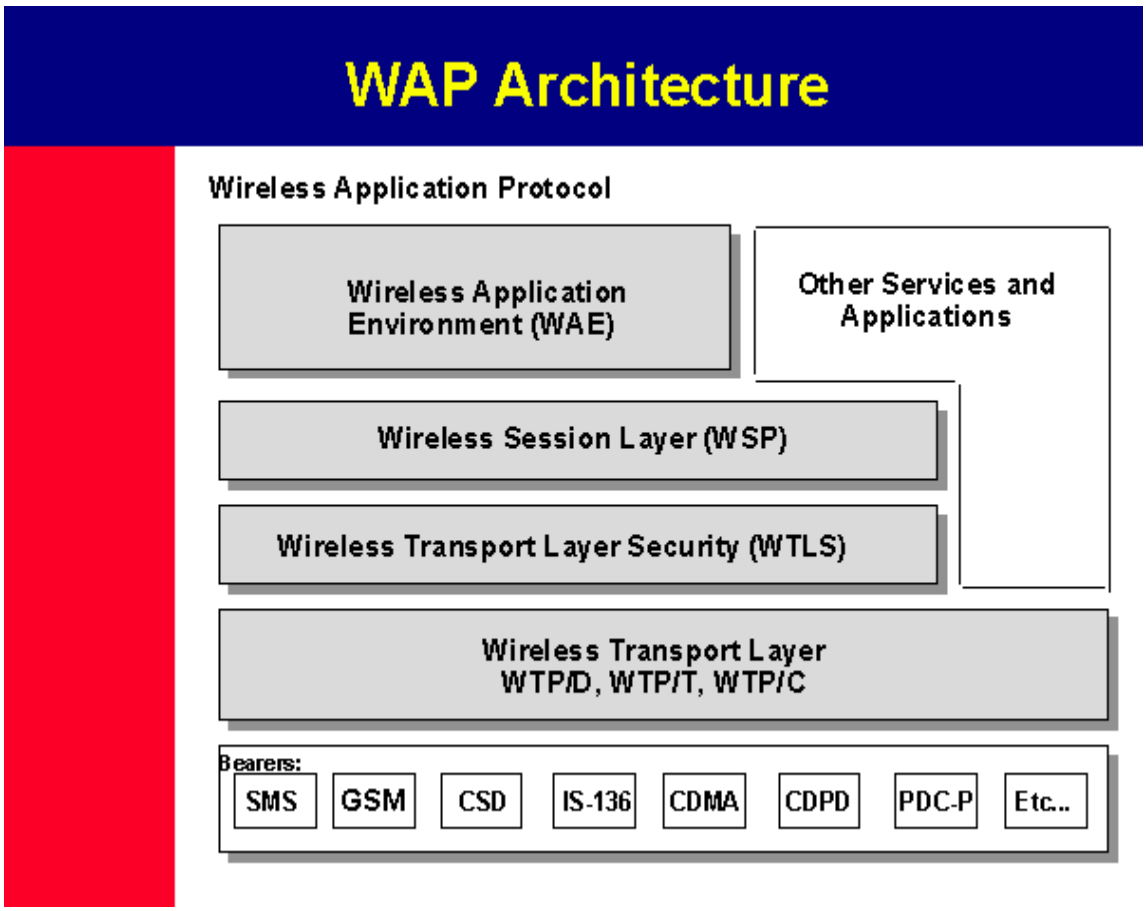


Figur.5

Först sänder telefonen en förfråga (*request*) i WML-kod till en *WAP Gateway*. I WAP Gateway översätts WML-koden till XML-kod för att sedan skicka förfrågan vidare via Internet med HTTP-protokollet. Med hjälp av HTTP-protokollet transporteras XML-koden som innehåller den ursprungliga förfrågan från telefonen vidare till en web-server. Web-servern tar hand om förfrågan och skickar tillbaka svaret till WAP Gateway som i sin tur översätter XML-koden till den WML-kod som ska skickas tillbaka till telefonen.

5.3 Protokollstack

WAP skapades för att snabbt kunna implementera effektiva applikationer för trådlösa nät och har en struktur som är inspirerad av WWW, med en protokollstack som liknar figur.6 här nedan:



Figur.6

Protokollstacken består av oberoende lager för att på så sätt vara skalbar både när det gäller processorkraft, minne och bandbredd.

Transportnivån ligger ovanför de olika bärarna (*bearers*) som ska skicka paket över det trådlösa nätet. Sessionsnivån sköter uppkoppling och nedkoppling av en kanal, samt att den kan hålla ett tillstånd hos server och klient. Detta kan jämföras med HTTP för WWW. Applikationsnivån ger applikationen tillgång till olika händelser (*events*) i systemet.

WAP arkitekturen följer OSI-modellen (*OSI layer model*) och innehåller tre huvudskikt:

- **Applikation lager (WAE, WTA):** *Wireless Application Enviroment* (WAE) och *Wireless Telephony Application* (WTA) är huvudinterface mot klientapparater (*client devices*) och specificerar ett märkordspråk, ett skriftspråk och en telefoni-interface. WAE och WTA lägger på ett antal enkla grundbehov hos klientapparater. T.ex. måste en klientapparat kunna bevara en *”history list”* av den senaste besökta platsen för att en användare ska kunna navigera sig ”bakåt”.

- **Session lager (WSP, WTLS):** *Wireless Session Protocol* (WSP) och *Wireless Transport Layer Security* (WTLS) förser applikationsskiktet (WAE och WTA) med en förbindelsebaserad service. Det finns ett gränssnitt för två sorters sessionsservice. Det första är förbindelse-orienterad (*connection-oriented)* service, vilket fungerar ovanför transaktionslagret WTP och den andra är förbindelselös (*connectionless)* service som arbetar ovanför en datagram service, WPD (*Wireless Datagram Protocol*).

- **Transport lager (WTP, Bearer service):** *Wireless Transaction Protocol* (WTP) och bärarna står för pålitlig transmission av WSP-datapaket mellan klient och server över en trådlös länk.

5.4 Varför WAP?

5.4.1 Introduktion

Antalet mobiltelefonanvändare växer snabbare än antalet PC-användare med Internetförbindelse. I framtiden kommer många av de tjänster som erbjudits via Internet också att kunna nå mobila användare. WAP-telefoner och WAP-datorer kommer därför att spela en viktig roll[19].

5.4.2 Varför satsa på WAP?

- WAP klarar alla typer av datatransport t.ex. TCP/IP, UDP/IP, GUTS, SMS och USSD.
- WAP använder det redan befintlig märkordspråkets teknologi (XML).
- WAP optimerar innehåll och luftlänk (*airlink*) protokoll.
- WML komponenter passar in i den existerande mobiltelefonens användarinterface, vilket medför att man inte behöver omskola användaren.
- WAP använder web HTTP 1.1 server.
- WAP arkitektur har flera modulära enheter, tillsammans formar de en fullständig Internetenhet. WML-innehåll är tillgänglig via HTTP 1.1 *request*.

5.5 WML

5.5.1 Vad är WML?

Wireless Markup Language, WML är ett märkordspråk baserad på XML och är avsett för att presentera web-sidor i bandbegränsade apparater, som t.ex. mobiltelefoner. WML ärver XMLs karaktäristiska uppsättning. WML är designad med tanke på nedanstående begränsningar:

- Liten skärm och begränsad inmatningsmöjligheter.
- Bandbegränsad nätverkförbindelse.
- Begränsade minnes- och uträkningsresurser.

WML inkluderar fyra stora huvudfunktioner[20]:

1. Textpresentation och layout: WML kan infoga bilder i presentationen och har vissa möjligheter att formatera texten, t.ex. fetstil, kursiv o.s.v.

2. *Deck/card* metafor: all information i WML organiseras i en mängd *card* och *deck*. *Card* specificerar en eller flera enheter av användarinteraktion (t.ex. en valmeny eller en skärm av text). Det logiska är att en användare navigerar sig igenom ett antal WML-*cards*. För varje *card* går användaren igenom innehållet, gör sitt val och går vidare till ett annat *card*. *Cards* är grupperade i *decks*. Ett WML-*deck* motsvarar en HTML-sidan.

3. *Inter-card* länkning: WML inkluderar support för tydlig navigering mellan *cards* och *decks*.

4. Variabler: alla WML-*decks* kan parametriseras. Då kan man använda variabler istället för strängar. Detta tillåter en mer effektiv användning av nätresurserna.

5.5.2 Syntax

WML använder sig liksom HTML av märkord (*taggar*). De kan vara av två format:

<tag> content </tag>
eller bara
<tag/>

Det första innebär att alla taggar som startas med en start-tagg (<tag>), måste avslutas med en slut-tagg (</tag>). Det andra används för att XML kräver att alla taggar måste stängas. Vissa taggar innehåller ingen *”content”* och använder samma tagg som både start- och slut-tagg, t.ex.
 byter rad i HTML men i WML används
.

