ARCADA

Utveckling av ett webbforum med innehållshanteringssystemet Drupal

Christoffer Lindqvist

Examensarbete

Informationsteknik

2010

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	2791
Författare:	Christoffer Lindqvist
Arbetets namn:	Utveckling av ett webbforum med innehållshanteringssystemet Drupal
Handledare (Arcada):	Johnny Biström
Uppdragsgivare:	Nord Software Ltd.

Sammandrag:

Detta examensarbete består av två huvuddelar. Den första delen är en teoretisk analys av innehållshanteringssystem i allmänhet och en forskning i hur det populära innehållshanteringsverktyget Drupal är uppbyggt. Den andra delen av arbetet är en praktisk genomgång av hur funktionalitet för ett webbforum kan byggas upp med hjälp av Drupals ramverk.

Den teoretiska analysen undersöker vad ett innehållshanteringsverktyg är, vad det används till och vilka huvuddrag det i allmänhet består av. Sedan följer en noggrann undersökning av innehållshanteringsverktyget Drupal. Drupals arkitektur och funktionalitet ses över för att få en förståelse för hur ramverket kan utnyttjas för att bygga praktiska webbapplikationer. Även om ramverket inte är objektorienterat så är vissa objektorienterade koncept och designmönster implementerade. Dessa granskas också för att förstå dataflödet i systemet.

I den praktiska delen undersöks en del funktionalitet som ingår i webbforumet. Kunskapen från den teoretiska delen användes för att bygga de moduler som behövdes och för att integrera funktionaliteten med Drupal. Alla funktioner förklaras med hjälp av kodexempel för att understryka hur de har implementerats.

Nyckelord:	Drupal, Innehållshanteringssystem, Webbforum, Modul, Nod, Objekt orientering, Utvidga, Åsidosätta
Sidantal:	74
Språk:	Svenska
Datum för godkännande:	5.5.2010

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	2791
Author:	Christoffer Lindqvist
Title:	Development of a web community with the content management system Drupal
Supervisor (Arcada):	Johnny Biström
Commissioned by:	Nord Software Ltd.

Abstract:

This degree thesis consists of two main parts. The first part is a theoretical analysis of content management systems in general as well as a research in how the popular content management tool Drupal is built. The second part of the thesis is a practical review of how web community features can be built using the Drupal framework.

The theoretical analysis examines what a content management system is, what it is used for and which main features it generally consists of. After this the content management tool Drupal is closely examined. Drupals architectural design and features are reviewed to gain understanding of how the framework can be utilized to build practical web applications. Even though the framework is not object oriented some object oriented concepts and design patterns are adopted. These are also examined to understand the data flow in the system.

The practical part examines some features that where developed for the web community. The knowledge gained from the theoretical part was used to build the modules needed for the functionality to work together with Drupal. All features are explained with the help of code examples to emphasize how the different parts of the functionality have been implemented.

Keywords:	Drupal, Content management system, Community, Module, Node, Object orientation, Extend, Override
Number of pages:	74
Language:	Swedish
Date of acceptance:	5.5.2010

INNEHÅLL

Figurer

Förkortningar

API	Application Programming Interface
CMS	Content Management System
GPL	General Public License
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
LAN	Local Area Network
PHP	Hypertext Preprocessor
RBAC	Role-Based Access Control
RSS	Really Simple Syndication
URL	Uniform Resource Locator
XML-RPC	Remote Procedure Call protokoll som använder XML

1 INLEDNING

1.1 Utgångspunkt

Detta examensarbete är gjort som ett projekt för min nuvarande arbetsgivare Nord Software. Syftet med projektet är att skapa ett webbforum som är avsett för att diskutera aktuella ämnen och händelser inom olika områden, med andra ord ett webbforum där användare kan bidra till diskussioner och utmaningar genom att läsa, skriva och kommentera artiklar. Projektet är uppbyggt med innehållshanteringssystemet Drupal (version 6) och webbplatsen upprätthålls i LAMP (Linux, Apache, MySQL, PHP) miljö.

1.2 Målsättningar

Detta examensarbete är indelat i två huvuddelar, en teoretisk och en praktisk. Den teoretiska delen består både av att generellt analysera vad ett innehållshanteringssystem är samt av att forska i hur Drupal är uppbyggt och vilken funktionalitet systemet erbjuder. Den praktiska delen av arbetet består av att utveckla en del funktionalitet för webbplatsen med hjälp av Drupals ramverk.

1.2.1 Teoretiska

Till de teoretiska målsättningarna med arbetet hör att få en djupare förståelse i hur innehållshanteringssystemet Drupal är uppbyggt samt att kunna identifiera de problem och begränsningar som användningen av ramverket medför. Eftersom projektet går ut på att skapa ett webbforum så är ett mål också att redogöra varför Drupal är lämpligt just för denna typ av webbplatser. Drupal är inte ett objektorienterat ramverk men implementerar dock en del objektorienterade koncept och därför är ett av målen med arbetet också att utreda hur dessa är implementerade.

1.2.2 Praktiska

De praktiska målsättningarna är att klarlägga hur man på rätt sätt utvecklar moduler för Drupal och vad dessa moduler skall innehålla. Andra mål är att ta reda på hur man åsi-

dosätter Drupals standardfunktionalitet för att implementera egna anpassade lösningar samt hur man utnyttjar ramverket för att integrera ny funktionalitet för existerande moduler.

1.3 Begränsningar

Detta examensarbete kommer inte att omfatta den grafiska utformningen av webbforumet, utan kommer att fokusera på hur Drupals arkitektur kan utnyttjas för att skapa kraftfull funktionalitet.

2 INNEHÅLLSHANTERINGSSYSTEM

Det här kapitlet lägger grunden till att förstå hur ett CMS (Content Management System) fungerar och vilken nytta en webbapplikation har av att använda ett. Problemet med webbplatser, vilket leder till ett behov av innehållshanteringssystem, är den grad av komplexitet som till exempel användarhantering, mängden material eller underhåll för med sig.

2.1 Definition

Ett CMS är ett ramverk med verktyg för att skapa, redigera och hantera innehåll som visas på en webbplats. Detta ramverk kan hantera en stor del av webbplatsens funktionalitet, som till exempel reglera när innehållet visas, hur många gånger innehållet visas för en viss användare, och hantera hur innehållet samverkar med eller ansluter andra delar av webbplatsen. Ramverket möjliggör också för mindre tekniska personer att enkelt hantera och administrera innehåll på en webbplats utan omfattande kunskap om hur funktionaliteten är implementerad.

2.2 Funktionsprincip

En av de viktigaste funktionerna i ett CMS är att det skiljer webbplatsens design, innehåll samt dess affärslogik och funktioner, vilket gör det enkelt att ändra på någon självständig del av webbplatsen utan att påverka någon annan del. (Peacock, 2009:12)

Alla CMS använder sig av en viss typ av arbetsflöde som kan variera mellan olika system. Den grundläggande funktionaliteten för ett CMS kan dock generellt delas in i fyra kategorier:

- Skapandet av innehåll
- Innehållshantering
- Publicering
- Presentation

2.2.1 Skapandet av innehåll

CMS som är avsedda för webben har ofta inbyggda texteditorer och andra verktyg som hjälper användaren att skapa innehåll för webbplatsen på ett sätt som inte kräver några programmeringskunskaper eller andra tekniska färdigheter. Detta gör dessa system mycket attraktiva för många användare.

2.2.2 Innehållshantering

Skapat material sparas i en databas med annan relevant information som till exempel användaridentifikation, kategorier eller taggar. Vid det här skedet är inte själva materialet kopplat till någon som helst typ av presentationslager, utan materialet finns som rå text i databasen. Detta för att skapa den nivå av abstraktion som behövs för att dynamiskt kunna presentera materialet eller för att kunna ändra på någon del i systemet utan att påverka innehållet. Detta möjliggör också skapandet av mer anpassad funktionalitet. Genom att materialet är sparat i råformat är det enkelt att skriva funktionalitet för att till exempel skapa kopplingar mellan användare i systemet för att kunna dela material.

2.2.3 Publicering

Innehållshanteringssystem har ofta mycket avancerad publiceringsfunktionalitet som automatiskt kan applicera utseende och layout vid publiceringen. Systemet ser till att innehållet är konsekvent över hela webbplatsen och möjliggör en mycket hög standard på utseende. Detta gör också att användarna kan koncentrera sig på att skriva innehållet och lämna utseendet helt åt systemet. Vissa CMS har också en så kallad "draft" funktionalitet, vilket betyder att man kan spara sitt material utan att publicera det för allmänheten på webbplatsen. Detta kan vara en mycket användbar funktion om man inte hinner skriva färdigt det man höll på med och vill fortsätta senare.

I vissa CMS lösningar kan användare inte direkt publicera sitt material utan att det gått genom en process för att godkänna innehållet.

2.2.4 Presentation

Presentationslagret i ett CMS fungerar på så sätt att det efterfrågade materialet visas på webbplatsen med hjälp av på förhand definierade mallar. I vissa system kan användaren välja mellan mallar och i andra bestämmer ägaren av webbplatsen hur materialet formateras förrän det presenteras. Den slutliga webbsidan som visas för användaren byggs alltså dynamiskt upp utgående från materialet i databasen och design mallar. Varken materialet eller designmallarna ändrar utgående från att ett en sida renderas.

Beroende på arkitekturen ett system använder kan renderingen av materialet skötas på olika sätt. Allt material kan till exempel samlas ihop och presenteras. Vissa CMS har en inbyggd cache funktionalitet som kan användas för att lagra ofta renderade webbsidor i en databas för att snabba upp processen att visa materialet.

2.3 Grundläggande krav

Ett CMS är ett ramverk som erbjuder funktionalitet för att hantera material på olika sätt. Det är uppenbart att ett enda CMS inte går att tillämpa i alla situationer man stöter på. Arkitekturen på systemet berättar mycket om vad som systemet är ämnat för och för vad det inte är lika användbart. Det går dock att nämna några grundläggande krav för vad ett CMS skall innehålla.

2.3.1 Integritet

Trots begränsningarna av grundläggande webbprotokoll som används idag, behöver många webbsidor information genom användarinteraktion. Denna information måste kunna skyddas från kapning eller modifiering. Ett CMS bör hantera detta på ett sätt som gör det enkelt också för utvidgningar av systemet att skydda informationen. (Brampton, 2008:10)

2.3.2 Autentisering

Ramverket måste utforma grunderna för kontroll av användare via någon form av autentisering. Men systemet måste vara så flexibelt som möjligt för att minska koden som

behövs för att hantera kraven som kan omfatta allt från enstaka administrativa användare till att hantera hundratusentals olika användare samt olika system för autentisering. (Brampton, 2008:10)

2.3.3 Åtkomstkontroll

Åtkomstkontroll är något som alltid krävs, om så bara för att begränsa vem som kan konfigurera webbplatsen. Ofta krävs det mycket olika grupper av användare som tilldelas olika privilegier. De flesta är överens om att det bästa tillvägagångssättet är rollbaserad åtkomstkontroll (RBAC). Detta innebär att det är roller som beviljas behörigheter, och användare tilldelas roller. (Brampton, 2008:11)

2.3.4 Utvidgningsmöjlighet

Ett ramverk är användbart om det enkelt kan utvidgas. Det finns ingen enskild för användaren synlig funktionalitet som är nödvändig för varje webbplats, så i bästa fall tas alla funktioner bort från ramverket. Varje funktion som är synlig för användaren kan sedan läggas till som en utvidgning. När kraven för att bygga en webbplats anges, visar det sig att det finns flera olika typer av utvidgningsmöjligheter. En välkänd klassificering är i komponenter, moduler, insticksprogram och mallar. (Brampton, 2008:11)

2.3.5 Säkerhet och felhantering

Det finns många olika slags hot som är riktade mot webbplatser idag. För att vara effektiv, måste säkerheten vara inbyggd från början så att inte bara ramverket har bästa möjliga säkerhet, utan det ger också en bra miljö för att bygga säkra utvidgningar. Fel är betydande, både som ett användbarhetsproblem och som en potentiell säkerhetsrisk, så en standardiserad felhanteringsmekanism behövs också. (Brampton, 2008:11)

3 DRUPAL

Drupal är ett gratis, mycket modulärt innehållshanteringssystem licenserat under GPL (General Public License). Systemet tillåter enskilda individer eller stora grupper av användare att enkelt publicera, hantera och ordna ett brett utbud av innehåll på en webbplats. (Peacock, 2009:13)

Till skillnad från andra system är Drupal uppbyggt mer som ett innehållshanteringsramverk bestående av komponenter som kan användas som sådana eller konfigureras helt enligt eget behov. Denna design erbjuder stor flexibilitet samtidigt som den tillåter personer som inte är programmerare att skapa innehållsrika webbplatser. Den här typen av abstraktion och generalisering har en stor fördel jämfört med de mer traditionella uppgiftsorienterade tillvägagångssätten eftersom man kan skapa ny funktionalitet på webbplatsen genom att använda befintliga komponenter i stället för att utveckla helt nya. Denna flexibilitet medför även en viss grad av komplexitet, eftersom man måste förstå hur de olika komponenterna hänger ihop förrän man kan utnyttja dem till fullo. (drupal.org, 2010 c)

3.1 Historia

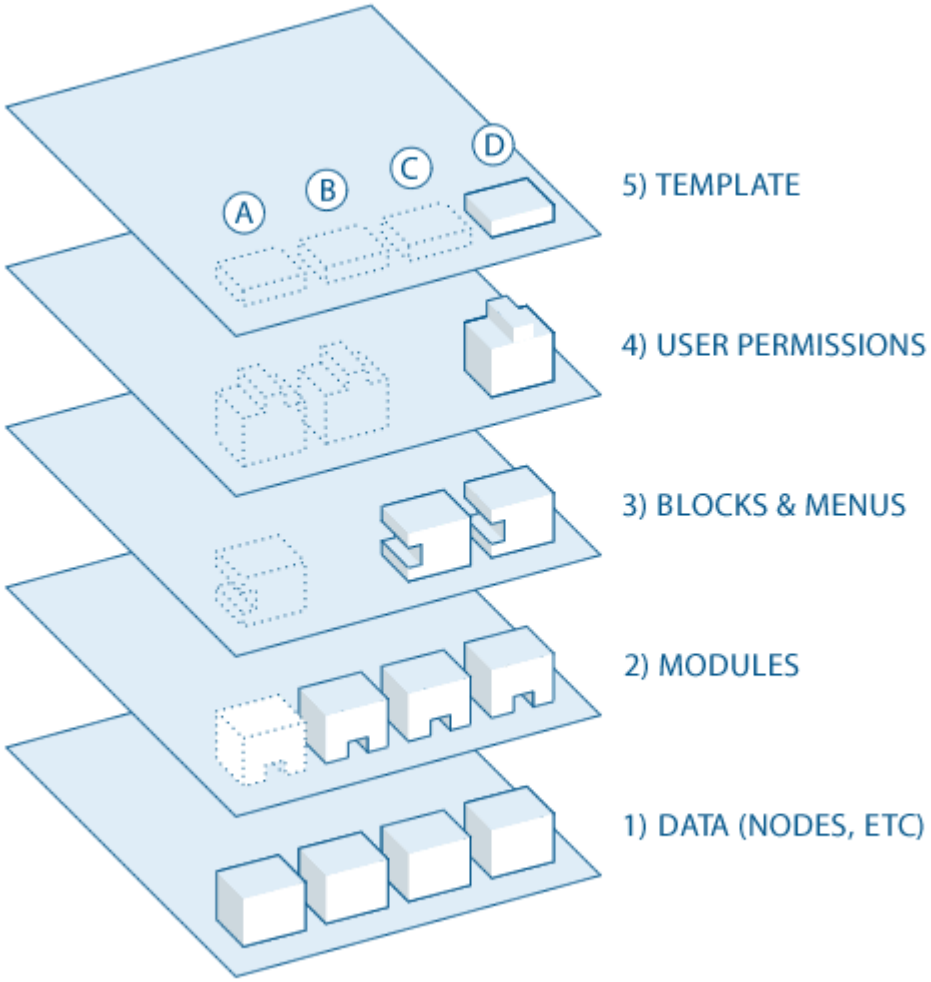
Projektet startades av Dries Buytaert (född den 19 november 1978 i Belgien) medan han studerade vid universitetet i Gent (drupal.org, 2010 b). Det började med en enkel nyhetssida som Buytaert satte upp på ett LAN (Local Area Network) med sina studiekompisar där de kunde gå in och skriva meddelanden till varandra för att hålla kontakten. Det var först efter att Buytaert examinerats från universitetet som han bestämde sig för att lägga upp sidan på Internet. Buytaert ville ursprungligen registrera sidan under namnet "dorp" som betyder by, men medan han letade efter lediga domännamn så blev det "drop" på grund av felstavning.

