Technische documentatie Edubox 2.0 EML processing

Onderwijstechnologisch expertisecentrum OTEC
Open Universiteit Nederland

Technische Documentatie Edubox 2.0 EML processing

Architectuur Edubox en Omnimark

OTEC 2001/12

OTEC werkdocumenten

De Open Universiteit Nederland ontwikkelt en verzorgt open hoger afstandsonderwijs en is tevens een centrale partner in het consortium voor vernieuwing van het Hoger Onderwijs. Onderwijstechnologische vernieuwingen krijgen daarbij speciale aandacht. Binnen de Open Universiteit Nederland is de onderwijskundige en onderwijstechnologische expertise samengebracht in het Onderwijstechnologisch expertisecentrum (OTEC). Dit centrum vervult taken in het kader van ontwikkeling, vernieuwing, onderzoek en evaluatie van het onderwijs van de Open Universiteit Nederland en haar consortiumpartners. Deze taken worden veelal uitgevoerd in nauwe samenwerking met directoraten en faculteiten van de OU en/of samenwerkingspartners in het consortium.

De werkzaamheden van het OTEC leiden regelmatig tot producten, zoals voorlopige onderzoeksresultaten, strategische stellingnames en overwegingen op projektniveau, functionele specificaties van informatiesystemen, en dergelijke. Deze worden door het OTEC vastgelegd en onder de aandacht gebracht in een reeks werkdocumenten waarvan het voorliggende deel uitmaakt.
Naast deze reeks werkdocumenten geeft het OTEC een reeks rapporten uit met een meer geformaliseerd en/of afgerond karakter.

De OTEC rapporten en sommige werkdocumenten kunnen worden besteld bij:

Open Universiteit Nederland
secretariaat OTEC
Postbus 2960
6401 DL Heerlen
Tel. 045-5762406 of 5762942
Fax. 045-5762802

of opgehaald via Internet: http://www.ou.nl/OTEC

Onderwijstechnologisch expertisecentrum (OTEC)
Open Universiteit Nederland

Technische documentatie Edubox 2.0 EML processing
Edubox architectuur en Omnimark

Colofon

Titel:	Technische documentatie Edubox 2.0 EML processing
Subtitel:	Edubox architectuur en Omnimark
Auteurs:	Hubert Vogten, Marc Verhooren
Projectleiding:	Ghislain Rodenburg
Projectondersteuning:	Mieke Haemers
Uitgifte:	OTEC
Datum druk:	24 maart 2004

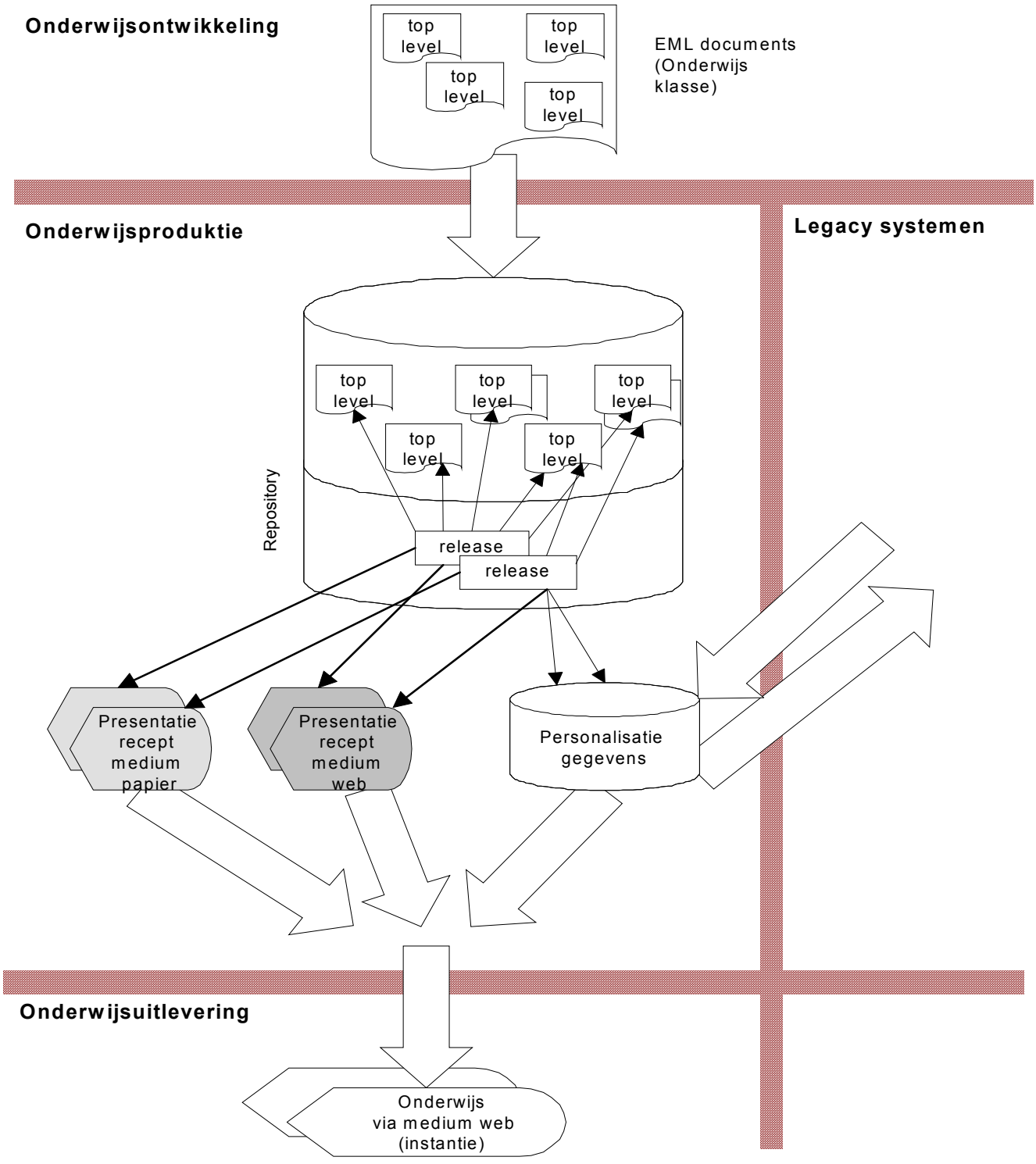
Inhoudsopgave

Inleiding

Dit document beschrijft de algemene Edubox architectuur en de modules die in Omnimark gemaakt zijn om EML om te zetten naar een door de Edubox Webplayer te interpreteren formaat (tussenformaat in XML). Verder is beschreven op welke wijze de programma's kunnen worden aangeroepen.

Edubox architectuur

Edubox architectuur



Edubox bestaat uit een drietal lagen te weten: onderwijsontwikkeling, onderwijsproductie en onderwijsuitlevering.

Onderwijsontwikkeling

Onderwijsontwikkeling is het maken van EML content. De Edubox architectuur doet geen uitspraken over deze omgeving. Voor de Edubox architectuur is deze omgeving dan ook een black-box waarvan alleen het resultaat bekend is, namelijk een EML bestand. Bovendien is ook niet bekend hoe dit resultaat aan het Edubox systeem wordt overgedragen. Dit kan per klant implementatie verschillen. Daarom is de onderwijsontwikkeling ook door een 'muur' gescheiden van de rest van het Edubox systeem.

Met andere woorden betekent dit dat de Edubox klant bepaalt:
- hoe EML materiaal wordt geproduceerd en welke tooling hiervoor wordt gebruikt Momenteel ondersteunt Edubox de Framemaker+SGML met Visual Sourcesave oplossing.
- hoe de A.O. omtrent het auteursprocess er uit ziet
- wanneer en hoe vaak materiaal aan de rest van het Edubox systeem wordt aangeboden.

In de figuur bestaat het aangeleverde document uit verschillende kleinere componenten. Dit zijn de zogenaamde toplevel elementen. Toplevel elementen zijn de eenheden die als component doorgegeven kunnen worden aan de rest van het Edubox systeem. Iedere component heeft een unieke id met een versie nummer. De onderwijsontwikkelingslaag is verantwoordelijk voor het uitdelen van deze unieke id's en versienummers.

Onderwijsproductie

Het materiaal dat van de onderwijsontwikkeling wordt aangeleverd, wordt nu in productie genomen. Dit betekent dat de componenten in de database worden opgeslagen. Op deze manier ontstaat een repository van componenten waarbij van iedere component meerdere versies kunnen bestaan. In EML worden de links tussen deze componenten dynamisch gedefinieerd. Hierbij kan gedacht worden aan verwijzing in de trant van verwijs naar de hoogste versie van component X. Dit betekent dat het oplossen van een dergelijke link tijdstipgebonden is. Immers op verschillende tijdstippen kunnen verschillend versie van component X aanwezig zijn in de repository. Dit is een ongewenst neveneffect omdat:
- te allen tijden angetoond moet kunnen worden welk materiaal gebruikt is tijdens de onderwijsuitvoering.
- herpublicaties, of publicaties voor andere media op een ander tijdstip dan de eerste publicatie toch hetzelfde materiaal (inhoudelijk) moeten opleveren.

Beide punten zou men kunnen typeren aan reproduceerbaarheid. Om reproduceerbaarheid te garanderen is het begrip 'release' ingevoerd. Een release legt alle dynamische links, zoals die in EML bestaan, vast op het bepaald tijdstip en bewaart deze. Elke volgende bewerkingsstap op het materiaal zal zijn gebaseerd op een release.

EML is ontworpen om mediumneutraal te zijn. De architectuur voorziet hierin door voor ieder medium een publicatierecept generator aan te bieden. Een publicatie recept is een verzameling bestanden en programma's die in staat zijn om voor een bepaalde gebruiker het gewenste gepersonaliseerde onderwijs te genereren. Momenteel ondersteunt het Edubox systeem alleen een publicatierecept generator voor het web.

De gegevens die nodig zijn om het materiaal te kunnen personaliseren worden opgeslagen in een aparte database (dossier database).