För att kunna ge mer information till taggarna har WML stöd för attribut. <tag attr=”abcd”/>

Till skillnad från HTML är WML *case-sensitive,* d.v.s. att ”<tag> content </tag>” inte är det samma som ”<TAG> content </TAG>”.

5.5.3 Events

Det finns möjlighet att definiera vissa händelser (*events)*, t. ex. vid navigering (föregående eller nästa *card*). Dessa *events* har räckvidd inom det block de är definierade i. Om de definieras inom *deck*, kan de anropas från alla *cards* inom samma *deck*, och om de är definierade inom ett *card* är räckvidden endast inom samma *card*. Om samma händelse är definierad på olika ställen i dokument kommer den mest lokala händelsen att styra. Om en händelse är definierad i *deck* och även i det *card* som utlöser händelsen kommer *card*-definitionen att gälla.

WML inkluderar en funktion för att hålla reda på vilka sidor man besökt. Denna historifunktion består av en stack med URL-adresser. Denna stack

kan tömmas, utökas eller minskas. När man surfar till föregående URL kommer aktuell URL att tas bort från historiestacken för att spara minne. Det är viktigt att historiestacken hålls liten.

Go är ett *event* som används vid navigering, *go* kan navigera både till *card* och *deck.*

Attribut till *Go*:

- *URL:* Destinationsadress.
- *VARS:* Möjlighet att skicka variabler.
- *SENDREFERER:* Om aktiell URL ska representeras av ”HTTP referer” eller ej. Detta attribut möjliggör viss access-kontroll, när en kortlek kan se vem anroparen är.
- *METHOD:* Typ av HTTP-överföring. WML har för tillfället stöd för POST och GET.
- *ACCEPT-CHARSET:* Specificerar vilka teckensnitt mottagaren måste klara för att kunna visa dokumentet korrekt.

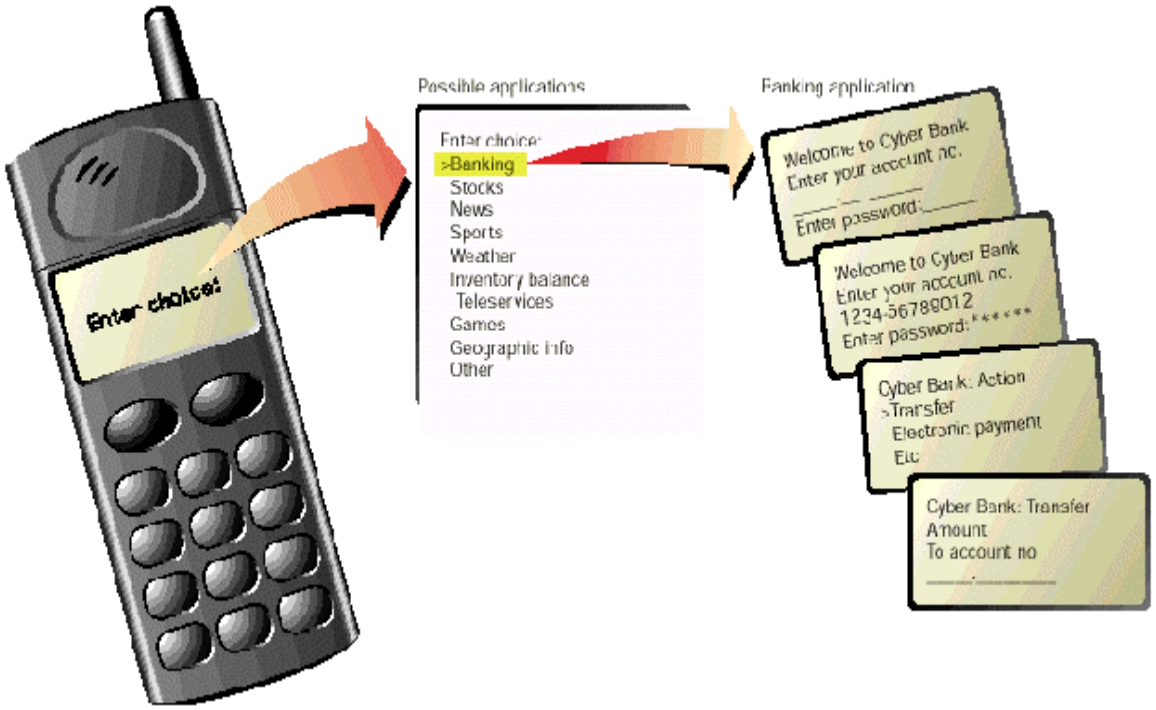
A är ett enklare navigerings-*event*, som specificerar en destinations-URL (fungerar som länkar i HTML) eller *prev* för att komma till föregående URL på historiestacken.

DO-elementet tillåter användaren att interagera med ett *card*. Vilken typ av inmatning som används beror på vilken apparat applikationen körs på. Det kan vara *YES*-knappen på telefonen, eller en röst- aktiverad funktion.

5.5.4 Struktur av WML-deck

En ensam kollektion av flera *card* kallas för WML-*deck*. Ett *card* innefattar strukturerade innehåll- och navigeringsspecifikationer. Ett giltigt WML-*deck* är ett giltigt XML-dokument och därför måste den innehålla en XML-deklaration och en dokumenttypdeklaration.

Ett WML-dokument (*deck*) laddas ner från servern med en enda förfråga. Klienten kan därefter navigera igenom alla *cards* tills nästa förfråga skickas iväg till servern. Här nedan i figur.7 illustreras ett WML-*deck* med sina respektive *cards*.



Figur.7

I figur.7 när man väljer ”*Banking*” laddas ett *deck ner* med alla *cards* associerade med denna tjänst.

Ett giltigt WML-*deck* är ett giltigt XML-dokument och därför måste det innehålla en XML-deklaration och en dokumenttypsdeklaration. Här nedan presenteras ett enkelt exempel på ett WML-dokument:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                "http://www.wapforum.org/DTD/wml_1.1.xml">