Efter att drop.org etablerat sig på Internet utvecklades sidan långsamt till en personlig experimentmiljö som drevs av diskussion och flödet av idéer. Dessa idéer prövades som nya tillägg direkt på drop.org.

Det var först i januari 2001som Buytaert bestämde sig för att dela ut programvaran bakom drop.org under namnet "Drupal". Syftet med detta var att andra skulle få möjligheten att använda och utvidga plattformen så att fler människor kunde utforska nya vägar för utvecklingen. Namnet Drupal , uttalas "droo-puhl", härstammar från det engelska uttalet av det nederländska ordet "druppel", vilket på engelska betyder "drop". (drupal.org, 2010 a)

3.2 Arkitektur

För att kunna förstå Drupal och utnyttja dess möjligheter är det viktigt att förstå hur systemet är uppbyggt och ha en överblick av dataflödet. Systemet är indelat i fem lager (Figur 1), och varje lager har sin egen funktion för att kunna skapa en nivå av abstraktion, ordning och flexibilitet.



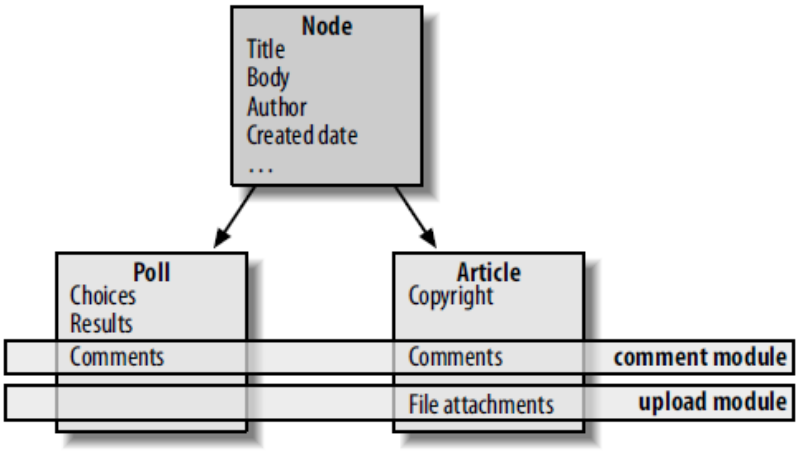
Figur 1. Illustration av Drupals arkitektur (drupal.org, 2010 c)

3.2.1 Noder

Grunden för hur Drupal fungerar sitter i hur data sparas i databasen. Noder är det som gör Drupal till ett abstrakt ramverk. En nod är i grund och botten ett generiskt objekt som representerar data. Nästan allt innehåll som skapas på webbplatsen, det må vara ett blogginlägg eller en nyhetsartikel, är det en instans av en nod. Förutom sina grundegenskaper så har alla noder en innehållstyp som definierar deras utvidgade funktionalitet. Med andra ord så ärver varje innehållstyp nodens egenskaper. Fastän alla innehållstyper är avsedda för olika ändamål så sparas grundnoden för alla typer i samma databastabell. Detta försäkrar att alla noder är konstruerade från samma utgångspunkt och kan hante-

ras på samma sätt. Det innebär också att varje nod typ kan anpassas för ett specifikt behov samt att deras gemensamma struktur kan användas för att organisera, söka och knyta ihop noder till andra noder. (Byron et al. 2008:10)

Eftersom denna abstraktion är så inbyggd i Drupal är det möjligt för utvecklare att enkelt utvidga en nods egenskaper genom att skriva en modul som möjliggör att mera information kan sparas utan att behöva oroa sig om nodens innehållstyp (Figur 2). På grund av nodernas gemensamma struktur är det möjligt att ha ett enhetligt administrationsgränssnitt i vilket man kan knyta ihop den utvidgade funktionalitet till en viss innehållstyp. (VanDyk, 2008:5)



Figur 2. Illustration av relationen mellan noder och moduler. (Byron et al. 2008)

Drupal distribueras med två stycken färdigt konfigurerade innehållstyper, sidor och historier. Dessa innehållstyper erbjuder ingen specifik funktionalitet utöver de som noden har. Enda skillnaden mellan dem är deras inställningar för att visa innehåll på webbplatsen. Sidor är typiska för statiskt innehåll, som till exempel en kontaktsida för ett företag. De visar ingen information om författaren eller datum då innehållet är skrivet.

Historier är anpassade för nyhetsartiklar eller blogginlägg. De visas upp på webbplatsens framsida som standard samt innehåller både författare och tidsstämpel. Historier sorteras också enligt tidsstämpel i fallande ordning.

3.2.2 Moduler

Drupal är ett modulärt ramverk. Modulkonceptet skapar ett enhetligt system för att utvidga Drupals funktionalitet. Allting från fundamentala funktioner som noder och användarhantering till allt mer komplexa datahanteringsfunktioner är gjorda med hjälp av moduler. En modul är i grund och botten en samling filer med strukturerad kod som kan använda sig av Drupals interna regeluppsättningar för att integrera ny funktionalitet i ramverket. (Butcher, 2008:9)

Moduler kan delas in i två grupper, de som finns färdigt i själva Drupal installationen, och de som tillhandahålls av Drupals användare. Nästan alla moduler kan inaktiveras om de inte behövs för att minska på mängden kod en webbplats måste ladda och inkludera och på så vis också höja prestandan. Det finns dock en del moduler som inte kan inaktiveras eftersom de utgör själva grunden för systemet som allting annat bygger på. (Byron et al. 2008:10)

Moduler kan utvecklas på olika sätt och för olika ändamål, men för att utnyttja Drupals kraft är en bra modul en som möjliggör funktionalitet på ett sådant sätt att den enkelt kan integreras med andra moduler. Man strävar till att bygga ihop funktionalitet av olika delar i stället för att skapa en modul som gör allt. På det viset bibehåller systemet den nivå av flexibilitet som möjliggör full kontroll av hur webbplatsen fungerar och ser ut (Byron et al. 2008:10). Kommentarmodulen är ett perfekt exempel på detta, eftersom att den är byggd för att enkelt kunna knytas till olika innehållstyper istället för att vara direkt kopplad till en specifik typ av innehåll.

3.2.3 Block

Medan en nod är till för att skapa och visa större mängder information, är block till för att visa mindre bitar information och kan placeras ut på olika ställen på webbplatsen. Block kan till exempel visa navigationsmenyer, en lista på de tio senaste nyheterna eller något unikt för den inloggade användaren. (Butcher, 2008:16)

Ett block är inte en typ av innehåll i sig själv, utan en modul med hjälp av vilken det går att placera ut information på webbplatsen. Alla block kan konfigureras samt placeras ut

till definierade regioner via webbplatsens administrationsgränssnitt. (Butcher, 2008:16) En region definieras av den designmall som webbplatsen implementerar och kan vara till exempel sidans högra kant.

Blockmodulen är integrerad i Drupal och tillåter block att skapas antingen via administrationsgränssnittet eller via andra moduler. Varje modul kan definiera flera block som visar olika information och kan ha olika inställningar. Till exempel kan ett block visa en lista av rubriker för ett antal av de senaste skapade nyhetsartiklarna på webbplatsen. Moduler kan också definiera inställningar för sina block som kommer upp i administrationsgränssnittet för det blocket. I exemplet med nyhetsartiklarna kunde modulen skapa en inställning för hur många rubriker som skall listas.

3.2.4 Menyer

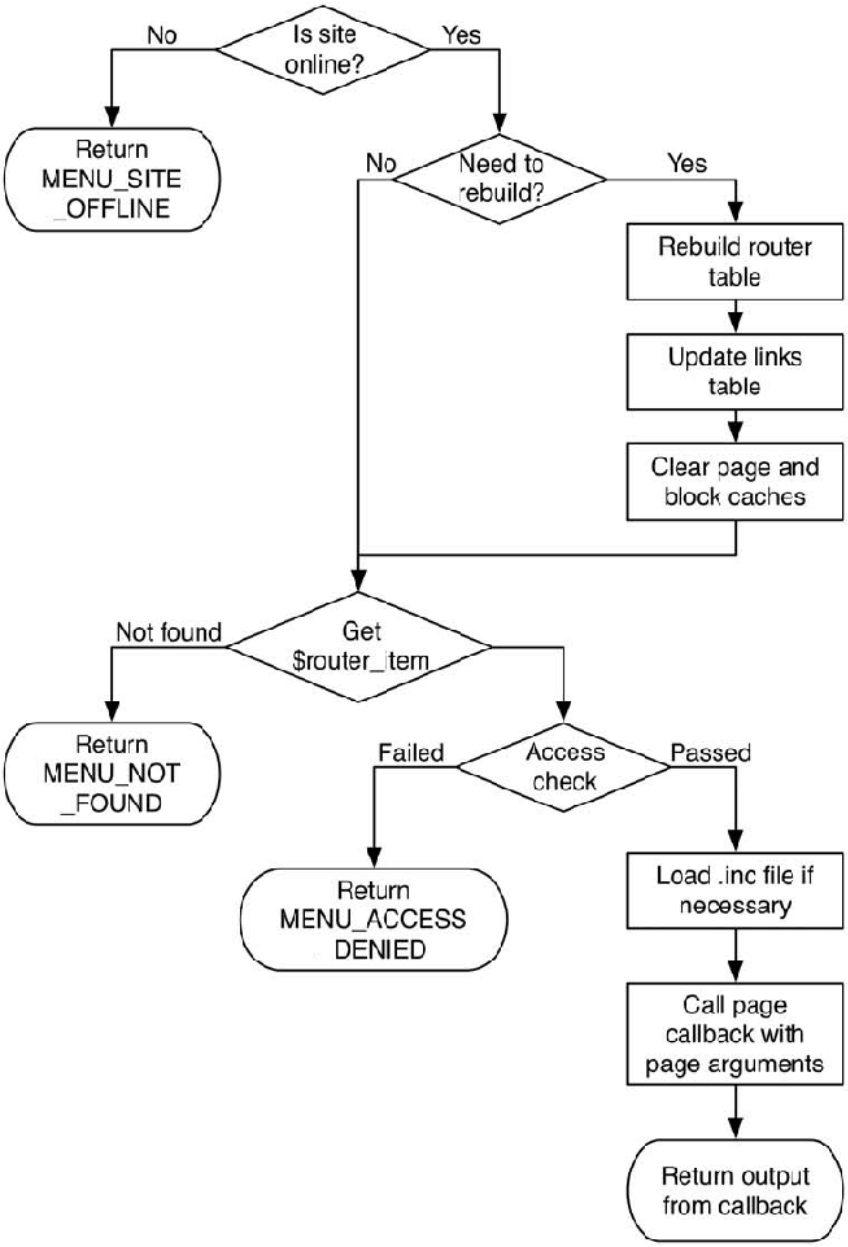
Drupal har ett komplext menysystem som tjänar tre syften. Det fungerar som ett verktyg för att kartlägga sökvägar till specifika hanteringsrutiner, definierar åtkomstkontroll samt anpassar menyernas struktur och utseende. (VanDyk, 2008:59)

Drupal håller reda på vilka sökvägar som är ihopkopplade till vilka hanteringsrutiner med hjälp av en tabell i databasen. En hanteringsrutin är i denna kontext namnet på den PHP funktion som skall köras när en förfrågan görs. Systemet håller också reda på vilka sökvägar som är menyalternativ på webbplatsen med hjälp av en annan tabell. (VanDyk, 2008:59)

När en webbläsare gör en förfrågan till Drupal så måste systemet gå igenom en serie steg för att kunna skapa och visa det material som frågas efter (Figur 3).

- Först måste systemet få fram den sökväg som skall användas. Detta görs genom att systemet klipper bort en del av förfrågningens URL (Uniform Resource Locator).
- Nästa steg är att kontrollera om de tabeller som kartlägger hanteringsrutinerna måste genereras om för att de skall innehålla den senaste informationen.

- Sedan måste den rätta hanteringsrutinen som motsvarar sökvägen hittas i databasen efter vilket ett objekt som beskriver hanteringsrutinen skapas.
- Efter det laddas alla nödvändiga objekt från databasen som skall skickas till hanteringsrutinen.
- Därefter kontrolleras det om användaren har rättigheter till denna hanteringsrutin. Rättigheter kan sättas per hanteringsrutin och är vanligtvis definierade i den modul till vilken hanteringsrutinen hör. Olika rättigheter kan tilldelas olika typer av användare via Drupals administrationsgränssnitt.
- Om menyalternativets titel skall översättas till något annat språk så görs det här. Detta för att alla menyalternativs titlar sparas i originalformat i databasen och översättningen är uppskjuten tills de skall renderas.
- Sedan laddas alla nödvändiga inkluderingsfiler.
- Till sist kallar systemet på hanteringsfunktionen och returnerar resultatet för att visa innehållet på webbplatsen.



Figur 3. Flödesschema över hur en förfrågan hanteras av Drupal (VanDyk, 2008)

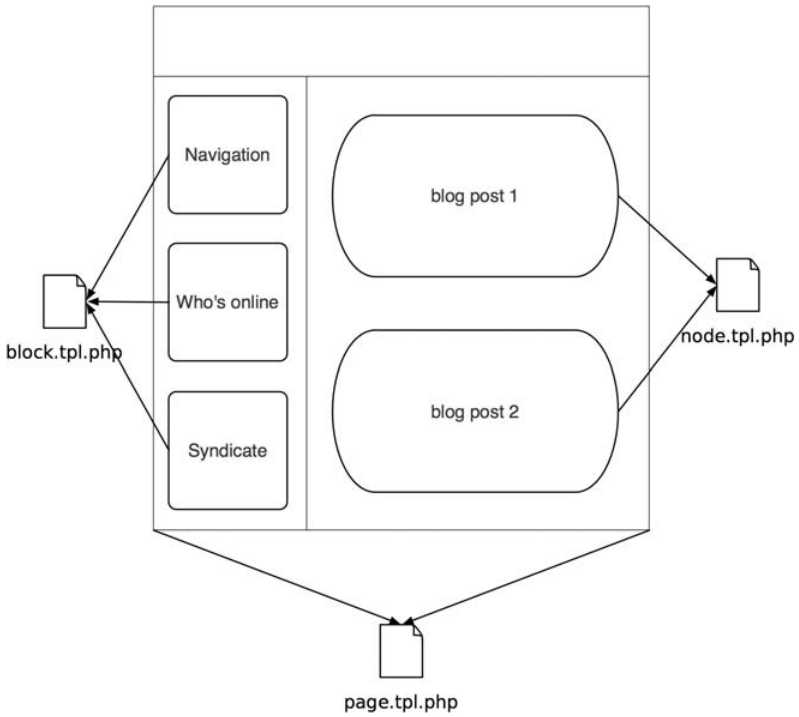
3.2.5 Användarrättigheter

Drupal använder sig av rollbaserad åtkomstkontroll (RBAC) för att hålla reda på rättigheter för olika användare. Det betyder att alla användare tilldelas en eller flera roller i systemet. En roll är egentligen bara en identifierare på en grupp av användare. Rättigheter knyts sedan till dessa roller för att på ett effektivt sätt kunna begränsa vad olika användargrupper kan komma åt på webbplatsen. (Peacock, 2009:42)

Användarrättigheter för innehåll på webbplatsen definieras i moduler och kan tilldelas olika roller via Drupals administrationsgränssnitt. Drupal distribueras med två standardroller, "anonym användare" och "autentiserad användare". Dessa roller tilldelas användare automatiskt och kan inte tas bort, men deras rättigheter kan modifieras enligt behov.

