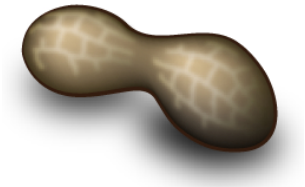
7U Content Management System

IN3700 BSc-Project

Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Opleiding Technische Informatica



TU Delft

7U Digital Handmade Originals

Door
Berend Wouda
1100726

Thomas de Ruiter
1100386

Datum
9 juli 2007

Commissie
E. van Tol
7U Digital Handmade Originals, Begeleider

P.R. van Nieuwenhuizen
TU Delft, Begeleider

B.R. Sodoyer
TU Delft, Coördinator

Dit verslag verslaat het verloop van het BSc-project van Berend Wouda en Thomas de Ruiter. Het project is een onderdeel van de bachelor opleiding Technische Informatica van de faculteit EWI van de TU Delft.

Het project behelst het ontwerpen en implementeren van een content management systeem, ter vervanging van een reeds bestaand systeem. Dit voor 7U Digital Handmade Originals, een bedrijf uit Rotterdam dat zich onder andere toelegt op het ontwerpen en implementeren van websites (al dan niet voorzien van een content management system).

Het project liep van 10 april tot en met 12 juli, in deze tijdspanne is het project opgezet, een planning gemaakt, de analyse gedaan, ontwerp geschreven en een implementatie geprogrammeerd. Dit alles verliep naarmate de tijd vorderde steeds minder goed volgens de planning. Ook is de implementatie nog niet volledig dankzij tijdgebrek.

Er is een werkend systeem, dat ondanks enige missende functionaliteit, toch zeker een solide basis kan bieden voor een goed content management system.

Voorwoord

Dit document is het *Verslag* voor het *TU Delft 'IN3700 BSc-Project'* van *Thomas de Ruiter* en *Berend Wouda*. Het verslaat het *CMS3* project gedaan in opdracht van *7U Digital Handmade Originals*.

Dit document bevat de volgende onderdelen:

- Een inleidend geheel dat het bedrijf en de opdracht introduceert.
- Een bespreking van de voorbereiding van het project. Hier worden een overzicht van het plan van aanpak en de planning en de reflectie erop gegeven.
- Een bespreking van de analyse fase van het project.
- Een bespreking van de ontwerp fase van het project.
- Een bespreking van de productie fase van het project. Hier worden een overzicht van de implementatie en gerelateerde activiteiten, en de reflectie erop gegeven.
- Een bespreking van de afronding van het project. Hier worden een reflectie op de acceptatie en de ingebruiksname gegeven.
- De conclusie van het verslag. Hier worden de resultaten en het project in het algemeen nog eens besproken.
- De 'deliverables' als appendices.

Dankbetuigingen

Tot slot willen wij graag nog bedanken:

7U voor de lekkere lunches, de goede werkomgeving en de fanatieke strijd onder de lunch.

Peter voor de goede begeleiding tijdens de gezellige bijeenkomsten.

Leonie voor het verzorgen van Thomas zodat hij harder aan het project kon werken.

Lauren voor het maken van het project logo.

Cafeïne voor die late en vroege uurtjes.

Versie

4 Juli 2007

Eerste versie. Deze versie bestaat uit de structuur en opmaak van het document, en kernachtige beschrijvingen van de inhoud per sectie.

9 Juli 2007

Definitieve versie. Deze versie is compleet, en bevat alle bijlagen.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Deel I.

Verslag

1. Inleiding

1.1. 7U Digital Handmade Originals

7U Digital Handmade Originals is een redelijk jong bedrijf uit Rotterdam dat zich bezig houdt met het ontwerpen en realiseren van multimedia producten, voornamelijk websites en webapplicaties. Naast de negen vaste werknemers zijn er vaak ook één of twee stagairs van de studie Communication & Multimedia Design (van de Willem de Kooning Academie) te vinden. 7U heeft al menig multinational tot haar klanten mogen rekenen.

1.2. Opdrachtomschrijving

Het volgende is een samenvatting van de opdrachtomschrijving, zoals die gevonden kan worden in Appendix A.

1.2.1. Aanloop

7U is bedrijf uit Rotterdam dat zich bezig houdt met het ontwerpen en realiseren van websites en webapplicaties. Om klanten de mogelijkheid te bieden zelf delen van hun website aan te passen is een eigen Content Management System gemaakt, maar door tijdsdruk, personeelsveranderingen en nog vele andere redenen heeft de ontwikkeling ervan een sterk pragmatisch karakter gehouden. Het CMS werd telkens opnieuw aangepast naar de specifieke wensen van de klant, en hierdoor is er nooit een definitieve versie van het CMS onstaan, noch bestond de noodzaak om documentatie van het CMS aan te leggen.

In 2006 zijn de eerste stappen gezet om een basis CMS samen te stellen welke zonder wijzigingen aan de back-end ingezet kan worden voor relatief eenvoudige websites. Dit is voornamelijk een grafische 'makeover' van de back-end van het bestaande CMS, welke sterk vervuild is door al zijn 'groeistuipen'.

1.2.2. Opdracht

1. Analyseer de wensen en ideeën van 7U aangaande het CMS systeem.
2. Ontwerp en ontwikkel een CMS systeem op basis van de wensen van 7U dat modulair uit te breiden is, maar waarvan de basis stabiel blijft.
3. Documenteer de code goed en zorg voor uitleg hoe het CMS uit te breiden is.
4. Tot slot zou 7U het systeem graag in de praktijk willen testen door het aan een concreet project te koppelen zodat het voorgelegd kan worden een concreet team van eindgebruikers.

De studenten zullen beide minimaal vier werkdagen per week aan het werk op kantoor in Rotterdam zijn. De projectleider zal Elja van Tol zijn, en de projectleiders en de ontwerper van de interface zullen ook bij het project betrokken worden.

2. Voorbereiding

2.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan de activiteiten gedaan omtrent de voorbereiding van het systeem. Dit houdt onder andere de volgende dingen in:

- Het plan van aanpak voor het project en onze reflectie erop.
- De planning voor het project en onze reflectie erop.

2.2. Plan van aanpak

2.2.1. Samenvatting

Het volgende is een samenvatting van het plan van aanpak, zoals die gevonden kan worden in Appendix B.

2.2.1.1. Studie

Het project moet voldoen aan de standaarden van de TU Delft. Een verslag moet geschreven worden en een presentatie moet gehouden worden.

2.2.1.2. Project

7U wil graag een CMS dat van de grond af aan opnieuw is ontworpen en geïmplementeerd om zo een solide basis te hebben die voor elke klant gebruikt kan worden. Om 7U in staat te stellen eenvoudig uitbreidingen te maken dient er documentatie te zijn aan de hand waarvan uitbreidingen ontwikkeld kunnen worden.

Het project moet daarom voor 7U ook aan een aantal voorwaarden voldoen. Deze voorwaarden (AC1 tot en met AC4, gegeven in appendix B, sectie 2.3) zijn gegeven in tabel 2.1.

Voorwaarde	Definitie
• Het prototype is een basiskern voor modulair opgebouwde eindproducten, en kan met (eindproduct-specifieke) extensies worden uitgebreid.	**AC1**
• Het prototype is compatible met PHP versie 5.2.1.	**AC2**
• Het prototype is compatible met MySQL versies 4.1.22 tot en met 5.0.37.	**AC3**
• De documentatie ondersteunt het ontwikkelen van uitbreidingsmodules (extensies).	**AC4**

Tabel 2.1.: Acceptatievoorwaarden

Daarnaast zijn er nog een aantal overige voorwaarden waaraan wellicht ook voldaan zou kunnen worden als dat mogelijk is binnen het bestek van het project. Deze voorwaarden (OC1A tot en met OC1E, ook gegeven in appendix B, sectie 2.3) zijn gegeven in tabel 2.2.

Verder geeft appendix B (sectie 2.4) ook een overzicht van alle tijdens het project in te leveren documenten.

Voorwaarde	Definitie
• De volgende modules moeten worden geïmplementeerd:	**OC1**
• Paginas	**OC1A**
• Mappen	**OC1B**
• Nieuws	**OC1C**
• Standaard Beheer	**OC1D**
• Klantvriendelijk Beheer	**OC1E**

Tabel 2.2.: Optionele voorwaarden

2.2.1.3. Aanpak

De aanpak voor het project is een klassieke vorm van software engineering: een doorloping van de fasen analyse, ontwerp, implementatie en oplevering (zie appendix B, sectie 3.1). Een gedetailleerde beschrijving van alle fasen is gegeven in appendix B (sectie 3). Tijdens alle fasen wordt documentatie geschreven, en ingeleverd (zie appendix B, sectie 2.4).

2.2.1.4. Inrichting

Appendix B (sectie 4.1 tot en met 4.5) geeft een overzicht van alle (administratieve) technieken die gebruikt worden tijden het project. De code en documentatie worden geschreven met behulp van standaarden, en alle communicatie wordt genotuleerd. Appendix B (sectie 4.6) geeft een overzicht van alle te gebruiken hulpmiddelen.

2.2.2. Reflectie

De versie gegeven in appendix B is de laatste versie van het plan van aanpak. Het plan is niet veel veranderd sinds het origineel, de meeste aanpassingen waren dingen die op het moment van maken nog niet bekend waren (bijvoorbeeld uiteindelijke inleverdata), of kleine aanpassingen ter verduidelijking van het geheel. De enige grote verandering is de reductie van de eis om PHP 4 en 5 te ondersteunen naar de eis om alleen PHP 5 te ondersteunen. Zie verder de versie notering in het voorwoord van appendix B.

Het plan van aanpak was vooral nuttig in het begin van het project, toen we er nog niet helemaal in zaten en niet echt wisten wat te doen. Het plan heeft ons geforceerd er eerst over na te denken alvorens aan de slag te gaan.

Echter, hoewel in het begin het plan nuttig was om mee te starten, bleek later dat de gekozen aanpak voor de analyse niet goed opging voor het type systeem dat moest worden gebouwd. De klassieke gebruikerscasus aanpak voor de definiëring van de functionaliteit van het systeem was niet afdoende aangezien er een hoop 'verscholen functionaliteit' in het systeem zat: ook de developer kant van het systeem moest aan de voorwaarden van de klant voldoen (zie hoofdstuk 3).

Richting het einde van het project raakte het plan steeds meer in de vergetelheid. Vooral tijdens de ontwerp en implementatie fasen werd meer gekeken naar de beschikbare tijd, en wat er belangrijk was om gedaan te worden. Daarnaast week het systeemontwerp af van wat er beschreven stond in het plan van aanpak (zie hoofdstuk 4), wat ook voor meer zorgde voor een ad hoc bedachte aanpak.

2.3. Planning

2.3.1. Samenvatting

Het volgende is een samenvatting van de planning zoals die gevonden kan worden in Appendix C.

2.3.1.1. Activiteiten

De planning bevat een activiteitenplanning en een mijlpalenkalender. Deze zijn te vinden in Appendix C hoofdstuk 1 en 2, respectievelijk. Ze zijn uiteraard gerelateerd, en worden daarom beide hier besproken.

In de eerste week worden het plan van aanpak en de planning opgesteld. Daarna word gelijk begonnen aan de analyse, gevolgd door het systeemontwerp, het objectontwerp, en de implementatie. Op de Gantt diagrammen in appendix C (figuren 1.1 tot en met 1.4, en tabel 2.1) is het exacte verloop te zien:

- 1 week voorbereiding.
- Ongeveer 3 weken analyse.
- Ongeveer 4 weken systeemontwerp.
- Ongeveer 7 weken objectontwerp.
- Ongeveer 7 weken implementatie.
- Ongeveer 3 weken afronding.

Dit zijn echter totalen, zoals op de Gantt diagrammen te zien is vinden het systeemontwerp, het objectontwerp, en de implementatie onder andere gelijktijdig plaats. De laatste twee duren 7 weken lang qua tijdsbestek (waarvan de eerste 4 weken tijdens het systeemontwerp) en vinden precies tegelijk plaats. Echter, de hoeveelheid werk (per twee personen) komt neer op het volgende (zie ook appendix C, figuur 1.1):

- 1 week voorbereiding.
- Ongeveer 3 weken analyse.
- Ongeveer 1,5 week systeemontwerp.
- Ongeveer 2,5 week objectontwerp.
- Ongeveer 2,5 week implementatie.
- Ongeveer 3 weken afronding.

Waarom de fasen tegelijk plaatsvinden wordt besproken in sectie 2.3.2.

In de planning (appendix C, hoofdstuk 1) staan ook de speciale (terugkerende) activiteiten aangegeven. Zo is er ongeveer eens in de twee weken een voortgangsgesprek en eens in de anderhalve week een management update. Ook speciale data waarop wel of niet gewerkt wordt zijn aangegeven.

2.3.1.2. Taakverdeling

Naast de tijdsplanning is er ook nog een taakverdeling. Deze komt in grote lijnen neer op het volgende:

- Het plan van aanpak, de planning, en het verslag worden opgesteld door ons beiden.
- De analyse en het ontwerp worden door beiden gemaakt.
- Berend richt zich vooral op de documentatie van de analyse en het ontwerp.
- Thomas richt zich vooral op de implementatie naar de analyse en het ontwerp.

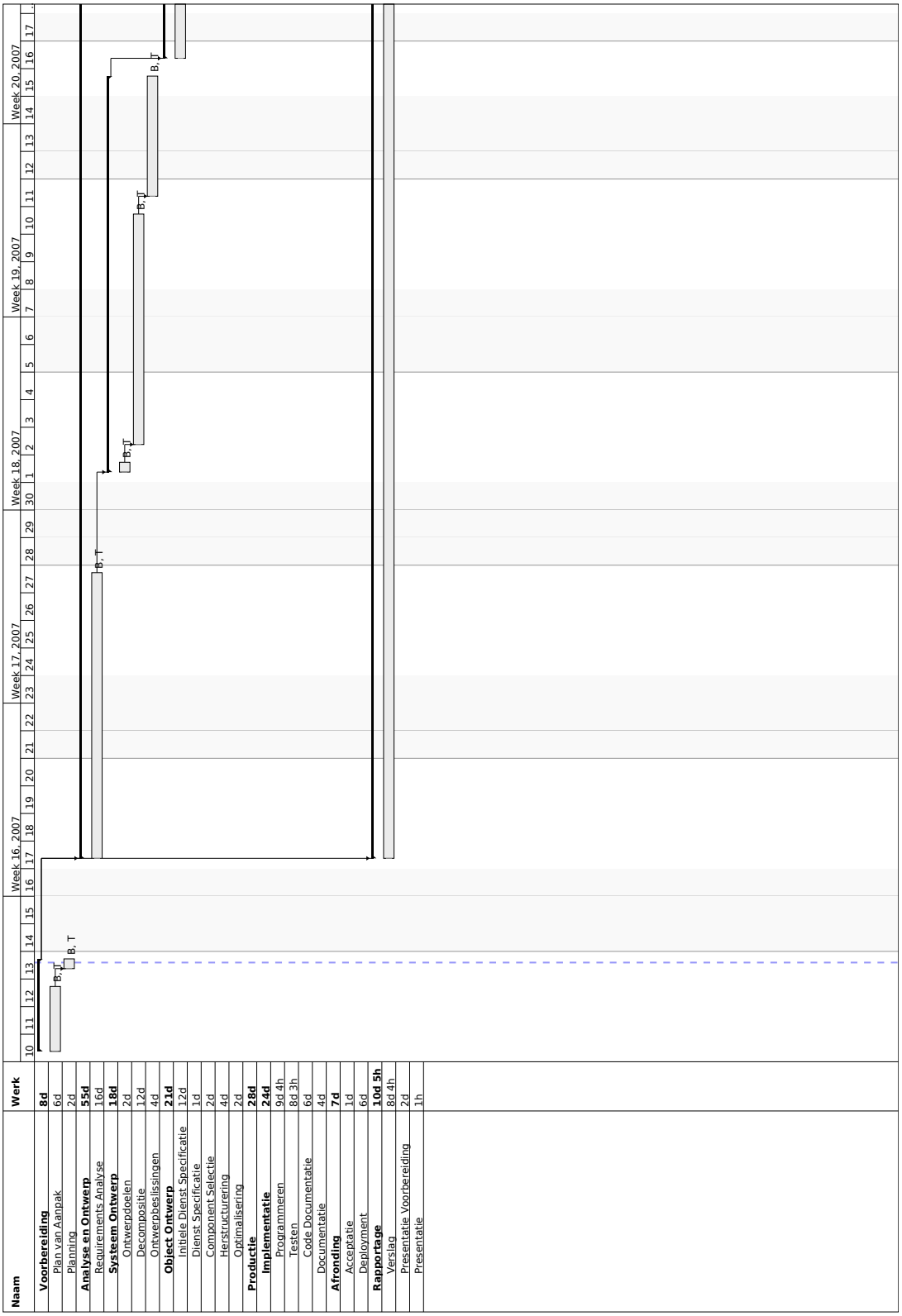
2.3.2. Reflectie

2.3.2.1. Planning

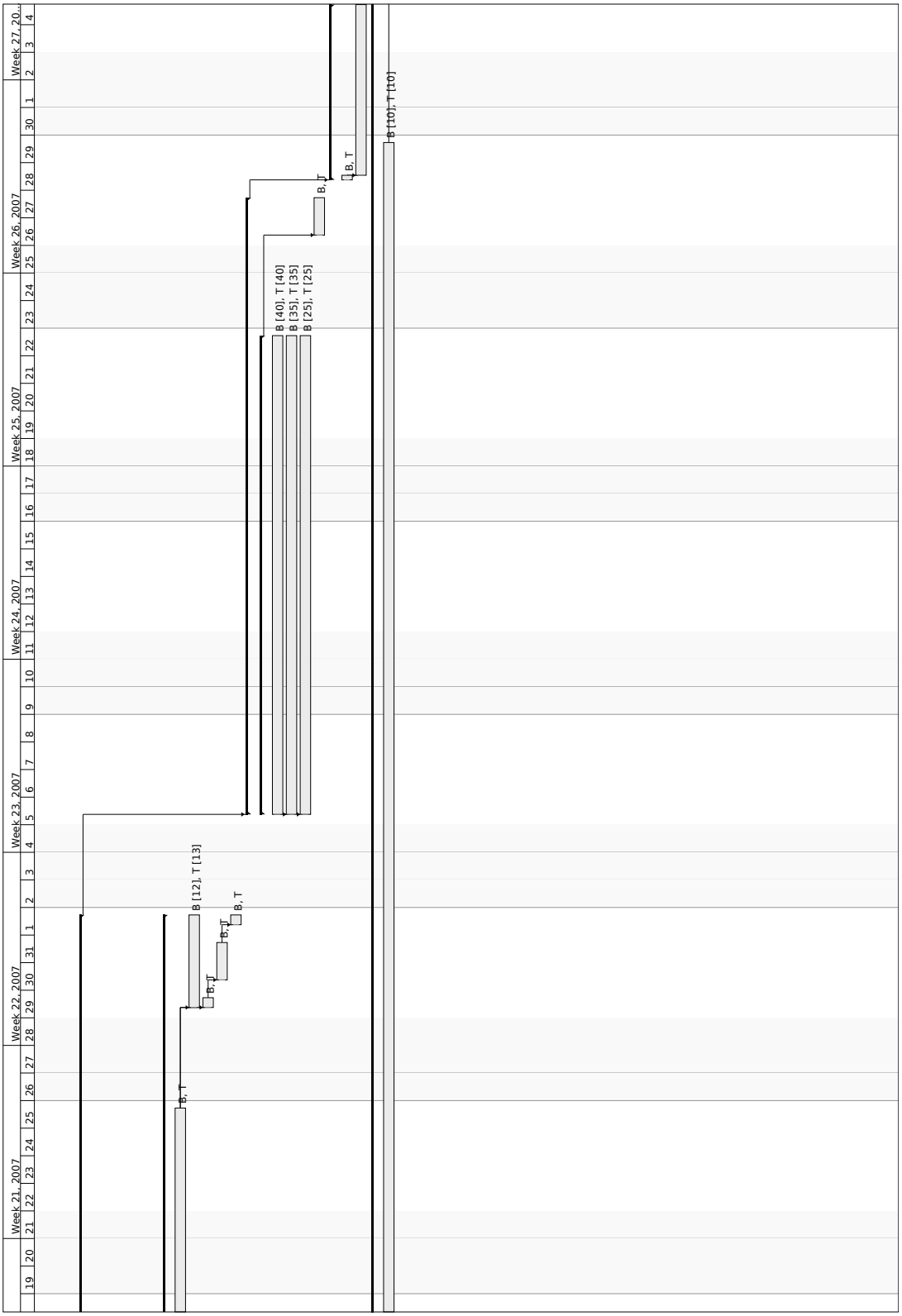
Appendix C geeft de activiteitenplanning zoals die was aan het einde van het project. De originele planning echter, is gegeven in figuren 2.1, 2.2 en 2.3. Vergelijking van de twee planningen laat zien dat er veel veranderd is.

De originele planning komt neer op:

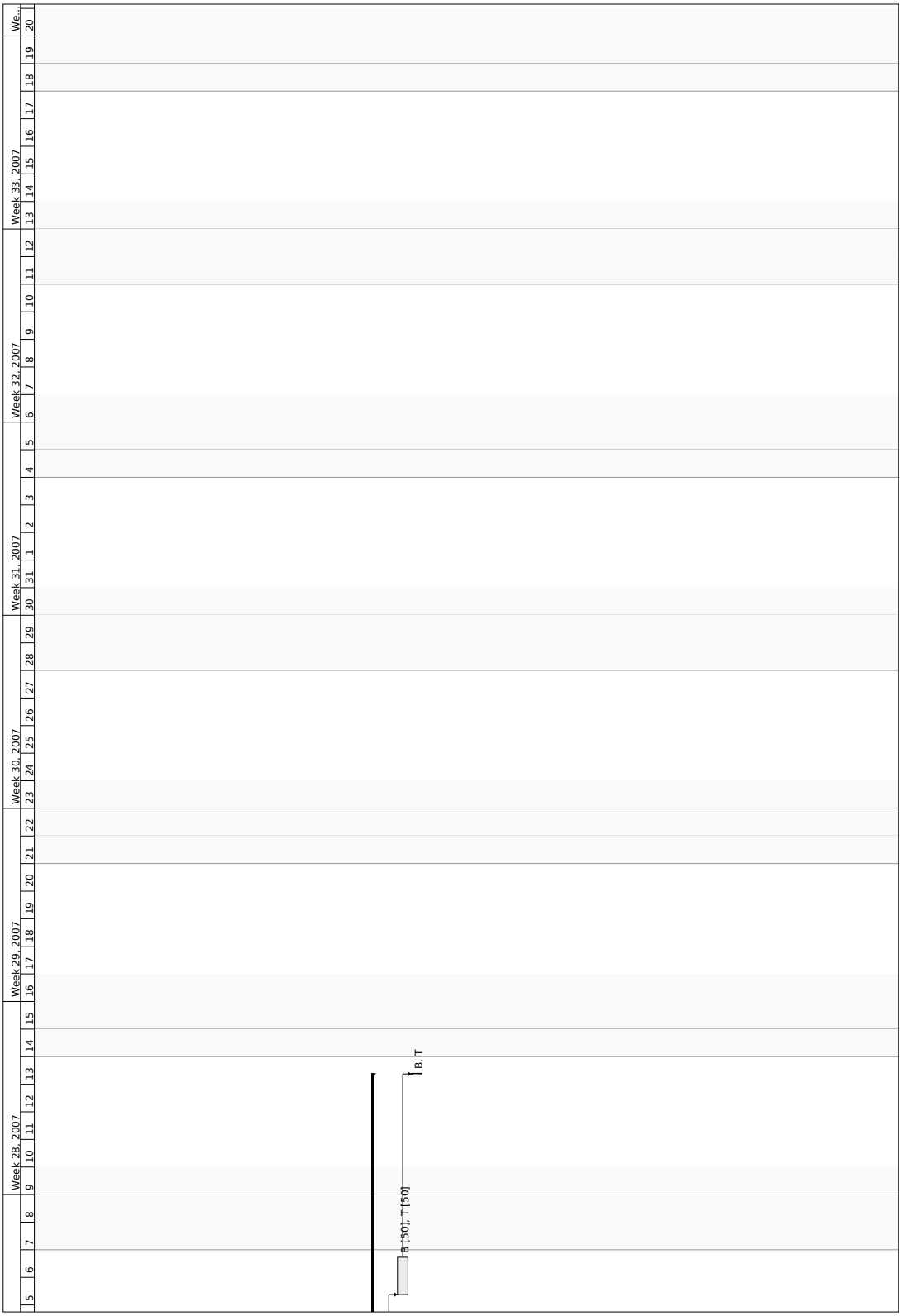
- 1 week voorbereiding.
- 2 weken analyse.



Figuur 2.1.: Gantt diagram van de originele activiteiten planning (eerste gedeelte)



Figuur 2.2.: Gantt diagram van de originele activiteiten planning (tweede gedeelte)



Figuur 2.3.: Gantt diagram van de originele activiteiten planning (derde gedeelte)

- Ongeveer 2 weken systeemontwerp.
- Ongeveer 3 weken objectontwerp.
- Ongeveer 3 weken implementatie.
- Ongeveer 3 weken afronding.

De originele versie van de planning is gemaakt aan de hand van (af)schattingen en de 'x2' regel. Na de controle door de TU begeleider werd ons aangeraden meer tijd voor de implementatie in te plannen, wat daarna ook gedaan werd.

Echter, tijdens de analysefase bleek dat de planning niet afdoende was, aangezien de fase ten opzichte van de planning uitliep van 2 naar 3 weken. Dit vereiste een herplanning aangezien er voor de rest een week minder beschikbaar was. Echter, tijdens de analyse waren wel al een paar lichte voorimplementaties gedaan als tryouts (zie hoofdstuk 5). Dit was de oorzaak dat we de systeemontwerp fase verkleinden tot 4 dagen in totaal.

Na deze verandering van de planning ontdekte de TU begeleider een gemis in tijd wanneer de totale duur en werk met elkaar vergeleken werden. Dit bleek uiteindelijk veroorzaakt door een foutieve persoonsinzetting: op sommige punten stonden personen ingepland op 110% per dag. Dit kwam doordat het verslag gepland stond als 10% werk elke dag voor de gehele duur van het project. De planning is toen aangepast zodat elke persoon 100% per dag werkte. Dit had echter als gevolg dat er meer tijd nodig was voor dezelfde hoeveelheid ingepland werk, en dus moest de hoeveelheid werk ingekort worden. Dit had weer een aanpassing tot gevolg. Na deze aanpassing waren de tijdsduren weer hetzelfde.

Tijdens de systeemontwerp fase werd al snel duidelijk dat 4 dagen veel te weinig was voor het type product dat ontworpen werd (zie hoofdstuk 4) en liep de fase vervolgens enorm uit. Dit werd gepareerd door middel van het 'officieel' maken van de voorimplementaties die al gedaan werden: de implementaties die Thomas maakte werkten zo goed dat delen van het systeem al redelijk af waren, en het systeemontwerp gelijktijdig met zekerheid door Berend kon worden gedocumenteerd. Het officieel maken van het tegelijkertijd werken aan het ontwerp en de implementatie werd gedaan door de planning aan te passen naar de uiteindelijk versie, waarin de ontwerp fasen en implementatie fase tegelijk stonden ingepland. Hiernaast werd ook nog wat tijd gewonnen door de afronding te verkorten: we hadden kennis genomen van het feit dat het verslag slechts drie dagen van te voren ingeleverd hoefde te worden in plaats van de eerder aangenomen 2 weken, en op dit moment in het project wisten we ook wel al dat de acceptatie en ingebruiksname niet of nauwelijks zouden plaatsvinden (zie hoofdstuk 6). Een gedeelte van deze tijd was echter wel nodig voor het schrijven van het verslag; van de 10% per dag was nooit wat gekomen, dus we moesten nog beginnen.

De uiteindelijke planning is echter ook niet helemaal representatief, na de uitgave is er door de tijdsdruk en de tentamens niet heel erg voldaan aan de rest van de planning zoals die toen beschreven stond. Door diezelfde tijdsdruk kwamen we niet meer toe aan het heropstellen van de planning, maar het komt er uiteindelijk op neer dat er meer tijd voor implementatie nodig was en minder tijd voor het verslag beschikbaar was. De acceptatie en ingebruiksname zijn daarom volledig overgeslagen, het systeem is daarnaast nog niet af genoeg om in gebruik te worden genomen (zie hoofdstuk 6). Het systeem is echter wel af genoeg om een demo te kunnen geven, en dit brengt het geheel tot een goed einde: de laatste week kan worden besteed aan het voorbereiden van de presentatie, zoals gepland.

2.3.2.2. Speciale activiteiten

De voortgangsgesprekken waren altijd gezellige bijeenkomsten waarbij de TU begeleider ons veel hielp. Vaak kwam het er op neer dat het wel goed zat, en werd er meer over de toekomst (van het project) gesproken.

De anderhalfwekelijkse update is nooit van de grond gekomen: de rest van het bedrijf was te druk bezig met werk dat moest gebeuren. Vooral een spoedopdracht met een hele harde deadline gooide roet in het eten in dat opzicht. De communicatie tussen ons en het management was daarom niet erg goed te noemen. Wel was er nog enigzins communicatie met de 7U begeleider.

2.3.2.3. Taakverdeling

Tijdens het verloop van het project bleek al snel dat Thomas veel liever en beter programmeerde dan documenteerde, en dat het bij Berend meer vice versa lag. Tot aan het officieel maken van de voorimplementaties deed Berend al voornamelijk de documentatie en Thomas de voorimplementaties, en daarna ging dit zo door totdat de documentatie af was (waarna beiden implementeerde). De communicatie tussen Berend en Thomas was meer dan voldoende om beiden hun werk goed te kunnen laten doen.

3. Analyse

3.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan de activiteiten gedaan omtrent de analyse van het systeem. Dit houdt onder andere de volgende dingen in:

- De analyse van het huidige systeem.
- De analyse van het nieuwe systeem.
- Onze reflectie op de analyse fase.

3.2. Huidig systeem

De volgende subsecties zijn een samenvatting van de secties 2.2 tot en met 2.5 uit appendix D.

3.2.1. Concepten

Het huidige systeem bestaat uit een aantal concepten, en wel de volgende:

- **Class:** Statisch benaderbare functionaliteit, en sjabloon voor objecten.
- **Object:** Dynamisch benaderbare functionaliteit met eigen (persistente) waarden.
- **Action:** Functionaliteit uitvoerbaar op een class of object.
- **Webservice:** Functionaliteit uitvoerbaar op een class of object, dat alleen resulterende data teruggeeft voor gebruik in de webbrowser.
- **Field:** Een persistente eigenschap van een object.
- **Relation:** Een koppeling tussen twee objecten.
- **Right:** Een recht op een object.
- **Template:** Een sjabloon waar dynamische data in voor kan komen.
- **Include:** Een object waarvan de zichtbare output in andere objecten kan worden opgenomen.

3.2.2. Functionaliteit

Het huidige systeem heeft twee vormen van functionaliteit:

- Statische aanroepen van acties of webservices op klassen.
- Dynamische aanroepen van acties of webservices op objecten.

3.2.3. Ontwerp

Het huidige systeem laat gebruikers toe requests te doen aan het systeem die statisch of dynamisch zijn. Ze kunnen acties (gewone functionaliteit) of webservices (functionaliteit gebruikt door de webbrowser voor interactiviteit) aanvragen. Acties gebruiken templates (en soms includes) om de output van een object of klasse te genereren.

Objecten kunnen koppelingen met elkaar hebben, en kunnen rechten op elkaar hebben. Objecten gebruiken fields om data in op te slaan, dit wordt naar de database doorgevoerd.