<wml>
   <card newcontext="true">
      <do type="accept">
         <go href="#card2">
         </go>
      </do>
      <p align="center">
       Hello, hello!
       This is the first card
      </p>
   </card>
   <card id="card2">
      <p>
       This is the second card.
       Goodbye.
      </p>
   </card>
</wml>
```

Presentationen av WML-koden från föregående sidan i en WAP-telefon skulle se ut på följande sätt:

Figur.8

Figur.9

I förra exempel ser vi ett *deck* med två *cards* . *Card* 1 innehåller en länk till *card* 2, vilken kommer att aktiveras av användaren genom ett tryck på *YES*. När *card* 2 aktiveras kommer texten ”*This is the second card.*” att visas.

Har apparaten som applikationen körs på ett litet fönster är det bra att *deck* är indelad i *cards* som blir en naturlig avgränsning, d.v.s. ett *card* kan visas per ruta. Sedan är det effektivt att ladda hem ett helt *deck* och sedan ”surfa” i minnet.

5.5.5 Bilder

WML tillåter visning av bilder men det handlar om små bilder med en liten upplösning. Bilder kan infogas i ett WML-dokument med taggen <img/> och med hjälp av en mängd attribut. Ett exempel på WML-koden för bild ser ut så här:

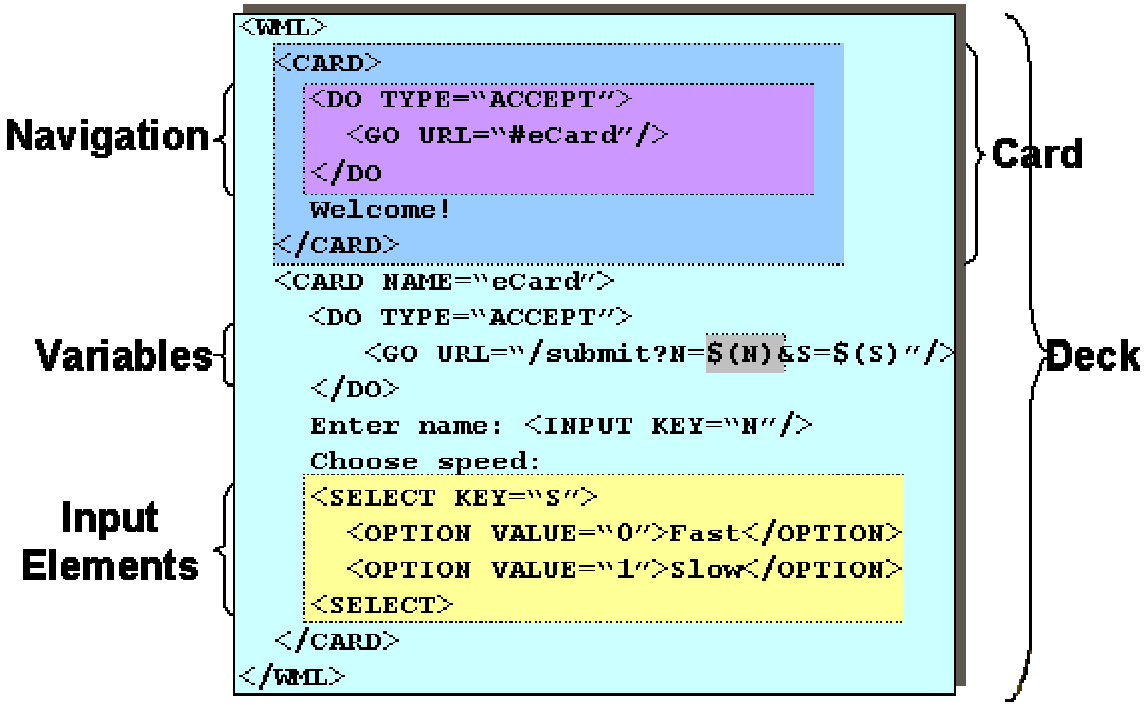
**

I telefonen skulle detta visas på följande sätt:

Figur.10

5.5.6 WML exempel

Följande figur förtydligar en WML-dokumentstruktur:



Figur.11

5.5.7 WML-script

WML-script baseras på *javascript* men är anpassat till resursbegränsade klienter. En stor skillnad är att WML-scripten kompileras till *bytecode* innan det skickas till klienten. Det ger ett bättre utnyttjande av nätet. Av samma orsak har scriptet bantats ner. Flera av javas avancerade funktioner saknas.WML-script ger en möjlighet att utöka WML, på ett sätt som inte tillåts i WML.

Med WML-script kan man:

- Kontrollera indata
- Använda apparatens funktioner för att t. ex. ringa, skicka meddelanden, lägga till nya poster i telefonboken eller anpassa SIM-kortet.
- Generera lokala felmeddelanden och på så vis minska behovet av kostsam *”round-trip”* för att visa varningar, fel eller bekräftelser.
- Utöka apparatens mjukvara efter att den har levererats till kund.

5.6 WapIDE

WAP Integrated Development Enviroment, WapIDE, underlättar skapandet av applikationer som använder WAP. Den möjliggör en test miljö för WAP-applikationer. En WapIDE är en komplett utvecklingsmiljö som hjälper utvecklare i nedanstående områden[27]:

1. Design av WML-baserade applikationer, inklusive en synlig presentation av simuleringens resultat (Browser & Application designer).

2. Design av nya apparater, med vilken man kan testa och se en WML-applikation i den nya apparaten (Device designer).

3. Utveckling av WAP-servrar. När WapIDE används som en simulerad terminal över ett nätverk (Server toolset).

Figur.12

6 Implementering

6.1 Introduktion

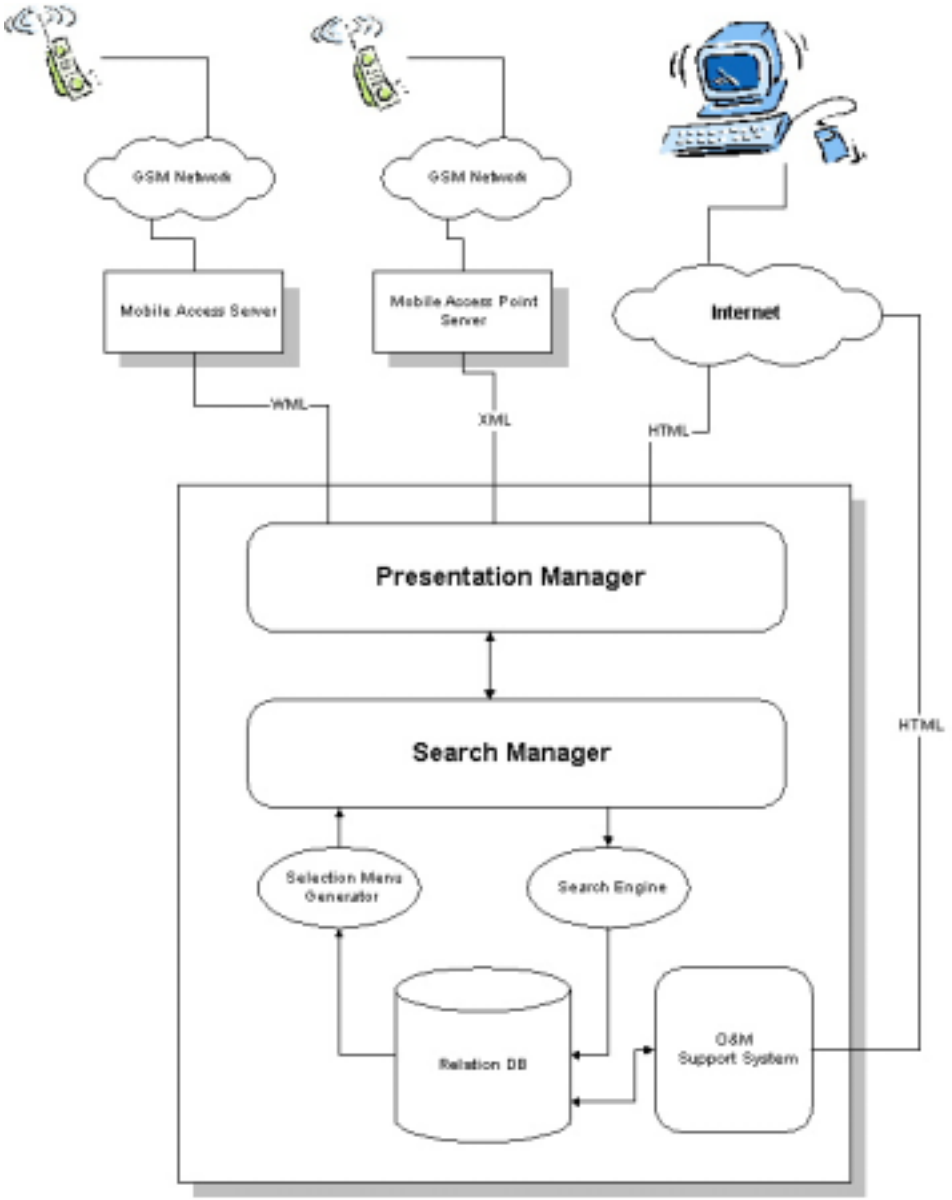
Arbetet har utförts enligt en iterativ modell. Med iterativ menas att synpunkter hela tiden har tagits i beaktande och applikationen har justerats därefter. I början av utvecklingsarbetet gjordes stora förändringar i systemarkitekturen, senare under implementeringsfasen justerades endast mindre detaljer av designen.

Vid designen av systemarkitekturen togs hänsyn till systemet som helhet. När vi hade ett färdigt förslag att jobba efter, insåg vi hur stort och komplext det hela hade blivit och efter några samtal med vår handledare på Ericsson kom vi överens om att vi skulle begränsa vårt arbete till sökmaskinsdelen. Vårt arbete med sökmaskinen kan sammanställas, i stora drag, av en databas där sökningarna ska göras och av en server som kan hantera både HTML-request från en web-browser och WML-request från en WAP-telefon. Senare under implementeringen har vi lagt till funktioner för att identifiera en speciell sorts förfråga, gjord med hjälp av XML. Dessa ”*XML-request*” genereras av en MAPS-maskin (Mobile Access Point Server) som ingår i ett EU-projekt som Ericsson arbetar med. Detta XML-request görs enligt ett gränssnitt som kallas för CEI (Common Engine Interface). För att CEI ska kunna använda sig av XML har man skrivit en DTD som definierar vilka parametrar man får skicka över mellan maskinerna.