3.2.6 Mallar

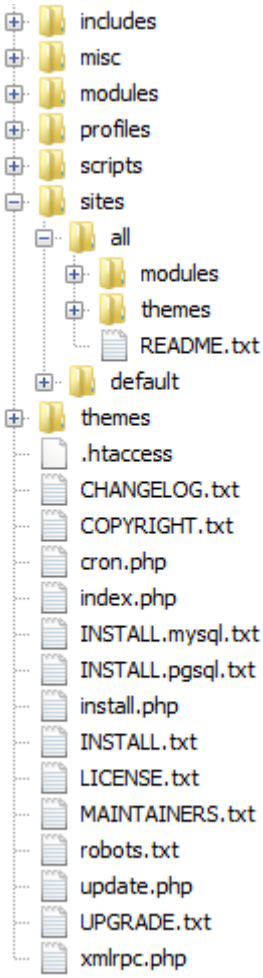
Drupals presentationslager bygger på användningen av mallar som till största del består av XHTML (eXtensible Hypertext Markup Language), CSS (Cascading Style Sheets) samt PHP platshållare för att lägga till dynamiskt innehåll. Alla olika komponenter i systemet har sin egen mall med hjälp av vilken de visar sitt innehåll. Det vill säga alla noder använder en mall och alla block använder en annan. En mall kan också inkludera andra komponenters mallar för att skapa en enhetlig sida (Figur 4). För varje mall finns också en uppsättning registrerade funktioner som kan användas för att åsidosätta standardfunktioner i moduler för att ge fullständig kontroll över hur modulerna genererar den data som skall visas. (drupal.org, 2010 c)



Figur 4. Illustration av hur komponenter använder sig av mallar (VanDyk, 2008)

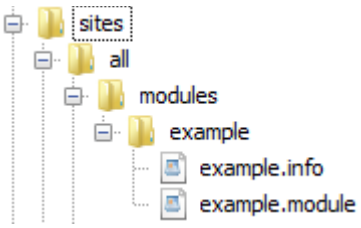
3.3 Filstrukturen

Drupals filstruktur är uppbyggd så att systemets grundläggande funktionalitet är åtskiljd från webbplatsens funktionalitet (Figur 5). Detta förenklar avsevärt uppdateringsprocessen av ramverket samt administreringen av webbplatsens filer.



Figur 5. Filstrukturen för en standard Drupal installation

De moduler som utvecklas för webbplatsen placeras i "modules" katalogen vilken återfinns som underkatalog till "sites/all". En modul består av en katalog som innehåller minst två filer, en infofil samt en modulfil (Figur 6). Infofilen används för att registrera modulen i systemet, specificera vilken version av Drupal den är byggd för samt för att definiera ifall den är beroende av andra moduler. I modulfilen finns de funktioner som modulen använder. Förutom dessa filer kan modulkatalogen innehålla bland annat underkataloger, bilder och modellfiler.



Figur 6. Illustration av en moduls filstruktur

Temafilerna för webbplatsen läggs i sin tur i "themes" katalogen vilken också kan återfinnas son underkatalog till "sites/all". Ett tema är en samling filer som definierar presentationslagret för webbplatsen. Ett tema definierar vanligen gemensamma modellfiler, stilfiler och bilder för webbplatsen.

3.4 Kommunikationsmodell

Att skapa en webbplats med ett "en till många" förhållande är något som de flesta CMS klarar av idag. Däremot kan Drupal utvidga funktionaliteten till någon form av kommunikationsmodellen "många till många". I stället för att till exempel bara ha en blogg eller funktionalitet för att hantera grupper av användare på webbplatsen, så kan man med Drupal tillåta en blogg för varje användare, låta användarna ordna sitt innehåll så att det kan visas upp individuellt med deras egna stilmallar. Systemet kan sedan skapa till exempellistor med de nyaste blogginläggen utgående från alla användares bloggar. (drupal.org, 2010 c)

Eftersom webbforum aspekten är så inbyggd i Drupal är det inte så annorlunda att kombinera funktionalitet för att skapa ett "en till många" förhållanden än att skapa ett "många till många" förhållande. Att ansluta de befintliga egenskaperna med alla aspekter av webbplatsen är något som smidigt kan förverkligas med Drupal. (drupal.org, 2010 c)

3.5 Händelsehantering

Drupal har ett ganska komplext händelsehanteringssystem. Förutom de fördefinierade händelser som finns inbyggda i systemet går det att definiera egna som kan utlösas vid olika tillfällen.

3.5.1 Krokar

Drupals händelsehantering tillåter moduler att ingripa i systemets interna processer vid specifika händelser för att kunna utvidga webbplatsens funktionalitet. När en händelse inträffar så avgör systemet vilka moduler som implementerat den krok (eng. hook) som är ihopkopplad med händelsen och kör den. En krok är i praktiken inget annat en än PHP funktion som är definierad med ett specifikt namn för att systemet skall kunna känna igen den. Namnet bestäms både av den modul som implementerar den samt vilken händelsetyp det är frågan om. Till exempel om man vill göra något specifikt när en ny användare registrerar sig på webbplatsen så skulle en modul implementera "hook_user" funktionen. Varje krok har definierade parametrar samt en specifik struktur på returvärdet. (VanDyk, 2008:35)

3.5.2 Åtgärder

Åtgärder är enskilda uppgifter som systemet kan utföra, exempelvis skicka en e-post eller lägga till en nod på webbplatsens framsida. Moduler kan utlösa dessa åtgärder när vissa händelser i systemet inträffar, till exempel när en nod skapas eller när en användare loggar in. Moduler kan också definiera ytterligare åtgärder genom att implementera kroken "hook_action_info". En åtgärd är egentligen ett inlägg i databasen som definierar vilken funktion som skall kallas på när åtgärden körs. (VanDyk, 2008:37)

3.5.3 Utlösare

Utlösare är händelser som kör åtgärder. Drupal har en standardmodul för att via administrationsgränssnittet koppla ihop åtgärder med vissa händelser i systemet. Det är också möjligt att i en modul skapa anpassade utlösare med hjälp av kroken "hook_hook_info". Då är den modulen ansvarig för att upptäcka när en händelse inträffar, ta reda på vilka

åtgärder är förknippade med den händelsen samt att kalla på de diverse åtgärderna. (VanDyk, 2008:54)

3.6 Säkerhet

En användare kan på många sätt försöka manipulera en webbplats, till exempel genom att exekvera skadlig kod i systemet, manipulera innehållet i databasen, komma åt innehåll som man inte har rättigheter till eller helt enkelt använda webbplatsen för att skicka ut e-post. Drupal erbjuder ett antal verktyg för att förhindra sådana säkerhetsöverträdelser.

3.6.1 Användardata

Vid hantering av användardata i Drupal är standardutförande att man sparar exakt det användaren skrivit i databasen och filtrera det när det visas på webbplatsen. Databasen skall alltid innehålla en noggrann återgivning av det användaren skrivit, men det är dock viktigt att man är medveten om de säkerhetsaspekter som måste tas i beaktande när data sparas. När informationen sedan skall visas så tas all potentiell körbar kod bort. (VanDyk, 2008: 453)

Drupal erbjuder några filtreringsfunktioner med hjälp av vilka man undviker att skadlig kod kan köras när en sida renderas.

- check_plain() översätter alla specialtecken till HTML (Hypertext Markup Language) entiteter, vilket gör att textsträngen kan visas bokstav för bokstav på webbplatsen. Om funktionen inte används så kan till exempel skadlig Javascript kod köras när sidan renderas.
- check_markup() används för text som kan innehålla HTML. Funktionen kör texten genom alla filter som definierats.
- filter_xss() filtrerar bort alla tecken och konstruktioner som kan lura webbläsaren, ser till att alla HTML entiteter är välformade, ser till att alla HTML element och attribut är välformade samt tar bort alla skadliga protokoll från HTML element.

- check_url() funktionen tar bort skadliga protokoll från en webbadress förrän den används i ett HTML attribut.

Den viktigaste regeln för att hantera användardata är att ingen information som användaren har givit någonsin skall visas som sådan på webbplatsen. (drupal.org, 2010 q)

3.6.2 Databastillgång

Ett vanligt sätt att manipulera en webbplats databas är genom SQL-injektion, vilket innebär att en användare utnyttjar hanteringen av den information som SQL-satser byggs upp av.

För att undvika SQL-injektionsattacker så skall abstraktionslagret för databasen användas. Drupal erbjuder bland annat funktionen db_query() för att utföra en databasförfrågan i den aktiva databasen. Men att bara använda denna funktion är inte tillräckligt utan det är upp till utvecklaren att göra koden säker. En SQL-sats som hämtar information om en nod med en specifik innehållstyp kunde se ut såhär.

```
$result = db_query("SELECT n.nid, n.title FROM {node} n WHERE n.type =
'$type'");
```

Denna förfrågan är sårbar för SQL-injektion eftersom variabeln $type inte hanteras korrekt. Man kunde till exempel ge strängen "`story' UNION SELECT s.sid, s.sid FROM {sessions} s WHERE s.uid = 1 /*`" som $type vilket innebär att SQL-satsen ser ut såhär.

```
SELECT n.nid, n.title FROM {node} n WHERE n.type = 'story' UNION SELECT s.sid,
s.sid FROM {sessions} s WHERE s.uid = 1/*'
```

Det skulle ge som resultat all giltiga sessions ID för administratörskontot vilka kunde användas för att få fulla rättigheter på webbplatsen.

För att förhindra det här så erbjuder db_query() funktionen att platshållare används för de parametrar som skall inkluderas i SQL-satsen. Drupal klarar då av att ersätta dessa

platshållare med korrekt "escape" argument i SQL-satsen. Giltiga platshållare som stöds av Drupal är:

- %d - heltal.
- %f - flyttal.
- %s - string (skall inkapslas i situationstecken).
- %b - binär data.
- %% - omskrivs till %.

Det korrekta sättet att skriva denna SQL-stats skulle då vara.

```
$result = db_query("SELECT n.nid, n.title FROM {node} n WHERE n.type = '%s'",
$type);
```

SQL-satsen respekterar dock inte ännu åtkomstkontroll utan söker bland alla noder som finns i databasen. Om användaren inte skall ha tillgång till vissa noder så måste SQL-satsen innehålla en kontroll av rättigheter. Med hjälp av funktionen db_rewrite_sql() kan utvecklare automatisk inkludera denna åtkomstkontroll i SQL-satsen om sådan skall finnas.

```
$result = db_query(db_rewrite_sql($sql_query), $type);
```

Nu kontrollerar Drupal om det finns några inlägg i node_access tabellen som hindrar användaren från att se vissa typer av noder och skriver om SQL-satsen förrän den exekveras.
(drupal.org, 2010 p)

3.7 Objektorientering

Drupal kritiseras ofta för att inte vara objektorienterat. Fastän Drupal inte drar nytta av den objektorienteringsfunktionalitet som finns inbyggd i PHP kan ändå flera objektorienterade koncept och designmönster hittas i systemets arkitektur. (drupal.org, 2010 n)

När Drupal konstruerades så gjordes beslutet att inte använda PHPs klasstrukturer av ett antal olika skäl.

- Drupal är byggt på PHP version 4 och stödet för objektorienterade strukturer var inte tillräckligt utvecklat för att uppfylla de behov som fanns.
- Designen på Drupal bygger i stor grad på moduler som var och en definierar sina egna funktioner samt också hanterar inkluderingen av filer för att minska på mängden kod som måste laddas vid varje förfrågan. Detta eftersom PHPs prestanda lider ju fler filer som måste laddas, särskilt om en PHP accelerator saknas. Funktioner är på så vis definierade innanför andra funktioner med hänsyn till det exekverande omfånget (eng. runtime scope). Denna typ av inkapsling är inte tillåten med klassdeklarationer, utan en klass måste alltid deklareras på högsta nivån.
- Drupal använder också vissa designmönster som inte stöddes av objektorienterad PHP när systemet byggdes. Till exempel använder temasystemet en metod som liknar kategorier i objektorienterad C.

3.7.1 Koncept

Fastän Drupal inte använder klasstrukturer så är ändå flera objektorienterade koncept implementerade i systemet.

3.7.1.1 Objekt

Det finns flera datastrukturer i Drupal som liknar objekt, exempelvis noder, moduler eller användare. En användare är ett objekt innehållande till exempel information om användarens konto och sessionsdata. Likaså är en nod ett objekt innehållande information om den innehållstyp den hör till. I båda fallen är datastrukturen definierad av en databastabell istället för en klass. Drupal utnyttjar sin relationsdatabas för att tillåta att moduler utvidgar objekten med tilläggsinformation. Moduler kan också anses vara objekt eftersom de uppfyller kraven som kontroller i många fall. (drupal.org, 2010 n)

3.7.1.2 Abstraktion

Drupals händelsehanteringssystem är basen för dess abstraktionsgränssnitt. Händelsehanteringsrutinerna definierar de åtgärder som kan utföras på eller av en modul. Genom att modulen implementerar rutinen så måste den utföra en åtgärd och returnera en specifik typ av information, men den funktion som kallat på rutinen behöver inte veta någonting om varken modulen eller om hur själva åtgärden utfördes för att få den information den behöver. (drupal.org, 2010 n)

3.7.1.3 Inkapsling

Drupal har inte ett specifikt sätt att begränsa åtkomsten till ett objekts medlems variabler och funktioner som de flesta andra objektorienterade system, utan litar på att utvecklaren följer konventioner för att uppnå detta. Eftersom funktioner i Drupal delar en gemensam namnrymd så skiljs de åt genom användningen av prefix, och på detta sätt säkerställer att inga konflikter uppstår. Konventioner skiljer också på en moduls publika och privata medlemsfunktioner. En intern funktion som inte är avsedd att nås utanför modulen föregås av ett understreck. Till exempel alla händelsehanteringsrutiner är publika eftersom de skall kallas när någonting skall processeras, men själva funktionen som utför en del av operationen kan vara deklarerad som privat. (drupal.org, 2010 n)

3.7.1.4 Polymorfism

Noder är polymorfa i den klassiska bemärkelsen. När en modul skall rendera en nod så kommer den egentliga rendering att bero på vilken innehållstyp den aktuella noden har, eftersom olika typer av noder visar olika information. Detta är direkt jämförbart med att klassen för ett objekt bestämmer hur renderingen sker. Renderingen av en nod kan också vara påverkad av det tema som webbplatsen implementerar. Teman är polymorfiska på samma vis som noder, de renderar innehåll på olika sätt beroende på den aktuella designen fastän gränssnittet är det samma. (drupal.org, 2010 n)

3.7.1.5 Arv

Moduler och teman kan också ses som klasser som ärver sitt beteende av en abstrakt basklass, även om de själv kan definiera sina funktioner. För teman är funktionerna för basklassen definierade i en inkluderingsfil samt också i de moduler som implementerar teman. Ett tema kan antingen använda eller åsidosätta det standard beteende för rendering som finns. Moduler har på samma sätt möjligheten att åsidosätta de standarddefinierade händelsehanteringsrutinerna genom att implementera egna. (drupal.org, 2010 n)

3.7.2 Designmönster

Drupals interna struktur är dock mer komplicerad än enkla arv och polymorfism. Systemet implementerar nämligen också ett antal etablerade designmönster.

3.7.2.1 Singleton

Moduler och teman inkapslar i allmänhet inga data. Vad som skiljer en modul från en annan är den uppsättning funktioner de definierar, så de kunde ses som klasser med en *singleton* instans. (drupal.org, 2010 n)

3.7.2.2 Decorator

Drupal bygger i hög grad på *decorator* designmönstret. Den polymorfism som nodobjekten implementerar är bara en liten del av kraften bakom nodsystemet. Användningen av händelsehantering för noder som tillåter att moduler utvidgar nodernas beteende utan att behöva delklasser är en annan intressant del.