Het systeem en de modules gebruiken configuratie bestanden voor hun instellingen.

3.2.4. Problemen

De code van het huidige systeem is sterk vervuild door de continue aanpassingen en uitbreidingen. Het moet daarnaast ook nog eens backwards compatible blijven. Vele onderdelen van de code hoeven niet in een basis CMS, en sommige onderdelen worden nooit meer gebruikt.

3.3. Nieuw systeem

3.3.1. Eisen

De volgende subsecties zijn een samenvatting van de secties 4.2 tot en met 4.4 uit appendix D.

3.3.1.1. Functionele eisen

De onttrokken functionele eisen aan het nieuwe systeem zijn gegeven in tabel 3.1.

3.3.1.2. Non-functionele eisen

De onttrokken non-functionele eisen aan het nieuwe systeem zijn gegeven in tabellen 3.2, 3.3, 3.4, 3.5, en 3.6.

3.3.1.3. Pseudo eisen

De onttrokken pseudo eisen aan het nieuwe systeem en het project zijn gegeven in tabellen 3.7, 3.8, 3.9, 3.10, en 3.11.

3.3.2. Modellen

De volgende subsecties zijn een samenvatting van de secties 4.5.1 tot en met 4.5.5 uit appendix D.

3.3.2.1. Actoren

Het nieuwe systeem bevat de volgende actoren:

- **User:** Een gebruiker.
- **Power User:** Een gebruiker met administatieve rechten.
- **User Agent:** Een computer entiteit die voor een gebruiker werkt.
- **Developer:** Een ontwikkelaar die met het CMS werkt.

3.3.2.2. Scenarios

De scenarios zijn te vinden in appendix D (sectie 4.5.2).

3.3.2.3. Gebruikerscasus modellen

De gebruikerscasussen zijn te vinden in appendix D (sectie 4.5.3). Figuur 3.1 geeft een UML overzicht van de gebruikerscasussen van het systeem.

Het figuur laat zien dat de gebruikers statische, dynamische, en bestandsaanvragen kunnen doen.

3.3.2.4. Initële object modellen

De intiële object modellen zijn allen te vinden appendix D (sectie 4.5.4). Het meest interessante model is gegeven in figuur 3.2, de rest zijn redelijk standaard.

Het figuur laat zien dat elke entiteit een manager heeft, en dat klassen en objecten acties en webservices met zich hebben.

Eis	Definitie
• Een gebruiker kan een actie op een klasse aanvragen.	**FR1**
• Een gebruikersagent kan een webservice op een klasse aanvragen.	**FR2**
• Een gebruiker kan een een actie op een object aanvragen.	**FR3**
• Een gebruikersagent kan een webservice op een object aanvragen.	**FR4**
• Een gebruiker kan een bestand opvragen.	**FR5**

Tabel 3.1.: Functionele eisen

Eis	Definitie
• Toegang tot objecten moet door een rechten beheer systeem geregeld worden.	**NFR1**
• Opvraag van acties en webservices op een klasse moeten door een rechten beheer systeem geregeld worden.	**NFR2**
• Toegang tot bestanden moet door een rechten beheer systeem geregeld worden.	**NFR3**
• Het systeem mag gebruikers niet toestaan toegang te krijgen tot interne onderdelen van het systeem.	**NFR4**
• SQL injecties moeten worden voorkomen.	**NFR5**
• Cross site scripting (XSS) moet worden voorkomen.	**NFR6**

Tabel 3.2.: Veiligheidseisen (non-functioneel)

Eis	Definitie
• Als een module een fout genereerd, mag het systeem (netjes) falen.	**NFR7**

Tabel 3.3.: Kwaliteitseisen (non-functioneel)

Eis	Definitie
• Het systeem mag maximaal 10 seconden over een aanvraag doen.	**NFR8**
• Het systeem moet 10 aanvragen per seconde aankunnen.	**NFR9**

Tabel 3.4.: Performance eisen (non-functioneel)

Eis	Definitie
• Het systeem moet een 'ontwikkel mode' en een 'productie mode' hebben.	**NFR10**
• In de 'development mode' moet het systeem de performance en informatie zoals generatie tijden loggen.	**NFR11**
• In de 'development mode' moet het systeem fouten laten zien.	**NFR12**
• In de 'production mode' moet het systeem fouten bij de ontwikkelaars afleveren en de gebruiker een algemene foutmelding laten zien.	**NFR13**

Tabel 3.5.: Debug eisen (non-functioneel)

Eis	Definitie
• Een handleiding voor het maken van modules moet gemaakt worden. Het moet ondersteuning bieden voor het eenvoudig maken van nieuwe modules.	**NFR14**

Tabel 3.6.: Documentatie eisen (non-functioneel)

Eis	Definitie
• Het systeem moet in PHP geprogrammeerd worden, en compatible zijn met PHP 5.2.1.	**PR1**
• Externe software moet een bruikbare licensie hebben.	**PR2**
• Er moet een installatie applicatie voor het CMS zijn.	**PR3**
• Er moet een beheer applicatie voor modules zijn.	**PR4**

Tabel 3.7.: Project eisen (pseudo)

Eis	Definitie
• Het systeem moet in modules ingedeeld zijn.	**PR5**
• Elke module moet PHP code en bronnen zoals templates, plaatjes en andere bestanden kunnen bevatten.	**PR6**
• Modules mogen metadata bevatten.	**PR7**
• Alle PHP code in de modules moeten dezelfde naam en code conventies volgen.	**PR8**
• Modules moeten regressietesten bevatten voor ieder klasse.	**PR9**
• Externe software moet elders in het systeem ondergebracht worden.	**PR10**

Tabel 3.8.: Module eisen (pseudo)

Eis	Definitie
• Het template systeem moet automatisch de juiste template engine kiezen.	**PR11**
• Het template systeem moet minimaal Smarty ondersteunen.	**PR12**
• Het systeem mag meerdere template engines ondersteunen.	**PR13**
• Het systeem moet invoegbare objecten die in templates gebruikt kunnen worden aankunnen.	**PR14**

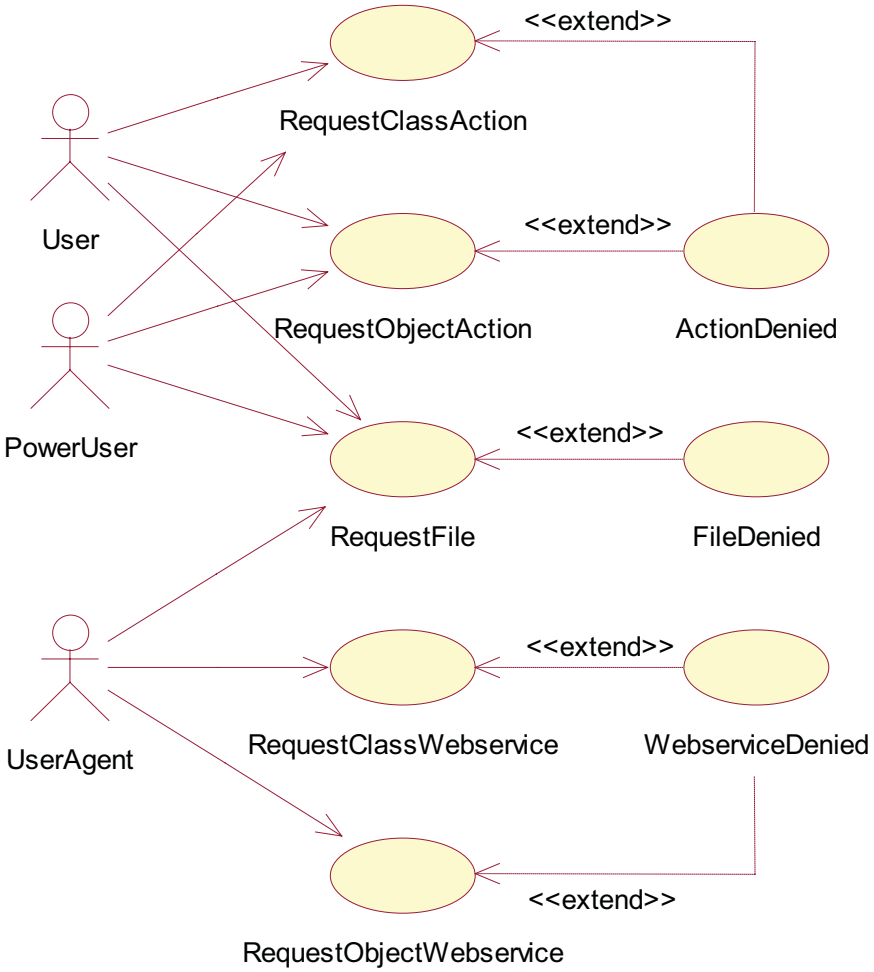
Tabel 3.9.: Template eisen (pseudo)

Eis	Definitie
• Het systeem moet een database abstractie laag hebben.	**PR15**
• Het systeem moet minimaal MySQL (versien 4.1.22 tot en met 5.0.37) ondersteunen.	**PR16**
• Het systeem mag PostgreSQL ondersteunen.	**PR17**
• Het systeem mag SQLite ondersteunen.	**PR18**

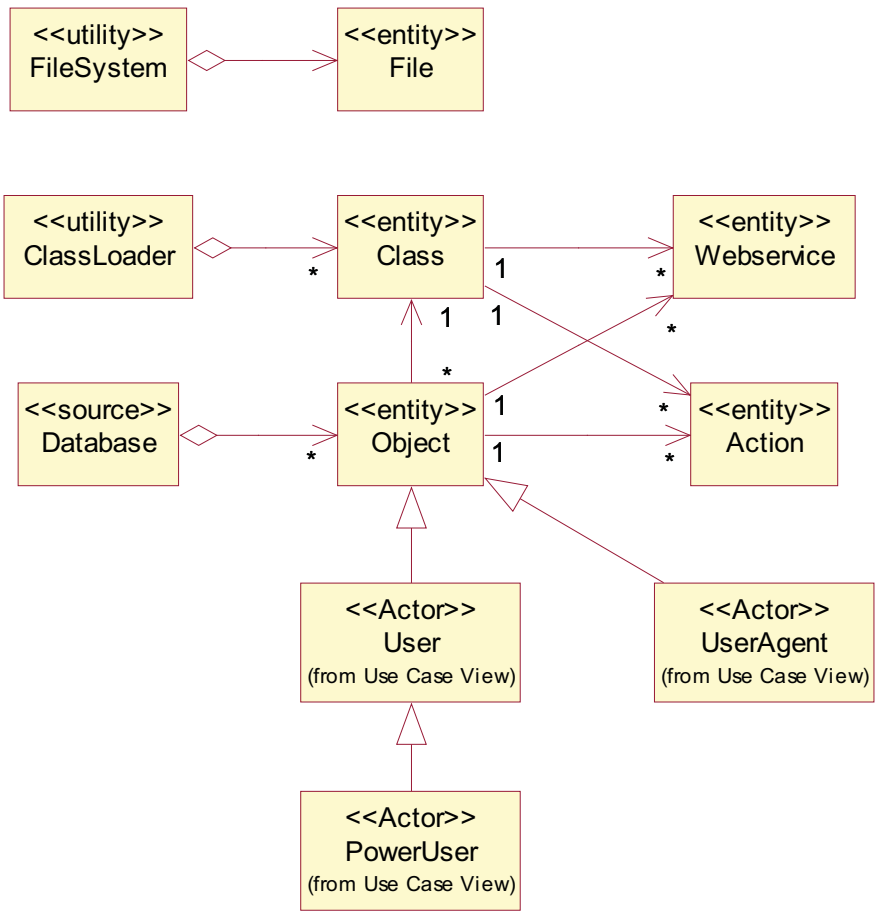
Tabel 3.10.: Database eisen (pseudo)

Eis	Definitie
• Het systeem moet een sessie klasse hebben.	**PR19**
• Alle kinderen van een object moeten in één keer verwijderd kunnen worden.	**PR20**

Tabel 3.11.: Overige implementatie eisen (pseudo)



Figuur 3.1.: Gebruikerscasus diagram.



Figuur 3.2.: Klasse diagram van de entity objecten.

3.3.2.5. Dynamische modellen

De dynamische modellen zijn allen te vinden in appendix D (sectie 4.5.5). Één van de modellen is gegeven in figuur 3.3, de rest zijn soortgelijk.

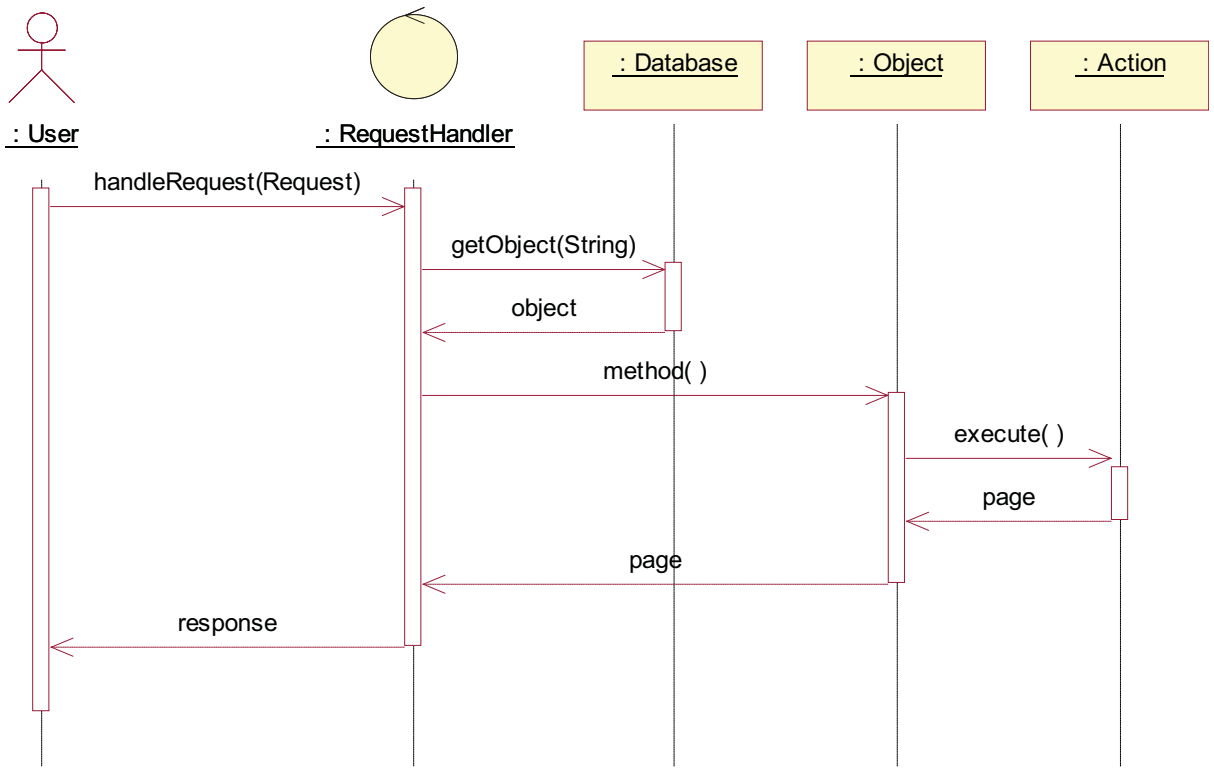
Het figuur laat zien dat de aanvraag gedaan wordt, de manager de entiteit ophaald, waarna daarop een functionaliteit wordt aangeroepen.

3.4. Reflectie

Het meest belangrijk, en wat al snel bleek, is dat de 'Requirements Elicitation' aanpak (met behulp van scenarios en gebruikerscasussen) niet de goede aanpak is voor het ontwikkelen van een systeem waarbij ontwikkelaars een onderdeel van de klantgroep zijn (zoals in ons geval een framework of basis voor uitbreidbare producten). Met dit soort systemen zijn de algemene werking, interfaces en API's heel belangrijk, en kan de klant deze in principe dicteren. Echter, met onze elecitatie methoden konden we dit niet voor elkaar krijgen. Alle methoden werken in termen van functionaliteit van het systeem, gezien vanuit de gebruiker. Een scenario betrekt het gebruik van een bepaalde mogelijk van het systeem, wat geen betrekking heeft op bijvoorbeeld het schrijven van een module voor het systeem. De zin "Elja wil een module implementeren. Hij maakt een PHP bestand aan." kwam veelvuldig voor als voorbeeld van de nuttelоosheid van de aanpak. Dit alles betekende dat we eigenlijk niet echt een basis hadden om mee te werken, wat de motivatie niet al te goed deed.

Dit probleem is uiteindelijk opgelost op twee manieren: een lange brainstorm sessie met de ontwikkelaar bracht veel van het verwachtte aan het licht, en de interfaces zijn door ons zelf opgesteld tijdens het ontwerp. Dit verlichtte ons van het proberen een systeem naar de wensen van de klant te ontwerpen; in plaats daarvan ontwierpen we een systeem naar onze kennis van standaarden en bestaande API's. Dit werkte omdat al eerder bewezen is dat een ontwikkelaar zo'n systeem kan gebruiken. En in het geval dat ze het er toch niet mee eens waren, konden we het altijd nog aanpassen. Dit was meer een soort trial-and-error aanpak.

De scenarios en gebruikerscasussen waren verder natuurlijk wel bruikbaar voor de analyse en ontwerp van de basis functionaliteit van het systeem. Vanuit deze modellen zijn dan ook de initiële objecten gevonden, en is de basis werking van het systeem vastgelegd.



Figuur 3.3.: Sequence diagram van de RequestObjectAction gebruikerscasus.

4. Ontwerp

4.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan de activiteiten gedaan omtrent het ontwerp van het systeem. Dit houdt onder andere de volgende dingen in:

- Het systeemontwerp van het systeem.
- Het objectontwerp van het systeem.
- Onze reflectie op de ontwerp fase.

4.2. Systeemontwerp

Het systeem is ingedeeld in twee subsystemen, welke eigenlijk systemen op zich zijn. Het basis framework is Amy, een abstractielaag voor vele zeer bruikbare en innovatieve functionaliteiten. Op deze basis draait Kif, de CMS implementatie.

Het complete systeemontwerp is gegeven in appendix E.

4.2.1. Amy

De volgende subsecties zijn een samenvatting van de secties 3.1 tot en met 3.4 uit appendix E.

4.2.1.1. Overzicht

Amy is een abstactielaag die vele standaarden voorschrijft en daardoor ook uniforme toegang biedt.

Het bevat in de kern een manager voor bronnen zoals klassen en bestanden, die geladen kunnen worden vanaf het bestandssysteem of vanuit een gecompresseerd bestand. De klassen en bestanden zijn ingedeeld in packages, welke verdeeld zijn over modules. Dit maakt het systeem modulair en overzichtelijk. Vanuit de code is dit allemaal geen probleem, omdat de manager aangesproken kan worden voor het ophalen van een bestand.

Amy biedt een database abstractie laag die MDB2[1] gebruikt als basis.

Amy heeft ook ondersteuning voor persistente objecten door middel van 'peanuts'. Dit zijn objecten met geannoteerde attributen die automatisch in bijvoorbeeld een database kunnen worden opgeslagen.

Amy's web ondersteuning heeft de vorm van servlets. Dit zijn klassen die aanvragen kunnen regelen.

Het maken van webpaginas gebeurt door middel van templates en template engines.

Naast deze kerntechnologiën bevat Amy ook nog een aantal herbruikbare hulp klassen.

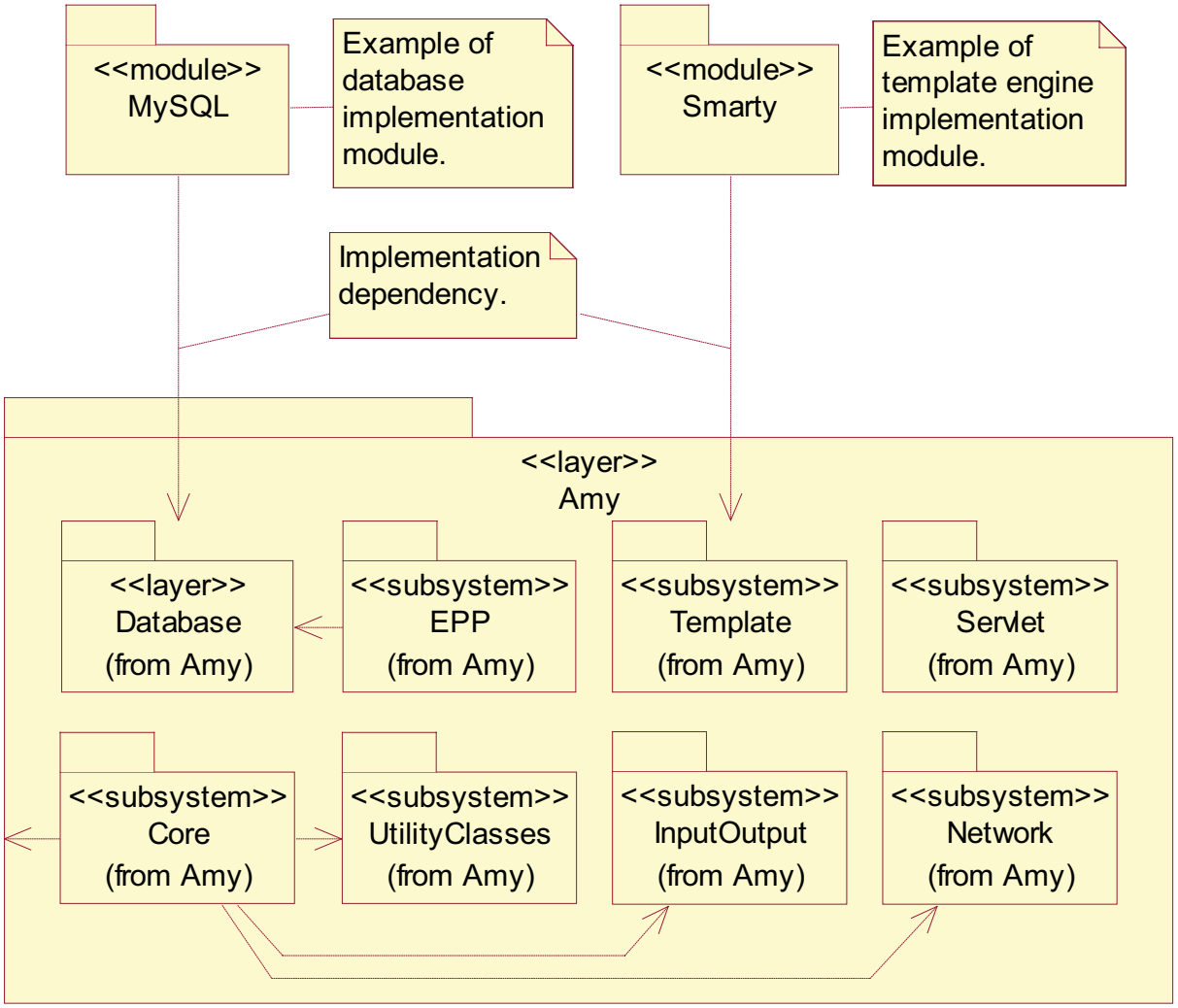
4.2.1.2. Ontwerpdoelen

Amy is opgesteld om foutbestendig, veilig, uitbreidbaar, modificeerbaar en overzetbaar te zijn. Daarnaast moet de code leesbaar zijn en moet het systeem bruikbaar en eenvoudig zijn voor ontwikkelaars. Zie verder appendix E (sectie 3.2).

4.2.1.3. Architectuur

Figuur 4.1 geeft een overzicht van de opbouw van Amy's subsystemen. De correspondentie van de subsystemen naar hardware en software is erg eenvoudig, en te zien in appendix E (sectie 3.3.2). De correspondentie geeft ook de bestandsindeling weer.

[1] Een database abstractie laag gemaakt door de PEAR groep. Zie hoofdstuk 5.



Figuur 4.1.: Amy Subsysteem Decompositie

Het figuur laat zien dat Amy is opgebouwd uit een aantal subsystemen waarvan sommige elkaar nodig hebben, en waarvan de bovenste belangrijk zijn voor de interface van de laag.

Ook is te zien dat Smarty and MySQL de respectievelijk template engine en database abstractie laag implementeren.

4.2.1.4. Ontwerpkeuzes

Appendix E (sectie 3.4) beschrijft de ontwerpkeuzes die gemaakt zijn voor Amy.

Belangrijk is de hoe hoe het systeem draait, aangezien het een webapplicatie is die op PHP draait. PHP start met iedere request overnieuw op, en zodoende wordt de opstart procedure van Amy telkens doorlopen. De klassen worden geladen, de aanvraag wordt geprepareerd, en uiteindelijk wordt de juiste servlet aangeroepen. Deze voert dan de aangevraagde functionaliteit uit.

De persistentiteit van objecten wordt geregeld door de peanuts en de database abstractielaag.

4.2.2. Kif

De volgende subsecties zijn een samenvatting van de secties 4.1 tot en met 4.4 uit appendix E.

4.2.2.1. Overzicht

Kif definiëert 'virtuele' Kif objecten, die intern uit een aantal objecten bestaan. Een aantal van deze objecten neemt een vorm van functionaliteit voor zijn rekening, maar er zijn ook persistentie leverende peanuts en templates voor de resultaten.

4.2.2.2. Ontwerpdoelen

Kif is opgesteld om responsief, snel, robuust, foutbestendig, veilig, goedkoop in uitbreiding, en aanpasbaar te zijn. De code moet ook leesbaar en nagaanbaar zijn. Verder moet het systeem bruikbaar en eenvoudig zijn voor ontwikkelaars. Zie verder appendix E (sectie 4.2).

4.2.2.3. Architectuur

Figuur 4.2 geeft een overzicht van de opbouw van Kif's subsystemen. De correspondentie van de subsystemen naar hardware en software is erg eenvoudig, en te zien in appendix E (sectie 4.3.2). De correspondentie geeft ook de bestandsindeling weer.

Het figuur laat zien dat Kif op Amy berust, en gebruikt maakt van de interface aangeboden door de laag. Amy op haar beurt laadt de klassen en bronnen van Kif.

Verder is te zien dat Kif uit twee subsystemen bestaat: een subsysteem dat interfaces en standaard implementaties voor Kif objecten voorschrijft, en de hoofd servlet die de functionaliteitsverdeling voor zijn rekening neemt.

4.2.2.4. Ontwerpkeuzes

Appendix E (sectie 4.4) beschrijft de ontwerpkeuzes die gemaakt zijn voor Kif.

Kif's systeem uitvoering begint waar de van Amy eindigt. De servlet is aangeroepen en de juiste entiteit wordt geladen en aangeroepen. Deze genereert een resultaat, wat vervolgens teruggestuurd wordt.

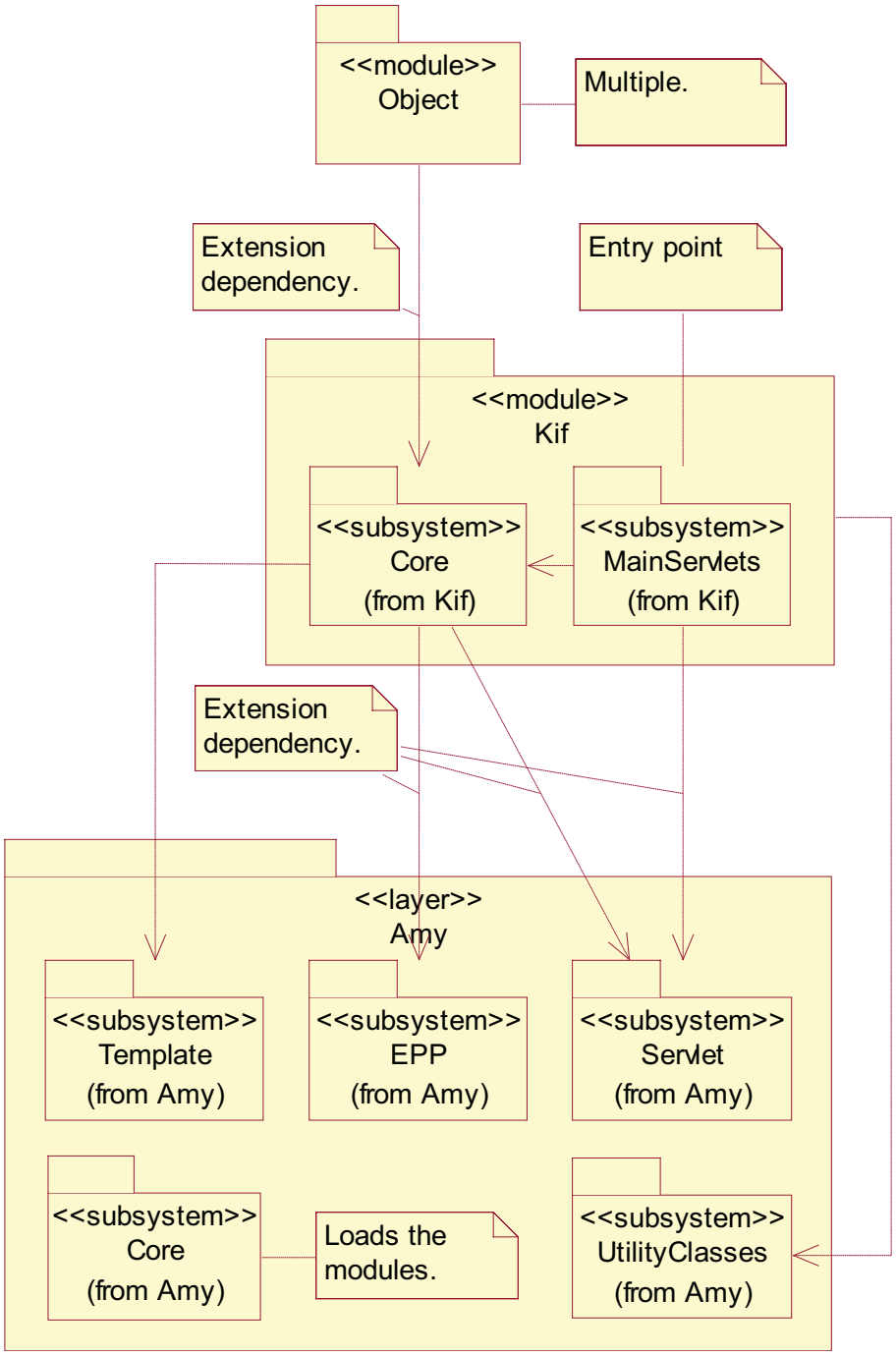
Kif's persistentiteit maakt geheel gebruik van Amy's peanuts.

4.3. Objectontwerp

Het objectontwerp is gegeven in appendix F. Wegens de enorme grootte van het document is het ingekort tot alleen de tekst en slechts een aantal voorbeelden van de API documentatie.

In appendix F (hoofdstuk 1) worden de naamconventie en programmeerstandaard gegeven. Deze komen in grote lijnen overeen met de PHP en Java standaarden.

De volgende subsecties zijn een samenvatting van de hoofdstukken 2 en 3 uit appendix F.



Figuur 4.2.: Kif Subsysteem Decompositie

4.3.1. Amy

Voor Amy zijn een aantal trade-offs gedaan: de veiligheid en mooie organisatie en abstractie hebben een negatieve invloed op de antwoord tijd. Dit wordt geprobeerd te verhelpen door middel van caching.

Amy's package en klasse organisatie en beschrijving zijn te zien in appendix F (secties 2.2 en 2.3 respectievelijk).

4.3.2. Kif

Voor Kif zijn ook een aantal trade-offs gedaan: zoals bij Amy hebben de (extra) veiligheid en organisatie en abstractie ook een negatieve invloed op de antwoord tijd.