Senare i detta dokument kommer vi att beskriva alla delblock som omfattar vår lösning på sökmaskinen vilken är implementerad i programmeringsspråket Erlang.

Nedan följer en bild som illustrerar hur sökmaskinen är uppbyggd och på vilka sätt den kan nås. Underhållet av databasen görs via en web-browser.

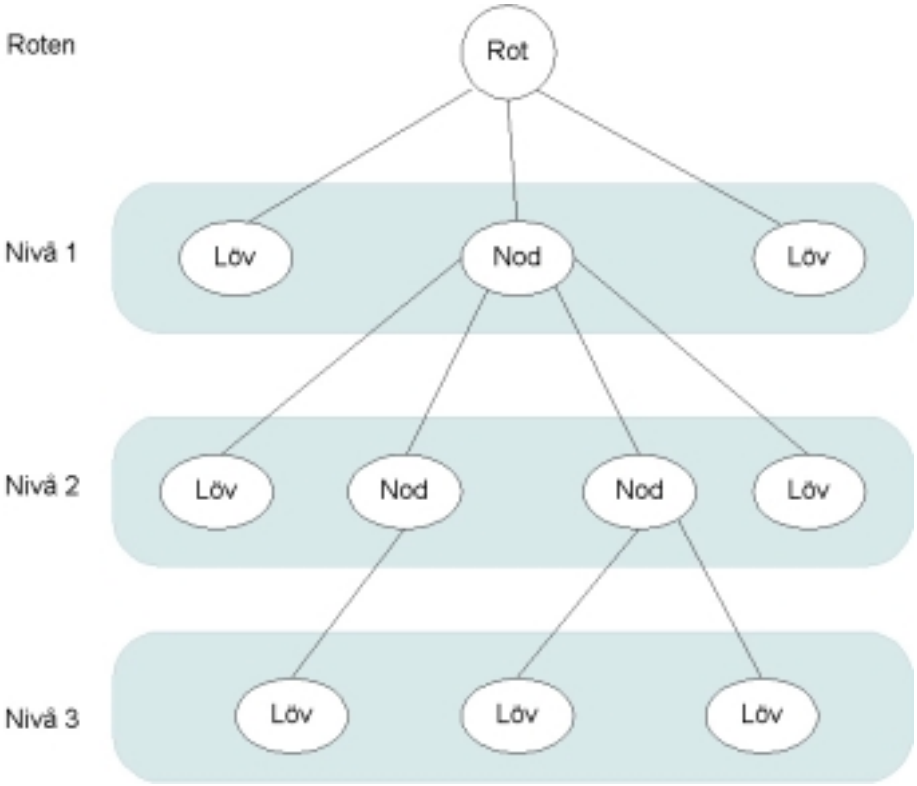
Contact Brokers System, Search Engine:



Figur.13

6.2 Analys

Vad är det som vi vill presentera? Hur ska databasen vara uppbyggd? Efter diskussioner med handledare och arbetskamrater på Ericsson beslöt vi oss för att bygga upp hela systemet i nivåer med hjälp av en trädstruktur, illustrerad nedan:



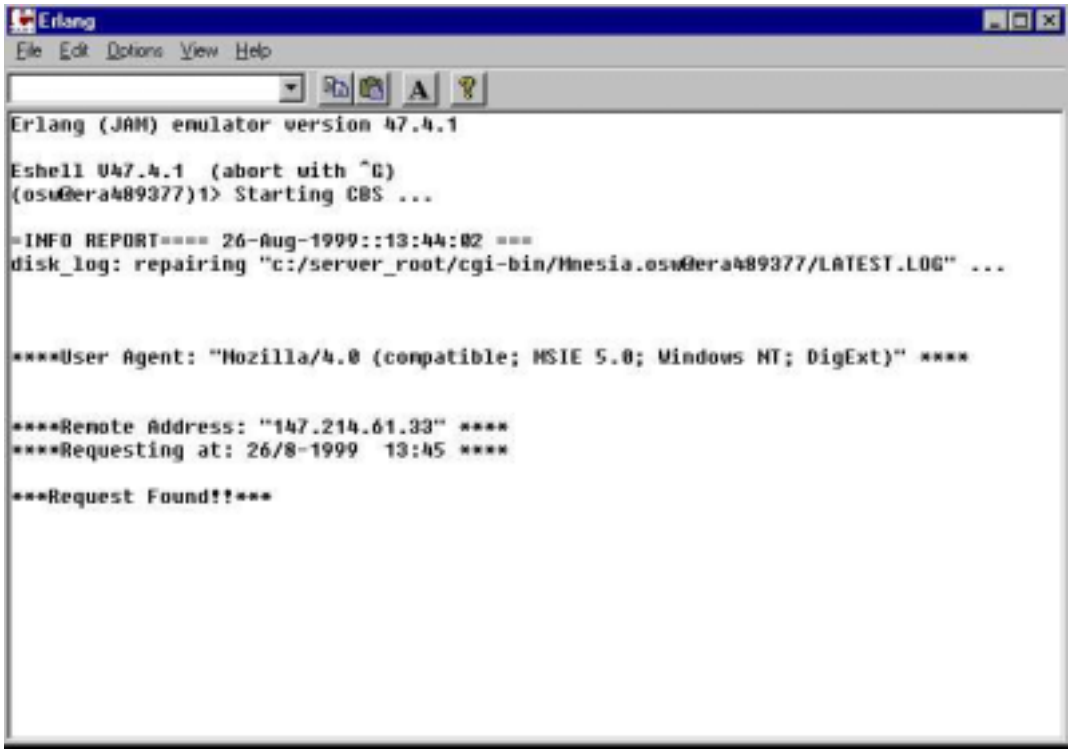
Figur.14

Vid den första sökningen i databasen ska alltid roten av trädet nås och det som skickas tillbaka till klienten är länkar till nästa nivå i trädet. På detta sätt åstadkoms ett enkelt sätt att söka igenom databasen från en WAP-telefon. Sökningen ska också kunna göras med fritextsökning och resultatet ska presenteras i form av länkar till alla poster i databasen som fick en träff.

6.3 Design

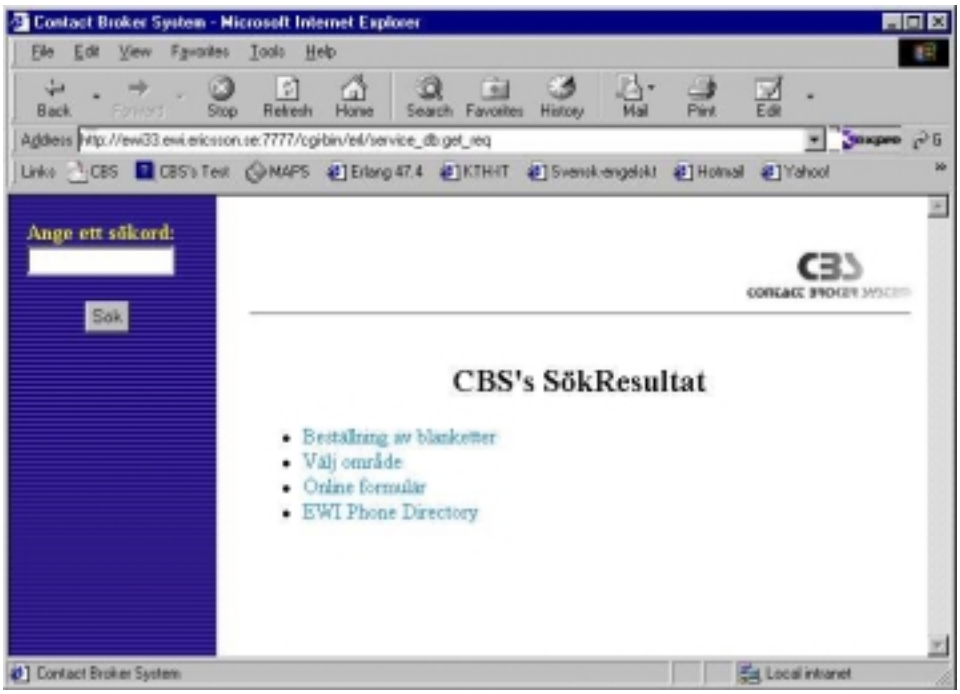
Sökmaskinen ska underlätta sökningarna som görs från en WAP-telefon. När den används mot en web-browser ska användargränssnittet förbättras och sökresultaten visas på ett mer utförligt sätt än vad det gör på en telefondisplay.

Systemadministratorn kan följa händelserna i sökmaskinen genom att kontrollera dessa i ett Erlang-kommandofönster, vilket kan se ut på följande sätt:



Figur.15

För att kunna testa sökmaskinen har vi lagt in EWI:s telefonbok i databasen. När man accessar servern från en web-browser görs en *request* över roten av databasen. En meny från trädets nästa nivå genereras i en HTML-sida som skickas tillbaka till klienten. Den aktuella rotmenyn ser ut på följande sätt:



Figur.16

Samma meny kan ses i en WAP-telefon, där man i stället för att generera en HTML-sida, har genererat en WML-*deck* (”WML-sida”). I figur.17 och 18 visas hur det skulle se ut i en WAP-telefon och vi använder oss då av ett verktyg som Ericsson har utvecklat för att simulera en WAP-telefon. Den heter WapIDE (se avsnitt 5.6) och består av flera (hittills fyra) utvecklingsverktyg av vilka vi använder WAP-browsern för att presentera vår sökmaskin.

Startfönster till CBS:

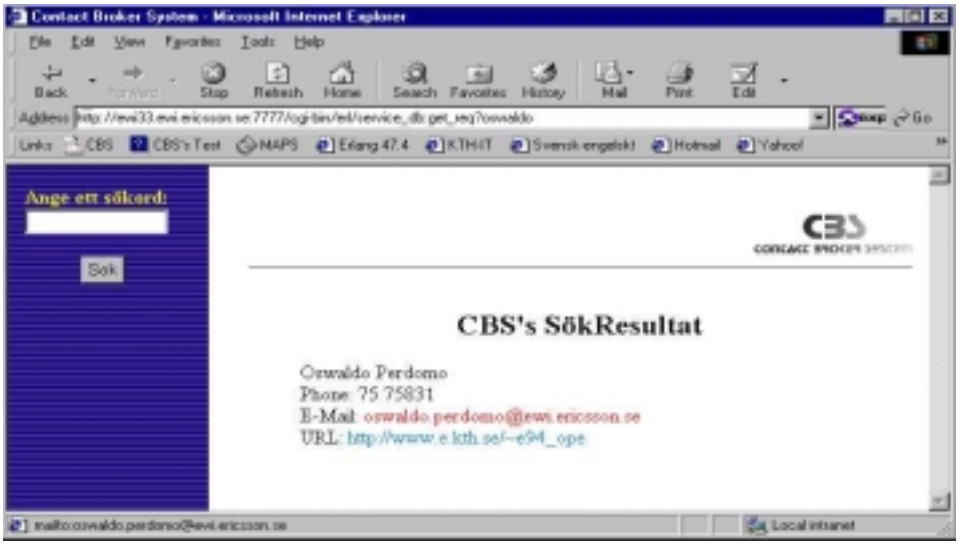
Figur.17

Sökmeny från rotnivån:

Figur.18

Menyerna som presenteras i båda fallen (i web- och WAP-browsern) är länkar till nästa nivå i sökträdet. Om man klickar på någon länk görs det en ny *request* till servern och det genererar antingen en ny meny med

nästa nivå i sökträdet eller, i de fall där det är sista nivån i trädet (ett löv), så genereras en sida med den information (eller tjänst) som finns där. Detta illustreras nedanför i figur.19:



Figur.19

Samma resultat i en WAP-telefon i figur.20:

Figur.20

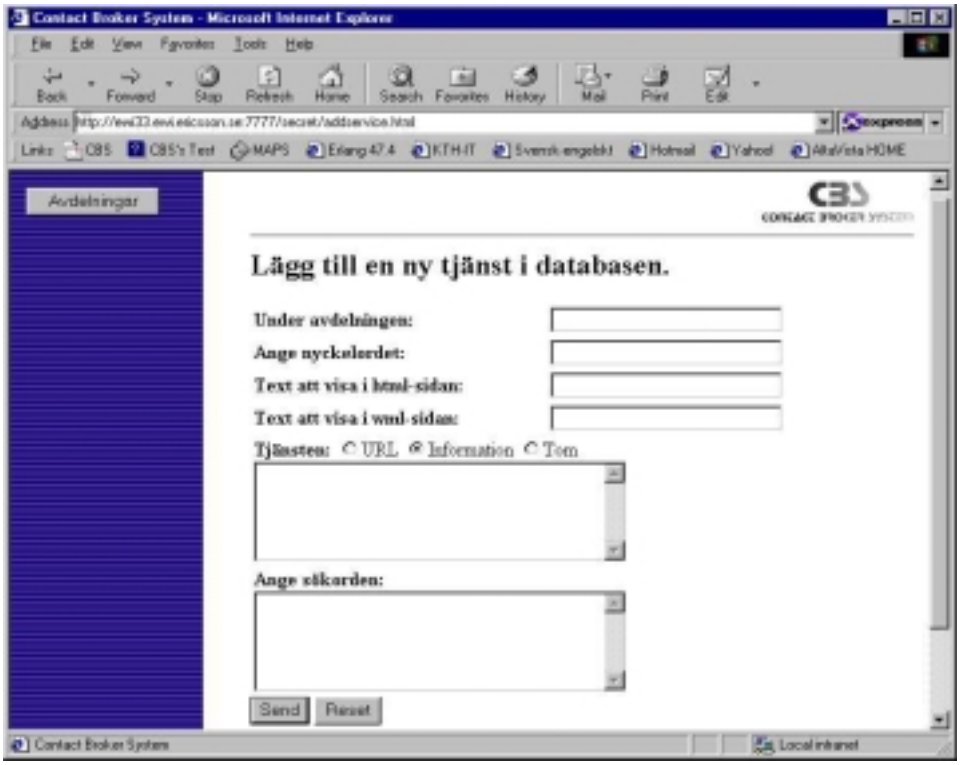
Vid fritextsökning ska ett sökord anges. I web-browsern finns det ett textfält för detta längst upp till vänster i fönstret. I WAP-telefonen kan detta ses som ett val sist i menyn. Trycker man på ”Sök” kommer man till ett fönster där ett sökord kan anges och en *request* för detta ord skickas iväg genom att trycka på YES-knappen. Detta kan ses i bilden nedan:

Figur.21