En standardnod med innehållstypen historia har mycket lite data förknippad med sig. Om man till exempel skulle vilja tillåta filer att laddas upp och förknippas med en nod så skulle man kunna skapa en ny innehållstyp med egenskaperna av en historia plus möjligheten att lägga till filer. Drupals uppladdningsmodul uppfyller detta behov på ett mycket mer modulärt sätt genom att använda nodens API (Application Programming Interface) för att ge varje nod möjligheten att kunna bifoga filer.

Det här beteendet att utvidga funktionaliteten av ett visst objekt oberoende av andra instanser av samma objekt kunde imiteras med *decorator* designmönstret genom att skapa en *decorator* klass som sveper den ursprungliga klassen. Drupal gör detta med hjälp av dess händelsehanteringssystem samt nodens API. (drupal.org, 2010 n)

3.7.2.3 Observer

Observer designmönstret är implementerat i Drupal genom att många händelsehanteringsrutiner tillåter moduler att registrera sig som observatörer av Drupals objekt. Om det till exempel sker en förändring i en vokabulär i Drupals taxonomisystem så anropas en händelsehanteringsrutin för att uppdatera alla moduler som implementerar den. Genom att modulerna har implementerat hanteringsrutinen så har de registrerat sig som observatörer till vokabulärobjektet och nödvändiga ändringar kan då tillämpas efter behov. (drupal.org, 2010 n)

3.7.2.4 Bridge

Principen för *Bridge* designmönstret är använd för att implementera abstraktionslagret för databasanvändningen. Detta abstraktionslager möjliggör att moduler kan skrivas på ett sätt som gör dem oberoende av typen av databas som används. Detta gör det enkelt att skriva nya databaslager som överensstämmer med den API som *Bridge* mönstret definierar för att lägga till stöd för flera databassystem utan att behöva ändra på Drupals interna processer. (drupal.org, 2010 n)

3.7.2.5 Chain of Responsibility

Menysystemet drar nytta av *Chain of Responsibility* designmönstret. För varje förfrågan kontrollerar systemet om det finns en modul för att hantera den, om användaren har rättigheter att komma åt innehållet samt vilken hanteringsrutin som skall ta hand om förfrågan. Detta förverkligas genom att systemet skickar ett meddelande till menyalternativet som motsvarar sökvägen av förfrågan. Om det menyalternativet inte kan hantera förfrågan skickas den uppåt i menykedjan tills en modul hanterar förfrågan, nekar åtkomst till den eller menykedjan är har gåtts igenom. (drupal.org, 2010 n)

3.7.2.6 Command

Drupals händelsehanteringssystem använder sig av *Command* designmönstret för att moduler inte skall behöva implementera alla händelserutiner utan bara de som själva modulen kommer att behöva. Också själva händelserutinerna drar nytta av detta designmönster för att minska på antalet funktioner som måste implementeras genom att de skickar med en indikator för den operation som skall utföras tillsammans med de andra argumenten till funktionen. (drupal.org, 2010 n)

3.8 Egenskaper för webbforum

Drupal har en hel del inbyggda funktioner som är typiska för ett webbforum. Förutom den befintliga funktionaliteten så erbjuder Drupals egna webbforum en hel del intressanta och användbara moduler för att vidare utvidga webbplatsens egenskaper.

3.8.1 Forum

Drupal erbjuder till exempel en standard forummodul med vilken man kan skapa diskussionstrådar. Med hjälp av forumen kan användarna skapa diskussion om olika ämnen. Ett ämne består av ett inledande inlägg som sedan kan kommenteras. Forum kan både grupperas samt läggas under varandra för att skapa en bra struktur och göra det enkelt för användare att hitta ett diskussionsområde av intresse (drupal.org, 2010 h).

3.8.2 Kommentarer

Med hjälp av Drupals inbyggda kommentarmodul kan man tillåta kommentarer på nästan alla typer av inlägg. Möjligheten att kommentera är en vital del för att involvera användare och skapa diskussion. Kommentarers beteende och utseende kan modifieras för den typ av innehåll de hör till. Till exempel går det att ställa in hur många kommentarer som skall visas per sida och vilken ordning de skall komma i. Kommentar kan också visas som en vanlig lista eller som en lista av trådar (drupal.org, 2010 i).

3.8.3 Omröstning

Drupals omröstningsmodul erbjuder en enkel omröstning i form av en fråga med ett antal möjliga svar som användarna kan svara på. Systemet håller reda på antalet svar på frågorna och visar löpande statistik. Omröstningen har också inställningar för hur länge den skall vara aktiv (drupal.org, 2010 k). Förutom denna inbyggda funktionalitet finns det ett antal andra möjligheter som är utvecklade och publicerade av Drupals användare.

3.8.4 Blogg

Bloggmodulen tillåter registrerade användare att upprätthålla sin egen blogg på webbplatsen. En blogg består av enskilda inlägg som ordnas enligt tidsstämpel. Det är också möjligt att kommentera blogginlägg. Modulen är inte nödvändig för att upprätthålla en blogg för hela webbplatsen, utan den kan skapas med hjälp av standardiserade innehållstyper som till exempel sidor. (drupal.org, 2010 l)

Drupals bloggegenskaper tar inte slut där, utan det finns en inbyggd blogg API modul med vilken det går att tillåta blogginlägg från externa applikationer som stöder XML-RPC (Remote Procedure Call protokoll som använder XML) baserad kommunikation. Förutom blogginlägg så stöder denna modul också inlägg till forum. (drupal.org, 2010 m)

3.8.5 Taxonomi

Med hjälp av taxonomimodulen i Drupal är det möjligt att hierarkiskt organisera material på webbplatsen. Modulen kan användas för att klassificera noder av samma innehållstyp genom att tilldela dem kategorier och underkategorier. Till exempel kan en nod som representerar ett musikaliskt verk ha en genre samt en tidsperiod enligt vilka den kan identifieras. I Drupal skapas dessa kategorier som vokabulärer vilka innehåller ett antal termer. Termer kan ordnas hierarkiskt inom en vokabulär. Vokabulärer kan vara både fördefinierade eller öppna för användare att utöka i samband med skapandet av materialet. Genom att användare får definiera egna termer i vokabulären är det möjligt att skapa ett system för markering av material, vilket är vanligt för webbforum. (drupal.org, 2010 r)

3.8.6 RSS

En annan intressant funktionalitet som Drupal erbjuder är möjligheten att samla material från andra webbplatser i RSS (Really Simple Syndication) format. Webbplatsens administrator kan välja hur ofta informationen skall uppdateras från olika källor samt hur materialet skall grupperas och visas på webbplatsen. Modulen som gör detta möjligt skapar också automatiskt ett informationsblock för varje informationskälla samt för varje grupp som sedan kan placeras på webbplatsen. Förutom möjligheten att hämta material från olika webbplatser så möjliggör modulen även att skapa RSS material från innehållet på den egna webbplatsen. (drupal.org, 2010 n)

3.8.7 Flerspråkighet

Drupal kommer med en modul som möjliggör att webbplatsen kan visas på olika språk. När en sida renderas så kontrollerar modulen vilket språk materialet skall översättas till och försöker hitta översättningen i databasen. Om ingen översättning kan hittas så sparas texten så att det enkelt skall gå att identifiera de textsträngar som inte har en översättning. Modulen har två sätt för att tillhandahålla översättningar. Det första är att via administrationsgränssnittet söka efter textsträngar som inte är översatta för ett språk och där specificera en översättning. Det andra sättet är att importera översättningar från filer i GNU gettext Portable Object format till databasen. (drupal.org, 2010 f)

3.9 Plattform

Drupal är byggt att fungera på ett antal olika plattformer som till exempel Linux, Windows, BSD, Mac OSX och Solaris. Det är också utvecklat för att stöda både Apache och Microsoft IIS webbservermiljöer. (drupal.org, 2010 e)

Drupal har också stöd för olika databasmiljöer genom användningen av ett abstraktionsskikt vars avsikt är att bevara den syntax och kraft som finns i SQL så mycket som möjligt. Det ger också ett strukturerat och enhetligt gränssnitt för att dynamiskt konstruera databasfrågor samt upprätthålla säkerhetskontroller. (api.drupal.org, 2010 a)

3.10 Prestanda och optimering

Drupals arkitektur är utformad med flexibilitet i åtanke, denna flexibilitet kommer dock till ett visst pris. När antalet moduler som används växer så ökar också komplexiteten för att hantera en förfrågan. Medan de flesta optimeringar görs på servernivå så har Drupal en del funktionalitet som kan ge betydande prestandaförbättringar. (VanDyk, 2008:527)

3.10.1 Cache

Vanligtvis gör Drupal ett dussintal förfrågningar till databasen när man besöker en sida för att hämta alla uppgifter som behövs för att visa sidan i webbläsaren. Om man har en stor webbplats så kan antalet förfrågningar stiga upp till hundratals. Och det kombinerat med tusentals samtidiga användare resulterar allt som oftast i att servern inte hinner leverera all information inom rimlig tid. (drupal.org, 2010 g)

Drupal distribueras med en modul som kan minska belastningen på databasen genom att komprimera och lagra webbsidor i databasen för att den skall finnas snabbt tillgänglig när en användare gör en förfrågan på samma webbsida. Genom att använda denna cache så minskas databasförfrågningar till en istället för de många som annars skulle ha utförts. Detta fungerar dock bara för anonyma användare och är användbart endast för statisk information så denna modul är inte alltid användbar. (VanDyk, 2008:536)

3.10.2 Bandbredd

En annan prestandaoptimering för att minska antalet förfrågningar till servern samtidigt som den totala storleken på filerna minskar är att möjliggöra den inbyggda funktionaliteten för att komprimera CSS- och JavaScriptfiler. Drupal kan söka efter alla CSS-filer som olika moduler implementerar och sedan komprimera dem i en och samma fil som sparas på servern. Likaså kan alla JavaScriptfiler packas ihop i en enda fil som den också sparas i filsystemet. Detta minskar på antalet förfrågningar per sida och den totala storleken på hämtad data. (VanDyk, 2008:536)

Om cachen för webbsidor är aktiverad så komprimeras informationen med hjälp av zlib i PHP förrän den sparas i databasen. När en förfrågan på en tillfälligt lagrad webbsida sker så kontrollerar Drupal om användarens webbläsare har inbyggt stöd för gzip kodning, om den gör det så skickas den komprimerade informationen direkt till webbläsaren. Annars packar systemet upp informationen förrän den skickas iväg. (VanDyk, 2008: 536)

3.10.3 Sessioner

Drupal sparar sessionsdata för användare i databasen i stället för att lagra den i filer vilket är standardförfarande. Detta gör det enklare att använda flera servrar för systemet men tillför en viss driftskostnad till databasen för att den måste hantera varje användares sessionsinformation. Om det är fråga om en stor webbplats så kan databastabellen för sessionerna bli mycket stor. PHP möjliggör hantering av sessioner och Drupal utnyttjar dess konfigurationsmöjligheter i systemets egna konfigurationsfil. Om en användare inte loggar in på webbplatsen under två dagar så kommer sessionen att tas bort. Om sessionstabellen växer kraftigt så är det en god idé att minska på tidsintervallet för borttagningen av sessioner från databasen. (VanDyk, 2008:537)

3.10.4 Felrapportering

Drupal har ett inbyggt felrapporteringsverktyg som gör det möjligt att spara information om möjliga fel som uppstår i systemet. Informationen kan sparas antingen i databasen eller i serverns egna loggfiler. Felrapporteringen överensstämmer med RFC 3164 standarden, The BSD syslog Protocol, för prioritering av felmeddelanden. (VanDyk, 2008:537)

Som standard sparar Drupal information om eventuella fel i databasen. Tabellen som innehåller denna information kan växa mycket snabbt om den inte töms med jämna mellanrum. För att minska på antalet inlägg i tabellen går det att med hjälp av en inställning i Drupal bestämma antalet felmeddelanden som sparas. Drupal använder sig av cron för att tömma tabellen med jämna intervall. Om cron inte körs regelbundet så tas ingen in-

formation bort från databasen vilket kan skapa en märkbar driftskostnad. (VanDyk, 2008:538)

Med hjälp av en modul kan Drupal också spara eventuella felmeddelanden i serverns egna loggfiler med hjälp av syslog() funktionen i PHP. Denna metod utesluter att databasfrågor måste skickas men kan medföra en viss grad av prestandaförlust genom att servern måste utföra skrivprocessen till hårddisken. (VanDyk, 2008:538)

3.10.5 Cron

Att cron är ordentligt konfigurerad är mycket viktigt ur webbplatsens prestandasynvinkel. Många moduler har uppgifter som måste skötas regelbundet. Cron utför till exempel indexering av webbplatsens innehåll för sökmodulen samt underhållsuppgifter som att ta bort gamla logginlägg från databasen. (drupal.org, 2010 o)

Cron är en tidsbaserad schemaläggare i Unix-liknande operativsystem som oftast kör på den webbserver där Drupal installationen finns men kan också konfigureras att köra över nätverket. Drupal använder cron genom att göra en anonym förfrågan till webbplatsens cron.php fil som i sin tur utför uppgifter åt installerade moduler. (drupal.org, 2010 o)

3.10.6 Automatisk belastningsbegränsning

Drupal har en standardmodul för att automatiskt kunna begränsa belastningen på webbservern. Modul beräknar belastningen genom att med givna mellanrum kontrollera antalet samtida användare och om det visar sig att gränsvärdet har uppnåtts stänger systemet av viss funktionalitet för att minska på belastningen. Själva modulen skapar dock själv en viss driftskostnad genom att den måste göra förfrågningar till databasen för att kunna bestämma antalet samtida användare. Administratören på webbplatsen kan genom Drupals administrationsgränssnitt konfigurera hur ofta modulen skall göra förfrågningar till databasen, sätta olika gränsvärden för samtida anonyma och autentiserade användare samt vilken funktionalitet som skall inaktiveras när gränssvärdet överskrids. (VanDyk, 2008:539)

4 UTVECKLING AV WEBBFORUMET

Den praktiska delen av detta examensarbete går ut på att utveckla funktionalitet för webbforumet. Detta kapitel redogör för några av de mest centrala funktionaliteter som skapades. Exempelkoden för funktionerna är förenklad för att ge en bättre helhetsbild över hur funktionaliteten är implementerad.

4.1 Registrering

En uppenbar men ändå viktig del av webbforumet är registreringsprocessen. Genom att registrera sig på webbplatsen kan användare bidra med att skriva artiklar och kommentera det andra skrivit. Dessutom ges alla registrerade användare en egen profilsida där de kan berätta om sig själva för andra användare. Registreringen sker genom att fylla i ett formulär som kommer att bygga upp användarens profil. Efter att all obligatorisk information har fyllts i och villkoren för att registrera sig har uppfyllts, kan registreringen lämnas in och ett e-post med en välkomsthälsning samt ett engångslösenord skickas till den adress som användaren registrerat sig med. Användaren ombeds att byta ut sitt lösenord efter inloggning med engångslösenordet.

4.1.1 Inbyggd funktionalitet

Drupals användarmodul hanterar inloggningssystemet samt en registreringsprocess som består av ett simpelt formulär med användarnamn samt e-postadress. Drupal har en standardmodul för användarprofiler med hjälp av vilken det går att spara mer information om användaren. Modulen möjliggör även att HMTL fält, för att samla in denna information av användaren, skapas både på registreringssidan och på profilsidan. Systemet hanterar också utskickningen av e-post med engångslösenord när en registrering görs.

4.1.2 Utveckling

Det första som måste göras är att åsidosätta Drupals standarbeteende för att visa registreringssidan. Detta kan göras på flera sätt, men eftersom det är en utvidgning av den

inbyggda användarmodulen så är det mera naturliga sättet att skapa en ny modul som implementerar krokar. Kroken "hook_theme()" används för att i systemet registrera vilka temakrokar som implementeras av modulen. Temakrokar är Drupals sätt att implementera anpassade presentationsmodeller. För att använda vår egen presentationsmodell för registreringen så skall temakorken "user_register" definieras.

```
function modulename_theme()
{
   return array(
      'user_register' => array(
         'arguments' => array('form'=>NULL),
         'template' => 'user-register'
      ),
   );
}
```

Varje temakrok kan definiera ett antal inställningar för hur systemet skall implementera den. För vårt ändamål behövs endast en argumentlista samt namnet på den modellfil som innehåller HTML koden för vår registreringssida. Argumentlistan innehåller de parametrar som skall skickas med till modellfilen när den renderas. Endast namnet på den modellfil som skall användas får ges åt temakroken eftersom systemet automatiskt lägger till ändelsen ".tpl.php". Så filen som skapas skall då namnges "user-register.tpl.php".