Legacy systemen

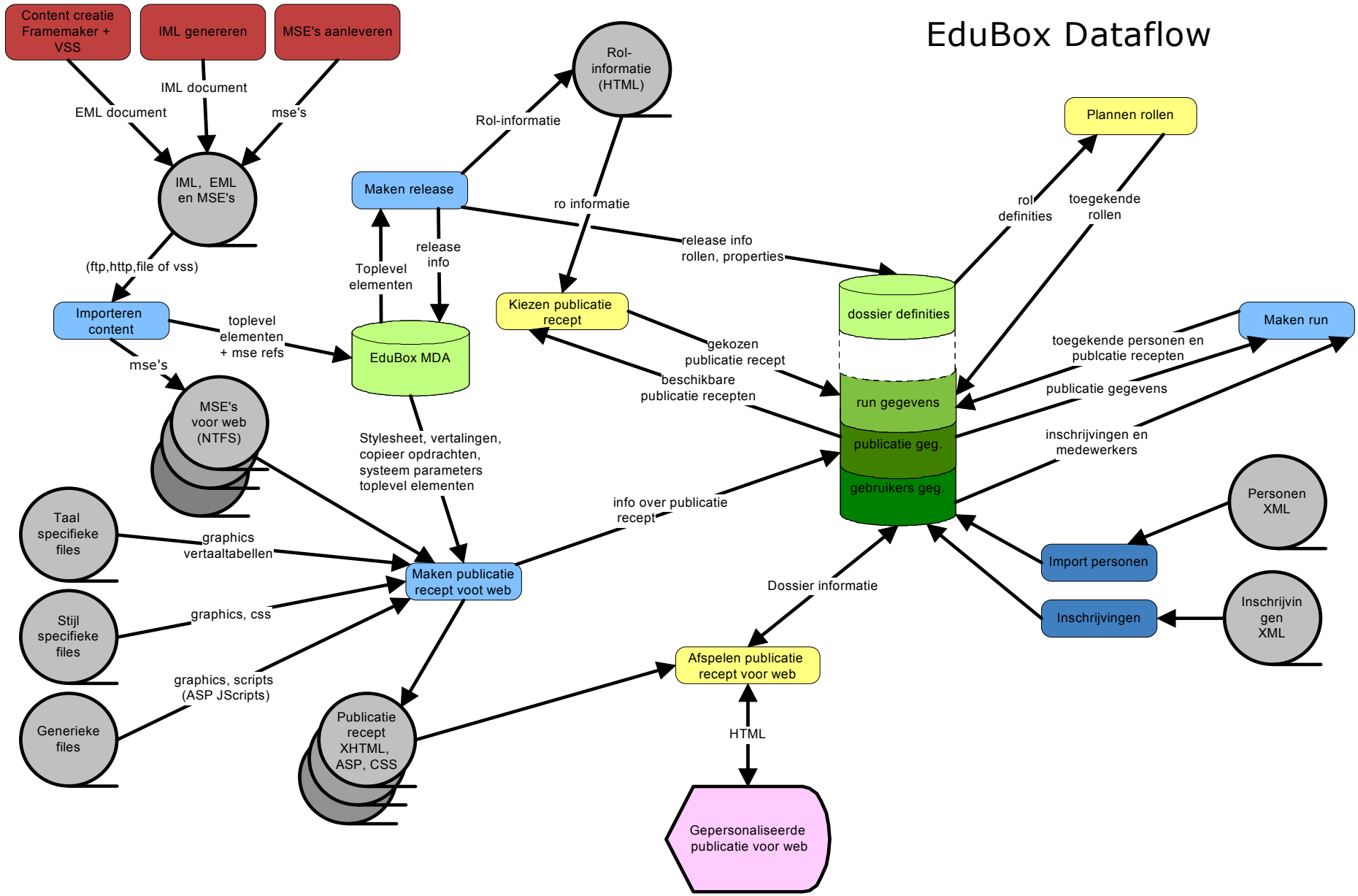
Dit zijn alle systemen die bij een klant 'draaien'. Enerzijds leveren deze legacy systemen de gegevens aan voor de initiële vulling van de dossier database. Hierbij kan gedacht worden aan zaken zoals gebruikersgegevens en inschrijvingen voor cursussen.

Aan de andere kant kunnen resultaten van de uitwijsuitlevering teruggekoppeld worden naar de legacy systemen. Hoe een en ander eruit ziet wordt niet door de architectuur beschreven en is klantafhankelijk.

Onderwijsuitlevering

De onderwijsuitlevering bevat het eigenlijke onderwijsproces. Iedere gebruiker krijgt de gepersonaliseerde versie van het onderwijsmateriaal aangeleverd voor een publicatierecept voor een bepaald medium. Hoe dit proces eruit ziet, is in zijn geheel beschreven via de EML en vormt een black box ten aanzien van de architectuur. Het is mogelijk om tijdens onderwijsuitlevering te wisselen van publicatierecept indien er meerdere publicatierecepten voor een release bestaan.

Edubox dataflow



De onderwijsontwikkeling bestaat uit een drietal processen die in het schema worden weergegeven door een rode kleur. Het eerste proces is het maken van de inhoud. De tooling die hierbij op dit moment wordt gebruikt zijn Framemaker in combinatie met Visual Sourcesafe. Output van dit proces is een EML bestand. Vervolgens wordt op basis van dit EML bestand een IML bestand gemaakt. IML zal later de basis vormen voor de import operatie. Tooling hiervoor is de IML generator. Output van dit proces is een IML file. Tot slot zal voor iedere MSE referentie ook nog het daardwerkelijk bronbestand aangeleverd moeten worden.

Alle bestanden wordt bij elkaar gereed gezet. Afhankelijk van de klantimplementatie zijn hiervoor een aantal opties:

- op een geshared filesysteem
- een http site
- een ftp site
- een visual sourcesafe directory.

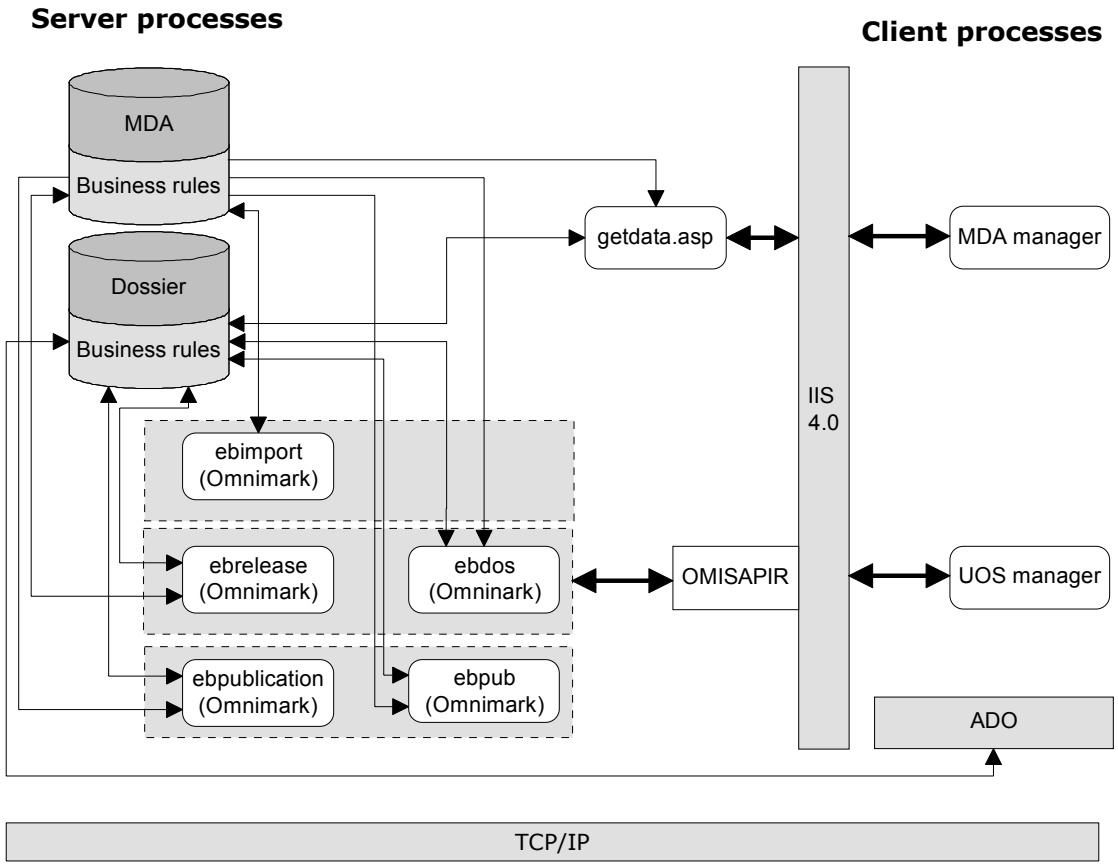
Alle blauw gekleurde processen maken onderdeel uit van de onderwijsproductie. Dit begint met het importeren van de content in de MDA zoals die is aangeleverd vanuit de onderwijsontwikkeling. De EML content wordt rechtstreeks in de MDA opgeslagen. De MSE's worden op een NTFS bewaard waarbij referenties hiernaar in de MDA zijn opgeslagen.
Na de importactie kan op basis van dit nieuwe materiaal een release gemaakt worden. Een release groepeert EML elementen volgens linking regels zoals gedefinieerd in de EML en bewaart deze in de MDA. Bovendien worden nu al enkele inhoudelijke pagina's aangemaakt die onafhankelijk zijn van het gekozen presentatierecept. Bovendien worden alle definities van properties en rollen al in de dossierdatabase weggeschreven.
Na het maken van de release kan een publicatierecept aangemaakt worden op basis van zo'n release door het verwerken van de EML inhoud. Momenteel worden alleen publicatierecepten voor het web ondersteund. Voor het maken van een dergelijk publicatierecept wordt gebruik gemaakt van diversie configuratiebestanden ten aanzien van taal en vormgeving die in de MDA zijn opgeslagen. Het gegenereerde bestand op basis van de EML inclusief een set nog te kopiëren bestanden vormt tezamen het publicatierecept dat op de website geplaatst wordt. De regels voor de te kopiëren bestanden zijn ook opgeslagen in de MDA.
Nadat een publicatie is gemaakt, kan worden besloten om een of meerdere runs aan te maken. Een run wordt gevormd door een release, een reeks publicatierecepten, en personen in bepaalde rollen aan elkaar te koppelen.

Deze procesen horen ook tot de onderwijsproductie. Het betreft hier processen die personen en inschrijvingen importeren op basis van aangeleverde XML files.

Met geel worden alle processen weergegeven die gedurende de uitvoering van het onderwijs plaatsvinden. Het afspelen van het publicatierecept is de meest voor de hand liggende. Maar ook het verder inplannen en onderverdelen van rollen behoort tot deze processen. Tot slot kan iedere gebruiker op een willekeurig moment ervoor kiezen om van rol of publicatierecept te wisselen.

EML Processing

EML Processing



EML processing bestaat uit een aantal server processen en een aantal client processen. Deze processen communiceren met elkaar via een webserver (IIS4).

Op de server draaien vijf Omnimark-processen die de EML processing voor hun rekening nemen. Deze processen gedragen zich als webservers en wachten tot ze een verzoek (http-request) krijgen om een actie uit te voeren.
De Omnimark-processen maken drie verschillende fasen van EML processen mogelijk, nl:

- EML import (*ebimport*);
- Release maken (*ebrelease* & *ebdos*). Hierbij is het zo dat ebrelease het primaire proces is, dat eenmalig wordt aangeroepen. Dit proces voert zelf een aantal kleinere taken uit en roept zelf een aantal malen ebdos aan om bepaalde opdrachten uit te voeren;
- Publicatie maken (*ebpublication* & *ebpub*). De werking is gelijk aan het maken van een release: ebpublication is het primaire proces en roept ebpub aan om een aantal specifieke taken uit te voeren;

De processen lezen uit en schrijven naar de databases hoofdzakelijk door middel van de in de databases vastgelegde business-rules. Deze business-rules bestaan vooral in de vorm van views en stored-procedures.

De drie hoofdprocessen worden aangeroepen door twee client-processen, de *MDA manager* en de *unit-of-study-manager*, die het interface naar de gebuiker toe vormen.