När en *request* från en fritextsökning kommer in till servern genereras antingen en meny med alla de poster i databasen som fick träff eller ett felmeddelande om sökordet inte gav någon träff.

Databasen behöver underhållas och för detta har vi gjort ett gränssnitt i HTML. Hela databasen underhålls från en web-browser. Underhållet är begränsat till att lägga in nya tjänster (eller noder) i databasen, att ändra i befintliga poster i databasen och att ta bort poster ifrån databasen. För att få tillgång till underhållssidorna måste man ange en login och ett lösenord, så att inte obehöriga kan göra ändringar i databasen.

Om man loggar in på sidan där man lägger till nya poster ser gränssnittet ut på följande sätt som i figur.22:



Figur.22

Det som fylls i på den här sidan sänds tillbaka till servern och sparas i databasen. Det som bestämmer på vilken nivå i trädstrukturen som den nya posten ska hamna, är inmatningen i ”Under avdelningen”, d.v.s. här matas en förälder in till posten.

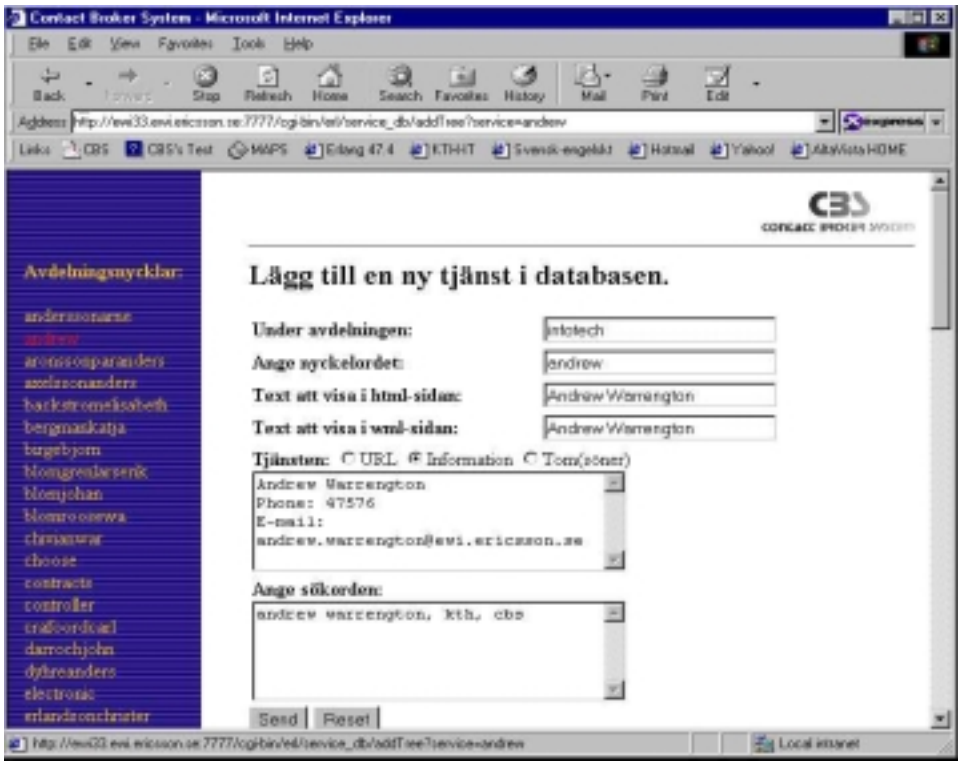
Den andra inmatningen, ”Ange nyckelordet”, ska vara ett unikt namn på den här nya posten. Med unikt namn menas att namnet inte upprepas i mängden av nyckelord från databasen.
Följande två inmatningar är den text som ska visas som länk till den här posten, när menyerna genereras.

Under ”Tjänsten”, kan man välja vilken sort av tjänst som posten ska leverera. Med ”URL” menas, att det i textarean ska finnas en URL, det kan vara en URL till ett formulär, till andra web-sidor, o.s.v. Väljer man ”Information” innebär detta att innehållet i textarean är just den informationen som ska visas när man accessar den här posten i databasen. Med ”Tom” menas att textarean lämnas tom. Detta används för att bilda

en son i trädstrukturen som vid ett senare tillfälle kan bli förälder. Med andra ord är detta inget löv, som levererar en tjänst, utan den bara länkar vidare till andra nivåer av trädstrukturen.

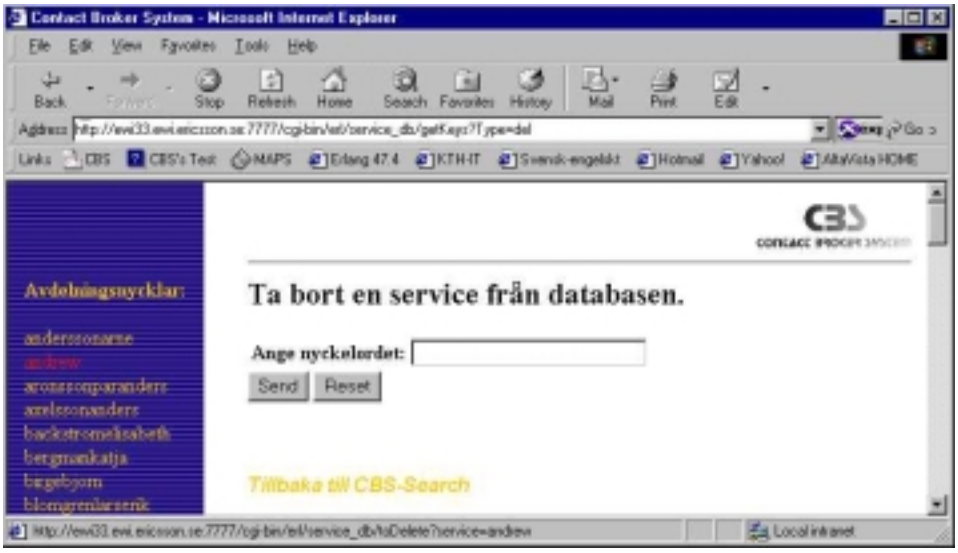
I textarean för ”Ange sökorden” skrivs vilka ord som ska ge träff på den här posten vid fritextsökning.

Om man trycker på knappen längst upp till vänster, ”Avdelningar”, får man tillbaka alla nyckelord som finns i databasen. Klickar man sedan på ett av dem visas den data som finns sparad i databasen för just den valda posten. Detta illustreras nedan:



Figur.23

När man loggar in på sidan där man tar bort poster ifrån databasen, ska man ange det nyckelord för den posten i databasen som man vill ta bort. Ett annat sätt är att trycka på knappen märkt ”Nyckelorden” då fås tillbaka alla nyckelorden i databasen, sedan är det bara att klicka på det som man vill ta bort, se figur.24.



Figur.24

6.4 Implementering av web-servern

Erlangs bibliotek innehåller en modul som implementerar en web-server enligt HTTP 1.0 specifikationen. Modulen heter ”*httpd*” och är en av många moduler som finns tillgängliga i containern ”*Internet Services Applications (INETS*)”[8].

Httpd-servern implementerar ett antal applikationer, av vilka kan nämnas *Secure Socket Layer (SSL), Common Gateway Interface (CGI), User Authentication* m.m.
Servern är kompatibel med *Apache*-servern och kan konfigureras enligt de direktiv som *Apache*-servern använder.

För att starta servern måste en konfigurationsfil editeras (det finns en redan som default) och ett portnummer väljas. Portnumret vi har valt är 7777 och då heter vår konfigurationsfil 7777.conf.
När man startar Erlang är det lämpligast att starta det med en flagga som informerar om att *Inets*-modulen ska användas, samt sökvägen (”*path*”) till konfigurationsfilen, 7777.conf i detta fall. Man kan bortse från att starta Erlang på detta sättet, men då måste en del andra kommandon anges i Erlangkommandofönster.

Erlang startas på följande sätt:

```
> erl.exe -config inets
```

Där *inets* är namnet på filen som anger konfigurationsfilen, nämligen *inets.config*. Den ser ut så här i vårt fall:

```
[ { inets, [{services,
        [{httpd, "/server_root/conf/7777.conf"}]}]}].
```

Sedan är det bara att starta *inets*-applikationen i erlangskommandofönstret med följande kommando:

```
1> application:start(inets).
```

Nu är servern igång och kan nås på adressen: http://my.ip.address:7777. Detta förutsätter att det finns en katalogstruktur som matchar de kataloger som anges i konfigurationsfilen, 7777.conf.

6.5 Konstruktion av databasen

Databasen är gjord i *Mnesia DataBase Management System* (Mnesia DBMS) som är beskriven i avsnittet 3.2.

Databasen är uppbyggd m.h.a. *records*. Vi använder oss av en record som innehåller alla de fält med den informationen som behöver sparas. Här nedan presenteras det record som vår databas baseras på:

```
-record(databasen, {baseKey,
                    html_title,
                    wml_title,
                    keywords,
                    parents,
                    sons,
                    service}).
```

Där

- ”databasen” är namnet på databasen.
- ”*baseKey*” är fältet där nyckelordet ska sparas.
- ”HTML_title” anger rubriken som ska presenteras i en web-browser.
- ”WML_title” här sparas rubriken som ska visas upp i en WAP-telefon.
- ”*keywords*” är en lista med alla de ord som ska ge träff på den här posten när man gör en fritext-sökning.
- ”*parents*” är en lista med alla föräldrarna till den här posten.
- ”*sons*” är en lista med alla barn till den här posten.

- ”service” är en lista som innehåller den typ av tjänst som den här posten levererar:

  - Tomt, om det bara är en nod som leder till andra noder, d.v.s. ingen tjänst.
  - En URL, om den här posten exempelvis leder till en sida med formulär eller en annan URL som innehåller en bättre presentation för denna tjänst.
  - Information, om denna post innehåller den information som ska presenteras antingen i webb-browsern eller i wap-telefonen.

För att kunna testa sökmaskinen har vi laddat upp databasen med EWIs telefonkatalog. Nedan kan ses hur en bit av databasen ser ut:

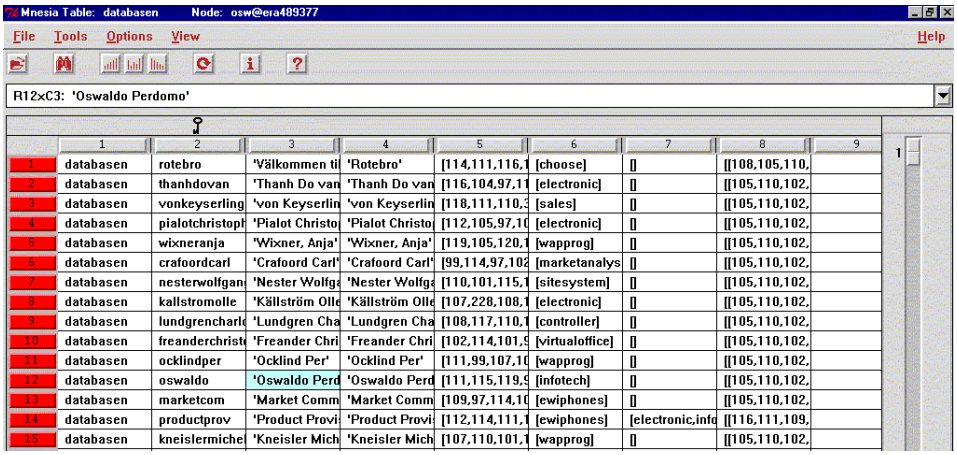
```
{tables,[{databasen,[baseKey,
                     html_title,
                     wml_title,
                     keywords,
                     parents,
                     sons,
                     service]}]}.

{databasen,oswaldo,
           'Oswaldo Perdomo',
           'Oswaldo Perdomo',
           "oswaldo perdomo cbs erlang wap",
           [infotech],
           [],
           ["infoOswaldo Perdomo\r\nPhone: 75 75831\r\nE-mail:
             oswaldo.perdomo@ewi.ericsson.se\r\nURL:
             http://www.e.kth.se/~e94_ope"]}.