Kif's package en klasse organisatie en beschrijving zijn te zien in appendix F (secties 3.2 en 3.3 respectievelijk).

4.4. Reflectie

4.4.1. SDD

Oorspronkelijk was voor het SDD 4 dagen ingepland. Dit leek ons voldoende omdat we al een hoop hadden gedaan tijdens de analyse fase wat niet strict analyse was, en het SDD niet een heel groot document was in onze methode.

Dit bleek echter grandioos te kort te zijn, toen we erachter kwamen dat het SDD de perfecte plek was om al onze ontwerpideeën te documenteren. Dit leidde tot een splitsing van het document in een stuk over Amy en een stuk over Kif, om ons framework systeem idee te profileren. Daarnaast kwam er bij elk systeem ook nog eens een overzichtshoofdstuk, wat alle speciale ideeën en ontwerpen beschreef die anders niet in het SDD voorkomen. Deze hervorming van het SDD naar een versie die ons schikte kostte nogal wat tijd, en dit veroorzaakte de vertraging in de planning op dat punt.

Nu we een document hadden met onze ideeën konden deze echter wel naar de begeleiders toe gecommuniceerd worden.

Het SDD schortte echter aan een paar kanten. Het is het enige document wat niet officieel afgekomen is, aangezien er een paar secties missen.

Tijdens het schrijven van het SDD hadden we geen idee hoe we de rechten en veiligheid moesten aanpakken. We hebben dit toen laten zitten voor later, en de desbetreffende onderdelen van het SDD leeggelaten. Hetzelfde geldt voor het database schema.

Daarnaast was er nog het hoofdstuk over subsysteem diensten, wat gebruikt wordt om een centrale interface specificatie te houden tussen ontwikkelaars. Deze wordt normaal langzaam ingevuld tijdens de systeem- en objectontwerpen, maar wij kwamen daar niet aan toe. Ons objectontwerp werd automatisch gegenereerd en aangezien we maar met twee personen waren was het nut van het hoofdstuk eigenlijk ook ver te zoeken. Het niet invullen van dit hoofdstuk heeft ons een hoop tijd gescheeld.

Deze drie secties zijn dus nooit ingevuld. Ondertussen zijn er wel ideeën en implementaties voor de rechten en veiligheid, maar die zijn wegens tijdsgebrek niet meer gedocumenteerd in het SDD.

4.4.2. Database abstractie

In het originele ontwerp was een database abstractie laag toegekend aan Amy. Deze was bedoeld als uniforme toegang tot verschillende databases, met datatypes als peanut eigenschappen. Dit was redelijk veel werk, en hoewel we al aardig op weg waren is later in het project toch gekozen voor een al bestaande database abstractie, genaamd MDB2 (zie hoofdstuk 5). De grootste reden om dit te doen was omdat de MySQL functies in PHP veel te wensen overlieten op het gebied van 'prepared statements'. Na een aantal types uitgeprobeerd te hebben is MDB2 onderzocht en geinstalleerd (het is niet een al geïntegreerd deel van PHP zoals de andere MySQL functies) en is er een lichte database abstractie laag over heen gemaakt. Het is echter nog steeds mogelijk (en legaal) om direct met MDB2 te werken.

Het gebruik van MDB2 had ook invloed op het ontwerp van de peanuts. Deze zouden oorspronkelijk datatypes gebruiken als attributen, maar deze vervielen toen MDB2 in gebruik werd genomen. MDB2 specificeert echter zelf ook al een aantal datatypes, dus deze zijn consequent gebruikt. Het enige nadeel

is dat dit geen objecten zijn, en we dus niet direct geavanceerde functionaliteit aan datatypes kunnen verbinden. De huidige methoden gebruiken echter al reflectie, dus dit is een waarschijnlijke vervanger.

4.4.3. Verloop van de ontwerpfasen

Zoals als in de planning (sectie 2.3.2) te zien was, verliepen de systeemontwerp, objectontwerp, en implementatie fasen redelijk tegelijk. De reden hiervoor is dat tijdens de analyse fase al begonnen was met het maken van verkennende voorimplementaties. Deze waren nodig omdat we onbekend waren met PHP 5, en we moesten weten wat mogelijk was voordat we wilde plannen gingen maken.

Veel bleek mogelijk met PHP 5, en tegen de tijd dat de analysefase was afgerond was zodoende al een basissysteem beschikbaar. Dit werd gevolgd door het systeemontwerp, en zoals eerder vermeld konden we ons helemaal in het SDD uitleven. Doordat Thomas liever programmeerde schreef Berend de documentatie ongeveer tegelijk met de implementatie zoals die voortkwam. Sinds het ODD gegenereerd wordt uit de code, is dit waarom de implementatiefase tegelijk plaatsvond met de systeem- en ontwerpfasen.

Hoewel dit niet een gelukkige aanpak lijkt, bleek het voor ons framework zeer goed te werken. Een ieder deed waar hij goed in was, er was geen overlap in wat gedaan werd dus communicatie was niet kritiek. Het ontwerp hield de implementatie uit de knoop terwijl de implementatie het ontwerp verifiëerde. Op een gegeven moment was het ontwerp klaar en konden beiden aan de implementatie. Dit is naar onze mening de reden waarom we toch een groot deel van de implementatie nog hebben weten te voltooien.

5. Productie

5.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan de activiteiten gedaan omtrent de implementatie van het systeem. Dit houdt onder andere de volgende dingen in:

- De implementatie van het systeem.
- Een demonstratie van het systeem.
- Het testen van de implementatie.
- De ontwikkelomgeving.
- Onze reflectie op de implementatie fase.

5.2. Implementatie

Aan de implementatie van het systeem is eerder begonnen dan oorspronkelijk in de planning stond. Dit gebeurde omdat er onvoldoende ervaring was met PHP 5. Thomas heeft een aantal dingen geprobeerd te implementeren om zodoende de mogelijkheden van PHP 5 te verkennen. Deze elementen zijn in het systeemontwerp verwerkt en deze implementaties hebben de basis voor de verdere ontwikkeling van het systeem gevormd.

5.2.1. Voorimplementatie

Om te onderzoeken of het mogelijk was in PHP 5 onze ideeën ook daadwerkelijk te implementeren, is Thomas tijdens de ontwerpfase al begonnen met het implementeren van een basis waarop het verdere systeem gebouwd kon worden. Deze basis is uiteindelijk onder de naam 'Amy' uitgegroeid tot een implementatie van een kleine subset van de Java EE JRE in PHP 5. Dit was voornamelijk noodzakelijk om voldoende abstractie te kunnen bieden, zoals bijvoorbeeld de verschillende voorkomens van de modules zoals in sectie 5.2.1.2 wordt besproken.

5.2.1.1. ClassLoader

PHP 5 heeft van zichzelf geen ondersteuning voor packages. Om toch packages te kunnen gebruiken is er een ClassLoader geprogrammeerd die in staat is een klasse te laden aan de hand van zijn volledige naam. De volledige naam van de klasse is de naam van de klasse inclusief de naam van het package van de klasse.

Met de `__autoload` functie van PHP 5 wordt automatisch geprobeerd de klassen te laden wanneer ze worden gebruikt.

5.2.1.2. ResourceManager

Naast de indeling in klassen is er ook een indeling in modules. Een module bevat delen van packages, en packages kunnen worden gespreid over meerdere modules. Om elementen (verder resources) uit een package te kunnen laden is een ResourceManager geprogrammeerd. Deze ResourceManager kan Resources vinden en geeft de mogelijkheid middels een InputStream de inhoud van een Resource te lezen.

Modules zijn te vinden in twee varianten. De eenvoudigste variant is een map in het filesysteem, de andere variant is een Amy-file, een zipfile met daarin dezelfde inhoud die de map van de eerste variant zou hebben.

Door de abstractie gegeven door de ResourceManager ziet de rest van het systeem geen verschil tussen de verschillende type modules en zouden andere types eenvoudig kunnen worden geïmplementeerd.

5.2.1.3. BootClassLoader

Na de implementatie van de ResourceManager is de implementatie van de ClassLoader zodanig aangepast dat deze gebruik kon maken van de ResourceManager. Om de ResourceManager en de ClassLoader te kunnen laden is toen de BootClassLoader geschreven. Deze kan uit beide soorten modules precies voldoende klassen laden dat minimaal nodig is voor de werking van de echte ClassLoader.

5.2.2. Amy

Zoals gezegd vormden de hiervoor genoemde subsystemen de basis van Amy. Op deze basis zijn een aantal uitbreidingen gedaan om er een framework van te maken.

5.2.2.1. Instantie

Naast de modules die gedeeld kunnen worden tussen verschillende instanties van het systeem, was er ook een instantie specifiek subsysteem nodig. Dit subsysteem is zo klein mogelijk gehouden: het bestaat uit de BootClassLoader, een configuratie file en een aantal lege directories die bedoeld zijn voor het bevatten van modules, software van derden, opslag van tijdelijke bestanden en opslag van geüploade bestanden.

5.2.2.2. Database abstractie

Reeds vroeg in het implementatie traject is begonnen aan het schrijven van een database abstractie laag. Uiteindelijk is er toch besloten het overgrote deel hiervan uit te besteden door gebruik te maken van MDB2, een reeds door het PEAR[1] project geschreven database abstractie laag. Berend heeft enkele functies van MDB2 ingekapseld in eigen klassen om een aantal standaardinstellingen te zetten en fouten als excepties te kunnen gooien.

5.2.2.3. Peanuts

Berend heeft zich, toen hij aan het implementeren sloeg, gestort op de Peanuts en de PeanutManagers. Voor de Peanuts is een 'Object/Resource Mapper' geprogrammeerd dat met behulp van reflectie de phpDoc van Peanut klassen inziet en daar de '@persistent' annotaties uit leest. Peanut objecten met attributen met een dergelijke annotatie krijgen automatisch een eigen tabel in de database waarin de waarden van de betreffende attributen van de Peanut worden bijgehouden. Het subklassen van Peanuts vormt hiervoor geen probleem, de enige voorwaarde is dat persistente eigenschappen niet overschreven worden.

Amy bevat naast de basis EntityPeanut (een peanut met persistente eigenschappen) ook een StandatdEntityPeanut die het systeem in staat stelt een hierarchische boom van peanuts te onthouden.

Naast de database implementatie zijn ook andere persistentiteits implementaties mogelijk, met behulp van de interface for the PeanutManagers. De manager kan ook uitgebreid worden door middel van extensie als nieuwe functionaliteit nodig is voor bijvoorbeeld een nieuw type peanut (dit is ook gedaan voor de StandardEntityPeanut).

5.2.2.4. Templates

Na de implementatie van de ResourceServlet (zie 5.2.3.2) is de ondersteuning voor templates geschreven. Dit is gedaan in de vorm van een eenvoudige TemplateEngine interface en een TemplateEngineFactory. De factory kijkt met aan de hand van de extensie (zie 5.2.2.6) van het template wat voor type template het betreft en maakt de bijbehorende TemplateEngine aan. Tot nu toe is er slechts ondersteuning

[1] `http://pear.php.net/`

geïmplementeerd voor de template engine Smarty[2], maar niets staat het gebruik van andere engines in de weg.

5.2.2.5. Servlets

Gelijk met het begin van de ontwikkeling aan de oorspronkelijke database abstractie laag is ook begonnen met het ontwikkelen van de Servlets met de bijbehorende ServletContainer, Request en Response. De ServletContainer bevat tevens de statische main-methode die wordt aangeroepen nadat de BootClassLoader zijn werk heeft gedaan. Deze methode maakt de ServletContainer aan en start de in de configuratie ingestelde standaard servlet. De ServletContainer en de Servlets moeten door de aard van PHP helaas voor elke request opnieuw worden geladen. De Servlet objecten kunnen dan ook niet worden gebruikt voor het opslaan van algemene persistente data. Hiervoor kunnen Sessies en Peanuts worden gebruikt.

5.2.2.6. Hulpklassen

Tijdens het ontwikkelen ontstond de wens om een aantal zaken te vereenvoudigen, zoals bijvoorbeeld het bijhouden van log-berichten, het vinden van het bestands-type aan de hand van de extensie of het lezen van een configuratie-bestand. Hiervoor zijn klassen geschreven die in het amy_util package zijn opgenomen. De genoemde functies zijn ook al in het systeemontwerp opgenomen, maar bijvoorbeeld de Path klasse, met een methode om het relatieve pad tussen twee absolute paden te geven, is later pas geschreven. Hetzelfde geldt voor de annotatie hulpklassen. Deze waren niet in het ontwerp aanwezig, maar zijn in het kader van de herbruikbaarheid in het leven geroepen.

5.2.2.7. PermissionManager

Om op peanuts rechten te kunnen bijhouden is een PermissionManager geprogrammeerd in twee smaken. In de eerste plaats is er een DummyPermissionManager geschreven die altijd iedereen alle rechten geeft, deze kan worden gebruikt in situaties waar eigenlijk geen rechtenbeheer nodig is. Daarnaast is een DatabasePermissionManager geschreven die in een database tabel de rechten op peanuts bij kan houden.

De PermissionManager wordt meegeven aan de PeanutManager die vervolgens hiermee voor elke bewerking kan controleren of de gebruiker wel het recht heeft deze bewerking uit te voeren.

Het controleren van de rechten is helaas ook nog niet daadwerkelijk geïmplementeerd. Daarnaast moet er ook nog een systeem worden geprogrammeerd om bij te houden dat iemand is ingelogd, en deze informatie beschikbaar stelt aan de PermissionManager.

5.2.2.8. VendorManager

Voor het gebruik van software van derden binnen het systeem is een VendorManager in het leven geroepen. Deze geeft het absolute pad naar bestanden uit software van derden uit een van de 'vendors' mappen (die gedeeld kunnen worden tussen verschillende instanties van het systeem). De locaties van de mappen worden in het configuratiebestand ingesteld.

De VendorManager is bedoeld om een goede scheiding met software van derden mogelijk te maken. Hij wordt vooralsnog slechts gebruikt voor het laden van Smarty in de betreffende template engine.

Wat nog niet is geïmplementeerd, maar wat wel wenselijk is, is het aanspreken van 'vendor' bestanden als een resource. Dit zou het mogelijk maken om ook software van derden te gebruiken in webapplicaties die worden geschreven met behulp van dit systeem, bijvoorbeeld een Javascript HTML editor als de FCKeditor[3].

5.2.3. Kif

Kif is parallel aan Amy ontwikkeld. Aan de ene kant werd de implementatie van (onderdelen van) Amy getest door het gebruik in Kif. Anderzijds werd tijdens de implementatie van Kif steeds die onderdelen voor Amy geïmplementeerd die nodig waren voor de verdere implementatie van Kif.

[2]Een bekende template engine voor PHP. Zie `http://smarty.php.net/`.
[3]De FCKeditor wordt in het oude systeem reeds veelvuldig gebruikt, zie `http://www.fckeditor.net/`.

5.2.3.1. Servlet

Het kif_Servlet is het punt waar je Kif binnen komt. Dit servlet wordt vanuit de ServletContainer aangeroepen omdat het in de configuratie staat ingesteld als standaard servlet. Het servlet dispatched afhankelijk van het URL het request naar een ander servlet.

Voor het dispatchen van het request wordt gekeken of het een resource, statisch dan wel dynamisch request betreft. Dit is eenvoudig aan het URL te zien. Resource requests beginnen (relatief ten opzichte van het systeem) met een apenstaartje '@', statische requests beginnen met een dollar teken '$'. De overige requests betreffen dynamische requests.

De informatie die is ontdekt betreffende het request word in de servletcontext bijgehouden voor gebruik in de servlets waarnaar wordt gedispatched. Voor dynamische requests wordt ook de betreffende peanut opgevraagd en in de context opgeslagen, vervolgens wordt aan de hand van de klassenaam van de peanut de juiste klassenaam van de bijbehorende servlet geconstrueerd, als deze klasse niet bestaat wordt teruggevallen op de DefaultServlet. Tot slot wordt het request naar de gevonden klasse gedispatched.

5.2.3.2. ResourceServlet

Het ResourceServlet is na het kif_Servlet het eerste servlet dat is gebouwd voor Kif. Dit servlet heeft als doel statische content te serveren, zoals bijvoorbeeld plaatjes, Javascript of stylesheets.

Alle URL's waarvan het pad (binnen het systeem) begint met een apenstaartje '@', worden afgehandeld door dit servlet. Het deel van het pad na het apenstaartje duidt een resource aan. De ResourceManager wordt gebruikt om de betreffende resource te vinden en te serveren.

Ter beveiliging worden alle resources met PHP code geblokkeerd. Een alternatief beveiligingssysteem is besproken maar vanwege het gebrek aan tijd nog niet geïmplementeerd. Het idee is rechten te beheren aan de hand van de bestandsnaam van een resource. Standaard zouden dan resources niet toegangkelijk zijn, bestanden waarvan de naam begint met een underscore '_' zijn slechts toegangkelijk voor ingelogde mensen en bestanden waarvan de naam begint met een plus '+' zijn voor iedereen toegangkelijk.

5.2.3.3. Standaardklassen

Voor Kif is een set standaard klassen geïmplementeerd. Dit betreft DefaultPeanut en alle bijbehorende klassen: DefaultStaticActions, DefaultStaticServices, DefaultDynamicActions, en DefaultDynamicServices. Deze klassen dienen zowel als basis om eigen peanuts voor Kif op te baseren, als om te worden gebruikt als fall-back, indien er voor een Peanut geen eigen bijbehorende klassen zijn geschreven. Deze klassen bieden basisfunctionaliteit, zoals het tonen van de peanut door het uitvoeren van een template, en het inkapselen van return waarden van services in JSON code.

5.3. Demonstratie

Ter demonstratie en wellicht om als basis te dienen voor het toekomstige systeem, is er een beheerapplicatie geschreven. Met behulp van deze applicatie is het mogelijk peanuts aan te maken, te bewerken en te verwijderen. Dit geeft dus de mogelijkheden die minimaal nodig zijn om, in combinatie met een eenvoudige voorbeeld site, een demonstratie te geven van wat het systeem reeds allemaal kan.

5.4. Testen

Voor het draaien van regressietesten is in de ontwikkelomgeving (zie 5.5) al het benodigde geïmplementeerd. Echter, de spaarzame tests die geschreven waren zijn gesneuveld bij de overgang op het nieuwe database-abstractie systeem (zie 5.2.2.2). Nieuwe testen zijn niet geschreven omdat hiervoor geen tijd is genomen in de haast zoveel mogelijk van het systeem werkend te krijgen.

Het belang van de unit tests wordt zeker onderkend, en het verdient met grote nadruk de aanbeveling de missende tests zo spoedig mogelijk te schrijven, om zo bij latere veranderingen een goede controle te hebben van de werking van het systeem (zie ook sectie 7.3).

5.5. Ontwikkelomgeving

Naast de implementatie is er ook een ontwikkelomgeving ingericht om te werken aan het systeem. Deze ontwikkelomgeving is bruikbaar voor verdere ontwikkeling aan het systeem. Het kan uiteraard ook heel goed worden gebruikt voor het schrijven van toekomstige webapplicaties die gebruik maken van dit systeem.

De omgeving is ingericht met het oog op het vereenvoudigen van het programmeren, testen en deployen van het systeem.

5.5.1. Eclipse

De ontwikkeling is geheel gedaan vanuit Eclipse[4] met de PHP plugins voor Eclipse. De verschillende modules zijn stuk voor stuk aparte projecten binnen Eclipse.

5.5.2. Phing

Om gemakkelijk van een module een Amy-file te construeren of de module te deployen (aan de instantie toe voegen) wordt Phing[5] gebruikt. Er is voor Phing een build-file geschreven die deze taken kan doen, maar die ook de API-documentatie kan genereren en de unit-tests kan uitvoeren. Het aanroepen van Phing is eenvoudig in Eclipse te integreren.

Helaas is Phing nog in ontwikkeling en niet geheel bugvrij. Een van de 'features' van de stabiele versie op het moment van schrijven, is dat er bij het genereren van een zip-bestand onder Windows het volledige pad word opgenomen in het bestand. Om Phing zo ver te krijgen dit niet te doen, moesten twee bestanden uit de ontwikkelversie van Phing worden gekopiëerd over de stabiele versie.

5.5.3. Subversion

Voor het versiebeheer van de implementatie en de documentatie is Subversion[6] gebruikt.

De verschillende modules van het systeem hebben elk hun eigen plek binnen de Subversion repository. Verschillende versies kunnen bijgehouden door middel van tags en branches, aan de hand van een handleiding die tijdens het project geschreven is.

5.6. Reflectie

Zoals eerder vermeldt, is in de analyse niets van het bovenstaande naar voren gekomen. Een groot deel van het systeem zoals hier beschreven is ontworpen tijdens de systeemontwerp fase, maar een aantal dingen ook niet. Om in termen van het plan van aanpak te spreken, we voerden herstructurering en optimalisatie uit zonder al de documentatie continu aan te passen, en vonden geregeld meerdere 'solution' klassen nodig. Omdat het ODD gegenereerd wordt, werden deze klassen wel automatisch gedocumenteerd. De linksligging van de documentatie zorgde voor een goed doorlopend implementatie traject zonder onderbrekingen, wat zeker heeft bijgedragen aan het afkomen van de hoeveelheid implementatie die is gedaan.

De implementatie van de peanuts duurde een stuk langer dan verwacht, en dit is vooral de reden dat laat aan het verslag begonnen is.

De implementatie vondt ook al plaats tijden het ontwerp, zoals beschreven in sectie 4.4. Zoals gezegd was dit vanwege de taakverdeling en de voorimplementaties.

Hoewel reeds in een vroeg stadium is begonnen met het schrijven van code, is uiteindelijk niet alles geïmplementeerd wat gewenst is. We zijn van mening dat we de tijd die nodig is voor het project verkeerd hebben ingeschat, en dat de omvang van de te schrijven software uiteindelijk eigenlijk te groot is voor een project met een looptijd van één kwartaal. Daarom is ook de acceptatie eis AC4 (nog) niet vervuld: de developer handleiding zal nog even op zich moeten laten wachten. Wel is er al een (deel van een)

[4] Een Integrated Development Environment. Zie het plan van aanpak en `http://www.eclipse.org`.
[5] Een build tool vergelijkbaar met Ant, geschreven in PHP. Zie `http://www.phing.info`.
[6] Zie het plan van aanpak en `http://subversion.tigris.org`.

demo gemaakt (aan de hand van optionele voorwaarde OC1D), welke nodig is voor de presentatie, maar ook voor de acceptatie (goedkeuring) zoals die zal gaan plaatsvinden.

Ondanks dat niet alles is geïmplementeerd zijn wij wel trots op het resultaat, en zijn wij van mening dat dit een solide en innovatieve basis vormt om een goed CMS mee te bouwen.

6. Afronding

6.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan de activiteiten gedaan omtrent de afronding van het project. Dit houdt onder andere de volgende dingen in:

- De acceptatie van het systeem en onze reflectie erop.
- De ingebruiksname van het systeem en onze reflectie erop.

6.2. Acceptatie

De acceptatie fase is origineel beschreven als een proces waarin de opdrachtgever het product controleert aan de hand van de aan het begin van het project vastgestelde voorwaarden (zie appendix B, sectie 3.9).

Dit proces heeft nog niet plaatsgevonden. Zoals beschreven in hoofdstuk 5 is het product nog niet af genoeg om als zodoende afgeleverd te worden, en dus is het product nog niet geaccepteerd.

Een eerdere onofficiele acceptatie zal wel plaatsvinden, aan de hand van het huidige systeem en de eerder genoemde demonstratie. Dit zal ook een doorbespreking van de mogelijkheden en doorgangsplannen omvatten, aangezien de opdrachtgever graag zijn product af ziet. Omdat één van de ontwikkelaars bij het bedrijf werkzaam is, zal verder aan het product gewerkt worden.

6.3. Ingebruiksname

Gelijk aan de acceptatie is de ingebruiksname nog geen punt: het systeem is nog niet voldoende af.

Het originele idee was om het project aan een opleveringsproject te koppelen, waarbij het nieuwe CMS gebruikt zou worden als de back-end van het op te leveren project (zie appendix B, sectie 3.10). Dit is echter nooit van de grond gekomen, vooral vanwege de drukte bij de andere medewerkers van het bedrijf. Nu is duidelijk dat dit ook geen reële optie geweest zou zijn, de druk die daardoor gecreëerd zou zijn had funest geweest voor de propere documentatie van het project en de innovatieve en mooie implementatie van het systeem.

De toekomstige ingebruiksname van het product zal bestaan uit het opslaan en basis versioneren van het systeem in de versiebeheer. Vanuit hier kan het systeem dan gebruikt worden voor welk opleveringsproject dan ook, vanwege de ondersteuning voor modules.

7. Conclusie

7.1. Inleiding

In dit hoofdstuk wordt aandacht besteed aan het project in het algemeen, en het verslag afgesloten. Dit houdt onder andere de volgende dingen in:

- Een samenvatting van de resultaten van het project en de reactie erop.
- De aanbevelingen voor de toekomst van het systeem.
- Onze reflectie op het gehele project.

7.2. Resultaten

De resultaten van dit project zijn prototypes van twee systemen, en een demo webapplicatie die met die twee systemen is gebouwd. Daarnaast is er een herbruikbare ontwikkelomgeving, en interne documentatie ten behoeve van het gebruik van standaarden gemaakt.

Amy is een framework dat innovatieve en beproefde technologiën met elkaar combineert. Wij persoonlijk vinden het systeem zo gaaf dat we graag een open source of in ieder geval een "Jullie mogen het ook gebruiken" licentie zouden willen zien voor Amy, zodat we het zelf ook kunnen gebruiken.

Kif is een implementatie gebouwd op Amy die van alle nieuwe technologiën dankbaar gebruikt maakt. De functionaliteit is vergelijkbaar met het oude systeem, maar dan correct georganiseerd en geïmplementeerd. Dit is een propere herschrijving, en dat geeft veel voldoening. Het voegt een stukje toe aan de nette code in de wereld.

Na deze lichte ophemeling is het wel verstandig de feiten ook onder ogen te nemen. Beide systemen zijn nog niet voldoende af: ze werken wel, maar ze hebben nog aardig wat ruwe kantjes.

Desondanks zijn wij van mening dat de implementatie erg geslaagd is. Wat echter nog beter is, zijn de positieve reacties die we tot nu toe gekregen hebben. De acceptatie is per definitie gefaald (aan niet alle voorwaarden is voldaan, aangezien de handleiding nog niet is geschreven) maar over het verloop van het project nam het systeem toch al een meer innovatieve vorm aan, iets wat bij 7U toch wel past.

7.3. Aanbevelingen

Er zijn een groot aantal aanbevelingen te doen voor deze systemen, simpelweg omdat ze nog niet af zijn. Een overzicht:

- Het schrijven van de module ontwikkel handleiding (AC4).
- Het verder werken aan het systeem:
  - Het implementeren van dingen die missen die nodig of kritiek zijn, zoals de rechten en relaties.
  - Het maken van de echt benodigde modules, zoals bijvoorbeeld pagina en news objecten.
  - Het implementeren van niet noodzakelijke maar wel fijne dingen, zoals ondersteuning voor het sorteren van peanuts.
- Het productieklaar maken van het systeem. Dit houdt in dat de regressietesten geïmplementeerd en gebruikt moeten gaan worden.

Aangezien één van de ontwikkelaars bij 7U werkt, zijn deze aanbevelingen waarschijnlijk niet lang meer van toepassing...

7.4. Reflectie

7.4.1. Project

De grootste les die uit dit project getrokken kan worden is dat een systeem dat zo ambitieus is, niet in het tijdsbestek van een kwartaal compleet gerealiseerd kan worden. Wat *wel* interessant is, is dat je in een kwartaal blijkbaar erg ver kunt komen, *inclusief* volledige documentatie.

Wij vonden het vooral leuk dat ons veel vrijheid werd geboden tot innovatie en onderzoek, in plaats van een gestressde rommelimplementatie onder zware tijdsdruk. Daarnaast is het natuurlijk geweldig dat het ook daadwerkelijk zo goed werkt. We hadden veel mogelijkheden, en denken de beste toch wel uitgekozen te hebben. Dit heeft 7U uiteindelijk een vooruitstrevend en daarom competitief CMS opgeleverd.

7.4.2. Organisatie

De organisatie van het project was minder geweldig. Onze planningen waren niet goed en we kampten continue met een achterstand. Ook konden we ons niet goed aan het plan van aanpak houden. Het is echter uiteindelijk wel goedgekomen, dus het kan niet *zo* slecht geweest zijn. Al met al is dit een goede en vooral leerzame ervaring geweest op het gebied van planning en organisatie.

7.4.3. Methode

Wat ook minder goed uitpakte was de keuze voor ontwikkelmethode. Zoals vermeldt was deze niet bruikbaar voor het type systeem wat wij gingen ontwikkelen, en dat heeft ons vooral in het begin parten gespeeld. We wisten niet goed wat te doen, liepen vertraging op en twijfelden of onze aanpak wel goed en afdoende was. Gelukkig is dit (voornamelijk) goedgekomen door de informele sfeer die in het bedrijf heerst en de extra communicatie die er daarom in het begin is geweest.

7.4.4. Communicatie

De communicatie tijdens het project was niet sterk aanwezig. Tussen Berend en Thomas ging het over het algemeen wel goed, en hetzelfde tussen hun en de TU begeleider, maar met de 7U begeleider en het management was de communicatie maar weinig. Uiteindelijk bleek het niet zo heel veel uit te maken omdat wij zo veel vrijheid hadden om te maken wat we wilden, maar over het algemeen genomen lijkt ons een hogere mate van bemoeiing wel verstandig.

7.4.5. Omgeving

De werkomgeving bij 7U is ronduit goed. Je hebt je eigen volwaardige werkplek die geregeld wordt schoongemaakt. De lunch is gratis, en tijdens de lunch wordt er heftig gesocialiseerd (men speelt er schietspelletjes). Er staat altijd muziek aan tijdens het werk. De software die je nodig hebt is beschikbaar of kan beschikbaar worden gemaakt, hetgeen allemaal bijna volledig geautomatiseerd gaat. Als er een probleem is heeft de systeemadministrator er geen moeite mee je even te helpen. En tot slot zijn er geen noemenswaardige spanningen op de werkvloer, behalve misschien dat Ron altijd wint met schieten.

Dit is een omgeving die je als Bachelor Project student wilt hebben.

7.4.6. Conclusie

Al met al is het project naar onze tevredenheid afgerond. We hebben veel geleerd, veel gave dingen gedaan, en een leuke tijd gehad.

Ondanks dat niet alles is geïmplementeerd zijn wij trots op het resultaat, en zijn wij van mening dat dit een solide en innovatieve basis vormt waarmee een competitief CMS gebouwd kan worden.

Bibliografie

[1] Bernd Bruegge & Allen H. Dutoit. *Object-Oriented Software Engineering: Conquering Complex and Changing Systems.* Prentice Hall, 1st edition, October 1999.

Deel II.

Appendices

Appendix A: Opdrachtomschrijving

CMS 3
Opdracht Omschrijving

7U Digital Handmade Originals

Rotterdam, 30 maart 2007

Aanloop

7U is bedrijf uit Rotterdam dat zich bezig houdt met het ontwerpen en realiseren van websites en webapplicaties. Enige jaren geleden werd het steeds belangrijker om klanten de mogelijkheid te bieden om zelf delen van hun website aan te passen met behulp van een Content Management Systeem.