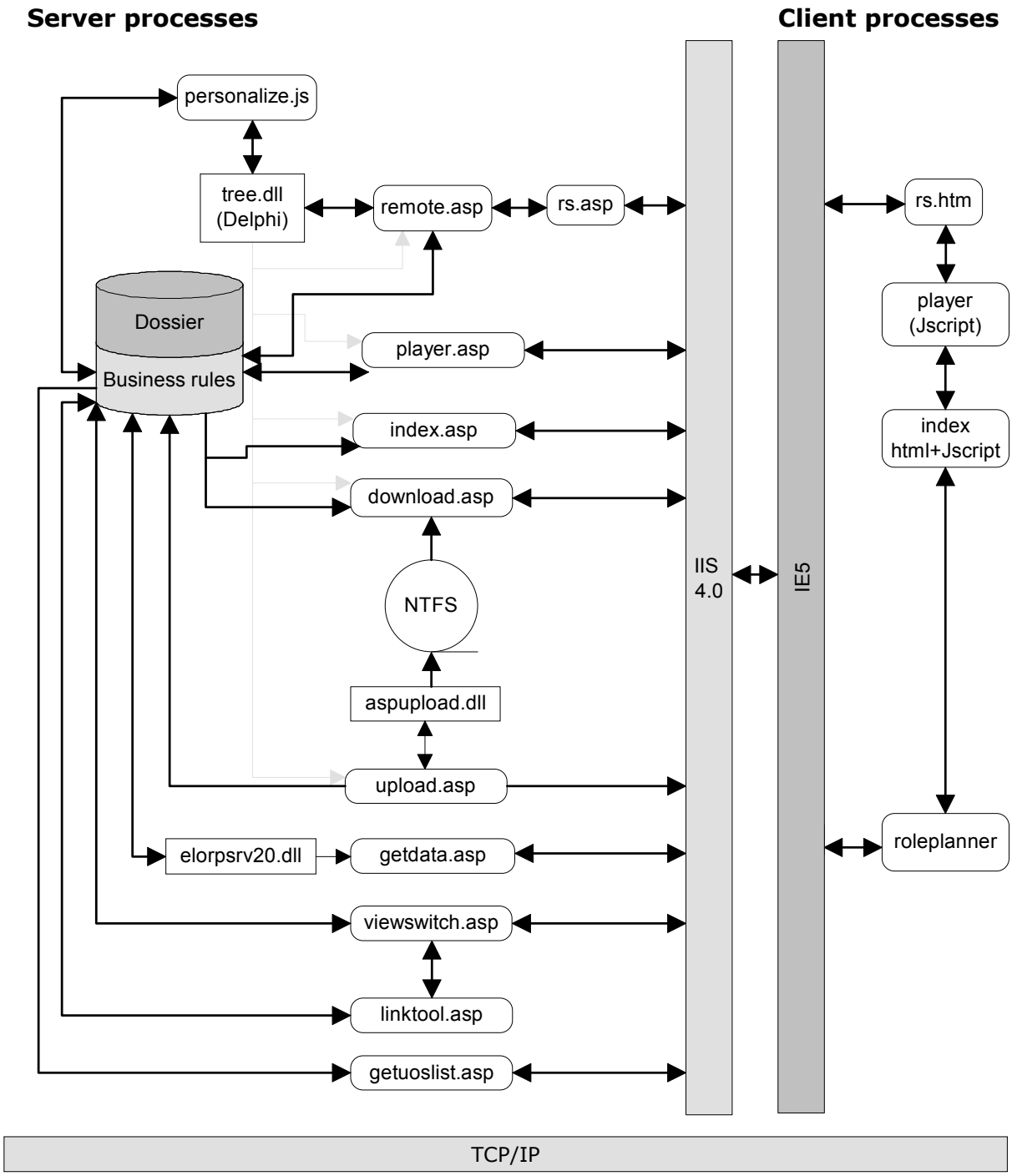
De *MDA manager* verzorgt het importeren van EML (roept hiervoor dus *ebimport* aan, met een aantal parameters) en het toekennen van een runplanner aan een geïmporteerde unit-of-study. Om met de databases te kunnen communiceren gebruikt de MDA manager een stuk middle-ware dat in de vorm van het bestand getdata.asp op de server aanwezig is.

De *unit-of-study-manager* verzorgt het maken van releases (aanroepen van *ebrelease*), het maken van publicaties (aanroepen van *ebpublication*) en het maken van runs. Om met de database te kunnen communiceren wordt gebruik gemaakt van ADO en ODBC.

Om het aanspreken van deze processen mogelijk te maken is *OMISAPIR.DLL* als plug-in op de webserver geïnstalleerd. Wanneer deze DLL benaderd wordt met de juiste procesnaam, dan weet deze DLL via welke poort dit proces te benaderen is.
De *MDA manager* en de *unit-of-study-manager* roepen op deze manier de 3 primaire omnimark processen aan. Maar ook roepen de primaire omnimark processen de subprocessen op deze manier aan.

EML Delivery

EML Delivery



Onder EML delivery worden de processen en applicaties verstaan die nodig zijn om een gemaakte run van een publicatie aan de gebruiker gepersonaliseerd te kunnen aanbieden (door middel van de '*EML Player*').

De basis van de player wordt gevormd door de processen *index.asp* en *player.asp*. Het aanroepen van de EML player heeft als gevolg dat deze twee processen beginnen te lopen en zich aan de client-zijde representeren als het geraamte van de player, in HTML + Jscript-vorm.

Het aanleveren van gepersonaliseerde inhoud aan de player (bijv. de activiteiten-boom) wordt verzorgd door *tree.dll* en *personalize.js*. *Tree.dll* analyseert de aangeboden XHTML-bestanden en laat alle noodzakelijke berekeningen/evaluaties uitvoeren door *personalize.js.* Dit laatste proces haalt ook de evt. benodigde gegevens uit de database.

De communicatie tussen de player en *tree.dll* verloopt met behulp van remote scripting. Om remote scripting mogelijk te maken zijn aan de server-zijde de bestanden *rs.asp* en *remote.asp* aanwezig en aan de client-zijde *rs.htm*. *rs.asp* en *rs.htm* maken remote scripting als principe mogelijk en remote.asp verzorgt de interface naar *tree.dll*.

Het uploaden en downloaden van bestanden wordt verzorgd door *aspupload.d*ll en de processen *upload.asp* en *download.asp*.

De roleplanner draait in de vorm van een ActiveX object op de client en maakt gebruik van *elorpsrv20.dll* en *getdata.asp* op de server, die als middle-ware functioneren.

Verder maakt de player gebruik van *linktool.asp* om dynamisch links te kunnen oplossen en van *viewswitch.asp* om van rol te kunnen wisselen.

Om in de web-portal de lijst met units-of-study te kunnen tonen die voor de ingelogde gebruikers beschikbaar zijn, draait *getuoslist.asp* op de server.

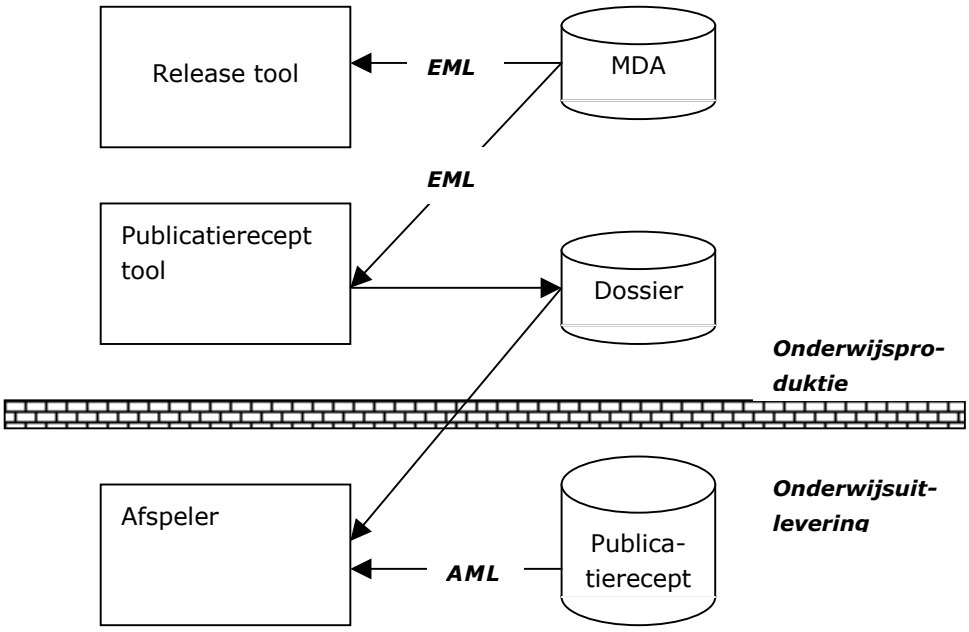
Edubox tussenformaat

Inleiding

Als onderdeel van de Edubox architectuur is besloten om te gaan werken met een driedubbele parsing van het EML bronmateriaal. De eerste parsingslag vindt plaats tijdens het maken van de release. Alle gegevens die medium en publicatierecept neutraal zijn worden in deze fase aangemaakt. Het betreft hier met name zaken als:

- dossier opbouw en structuur door middel van properties en rollen
- notificaties en andere triggers voortkomend uit de EML definities
- role informatie t.b.v. de linktool.

De tweede parsing slag is het maken van het publicatierecept. Een publicatierecept bevat alle voorschriften en ingrediënten (vandaar de term recept) voor het 'bereiden' van een publicatie. Hierbij kan gedacht worden aan pure inhoud, maar ook instructies tot de personalisatie van deze inhoud. Het 'bereiden' van een dergelijk publicatierecept wordt ook wel afspelen genoemd. Dit afspelen is dan ook het laatste parsing process dat plaatsvindt. Schematisch ziet het een en ander er als volgt uit:



Wat in de figuur meteen duidelijk wordt, is het verschil tussen de eerste twee parsing slagen en de laatste. Ten eerste behoren de eerste twee parsing slagen tot de zogenaamde onderwijsproductie. Het betreft hier het voorbereidende werk dat nodig is om onderwijs te kunnen leveren. In het traditionele proces is deze fase te vergelijken met het drukken en verspreiden van het studiemateriaal. Qua performance stellen deze parsing slagen relatieve lage eisen omdat ze eenmalig per type publicatierecept worden uitgevoerd. De parsing vindt plaats op basis van het aangeleverde EML formaat. Een van de producten is het publicatierecept. Het publicatierecept is in feite ook een XML formaat net als EML. Echter de focus is hier veel eerder om de primaire functies van Edubox vanuit een technisch perspectief, in plaats van een auteursperspectief. In feite heeft de vertaling plaatsgevonden

van de semantisch rijke EML naar de concrete invulling van deze semantiek door de gekozen technische oplossing.

Dit vertaalde EML wordt AML genoeemd. AML, oftewel Additional Markup Language, is het formaat dat wordt geparsed door de afspeler. Zoals reeds vermeld staat dit formaat veel dichter bij de techniek dan EML. Gezien het feit dat de afspeler onderdeel uitmaakt van de onderwijsuitlevering is dit ook noodzakelijk. Immers het vertalen van dit formaat gebeurt iedere keer als een gebruiker het materiaal raadpleegt en daarom is het parsen van het AML formaat zeer zeker tijdkritsch.

Uitgangspunten van AML

AML is de XML applicatie die door de afspeler vertaald wordt tijdens de onderwijsuitlevering. De reden om voor een dergelijk Edubox tussenformaat te kiezen zijn de volgende:

- EML was/is aan veranderingen onderhevig. Vaak zijn deze verandering gericht op het auteurs/onderwijskundig perspectief. Voor de techniek is de impact vaak beperkt. Het is daarom verstandig een 'dam' in te bouwen die de impact op het Edubox systeem zo veel mogelijk beperkt. AML dient als deze barriere. Via AML is het mogelijk om de primaire low-level functionaliteit van het Edubox systeem te benaderen. Hoe en welke EML constructie vertaald worden in deze AML applicatie is dan slechts de verantwoordelijkheid van de eerste twee parsing slagen. De rest van het Edubox systeem wordt niet door EML wijzigingen beïnvloed.
- Een ander goede reden voor de keuze van het AML formaat is het feit datde parsing van het onderwijsrecept snel moet gebeuren. Het EML formaat is te complex om dit te kunnen realiseren.
- De AML applicatie maakt het mogelijk om voor modularisatie te zorgen in het Edubox systeem. Het onderwijsproductieproces en het onderwijsuitleveringsproces kunnen onafhankelijk van elkaar (door) ontwikkeld worden met een tweetal duidelijk koppelvlakken. Te weten de dossier database met zijn middleware (in versie 2.0x van Edubox door middel van stored procedures) en de AML applicatie.