Nästa steg är att lägga till de fält som behövs för att samla in all information om användaren vid registreringen. Formulär i Drupal skapas inte genom att manuellt skriva den HTML kod som behövs, utan genom en formulär API. Information om de fält man behöver skickas med till modellfilen i en lista. Formuläret renderas sedan genom att ge denna lista som parameter till en API funktion. Eftersom det enkelt går att skapa formuläret med hjälp av Drupals profilmodul, behöver vår modul inte skapa informationslistan. Profilmodulen skapar två tabeller i databasen när den aktiveras. I den ena tabellen sparas information om de fält som skapas och i den andra sparas informationen som användarna matar in.

När alla fält för att samla in information har skapats så är nästa steg att ta bort de standardfält från registreringsformuläret som inte är nödvändiga. Detta görs genom att implementera en annan krok i vår modul som heter "hook_form_alter". Denna krok kan användas för att manipulera vilket formulär som helst i systemet förrän det renderas.

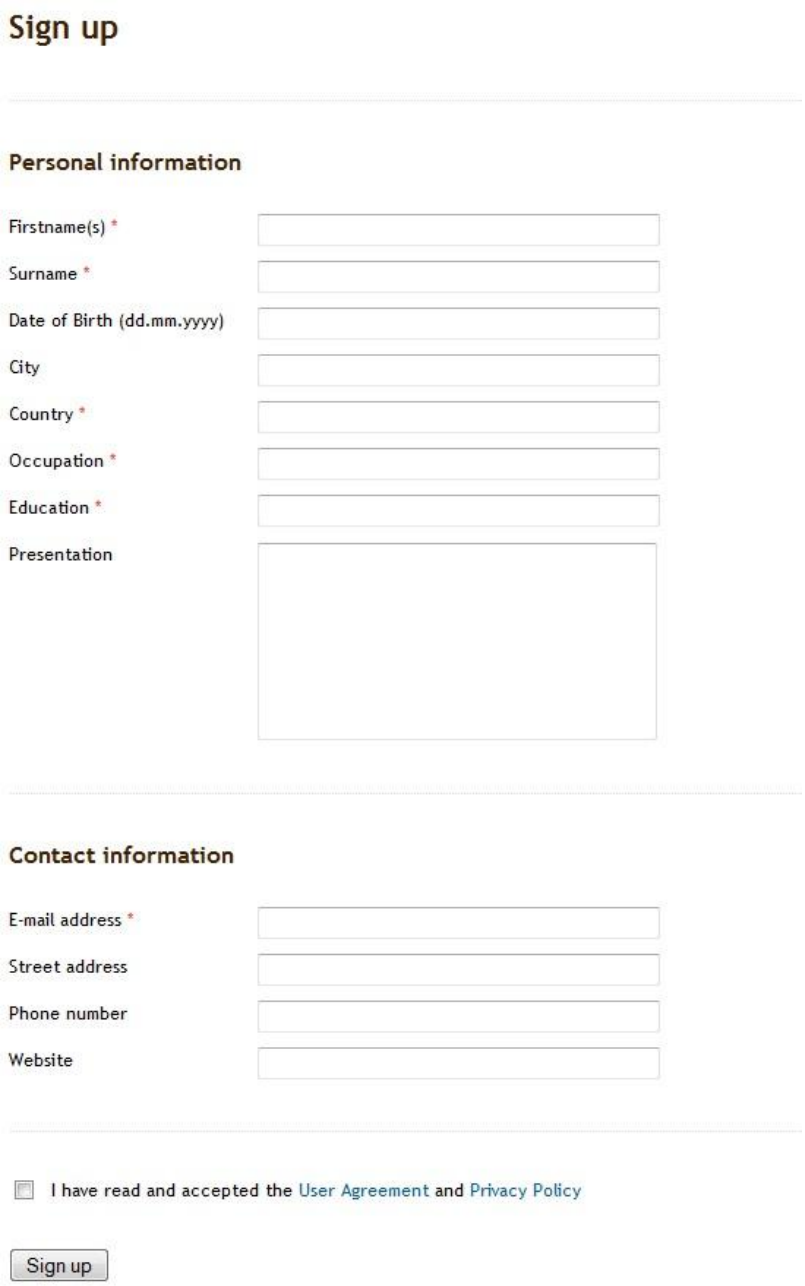
Genom att ta bort informationen från listan för de fält som inte önskas så kommer de inte att renderas i modellfilen. Här kan också anpassade valideringsfunktioner specificeras. Eftersom listan innehållande formulärinformationen ges som en referens så behöver funktionen inte returnera någonting.

```
function modulename_form_alter(&$form, &$form_state, $form_id)
{
   switch($form_id)
   {
      case 'user_register':
         unset($form['account']['name']);
          $form['#validate'] = array('_validate_user_register_form');
         break;
      default:
         break;
   }
}
```

När den rätta informationen visas på registreringssidan är det endast valideringen av informationen som kvarstår att implementera. Anpassade valideringsfunktioner kan skapas för vilket som helst formulär i systemet genom att specificera namnet på funktionen i listan för formulärets information. Eftersom vi redan gjort det i "hook_form_alter" så behöver vi bara skapa funktionen. Eftersom Drupal sköter om att städa upp informationen från skadlig kod så behövs endast specifik validering göras. I det här fallet så kontrollerar funktionen att användaren accepterat användaravtalet samt sekretesspolicyn.

```
function _validate_user_register_form($form, &$form_state)
{
   foreach( $form_state['values'] as $field_name=>$field_value )
   {
      if( $field_name==='profile_accept_terms' && $field_value==0 )
      {
            form_set_error('profile_accept_terms', t('You have to accept the
User Agreement and Privacy Policy.'));
      }
   }
}
```

Efter att vår modul har aktiverats via Drupals administrationsgränssnitt kommer den nya registreringssidan (Figur 7) alltid att visas när en användare besöker den.



Figur 7. Formuläret som skapas av modulen med vilket nya användare kan registrera sig på webbforumet

4.2 Artiklar

Webbplatsens huvudsakliga funktion är att användarna skall kunna skriva, läsa och kommentera artiklar inom olika områden. Det skall vara möjligt att skapa en artikel vilken tilldelas en kategori samt underkategori. Artikeln skall också kunna ges ett valbart antal markeringar för att öka sannolikheten att hittas bland mängden. Artiklarna skall också kunna tilldelas ett citat vilket är till för att kort presentera artikeln när den visas i

diverse listor. Användarna skall ha möjligheten att kommentera andras artiklar samt att svara på andra kommentarer.

4.2.1 Inbyggd funktionalitet

Drupals nodsystem är den idealiska lösningen för att skapa stöd för artiklar. Genom att skapa en ny innehållstyp är det enkelt att särskilja artiklar från resten av materialet på webbplatsen. Taxonomimodulen erbjuder stöd för hierarkiska kategorier samt markeringen av artiklarna. Den inbyggda kommentarmodulen gör i sin tur det möjligt att kommentera artiklarna och svara på andra kommentarer. Drupal har alltså inbyggt stöd för huvudmomenten i artikelsystemet.

4.2.2 Utveckling

Det första som måste göras är att skapa en modul vilken definierar en ny innehållstyp för artikeln. Detta kan göras antingen via Drupals administrationsgränssnitt eller genom att skapa en ny modul. I det här fallet är en ny modul den bättre lösningen på grund av att vi behöver anpassad hantering för kategorier, markeringar och citat. Innehållstyper skapas genom att implementera kroken "hook_node_info" i modulfilen.

```
function modulename_node_info()
{
   return array(
      'article' => array(
         'name' => t('Article'),
         'module' => 'article',
         'description' => 'A <em>article</em> is a content type...',
         'has_title' => true,
         'has_body' => true,
         'title_label' => t('Title'),
         'body_label' => t('Content')
      )
   );
}
```

Kroken returnerar en lista som beskriver de innehållstyper som skall implementeras. Detta räcker för att systemet skall känna igen vår innehållstyp och lägga till en länk, vilken pekar på sidan för att skapa en ny artikel, i menyn. Det går dock inte ännu att skriva någonting eftersom artikeln behöver ett formulär för att samla in informationen. Formuläret skapas genom att implementera kroken "hook_form".

```
function modulename_form(&$node, $form_state)
{
   $form = array();

   /**
   *  The site admin can decide the label of the title and body,
   *  thus we need to load these settings so we can build the form correctly.
   */
   $content_type = node_get_types('type', $node);

   if( $content_type!==false )
   {
      $form['title'] = array(
         '#type'=> 'textfield',
         '#title' => check_plain($content_type->title_label),
         '#required' => true,
         '#default_value' => $node->title
      );
      $form['body'] = array(
         '#type' => 'textarea',
         '#title' => check_plain($content_type->body_label),
         '#rows' => 20,
         '#required' => true,
         '#default_value' => $node->body
      );
      $form['teaser'] = array(
         '#type' => 'textarea',
         '#title' => t('Quote'),
         '#required' => false,
         '#default_value' => isset($node->teaser) ? $node->teaser : ''
      );
   }
   return $form;
}
```

Systemet lägger automatiskt till en knapp till formuläret för att spara artikeln samt en för att förhandsgranska den. Systemet kan automatiskt spara informationen från dessa tre fält i databasen eftersom de alla är standardfält för en nod.

Nästa steg är att lägga till behörighetsinställningar för innehållstypen. Dessa skapas med hjälp av två krokar, "hook_perm" och "hook_access". Den tidigare definierar vilka behörigheter som modulen implementerar.

```
function modulename_perm()
{
   return array(
      'create articles',
      'delete own articles',
      'delete any articles',
      'edit own articles',
      'edit any articles'
   );
}
```

Den senare kontrollerar om användaren har en viss behörighet. All behörighetskontroll skall implementeras i denna funktion för att både vara konsistent och för att administratorn skall ha fulla rättigheter.

```
function modulename_access($op, $node, $account)
{
   switch($op)
   {
       case 'create':
          return user_access('create articles', $account) && $account->uid ?
true : null;
          break;
       case 'update':
          return user_access('edit any articles', $account) ||
(user_access('edit own articles', $account) && ($node->uid == $account->uid))
? true : null;
          break;
       case 'delete':
          return user_access('delete any articles', $account) ||
(user_access('delete own articles', $account) && ($node->uid == $account-
>uid)) ? true : null;
          break;
       default:
          break;
   }
}
```

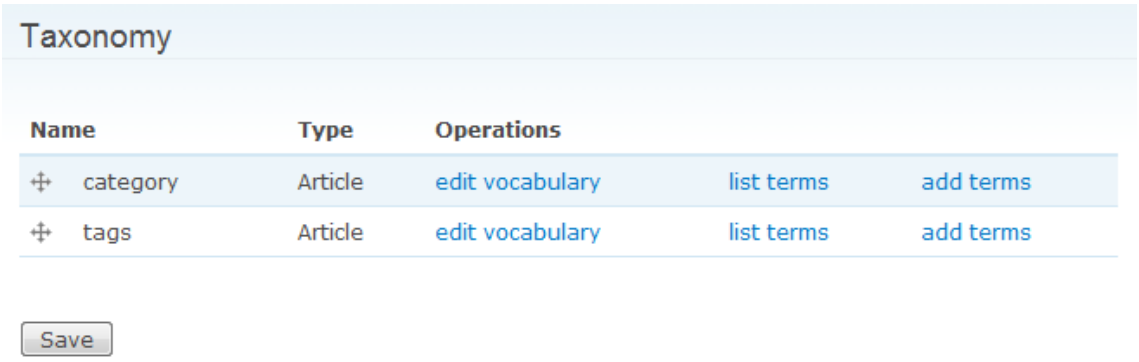
Efter detta går det att via administrationsgränssnittet välja vilka användarroller som har rättigheter att se, skriva, uppdatera och ta bort artiklar (Figur 8).

Permission	anonymous user	authenticated user
article module		
create articles	☐	☑
delete any articles	☐	☐
delete own articles	☐	☑
edit any articles	☐	☐
edit own articles	☐	☑

Figur 8. Administrationsgränssnittet för att tilldela olika typer av användare rättigheter att skapa, uppdatera och ta bort artiklar

Nu återstår det att lägga till kategorier samt aktivera kommenteringen av artiklarna. För att skapa kategorier måste man först aktivera taxonomimodulen. Efter det så dyker en inställningssida upp i administrationsgränssnittet där man får lägga till så kallade vokabulärer som innehåller termer. Dessa termer kan ordnas hierarkiskt för att skapa de ka-

tegorier och underkategorier vi behöver. Vi behöver två vokabulärer, en för kategorier och en för markeringar (Figur 9).



Figur 9. Administrationsgränssnittet för att lägga till och editera vokabulärer i Drupal

Vokabulärerna knyts till vår nyss skapade innehållstyp genom välja den under inställningarna för respektive vokabulär. Taxonomi modulen erbjuder ett standard beteende för att välja och spara termer när en nod skapas eller uppdateras, men vi vill åsidosätta detta beteende på grund av den layout som webbplatsen skall ha. Detta görs genom att sätta en kontrollvariabel i Drupals globala omgivning vilken taxonomimodulen har åtkomst till. Variabeln sätts varje gång artikel modulen laddas genom att placera den i "hook_init".

```
function article_init()
{
   variable_set('taxonomy_override_selector', true);
}
```

För att skapa formulärelement för kategorier och markeringar måste först alla vokabulärer för noden hämtas, sedan kan elementen byggas upp utgående från dem. För att göra detta så lades följande kod till i kroken "hook_form".

```
// Get this nodes vocabularies
$vocabulary_list = article_get_node_vocabulary_list($node);

foreach( $vocabulary_list as $vocabulary )
{
   switch($vocabulary->name)
   {
      // Tag element
      case 'tags':
         $tag_string = implode(', ', article_get_node_tag_list($node, $vocabu-
lary));
```

```
         $form['tags'] = array(
            '#type' => 'textfield',
            '#title' => $vocabulary->name,
            '#description' => filter_xss_admin($vocabulary->help),
            '#required' => $vocabulary->required,
            '#default_value' => $tag_string,
            '#autocomplete_path' => 'taxonomy/autocomplete/'. $vocabulary-
>vid,
         );
         break;

      // Category elements
      case 'category':

         $term_list = article_get_vocabulary_term_list_by_parent($vocabulary,
0);

         $form['category'] = array(
            '#type' => 'select',
            '#title' => t('Choose a category for your article'),
            '#options' => $term_list,
            '#required' => $vocabulary->required,
            '#default_value' => isset($node->category) ? $node->category : 0
         );

         $form['sub_category'] = array(
            '#type' => 'hidden',
            '#title' => '',
            '#default_value' => isset($node->sub_category) ? $node-
>sub_category : '',
         );
         break;

      default:
         break;
   }
}
```

För att sedan spara de värden som man valt för kategorier och markeringar använder vi "hook_insert" kroken, eftersom den kallas på när en ny nod av denna modulens innehållstyp håller på att införas i databasen. Kroken använder taxonomimodulens funktionalitet för att spara informationen. Samma kod körs också vid uppdatering av en existerande artikel, men då via "hook_update". Funktionen laddar först vokabulären för markeringarna eftersom taxonomimodulen behöver veta vokabulärens id.

```
function modulename_insert($node)
{
   $vocabulary = article_get_node_vocabulary_list($node, 'tags');
   $term_map = array(
      'tags' => array(
          $vocabulary->vid => $node->tags
      ),
      'sub_category' => $node->sub_category
   );
   taxonomy_node_save($node, $term_map);
}
```

Förrän en artikelnod editeras eller visas måste information om vilka kategorier och markeringar som noden har knutna till sig hämtas. Detta görs genom att implementera kroken "hook_load", vilken alltid körs när modulen laddat ett nodobjekt. Funktionen returnerar ett objekt som innehåller ytterligare egenskaper för noden vilka skall integreras med nodobjektet.