{databasen,andrew,
           'Andrew Warrengton',
           'Andrew Warrengton',
           "andrew warrengton, cbs, erlang",
```

[infotech],
[],
["infoAndrew Warrengton\r\nPhone: 40 47 57 6\r\nMobile: 070-48 56 4 34\r\nE-mail: andrew.warrengton@ewi.ericsson.se\r\nPhoto: http://inside2.ewi.ericsson.se/ContactUs/ewi_staff"]}.

Mnesia hanterar varje post i databasen som en tupel. En mera läsbar databas kan skapas m.h.a. tabeller. Här använder vi Erlangs TV-modul (Table Visualizer)[9]:



	1	2	3	4	5	6	7	8	9
1	databasen	rotebro	'Välkommen ti	'Rotebro'	[114,111,116,1	[choose]	[]	[[108,105,110,	
2	databasen	thanhdovan	'Thanh Do van	'Thanh Do van	[116,104,97,11	[electronic]	[]	[[105,110,102,	
3	databasen	vonkeyserling	'von Keyserlin	'von Keyserlin	[118,111,110,3	[sales]	[]	[[105,110,102,	
4	databasen	pialotchristoph	'Pialot Christo	'Pialot Christo	[112,105,97,10	[electronic]	[]	[[105,110,102,	
5	databasen	wixneranja	'Wixner, Anja'	'Wixner, Anja'	[119,105,120,1	[wapprog]	[]	[[105,110,102,	
6	databasen	crafoordcarl	'Crafoord Carl'	'Crafoord Carl'	[99,114,97,102	[marketanalys	[]	[[105,110,102,	
7	databasen	nesterwolfgan	'Nester Wolfga	'Nester Wolfga	[110,101,115,1	[sitesystem]	[]	[[105,110,102,	
8	databasen	kallstromolle	'Källström Olle	'Källström Olle	[107,228,108,1	[electronic]	[]	[[105,110,102,	
9	databasen	lundgrencharlo	'Lundgren Cha	'Lundgren Cha	[108,117,110,1	[controller]	[]	[[105,110,102,	
10	databasen	freanderchriste	'Freander Chri	'Freander Chri	[102,114,101,9	[virtualoffice]	[]	[[105,110,102,	
11	databasen	ocklindper	'Ocklind Per'	'Ocklind Per'	[111,99,107,10	[wapprog]	[]	[[105,110,102,	
12	databasen	oswaldo	'Oswaldo Perd	'Oswaldo Perd	[111,115,119,9	[infotech]	[]	[[105,110,102,	
13	databasen	marketcom	'Market Comm	'Market Comm	[109,97,114,10	[ewiphones]	[]	[[105,110,102,	
14	databasen	productprov	'Product Provi	'Product Provi	[112,114,111,1	[ewiphones]	[electronic,info	[[116,111,109,	
15	databasen	kneislermiche	'Kneisler Mich	'Kneisler Mich	[107,110,101,1	[wapprog]	[]	[[105,110,102,	

Figur.25

Databasen är uppbyggd i en trädstruktur. Varje post håller reda på vem eller vilka dess föräldrar är samt vilka barn den har. Vid insättning av en ny post behövs det anges vem dess förälder ska vara. Vid bortagning av posten P ska Ps föräldrar bli föräldrar till Ps barn.

6.6 Kommunikation mellan klient och server

Vi har skapat en URL på serverdatorn på porten 7777 och använt ett CGI-*script* för att styra kommunikationen mellan servern och klienten. CGI-*scriptet* är skrivet i Erlang och hanterar *request* från web-browsern och WAP-telefonen. Senare under implementeringen har vi lagt till ett annat CGI script som hanterar *xml-request* från en MAPS-maskin.

Alla förfrågningar som görs till sökmaskinen hanteras i princip på samma sätt. Den största skillnaden i behandlingen sker i blocket ”*Presentation Manager*” där det gäller att ta emot ett *request* och skicka iväg respektive svar som kan visas i den sorts browser som klienten använder. I alla de resterande blocken i designen behandlas alla slags *request* på ett mycket liknande sätt. Nedan förklaras hur ett *request* hanteras av servern.

6.6.1 WML/HMTL-request

Det första som sker när ett *request* når servern är identifikation av vilken sorts maskin som den kommer ifrån, d.v.s. en WAP- eller en web-browser (WML- eller HTML-request). Detta görs genom att kontrollera vilken ”*User Agent*” som klienten använder sig av. Den variabeln sätts av klientens browser och sänds till servern vid *request*. Sedan återstår bara att identifiera det sökord klienten söker. Här är det mycket användbart att veta att WML-transporten görs m.h.a. HTTP-protokollet. Detta innebär att både WML- och HTML-*request* ser ut på samma sätt, nämligen:

http://192.36.156.22:7777/cgi-bin/erl/service_db:get_req?oswaldo

CGI-modulen heter ”*service_db*” och innehåller funktionen ”*get_req*” vilken förväntar sig en parameter (sökord), i detta fall ”oswaldo”. Sökordet kan plockas fram m.h.a. Erlangs *httpd-parser*. Följande kod är en del av funktionen *get_req*, där man kan se hur sökordet tilldelas variabeln ”*Seek*” och om sökordet saknas, tilldelas ”*Seek*” atomen ”*root*”, vilket innebär att sökningen görs på rotnivån.

get_req(Env, Input) ->

```
case req_type(Env) of
      wml_query ->
        case httpd:parse_query(Input) of
             [{[],[]}] ->
               get_answer(root, wml_query);
             [{Seek,_}] ->
               get_answer(list_to_atom(Seek) ,wml_query),
                  …
```

Från ”*get_req*” anropas en annan funktion, ”*get_answer*”. I ”*get_answer*” känner vi till sökordet och vilken sorts request som har gjorts, så det är här vi gör WML- eller XML- eller HTML-formateringen, samt gör en sökning i mnesiadatabasen för att ta reda om sökordet finns med eller inte. En bit av denna funktion kan ses här nedan:

```
get_answer(Requested, QueryString)->
   Fun = fun()->
           mnesia:read({databasen, Requested})
           end,
   {_, Found}= mnesia:transaction(Fun),
   case QueryString of
      wml_query ->
        if
        Found == [] ->
           %% Sökordet fanns inte med i databasen
                  …
        true ->
           %% Här sätts några variabler
                  …
           case Requested of
             root ->
                   [wml_head(),
                    wml_intro(),
                    wml_select_head(),
                    getLinks(Sons, [], RecFound, wml_query),
                    wml_friText(wml_query),
                    wml_select_bottom(),
                    wml_bottom()
                   ];
              Others -> …
```

Som man kan se så använder denna funktion en del andra funktioner, vilka i detta fall returnerar WML-kod. Liknande funktioner finns för både HTML och XML. Här nedan presenteras några av dessa funktioner:

```
wml_head()->
   "Content-type: text/x-wap.wml\r\n
     <?xml version=\"1.0\"?>
     <!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"
               \"http://www.wapforum.org/DTD/wml_1.1.xml\">
     <wml>".

wml_bottom()->
   "</wml>".

wml_card(Text, Href)->
   "<card>
      <do type=\"accept\">
         <go href=\"" ++ Href ++ "\"/>
      </do>
      <p>"
         ++ Text ++
     "</p>
   </card>".
```

En annan viktig funktion som anropas är ”*getLinks*”. Den har till funktion att leverera en lista med länkar till alla söner som sökordet ledde till.

Exempelvis skulle ett *request* med respektive svar se ut på följande sätt:

Request (default = ’root’):

http://192.36.156.22:7777/cgi-bin/erl/service_db:get_req

Svar:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
                  "http://www.wapforum.org/DTD/wml_1.1.xml">
```

```
<wml>
 <card id="menu">
  <p>
  <select>
   <option onpick="http://192.36.156.22:7777/cgi-bin/erl/
       service_db:get_req?reservation">Blanketter</option>
   <option onpick="http://192.36.156.22:7777/cgi-bin/erl/
       service_db:get_req?choose">V&#228;lj omr&#229;de</option>
   <option onpick="http://192.36.156.22:7777/cgi-bin/erl/service_db:
       get_req?onlineForms">Online formul&#228;r</option>
   <option onpick="http://192.36.156.22:7777/cgi-bin/erl/
       service_db:get_req?ewiphones">Phone Directory</option>
   <option onpick="#search">S&#246;kning</option>
  </select>
  </p>
 </card>

 <card id="search">
  <do type="accept">
   <go href="http://192.36.156.22: 7777/cgi-bin/erl/
       service_db:search?$seek"/>
  </do>
  <p>
   Ange s&#246;kordet:<br/>
   <input type="text" name="seek"/>
  </p>
 </card>
</wml>
```

6.6.2 Fritext-sökning

En förfrågan om fritext-sökning kan göras genom att anropa funktionen ”*search*” som finns med i CGI-modulen ”*service_db*”. Detta *request* ser ut på följande sätt:

http://192.36.156.22:7777/cgi-bin/erl/service_db:search?seek=kalle

Här söks ordet ”kalle” på fältet ”*keywords*” i varje post av databasen. De poster som innehåller sökordet får träff och en lista med länkar till alla dessa poster returneras för att sedan skickas som svar till klienten.