Hieronder volgt een lijst van uitgangspunten die van kracht waren tijdens het ontwerp van de AML applicatie:

- AML is specifiek ontworpen voor de web afspeler van Edubox. Daarom is ook gekozen om XHTML op te nemen in de AML applicatie. Het grote voordeel is dat statische inhoud al door de publicatiereceptgenerator aangemaakt kan worden en niet meer door de afspeler vertaald hoeft te worden. Dit betekent ook dat alle additionele tags van AML ten opzichte van XHTML te maken hebben met de personalisatie van de inhoud. N.B.: XHTML stelt specifieke eisen, afgeleid van de XML standaard, aan het HTML formaat met name met betrekking tot well-formedness van documenten !!!
- AML moet snel verwerkt kunnen worden. Sommige keuzes zijn daardoor onder andere bepaald door de sterke en zwakke punten van de gebruikte parser in de afspeler. Mede daardoor is het ook niet de bedoeling om AML tot een of andere standaard te verheffen buiten het Edubox systeem. AML is semantiekloos vanuit een onderwijskundig perspectief. Dit betekent dat een EML document wel te vertalen is naar een AML document, maar niet vice-versa.

AML in detail

XHTML

Deze tag is de verzamelnaam voor alle XHTML tags. XHTML is niet 'echt' opgenomen in de AML DTD omdat de afspeler toch niet validerend parsed. Het XHTML element representeert dus een willekeurige XHTML tag die door de gekozen browser wordt ondersteund.

INTERACTIONS

Interactions representeert alle vraagtypen die momenteel door Edubox worden ondersteund. Interactions heeft een aantal attributen die beschrijven hoe de vragen getoond moeten worden. Via deze attributen wordt vastgelegd:

- of er feedback getoond moet worden na het beantwoorden van de vragen (feedback)
- of er hints getoond moeten worden (hints)
- of de vragen in een vaste of willekeurige volgorde getoond moeten worden (sequence)
- het aantal vragen dat onderdeel uitmaakt van dit interactie formulier (number-test-items)
- het aantal keren dat de vragen beantwoord mogen worden (trials).

ANCHOR

Bepaald anchors voor EML links. De quicknav en caption attributen worden momenteel niet meer gebruikt.

CHARACTERIZATION

Dit is de AML vertaling van de Special elementen in EML. Aan de hand van de meegegeven attributen wordt bepaald hoe de inhoud van de charactization getoond moet worden.

JUMP

Dit is de AML equivalent van de XHTML <A> tag.

METADATA

Deze tag zal een link naar de metadata tonen. Aangezien de metadata geen echte inhoud zijn, moet de afhandeling anders plaatsvinden dan bij normaal materiaal. Momenteel wordt op basis van deze tag een dialoog venster getoond met de metadata.

USER-COMPLETED

Representeert een gebruikersdialoog waarmee de gebruiker aangeeft of hij een activiteit heeft afgerond. De invoer van de gebruiker wordt opgeslagen in een property zoals meegegeven in het property-id attribuut.

SHOW-PROPERTY

Toont de waarde van een dossier property. Het attribuut property-id geeft aan welke property getoond moet worden. Het attribuut property-of geeft aan of dit een property is van de gebruiker of van de persoon die door de gebruiker wordt begeleid.

SHOW-PROPERTY-GROUP

Vergelijkbaar met SHOW-PROPERTY, maar dan voor een hele group van properties uit het dossier. Groepen worden in EML gedefinieerd.

SET-PROPERTY

Geeft de mogelijkheid om de waarde van de property te veranderen. Ook hier geeft de property-id aan welke property gewijzigd moet worden. Het attribuut property-of geeft wederom aan van wie de property is (gebruiker, of persoon die begeleid wordt). Optioneel kunnen notificaties worden gespecificeerd. Nadat de property is gezet door de gebruiker dient het systeem een trigger te sturen voor de notificatie waarvan het id in het attribuut id is opgeslagen.

SET-PROPERTY-GROUP

Doet hetzelfde als SET-PROPERTY, maar dan voor een hele groep properties.

FOR-PERSON-IN-ROLE

Via dit element wordt een overzicht getoond van bepaalde informatie voor elke gebruiker die lid is van een bepaalde rol. Het role-id attribuut bepaalt de rol waarom het hier gaat. De volgende zaken kunnen getoond worden per gebruiker:

- de waarde van een property of property-group
- informatie over de property of property-group (type, initiële settings enz.)
- de rollen die een gebruiker mag aannemen
- toegangsinformatie per activiteit (eerste keer, laatste keer, aantal keren enz.)
- voortgangsinformatie per activiteit (afrondingen).

SHOW-MESSAGES

Toont alle berichten die vanuit een bepaald communicatie object werden verstuurd. In feite vormt dit een filter (of view) op alle ontvangen berichten.

SEND-MESSAGE

Genereert een mail formulier. Hierbij kan een standaardbericht en subject worden meegegeven. De ontvangende partijen worden per role gespecificeerd. De gebruiker kan oftewel een keuze maken uit de personen in de rol, of alle personen in de rol ontvangen het bericht.

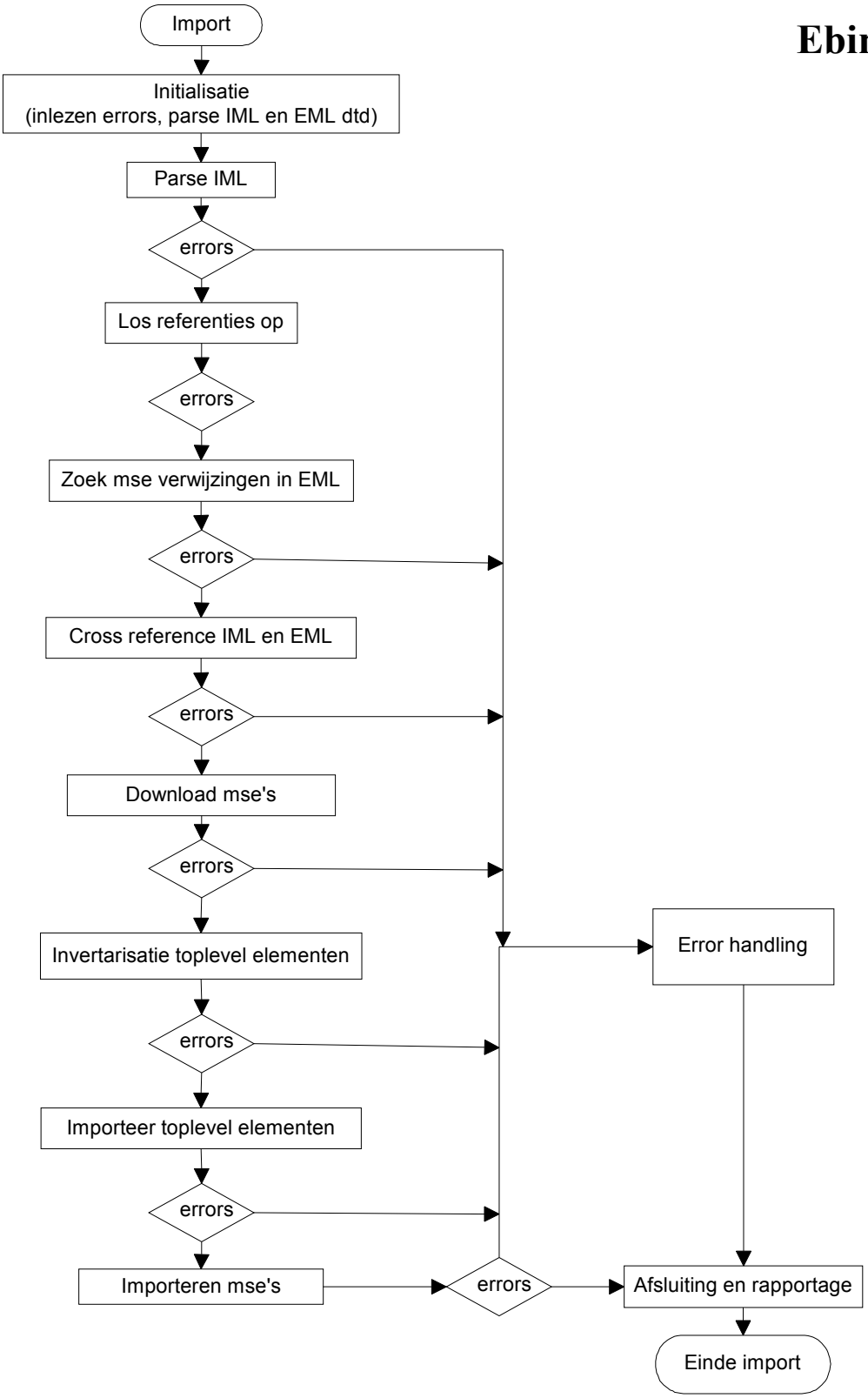
Text-line

Deze tag representeert een tekstregel. Dit is van belang wanneer regels (medium en publicatierecept neutraal) dienen te worden opgeslagen. Dit element is een eerste poging om te komen tot een gemeenschappelijk formaat voor alle publicatierecept generatoren. Dit is met name van belang voor informatie die vanuit een afspeler weer terug gaat naar de dossier database. Dit is in feite input met markup informatie.

RUN-INFO

Via dit element kan allerlei informatie over een run worden opgevraagd via startdatum, einddatum, rollen enz.

Ebimport



Ebimport

Globale structuur:

- ServiceInitialize
  - *g_read_settings (1)*
  - g_init (2)
    - *Inlezen error-messages (2.1)*
    - *Parse IML DTD (2.2)*
    - *Parse EML DTD (2.3)*
- ServiceMain
  - *Parse IML (3)*
  - *g_fill_in_refs (4)*
  - *Parse EML Fase 1 (5)*
  - *g_validate_references (6)*
  - *g_download_MSE (7)*
  - *Parse EML Fase 2 (8)*
  - *Parse EML Fase 3 (9)*
  - *g_write_MSE_elements (10)*
  - *always (11)*

1. g_read_settings

Inlezen van het bestand 'settings.xml' dat behoort bij dit proces. Er wordt een XML parse uitgevoerd van deze file. Tijdens deze parse worden de naam, evt. key en de waarde van een setting gelezen en met behulp van de functie 'g_set_initial_value' de juiste variabele van een waarde voorzien. 'g_set_initial_value' is voor elk proces specifiek omdat hierin de namen van de te vullen variabelen hard gecodeerd staan; toevoegen van een variabele aan 'setting.xml' betekent dus aanpassen van 'g_set_initial_value'!

2.1 Inlezen error-messages

Inlezen van het bestand met foutmeldingen. Het bestand wordt gescand met behulp van pattern-matching: per melding worden het nummer en de bijbehorende melding gelezen en deze op een shelf ('g_s_error_messages') gezet met als key het nummer en als inhoud de inhoud van de boodschap.

2.2 Parse IML DTD

Parsen van de IML DTD zodat de parser deze al in het geheugen heeft wanneer de IML verwerkt gaat worden.

2.3 Parse EML DTD

Parsen van de EML DTD zodat de parser deze al in het geheugen heeft wanneer de IML verwerkt gaat worden.

3. Parse IML

Parsen van het IML bestand. In deze fase wordt een inventarisatie gedaan van de MSE bestanden die in het IML bestand zijn gespecificeerd. Er wordt een aantal shelves gevuld dat aangeeft voor welke media welke bestanden gespecificeerd zijn, waar deze te vinden zijn etc.
Ook wordt in deze fase het betreffende EML document opgehaald en in een tijdelijk SGML bestand opgeslagen.

4. g_fill_in_refs

Het invullen van ontbrekende 'Ref-worldwide-unique-id's' op plaatsen waar dit nodig en mogelijk is. Dit is alleen mogelijk op plaatsen waar een referentie naar een element wordt gelegd met behulp van 'Id-ref'. Het EML document wordt twee maal doorlopen:

- de eerste keer wordt een inventarisatie gemaakt van alle elementen die het attribuut 'Id' ingevuld hebben. Van een dergelijk element wordt opgeslagen welk 'Worldwide-unique-id' het heeft.
- De tweede keer worden de elementen opgezocht waarvan 'Id-ref' is ingevuld en 'Ref-worldwide-unique-id' niet is ingevuld. Wordt een dergelijk element gevonden dan wordt op de in de eerste stap opgebouwde shelf gekeken welk 'worldwide-unique-id' behoort bij het gevonden 'Id'. Wanneer geen bijbehorend 'worldwide-unique-id' gevonden wordt, wordt een foutmelding gegenereerd, want dan betreft het een referentie naar een element dat niet binnen het betreffende document is gespecificeerd (een verwijzing naar een element buiten het huidige document mag immers alleen maar met behulp van een 'Ref-worldwide-unique-id' en nooit m.b.v een 'Id-ref'!).