De keuze diende toen gemaakt te worden tussen het kopen van een bestaand systeem of het ontwikkelen van een eigen systeem. Er is toen voor het laatste gekozen aangezien dat meer vrijheid betekende in de ontwerpmogelijkheden en we op die manier de klanten eenvoudiger maatoplossingen konden aanbieden.

Er is toen begonnen met de ontwikkeling van het 7U CMS maar door de tijdsdruk, personeelsveranderingen en nog vele andere redenen heeft deze ontwikkeling een sterk pragmatisch karakter gehouden. Eigenlijk was het CMS meer een verzameling code, die telkens opnieuw werd aangepast en uitgebreid om aan de specifieke wensen van de klant te voldoen. Hierdoor is er nooit een definitieve versie van een CMS onstaan, noch bestond de noodzaak om documentatie van het CMS aan te leggen.

In 2006 zijn de eerste stappen gezet om een basis CMS samen te stellen welke zonder wijzigingen aan de backend ingezet kan worden voor relatief eenvoudige websites. Een van de eisen hiervoor is dat de grafische interface van het CMS gebruiksvriendelijker en praktischer moet worden. Een interface ontwerper heeft het CMS met alle gewenste basisfunctionaliteiten in schermvoorbeelden uitgewerkt en gedocumenteerd. Er is inmiddels begonnen met het omzetten van deze ontwerpen naar xhtml en css zodat ze aan het CMS gekoppeld kunnen worden. Dit is voornamelijk een grafische 'makeover' van de backend van het bestaande CMS. Op technisch vlak verandert er niet veel.

De ervaring van de afgelopen jaren heeft ons wel een duidelijk beeld gegeven hoe wij het 7U CMS op technisch vlak willen hebben. In grote lijnen hebben we deze functionaliteit ook gerealiseerd en hebben we nu een flexibele basis waarmee we snel een CMS op maat kunnen aanbieden. De onderliggende code / architectuur is echter sterk vervuild door alle 'groeistuipen' van het CMS.

Opdracht

De opdracht voor het project zou dan ook het beste als volgt omschreven kunnen worden:

1. Analyseer de wensen en ideeën van 7U aangaande het CMS systeem.
2. Ontwerp en ontwikkel een CMS systeem op basis van de wensen van 7U dat modulair uit te breiden is, maar waarvan de basis stabiel blijft.
3. Documenteer de code goed en zorg voor uitleg hoe het CMS uit te breiden is.
4. Tot slot zou 7U het systeem graag in de praktijk willen testen door het aan een concreet project te koppelen zodat het voorgelegd kan worden een concreet team van eindgebruikers.

Naar ons idee werkt dit het beste wanneer de studenten beide minimaal vier werkdagen per week op kantoor in Rotterdam aan het systeem werken. De projectleider zal Elja van Tol zijn, de ontwerper en

bouwer van het huidige systeem. Ook de projectleiders en de ontwerper van de interface zullen bij het project betrokken worden.

Appendix B: Plan van aanpak

CMS 3
Plan van Aanpak

Berend Wouda Thomas de Ruiter

14 Juni 2007

Voorwoord

Dit document is het *Plan van Aanpak* voor het *TU Delft 'IN3700 BSc-Project'* van *Thomas de Ruiter* en *Berend Wouda*.

In dit document wordt de aanpak van de specifieke uitvoering van dit project uiteengezet. Naast een introductie van het bedrijf en het project worden er een propere afbakening en beschrijving van de taken en verwachtingen geformaliseerd, en worden de ontwikkelmethoden en -gereedschappen gegeven.

Dit document dient als overeenkomst tussen de opdrachtgever en de ontwikkelaars over de inhoud en uitvoering van het project.

Versie

Het is mogelijk dat dit document wordt aangepast, onder de voorwaarde dat beide partijen (opdrachtgever en ontwikkelaars) er mee instemmen. Elke afwijking zal hieronder worden gedocumenteerd.

13 April 2007

Eerste versie.

25 April 2007

Na overweging is de opdrachtgever met de ontwikkelaars overeengekomen dat de te ondersteunen versies van PHP wordt verminderd tot alleen (de huidige versie van) PHP 5.

Dit betekent dat het acceptatiecriterium: "Het prototype is compatible met PHP versies 4.3 tot en met 5.2.1." is veranderd naar "Het prototype is compatible met PHP versie 5.2.1."

De acceptatiecriteria zijn nu ook genummerd.

27 April 2007

Op aanrading van Peter van Nieuwenhuizen zijn de volgende aanpassingen gemaakt:

- Diverse tekstuele verbeteringen.
- Duidelijkere introductie van afkortingen.
- Niet bekende inleverdata voor documenten weggelaten.
- Verduidelijking en prioritisatie van optionele voorwaarden.
- Geen aangegeven hoofdfase meer.

14 Juni 2007

De definitieve inlever- en peresentatiedata zijn gegeven.

Inhoudsopgave

Inleiding

Opdrachtgever

7U Digital Handmade Originals is een redelijk klein jong bedrijf uit Rotterdam dat zich bezig houdt met het ontwerpen en realiseren van multimedia producten, voornamelijk websites en webapplicaties. Er werken negen personen. Daarnaast zijn er vaak ook één of twee stagiairs van de studie Communication & Multimedia Design (van de Willem de Kooning Academie) te vinden. 7U heeft al menig multinational tot haar klanten mogen rekenen.

Aanleiding

Enige jaren geleden werd het voor 7U steeds belangrijker om klanten de mogelijkheid te bieden om zelf delen van hun website aan te passen met behulp van een Content Management Systeem (CMS). Er is toen besloten een eigen systeem te ontwikkelen, aangezien op die manier de klanten eenvoudiger maatoplossingen kan worden aangeboden.

Er is toen begonnen met de ontwikkeling van het 7U CMS maar door de tijdsdruk, personeelsveranderingen en nog vele andere redenen heeft deze ontwikkeling een sterk pragmatisch karakter gehouden. Eigenlijk was het CMS meer een verzameling code, die telkens opnieuw werd aangepast en uitgebreid om aan de specifieke wensen van de klant te voldoen. Hierdoor is er nooit een definitieve versie van een CMS ontstaan, noch bestond de noodzaak om documentatie van het CMS aan te leggen.

Er is daarom behoefte aan een van de grond af aan opnieuw ontworpen en gedocumenteerd systeem, gebaseerd op het bestaande systeem.

1 Studie

1.1 Bachelor Project

Het project wordt gedaan in de vorm van een Bachelor Project voor de studie Technische Informatica van de TU Delft. Er zal daarom aan een aantal standaarden moeten worden voldaan.

- Het project dient op een gestructureerde en gedocumenteerde wijze te geschieden.
- Gedurende het project moet duidelijke en beknopte documentatie worden geleverd, aan de hand waarvan het proces kan worden gecontroleerd.
- Het project wordt afgesloten met het leveren van een verslag, en het houden van een presentatie.

1.2 Verslag

Aan het einde van het project moet een verslag van het project worden ingeleverd bij de (TU Delft) begeleider. Dit verslag zal het verloop van het project beschrijven, en er op reflecteren.

De documenten (*deliverables*) van het project zullen worden bijgesloten als bijlages, en korte samenvattingen van deze zullen in het verslag worden ingesloten.

De inleverdatum van het verslag is **9 Juli 2007**.

1.3 Presentatie

Na het inleveren van het verslag moet een presentatie gehouden worden. Deze moet ongeveer 20 minuten duren en zal een overzicht geven van het project. Daarnaast zal er een aantal representatieve punten worden uitgelicht, en zal er een demo van het prototype worden gegeven.

Tijdens deze presentatie zullen alle begeleiders aanwezig zijn.

De presentatie vindt plaats op **12 Juli 2007**.

2 Project

2.1 Doelstelling

7U heeft behoefte aan een CMS dat van de grond af aan opnieuw is ontworpen en geïmplementeerd om zo een solide basis te hebben die voor elke klant gebruikt kan worden. Om 7U in staat te stellen eenvoudig uitbreidingen te maken dient er documentatie te zijn aan de hand waarvan uitbreidingen ontwikkeld kunnen worden.

2.2 Opdrachtformulering

"Ontwerp een CMS basis naar de wensen en ideeën van 7U, implementeer een prototype, en schrijf documentatie aan de hand waarvan uitbreidingen kunnen worden gemaakt."

2.3 Voorwaarden

Het eindresultaat van het project moet voldoen aan de voorwaarden in tabel 2.1 (de algemene acceptatievoorwaarden).

Voorwaarde	Definitie
• Het prototype is een basiskern voor modulair opgebouwde eindproducten, en kan met (eindproduct-specifieke) extensies worden uitgebreid.	**AC1**
• Het prototype is compatible met PHP versie 5.2.1.	**AC2**
• Het prototype is compatible met MySQL versies 4.1.22 tot en met 5.0.37.	**AC3**
• De documentatie ondersteunt het ontwikkelen van uitbreidingsmodules (extensies).	**AC4**

Tabel 2.1: Acceptatievoorwaarden

Na oplevering van het eindresultaat, is er de optie te voldoen aan een aantal van de voorwaarden in tabel 2.2. De hoger geplaatste voorwaarden hebben een hogere prioriteit.

Voorwaarde	Definitie
• De volgende modules moeten worden geïmplementeerd:	**OC1**
• Paginas	**OC1A**
• Mappen	**OC1B**
• Nieuws	**OC1C**
• Standaard Beheer	**OC1D**
• Klantvriendelijk Beheer	**OC1E**

Tabel 2.2: Optionele voorwaarden

2.4 Producten

De volgende producten (*deliverables*) zullen worden opgeleverd tijdens het project[1]. Dit wordt in hoofdstuk 3 in detail besproken.

[1] Dit is niet gerelateerd aan de documentatie voor het studie gerelateerde gedeelte.

- Planning
- Plan van Aanpak
- Requirements Analysis Document (Eisenanalyse)
- System Design Document (Systeemontwerp)
- Object Design Document (Objectontwerp)
- Prototype
- Documentatie

De eisenanalyse, het systeemontwerp, het objectontwerp, het prototype (het code commentaar) en de interne documentatie (handleiding en API) zullen in het Engels worden opgeleverd vanwege mogelijke toekomstige internationalisering.

Voor elk systeem (basis en modulen) zullen eigen producten worden opgeleverd.

2.5 Contactpersonen

De volgende personen spelen een rol in dit project:

Frank Tibben *7U, Manager*
Telefoon: 010-4776178
E-mail: frank@7u.nl

Elja van Tol *7U, Developer, 7U Begeleider*
Telefoon: 010-4776178
E-mail: elja@7u.nl

Peter van Nieuwenhuizen *TU Delft, TU Begeleider*
Telefoon: 015-2788036
E-mail: p.r.vannieuwenhuizen@ewi.tudelft.nl

Berend Wouda *TU Delft, Student*
Studienummer: 1100726
Telefoon: 06-26959403
E-mail: kirk@kirkwarez.com

Thomas de Ruiter *TU Delft, Student*
Studienummer: 1100386
Telefoon: 06-20746188
E-mail: thomas@de-ruiter.cx

3 Aanpak

3.1 Methodologie

Het project zal verlopen via een adaptie van de redelijk klassieke software engineering methode, gebruikmakend van op UML gebaseerde ontwerp methodes. Deze bestaat uit een aantal fasen:

- Het schrijven van een plan van aanpak
- Het maken van een planning
- Het maken van een eisenanalyse
- Het ontwerpen van het systeem
- Het implementeren van het systeem
- Het schrijven van documentatie
- De acceptatie van het systeem
- De ingebruiksname van het systeem

De fasen vinden plaats in deze volgorde, hoewel documentatie tijdens alle fasen geschreven zal worden. Elke fase (behalve de laatste twee) wordt afgesloten met de oplevering van een 'product' (*deliverable*), wat een document of een prototype kan zijn. De fasen worden in de rest van dit hoofdstuk beschreven.

Voor elk te implementeren systeem wordt deze methodologie opnieuw doorlopen. Het basissysteem zoals beschreven in paragraaf 2.3 wordt eerst geïmplementeerd, daarna worden (afhankelijk van de hoeveelheid beschikbare tijd) de modulen geïmplementeerd.

3.2 Plan van Aanpak

Als eerste zal een 'Plan van Aanpak' worden opgesteld. Middels dit document wordt de aanpak van de specifieke uitvoering van dit project uiteengezet. Naast een introductie van het bedrijf en het project worden er een propere afbakening en beschrijving van de taken en verwachtingen geformaliseerd, tevens worden de ontwikkelmethoden en -gereedschappen gegeven.

Het 'Plan van Aanpak', het document dat u nu leest, dient als overeenkomst tussen de opdrachtgever en de ontwikkelaars over de inhoud en uitvoering van het project.

3.3 Planning

Er zal een planning voor het verloop van het project worden gemaakt. Deze planning bevat de geplande uren voor de verschillende onderdelen, de totale duur van het project en de taakverdeling.

3.4 Analyse

3.4.1 Proces

De eerste grote fase van het project zal de identificatie en analyse van de eisen zijn. Deze fase is cyclisch van aard, en bestaat uit de volgende subfasen:

- Identificatie van de eisen aan het systeem
  - Analyse van het huidige systeem
    - Analyse van de huidige basis
    - Analyse van een aantal reeds opgeleverde producten
  - Interviews met de opdrachtgever
- Analyse van de eisen
  - Uitwerking/documentering van de ingewonnen informatie
  - Voorlegging voor goedkeuring aan de opdrachtgever

Tijdens de eisenidentificatie wordt eerst het huidige systeem (en de selectie aan producten) geanalyseerd op de mogelijkheden en (overduidelijke) tekortkomingen. Vervolgens wordt er van de opdrachtgever informatie ingewonnen over de verdere tekortkomingen van het huidige systeem, en welke (nieuwe) verwachtingen er aan het systeem zijn.

Hierna wordt een document opgesteld, het zogenaamde RAD (Requirements Analysis Document) welk de (door analyse) uitgevonden eisen en het gerelateerde voorontwerp van het systeem bevat. Dit wordt aangeleverd bij de opdrachtgever, die het goed- of afkeurt. Bij goedkeuring treedt de volgende fase (ontwerp) in werking, bij afkeuring wordt de huidige fase herhaald en worden de eisen en het daaraan verbonden ontwerp op een incrementele manier aangepast, totdat het RAD goedgekeurd wordt door de opdrachtgever.

Na goedkeuring van het RAD wordt dit document in principe niet meer aangepast, en dient het als overeenkomst tussen de opdrachtgever en de ontwikkelaar waarin is aangegeven hoe het nieuwe systeem er uit gaat zien.

3.4.2 Identificatie

De identificatie van de eisen aan het systeem gebeurt hoofdzakelijk door middel van (informeel van aard zijnde) interviews. Tevens staat de begeleider altijd stand-by voor het beantwoorden van directe vragen.

3.4.3 Analyse

De gevonden resultaten worden geanalyseerd door middel van het gebruik van UML-2 notaties om op hoog niveau systeem aspecten weer te geven. De te maken systemen kunnen volledig geanalyseerd worden met behulp van UML-2, het basissysteem heeft alleen geen gebruikersinterface.

De analyse bestaat uit het identificeren en documenteren van de volgende modellen:

- Actoren: De entiteiten die interactie gaan hebben met het systeem.
- Scenario's: De informele, concrete beschrijvingen van de acties die geinterviewde gebruikers zich voorstellen uit te voeren op het systeem.
- Gebruikscasussen: De formele, abstracte acties die actoren willen kunnen uitvoeren op het systeem, verkregen door abstrahering van de scenarios.
- Initiële analyse modellen: Modellen van de op een hoog niveau vereiste onderdelen van het systeem, die over het algemeen een afspiegeling zijn van de werkelijkheid.
- Dynamische modellen: Modellen van de mogelijke veranderingen in de staat van het systeem gedurende operatie, en wat er gebeurt tijdens de uitvoeringen van de gebruikerscasussen.
- Gebruikersinterfaces: Schetsen van de bedoelde gebruikersinteractieschermen, en de boom van navigatiepaden van het beginscherm naar elk bereikbaar scherm.

Elk model heeft een corresponderende UML beschrijving en/of diagram type, welk gebruikt zal worden voor een effectieve weergave.

Tijdens de analyse worden ook de geïdentificeerde eisen ingedeeld in de volgende drie categoriën:

- Functionele eisen: Eisen die aangeven wat een actor met het systeem moet kunnen doen.
- Niet-functionele eisen: Eisen die de voorwaarden van het systeem geven buiten wat het moet kunnen doen.
- Pseudo eisen: Eisen die de voorwaarden van het project geven met betrekking tot de ontwikkeling van het systeem.

3.5 Systeemontwerp

3.5.1 Proces

De fase volgend op de analyse van de eisen is het algemene systeemontwerp. Deze fase zal bestaan uit een aantal activiteiten:

- Het stellen van ontwerpdoelen
- Decompositie van het systeem in subsystemen
- Het maken van ontwerpbeslissingen

Hierna wordt een document opgesteld, het SDD (System Design Document), welke het globale ontwerp van het systeem bevat. Dit wordt aangeleverd bij de opdrachtgever, die het goed- of afkeurt. Bij goedkeuring treedt de volgende fase (implementatie) in werking, bij afkeuring wordt de betreffende activiteit herhaald en de onderdelen aangepast, totdat het SDD goedgekeurd wordt door de opdrachtgever.

Na het doorlopen van deze fase is het niet de bedoeling dat er nog aanpassingen aan het ontwerp worden gemaakt.

3.5.2 Ontwerpdoelen

Een lijst van ontwerpdoelen wordt opgesteld aan de hand van de niet-functionele eisen.

3.5.3 Decompositie

Een decompositie van het systeem op hoog niveau wordt geabstraheerd uit de vereisten. Alle relaties met aangrensende subsystemen worden geïdentificeerd, en de diensten van elk van deze subsystemen worden geïdentificeerd en gedocumenteerd. Deze documentatie bevat de corresponderende UML diagrammen.

3.5.4 Ontwerpbeslissingen

Vanuit de ontwerpdoelen en de decompositie worden beslissingen genomen aangaande de implementatie van de subsystemen. Deze beslissingen behelsen in ieder geval:

- Hardware/software: De toewijzing van subsystemen aan verschillende hardware en software componenten, en de organisatie daaromtrent.
- Persistente data beheer: Het exacte gebruik van data opslag systemen, en de interne persistente data organisatie.
- Toegangsbeheer en veiligheid: De toegang tot het systeem uitgedrukt in de gebruikers ervan, en de aanpak voor de veiligheid van het systeem.
- Globale werking: De werking van het systeem betreffende de interactie en synchronisatie tussen subsystemen, en de specifieke verwerking van opdrachten aan het systeem binnen het systeem tussen de subsystemen.
- Grensvoorwaarden: De initialisatie en deïnitialisatie van het systeem, en de manier waarop fouten worden afgehandeld.

3.6 Objectontwerp

3.6.1 Proces

Deze fase volgt na het systeemontwerp. Deze fase bestaat uit de volgende activiteiten:

- Dienst specificatie
- Component selectie
- Herstructurering
- Optimalisering

De activiteiten verlopen parallel en kunnen meerdere keren worden herhaald voor verschillende toevoegingen en wijzigingen gedurende het objectontwerp.

Hierna wordt een document opgesteld, het ODD (Object Design Document), welke het gedetailleerde ontwerp van het systeem bevat. Het deel van dit document betreffende de interface zal later automatisch worden gegenereerd aan de hand van de code.

Indien gewenst zal het ODD ter goedkeuring aan de opdrachtgever worden voorgelegd.

Na het doorlopen van deze fase is het niet de bedoeling dat er nog aanpassingen aan het ontwerp worden gemaakt.

3.6.2 Dienst specificatie

Gedurende deze activiteit zullen alle objecten met hun interfaces en de onderlinge relaties worden gedefiniëerd en gedocumenteerd. Deze definitie bestaat uit het volgende:

- Specificatie van de (missende) attributen, operaties en objecten. De bijbehorende documentatie bevat de corresponderende UML diagrammen.
- Specificatie van typen en signaturen. De bijbehorende documentatie bevat de corresponderende UML diagrammen.
- Voorwaarden voor de interface elementen (zoals precondities, postcondities en invarianten).
- Specificatie van uitzonderingen die binnen het systeem kunnen optreden.

De objecten zullen worden gegroepeerd in packages naar hun subsysteem en decompositie.

3.6.3 Component selectie

Externe componenten die geselecteerd zijn tijdens het systeemontwerp zullen worden geïnspecteerd en indien noodzakelijk bewerkt of ingekapseld. Dit kan extra objecten opleveren, die een service specificatie nodig hebben.

3.6.4 Herstructurering

Na de (initiële) specificatie en selectie zullen relaties tussen objecten onderling worden omgezet in objecteigenschappen. Dit kan extra objecten opleveren voor de één op veel relaties, die weer een service specificatie nodig hebben. Overerving zal worden gebruikt waar dit geschikt is om hergebruik van objecten en typen te bevorderen. Overerving en contracten zullen worden gebruikt om interfaces te generaliseren en details van de implementatie te verbergen.

3.6.5 Optimalisering

Het objectontwerp zal worden aangepast zodat het zal voldoen aan de ontwerpdoelen.

3.7 Implementatie

3.7.1 Proces

Na het ontwerpen van het systeem zal de implementatiefase beginnen. In deze fase zullen alle ontwerpen worden geïmplementeerd. Deze fase bestaat uit twee subfasen:

- Programmeren
- Testen

Deze fasen zullen enigszins parallel lopen, maar zijn vooral cyclisch. Eerst zal een deel worden geïmplementeerd, vervolgens zal de code worden getest. Dit wordt herhaald tot het systeem werkt als vereist.

Deze fase zal het werkende systeem zelf opleveren, dat dan klaar is voor ingebruikname. Idealiter zouden geen verdere updates aan de code nodig moeten zijn.

3.7.2 Programmeren

Het programmeren zal geschieden in PHP. Van de code zal API documentatie worden gegenereerd.

3.7.3 Testen

Er zal op twee manieren worden getest:

- Unit testen: Het systematisch testen van de integriteit van elk object.
- Integratie testen: Het testen van de bruikbaarheid van het hele systeem. Dit zal worden gedaan door alle gebruikerscasussen uit te voeren en de resultaten te inspecteren. Dit test de werking van het hele systeem.

3.8 Documentatie

Gedurende en na de implementatie zal documentatie gemaakt worden. Deze fase zal een handleiding voor ontwikkelaars opleveren. Deze handleiding bevat het ontwerp, een gedetailleerde beschrijving van de implementatie, aanwijzingen betreffende onderhoud van het systeem, en uitleg voor het maken van uitbreidingen.

3.9 Acceptatie

Na het afronden van de documentatie zal het product worden gepresenteerd aan de opdrachtgever. De opdrachtgever zal dan het product beoordelen aan de hand van de aan de start van het project overeengekomen acceptatievoorwaarden, en de later vastgestelde functionele eisen.

3.10 Ingebruikname

Het systeem zal worden gekoppeld aan een concreet project. Na de acceptatie zal de implementatie worden gebruikt om dit project op te leveren. De documentatie zal worden opgeslagen op het interne bedrijfsnetwerk. Onderhoud van het systeem is geen onderdeel van dit project maar zal wel worden ondersteund door specifiek daarvoor geschreven documentatie.

4 Inrichting

4.1 Code

Bij het schrijven van de PHP code zal de 'PHP Coding Standard'[1] worden gehanteerd. Voor dit project zal op een aantal punten van deze standaard worden afgeweken, alle verschillen zullen worden gedocumenteerd.

4.2 Versiebeheer

Een conventie voor de inrichting van de Subversion repository zal worden opgesteld en gedocumenteerd. Deze conventie zal worden gehanteerd gedurende het project. Het wordt aanbevolen om ook na het project deze conventie te blijven gebruiken.

4.3 Documentatie

Documenten zullen worden geschreven in LaTeX. Uiteindelijke versies zullen intern beschikbaar worden gesteld in PDF formaat (en afdrukken ervan). API documentatie zal worden gegenereerd met behulp van phpDocumentor.

4.4 Bijeenkomsten

Van zowel officiele als informele bijeenkomsten zullen notulen worden opgesteld. Deze zullen aan de betrokkenen worden gestuurd ter becommentariëring en (indien van toepassing) goedkeuring.

4.5 Logboek

Gedurende het project zal een logboek worden bijgehouden waarin een overzicht wordt gegeven van wat er gedaan wordt. Ook zullen notulen van bijeenkomsten en notities van afspraken worden bijgehouden. Dit zal zorgdragen dat alle informatie ten alle tijde duidelijk beschikbaar is.

4.6 Faciliteiten

Gedurende het project zal gebruik worden gemaakt van de volgende middelen:

- Linux en Windows, de besturingssystemen waarvoor en waarop ontwikkeld zal worden.
- LaTeX gebundeld in MiKTeX en TeXnicCenter worden gebruikt voor het schrijven van de verschillende documenten.
- phpDocumentor zal worden gebruikt voor het genereren van de API documentatie.
- Subversion, TortoiseSVN, Subclipse en Trac als tools voor het versiebeheer. Trac levert tevens een bugtracking/ticket systeem en een wiki.

[1] `http://www.dagbladet.no/development/phpcodingstandard/`

- Eclipse wordt in combinatie met Eclipse PDT en Eclipse WTP gebruikt als ontwikkelomgeving, het is een IDE die oorspronkelijk ontwikkeld is voor Java ontwikkeling, maar met de genoemde plugins kan het ook uitstekend worden gebruikt voor het ontwikkelen van PHP applicaties.
- PHPUnit[2] zal worden gebruikt voor het testen.
- MySQL en Apache met PHP worden gebruikt als respectievelijk de database- en webserver. Onder Windows zal hiervoor de bundel WAMP[3] worden gebruikt.
- De werkplekken met werkstations en toegang tot servers en het Internet worden door 7U ter beschikking gesteld.

[2] `http://www.phpunit.de/`
[3] `http://www.wampserver.com/`

Bijlagen

De volgende bijlagen zijn bijgevoegd.

- Appendix A bevat de planning voor dit project. Deze is ook te vinden op `https://pavlov/trac/cms3/wiki/Planning`. De hoofdlijnen zijn:
  - Het project loopt van 10 april 2007 tot en met 13 juli 2007.
  - Er wordt op 4 dagen in de week aan het project gewerkt (dinsdag tot en met vrijdag), per dag een periode van 8 uur.
  - Eens in de (gemiddeld) twee weken is er op vrijdagmorgen (ongeveer 9:00 to 10:00) een voortgangsgesprek met de TU begeleider.
  - Op de dinsdag voor een voortgangsgesprek worden de beschikbare/benodigde documenten ingeleverd bij de TU begeleider.
  - Op Hemelvaartsdag (17 mei 2007) wordt er niet aan het project gewerkt.
  - Op 3 juli 2007 heeft Thomas een tentamen, en zal dan niet of minder aanwezig zijn.
- Appendix B bevat de begrippenlijst voor dit project. Dit is de algemene begrippenlijst, en is daarom in het Engels. Deze is ook te vinden op `https://pavlov/trac/cms3/wiki/Glossary`.

Appendix C: Planning

CMS 3
Planning

Berend Wouda Thomas de Ruiter

14 Juni 2007

Voorwoord

Dit document is de *Planning* voor het *TU Delft 'IN3700 BSc-Project'* van *Thomas de Ruiter* en *Berend Wouda*.

In dit document worden de activiteiten en de taken verdeeld en ingepland.

Dit document dient als een stevige richtlijn voor het project. Elke afwijking zal worden gedocumenteerd.

Versie

13 April 2007

Eerste versie.

18 April 2007

Meer implementatie tijd is ingepland.

10 Mei 2007

De planning is aangepast vanwege de uitloop van de Analyse. Tijdens de uitloop zijn wel als systeemontwerp en lichte implementatie (try-outs) gedaan, wat bij die fasen tijd scheelt. De systeemontwerp fase is daarom verkort tot 4 dagen, en de taakverdeling heeft een redelijke vorm aangenomen.

11 Mei 2007

De planning is aangepast vanwege een fout in de aangeving van persoonsinzetting. De percentages stonden niet goed, wat vaak 110% per dag gaf en dus ook meer werk aangaf. Er komt nu niet meer dan 100% inzet per dag voor.

Ook is de planningstabel nu ingevoegd.

14 Juni 2007

De planning is rigoreus aangepast om het verloop tot nu toe aan te geven, en de veranderingen in de planning voor het einde van het project. De presentatiedatum ligt nu vast, de inleverdatum voor het verslag is met een week verlengd en de acceptatie/ingebruiksname zijn minder belangrijk gemaakt.

Inhoudsopgave

Inleiding

Indeling

Deze planning bevat een activiteitenplanning, een mijlpalenkalender en een taakverdeling.

Activiteitenplanning

De activiteitenplanning is gegeven als een 'Gantt-chart', waarop te zien is wat wanneer door wie gedaan wordt. Het laat ook de mogelijkheden tot gelijktijdig verloop van (gedeelten van) fasen zien. Op deze kaart is ook te zien op welke datum een fase in gaat en tot welke datum een fase duurt.

Mijlpalenkalender

De mijlpalenkalender geeft aan op welke data er een beslissing genomen moet worden of een product (*deliverable*) klaar moet zijn. Dit zijn in essentie deadlines.

Taakverdeling

De taakverdeling geeft globaal aan hoe de beschikbare (typen) taken verdeeld worden over de beschikbare ontwikkelaars. De specifieke taakverdeling is te vinden in het Gantt diagram bij de activiteitenplanning.