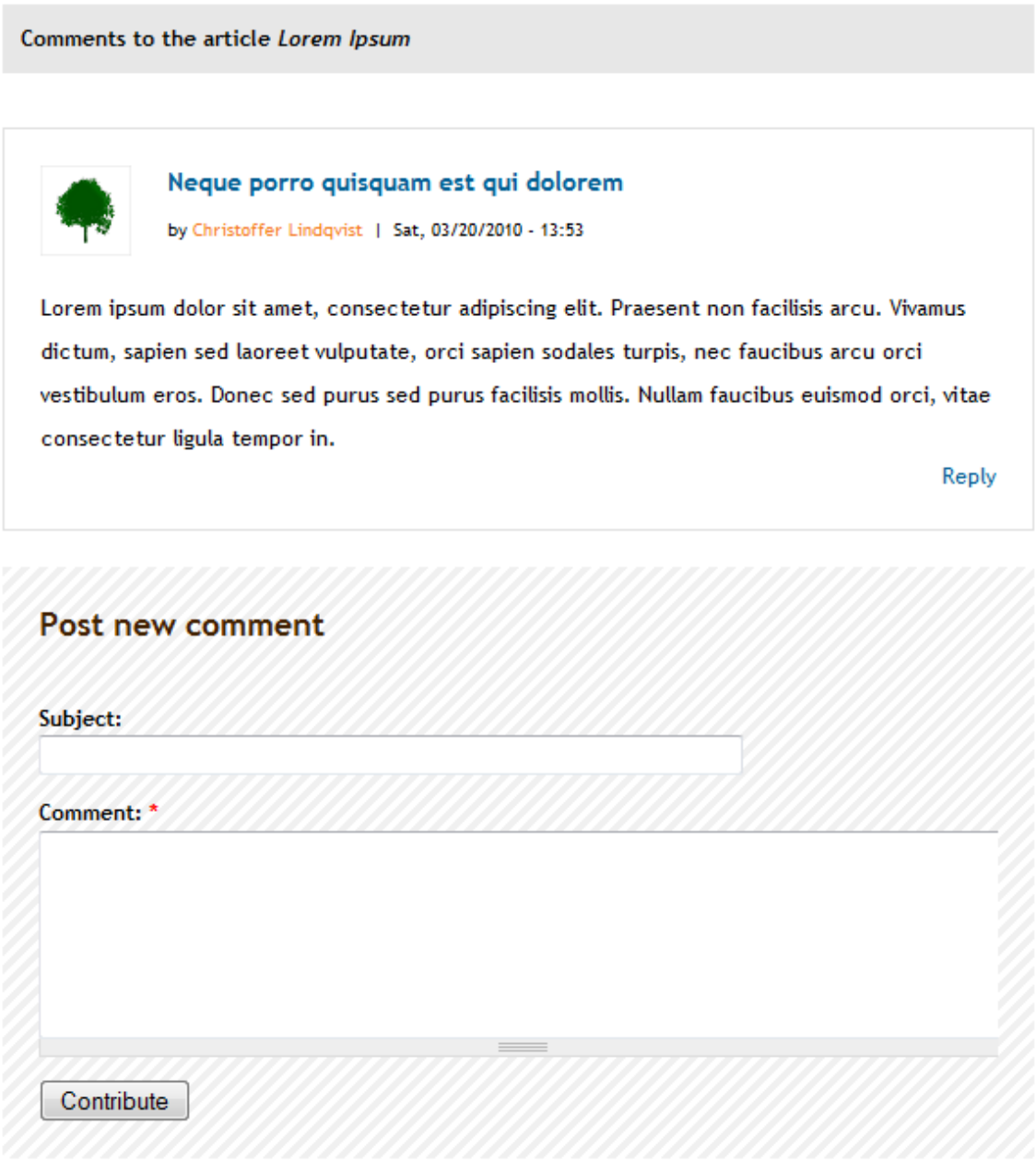
```
function modulname_load($node)
{
   $properties = new stdClass();

   // Get the category vocabulary
   $vocabulary = _modulename_get_node_vocabulary_list($node, 'category');

   // Get the sub-category fot this article
   $properties->sub_category = _modulename_get_node_sub_category($node, $voca-
bulary);

   // Get the category for this article
   $properties->category = _ modulename_get_node_category($properties-
>sub_category);

   // Map all sub-categories to their parents
   $properties->sub_category_map =
_modulename_get_node_sub_category_map($vocabulary);

   return $properties;
}
```

När alla information finns tillgänglig för artikeln återstår att åsidosätta modellfilerna för att skapa och editera en artikel samt den för att visa noden. Detta görs enkelt med kroken "hook_theme". Själva filerna skapas i katalogen för modulen och ges filändelsen ".tpl.php".

```
function modulename_theme()
{
    return array(
        'node' => array(
            'template' => 'node-article',
        ),
        'article_node_form' => array(
            'template' => 'article-node-form',
            'arguments' => array('form' => null)
        )
    );
}
```

För att ha möjligheten att kommentera artiklarna så måste man via administrationsgränssnittet kryssa för ett alternativ under artikelns inställningar. Kommentarerna visas under artikeln de hör till tillsammans med formuläret för att skapa dem (Figur 10). Funktionaliteten för kommenteringen behöver inte modifieras, utan det enda som måste

göras är att åsidosätta den modellfil som visar ett kommentarinlägg samt den som visar formuläret.



Figur 10. Visualisering av artiklars kommentarer samt formuläret med vilka de skapas

Den modellfil som visar ett kommentarinlägg hör till de standardfiler som ett tema har och kan hittas i katalogen för temat. Modellfilen innehållande formuläret hör till själva kommentarmodulen och måste därför åsidosättas med hjälp av kroken "hook_theme".

```
function themename_theme()
{
   return array(
      'comment_form' => array(
         'arguments' => array('form' => NULL),
         'template' => 'comment-form'
      )
   )
}
```

Eftersom vi inte har någon egen modul för kommentarer så skall denna krok placeras i filen "template.php" i katalogen för temat. Denna fil kan användas för att åsidosätta modellfiler och funktioner som renderar HTML på webbplatsen.

4.3 Omröstning

Artiklar och deras kommentarer skall individuellt kunna ges röster i form av en enkel "tummen upp tummen ner omröstning". Resultatet skall hela tiden visas och uppdateras direkt när användaren röstar. Inloggade användare skall endast kunna ge en röst per artikel eller kommentar. Anonyma användare skall endast ha möjlighet att se resultatet, inte rösta.

4.3.1 Inbyggd funktionalitet

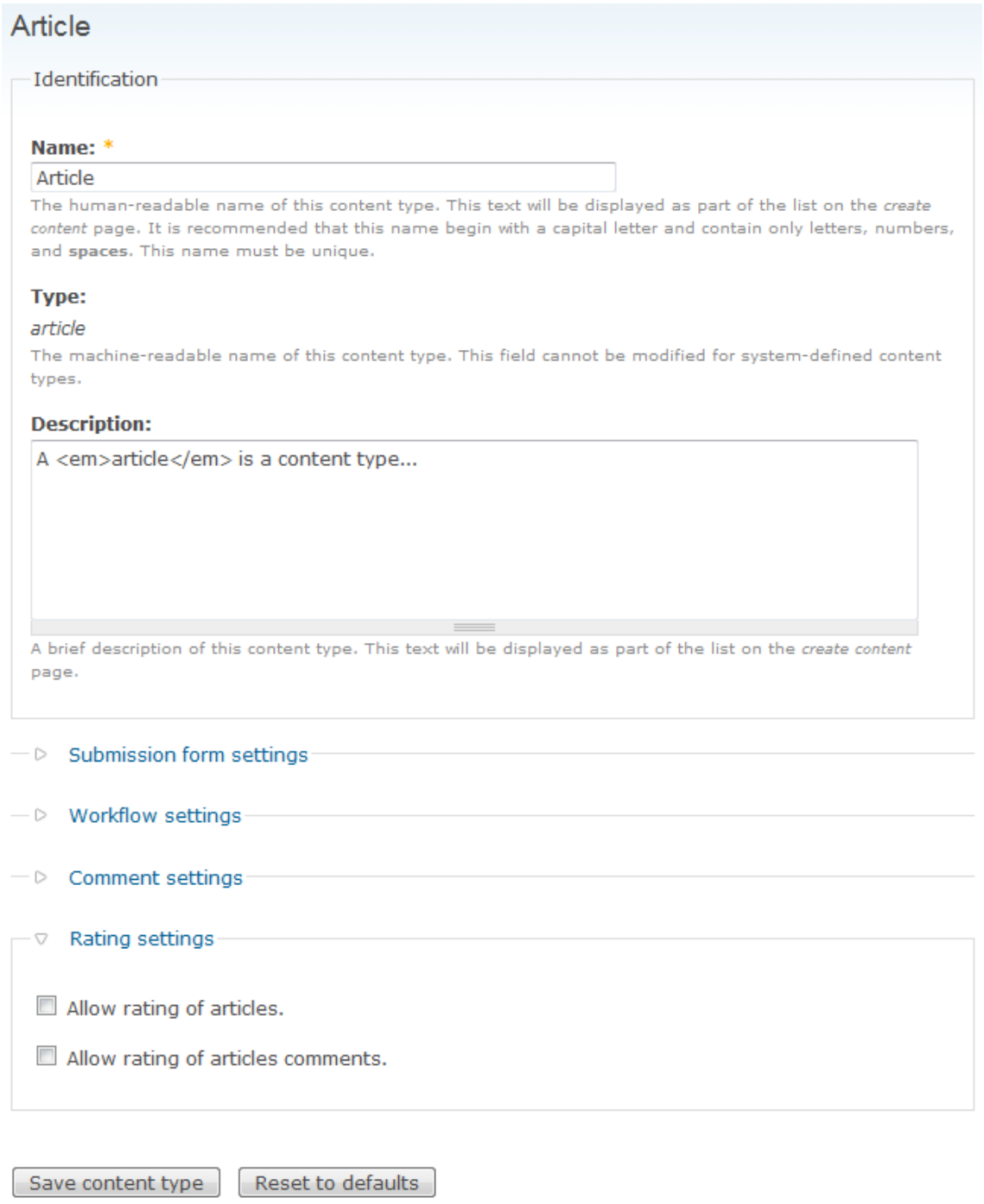
Drupal har ingen inbyggd funktionalitet som stöder omröstning för noder eller kommentarer. Det finns dock en hel del olika omröstningsmoduler som Drupals användare har gjort och publicerat, men ingen av dem är direkt applicerbar för vårt ändamål. Flera av de moduler som finns är baserade på en API modul som stöder bland annat lagring och hämtning av röster i olika format.

4.3.2 Utveckling

För att få funktionaliteten för omröstningen att bli så dynamisk som möjligt och enkel att implementera för både artiklar och kommentarerna så gjordes den som en modul. På så vis kan man enkelt knyta ihop den till de innehållstyper man vill. För att göra det möjligt att via administrationsgränssnittet knyta röstningsmodulen till innehållstyper så implementerades kroken "hook_form_alter" i modulen.

```
function modulename_form_alter(&$form, $form_state, $form_id)
{
   if( $form_id==='node_type_form' )
   {
      $form['rating'] = array(
         '#type' => 'fieldset',
         '#title' => t('Rating settings'),
         '#collapsible' => TRUE,
         '#collapsed' => TRUE
      );

      $form['rating']['rating'] = array(
         '#type' => 'checkbox',
         '#title' => 'Allow rating of '.$form['#node_type']->type.'s.',
         '#default_value' => variable_get('rating_'. $form['#node_type']-
>type, FALSE)
      );

      $form['rating']['rating_comment'] = array(
         '#type' => 'checkbox',
         '#title' => 'Allow rating of '.$form['#node_type']->type.'s com-
ments.',
         '#default_value' => variable_get('rating_comment_'.
$form['#node_type']->type, FALSE)
      );
   }
}
```

Funktionen inför två alternativ till administrationsformuläret för innehållstyper. Ett som gör det möjligt att rösta på innehåll av den innehållstypen och ett annat som gör det möjligt att rösta på kommentarer för den innehållstypen (Figur 11).



Figur 11. Administrationsgränssnittet för artikelinnehållstypen

Drupal sköter själv om att spara inställningarna som ihållande variabler i en tabell i databasen. Namnet på en variabel skapar systemet utgående från namnet på HTML elementet plus innehållstypens namn. Variablerna kan sedan kommas åt med hjälp av Drupals variable_get() funktion.

För att lägga till HTML koden för röstningsmodulen till noder med rätt innehållstyp används kroken "hook_nodeapi", vilken är till för att agera på noder som är definierade av anda moduler. Först kontrolleras om den aktiva nodens innehållstyp tillåter röstning genom att söka efter den variabel som sätts om röstningen är aktiverad. Sedan kontrolleras om vi är på väg att visa denna nod eller skapa den. Om vi skall skapa noden så sparas grundvärden för den med hjälp av API modulen för omröstningen. Om vi däremot skall visa den så söker vi först efter nuvarande resultatet för noden, sedan kontrolleras att användaren har rättigheter att rösta (rättigheterna är definierade i "hook_perm") samt att denna nod inte är skapad av användaren. Sedan kontrolleras ännu om användaren redan har röstat på denna nod. Utgående från denna data så läggs HTML koden för omröstningen till nod objektet med hjälp av Drupals theme() funktion. Detta kommer ännu inte att visa omröstningen på webbsidan utan modellfilen för artikelnoden måste ändras så att den renderar informationen från nod objektet. Detta är gjort för att det skall vara enkelt att placera ut informationen var man vill på webbsidan.

```
function modulename_nodeapi(&$node, $op, $teaser=NULL, $page=NULL)
{
   // Check if this node is to be rated
   if( variable_get('rating_'. $node->type, FALSE) )
   {
      switch ($op)
      {
         // If we are about to view a node
         case 'view':
            global $user;

            $data = array();

            // Get the current rating for the node
            $data['current_rating'] = _rating_get_current_rating('node',
$node->nid);

            // Check user permissions and owner status
            if( user_access('rate nodes') && $node->uid!==$user->uid )
            {
               // Get user votes for this node
               $user_vote_count = _get_user_vote_count('node', $node->nid);

               // If user has voted
               if( $user_vote_count>0 )
               {
                  $data['user_has_voted'] = true;

                  // Add rating view to node
                  $node->rating['values'] = theme('rating_view', 'node',
$node->nid, $data);
               }
               // Otherwise
               else
```

```
               {
                  // Create voting url's
                  $data['vote_up_url'] = url('node/'. $node->nid .'/vote/'.
VOTE_UP);
                  $data['vote_down_url'] = url('node/'. $node->nid .'/vote/'.
VOTE_DOWN);

                 // Add rating view to node
                 $node->rating['values'] = theme('rating_view', 'node', $node-
>nid, $data);
               }
            }
            else
            {
               // Add rating view to node
               $node->rating['values'] = theme('rating_view', 'node', $node-
>nid, $data);
            }
            break;

         // If we are about to insert a node
         case 'insert':
            // Set default rating values for the node
            $cached = array(
               array(
                  'content_type' => 'node',
                  'content_id' => $node->nid,
                  'value_type' => 'points',
                  'value' => 0,
                  'function' => 'sum',
                  'tag' => 'rating-up'
               ),
               array(
                  'content_type' => 'node',
                  'content_id' => $node->nid,
                  'value_type' => 'points',
                  'value' => 0,
                  'function' => 'sum',
                  'tag' => 'rating-down'
               )
            );

            votingapi_add_results($cached);
            break;
         }
      }
}
```

Motsvarande kod används också i kroken "hook_comment", vilken tillåter moduler att utvidga funktionaliteten för kommentarer genom att agera när vissa åtgärder för en kommentar utförs.