N.B. in de eerste stap wordt het tijdelijke SGML bestand gelezen dat tijdens de IML parsing fase is aangemaakt; tijdens de tweede stap wordt alles naar het definitieve SGML bestand geschreven dat voor de verdere fasen van de import zal worden gebruikt. Dit gebeurt door alle elementen door te laten (direct weg te schrijven) die niet gewijzigd hoeven te worden en de elementen die wel gewijzigd moeten worden opnieuw te schrijven naar het bestand.

5. Parse EML Fase 1

Het opzoeken van MSE specificaties ('Figure-source', 'Formula', 'Video', 'Audio') in het EML document. Gevonden specificaties worden opgeslagen op de shelf 'g_s_MSE_references'. Eenzelfde figuur kan meerdere keren gebruikt zijn in een EML document maar wordt toch maar één keer opgeslagen.

6. g_validate_references

Validatie IML/EML wat betreft MSE's. De functie doet drie controles:

- controleren of het aantal MSE's dat in IML is gespecificeerd gelijk is aan het aantal dat in EML is gedeclareerd
- controleren of alle in EML gedeclareerde MSE's terug te vinden zijn in IML
- controleren of alle in IML gespecificeerde MSE's terug te vinden zijn in EML.

Voor elk van deze controles geldt dat wanneer een controle een fout detecteert, een waarschuwing wordt gegenereerd. De applicatie gaat verder met valideren en stopt pas met een foutmelding wanneer de validatie ten einde is.

7. g_download_MSE

Deze functie haalt de MSE's op en zet ze bij elkaar in een directory. De files worden gehaald van de plaats die aangegeven staat in IML in het element 'URI'. Als de inhoud van dit element leeg is, dan wordt verondersteld dat de desbetreffende MSE reeds aanwezig is op het filesysteem en (de referentie) reeds aanwezig is in de database. De bestanden worden volgens het juiste protocol (te zien aan de URI) opgehaald en in een directory geplaatst. De directory-structuur hiervoor is als volgt:

<offset><medium><datum>, dus bijv. 'd:\MSE\web\20000926\'

Wanneer tijdens het kopiëren iets mis gaat wordt een waarschuwing gegenereerd; de functie blijft echter doorlopen en gaat verder met de volgende MSE. Wanneer de functie is afgerond wordt gekeken of waarschuwingen zijn opgetreden; is dit het geval dan wordt de

import afgebroken met een fatale foutmelding. Op de shelf 'g_s_MSE_import_history' wordt bijgehouden welke bestanden succesvol zijn gekopieerd.

8. Parse EML Fase 2

In deze fase wordt een inventarisatie gedaan van de versies van alle aanwezige toplevel elementen. Op de shelf 'g_s_top_level' worden alle toplevel elementen opgeslagen met als key hun 'Worldwide-unique-id' en als waarde hun versienummer. Alleen de nieuwste versie binnen het document van een toplevel element wordt opgeslagen. Een toplevel element wordt herkend aan het feit dat de elementnaam op de configuratie-shelf 'g_s_top_level_elements' staat.

9. Parse EML Fase 3

In deze fase gebeurt het echte importeren in de MDA database. Het EML document wordt opgeknipt in toplevel elementen. Van elk toplevel element wordt de inhoud in de database opgeslagen. Verder worden nog enkele andere gegevens over dit element los opgeslagen, zoals zijn 'worldwide-unique-id' en versienummer. Een toplevel element kan op zijn beurt één of meerdere toplevel elementen bevatten. In dat geval wordt het geneste toplevel element uit de inhoud van het element gehaald en vervangen door een referentie (volgens EML-conventies) naar dit toplevel element. Ook wordt bij elk toplevel element een veld opgeslagen dat aangeeft welke verwijzingen naar andere toplevel elementen binnen dit element voorkomen.
MSE's worden ook als toplevel beschouwd.

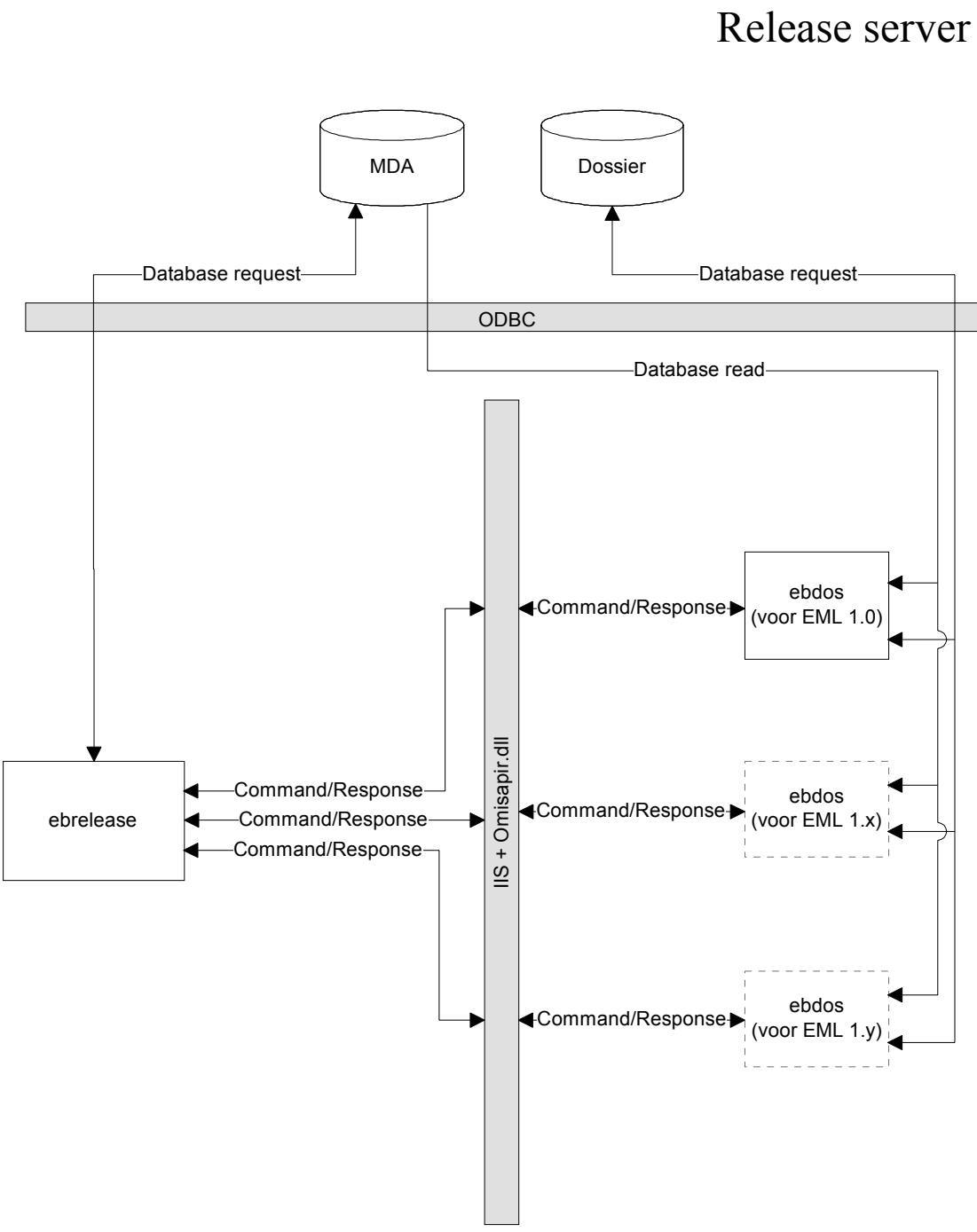
10. g_write_MSE_elements

De MSE files voor ieder medium zoals aangegeven is in de IML file, worden nu in de MDA opgeslagen. Bovendien wordt opgeslagen waar het MSE bestand op het filesysteem terug te vinden is. Indien in de IML geen URI element (te detecteren via de g_s_medium_types shelf) voorkomt, wordt het MSE al als bekend verondersteld. Er wordt gecontroleerd of dit element ook daadwerkelijk in de MDA terug te vinden is. Zo niet, genereert het systeem een waarschuwing.

11. Always

Als laatste stap wordt de connectie met de MDA database gesloten. Vervolgens worden alle tijdelijke bestanden verwijderd. Hierna wordt het resultaat van het import proces als HTTP response teruggegeven. Dit is oftewel een lijst van geïmporteerde MSE's en toplevel elementen, oftewel een lijst met foutmeldingen.

Release Server

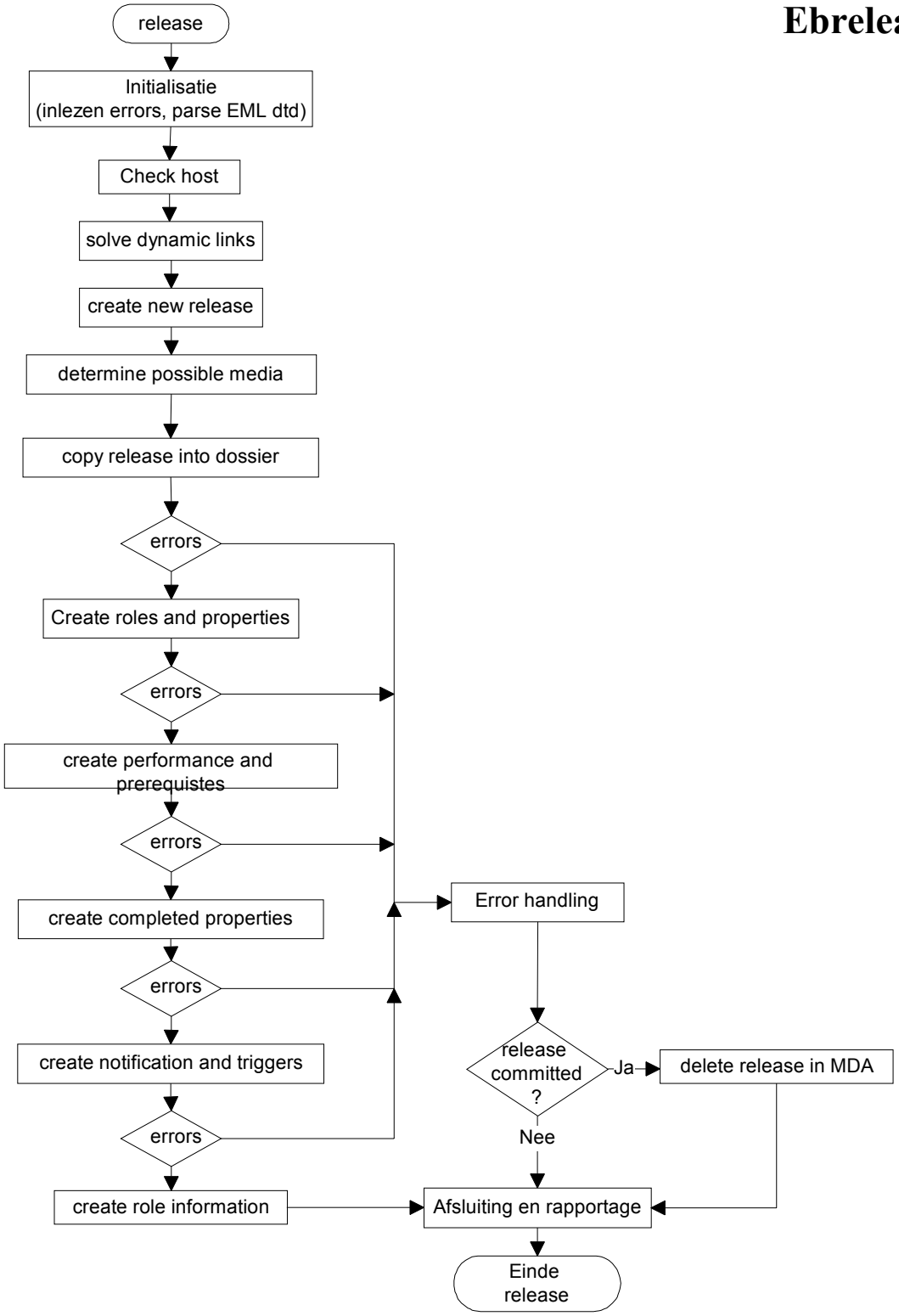


Het release server proces bestaat uit een tweetal (of zelfs meerdere) subprocessen. Het eerste is het ebrelease proces. Dit proces stuurt de creatie van een release. Dit gebeurt onder andere door het aanroepen van een reeks subprocessen. Deze subprocessen maken geen onderdeel uit van het ebrelease proces, maar zijn aparte programma's. Het idee hierachter is dat in deze onderdelen alle afhandeling gebeurt van EML versie afhankelijk code. Daarom is het ook voorstelbaar dat voor een nieuwe versie van EML een nieuw subproces ontstaat.