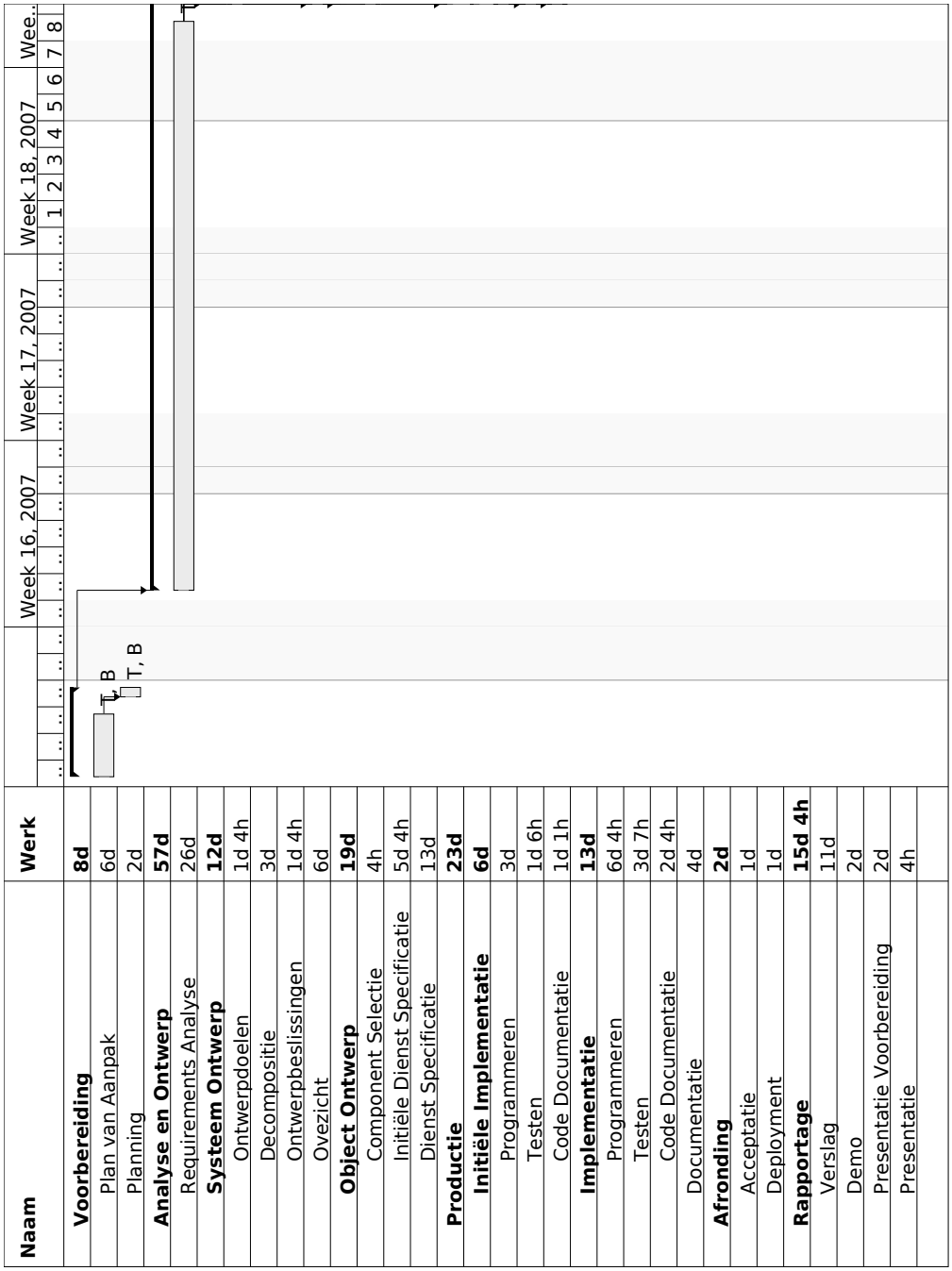
1 Activiteiten

Figuur 1.1 is de planning van dit project. Het Gantt diagram is te zien in figuren 1.2 tot en met 1.4.
Het volgende zijn de hoofdlijnen:

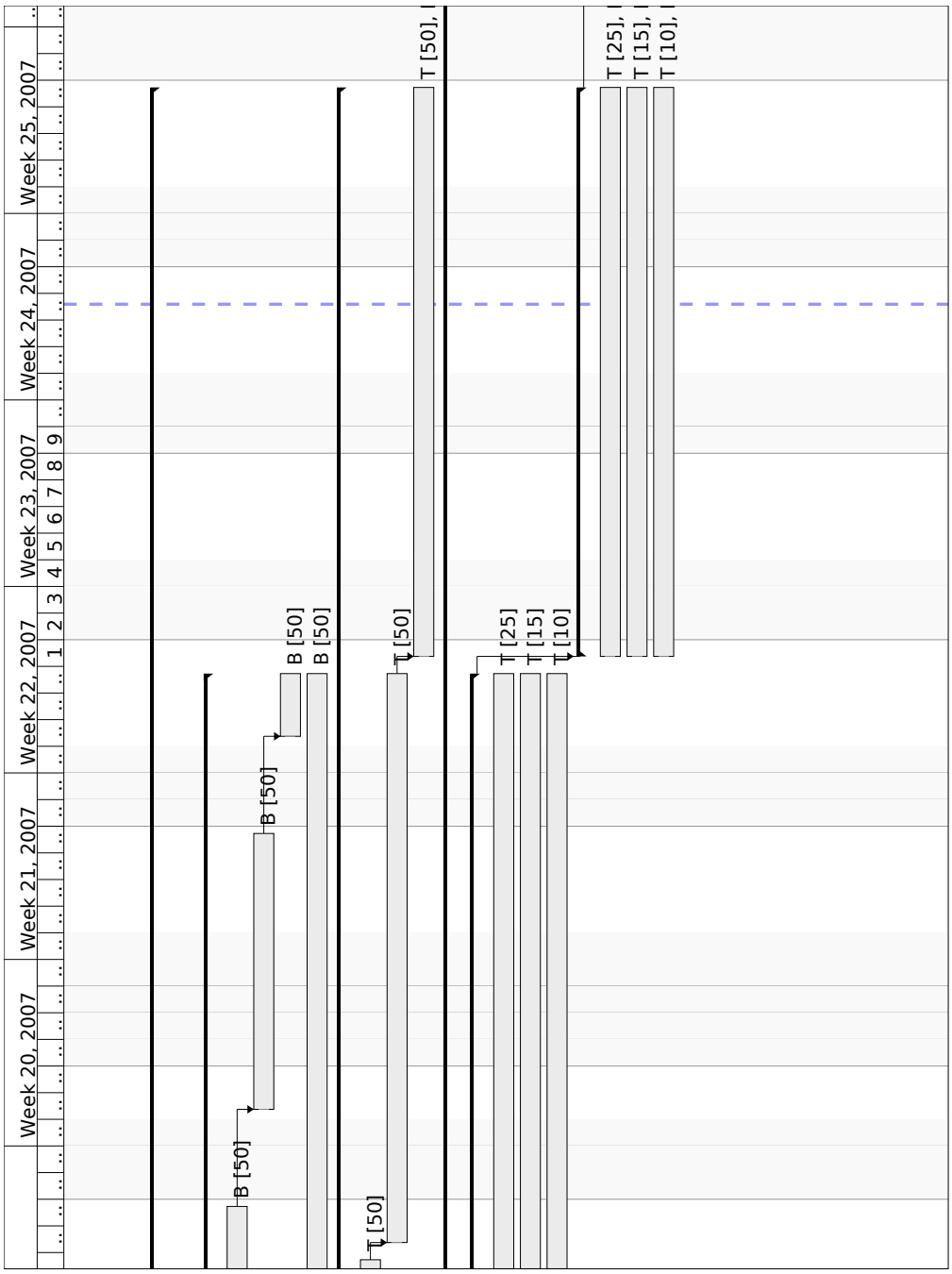
- Het project loopt van 10 april 2007 tot en met 13 juli 2007.
- Er wordt op 4 dagen in de week aan het project gewerkt (dinsdag tot en met vrijdag), per dag een periode van 8 uur.
- Eens in de (gemiddeld) twee weken is er op vrijdagmorgen (ongeveer 9:00 to 10:00) een voortgangsgesprek met de TU begeleider.
- Op de dinsdag voor een voortgangsgesprek worden de beschikbare/benodigde documenten ingeleverd bij de TU begeleider.
- Eens in de week à anderhalve week is een 'update' met de opdrachtgever.
- Op Hemelvaartsdag (17 mei 2007) en de vrijdag erna (18 mei 2007) wordt er niet aan het project gewerkt.
- Op 3 juli 2007 heeft Thomas een tentamen, en zal dan niet aanwezig zijn.

WBS	Naam	Start	Finish	Werk	Tijdsduur	Laks	Kosten	Toegewezen aan
1	**Voorbereiding**	**10 apr**	**13 apr**	**8d**	**4d**		**0**	
1.1	Plan van Aanpak	10 apr	12 apr	6d	3d		0	B, T
1.2	Planning	13 apr	13 apr	2d	1d		0	B, T
2	**Analyse en Ontwerp**	**17 apr**	**22 jun**	**57d**	**38d**	**10d 2h**	**0**	
2.1	Requirements Analyse	17 apr	8 mei	26d	13d		0	B, T
2.2	**Systeem Ontwerp**	**9 mei**	**31 mei**	**12d**	**12d**	**23d 2h**	**0**	
2.2.1	Ontwerpdoelen	9 mei	11 mei	1d 4h	3d	8d	0	B
2.2.2	Decompositie	15 mei	25 mei	3d	6d	10d	0	B
2.2.3	Ontwerpbeslissingen	29 mei	31 mei	1d 4h	3d	10d	0	B
2.2.4	Ovezicht	9 mei	31 mei	6d	12d	23d 2h	0	B
2.3	**Object Ontwerp**	**9 mei**	**22 jun**	**19d**	**25d**	**10d 2h**	**0**	
2.3.1	Component Selectie	9 mei	9 mei	4h	1d	8d 2h	0	T
2.3.2	Initiële Dienst Specificatie	10 mei	31 mei	5d 4h	11d	10d 2h	0	T
2.3.3	Dienst Specificatie	1 jun	22 jun	13d	13d	10d 2h	0	B, T
3	**Productie**	**9 mei**	**27 jun**	**23d**	**27d**		**0**	
3.1	**Initiële Implementatie**	**9 mei**	**31 mei**	**6d**	**12d**		**0**	
3.1.1	Programmeren	9 mei	31 mei	3d	12d		0	T
3.1.2	Testen	9 mei	31 mei	1d 6h	12d		0	T
3.1.3	Code Documentatie	9 mei	31 mei	1d 1h	12d		0	T
3.2	**Implementatie**	**1 jun**	**22 jun**	**13d**	**13d**		**0**	
3.2.1	Programmeren	1 jun	22 jun	6d 4h	13d		0	B, T
3.2.2	Testen	1 jun	22 jun	3d 7h	13d		0	B, T
3.2.3	Code Documentatie	1 jun	22 jun	2d 4h	13d		0	B, T
3.3	Documentatie	26 jun	27 jun	4d	2d		0	B, T
4	**Afronding**	**28 jun**	**29 jun**	**2d**	**2d**	**6d 2h**	**0**	
4.1	Acceptatie	28 jun	28 jun	1d	1d	6d 2h	0	
4.2	Deployment	29 jun	29 jun	1d	1d	6d 2h	0	
5	**Rapportage**	**28 jun**	**12 jul**	**15d 4h**	**8d 2h**		**0**	
5.1	Verslag	28 jun	6 jul	11d	6d		0	B, T
5.2	Demo	10 jul	11 jul	2d	2d	2h	0	B, T
5.3	Presentatie Voorbereiding	10 jul	11 jul	2d	2d		0	B, T
5.4	Presentatie	12 jul	12 jul	4h	2h		0	B, T

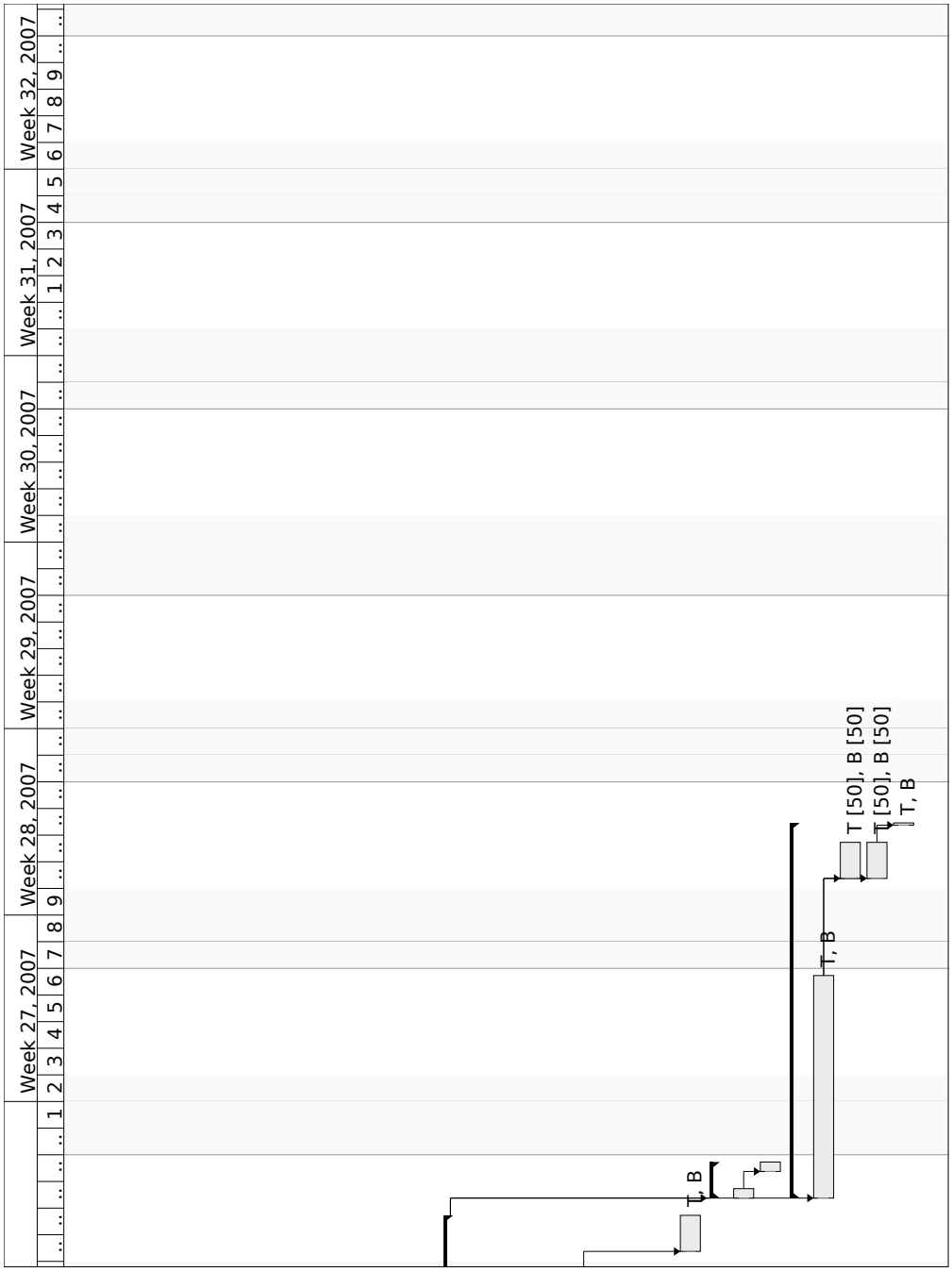
Figuur 1.1: Activiteiten planning



Figuur 1.2: Gantt diagram van de activiteiten planning (eerste gedeelte)



Figuur 1.3: Gantt diagram van de activiteiten planning (tweede gedeelte)



Figuur 1.4: Gantt diagram van de activiteiten planning, (derde gedeelte)

2 Mijlpalen

Tabel 2.1 geeft de mijlpalen van dit project aan, in chronologische volgorde.

Mijlpaal	Datum
Plan van Aanpak	16 april 2007
Planning	16 april 2007
Requirements Analysis Document	9 mei 2007
System Design Document	1 juni 2007
Object Design Document	26 juni 2007
Prototype	26 juni 2007
Documentatie	28 juni 2007
Verslag	9 juli 2007
Presentatie	12 juli 2007

Tabel 2.1: Mijlpalen

3 Taakverdeling

3.1 Ontwikkelaars

De ontwikkelaars op dit project zijn *Thomas de Ruiter* en *Berend Wouda*. Zij voeren alle ontwikkeltaken uit.

3.2 Taken

De taken tot de ontwerp fase bevatten heel veel samenwerking, gezien alles doorgesproken en overeengekomen moet worden. Pas na de analyse is het mogelijk ontwerptaken te verdelen op basis van bijvoorbeeld functionele eisen, en na het systeem ontwerp kunnen bijvoorbeeld subsystemen worden verdeeld onder de ontwikkelaars voor object ontwerp en implementatie. Deze paragraaf wordt aangepast naargelang verdelingen besloten worden.

3.2.1 Proces tijdens de analyse

Het is gebleken dat Berend zich het meest richt op de documentatie en Thomas op de implementatie, en dit stelt de ontwikkelaars in staat om in parallel veel gedaan te krijgen.

De (te) korte indeling van de implementatiefase is verlicht door een voorimplementatie (ten einde de ideeën die al bij de ontwikkelaars zelf liggen te testen), en de uitloop van de analyse is gecompenseerd door het systeemontwerp dat door de voorimplementatie en testen tevoorschijn komt. Dit is vooral gedaan door Thomas.

De analyse fase is richting het einde afgemaakt door Berend.

3.2.2 Proces tijdens het ontwerp

In de systeem ontwerp fase zijn de ontwerpdoelen vooral door Berend opgesteld (vanuit de nonfunctionele eisen), en de ontwerpbeslissingen zijn door Berend genotuleerd. De systeem decompositie is door Berend opgesteld en gediagrammeerd naar de opbouw van het prototype van Thomas. De subsysteem dienst specificatie is uitgesteld tot de object ontwerp fase.

In de object ontwerp fase is wederom de (kleinere hoeveelheid) documentatie door Berend gedaan, en heeft Thomas zich gericht op de specificatie van de objecten, vanuit en naast de implementatie.

3.2.3 Proces tijdens de implementatie

Na het afmaken van de object ontwerp documentatie hebben Berend en Thomas zich samen op de specificatie tegelijk met de implementatie gericht, teneinde de deadline te kunnen halen. Wat er nog miste aan de specificatie is voortgekomen uit de implementatie.

Appendix D: Requirements Analysis Document

CMS 3
Requirements Analysis Document

Berend Wouda Thomas de Ruiter

May 11, 2007

Preface

This document is the *Requirements Analysis Document* for the *CMS3* project for *7U Digital Handmade Originals*, done for the *Delft University of Technology 'IN3700 BSc-Project'* by *Thomas de Ruiter* and *Berend Wouda*.

This document contains the elicitated and analyzed requirements for the project. It contains

- an overview and analysis of the functions of the system to be replaced,
- the extracted wishes of the client (analyzed and formalized to requirements), and
- an initial model of the new system based on these functions and requirements.

This is a cyclic deliverable, so this version may not yet be complete.

Each version is to be agreed to by the client. The final version serves as an agreement over the functionality of the new system, defines the requirements to it, and specifies an initial model for it.

Version

9 May 2007

Initial version.

11 May 2007

A number of spelling errors that were found have been fixed. Furthermore, the RAD has been adapted to better describe the Action, Webservice and User Agent entity objects. The configuration files section has also been reworded.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose of the system

The purpose of the system is to provide a framework to ease the development of web applications by supporting extension through modules.

1.2 Scope of the system

The system is capable of managing modules and data on the basic, generic level. It does not recognize any specific functionality or differentiate between specific forms of data, enabling extensions to do this instead.

1.3 Objectives and success criteria of the project

1.3.1 Objectives

The main objectives of the project are listed in table 1.1.

Objective	Definition
• The proper, full-fledged, well documented design of the base system that supports extension through modules.	**OBJ1**
• The successful implementation of at least the base system.	**OBJ2**
• The provision of API documentation and a manual for implementing modules.	**OBJ3**

Table 1.1: Objectives

1.3.2 Success criteria

The success criteria of the project are listed in table 1.2.

Criterium	Definition
• The system provides a basis where (end)products can be created on with the use of extensions.	**SC4**
• The system is compatible with PHP version 5.2.1.	**SC5**
• The system is compatible with MySQL versions 4.1.22 through 5.0.37.	**SC6**
• The documentation supports the creation and development of modules.	**SC7**

Table 1.2: Success criteria

1.4 Definitions, acronyms, and abbrevations

CMS3 The shorthand form for *7U Content Management System* version 3, the full name of the system.

Base, Basis, Core The form of the system in that it itself does not do much specificly, but instead provides the generic loading and execution of extensions that do. These words may be used interchangeably.

Module, Extension A piece of software that 'plugs in' the base system and provides additional, specific functionality. The two words may be used interchangeably.

1.5 References

Plan van Aanpak The Plan of Approach (in Dutch) details the global approach of the project and the reasons for the project. It can be found at: `https://pavlov/trac/cms3/wiki/PlanvanAanpak`

1.6 Overview

1.6.1 Function

The system enables a modular design methodology that supports easy creation of web applications through the use of extensions. Together with the provided documentation it allows developers to create their own modules and run them on the basis.

Optionally, some modules may already be implemented. These are modules for technical management, easy management for customers, and an implementation for news, pages and folders.

1.6.2 Reason

The current CMS employed by 7U is not a very integral entity. The design ideas are good, but the implementation needs a fresh start: it was implemented piece by piece over the course of time, management, developers and resources. The current implementation consists of various pieces of specific code implemented when it was needed for a customer. Abstracting this out into an extension managing engine, with specific functionality in modules instead, will support easier and faster creation of (optionally very specific) web applications, and will allow far easier maintenance.

2 Current system

Please note that the following sections describe only the parts of the system that are interesting for analysis for the new system. The current system contains a lot more aspects, but these are not wanted in the new system.

2.1 Overview

The *7U Content Management System* is a framework for programming web applications. It allows (web)users to retrieve output generated by classes or objects stored in a database, and it allows them to edit the properties of those objects. It also allows the access to other available actions or webservices of a class or object. A templating system is used to output HTML, and can include the output of other classes or objects.

2.2 Concepts

2.2.1 Class

A *class* is an entity that defines the make-up of an object. It specifies the actions and webservices of an object, and its fields. It may also specify static actions and webservices.

2.2.2 Object

An *object* is an instance of a class. It has its own values for all its fields. Its actions and webservices are defined in its class. **...**

2.2.3 Action

An *action* is a behaviour that an object or class can perform on request, which communicates in the form of webpages. It has a name.

2.2.4 Webservice

A *webservice* is a behaviour that an object or class can perform on request, which communicates in the form of queries. It has a name.

2.2.5 Field

A *field* is a property of one object. It may contain a value (the type of the value is defined in the object's class).

2.2.6 Relation

A *relation* is a link between two objects (which may be the same). It has a name, and may be directed or undirected. If it is directed, it may have atttributes associated with it.

2.2.7 Right

A *right* is an authorization for one object to interact with another object. It has a type (read, write, delete, or super).

2.2.8 Template

A *template* is a predefined layout that includes dynamic content from objects in it.

2.2.9 Include

An *include* is a special type of object that outputs common webpage elements.

2.3 Functionality

2.3.1 Class access

Classes can be requested to perform an action or webservice (static).

2.3.2 Object access

Objects can be requested to perform an action or webservice (dynamic/instantiated).

2.4 System design

2.4.1 Objects

The CMS maintains a hierarchy of 'objects'. These objects are represented in a MySQL database as rows in a table. The table contains fields stating the class of the object, the object's id, name and parent. Some meta-data like the author, edit date and a sequence number for sorting purposes is also present in this table. The fields and their types are defined in the object's class, database tables are created automatically by the CMS if they don't exist.

2.4.2 Fields

The object instances can have attributes which are stored in the database, these are called 'fields'. Each object class that has fields has it's own table in the database to extend the standard objects table with the fields. The various field types are defined in 'field type' objects which perform transformations on the field prior to saving to the database or after reading from the database. The field type object can also generate standard views to show or edit the field value.

2.4.3 Relations

Two systems are implemented to link objects to each other. This can be used, for example, to link products to a shopping cart. The older system provides named undirected links, while the newer system provides named directed links. Additionally the newer system has the ability to store some attributes together with the link, for example an attribute can be used to indicate the ordered quantity of a product.

2.4.4 Actions

Objects can be viewed from a web agent by specifying the URL referring to the object id, and additionally you can specify an action. An example URL '`http://www.samiko.nl/index.php?objectID=39&action=edit`' would show the edit view of the object with id 39.

If no action is specified, the default action is 'show', which should generate the default view of the object.

2.4.5 Templates

The generation of the return data of actions is aided by the template engine smarty[1]. The action's object method collects the needed data, then smarty transforms this data with a template file in the needed output. This way, the layout is separated from the logic.

2.4.6 Includes

In the templates, 'includes' can be 'included'. An include can generate often used elements like a main menu or a display of the employee of the month.

2.4.7 Webservice

Object methods with names starting with an underscore, can be called from an URL. The return value is returned encoded in JSON[2] format, arguments to the methods should be encoded in the same format. The webservices are primarily used in AJAX[3] applications, which are of course very 'Web 2.0[4]'.

2.4.8 Static

Actions and webservices can also be called statically. This way, a method on a class can be called without referring to a object instance. This could be used for example to generate a view to create a new instance of an object.

2.4.9 Access management

The CMS employs an access management system to restrict access to objects. Users in the CMS are normal CMS objects, therefore the heart of this system is a database table which grants the rights 'read', 'write' and 'delete' on objects to other objects. When a new object is created, the rights of its parent are copied to the new object. In addition to this, the creator is granted all the rights to the object. The superusers have entries for all the rights without a target object, this way these rights apply on all objects.

2.4.10 Configuration files

Configuration files for the system can be found in three places. At first the core contains a configuration file with default values. Secondly a configuration file is placed in the 'extensions' folder, this file contains defaults for the extension, it can override the values from the core configuration file. The third configuration file is placed in the web root, it's values override both other configuration files, so it is to be used for site specific values.

Object classes can also have individual configuration files which specify what kind of parent and child objects are allowed. Object instances can have some custom key-value pairs, these can be used for example to specify a special template for a specific object.

2.4.11 Template generation

To ease development, basic templates for objects without a show or edit template can be generated by the CMS itself. These generated templates are saved as normal template files and can be further customized.

[1] `http://smarty.php.net/`
[2] `http://www.json.org/`
[3] `http://nl.wikipedia.org/wiki/Asynchronous_JavaScript_and_XML`
[4] A controversial phrase by Tim O'Reilly to indicate a 'new sort of Internet applications'

2.5 Problems

2.5.1 Code

Because the current system had to stay backwards compatible with the last few versions, and because new features had to be implemented at short notice to satisfy customers and designers, the code has over time become more and more messy. The code contains features which none of the developers remember. Many things can be done in multiple ways. Some (customer) specific functionalities are currently integral parts of the system as well.

3 Case studies

Please note that the following sections describe only the parts of the cases that may be interesting for analysis for the new system. They contain a lot more aspects, but many of these are not interesting for the new core system.

3.1 Installatievisie

3.1.1 Overview

Samiko is a professional association for plumbers, it set up *Installatievisie*[1], a portal to present its members and their products and services. 7U implemented this website using the CMS.

3.1.2 Features

Some features that make this website special are:

- It features the 'magazine', a catalog containing all products. Each product has links to the Samiko members that deliver the product.
- A member of *Samiko* can have his own site within the system. It will be accessible through its own domain name. The member is given a choice of some standard layouts or a custom layout can be made. On this site, the 'magazine' only shows products which can be delivered by the member.

3.1.3 Aspects

The following aspects of this site could be interesting for the system.

- Linking between two objects.
- Different behaviour based on the requested domain name.

[1] `http://www.installatievisie.nl/`

4 Proposed system

4.1 Overview

The proposed system is a CMS core to which modules can be added.

Its primary functions revolve around the access of entities and performing actions or webservices on/of those entities, by way of requests. Entities have access control placed upon them.

4.2 Functional requirements

The functional requirements that the users have of the system are listed in table 4.1. They have been deduced from the use cases, which are listed in the 'System models' section.

Note that these requirements are very generic, but this is on purpose. Because the system is a framework, it doesn't implement specific functionality. And since it is a web application, it therefore only deals with requests. The functional requirements are therefore the root of all the required request functionality. In this context, users may even be other systems. A difference is made between users and user agents to better stress the notion of webservicing.

4.3 Non-functional requirements

There are different types of non-functional requirements on the system:

- Table 4.2 gives the security related requirements of the system.
- Table 4.3 gives the quality related requirements of the system.
- Table 4.4 gives the performance related requirements of the system.
- Table 4.5 gives the debugging related requirements of the system.
- Table 4.6 gives the documentation related requirements.

These requirements have been extracted from the client during interviews.

4.4 Pseudo requirements

Due to the nature of the project and the system to be implemented, the pseudo requirements have been split up into two groups.

4.4.1 Project requirements

These are the general requirements that the client has of the project and its process. They are listed in table 4.7.

Requirement	Definition
• A user can request an action from a class.	**FR1**
• A user agent can request a webservice from a class.	**FR2**
• A user can request an action from an object.	**FR3**
• A user agent can request a webservice from an object.	**FR4**
• A user can request a file.	**FR5**

Table 4.1: Functional requirements

Requirement	Definition
• Access to objects must be restricted by a rights management system.	**NFR1**
• Invocation of actions and webservices on a class must be restricted by a rights management system.	**NFR2**
• Access of files must be restricted by a rights management system.	**NFR3**
• The system must not allow users to access otherwise inaccessible parts of the underlying platform.	**NFR4**
• Care must be taken to not make the system vulnerable to SQL injections.	**NFR5**
• Care must be taken to not make the system vulnerable to cross site scripting (XSS).	**NFR6**

Table 4.2: Security requirements (non-functional)

Requirement	Definition
• If a module generates an error, the system may graciously fail.	**NFR7**

Table 4.3: Quality requirements (non-functional)

Requirement	Definition
• The system should process requests within at most 10 seconds.	**NFR8**
• The system should be able to handle at least 10 requests per second.	**NFR9**

Table 4.4: Performance requirements (non-functional)

Requirement	Definition
• The system must have a 'development mode' and a 'production mode'.	**NFR10**
• In 'development mode' the system should log performance information like render times.	**NFR11**
• In 'development mode' the system should render errors and exceptions.	**NFR12**
• In 'production mode' the system should email detailed information of errors and exceptions to a specified email address and render a generic error message.	**NFR13**

Table 4.5: Debug requirements (non-functional)

Requirement	Definition
• A module creation manual should be made. It should support the easy creation of modules, giving clear instructions and the necessary requirements.	**NFR14**

Table 4.6: Documentation requirements (non-functional)

4.4.2 Implementation requirements

Developers also play a large role in the requirements to the new system. Because it is a framework that is to support modules, the design of the system is very important to developers.

The following are pseudo requirements of more in-depth design and implementation of the system, elicitated through exhaustive interviews with expert clients.

- Table 4.8 lists the requirements regarding modules.
- Table 4.9 lists the requirements regarding templates.
- Table 4.10 lists the requirements regarding databases.
- Table 4.11 lists the miscellaneous implementation requirements.

4.5 System models

4.5.1 Actors

The following are the actors interacting with the system:

User A person or computer entity using the system, such as a customer.

Power User A person or computer entity that performs secondary tasks such as installation and maintenance, such as an administrator.

User Agents A computer entity acting on behalf of the user, such as a webbrowser.

Developer A person that creates and maintains (functional) aspects of the system and its modules, such as an Elja.

4.5.2 Scenarios

The following scenarios are a limited number of examples of the use of the system, based on client elicitation and the system that will be replaced. They vary from theoretical installations and setups typical to (power) users, to more generic approaches typical to furtherthinking developers and other types of user agents.

The name after a colon indicates the use case that was later abstracted from (amongst others) that scenario. The scenario is an instance of that use case.

When an error occurs, the scenario ends prematurely. The steps afterwards will then not be executed.

- Tables 4.12, 4.13, 4.16, 4.18, and 4.19 show scenarios typical of a general web user. Of this kind of scenario an inifinite amount can be elicited among the clientbase.
- Tables 4.14, 4.15, 4.17, 4.20, 4.21, 4.22, and 4.23 show more generic scenarios. They contain (different levels of) a developer's view of the basic in-system functionality.

Note that these scenarios are by far not exhaustive! However, they already give a very good image of the system's functionality, and a complete set of use cases can be abstracted out of them. This is because it is known in advance that the sought functionality has to be very general, and because there are symmetries between the different use cases.

Requirement	Definition
• The system must be programmed in PHP, and must be compatible with PHP 5.2.1.	**PR1**
• Any included third party software must come with a license that allowes commercial use and redistribution.	**PR2**
• There should be a tool to do the initial installation of the CMS (installing the software, and creating the database and its tables).	**PR3**
• There should be a tool to do the installing, updating and removing of modules.	**PR4**

Table 4.7: Project requirements (pseudo)

Requirement	Definition
• The system must be divided into modules.	**PR5**
• Each module must be able to contain PHP code and resources like templates, template resources, images, and other files.	**PR6**
• Modules may contain metadata such as their version number, changelog, copyright information, and license.	**PR7**
• All PHP code in the modules must follow the set conventions for naming, packaging and coding (these should be defined in another document).	**PR8**
• Modules should contain unit tests for all the contained classes.	**PR9**
• Any included third party software should be placed unmodified in a separate folder and must be encapsulated somewhere inside the system.	**PR10**

Table 4.8: Module requirements (pseudo)

Requirement	Definition
• The templating system should recognize the type of template and use the right templating engine automatically (if available).	**PR11**
• The templating system should support Smarty.	**PR12**
• The system may support multiple template engines.	**PR13**
• The system must have 'includes' which can be used in 'templates'.	**PR14**

Table 4.9: Template requirements (pseudo)

Requirement	Definition
• The system should have a database abstraction layer.	**PR15**
• The system should at least support MySQL (versions 4.1.22 through 5.0.37).	**PR16**
• The system may support PostgreSQL.	**PR17**
• The system may support SQLite.	**PR18**

Table 4.10: Database requirements (pseudo)

Requirement	Definition
• The system should have a session class that manages session data and browser languages.	**PR19**
• Objects should be able to delete all their children if requested.	**PR20**

Table 4.11: Miscellaneous implementation requirements (pseudo)

Scenario name	`viewProduct: RequestObjectAction`
Participating actor instances	• `me: User`
Flow of events	1. I am browsing a webshop that runs on the system and click on a product. 2. The system finds the product in its database. 3. The system makes the product's page. 4. The system sends me the page and I see it.