För att temafunktionen skall kunna hitta någon HTML så måste kroken "hook_theme" definiera en modellfil som skall användas för att visa informationen.

```
function modulename_theme()
{
   return array(
      'rating_view' => array(
      'arguments' => array(
         'type'=>NULL,
         'id'=>NULL,
         'data'=>array()
      ),
      'template' => 'rating-view'
      )
   );
}
```

Nu kan HTML koden för omröstningen visas för en artikel eller kommentar (Figur 12), men själva processen för att rösta måste ännu implementeras.

Figur 12. Visualisering av hur omröstningsmodulen implementerades för artiklar

Processen för att registrerar en röst i systemet sker med en Ajax förfrågan när en användare klickar på symbolen för antingen tummen upp eller tummen ner. Först måste de sökvägar som används i förfrågningarna registreras i systemet så att de kan hanteras. Detta görs med hjälp av kroken "hook_menu", vilken egentligen är avsedd för att skapa länkar för menyer men kan också registrera endast sökvägar som är knutna till en funktion. Kroken skall returnera en lista menyalternativ med en sökväg som identifikator. Alternativen definierar vissa inställningar som bestämmer hur de skall behandlas när förfrågan sker. Argumenten som skickas till hanteringsfunktionen definieras av parametern "page arguments" genom att specificera deras position i sökvägen. Genom att använda "%" tecknet i sökvägen tillåter vi att vilket som helst tecken kan användas i dess ställe, därför är det viktigt att komma ihåg valideringen av parametrarna i funktionen som hanterar förfrågan.

```
function rating_menu()
{
   $items['node/%/vote/%'] = array(
      'title' => 'Node rating',
      'type' => MENU_CALLBACK,
      'page callback' => 'modulename_register_vote ',
      'page arguments' => array(0, 1, 3),
      'access arguments' => array('rate nodes')
   );
   $items['comment/%/vote/%'] = array(
      'title' => 'Comment rating',
      'type' => MENU_CALLBACK,
      'page callback' => 'modulename_register_vote',
      'page arguments' => array(0, 1, 3),
      'access arguments' => array('rate comments')
   );

   return $items;
}
```

Hanteringsfunktionen för förfrågningarna validerar först parametrarna och kontrollerar sedan att användaren har tillåtelse att registrera en röst för den här typen av innehåll. Själva rösten registreras med hjälp av API modulen i databasen. Efter det så byggs en lista med returdata upp och konverteras till JSON (JavaScript Object Notation) format förrän den returneras för att kunna uppdatera informationen på webbsidan med hjälp av JavaScript.

```
function modulename_register_vote($type, $id, $vote)
{
   // Validate wildcard parameters, will redirect on failure
   _modulename_validate_int(array($id, $vote));

   // Check if we are allowed to vote
   if( _modulename_allow_vote($type, $id, $vote) )
   {
      global $user;

      switch($vote)
      {
         case VOTE_UP:
            // Create vote
            $vote = array(array(
               'content_type' => $type,
               'content_id' => $id,
               'value_type' => 'points',
               'value' => VOTE,
               'uid' => $user->uid,
               'tag' => 'rating-up'
            ));

            votingapi_set_votes($vote);
            break;

         case VOTE_DOWN:
            // Create vote
            $vote = array(array(
```

```
                'content_type' => $type,
                'content_id' => $id,
                'value_type' => 'points',
                'value' => VOTE,
                'uid' => $user->uid,
                'tag' => 'rating-down',
            ));

            votingapi_set_votes($vote);
            break;

         default:
            return drupal_not_found();
            break;
      }

      // Get the current rating
      $rating = _rating_get_current_rating($type, $id);

      // Ajax return data
      $js = array(
         'content_type' => $type,
         'content_id' => $id,
         'rating_up' => $rating['up'],
         'rating_down' => $rating['down']
      );

      // Returning data in JSON format
      drupal_json($js);
      exit;
   }
   else
   {
      drupal_goto($_SERVER['HTTP_REFERER']);
   }
}
```

4.4 Prenumeration

Att prenumerera på artiklar som någon annan medlem skriver är en ganska typisk funktionalitet för ett webbforum. För att inte manuellt behöva leta upp om en viss skribent har publicerat en ny artikel vill man gärna bli informerad om detta automatiskt av systemet. När en artikel visas skall det på samma sida finnas en länk med vilken man enkelt kan börja prenumerera på artiklar av den författaren (Figur 13). När en författare skriver en ny artikel eller uppdaterar en existerande så skall alla användare som prenumererar på denna författare få ett meddelande på sin profilsida. Meddelande skall innehålla vem som har skapat eller editerat en artikel samt en länk som tar användaren direkt till artikeln.

Figur 13. Visualisering av de möjligheter som finns för att agera på skrivna artiklar

4.4.1 Inbyggd funktionalitet

Denna funktion omfattar två delar, användarrelationen och händelsehanteringen. Drupal har ingen inbyggd funktionalitet som direkt stöder dessa två delar. Däremot finns det färdiga lösningar att hittas på Drupals webbplats. Där finns en generell relationsmodul med vilken det enkelt går att skapa egna relationstyper. Modulen stöder att vissa inställningar kan göras per relationstyp. Till exempel är det möjligt att välja om relationen skall vara ömsesidig eller inte. Denna modul är precis vad som behövs för att skapa en användarrelation för prenumereringen.

På Drupals webbplats finns även en händelsehanteringsmodul som gör det möjligt att arkivera vissa händelser i systemet. Denna modul är dock svår att utvidga enligt våra behov så den mest optimala lösningen i detta fall är att skapa en egen modul.

4.4.2 Utveckling

Genom att aktivera relationsmodulen är det möjligt att via administrationsgränssnittet skapa en prenumerationsrelation vilken inte är ömsesidig samt är knuten endast till autentiserade användare. Modulen skapar tre tabeller i databasen vilka håller reda på befintliga relationer samt deras inställningar.

Nästa steg är att skapa en tabell i databasen för att kunna samla upp information om händelser i systemet. Eftersom vi vill ha möjligheten att senare kunna utvidga händelsehanteringen utöver noder och speciella innehållstyper så skall tabellen göras så generell som möjligt. De fält som är relevanta är användarens id, modulens namn till vilken händelsen hör, typen av modul (dvs. innehållstyp), vilken händelse som inträffat, tidpunkten samt ett för extra information. InnoDB användes som lagringsmotor för att kunna dra nytta av dess låsning av radnivåer. Indexering av kolumner gjordes också för de viktigaste söktermerna.

```
CREATE TABLE `activity` (
   `aid` int(11) NOT NULL AUTO_INCREMENT,
   `uid` int(11) NOT NULL,
   `module` varchar(50) NOT NULL,
   `type` varchar(25) NOT NULL,
   `operation` varchar(25) NOT,
   `created` int(11) NOT NULL,
   `data` text NOT NULL,
   PRIMARY KEY (`aid`),
   KEY `uid` (`uid`),
   KEY `module` (`module`),
   KEY `operation` (`operation`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;
```

Sedan skall en modul som hanterar aktiviteterna skapas. Eftersom vi tills vidare endast är intresserade av information om artiklar behöver vi inte spara annan data. Kroken "hook_nodeapi" möjliggör att händelser för noder kan sparas eftersom kroken körs när Drupal sparar noden. För att veta för vilka innehållstyper händelseinformationen skall sparas implementerar modulen en konfigurationssida i administrationsgränssnittet där dessa kan väljas. Inställningarna sparas i Drupals tabell över systemvariabler varifrån de kan hämtas för att utföra kontrollen.

```
function modulename_nodeapi(&$node, $op, $teaser=null, $page=null)
{
   switch($op)
   {
      case 'insert':
      case 'update':
         if( variable_get('activity_logging_node_'.strtolower($node->type),
false) && $node->status==1 )
         {
            $data = array(
               'operation'  => $op==='insert' ? t('create') : t($op),
               'node-id'    => $node->nid,
               'node-title' => $node->title,
               'node-type'  => $node->type,
            );

            _modulename_insert($node->uid, 'node', $node->type, $op, $data);
         }
         break;
   }
}
```

För att behålla möjligheten att utvidga modulen så skapas en generell funktion för att spara informationen i aktivitetstabellen. Denna funktion lägger endast till informationen till tabellen.

```
function _modulename_insert($uid, $module, $type, $operation, $data)
{
   $sql = "INSERT INTO {activity} (uid, module, type, operation, created,
data) ";
   $sql.= "VALUES (%d, '%s', '%s', '%s', %d, '%s')";

   db_query($sql, array($uid, $module, $type, $operation, time(), seriali-
ze($data)));
}
```

När informationen är sparad i databasen behövs funktionalitet för att få fram den igen samt skicka den till användarens profilsida. För att hämta aktiviteterna när användaren öppnar profilsidan används "hook_user", vilken tillåter moduler att reagera när någonting utförs på ett användarkonto. Kroken bygger först upp ett filter som innehåller parametrar enligt vilka själva sökfunktionen skall hitta de rätta aktiviteterna. I filtret ingår en lista på de författare vars artiklar användaren prenumererar på. Listan fås genom att göra en databasförfrågan som söker efter prenumerationer bland användarrelationerna.

```
function modulename_user($op, &$edit, &$account, $category = NULL)
{
   switch($op)
   {
      case 'view':
         if( user_access('view user activities') )
         {
            $filter = array(
               'uid' => $uid_list =
_modulename_get_relation_uid_list($account->uid, 'subscription', null),
               'module' => array('node'),
               'type' => array('article'),
               'operation' => array('insert', 'update')
            );

            if( is_array($filter['uid']) && count($filter['uid'])>0 )
            {
               $announcement_list = _modulename_get_activity($filter);

               $account->content['announcements'] = array(
                  '#type' => 'user_profile_category',
                  '#title' => t('Announcements')
               );

               $account->content['announcements'][] = array(
                  '#type' => 'user_profile_item',
                  '#value' => theme('show_subscription_announcements', $anno-
uncement_list)
               );
            }
         }
         break;
      default:
         break;
   }
}
```

Listan med aktiviteter skickas via en funktion som bygger upp meddelanden av informationen till en modellfil som skapar HTML för dem (Figur 14). HTML koden läggs sedan till användarkontoobjektet som renderar profilsidan.

Figur 14. Meddelanden om vad användare du har prenumererat till har skrivit på webbplatsen

Själva sökfunktionen som hämtar aktiviteterna för artiklarna bygger upp en databasförfrågan utgående från de parametrar som passas till den i filtret. Funktionen är byggd så generell som möjligt för att kunna stöda andra aktiviteter än de för artiklar.

```
function _modulename_get_activity($filter = null)
{
   $where_part_list = array();
   $parameter_list = array();

   // Build the WHERE clause
   if( is_array($filter) && count($filter)>0 )
   {
      $ops = array(
         'include' => " = '%s'",
         'exclude' => " != '%s'"
      );

      foreach( $filter as $column => $value_list )
      {
         $column = "a.{$column}";

         if( is_array($value_list)===false ||
count(array_intersect(array_keys($ops), array_keys($value_list)))===0 )
            $value_list = array('include' => $value_list);

         foreach( $value_list as $criteria => $values )
         {
            if( is_array($values) )
            {
               $placeholder_list = array();
               foreach( $values as $value )
               {
                  $placeholder_list[] = "'%s'";
                  $parameter_list[] = $value;
               }
               $where_part_list[] = $column . ($criteria==='exclude'?' NOT IN
':' IN ') .'('. implode(',', $placeholder_list). ')';
            }
            else
            {
              $where_part_list[] = $column . $ops[$criteria];
              $parameter_list[] = $values;
            }
         }
      }
   }

   // If we have a where part for the query
   if( count($where_part_list)>0 )
   {
      $where = implode(' AND ', $where_part_list);
      $where = "WHERE $where";
   }

   // Build the SQL query
   $sql =  "SELECT * FROM {activity} a ";
   $sql .= $where;

   // Run it
   $result = db_query($sql, $parameter_list);

   // Gather results
   $activity_list = array();
   while( $row = db_fetch_array($result) )
   {
      $row['data'] = unserialize($row['data']);
      $row['data']['aid'] = $row['aid'];
      $row['data']['uid'] = $row['uid'];
      $row['data']['module'] = $row['module'];
```

```
      $row['data']['type'] = $row['type'];
      $row['data']['operation'] = ($row['data']['operation'] ?
$row['data']['operation'] : $row['operation']);
      $row['data']['created'] = $row['created'];

      $activity_list[] = $row;
   }

   return _modulename_construct_messages($activity_list);
}
```

4.5 Sökning av användare

En sökfunktion för att hitta andra användare är en väsentlig del av webbplatsen. Det skall gå att söka användare enligt namn, yrke, ort och utbildning. Sökningen skall kunna utföras på ett eller flera av villkoren. Alla sökfält bör ha automatisk komplettering medan man skriver.

4.5.1 Inbyggd funktionalitet

Drupal har en standardiserad sökmodul med vilken man kan söka både innehåll på webbplatsen samt också användare. Denna modul stöder dock endast sökning av användare på basis av deras användarnamn, vilka inte är i användning. För att kunna göra en sökning baserad på användarnas profilinformation måste en egen sökfunktion implementeras. Drupal använder JavaScript biblioteket jQuery, vilket erbjuder funktionalitet för kompletteringen. Drupals formulär API stöder användningen av komplettering och har gjort det enkelt att implementera egna funktioner som söker informationen.

4.5.2 Utveckling

Eftersom vi redan har en egen modul för användare så är den naturligaste platsen att implementera sökfunktionen i samma modul eftersom vi då har all funktionalitet relaterad till användaren där.

Först måste ett menyalternativ registreras i systemet för att kunna komma åt söksidan. Menyalternativet placeras i Drupals navigationsmeny med hjälp av typ attributet i kroken "hook_menu". För alternativet definieras även vilken funktion som skall köras när en förfrågan till sökvägen sker. Funktionen som returnerar HMTL koden för söksidan genom att hämta den från en modellfil med hjälp av kroken "hook_theme".

```
function modulename_menu()
{
   $items = array();

   $items['search'] = array(
      'title' => t('Find Users'),
      'page callback' => 'modulename_show_search_form',
      'access arguments' => array('search users'),
      'type' => MENU_NORMAL_ITEM
   );

   return $items;
}
```

Modellfilen renderar ett formulär med hjälp av Drupals funktion drupal_get_form(), vilken tar som parameter namnet på den funktionen som bygger upp listan för formuläret. Formuläret innehåller fyra fält vilka representerar de villkor med vilka sökningen kan utföras.

```
function modulename_search_form()
{
   $form = array();

   $form['filter'] = array(
      '#tree' => true
   );
   $form['filter']['name'] = array(
      '#type' => 'textfield',
      '#title' => 'Name',
      '#autocomplete_path' => 'modulename/name/autocomplete'
   );
   $form['filter']['location'] = array(
      '#type' => 'textfield',
      '#title' => 'Location',
      '#autocomplete_path' => 'modulename/location/autocomplete'
   );
   $form['filter']['occupation'] = array(
      '#type' => 'textfield',
      '#title' => 'Occupation',
      '#autocomplete_path' => 'modulename/occupation/autocomplete'
   );
   $form['filter']['education'] = array(
      '#type' => 'textfield',
      '#title' => 'Education',
      '#autocomplete_path' => 'modulename/education/autocomplete'
   );
   $form['submit'] = array(
      '#type' => 'submit',
      '#value' => t("Find")
   );

   return $form;
}
```

Attributet "#autocomplete_path" specificerar sökvägen för den Ajax förfrågan vilken hanterar kompletteringen. Sökvägarna måste registreras i systemet med hjälp av kroken

"hook_menu". Dessutom måste funktionerna som skall hämta informationen från databasen skapas. Funktionen som hanterar kompletteringen av namnfältet söker i databasen både på för- och efternamn. Returvärdet konverteras till JSON format för att JavaScriptet skall kunna hantera informationen.

```
function modulename_name_autocomplete($string='')
{
   $matches = array();
   if( empty($string)==false )
   {
      $result = db_query("SELECT firstname, surname FROM {user} WHERE (first-
name LIKE '%s%%' OR surname LIKE '%s%%') AND uid > 1", $string);
      while( $obj = db_fetch_object($result) )
      {
         $name = check_plain($obj->firstname . ' ' . $obj->surname);
         $matches[$name] = $name;
      }
   }

   drupal_json($matches);
}
```