En bit av koden för denna funktion presenteras nedan:

```
search(Env, Input)->
   [{"seek", KeyWord}]= httpd:parse_query(Input),
   Fun= fun()->
            mnesia:all_keys(databasen)
end,
   {atomic, Keys} = mnesia:transaction(Fun),
   search(toLower(KeyWord), Keys, req_type(Env)).


search(KeyWord, Keys, QueryType)->
   Found= findWords(KeyWord, Keys),

   case QueryType of
      wml_query ->
         if
Found == [] ->
            %% Sökordet fanns inte med i databasen
         true ->
            PushLinks= pushLinks(Found, wml_query),
            [wml_head(),
             wml_select_head(),
             PushLinks,
             wml_friText(wml_query),
```

```
        wml_select_bottom(),
        wml_bottom()
       ]
    end;
html_query -> …
```

6.6.3 XML-request

XML-*request* görs mot ett annat CGI-*script*. XML transporteras också m.h.a. HTTP-protokollet så ett *request* ser ut på följande sätt:

http://192.36.156.22:7777/cgi-bin/erl/engine:start?<?XML version=\"1.0\"?><!DOCTYPE

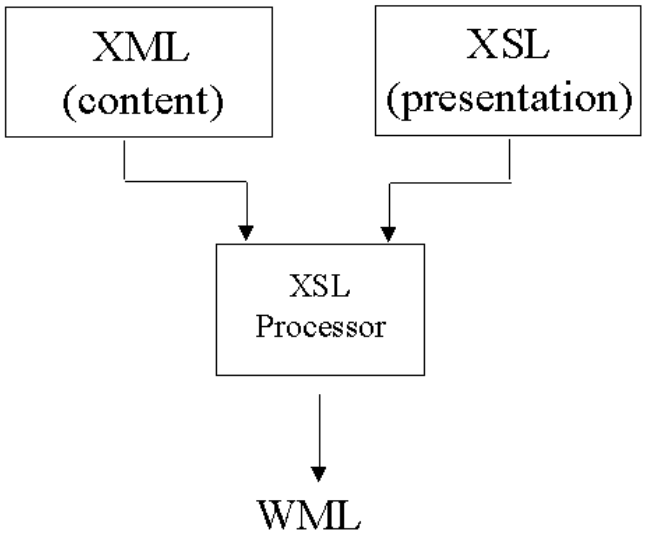
Hela XML läggs till som ”*input*”. Detta medför att parsningen blir betydligt svårare. Vi använder en XML-*parser* designad av Joe Armstrong [1], då han jobbade för Ericsson Telecommunications AB. Denna *parser* returnerar en trädstruktur och sedan är det bara att söka igenom trädet efter de element som behövs.

En XML-*request* i CEI-gränssnittet kan se ut på följande sätt:

```
<?XML version=\"1.0\" ?>
<!DOCTYPE invoke_application
  SYSTEM \"/usr/local/erlang/server_root/src/XML/DTD/sup.dtd\">
<invoke_application>
<head>
  …
</head>
<call>
  <function_id>service_db</function_id>
  <target_id>get_req</target_id>
  <param>
    <name>oswaldo</name>
    <value>none</value>
  </param>
```

</call>
</invoke_application>"

Den enda information vi behöver från detta XML finns lagrad innanför taggarna <call>…</call>. Med den informationen kan vi sedan göra anropet: *service_db:get_req(oswaldo)* som sätter igång en sökning på samma sätt som beskrevs längre upp i detta avsnitt.
Sökresultatet struktureras i ett XML-dokument som sedan skickas tillbaka till klienten, i detta fall MAPS-servern, som i sin tur omvandlar XML-dokumentet till WML m.h.a. en XSL-processor. Användningen av XSL-processorn ingår inte i vårt examensarbete och den kommer vi inte att beskriva. Det som är relevant för vårt arbete är att XSL-processorn kräver två parametrar: en XML och en XSL(se kapitel 4.5). XML-dokumentet har genererats av sökmaskinen och XSL-dokumentet har vi skrivit på så sätt att XSL-processorn genererar ett WML-*deck.*



Figur.26

På nästa sida följer XML som fås som svar till sökordet ”oswaldo”. Sökresultatet finns mellan taggarna <response>…</response> och det är denna information som senare används av XSL-processorn för att bilda upp ett WML-*deck.*

```
<?xml version="1.0" ?>
<eng_resp>
 <head>
  …
</head>
 <response>
  <cbs_resp Id="info">
   <contact Id="info">
    <target_id>service_db</target_id>
    <function_id>get_req</function_id>
    <parameter>
     <name>root</name>
     <value>none</value>
    </parameter>
    <title>
     <newline>Oswaldo Perdomo</newline>
     <newline>Phone: 75 75831</newline>
     <newline>E-mail: oswaldo.perdomo@ewi.ericsson.se</newline>
     <newline>Url: http://www.e.kth.se/~e94_ope</newline>
    </title>
   </contact>
  </cbs_resp>
 </response>
</eng_resp>
```

7 Framtida arbete

För att få ett komplett system med de specifika krav som anges i avsnitt 1.2 måste man arbeta vidare med delblock 2 (*assignment support system*) och delblock 3 (*contact support system*) som av tidsbrist ej ingår i detta arbete.

Användargränssnittet kan göras bättre med t.ex. bilder och användarprofiler som exempelvis kan lagra vilken sorts WAP-terminal användaren har, senast sökta poster, o.s.v.

Bättre stöd för att kunna uppdatera de olika ändringar som sker på WAP-fronten. Eftersom WML är under utveckling och det sker ständiga ändringar på WML-syntaxen, måste vår applikation uppdateras för att kunna arbeta med nyare versioner av WML.

Senare, för att få en bättre sökmaskin måste man tänka på att lägga in RDF-syntaxen vilket medför en klar förbättring i jämförelse med om data presenteras bara med XML. Med hjälp av RDF kan man göra bättre och snabbare metadatasökning.[23]

8 Slutsatser

Målet med att skapa en systemlösning för CBS har uppnått med vissa reservationer för förbättringar och framtida kompletteringar.

WAP kommer att ge stora möjligheter till nya spännande tjänster för mobila apparater.

CBS underlättar sökningar som görs från en WAP-terminal.

CBS kan köras både i fasta och trådlösa nätet med undantag för dess underhåll som måste göras från en web-browser.

Systemet skulle kunna tillämpas inom många område, främst inom organisationer där det är svårt att hitta en person med en vis ansvar eller att direkt beställa tjänster från företaget.

9 Källförteckning

[1] Concurrent Programming in ERLANG, Joe Armstrong, Robert Virding, Claes Wikström, Mike Williams, second edition, published by Prentice Hall Europe, 1996.

[2] Erlang System Architecture
http://www.erlang.org/doc/doc/doc/system_architecture_intro/part_frame.html

[3] Erlang Design Principles
http://www.erlang.org/doc/doc/doc/design_principles/part_frame.html

[4] Erlang System Principles
http://www.erlang.org/doc/doc/doc/system_principles/part_frame.html

[5] Mnemosyne
http://www.erlang.org/doc/doc/lib/mnemosyne-1.0.1/doc/index.html

[6] Mnesia
http://www.erlang.org/doc/doc/lib/mnesia-3.6/doc/index.html

[7] Mnesia Session
http://www.erlang.org/doc/doc/lib/mnesia_session-1.1/doc/index.html

[8] Internet Services Application (INETS)
http://www.erlang.org/doc/doc/lib/inets-2.3/doc/index.html

[9] The Table Visualizer (TV)
http://www.erlang.org/doc/doc/lib/tv-1.2.4/doc/index.html

[10] Extensible Markup Language (XML) 1.0
http://www.w3.org/TR/REC-xml

[11] Extensible Stylesheet Language (XSL) 1.0
http://www.w3.org/TR/WD-xsl

[12] HTML 4.0 Specification
http://www.w3.org/TR/REC-html40/

[13] Microsofts XSL Reference
http://msdn.microsoft.com/xml/xsl/reference/start.asp

[14] A User's Guide to Style Sheet
http://msdn.microsoft.com/workshop/author/css/css.asp

[15] Microsofts XML Tutorial
http://msdn.microsoft.com/xml/tutorial/

[16] Swick Ralph. Metadata and Resource Description
http://www.w3.org/Metadata/

[17] The XML FAQ
http://www.ucc.ie/xml/

[18] Resource Description Framework (RDF) Model and Syntax Specification
http://www.w3.org/TR/1999/PR-rdf-syntax-19990105/

[19] Wireless Application Protocol Architecture Specification
http://www.wapforum.com/

[20] Wireless Markup Language Specification
http://www.wapforum.com/

[21] http://www.sml.stockholm.se/dokument/pressmedia.html

[22] Business Arena Stockholm, B.A.S
c/oSML - Stockholms Mark och Lokaliseringsbolag.
Suzanne Liljegren, projektledare.
Box 12712
112 94 Stockholm
Tel: 08-7858080

[23] Jan Hedin, Ericsson Business Consulting, Tel: 070-5430560

[24] XML Guide
http://www.geocities.com/SiliconValley/Peaks/5957/10minxml.html

[25] HTML Guide
http://www.nada.kth.se/~viggo/svenskMosaic/HTMLPrimer.html
http://www.atiger.pp.se/sida.html
http://www.ekdahl.org/build.htm
http://www.hut.fi/~jkorpela/HTML3.2/

[26] XML- språket som vidgar Webben
http://www.xml-forum.com/fakta/xml-intro-cnet.shtml

[27] WapIDE
http://mobileinternet.ericsson.se/.