Table 4.12: viewProduct scenario

Scenario name	`editItem: RequestObjectAction`
Participating actor instances	• `me: User`
Flow of events	1. I'm looking at an item in the system and I click 'Edit'. 2. The system finds my item. 3. The system asks the item to allow me to edit it. 4. If I can, the item makes an edit page for itself and sends it to me, otherwise it sends me an error. 5. I fill out the edit page forms and click 'Submit'. 6. The system updates the item if possible. 7. If it could, the item makes a view of itself and sends it to me, otherwise it sends me an error.

Table 4.13: editItem scenario

Scenario name	`viewFolder: RequestObjectAction`
Participating actor instances	• `me: User`
Flow of events	1. I start up my webbrowser and enter a URL to a folder object. 2. The system retrieves the folder object from its database. 3. The folder object lists its child objects on a page as links to their URLs. 4. The system sends the page to my browser which then shows me it.

Table 4.14: viewFolder scenario

Scenario name	`haveEntityDoSomething: RequestObjectAction`
Participating actor instances	• `me: User`
Flow of events	1. I approach an entity available in the system, and tell it to do something. 2. The system retrieves the entity from its database. 3. The entity checks if it can do the thing I asked. 4. If so, the entity does the thing I asked, and it generates a representation of the result. Otherwise, it generates an error. 5. The system sends either the representation or the error to my browser, which then shows it to me.

Table 4.15: haveEntityDoSomething scenario

Scenario name	`viewAdmin: RequestClassAction`
Participating actor instances	• `me: Power User`
Flow of events	1. I start up my webbrowser and go to the administrative interface using my bookmark. 2. The system checks to see if I'm allowed access to the admin page. 3. If so, the system makes the admin page. Otherwise, it makes an error. 4. The system sends me the page or the error and I see it in my browser.

Table 4.16: viewAdmin scenario

Scenario name	`requestAdmin: RequestClassAction`
Participating actor instances	• `me: Power User`
Flow of events	1. I direct my webbrowser to the administrative interface using its specific URL. 2. The system inspects the URL and loads the administration class. 3. The class checks to see if I have access to the administrative interface. 4. If I do, the class renders the administrative interface. Otherwise, it renders an error. 5. The system sends me either page and I see it.

Table 4.17: requestAdmin scenario

Scenario name	`createObject: RequestClassAction`
Participating actor instances	• `me: User`
Flow of events	1. I click a link to create a new thing on a site powered by the system. 2. If I'm allowed to create such a thing, the system generates a creation page and sends it to my webbrowser. Otherwise, it generates and sends me an error. 3. In case I have been allowed, I fill out the creation page and click 'Submit'. 4. The system creates such a new thing and puts it on the site. 5. The system generates the new page and sends it to my browser so I can see it.

Table 4.18: createObject scenario

Scenario name	`displayChildren: RequestObjectWebservice`
Participating actor instances	• `me: User Agent`
Flow of events	1. An object is clicked in a page that I'm displaying (the page was generated by the system). 2. I send the system a query asking to give me the children of the selected object. 3. The system executes the query. 4. The system sends me the children of the object. 5. I display the children of the object in a list under the object.

Table 4.19: displayChildren scenario

Scenario name	`requestChildren: RequestObjectWebservice`
Participating actor instances	• `me: User Agent`
Flow of events	1. An object is clicked in a system-generated page that is being displayed by me. 2. I access the system through an object specific URL, and transmit a query asking for the children of the object. 3. The system retrieves the object from its database. 4. The object executes the query. 5. The system sends back the result of the object execution (the children of the object). 6. I receive the list of children and display them in a list under the object.

Table 4.20: requestChildren scenario

Scenario name	`performLocalFunctionality: RequestClassWebservice`
Participating actor instances	• `me: User Agent`
Flow of events	1. A user activates some local functionality in a system-generated page that is being displayed by me. 2. I access the system through a URL specific to the requested functionality, and transmit a query asking this functionality to be executed. 3. The system receives the query, and loads the indicated class. 4. The class executes the query. 5. The system sends back the result of the execution. 6. I receive the results and process them.

Table 4.21: performLocalFunctionality scenario

Scenario name	`viewPageWithImage: RequestFile`
Participating actor instances	• `me: User`
Flow of events	1. I use my webbrowser to view a page that contains a static[a] image on the system and it is rendering the generated result. 2. The browser accesses the system to retrieve the image, through its URL on the page. 3. The system inspects this URL and uses the information encoded in it to locate the appropriate image file. 4. The system streams this image file back to the browser. 5. The browser displays the image. [a]For example a template resource image.

Table 4.22: viewPageWithImage scenario

Scenario name	`loadCSS: RequestFile`
Participating actor instances	• `me: User Agent`
Flow of events	1. I load a page that contains a CSS stylesheet reference. 2. I accesses the system to retrieve the CSS through its URL. 3. The system locates the appropriate CSS file. 4. The system streams this file back to me. 5. I load in the CSS.

Table 4.23: loadCSS scenario

4.5.3 Use case model

The previous scenarios can be abstracted into 5 different use cases, with their extensions. Each scenario has already indicated of which of these use cases it is an instantiation.

1. Table 4.24 gives the use case for requesting an action from a class.
2. Table 4.25 gives the use case for requesting a webservice from a class.
3. Table 4.26 gives the use case for requesting an action from an object.
4. Table 4.27 gives the use case for requesting a webservice from an object.
5. Table 4.28 gives the use case for requesting a file.

The extensions are as follows:

1. Table 4.29 gives the use case extension for when an action may not be executed.
2. Table 4.30 gives the use case extension for when a webservice may not be executed.
3. Table 4.31 gives the use case extension for when a file may not be accessed.

Note that the use cases do not include technical error behaviour to reduce the amount of extensions. In all cases it may be assumed that when a step fails (due to for example incorrect input) an error page is generated and sent to the requester. See the non-functional requirements for more details on how errors are handled.

Figure 4.1 shows the notation of the use cases in an UML diagram. In this diagram the relations between use cases, and between use cases and actors can be easily seen.

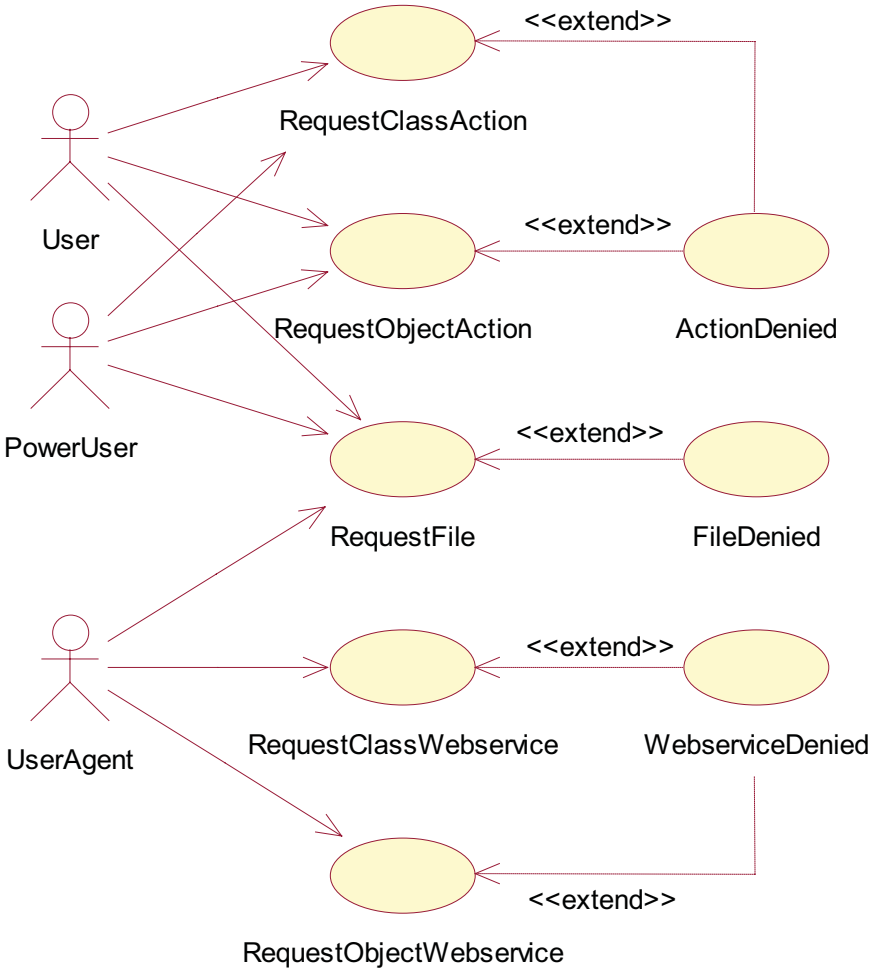


Figure 4.1: Use case diagram.

Use case name	`RequestClassAction`
Participating actors	• Initiated by `User` / `Power User`
Entry condition	1. The `User` / `Power User` performs an HTTP GET or POST request to the system. As request parameters, he supplies a fully qualified classname, and an action name with optionally action parameters.
Flow of events	2. The system locates the specified class and loads it. 3. The class checks if the specified action may be executed by/for the `User` / `Power User` . 4. The class executes the action. 5. The class generates a page with the result of the action. 6. The system performs an HTTP response back to the `User` / `Power User` . As response parameter, it supplies the page.
Exit condition	7. The `User` / `Power User` receives the page.
Special requirements	• None.

Table 4.24: RequestClassAction use case.

Use case name	`RequestClassWebservice`
Participating actors	• Initiated by `User Agent`
Entry condition	1. The `User Agent` performs an HTTP GET or POST request to the system. As request parameters, it supplies a fully qualified class-name, and a webservice name with optionally webservice parameters.
Flow of events	2. The system locates the specified class and loads it. 3. The class checks if the specified webservice may be executed by/for the `User` represented by the `User Agent` . 4. The class executes the webservice. 5. The system performs an HTTP response back to the `User Agent` . As response parameter, it supplies the result of the webservice.
Exit condition	6. The `User Agent` receives the result.
Special requirements	• None.

Table 4.25: RequestClassWebservice use case.

Use case name	`RequestObjectAction`
Participating actors	• Initiated by `User` / `Power User`
Entry condition	1. The `User` / `Power User` performs an HTTP GET or POST request to the system. As request parameters, he supplies a path to an object, and an action name with optionally action parameters.
Flow of events	2. The system retrieves the specified object from its database. 3. The object checks if the specified action may be executed by/for the `User` / `Power User` . 4. The object executes the action. 5. The object generates a page with the result of the action. 6. The system performs an HTTP response back to the `User` / `Power User` . As response parameter, it supplies the page.
Exit condition	7. The `User` / `Power User` receives the page.
Special requirements	• None.

Table 4.26: RequestObjectAction use case.

Use case name	`RequestObjectWebservice`
Participating actors	• Initiated by `User Agent`
Entry condition	1. The `User Agent` performs an HTTP GET or POST request to the system. As request parameters, it supplies a path to an object, and a webservice name with optionally webservice parameters.
Flow of events	2. The system retrieves the specified object from its database. 3. The object checks if the specified webservice may be executed by/for the `User` represented by the `User Agent` . 4. The object executes the webservice. 5. The system performs an HTTP response back to the `User Agent` . As response parameter, it supplies the result of the webservice.
Exit condition	6. The `User Agent` receives the result.
Special requirements	• None.

Table 4.27: RequestObjectWebservice use case.

Use case name	`RequestFile`
Participating actors	• Initiated by `User` / `Power User` / `User Agent`
Entry condition	1. The `User` / `Power User` / `User Agent` performs an HTTP GET or POST request to the system. As request parameters, he supplies a path to a file.
Flow of events	2. The system locates the specified file. 3. The system checks if the specified file may be accessed by the `User` / `Power User` / `User Agent` . 4. The system performs an HTTP response back to the `User` / `Power User` / `User Agent` . As response parameter, it supplies the file.
Exit condition	5. The `User` / `Power User` / `User Agent` receives the file.
Special requirements	• None.

Table 4.28: RequestFile use case.

Use case name	`ActionDenied`
Extends	• RequestClassAction • RequestObjectAction
Invoked	• If the `User` / `Power User` / `User Agent` is not allowed to execute the specified action.
Flow of events	1. The object/class generates an error page. 2. The system performs an HTTP response to the `User` / `Power User` / `User Agent` . As response parameter, it supplies the error page.

Table 4.29: ActionDenied use case.

Use case name	`WebserviceDenied`
Extends	• RequestClassWebservice • RequestObjectWebservice
Invoked	• If the `User Agent` is not allowed to execute the specified webservice.
Flow of events	1. The system performs an HTTP 403 response to the `User Agent` .

Table 4.30: WebserviceDenied use case.

Use case name	`FileDenied`
Extends	• RequestFile
Invoked	• If the `User` / `User Agent` is not allowed to access the specific file.
Flow of events	1. The system performs an HTTP 403 response to the `User` / `User Agent` .

Table 4.31: FileDenied use case.

4.5.4 Object model

From the use cases, a number of system objects can be deduced. These can be divided into boundary, control, and entity objects, and relations between them can be discovered.

4.5.4.1 Data dictionary

The following sections list the system's initial (high-level) objects. Figure 4.2 shows all the details of these objects in a (combined) object diagram.

Boundary objects From the use cases we know that all communication with the `User` goes through requests and responses. Thus, the following are the boundary objects:

- **Request**: The request a user sends to the system.
- **ClassActionRequest**: The request a user sends to the system to have a class execute an action.
- **ClassWebserviceRequest**: The request a user sends to the system to have a class execute a webservice.
- **ObjectActionRequest**: The request a user sends to the system to have an object execute an action.
- **ObjectWebserviceRequest**: The request a user sends to the system to have an object execute a webservice.
- **FileRequest**: The request a user sends to the system to retrieve a file.
- **Response**: The response a user gets from the system.
- **ClassActionResponse**: The response a user gets from the system when a class executed an action.
- **ClassWebserviceResponse**: The response a user gets from the system when a class executed a webservice.
- **ObjectActionResponse**: The response a user gets from the system when an object executed an action.
- **ObjectWebserviceResponse**: The response a user gets from the system when an object executed a webservice.
- **ActionDeniedResponse**: The response a user gets from the system when an action may not be executed.
- **WebserviceDeniedResponse**: The response a user gets from the system when a webservice may not be executed.
- **FileDeniedResponse**: The response a user gets from the system when a file may not be retrieved.

Control objects The use cases all share a common functionality, namely the notion of a request coming in and being handled. This gives rise to the following control object for all use cases:

- **RequestHandler**: Identifies the target Class or Object and makes it perform the specified Action or Webservice, or retrieves the target File.

Entity objects The use cases dictate the following entity objects:

- **Class**: A static entity with static attributes, actions and webservices.
- **Object**: An instance of a Class, with instance specific attributes, actions and webservices.
- **File**: A file containing data.
- **Action**: A function of a Class or Object that generates an output directly for the the User. The User's UserAgent will render this output directly. Such an output may be a page, an inline file, or a download.
- **Webservice**: A function of a Class or Object that generates a query response (a little bit of data) as output. A User's UserAgent interpretes the response and parses its data, and uses it for client-side functionality.
- **User**: An Object that (represents the entity that) uses the system.
- **PowerUser**: A User that is allowed to change the system, and/or install/modify it.
- **UserAgent**: A client system acting on behalf of a User. This system can access functionality hidden from the User.

Figure 4.5 shows the relations between these objects in a class diagram. It also shows the ClassLoader, which is an utility object.

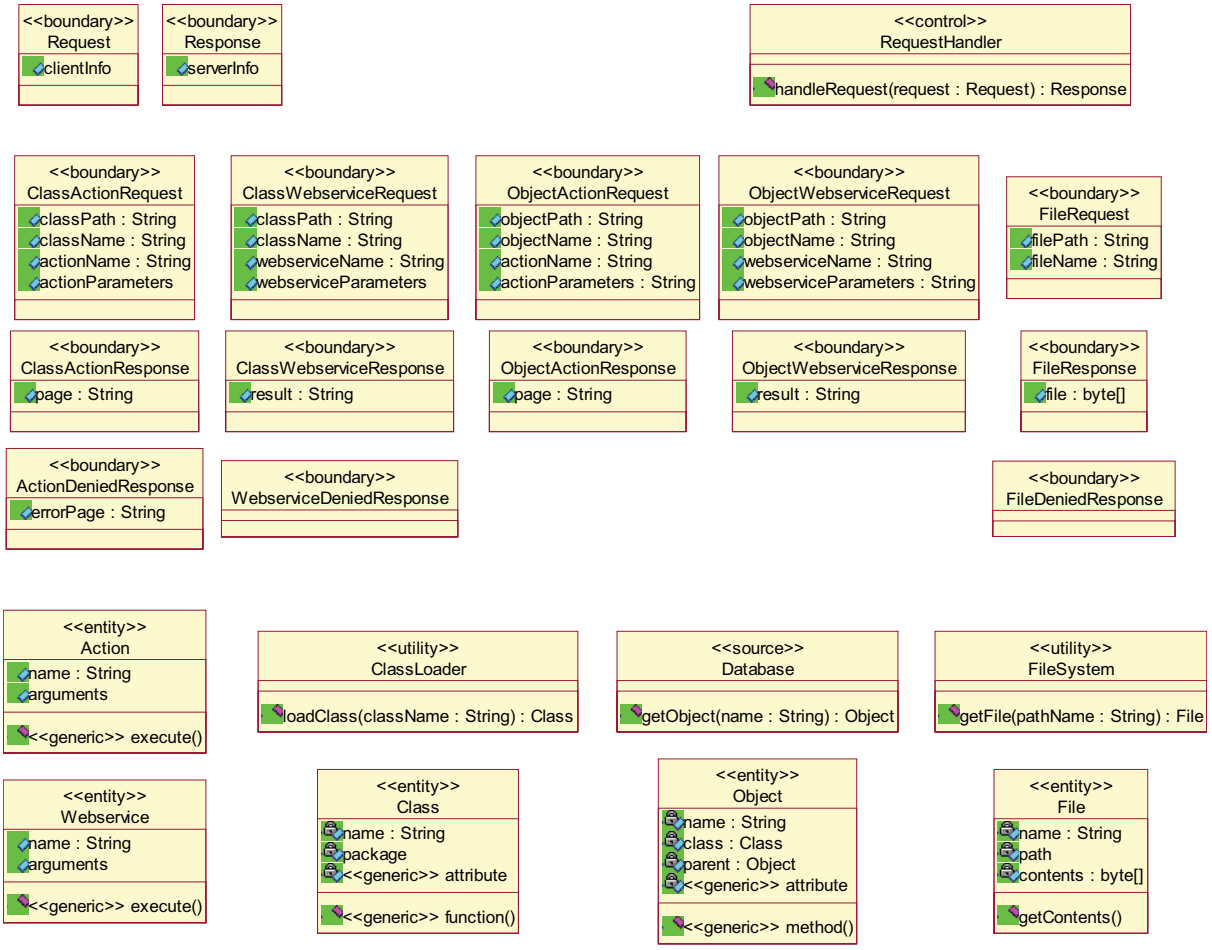


Figure 4.2: Data dictionary (object diagram).

4.5.4.2 Class diagrams

The aforementioned classes have relationships among themselves, which are shown in the following figures:

- Figure 4.3 shows the class diagram for the boundary objects. Note the dependancy arrows in the figure.
- Figure 4.4 shows the class diagram for the control objects.
- Figure 4.5 shows the class diagram for the entity objects. Although they are not entity objects, ClassLoader, Database and FileSystem are also included. Some entity classes have multiplicity specified; the Class and Object entities have zero or more actions and/or webservices while each action/webservice is linked to precisely one class. This means actions and webservices cannot be shared between entities. Also, there is a 1:n relationship between Classes and Objects, because Objects are instances of Classes.

There are no direct links between the three groups of classes, in terms of relationships. See the dynamic models section for information about interrelations between the groups.

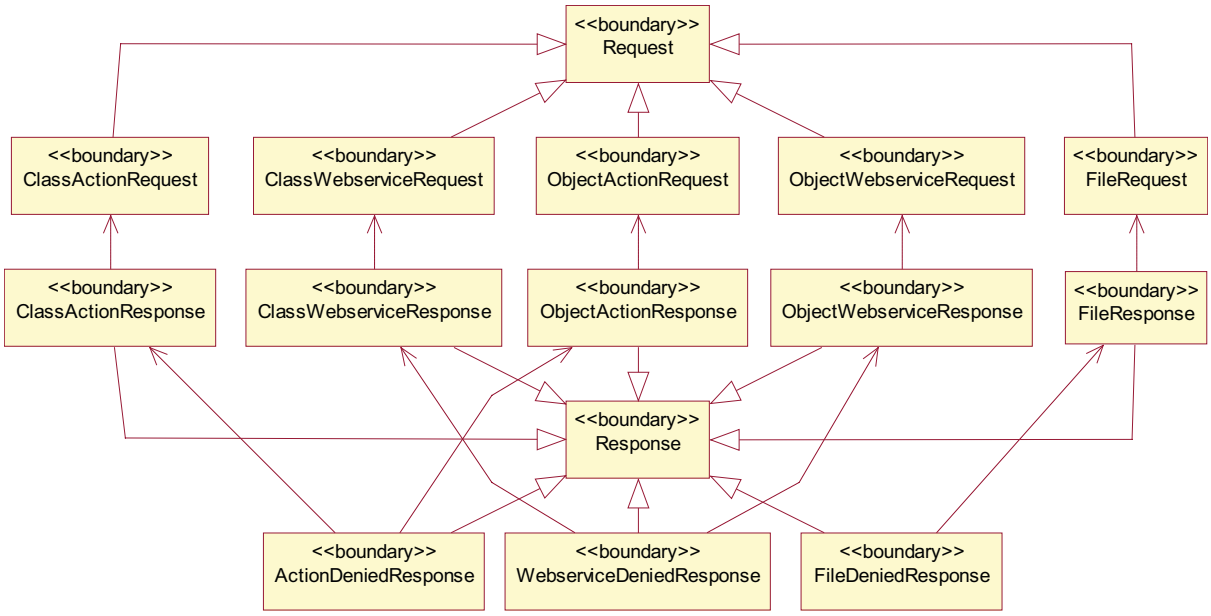


Figure 4.3: Class diagram for the boundary objects.

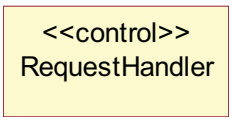


Figure 4.4: Class diagram for the control objects.

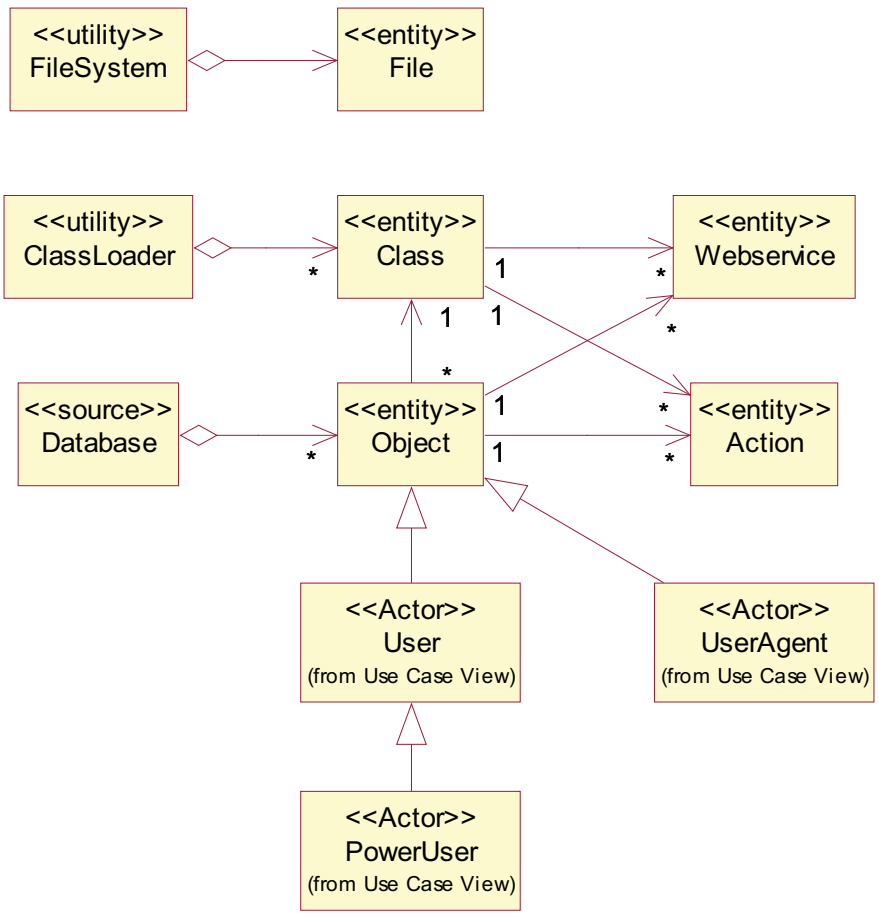


Figure 4.5: Class diagram for the entity objects.

4.5.5 Dynamic models

The use cases can be realized using sequence diagrams. The following 5 sequence diagrams correspond to the system's use cases:

- Figure 4.6 shows the sequence diagram for the RequestClassAction use case.
- Figure 4.7 shows the sequence diagram for the RequestClassWebservice use case.
- Figure 4.8 shows the sequence diagram for the RequestObjectAction use case.
- Figure 4.9 shows the sequence diagram for the RequestObjectWebservice use case.
- Figure 4.10 shows the sequence diagram for the RequestFile use case.

All sequence diagrams follow a similar pattern. They:

1. receive the request,
2. load the entity,
3. access the entity,
4. perform the functionality, and
5. return the result.

For file access step 4 is skipped.

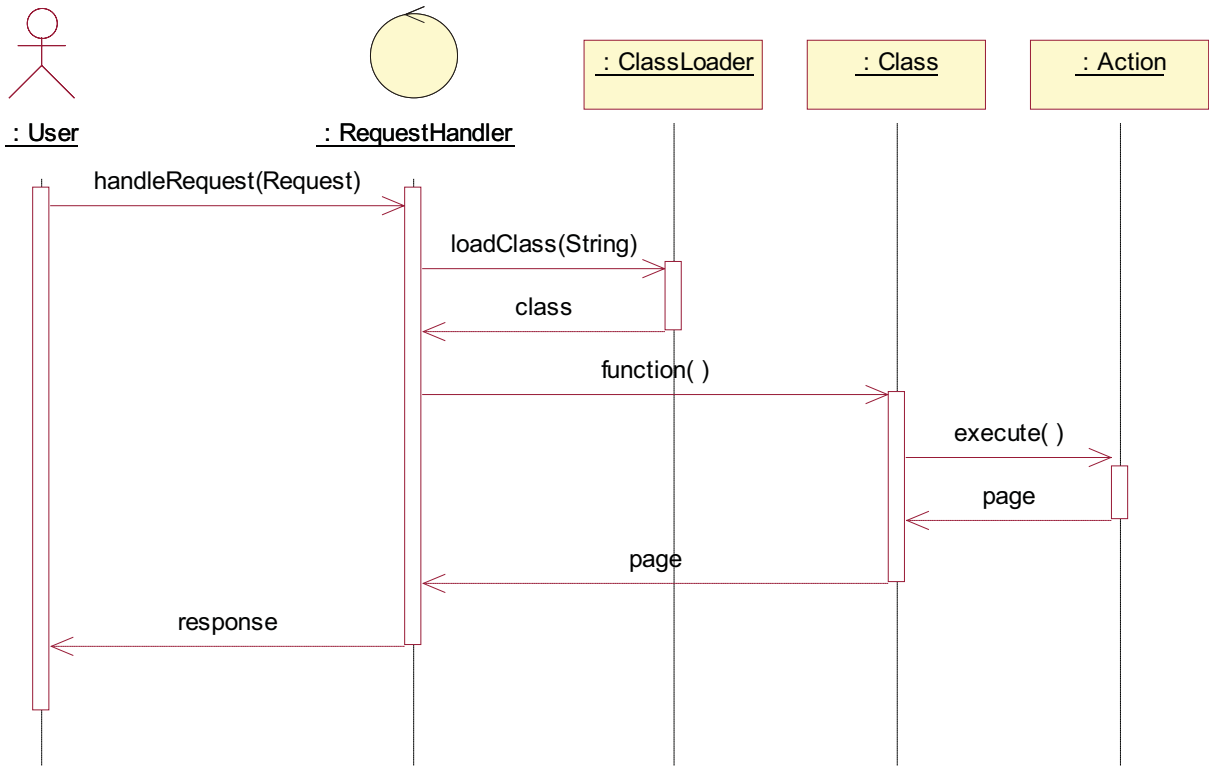


Figure 4.6: Sequence diagram for the RequestClassAction use case.

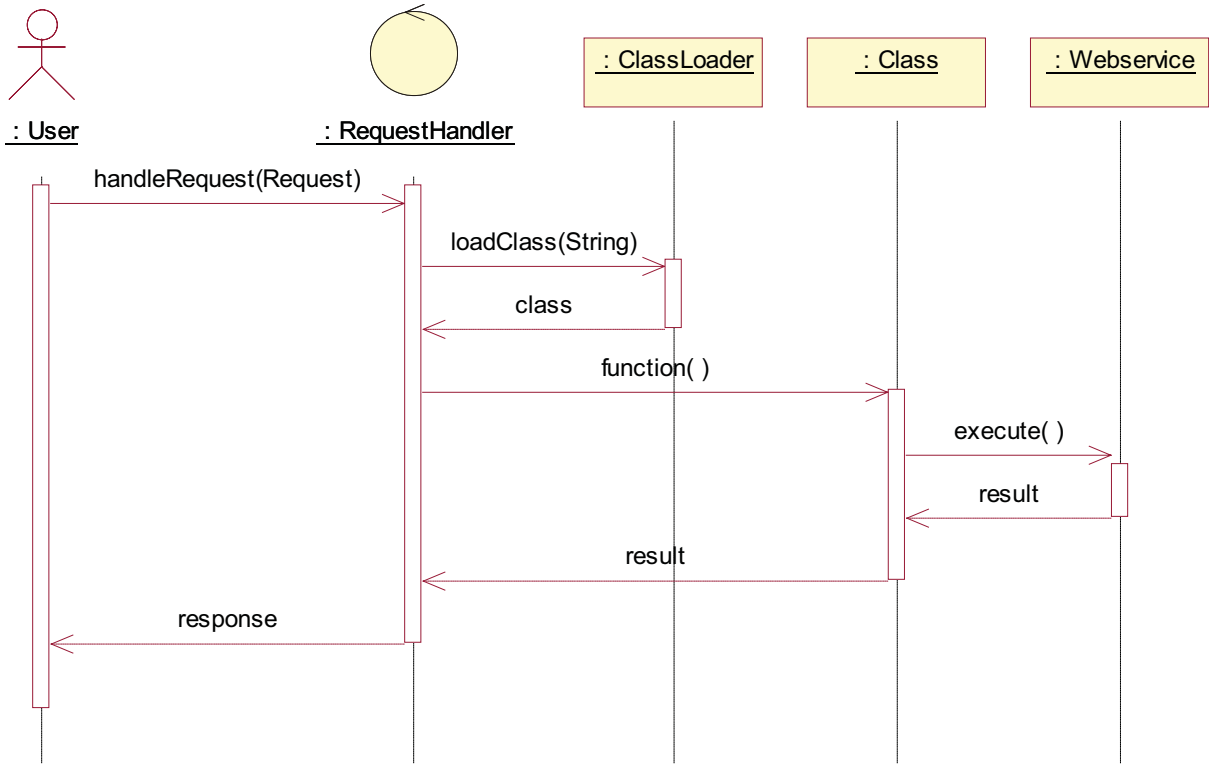


Figure 4.7: Sequence diagram for the RequestClassWebservice use case.