Dit mechanisme waarborgt de ondersteuning voor kleinere EML wijzigingen waarbij zelfs mixed content (materiaal van meerdere EML versies in een document). Voorwaardelijk hiervoor is natuurlijk dat de hoofdstructuur niet veranderd. Deze subprocessen heten ebdos programma's. De communicatie tussen het ebrelease hoofdprogramma en de ebdos subprogramma's verloopt via de web server en de omisapir.dll van Omnimark.

Naast de scheiding of het functioneel niveau is er ook een scheiding aangebracht op database niveau tussen het ebrelease hoofdprogramma en de ebdos subprogramma's. Het ebrelease programma heeft lees- en schrijftoegang tot de MDA database. Een ebdos programma leest alleen van de MDA database maar leest en schrijft bovendien naar de dossier database.

Ebrelease


Ebrelease
release
Initialisatie
(inlezen errors, parse EML dtd)
Check host
solve dynamic links
create new release
determine possible media
copy release into dossier
errors
Create roles and properties
errors
create performance and prerequistes
errors
create completed properties
errors
create notification and triggers
errors
create role information
Error handling
release committed ?
Ja
delete release in MDA
Nee
Afsluiting en rapportage
Einde release

Globale structuur:

- ServiceInitialize
  - *g_read_settings (1)*
  - *g_init (2)*
- ServiceMain
  - *Voorbereiding (3)*
    - g_check_host
    - g_get_element
    - g_create_release
    - g_read_media
    - *g_searchrefs (3.1)*
    - g_write_media
  - Kopiëren release in dossier database (4)
  - Aanmaken rollen en properties (5)
  - Performance en prerequisites (6)
  - Completed properties (7)
  - Triggers (8)
  - Informatie t.b.v. rollen (9)
  - Commit Dossier (10)
  - Always (11)

Uitleg:

1. g_read_settings

Inlezen van het bestand 'settings.xml' dat behoort bij dit proces. Er wordt een XML parse uitgevoerd van deze file. Tijdens deze parse worden de naam, evt. key en de waarde van een setting gelezen en met behulp van de functie 'g_set_initial_value' de juiste variabele van een waarde voorzien. 'g_set_initial_value' is voor elk proces specifiek omdat hierin de namen van de te vullen variabelen hard gecodeerd staan; toevoegen van een variabele aan 'setting.xml' betekent dus aanpassen van 'g_set_initial_value'!

2. g_init

Initialisatie. Dit houdt hier in het inlezen van het bestand met foutmeldingen. Het bestand wordt gescand met behulp van pattern-matching: per melding worden het nummer en de bijbehorende melding gelezen en deze op een shelf ('g_s_error_messages') gezet met als key het nummer en als inhoud de inhoud van de boodschap.

3. Voorbereiding

Voordat begonnen wordt met de daadwerkelijke inhoud van de release, moet aan een aantal voorwaarden zijn voldaan en moet een aantal functies worden uitgevoerd.
In de eerste plaats moet worden gecontroleerd of de aanroepende server geautoriseerd is om een release te maken (g_check_host). In de settings-file staat aangegeven welke server dit zou moeten zijn.
Nu wordt het betreffende element uit de database gelezen (g_get_element). Het id van het betreffende element wordt in het HttpRequest meegestuurd.
Vervolgens wordt de release aangemeld in de MDA-database (g_create_release). Benodigde parameters zijn het id van de unit-of-study en de naam van de release. Deze informatie is via het HttpRequest binnengekomen.

Nu wordt gekeken welke media beschikbaar zijn om uiteindelijk in te kunnnen publiceren (g_read_media). De id's (tevens naam) van deze media worden als keys op de shelf 'g_s_valid_media' opgeslagen.
Volgende stap is het verzamelen van de elementen die in deze release voorkomen en deze opslaan (g_searchrefs; zie beneden).
Als laatste stap in de voorbereiding worden de overgebleven media opgeslagen bij deze release; dit zijn dus de media waarnaar gepubliceerd kan worden voor deze release.

3.1 g_searchrefs

Deze functie zoekt de toplevel elementen op die in deze release meegenomen moeten worden. Dit gebeurt door het veld met de naam 'Structure' te scannen met behulp van find-rules. In dit veld staan alle referenties naar toplevel elementen, die deze unit-of-study bevat, in XML formaat opgeslagen. Ook staat bij elke referentie aangegeven welke regels voor het maken van een release moeten worden gebruikt (version-use en (use-)version) en welk element-type het hier betreft.
Deze gegevens worden gelezen door de find-rules en vervolgens wordt gekeken of het element een MSE betreft of een ander toplevel element.

- Wanneer het element een MSE blijkt te zijn wordt gekeken voor welke media dit element in de database aangemeld staat. Dit zijn de enige media waarnaar straks gepubliceerd kan worden. Dit zijn dus de enige media die over mogen blijven op de shelf 'g_s_valid_media'; alle andere media worden van deze shelf verwijderd. Wanneer dit gebeurd is, wordt het element aangemeld voor deze release; dit houdt in dat het element-id, de version-use en de use-version voor deze release worden vastgelegd.
- Wanneer het element geen MSE blijkt te zijn (en dus een ander toplevel element is) dan wordt gekeken of dit element daadwerkelijk in de database staat. Als dit het geval blijkt te zijn, wordt de meest geschikte versie van het element voor deze release gezocht volgens de aangegeven regels (use-version en version-use) met behulp van een serie views. Welke view gebruikt moet worden is afhankelijk van de version-use van het element. Als de juiste view is gekozen, levert deze het database-id van het meest geschikte element. Wanneer dit gebeurd is wordt het element aangemeld voor deze release; dit houdt in dat het element-id, de version-use en de use-version voor deze release worden vastgelegd.

N.B. We hebben de op dit moment meest geschikte versies dus nu vastgelegd. Wanneer een publicatie gemaakt gaat worden op basis van een release is dus eenduidig vastgelegd welke versies van de elementen gebruikt moeten worden en kunnen we er dus zeker van zijn dat de verschillende publicaties van exact dezelfde bronnen afkomstig zijn, ook al zijn er inmiddels nieuwere versies van sommige elementen.

4. Kopiëren release in dossier database

De release informatie moet nu ook aangemaakt worden in de dossier database (id van release, datum,naam, unit-of-study, wuid, versie en media). Aangezien ebrelease geen toegang heeft tot de dossier database zal dit moeten gebeuren via de ebdos functie commando: M_CREATE_RELEASE). Daarom wordt alle informatie als parameters doorgegeven aan de ebdos server component behorende bij de versie van de unit-of-study (zie 'g_launch_server_app'). Momenteel is dit altijd versie 1.0. Nadat de release informatie succesvol in de dossier database is overgenomen, vindt de commit van de release informatie in de MDA database plaats. Dit moet zo vroeg in het proces gebeuren omdat de ebdos server componenten voor de verdere verwerking (lees)toegang moeten hebben tot de release tabel in de MDA database. Dit betekent dat bij het optreden van fouten de release in de MDA vanaf dit punt expliciet opgeruimd moet worden (verloopt niet meer automatisch via een transactie).

5. Aanmaken rollen en properties

Als volgende stap roept de ebrelease component een functie aan voor het genereren van de rollen en de properties in EML via een ebdos server component. Het commando is M_ROLES_PROP. Ook hier wordt de versie van de ebdos component gebruikt behorende bij de unit-of-study versie; zie vorige stap.

6. Performance en Prerequisites

Als volgende stap worden de performance en prerequisite properties aangemaakt via een aanroep naar de ebdos server component met respectievelijk de M_PERFORMANCE_PROP en M_PREREQUISITE_PROP commando's.
De performance en prerequisite blob's kunnen niet rechtstreeks gevonden worden. Hiervoor worden de recursieve functies 'g_search_prequisite' en 'g_search_learning_objective' gebruikt. Beide functies scannen het eerder (in 3.1) vermelde database veld met de naam 'structure'. Indien een learning-objective of prerequisite gevonden is, en deze nog niet verwerkt is, wordt de boven beschreven ebdos functie aangeroepen. Indien een activiteit wordt gedetecteerd, zal de 'g_search_prequisite' of 'g_search_learning_objective' functie recursief worden aangeroepen voor deze activiteit, omdat activiteiten zelf ook prerequisites en/of learning-objectives kunnen bezitten.

7. Completed properties

Voor iedere activiteit moet tijdens het afspelen worden bijgehouden of deze al dan niet is afgerond door een gebruiker. Hiervoor wordt in het dossier een boolean property aangemaakt. De opdracht hiertoe wordt vanuit het ebrelease programma gegeven. Dit verloopt via de functie 'g_search_activity' die door middel van het scannen van het database veld met de naam 'structure' (zie 3.1) alle actitiviteiten zoekt en voor deze activiteiten het command M_COMPLETED_PROP geeft aan de overeenkomstige ebdos server component.

8. Triggers

Via de functie 'g_process_notifications' worden de notificaties en alle triggers in de dossier database aangemaakt. Triggers zijn die EML constructies die automatische reacties beschrijven op een gebeurtenis. Voorbeelden van triggers zijn: 'Notification' en 'Change-property' na een 'When-property-value-is-set'.
De functie 'g_process_notifications' roept de ebdos functie M_TRIGGERS aan voor die EML toplevel elementen die triggers bevatten. Dit is tijdens het importeren al bepaald en vastgelegd in de MDA en dus middels een query gemakkelijk op te vragen.

9. Informatie t.b.v. rollen

Per rol kan in EML informatie over de rol worden vastgelegd. Deze informatie is van belang voor de gebruiker zodat hij/zij kan bepalen wat van haar/hem verwacht moet worden bij het vervullen van die rol tijdens het afspelen. Aangezien deze informatie getoond wordt voordat de gebruiker de keuze voor een medium, presentatiestijl en taal heeft kunnen maken zal de weergave van deze informatie neutraal zijn ten opzichte van deze aspecten. Dit betekent dat we de informatie nu al kunnen genereren. Omdat de keuze voor een rol altijd in de webbrowser gemaakt moet worden, kan deze informatie dus in HTML formaat worden opgeslagen en gepresenteerd. Als eerste stap wordt een directory genaamd 'neutral' aangemaakt op de web server (de plaats van de directory staat in de database opgeslagen; in de huidige situatie wijst dit dus naar de web server). Vervolgens wordt het commando M_ROLE_INFO gegeven aan de juiste ebdos server component.