När användaren utför en sökning så skickas informationen i formuläret till en hanteringsfunktion förrän man omdirigeras till samma sida. Funktionen kontrollerar om användaren givit en eller flera sökparametrar och sparar dem i användarens session. Om inga parametrar har givits så sätts ett felmeddelande som visas när användaren omdirigeras.

```
function modulename_search_form_submit($form, &$form_state)
{
   // Remove any exisiting search filter from session
   _modulename_destroy_search_filter();

   // Get given keywords
   $filter = array_filter($form_state['values']['filter']);

   // If no keywords where given
   if( empty($filter)===true )
      form_set_error('search', t(error message...'));
   // Else store the filter in session
   else
      $_SESSION['search']['users']['filter'] = $filter;
}
```

När sidan renderas på nytt efter omdirigeringen så söker modulen efter användare utgående från informationen som sparats i sessionen. Informationen om användarna hämtas i två steg. I det första steget tar man reda på vilka användare som motsvarar de givna parametrarna. Eftersom all information inte finns i en och samma tabell i databasen så byggs SQL-frågan upp med INNER JOIN klausuler för att inkludera all relevant data.

```
function modulename_get_user_list()
{
   // Get the saved keywords from session if exists
   $filter = _mosulename_get_filter();

   if( empty($filter)===true )
      return array();

   global $user;
   $args = array();

   // Build query
   $query = 'SELECT {users}.uid FROM {users} ';

   foreach( $filter as $field=>$value )
   {
       switch($field)
       {
          case 'name':
             $query .= 'INNER JOIN {user} ON ( {users}.uid = {user}.uid ';
             $name_list = explode(' ', $value);
             $query .= 'AND (0 ';

             foreach($name_list as $name)
             {
                $query .= 'OR {user}.firstname LIKE "%%%s%%" ';
                $args[] = $name;

                $query .= 'OR {user}.surname LIKE "%%%s%%" ';
                $args[] = $name;
             }
             $query .= ")) ";
             break;

          case 'location':
          case 'occupation':
          case 'eduction':
             $alias = '{profile_values}_'.$field;
             $query .= 'INNER JOIN {profile_values} AS '.$alias.' ON (
{users}.uid = '.$alias.'.uid ';
             $query .= 'AND ( '.$alias.'.fid = ' . $profi-
le_field_id_map[$alias] . ' ';
             $query .= 'AND '.$alias.'.value LIKE "%%%s%%" ) ';

             $args[] = $value;
             break;

          default:
             break;
       }

   }

   // Discard user 0, admin, ourselves and disabled users
   $query .= "WHERE {users}.uid NOT IN (0, 1, $user->uid) AND {users}.status =
1 ";

   // Group by user id
   $query .= "GROUP BY {users}.uid ";

   // Run query
   $result = db_query($query, $args);

   // Fetch the results
   $user_list = array();
```

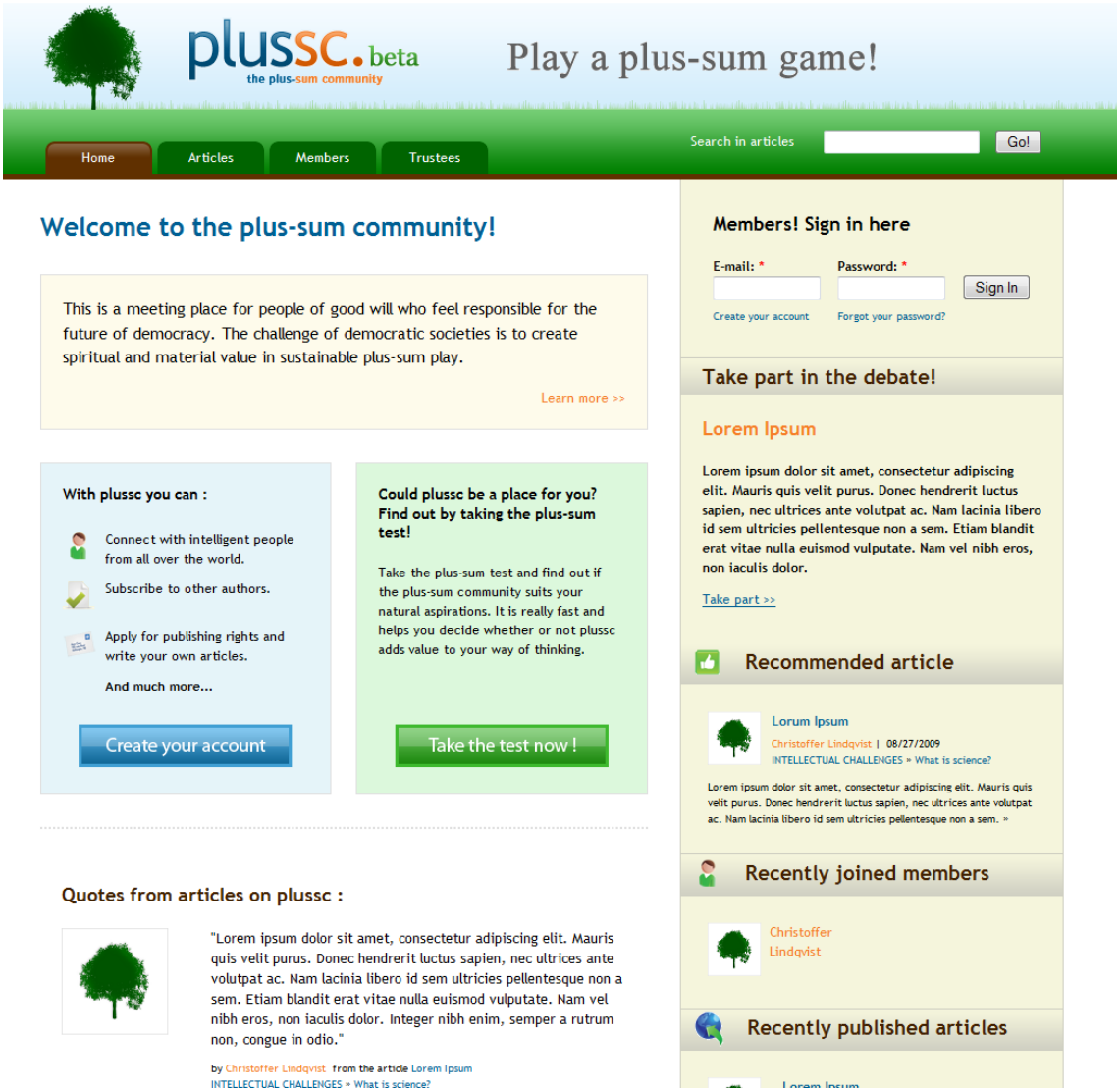
```
   $uid_list = array();
   while( $obj = db_fetch_object($result) )
   {
      $user_list[(string)$obj->uid] = $obj;
      $uid_list[] = $obj->uid;
   }
```

Det andra steget är att hämta all information om de användare som hittades. Informationen söks igen från olika tabeller i databasen och grupperas i en lista från vilken den enkelt kan hittas. Informationen returneras och visas sedan under formuläret på sidan.

```
   if( count($uid_list)>0 )
   {
      // Get the user information for the uids
      $query = "SELECT {profile_fields}.name,
                       {profile_values}.value,
                       {profile_values}.uid,
                       {user}.firstname,
                       {user}.surname
                FROM {profile_values}

                INNER JOIN {profile_fields} ON({profile_values}.fid = {profi-
le_fields}.fid)

                INNER JOIN {user} ON({user}.uid = {profile_values}.uid)

                WHERE {profile_values}.uid IN(".implode(',', $uid_list).")";

      $result = db_query($query);
      while( $obj = db_fetch_object($result) )
      {
         $user_list[(string)$obj->uid]->{$obj->name} = $obj->value;
         $user_list[(string)$obj->uid]->{'firstname'} = $obj->firstname;
         $user_list[(string)$obj->uid]->{'surname'} = $obj->surname;
        }
    }

    return $user_list;
}
```

5 DISKUSSION

Den färdiga produkten (Figur 15) blev ett fullt fungerande webbforum där användare kunde logga in och bidra med sina åsikter. Alla utvecklade moduler fungerade bra ihop med varandra och webbplatsen utgavs för betatestning för en utsedd testgrupp.



Figur 15. Framsidan för webbforumet

5.1 Drupal

Ett mål med detta examensarbete var att kunna identifiera de problem och begränsningar som användningen av Drupal medför. Drupals kraft och flexibilitet innebär en ganska

brant inlärningskurva, speciellt om man inte har tidigare erfarenhet av liknande system. För att kunna utnyttja ramverket till fullo måste man förstå dess arkitektur och framför allt hur det är tänkt att funktionalitet skall implementeras. Utan denna kunskap så skapas enkelt onödig och komplicerad kod vilken ofta orsakar mer problem än vad den egentligen löser. Dessutom kan det uppstå säkerhetsluckor i programvaran om utvecklaren inte känner till hur ramverket hanterar användardata eller autentisering av användare.

De praktiska målsättningarna för arbetet innehöll att klarlägga hur moduler utvecklas och hur man åsidosätter standard funktionalitet i Drupal. Moduler utvecklas lite olika beroende på ändamålet och har inte några egentliga standarddelar. Kapitel 4 visar med hjälp av exempel vilka delar en modul oftast innehåller samt hur de implementeras. Det förklaras också hur man åsidosätter Drupals standard funktionalitet för att presentera material på webbplatsen.

5.1.1 Fördelar

Drupal har en hel del fördelar som gör att ramverket står ut bland mängden. Systemets arkitektur möjliggör att ny funktionalitet enkelt kan läggas till i form av moduler vilka kan vara helt oberoende av varandra. Arkitekturen tillåter också att moduler enkelt kan samarbeta och utvidga varandras funktionalitet. En av de praktiska målsättningarna i arbetet var också att klarlägga på vilket sätt moduler kan samarbeta. Kapitel 4 visar att det finns ett antal krokar med hjälp av vilka det på ett smidigt sätt går att lägga till funktionalitet i andra moduler.

En stor fördel med Drupal är dess administrationsgränssnitt. Moduler har möjligheten att implementera inställningar vilka administratören kan använda för att anpassa hur modulerna samlar upp och presenterar data. Detta möjliggör att webbplatsen enkelt kan administreras utan att behöva ändra på den underliggande koden.

5.1.2 Nackdelar

Drupal är inte ett objektorienterat ramverk men systemet implementerar ändå i många fall olika objektorienterade koncept. Ett av målen med detta arbete var att utreda hur

dessa koncept är implementerade. Kapitel 3.6 redogör för hur diverse designmönster och koncept är använda och visar att den största nackdelen med att Drupal inte är objektorienterat är att alla funktioner som definieras av moduler finns i samma omfång. Systemets integritet beror därför till stor del på de regler enligt vilka utvecklare skall namnge funktionerna för att de skall kunna hittas och köras.

En annan nackdel med Drupal är att systemet är ämnat för en ganska bred användargrupp. Detta innebär att det finns en hel del färdiga moduler och lösningar i ramverket som kan vara svåra att anpassa enligt egna behov. Också databasstrukturen för de befintliga modulerna är inte i alla sammanhang den mest optimala. Detta innebär att man ibland måste skriva om redan befintliga moduler för att få den anpassade funktionalitet man söker.

5.2 Slutsats

Det är uppenbart att Drupal är ett mycket bra ramverk för att utveckla webbforum. Själva arkitekturen är planerad för att enkelt tillåta olika typer av användare skapa och publicera olika sorters material. Användarhanteringen är en viktig del av ett webbforum och Drupal har verkligen lyckats implementera den på ett sätt som både gör den enkelt att utvidga samt att administrera. Dessutom finns det en hel del standardmoduler som implementerar typisk funktionalitet som kan hittas på ett webbforum.
Det gäller dock att vara kritisk om man använder färdiga moduler eftersom de inte alltid är så bra som de verkar. Fastän själva funktionaliteten man söker finns med så kan de vara svåra att utvidga och anpassa enligt behov. Detta leder allt som oftast till att snabba och fula lösningar måste implementeras för att komma runt ett simpelt problem. Det gäller också att kontrollera de databastabeller som modulerna möjligtvis använder sig av för att försäkra sig om att de faktiskt är optimalt planerade.

För att sammanfatta så är Drupal ett riktigt bra ramverk för att bygga ett webbforum fastän systemet har en ganska brant inlärningskurva. De största fördelarna med ramverket ligger i hur arkitekturen är uppbyggd och användningen av etablerade designmönster.

KÄLLOR

Brampton, Martin. 2008. PHP5 CMS Framework Development. Packt Publishing. ISBN: 978-1-847193-57-5

Peacock, Michael. 2009. Drupal 6 Social Networking. Packt Publishing. ISBN: 978-1-847196-10-1

VanDyk, John K. 2008. Pro Drupal Development, Second Edition. Apress. ISBN: 978-1-4302-0989-8

Noble, Mark. 2008. Drupal 6 Site Builder Solutions. Packt Publishing. ISBN: 978-1-847196-40-8

Butcher, Matt. 2008. Learning Drupal 6 Module Development. Packt Publishing. ISBN: 978-1-847194-44-2

Byron, Angela. Berry, Addison. Haug, Nathan. Eaton, Jeff. Walker, James. Robbins, Jeff. 2008. Using Drupal. O`REILLY. ISBN: 978-0-596-51580-5

drupal.org. 2010 a. History. [www] [Hämtat 04.01.2010] http://drupal.org/node/769

drupal.org. 2010 b. Drupal history seen by Dries. [www] [Hämtat 04.01.2010] http://drupal.org/node/297669

drupal.org. 2010 c. The Drupal overview. [www] [Hämtat 04.01.2010] http://drupal.org/getting-started/before/overview

drupal.org. 2010 d. User: access and management settings. [www] [Hämtat 05.01.2010] http://drupal.org/node/310

drupal.org. 2010 e. System requirements. [www] [Hämtat 05.01.2010] http://drupal.org/requirements

drupal.org. 2010 f. Locale: multi-language support. [www] [Hämtat 05.01.2010] http://drupal.org/node/290

drupal.org. 2010 g. Cache support. [www] [Hämtat 05.01.2010] http://drupal.org/node/15367

drupal.org. 2010 h. Forum: create threaded discussions. [www] [Hämtat 06.01.2010] http://drupal.org/handbook/modules/forum

drupal.org. 2010 i. Comment: allow comments on content. [www] [Hämtat 06.01.2010] http://drupal.org/node/286

drupal.org. 2010 j. Poll: community voting. [www] [Hämtat 06.01.2010]
http://drupal.org/node/294

drupal.org. 2010 k. Blog: a blog for every user. [www] [Hämtat 06.01.2010]
http://drupal.org/handbook/modules/blog

drupal.org. 2010 l. BlogApi: post from blog tools. [www] [Hämtat 06.01.2010]
http://drupal.org/node/295

drupal.org. 2010 m. Aggregator: publishing syndicated content. [www] [Hämtat 06.01.2010]
http://drupal.org/node/289

drupal.org. 2010 n. Drupal programming from an object-oriented perspective. [www] [Hämtat 05.01.2010]
http://drupal.org/node/547518

drupal.org. 2010 o. Set up cron. [www] [Hämtat 20.02.2010]
http://drupal.org/getting-started/6/install/cron

drupal.org. 2010 p. Database access. [www] [Hämtat 20.02.2010]
http://drupal.org/node/101496

drupal.org. 2010 q. Handle text in a secure fashion. [www] [Hämtat 20.02.2010]
http://drupal.org/node/28984

drupal.org. 2010 r. Organizing content with taxonomy. [www] [Hämtat 06.03.2010]
http://drupal.org/handbook/modules/taxonomy

api.drupal.org. 2010 a. Database abstraction layer. [www] [Hämtat 05.01.2010]
http://api.drupal.org/api/group/database/6