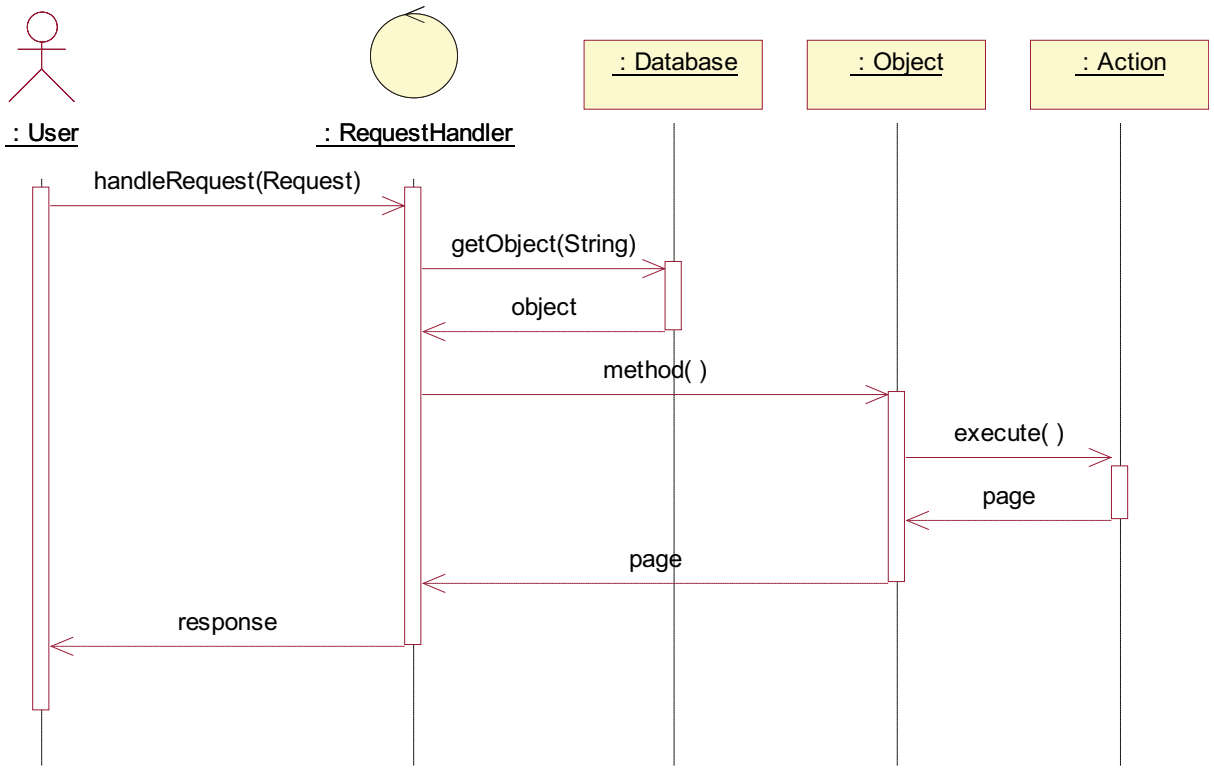


Figure 4.8: Sequence diagram for the RequestObjectAction use case.

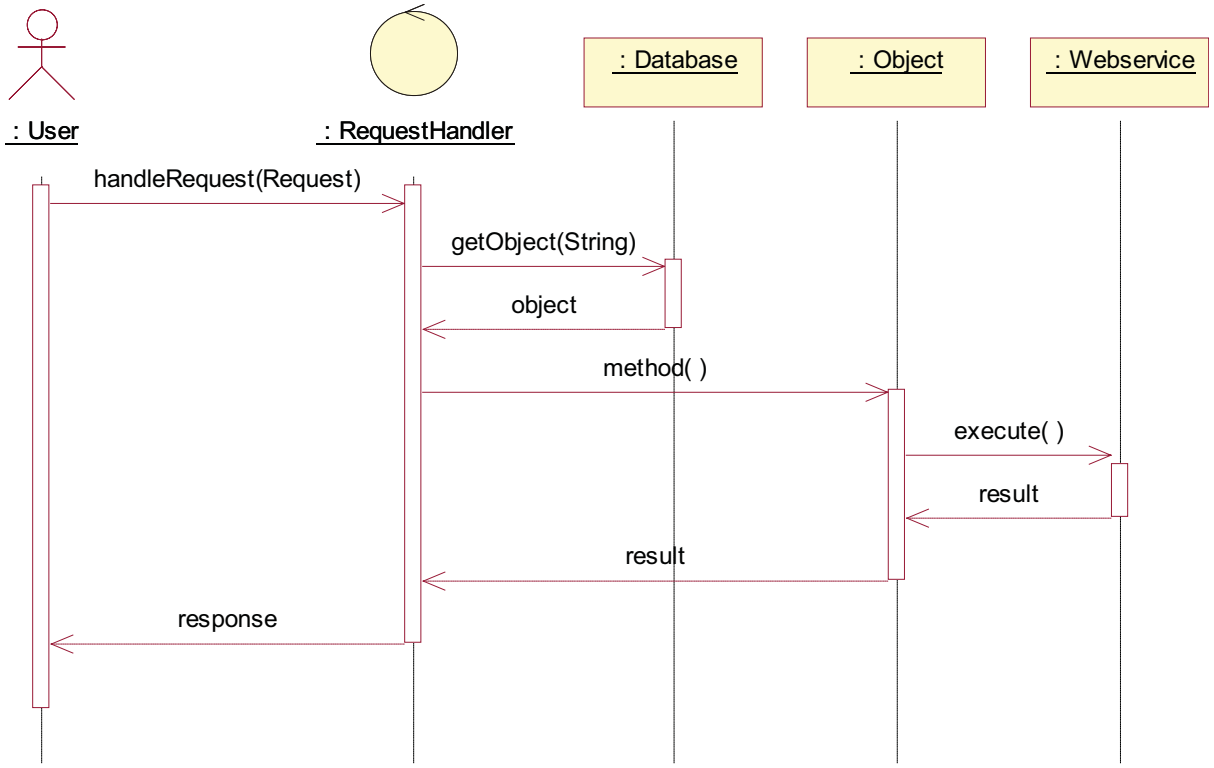


Figure 4.9: Sequence diagram for the RequestObjectWebservice use case.

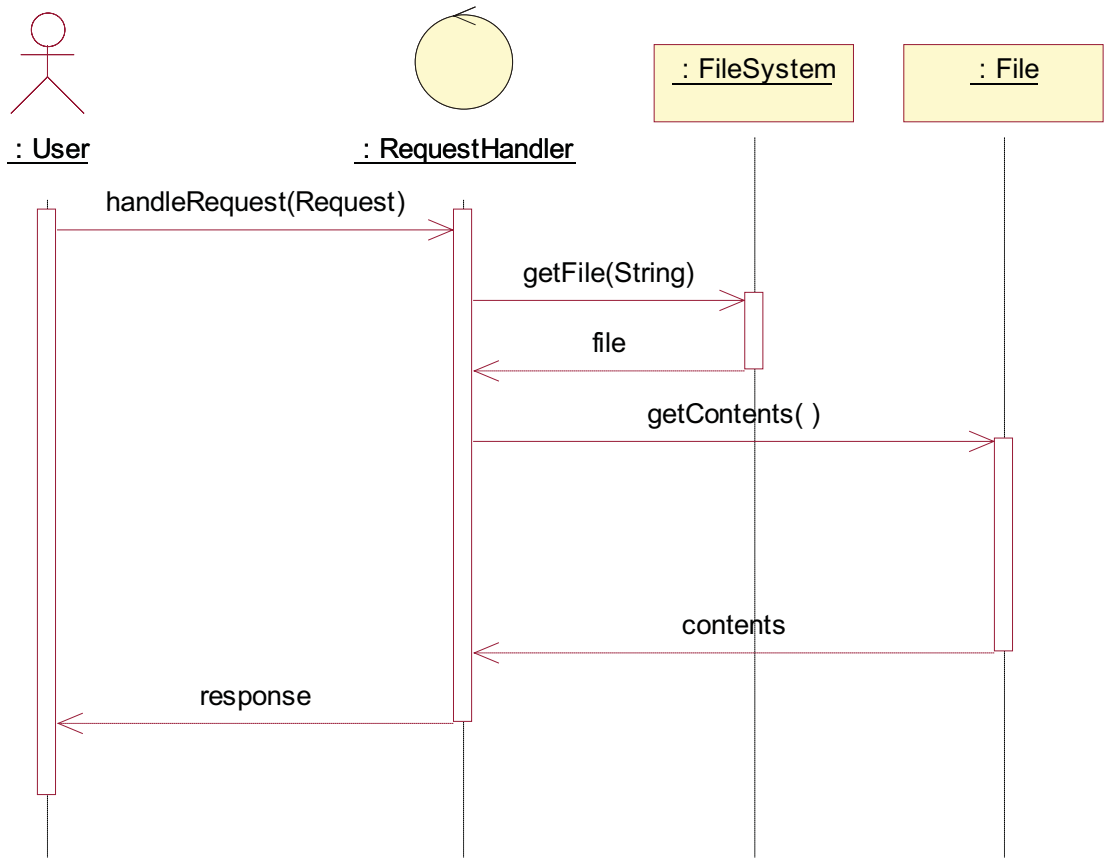


Figure 4.10: Sequence diagram for the RequestFile use case.

Appendix E: System Design Document

CMS 3
System Design Document

Berend Wouda Thomas de Ruiter

June 15, 2007

Preface

This document is the *System Design Document* for the *CMS3* project for *7U Digital Handmade Originals*, done for the *Delft University of Technology 'IN3700 BSc-Project'* by *Thomas de Ruiter* and *Berend Wouda*.

This document contains the overall design for the project. It contains

- (parts of) the overall design of the system to be replaced and its issues,
- the made design decisions for the implementation of the new system,
- the overall decomposition of the new system, and
- an initial specification of the interfaces of the subsystems.

This is a cyclic deliverable, so this version may not yet be complete.

Each version is to be agreed to by the client. The final version serves as an agreement over the design of the new system, and states the decisions made for implementation.

Version

1 June 2007

Initial version. Lacks the Access Control and Security sections, and the Amy Peanut database schema.

6 June 2007

A number of spelling errors have been fixed, and some sentences have been reworded. The section referral has been made more explicit.

15 June 2007

The Amy instance and module layouts have been slightly adapted to reflect the new SVN structure. The Kif module layout has also been slightly adapted for the same reason.

Contents

List of Figures

1 Introduction

1.1 Purpose of the system

The system allows the easy creation of a CMS driven site, using modules and templates. The system provides the basic functionality required to run a site, and modules may provide more domain specific functionality.

The design of the system is important because the modules need to be easy to develop, and the developers of the modules are subject to (some of) the subsystem interfaces of the system.

1.2 Overview of the system

The system consists of two major subsystems, called 'Amy' and 'Kif'. These two Futurama[1] characters are a couple in the series. Amy is a framework for making professional web applications, which is available as a layer subsystem for other subsystems. Kif is the CMS subsystem running on the Amy framework layer.

Next to these two main subsystems, there may be more (small) subsystems, which are called *modules*. Any 3rd party software exists out of this subsystem structure, being encapsulated by subsystem modules instead.

In chapter 3 Amy is discussed, and in chapter 4 Kif is discussed. For each system an overview, the design goals, the architecture, and the design decisions are given.

1.3 Definitions, acronyms, and abbreviations

CMS3 The shorthand form for *7U Content Management System* version 3, the full name of the system.

Base, Basis, Core The form of the system in that it itself does not do much specifically, but instead provides the generic loading and execution of extensions that do. These words may be used interchangeably.

Module, Extension A piece of software that 'plugs in' the base system and provides additional, specific functionality. The two words may be used interchangeably.

EPP Enterprise PHP Peanuts, a system comparable to Enterprise Java Beans, but for PHP.

1.4 References

Requirements Analysis Document The RAD details the functionality and initial design of the system. It can be found at: `https://pavlov/trac/cms3/wiki/RequirementsAnalysisDocument`

[1] An animated series by Matt Groening, for more information see `http://en.wikipedia.org/wiki/Futurama`.

2 Current system

Please note that sections 2.1 through 2.4 describe only the parts of the system that are interesting for the design of the new system. The current system contains a lot more aspects, but these are not wanted in the new system (See the Requirements Analysis Document).

2.1 Functionality

The current system's basic functionality is the same as proposed for the new system. Requests come in and are handled according to type, which are:

- Static action
- Static webservice
- Dynamic action
- Dynamic webservice

2.2 Architecture

Due to this the basic architecture is similar, and in fact similar to a lot of webservers. There is a subsystem that handles the incoming requests, and dispatches them to the right target. These targets (classes and objects of those classes) are partitioned into two subsystems ('cms' and 'extentions'), which correspond to core and module functionality, respectively. Inside these subsystems a class (for database objects) and its resources are placed in their own subsystem.

The dispatching subsystem is also part of the 'cms' subsystem.

2.3 Design

The current system is supposed to adhere to the following design goals:

- **Availability:** As long as it is running, the system will always show the user a result of his action(s), no matter what happened internally.
- **Security:** The system should not compromise the integrity of the data.
- **Extensibility:** It should be able to add new functionality to the system.
- **Adaptability:** The system should be able to support any sort of site.
- **Backwards Compatibility:** Added functionality should not break the old behaviour of the system.

In particular, the system does *not* adhere to the following design goals:

- **Modifiability:** The system is not originally meant for many modifications, even though they were made.
- **Readability:** The system is not easily understandable from the implementation (anymore).
- **Traceability:** The system's implementation is not easily mappable to currently known functionality (anymore).

2.4 Issues

As can be read in the RAD, the current system suffers from problems mostly pertaining to the need of backwards compatibility, while suffering from a lack of proper modifiability. This has had large effects on the readability and traceability of the code.

The current (lack of) architecture also causes the same effects on readability and traceability. All in all the ideas are good, but the design and implementation are not sufficient anymore.

3 Proposed system: Amy

3.1 Overview

Amy is a bottom layer subsystem that defines a set of standards for the design and implementation of any webserving system. In design, Amy is comparable to (a small subset of) the Java Runtime Environment.

Sections 3.1.1 through 3.1.4 list the abstractions it provides in the form of subsystems.

3.1.1 Database abstraction

A database abstraction layer allows non-specific access to an underlying database. Amy will provide at least a MySQL implementation for the database abstraction layer, although PostgreSQL and SQLite are on the wishlist.

3.1.2 Model-View-Controller abstraction

Amy provides support for usage of the Model-View-Controller (MVC) design pattern, through the abstractions in sections 3.1.2.1 through 3.1.2.3.

3.1.2.1 Enterprise PHP Peanuts

Enterprise PHP Peanuts (EPP) basically is a persistent object management subsystem akin to Enterprise Java Beans (EJB). It provides a way of storing and accessing objects (instances of classes) in a database. These objects are affectionally called *peanuts*, akin to Java's *beans*.

Peanuts are managed by PeanutContainers, which Amy implements a basis for. Peanuts themselves use a special kind of object attribute called DataTypes, which have defined database equivalents and can manage themselves in that regard. Standard peanut behaviour is also provided, and can be extended.

Peanuts represent the 'Model' in the MVC architecture.

3.1.2.2 Templates

A template engine management subsystem allows the encapsulation of template engines, making them adhere to an interface. An encapsulated template engine is automatically called if a template is encountered that has one of the file extensions supported by that engine.

Templates represent the 'View' in the MVC architecture.

3.1.2.3 Servlets

A central servlet management subsystem allows easy specification of servlets. Servlets are classes that take requests as input and give responses as output. The servlet system supplies generic request/response functionality which can be used by implemented servlets to interact with the requester. Servlets can also dispatch requests to other servlets.

Servlets are supposed to have only one instance, and no state. The state for a transaction can be stored in a ServletContext.

Servlets represent the 'Controller' in the MVC architecture.

3.1.3 Organization

Amy uses modules in order to organize functionality. Inside modules, classes and resources are organised in packages.

The core subsystem provides support for loading modules, and for loading classes and resources from packages. The loading of classes in particular can be done manually, or through PHP's `__autoload` function.

3.1.3.1 Packages

Packages are named folders of some sort, that may contain other packages, classes, and resources.

For packages, the *fully qualified name* is a sequence of package names, according to the package hierachy. The package names are separated by the underscore: '_'.

For classes, the fully qualified name is the aforementioned sequence of parent package names, with the class name appended (again seperated by an underscore). Classes also use their fully qualified names as their classname within PHP. This is to prevent name conflicts.

For resources, the fully qualified name is the aforementioned sequence of parent package names, with the resource path within that package appended (seperated by slashes: '/').

For example, Object's PHP classname is `amy_lang_Object`. It's filename is `amy/lang/Object.php`.

3.1.3.2 Modules

Modules are (small) subsystems that together define most of the (application specific) functionality. Each module provides a bit of functionality.

A module consists of the following base structure:

- `<module_name>`
  - `src`: The base of a tree with packages (folders) and classes/resources (files), made up according to Amy's package structure.
  - `doc`: The documentation of the module (optional).
  - `test`: The module's test set (optional).

Amy supports normal filesystem folder trees and ZIP compressed files containing folder trees.

Each module normally defines a set of classes that work after the Model-View-Controller architecture mentioned above. A module would then define one or more peanuts, each with one or more servlets and a set of templates. The peanut defines the model, and the servlets are the controllers that accept the different types of requests. The templates are used to generate the view.

Modules may override each other according to a predefined order (set in the configuration file(s)). This allows the easy replacement of objects or templates.

What modules may provide is described in the following paragraphs.

Database implementations Modules may provide implementations to support other database systems. These implementations have to adhere to the abstraction layer interface. For example, an SQLite implementation.

Peanuts and PeanutContainers Modules may define new types of objects that need persistent storage. These objects may for example be business objects in the application domain.

DataTypes Modules may define new types of self-managing object attributes. These attribute types can then be used in peanuts in that module or in other modules.

Template engines, templates and template resources Modules may include files that define the layout of output data. These can be Smarty templates for example.

Modules may also add more template engines. Modules can then use templates for that engine.

Servlets and ServletContainers Modules may provide request handlers. For example, a handler to manage a certain peanut type.

Configuration files Modules should define a config.ini file that defines settings for that module. This config file overrides the standard configuration (but only for the module in question).

Miscellaneous files Modules may also contain auxiliary PHP or other files that are used by things in the module.

3.1.4 Utility classes

Amy also supplies a number of utility classes:

3.1.4.1 Input/Output

This subsystem providea a generic stream implementation, and stream implementations for files and ZIP compressed files. This package is used by the core to load resources.

3.1.4.2 Network

This subsystem provides network related classes, such as an URL class.

3.1.4.3 Miscellaneous

This subsystem provides some uncategorized utility classes:

- Config: A class for easily loading configuration files (.ini files).
- Logger: A static logging service, that writes everything passed to it to a defined output.
- MimeTypes: A class that provides the correct mimetype for most popular file extensions.

Note that this subsystem is actually called the utility classes subsystem. This is because it is the only one with a collection of unrelated classes.

3.2 Design goals

The following are the design goals for Amy. They have mostly been taken from the non-functional and pseudo requirements listed in the RAD. Experiences with the current system and some developer wishes also had an influence.

3.2.1 Dependability

- **Fault tolerance:** The system should not fail on errors, but provide exceptions that can be caught (generalization of non-functional requirements **NFR7**, **NFR12** and **NFR13**).
- **Security:** The system should not be vulnerable to the most common web attacks (generalization of non-functional requirements **NFR5** and **NFR6**).

3.2.2 Maintenance

- **Extensibility:** The system should be extensible through modules (**PR5**).
- **Modifiability:** The parts of the system that have an interface defined should be modifiable by using modules to instate or replace the standard behaviour (generalization of pseudo requirements **PR11** through **PR18**).
- **Portability:** The system should run on any platform that supports PHP5 (generalization of **PR1**).
- **Readability:** The system's code should be clear and concise (taken from the current system's issues).

3.2.3 End user criteria

- **Utility:** The system should provide a logical interface for developers (developer wish).
- **Usability:** The system should allow easy module development for developers (developer wish).

3.3 Architecture

Sections 3.3.1 and 3.3.2 describe the architecture of the system, in terms of subsystems. These subsystems are connected with one another, and are placed upon a deployment platform.

3.3.1 Subsystem decomposition

Figure 3.1 shows the Amy system with its subsystems, and how they are related and connected.

A description of the subsystems has been given in 3.1. The Amy subsystems do not have many interconnections, since they provide a layer interface of separate components. The peanut subsystem accesses the database layer, and the core subsystem is responsible for loading the rest of the framework.

The subsystems' interfaces are given in the section on subsystem services (section 5).

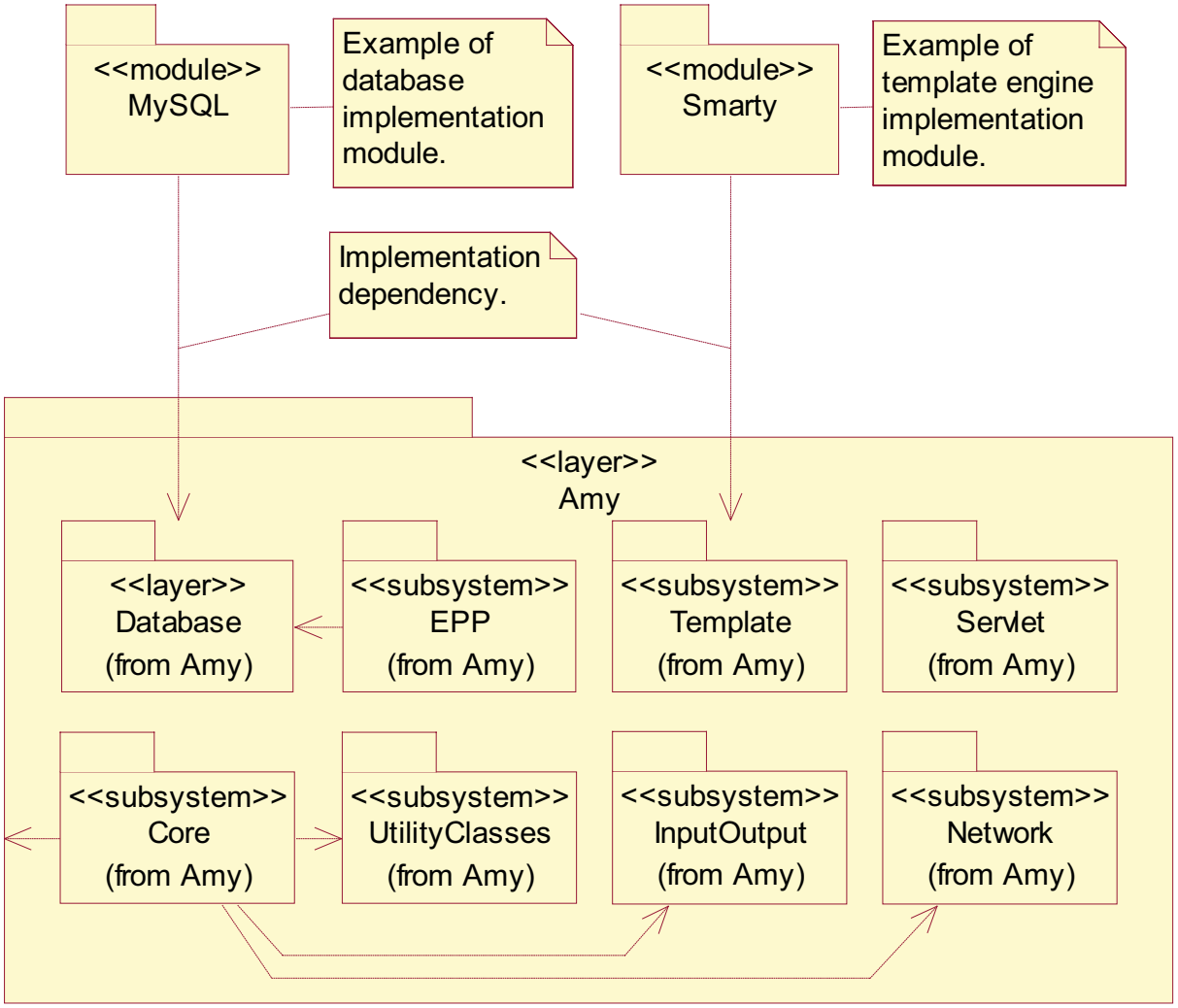


Figure 3.1: Amy Subsystem Decomposition

3.3.2 Subsystem mapping

Figure 3.2 shows how the Amy subsystems are mapped to hardware systems and (off-the-shelf) software components.

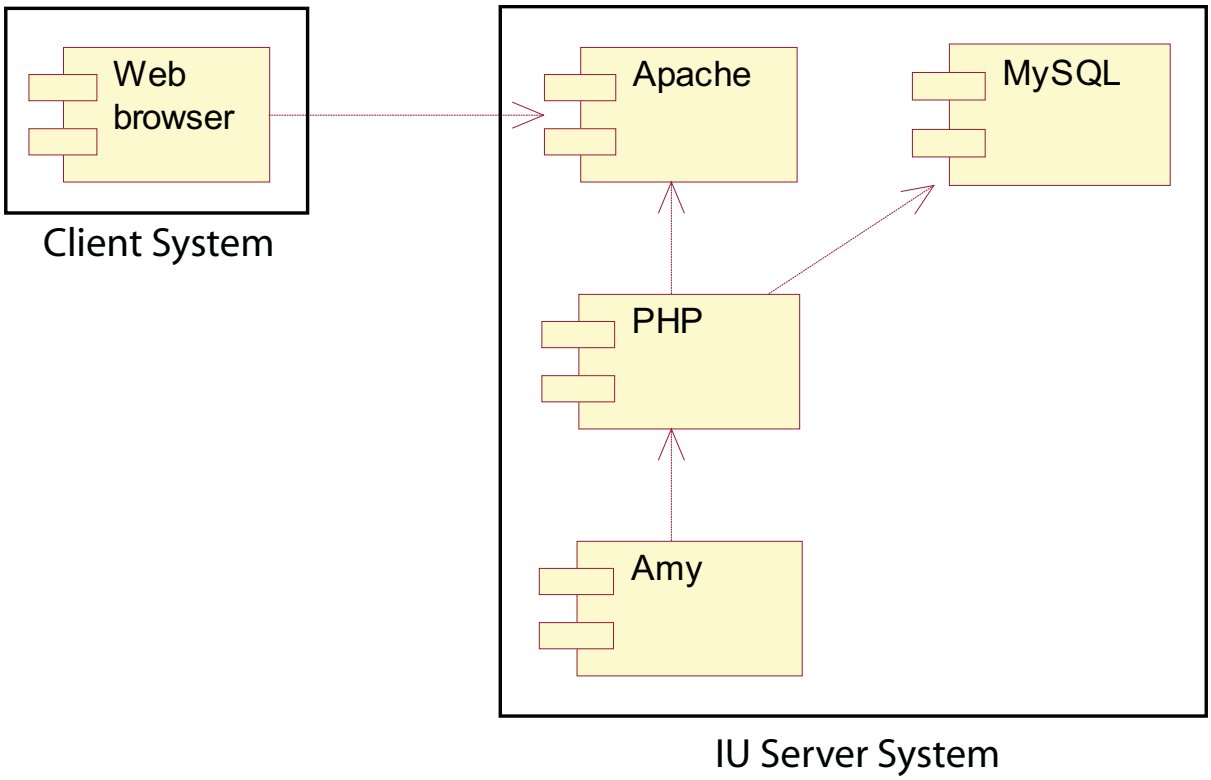


Figure 3.2: Amy Subsystem Mapping

The system's main deployment is on the IU server[1]. On this server, the bridge from software to hardware is gapped by a Linux operating system, on which Apache with PHP 5.2.1 is running. Amy runs on Apache/PHP. The system also runs a MySQL 5.0.37 database server, that Amy interacts with through PHP. Amy's files are stored directly on the Linux system.

Amy's file organisation is freely configurable in the build file properties. Modules can be placed anywhere, as can 3rd party software, and they can be used by multiple applications. However, for developers the following standard is recommended:

- Instance root.
  - `modules`: Storage of the used modules.
  - `store`: Storage of (flat) files that must persist.
  - `temp`: Storage of files that may be deleted at any time.
  - `vendors`: Storage of 3rd party software used by the modules.
  - `www`: Web root.
    - `index.php`: CMS entry point.
  - `config.ini`: Configuration file.

Note the placement of 'index.php'. This file is a part of Amy, and is required to run Amy.

Amy itself consists of a number of modules. Its core module looks like the following:

- `amy`: Module name.

[1] 7U's colocated server.

- `doc`: Documentation folder.
- `src`: Code folder.
  - `amy`: Amy package.
    - `db`: Database interface package.
    - `epp`: Enterprise PHP Peanuts package.
    - `io`: (File) input/output package.
    - `lang`: PHP language extension package.
    - `net`: Networking package.
    - `servlet`: Servlet interface and base functionality package.
    - `template`: Template engine interface and base functionality package.
    - `util`: Utility package.
- `test`: Test set folder.

There are also database implementation modules.

Although the main deployment is on Linux with Apache, the system should also be able to run on other webserver systems that support PHP 5.2.1 and filesystems, such as Windows with Apache or IIS.

The client side consists of a webbrowser.

3.4 Design decisions

The following are the decisions made for the global design of the system.

3.4.1 Persistent data management

There is a database abstraction layer provided by Amy. It is used for object persistence, and has a schema for that purpose. It can also be used directly.

3.4.1.1 Database abstraction

Amy has a database abstraction layer subsystem that provides universal access to an underlying database. This provides an interface for persistent data management.

The interface provides for the standard relational-based approach, providing resultsets for queries performed on tables of data. The underlaying database may be a relational database (using a one-to-one mapping) or some other way of storing data.

The interface is given in the section on subsystem services (section 5).

3.4.1.2 Object persistence

Amy also supplies the basis for object persistence in the form of 'peanuts'. Peanuts are objects that use 'fieldtypes' for their properties, and as such can easily be serialized into database entries.

An object that is a peanut must use fieldtypes for the properties that have to be stored in the database, and must extend the peanut class in Amy. A peanut must also have at least a name, and (optionally) a parent. Objects with no parents are root objects.

An instance of the PeanutContainer class allows access to objects in two ways: by a globally unique identifier, or through their (absolute) paths. Such a path is the sequence of parent names followed by a peanut's name, each separated by a slash: '/'. The PeanutContainer also takes care of storing the objects in the database automatically.