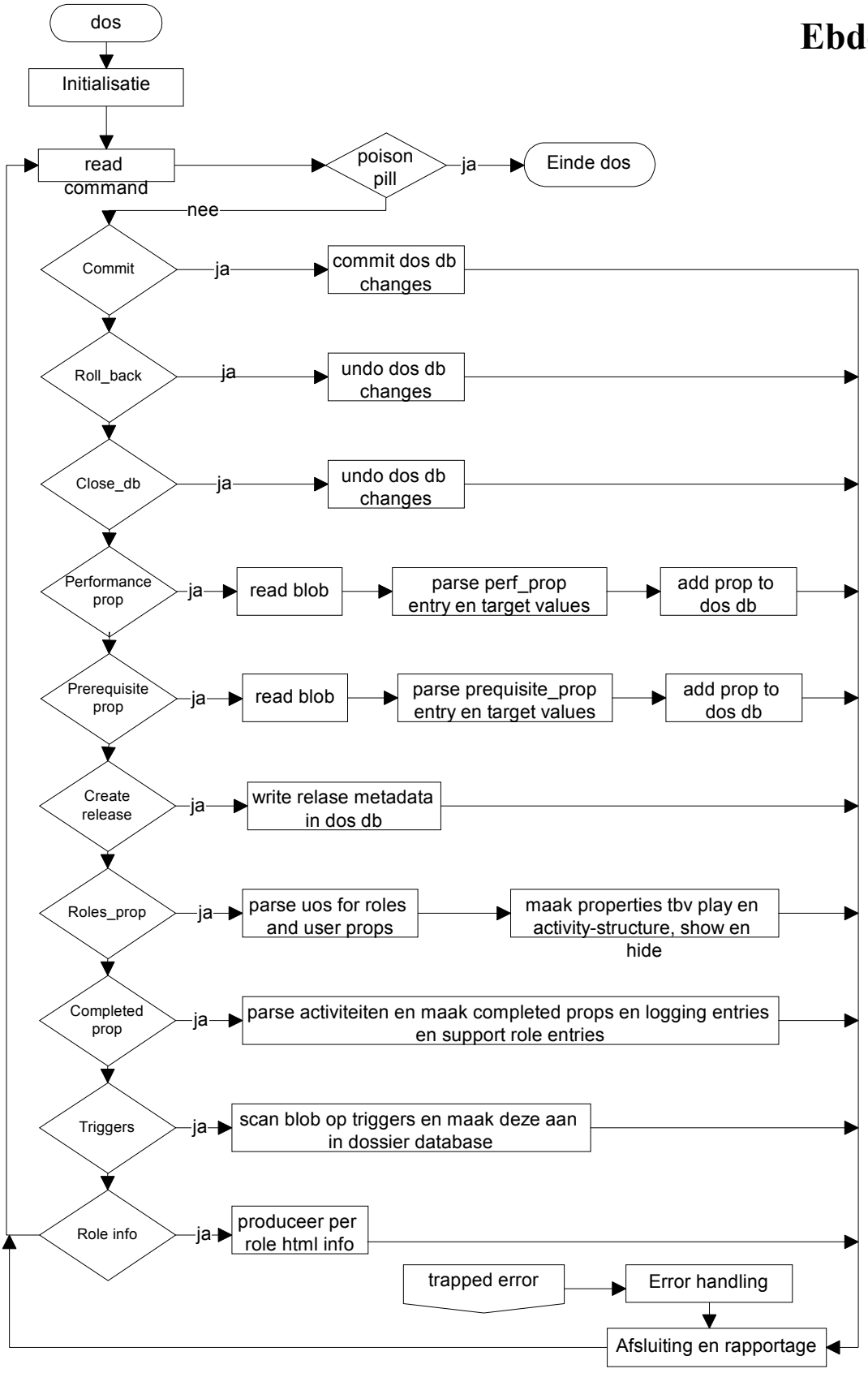
10. Commit Dossier

Nu wordt het commando aan de ebdos component gegeven om de wijzigingen in de dossier database permanent te maken (commit). Momenteel wordt dit commando slechts aan één ebdos component gegeven! (dus in principe fout). Tenslotte wordt ook de connectie met de MDA database gesloten.

11. Always

Het resultaat van het release proces wordt nu als HTTP response teruggegeven.

Ebdos


dos
Initialisatie
read command
poison pill
ja
Einde dos
nee
Commit
ja
commit dos db changes
Roll_back
ja
undo dos db changes
Close_db
ja
undo dos db changes
Performance prop
ja
read blob
parse perf_prop entry en target values
add prop to dos db
Prerequisite prop
ja
read blob
parse prequisite_prop entry en target values
add prop to dos db
Create release
ja
write relase metadata in dos db
Roles_prop
ja
parse uos for roles and user props
maak properties tbv play en activity-structure, show en hide
Completed prop
ja
parse activiteiten en maak completed props en logging entries en support role entries
Triggers
ja
scan blob op triggers en maak deze aan in dossier database
Role info
ja
produceer per role html info
trapped error
Error handling
Afsluiting en rapportage

Ebdos

Globale structuur:

- ServiceInitialize
  - *g_read_settings (1)*
  - *g_init (2)*
- *ServiceMain (3)*
  - *M_COMMIT (3.1)*
  - *M_ROLL_BACK (3.2)*
  - *M_CLOSE_DB (3.3)*
  - *M_PERFORMANCE_PROP (3.4)*
  - *M_PREREQUISITE_PROP (3.5)*
  - *M_CREATE_RELEASE (3.6)*
  - *M_ROLES_PROP (3.7)*
  - *M_COMPLETED_PROP (3.8)*
  - *M_TRIGGERS (3.9)*
    - *Notifications (3.9.1)*
  - *M_ROLE_INFO (3.10)*
  - *Always (3.11)*

Uitleg:

12. g_read_settings

Inlezen van het bestand 'settings.xml' dat behoort bij dit proces. Er wordt een XML parse uitgevoerd van deze file. Tijdens deze parse worden de naam, evt. key en de waarde van een setting gelezen en met behulp van de functie 'g_set_initial_value' de juiste variabele van een waarde voorzien. 'g_set_initial_value' is voor elk proces specifiek omdat hierin de namen van de te vullen variabelen hard gecodeerd staan; toevoegen van een variabele aan 'setting.xml' betekent dus aanpassen van 'g_set_initial_value'!

13. g_init

Initialisatie. Dit houdt hier in het inlezen van het bestand met foutmeldingen. Het bestand wordt gescand met behulp van pattern-matching: per melding worden het nummer en de bijbehorende melding gelezen en deze op een shelf ('g_s_error_messages') gezet met als key het nummer en als inhoud de inhoud van de boodschap.

14. ServiceMain

Dit is het hoofdonderdeel van deze service. Telkens als een aanroep vanuit ebrelease komt dan wordt hier gekeken welk commando gegeven is. Aan de hand van het gegeven commando gaat de service een aantal opdrachten uitvoeren. Hieronder staat beschreven wat deze commando's inhouden. Wanneer het uitvoeren van een commando beëindigd is (ofwel het is klaar ofwel er is een error opgetreden) dan wordt het resultaat teruggestuurd naar ebrelease. Voor meer uitleg hierover zie paragraaf 3.11: 'Always'.

14.1 M_COMMIT

Geeft het commit-commando aan de dossier database zodat de wijzigingen definitief worden. Verder worden zowel de MDA als de dossier database gesloten als deze nog open zijn.

14.2 M_ROLL_BACK

Sluit de databases die nog open staan zonder dat het commit-commando gegeven wordt, dus de wijzigingen worden teniet gedaan!

14.3 M_CLOSE_THE_DBS

Sluit de databases die nog open staan.

14.4 M_PERFORMANCE_PROP

Wanneer dit commando ontvangen wordt dan moet een learning-objective geparsed worden om te kijken of een performance property moet worden aangemaakt in de database en zo ja, dan moet de property worden aangemaakt.
Het learning-objective met zijn inhoud wordt middels een query uit de MDA database gelezen aan de hand van het database-id van dit element, dat als parameter met het commando is meegestuurd.
Tijdens het parsen worden de benodigde gegevens (id, bedoelde rol, datatype, etc.) verzameld over de property en vervolgens wordt deze property in de database aangemaakt middels een stored procedure (via functie 'g_add_property'). Als tweede stap moeten vervolgens de ingangseisen (entry-value) en/of de doelwaarden (target-value) worden opgeslagen voor de betreffende rollen. Ook dit gebeurt middels een stored procedure (via functie 'g_add_entry_or_target_value').

14.5 M_PREREQUISITE_PROP

Wanneer dit commando ontvangen wordt dan moet een prerequisite geparsed worden om te kijken of een performance property moet worden aangemaakt in de database en zo ja, dan moet de property worden aangemaakt.
Behalve dat het een ander type toplevel element betreft is de verwerking gelijk aan die voor een learning-objective (zie 3.4).

14.6 M_CREATE_RELEASE

Dit commando zorgt ervoor dat de release gekopieerd wordt van de MDA database naar de dossier database. Het kopiëren van de release betekent hier de definitie, dus zaken als id, naam, unit-of-study-versie etc. en niet de toplevel elementen die deel uitmaken van deze release. De benodigde gegevens zijn via parameters ontvangen. Het schrijven van de gegevens in de dossier database gebeurt middels een stored procedure die door de functie 'g_create_dos_release' aangeroepen wordt.

14.7 M_ROLES_PROP

Wanneer dit commando ontvangen wordt dan gaat het proces de rol-definities en property-definities opzoeken en in de database opslaan. Verder worden in deze stap ook nog enkele impliciete properties aangemaakt (properties die niet door de auteur zijn gedefinieerd maar die door het systeem worden gebruikt om en en ander goed te laten verlopen in de player).
Om de benodigde informatie te verkrijgen wordt in deze fase twee keer een parse van de unit-of-study inhoud gedaan. Dit is dus de gehele inhoud van de unit-of-study met uitzondering van alle toplevel elementen, want daar staat een referentie naartoe (zie beschrijving ebImport).

- Tijdens de eerste parsing worden alle rollen en properties opgezocht die door de gebruiker expliciet zijn gedefinieerd en worden enkele impliciete properties aangemaakt die de implementatie van het continue-element in de play verzorgen.
    - Om de gedefinieerde rollen te vinden worden alle elementen binnen het element <Roles> bekeken. Wanneer een rol wordt aangetroffen (element <Role>, <Learner> of <Staff>) dan wordt deze rol middels een stored procedure (met behulp van functie 'g_add_role') in de database opgeslagen. Bij deze rol worden ook enkele additionele gegevens opgeslagen zoals naam en type, waarvan de locatie van de rol-informatie (zie ebrelease (9)) en de parent rol de belangrijkste zijn. Om de parent rol vast te houden op dit punt wordt een shelf gebruikt

(g_s_parent_role_stack) die zich als stack gedraagt, d.w.z. voor elke geopende rol wordt het id achteraan de shelf toegevoegd, voor elke gesloten rol wordt de laatste van de shelf verwijderd. Op deze manier liggen alle geopende rollen eenduidig vast.