3.4.1.3 Database schema

The database has a defined layout for object serialization, which is given in table/figure/appendix...

3.4.2 Access control

...

3.4.3 Security

...

3.4.4 Boundary conditions

Because PHP is started every time a request is done, the boundary conditions are in fact part of the global software control. However, they are listed here separately, because the startup behaviour is still distinct.

The following steps are taken when Amy is started:

1. The HTTP server receives a request and starts up a PHP session.
2. Amy's index.php is run.
3. The index.php script instantiates the BootClassLoader class (which resides in the index.php file) into a single BootClassLoader object.
4. The BootClassLoader object loads all the Amy core classes into memory.
5. The index.php script instantiates the (now available) class loading related classes:
   - Class and resource manager classes.
   - (File) input classes.
   - The Object class.
6. The index.php script loads the root configuration file.
7. The index.php script calls the main method on the ServletContainer class.

The following steps are taken when Amy's execution ends:

1. The PHP file unloads all the loaded classes.
2. The HTTP server ends the PHP session and sends out (the remainder of) the response.

Amy uses exceptions for error behaviour, and encapsulates all the standard PHP errors. If an exception occurs, the ServletContainer catches it and it responds to the requester with an appropriate HTTP error.

3.4.5 Global software control

After Amy is loaded, the following steps are taken:

1. The ServletContainer class instantiates an object of itself.
2. The ServletContainer retrieves the class names of the request type and response type from the configuration file, and instantiates them.
3. The Request reads in the request data through the use of PHP's functions.
4. The ServletContainer retrieves the class name of the main servlet from the configuration file, instantiates it, and calls the servicing method with Request and Response as parameters.
5. The Servlet does its thing.
6. The ServletContainer has the Response output the response data to the requester through the use of PHP's functions.

4 Proposed system: Kif

4.1 Overview

Kif is the CMS3 implementation. It runs modules on the Amy layer, using the abstraction subsystems and utility classes, while adhering to the organisational requirements. It therefore provides some of its own peanuts, templates and servlets, made after the Amy architecture.

4.1.1 Kif objects

Kif runs its own higher level object abstraction. These notions of objects are not directly related to the PHP implementation of objects and classes, but to the general object oriented idea. The objects do not really exist as single entities, but make use of peanuts, templates and servlets. These *kif objects* are used for easily representing application domain entities for webservicing, while maintaining a developer oriented separation in the implementation.

4.1.1.1 High level

Kif objects are dynamic entities that have a state (which is saved in a database). Their functionality can act upon and change this state, which makes them perfect for representing things in the application domain.

However, kif objects may also specify static functionality, which is useful for editing *other* things, and performing tasks that require no state. This in turn is perfect for doing things not grounded in, but working on the application domain, such as controller tasks.

A kif object may also specify only static functionality. Such a kif object has no state and is called *static*. To differentiate, normal kif objects may also be called *dynamic*.

4.1.1.2 Low level

Kif objects actually consist of a number of files that together define the high level object. These files include:

- A peanut class.
- Templates.
- Template resources.
- Servlet classes:
  - A main object class.
  - Functionality class(es).
- Any auxiliary PHP classes used by the above.

Peanuts KifPeanuts define the model of a kif object. They extend KifPeanut, and implement the Peanut interface from Amy, so they can take care of a kif object's persistent properties.

The state of kif objects is stored in instances of their peanut classes, which are stored in a database through standard peanut serialization. Hence, static kif objects do not have associated peanuts.

Templates Templates define the view of a kif object. They are some form of layout that is driven by a template engine so that data can dynamically be inserted.

Template resources are files needed for the template to work, including but not limited to:

- CSS: Stylesheet information.
- Javascript: Client side script functionality.
- Images: Template graphics.

Templates are not needed for kif objects that only provide webservices.

Servlets The main entry point to a kif object is an extension of the KifObject class (which is an extension of the Amy Servlet class). It dispatches incoming requests to extensions of the KifActions class and KifWebservices class (which are both extensions of Servlet as well) which together implement the functionality and interface of the kif object. These classes define the controller for a kif object.

There are two types of functionality, which differ in response.

- Actions: Functionality that produces output directly useful for a user.
- Webservices: Functionality that produces output in the form of data. This output is not meant to be interpretable by a user.

These types can be implemented for static as well as dynamic requests, making for a total of four different request handling servlets. These four servlets make up the full interface of the object.

If more separation is required, the servlets may themselves dispatch to other servlets as well.

4.1.2 Main servlet

Kif provides the main servlet for the CMS. It is the entry point servlet, and it parses and dispatches incoming requests to three subservlets that take care of the first step of specialization. These four servlets have only one instance throughout a session.

The request types are the ones from the functional requirements **FR1** to **FR5** listed in the functionality section of the RAD. The main servlet implements the initial RequestHandler object (see the RAD), and with that all the use cases that were found.

The exact process of handling requests is listed in the section on global software control (section 4.4.5). For the names and locations of all the servlets see the section on subsystem mapping (section 3.3.2).

4.1.3 Organization

Kif consists of a number of modules. The main Kif module contains the main servlets and all the interfaces used by the other Kif modules. For the names and locations of all the servlets see the section on subsystem mapping (section 3.3.2).

Kif objects are defined in modules separate to Kif's main module. In such a module all the files needed for the kif object (which have been listed in 4.1.1.2) are contained. These modules can be added easily, providing more types of kif objects for the CMS to work with. Static functionality can be added by adding modules with static kif objects.

4.2 Design goals

The following are the design goals for Kif. They have mostly been taken from the non-functional and pseudo requirements listed in the RAD. Experiences with the current system and some developer wishes also had an influence.

4.2.1 Performance criteria

- **Response time:** The system should respond to a request within 10 seconds (**NFR8**).
- **Throughput:** The system should be able to accept at least 10 requests per second (**NFR9**).

4.2.2 Dependability

- **Robustness:** The system should not fail on invalid user input (taken from the current system).
- **Fault tolerance:** The system should not fail on errors, but instead inform the user (generalization of non-functional requirements **NFR7**, **NFR12** and **NFR13**).
- **Security:** The system should not be vulnerable to the most common web attacks (generalization of non-functional requirements **NFR5** and **NFR6**).

4.2.3 Cost

- **Extension cost:** The cost of developing modules should be minimal.

4.2.4 Maintenance

- **Adaptability:** The system should support any application domain (taken from the current system, and the generality of the functional requirements **FR1** through **FR5**).
- **Readability:** The system's code should be clear and concise (taken from the current system's issues).
- **Traceability:** The system should implement only the requirements (taken from the current system's issues).

4.2.5 End user criteria

- **Utility:** The system should provide a logical interface for developers (developer wish).
- **Usability:** The system should minimize kifclass/kifobject development work for developers (developer wish).

4.3 Architecture

Sections 4.3.1 and 4.3.2 describe the architecture of the system, in terms of subsystems. These subsystems are connected with one another, and are placed upon a deployment platform.

4.3.1 Subsystem decomposition

Figure 4.1 shows the Kif system with its subsystems, how they are related and connected, and how they are related and connected to the Amy layer. Note that the Amy layer has been reduced for clarity.

The subsystems' interfaces are given in the section on subsystem services (section 5).

4.3.2 Subsystem mapping

Figure 4.2 shows how the Kif subsystems are mapped to hardware systems, (off-the-shelf) software components, and Amy.[1]

The system's main deployment is on the IU server[2]. On this server, the bridge from software to hardware is gapped by the Amy abstraction layer software running on PHP 5.2.1 on Apache and MySQL 5.0.37, on Linux. Kif's files are stored directly on the Linux system.

Kif's core module looks like the following:

- `kif`: Module name.
  - `doc`: Documentation folder.

[1] Please excuse the lack of standard UML deployment diagram graphics, Rational Rose does not support it very well.
[2] 7U's colocated server.

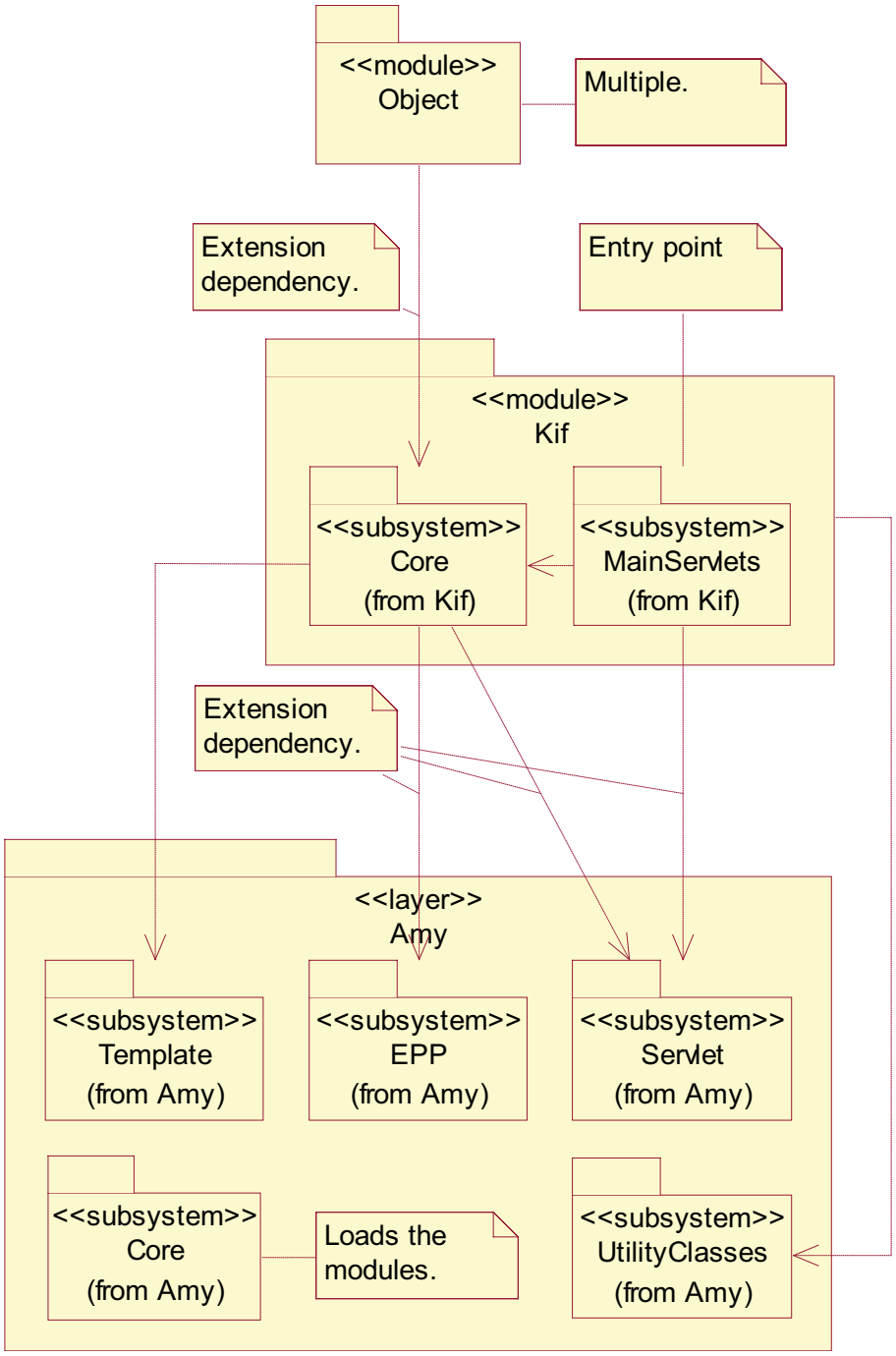


Figure 4.1: Kif Subsystem Decomposition

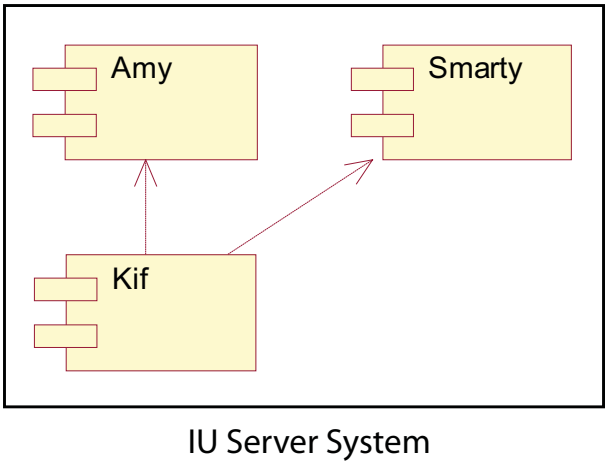


Figure 4.2: Kif Subsystem Mapping

- `src`: Code folder.
  - `kif`: Kif package
    - `Servlet.php`: Main servlet class.
    - `DynamicServlet.php`: Servlet class that handles dynamic functionality requests.
    - `StaticServlet.php`: Servlet class that handles static functionality requests.
    - `ResourceServlet.php`: Servlet class that handles resource (file) requests.
    - `Request.php`: Request class for receiving requests in the Kif system.
    - `Response.php`: Response class for sending responses from the Kif system.
    - `Object.php`: Abstract base servlet class that can handle action and webservice requests. This servlet class acts as entry point for a kif object.
    - `Peanut.php`: Base class for peanuts in the Kif system.
    - `Actions.php`: Abstract base servlet class that can handle action requests.
    - `Webservices.php`: Abstract base servlet class that can handle webservice requests.
    - ...
- `test`: Test set folder.
- `build.xml`: Phing build file.

Although the main deployment is fixed, the system should also be able to run on other systems that support Amy.

The client side consists of a (standards-compliant, or Microsoft) webbrowser that optionally supports Javascript (for the use of webservices).

4.4 Design decisions

The following are the decisions made for the global design of the system.

4.4.1 Persistent data management

Kif uses Amy's peanuts for persistent data storage. A kif object partly consists of a KifPeanut, which is its model component. The peanut defines the kif object's persistent properties, and has all the standard persistence behaviour.

4.4.2 Access control

...

4.4.3 Security

...

4.4.4 Boundary conditions

Because PHP is started every time a request is done, the boundary conditions are in fact part of the global software control. However, they are listed here separately, because the startup behaviour is still distinct.

The following steps are taken when Kif is started:

1. Amy is started.

The following steps are taken when Kif's execution ends:

1. Amy's execution ends.

Kif uses Amy's exception mechanism.
These things show that Kif is completely embedded in the Amy framework.

4.4.5 Global software control

After Kif's main servlet's service method is called, the following steps are taken:

1. The main servlet examines the request and dispatches the request to the static, dynamic, or resource servlet (depending on the type of request).
2. In case it was dispatched to the static servlet:
   a) The servlet dispatches the request and an empty context to the indicated kif object's entry point.
   b) That object invokes the requested action or webservice function (depending on the type of request).
   c) That function sets the response object with its result.
3. In case it was dispatched to the dynamic servlet:
   a) The servlet retrieves the peanut that belongs to the indicated kif object.
   b) The servlet dispatches the request and a context containing the peanut to the indicated kif object's entry point.
   c) That object invokes the requested action or webservice function (depending on the type of request).
   d) That function sets the response object with its result.
4. In case it was dispatched to the resource servlet:
   a) The servlet retrieves the requested resource.
   b) The servlet sets the response with the contents of the resource.

These steps are analogous to the use cases/sequence diagrams, albeit slightly different in approach (it's recursive instead of iterative).

5 Subsystem services

...

Appendix F: Object Design Document

CMS 3
Object Design Document

Berend Wouda Thomas de Ruiter

July 9, 2007

Preface

This document is the *Object Design Document* for the *CMS3* project for *7U Digital Handmade Originals*, done for the *Delft University of Technology 'IN3700 BSc-Project'* by *Thomas de Ruiter* and *Berend Wouda*.

This document contains the detailed design for the project. It contains (for both systems):

- An overview of the naming scheme and coding standard that should be used.
- An overview of the trade-offs made in the detailed design.
- A description of the packages, their hierarchies, and dependencies.
- A description of the classes, their interfaces, and dependencies.
- An attachment containing the precise API documentation.

This is a cyclic deliverable, so this version may not yet be complete.

Each version is to be agreed to by the client. The final version serves as an agreement over the design of the new system to be implemented.

Version

12 June 2007

Initial version.

14 June 2007

A spelling error has been fixed and an ambigious sentence has been reworded ("increases response time" instead of "reduces response time"'). The 'Packages' sections have been rewritten to refer to the SDD instead of the phpDoc, which does not contain package information (the SDD's subsystem description corresponds directly to the package layout).

9 July 2007

The complete API has been generated and included.

Contents

1 Introduction

1.1 Purpose of the system

The system allows the easy creation of a CMS driven site, using modules and templates. The system provides the basic functionality required to run a site, and modules may provide more domain specific functionality.

The detailed design of the system is important because the modules need to be easy to develop, and the developers of the modules are subject to (some of) the class interfaces of the system.

1.2 Overview of the system

The system consists of two major class collections, called 'Amy' and 'Kif'. These two Futurama[1] characters are a couple in the series. Amy is a framework for making professional web applications. Kif is the CMS running on the Amy framework.

Next to these there may be modules, which may provide extra classes. Any 3rd party software is also encapsulated in module classes.

In chapter 2 Amy is discussed, and in chapter 3 Kif is discussed. For each system an overview, the packages, and the classes are given.

1.3 Naming scheme

Amy (and therefore Kif) follows a naming scheme which is described in this project's Coding Standard.

This scheme includes:

- Package names describe the functionality of the package with one descriptive word.
- Class names are nouns describing the class' role in the application.

For more and complete information, see the Coding Standard.

1.4 Coding standard

Amy (and therefore Kif) follows a strict coding standard that is based on the PHP Coding Standard and some parts of the Java Coding Standard. This coding standard is described along with the naming schemes in the Coding Standard document, which lists errata on the official PHP Coding Standard to describe this project's coding standard.

These errata include:

- The coding standard enforces naming classes in PHP in such a way that packages are supported. This is done by prepending all the packages to the PHP class name, separated by underscores.
- Package names are all lowercase and class names are in CamelCase. Method names are CamelCase starting with a lowercase letter instead, and variables are all lowercase with underscores as separators.
- No special letter prefixes are to be used.
- Global variables are also not to be used.

[1] An animated series by Matt Groening, for more information see `http://en.wikipedia.org/wiki/Futurama`.

- Errors should be indicated by throwing exceptions, instead of indication by return value, error functions, or printing.

For more and complete information, see the Coding Standard.

1.5 Definitions, acronyms, and abbreviations

CMS3 The shorthand form for *7U Content Management System* version 3, the full name of the system.

Base, Basis, Core The form of the system in that it itself does not do much specifically, but instead provides the generic loading and execution of extensions that do. These words may be used interchangeably.

Module, Extension A piece of software that 'plugs in' the base system and provides additional, specific functionality. The two words may be used interchangeably.

EPP Enterprise PHP Peanuts, a system comparable to Enterprise Java Beans, but for PHP.

1.6 References

Coding Standard The CS describes the coding standard and the naming scheme for the system in terms of the PHP Coding Standard with errata. It can be found at: `https://pavlov/trac/cms3/wiki/CodingStandard`

Requirements Analysis Document The RAD details the functionality and initial design of the system. It can be found at: `https://pavlov/trac/cms3/wiki/RequirementsAnalysisDocument`

System Design Document The SDD gives the global design of the system and the decisions made for the implementation of the system. It can be found at: `https://pavlov/trac/cms3/wiki/SystemDesignDocument`

2 Amy

2.1 Overview

Amy's classes form the basis on which a web application is built.

2.1.1 Trade-offs

Amy's design incorporates a few trade-offs, which are listed in sections 2.1.1.1 through 2.1.1.3. The trade-offs are derived from the design goals and design decisions.

2.1.1.1 Security

Amy incorporates various security aspects to prevent against common web attacks. This requires more processing, so it increases the response time to requests.

2.1.1.2 Organization

Amy is built using highly organized and clean code. This enables a tremendous amount of readability, utility, and useability. It also adds to the extensibility, modifiability, portablity, and fault tolerance.

It requires more unoptimized, complex code however, which has a somewhat negative impact on the response time.

2.1.1.3 Caching

In order to improve the (now lowered) response speed of applications running on Amy it uses various levels of caching facilities. Peanuts are generally cached, as are servlets. Resources such as classes and templates are also cached. The trade-off is that this uses more memory per session.

2.2 Packages

Amy's package structure is described in the SDD (section 3.3.2). Amy resides in the `amy` package, and each of Amy's subsystems is contained within its own package. These subsystems do not contain any packages of their own.

The package dependency therefore corresponds to the subsystem dependency given in the SDD (section 3.3.1).

2.3 Classes

A specification of Amy's classes, and which classes they rely upon is given in attachment A.

3 Kif

3.1 Overview

Kif is the CMS implementation that makes use of Amy's classes. It either extends them or calls their methods.

3.1.1 Trade-offs

Kif's design incorporates a few trade-offs, which are listed in sections 3.1.1.1 through 3.1.1.2. The trade-offs are derived from the design goals and design decisions.

3.1.1.1 Security

Besides Amy's security measures Kif incorporates security specific to its implementation. This also requires more processing, so it reduces the response time to requests.

3.1.1.2 Organization

Kif is also built using separated and organized code. This enables a high amount of readability (and traceability!), utility, and usability. It also adds to the adaptability, robustness, and fault tolerance. Furthermore, it keeps the cost of module development low.

However, it also has a somewhat negative impact on the response time.

3.2 Packages

Kif's package structure is described in the SDD (section 4.3.2). Kif resides in the `kif` package, and each of Kif's subsystems is contained within its own package. These subsystems do not contain any packages of their own.

The package dependency therefore corresponds to the subsystem dependency given in the SDD (section 4.3.1).

3.3 Classes

A specification of Kif's classes, and which classes they rely upon is given in attachment B.

Attachment A: Amy API Documentation

amy_io_FileInputStream

Description

A FileInputStream obtains input from a file in a file system. What files are available depends on the host environment.

*Located in **/io/FileInputStream.php** (line **24**)*

```
amy_lang_Object
   |
   --amy_io_InputStream
      |
      --amy_io_FileInputStream
```

Direct descendents

Class	Description
amy_io_ZipFileInputStream	A ZipFileInputStream obtains input from a file inside a zip file. What files are available depends on the zip file.

Variable Summary

resource **$ifp**

Method Summary

amy_io_FileInputStream **__construct** (*string* **$name**)
void **close** ()
mixed **read** ([*int* **$length** = 1])
int **skip** (*int* **$length**)

Variables

resource **$ifp** (line 31)

File resource handler.

- **access:** protected

Methods

Constructor __construct (line 43)

Creates a FileInputStream by opening a connection to an actual file, the file named by the path name $name in the filesystem.

If the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a FileNotFoundException is thrown.

- **access:** public

amy_io_FileInputStream **__construct** (*string* **$name**)

- *string* **$name**: Path to the file to be opened.

Redefined in descendants as:

- amy_io_ZipFileInputStream::__construct() : Creates a ZipFileInputStream by opening a connection to an file entry within a zip archive.

close (line 122)

Closes the input stream and releases any system resources associated with the stream.

- **throws:** amy_io_IOException When IO stuff goes wrong.
- **access:** public

void **close** ()

Redefinition of:

amy_io_InputStream::close()
Closes the input stream and releases any system resources associated with the stream.

read (line 75)

Reads some number of bytes from the input stream. This method blocks until input data is available, end of file is detected, or an exception is thrown.

Returns false if no input is available and the end of file is detected.

- **return:** The input read from the stream, or false if there is no input available.
- **throws:** amy_io_IOException When file stuff goes wrong.
- **access:** public

mixed **read** ([*int* **$length** = 1])

- *int* **$length**: Number of bytes to be read from the input stream.

Redefinition of:

amy_io_InputStream::read()
Reads some number of bytes from the input stream. This method blocks until input data is available, end of file is detected, or an exception is thrown.

skip (line 103)

Skips over and discards n bytes of data from this input stream.

The skip method may, for a variety of reasons, end up skipping over some smaller number of

bytes, possibly 0. This may result from any of a number of conditions; reaching end of file before n bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If n is negative, no bytes are skipped.

The skip method of InputStream uses the read method to read $length bytes and then discards this data. Subclasses are encouraged to provide a more efficient implementation of this method.

- **return:** Actual number of bytes skipped.
- **throws:** amy_io_IOException When IO stuff goes wrong.
- **access:** public

int **skip** (*int* **$length**)

- *int* **$length**: Number of bytes to be sipped.

Redefinition of:

amy_io_InputStream::skip()
Skips over and discards n bytes of data from this input stream.

Inherited Methods

Inherited From amy_io_InputStream

- **amy_io_InputStream::close()**
- **amy_io_InputStream::read()**
- **amy_io_InputStream::skip()**

Inherited From amy_lang_Object

- **amy_lang_Object::getClass()**

amy_io_FileNotFoundException

Description

Exception that is thrown when a file could not be found.

*Located in **/io/FileNotFoundException.php** (line **23**)*

```
Exception
   |
   --amy_lang_Exception
      |
      --amy_io_IOException
         |
         --amy_io_FileNotFoundException
```

Variables

Inherited Variables

Inherited from Exception (Internal Class)

- **$code**
- **$file**
- **$line**
- **$message**
- **$string**
- **$trace**

Methods

Inherited Methods

Inherited From Exception (Internal Class)

- **constructor __construct ([$message =], [$code =])**
- **getCode ( )**
- **getFile ( )**
- **getLine ( )**
- **getMessage ( )**
- **getTrace ( )**
- **getTraceAsString ( )**
- **__clone ( )**
- **__toString ( )**

amy_io_InputStream

Description

Description | Descendents | Methods (details)

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of InputStream must always provide a method that returns the next byte of input.

- **abstract:**

*Located in **/io/InputStream.php** (line **27**)*

```
amy_lang_Object
    |
    --amy_io_InputStream
```

Direct descendents

Description | *Descendents* | Methods (details)

Class	Description
amy_io_FileInputStream	A FileInputStream obtains input from a file in a file system. What files are available depends on the host environment.

Method Summary

Description | Descendants | *Methods* (details)

void **close** ()
mixed **read** ([*int* **$length** = 1])
int **skip** (*int* **$length**)

Methods

Description | Descendents | Methods (*details*)

close (line 77)

Closes the input stream and releases any system resources associated with the stream.

- **throws:** amy_io_IOException When IO stuff goes wrong.
- **access:** public

void **close** ()

Redefined in descendants as:

- amy_io_FileInputStream::close() : Closes the input stream and releases any system resources associated with the stream.

read (line 40)

Reads some number of bytes from the input stream. This method blocks until input

data is available, end of file is detected, or an exception is thrown.

Returns false if no input is available and the end of file is detected.

- **return:** The input read from the stream, or false if there is no input available.
- **throws:** amy_io_IOException When IO stuff goes wrong.
- **access:** public

mixed **read** ([*int* **$length** = 1])

- *int* **$length**: Number of bytes to be read from the input stream.

Redefined in descendants as:

- amy_io_FileInputStream::read() : Reads some number of bytes from the input stream. This method blocks until input data is available, end of file is detected, or an exception is thrown.

skip (line 62)

Skips over and discards n bytes of data from this input stream.

The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly 0. This may result from any of a number of conditions; reaching end of file before n bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If n is negative, no bytes are skipped.

The skip method of InputStream uses the read method to read $length bytes and then discards this data. Subclasses are encouraged to provide a more efficient implementation of this method.

- **return:** Actual number of bytes skipped.
- **throws:** amy_io_IOException When IO stuff goes wrong.
- **access:** public

int **skip** (*int* **$length**)

- *int* **$length**: Number of bytes to be sipped.

Redefined in descendants as:

- amy_io_FileInputStream::skip() : Skips over and discards n bytes of data from this input stream.

Inherited Methods

Inherited From amy_lang_Object

amy_lang_Object::getClass()

amy_io_IOException

Description

Description | Descendents

Exception that is thrown when something IO related goes awry.

*Located in **/io/IOException.php** (line **23**)*

```
Exception
   |
   --amy_lang_Exception
      |
      --amy_io_IOException
```

Direct descendents

Description | *Descendents*

Class	Description
amy_io_FileNotFoundException	Exception that is thrown when a file could not be found.

amy_io_ZipFileInputStream

Description

A ZipFileInputStream obtains input from a file inside a zip file. What files are available depends on the zip file.

*Located in **/io/ZipFileInputStream.php** (line **24**)*

```
amy_lang_Object
   |
   --amy_io_InputStream
      |
      --amy_io_FileInputStream
         |
         --amy_io_ZipFileInputStream
```

Method Summary

amy_io_ZipFileInputStream **__construct** (*string* **$archive**, *string* **$name**)

Variables

Inherited Variables

Inherited from amy_io_FileInputStream

amy_io_FileInputStream::$ifp

Methods

Constructor __construct (line 39)

Creates a ZipFileInputStream by opening a connection to an file entry within a zip archive.

If the archive file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading, or the file entry in the zip archive cannot be opened then a FileNotFoundException is thrown.

- **access:** public

amy_io_ZipFileInputStream **__construct** (*string* **$archive**, *string* **$name**)

- *string* **$archive**: Path to the archive containing the file to be opened.
- *string* **$name**: Path within the archive to the file to be opened.

Redefinition of:

amy_io_FileInputStream::__construct()
Creates a FileInputStream by opening a connection to an actual file, the file named by the path name $name in the filesystem.

Inherited Methods

Inherited From amy_io_FileInputStream

- **amy_io_FileInputStream::__construct()**
- **amy_io_FileInputStream::close()**
- **amy_io_FileInputStream::read()**
- **amy_io_FileInputStream::skip()**

Inherited From amy_io_InputStream

- **amy_io_InputStream::close()**
- **amy_io_InputStream::read()**
- **amy_io_InputStream::skip()**

Inherited From amy_lang_Object

- **amy_lang_Object::getClass()**

Attachment B: Kif API Documentation

kif_Request

Description

The base request for Kif. It provides CMS specific functionality and attributes.

*Located in **/Request.php** (line **23**)*

```
amy_servlet_Request
   |
   --kif_Request
```

Method Summary

The **pathinfo** ()

Methods

pathinfo (line 31)

Returns the path information of this request.

- **return:** path of this request. The default (index) path gets translated to the home page first.
- **access:** public

The **pathinfo** ()

kif_ResourceServlet

Description

Services resource requests.

*Located in **/ResourceServlet.php** (line **23**)*

```
amy_servlet_Servlet
   |
   --kif_ResourceServlet
```

Method Summary

void **doGet** (*kif_Request* **$request**, *kif_Response* **$response**)

Methods

doGet (line 45)

Finds a requested resource and outputs it to the requester.

- **access:** public

void **doGet** (*kif_Request* **$request**, *kif_Response* **$response**)

- *amy_servlet_Request* **$request**: The request for the resource.
- *amy_servlet_Response* **$response**: The response that will be given.

kif_Response

Description

The base response for Kif. It provides CMS specific functionality and attributes.

*Located in **/Response.php** (line **23**)*

```
amy_servlet_Response
   |
   --kif_Response
```

kif_Servlet

Description

Kif's standard servlet. All KifObjects and their related servlets are descendants of this servlet.

This root servlet implements the dispatching of the request to the right servlet, with the correct context.

*Located in **/Servlet.php** (line **26**)*

```
amy_servlet_Servlet
   |
   --kif_Servlet
```

Method Summary

void **service** (*kif_Request* **$request**, *kif_Response* **$response**)

Methods

service (line 34)

Run the Servlet.

- **access:** public

void **service** (*kif_Request* **$request**, *kif_Response* **$response**)

- *amy_servlet_Request* **$request**: The request for functionality.
- *amy_servlet_Response* **$response**: The response that will be given.