- Binnen het <Roles> element kunnen ook de property en property-group definities gevonden worden. Deze staan gedefinieerd als eigenschappen van een rol of als algemene property. Ook properties worden opgeslagen middels een stored procedure (met behulp van functie 'g_add_property'). Bij elke property worden enkele additionele, relevante gegevens opgeslagen zoals datatype en intiële waarde etc. Om de eventuele property-group aan te duiden waarvan deze property deel uitmaakt, wordt ook weer een stack-mechanisme gebruikt (met behulp van 'g_s_parsed_group_stack'). Een property-group wordt op exact dezelfde manier behandeld en opgeslagen als een property.
- Als laatste rol moet eventueel nog de roleplanner worden aangemaakt. Dit is een rol die automatisch wordt aangemaakt mits de auteur één of meerdere staf-rollen heeft gedefinieerd. Is dit niet het geval dan kan ook geen roleplanner bestaan.
- In deze fase wordt ook het element <Play> doorzocht en worden evt. enkele properties aangemaakt om de play tijdens het afspelen goed te laten verlopen. Hiertoe worden in deze fasen properties aangemaakt die aangeven of een sequentie of selectie is afgerond (elementen <Activity-sequence> en <Activity-selection>) en properties aangemaakt wanneer ergens een gebruiker in een bepaalde rol moet aangeven of men verder mag (<Continue><Role-choice>). Wanneer een van deze elementen wordt aangetroffen wordt een teller opgehoogd om een unieke property-naam te kunnen genereren. Deze teller duidt dus de plek van het huidige element binnen zijn parent-element aan. De naam wordt gegenereerd uit het id van het parent-element (variabele 'g_s_structure_id' of 'g_s_play_id') in combinatie met de teller (variabele 'g_c_structure_count' of 'g_c_play_count') met behulp van de macro M_CONTINUE_PROP_ID. Vervolgens wordt de property op de normale wijze ('g_add_property') opgeslagen.

  N.B. Omdat in ebpub de condities gemaakt gaan worden die gebruik gaan maken van deze properties is het dus essentieel dat de tellers die op deze plek gebruikt worden volgens exact dezelfde regels worden opgehoogd als de tellers die in ebpub gebruikt worden om deze property-namen te construeren!!

- Tijdens de tweede parse worden impliciete properties aangemaakt die ervoor moeten zorgen dat het zichtbaar en onzichtbaar maken van delen van de boom evenals delen van de tekst in de player mogelijk is (deze properties houden in de player dus bij of een onderdeel zichtbaar is of niet). Deze zaken worden afgehandeld in de groep SHOWHIDE_PROPERTIES:
  Allereerst worden alle constructies opgezocht die aangeven dat iets conditioneel zichtbaar of onzichtbaar moet zijn in de player. Dit is het geval voor de elementen <Content-type>, <Activity-ref>, <Subactivity-ref> en <Unit-of-study-ref> wanneer het parent element <Show>, <Hide>, <Play>, <Activity-sequence> of <Activity-selection> betreft. Wanneer één van deze constructies is gevonden, moet een aantal zaken gebeuren:
  - Het is mogelijk dat een teller moet worden opgehoogd die de positie van een activiteit of unit-of-study binnen een boom beschrijft. Het gebruik van deze tellers is reeds eerder beschreven.
  - De initiële zichtbaarheid wordt in een variabele (g_s_initial_value) opgeslagen (functie 'g_s_get_attribute').
  - Wanneer het element een <Activity-ref> betreft en deze moet initieel onzichtbaar zijn dan moet een extra property worden aangemaakt (omdat in ebpub een extra conditie moet worden gemaakt) die deze activity zichtbaar maakt op het moment

dat er een notification naar deze activity plaatsvindt. Het hoeft niet zo te zijn dat er daadwerkelijk een notificatie naar deze activiteit plaatsvindt; in dit geval is deze property (en ook de conditie) dus overbodig. Het is echter op deze plaats moeilijk te bepalen of een notificatie gaat plaatsvinden.

- Er zijn situaties mogelijk die de initiële zichtbaarheid die eerder is opgehaald (die dus als zodanig in EML gespecificeerd was) overrulen. Dit wordt nu onderzocht volgens een aantal regels, die in de code staan uitgeschreven (meteen na het schrijven van de notification property) en evt. wordt de nieuwe initiële zichtbaarheid in de variabele (g_s_initial_value) gezet.
- De naam van de impliciete property wordt gegenereerd met behulp van het parent-id en de teller zoals reeds eerder beschreven en wordt in de variabele 'l_s_guid' gezet.
- Nu zijn alle gegevens compleet en wordt de impliciete property in de database geschreven met behulp van een stored-procedure (via functie 'g_add_property').

Er kan nog een ander constructie bestaan die leidt tot het aanmaken van impliciete properties:

- een activity-structure binnen een play
- een sequentie of selectie (die uiteraard binnen een activity-structure staat)
- een play of activity-structure (die door een EML-conditie zichbaar gemaakt kan worden)
- een content-type (dat door een EML-conditie zichbaar of onzichtbaar gemaakt kan worden).

Om deze properties te maken moet eventueel een teller verhoogd worden, moet de naam van de property geconstrueerd worden en vervolgens de property op de gebruikelijke wijze geschreven worden.

14.8 M_COMPLETED_PROP

Wanneer dit commando wordt ontvangen, moet voor een bepaald toplevel element (het id wordt meegegeven als parameter; het betreft hier altijd een activity of subactivity) een impliciete property worden aangemaakt die tijdens het afspelen bijhoudt of een activiteit is afgerond of niet. Om dit te doen wordt de inhoud van dit element uit de database gelezen (functie 'g_s_read_blob') en wordt een parse van dit element gedaan (groep: COMPLETED_PROPERTIES). Verder wordt deze fase gebruikt/misbruikt om een aantal zaken betreffende de activiteiten in de database op te slaan. In feite vinden drie belangrijke activiteiten plaats:

- Alle elementen met de naam <Activity> of <Subactivity> worden opgevangen. Een van de belangrijkste dingen die hier gebeuren is het construeren van de naam van de property. Deze wordt gemaakt door het (unieke) database-id op te halen uit de database (functie 'g_s_get_element_id') en hier nog iets voor te plakken. Dit gebeurt door middel van de macro 'M_COMPLETED_PROP_ID'. Verder wordt de initiële waarde van deze property (deze is van het type boolean) op FALSE gezet (hetgeen aanduidt dat de activiteit niet is afgerond), behalve wanneer het element <Unrestricted> is opgenomen. In dat geval wordt de initiële waarde op TRUE gezet; we gaan er dus van uit dat als afronding zonder beperking gebeurt, de activiteit altijd is afgerond. Nu wordt de property opgeslagen op de gebruikelijke manier.
- Om de logging van activiteiten in de player beschikbaar te maken moeten de activiteiten in een aparte tabel in de dossier-database worden opgenomen. Omdat in deze parsing-fase alle activiteiten voorbij komen die gebruikt gaan worden, wordt deze fase gebruikt om deze tabel te vullen. Elke activiteit wordt dus met behulp van een stored-procedure (via functie 'g_add_activity_logging') opgeslagen in een aparte tabel.
- Wanneer de activiteit een support-activity (een activiteit waarbij het mogelijk is in andermans dossier te lezen en te schrijven) blijkt te zijn dan moet dit in een aparte

table worden opgeslagen. Dat een activiteit een support-activity is, weten we door het aanwezig zijn van het element <Support-role>. De auteur kan hier ook aangeven voor welke rol(len) deze activiteit een support-activity is. Deze rollen worden opgeslagen op de shelf 'g_s_support_roles'. Als laatste stap tijdens het verwerken van een activiteit wordt gekeken of het een support-activity betreft en wordt voor elke rol een record in een tabel in de dossier-database aangemaakt met behulp van een stored-procedure (via functie 'g_add_support_role').

14.9 M_TRIGGERS

Wanneer dit commando wordt ontvangen moet de inhoud van een toplevel element worden doorzocht op triggers en moeten deze worden opgeslagen in de database. Triggers zijn EML constructies die een reactie beschrijven op een gebeurtenis.
Er wordt dus een SGML parse uitgevoerd (Groep: TRIGGERS) op de inhoud van een element waarvan het database-id is meegegeven met het commando. De inhoud van het document wordt uit de database gelezen (functie 'g_s_read_blob') en aan de parser aangeboden.
In groep TRIGGERS staan alle constructies genoemd die een trigger kunnen beschrijven:

- Completed><When-property-value-is-set>: Deze constructie geeft dat de betreffende activiteit is afgerond wanneer een bepaalde property een (bepaalde) waarde heeft. We parsen eerst nog een stuk verder (naar binnen) om het id van de property te bepalen en op te slaan (in variabele 'g_s_when_property_value_is_set') en de evt. waarde van de property te bepalen en op te slaan (in variabele 'g_s_property_value'). Hierna wordt de trigger opgeslagen middels een stored-procedure (via functie 'g_add_db_trigger').

N.B. Deze constructie kan op twee manieren worden opgebouwd, nl. door aan te geven welke waarde de property moet krijgen om de trigger te laten afgaan OF door geen waarde in te vullen (waarmee je aangeeft dat de property een waarde moet krijgen). Afhankelijk van de gekozen constructie ('g_s_property_value' is ingevuld of niet) worden de parameters voor de functie ingevuld. <

- <Completed><Change-property-value>: Als een activiteit is afgerond moet een bepaalde property een bepaalde waarde krijgen. Hiertoe wordt in het element <Activity> reeds bepaald welke property (het af zijn van een activiteit kan immers beschreven worden als het wijzigen van de completed-property van deze activiteit van FALSE naar TRUE) het afgaan van de trigger veroorzaakt. Hiertoe wordt eerst het database-id van de activiteit opgehaald ('g_s_get_element_id') en vervolgens met behulp van de macro 'M_COMPLETED_PROP_ID' en het database-id het id van de property geconstrueerd en wordt dit opgeslagen in een variabele ('g_s_completed_id') voor later gebruik. In het element <Change-property-value> wordt de trigger daadwerkelijk opgeslagen. Eerst worden, door een stukje verder (naar binnen) te parsen, de naam en waarde van de te wijzigen property bepaald en opgeslagen (variabelen: 'g_s_trigger_property_id' en 'g_s_trigger_property_value') en worden de evt. gespecificeerde notificaties opgeslagen (zie 3.9.1). Daarna wordt de constructie in de database opgeslagen middels een stored-procedure (via functie 'g_add_db_trigger').
  N.B. Voor elke notificatie die aan deze trigger verbonden is wordt een record in de database aangemaakt. Vandaar dat de shelf 'g_s_notifications' van voor naar achteren wordt afgelopen en telkens een aanroep van de functie 'g_add_db_trigger' plaatsvindt, waarbij alle parameters dus dezelfde inhoud hebben en alleen de inhoud van 'g_s_notifications' anders is.

14.9.1 Notifications

Wanneer we het element <Notification> tegenkomen moet de notificatie in de database worden opgeslagen. Eerst moeten we, door verder (naar binnen) te parsen nog enkele gegevens ophalen, nl:

- De rol(len) aan welke de notificatie gestuurd moet worden. Aangezien dit meer dan één rol kan zijn worden de id's van de rollen op een shelf geplaatst ('g_s_notification_roles').
- Een eventuele activiteit als link met de notificatie moet worden meegestuurd. Het database-id van deze activiteit wordt gebruikt en wordt met behulp van de macro 'M_GET_ID' opgehaald en in de variabele 'g_s_notification_activity' opgeslagen.
- Het eventuele onderwerp van de notificatie wordt gelezen en in de variabele 'g_s_notification_subject' opgeslagen.

Wanneer we al deze gegevens hebben kan de notificatie worden opgeslagen. Dit opslaan gebeurt in twee stappen:

- Het opslaan van de notificatie zelf, inclusief link-activiteit en onderwerp. Dit gebeurt met behulp van een stored-procedure (via functie 'g_s_add_notification').
- Het opslaan van de rollen waar deze notificatie naar verstuurd moet worden. Voor elke rol wordt een nieuw record aangemaakt, waarin het rol-id gekoppeld wordt aan het notificatie-id. Om dit te bewerkstelligen wordt de shelf met rollen ('g_s_notification_role') van voor naar achteren doorlopen en wordt voor elke rol een record aangemaakt middels een stored-procedure (via functie 'g_s_notification_roles').

14.10 M_ROLE_INFO

Dit commando impliceert dat de HTML pagina's gemaakt moeten worden die een beschrijving van de rollen bevatten. Om dit te doen wordt eerst de locatie opgeslagen waar de bestanden gezet moeten worden ('g_s_neutral_dir'). Daarna wordt de inhoud van de betreffende unit-of-study (het id is als parameter met het commando meegestuurd) opgehaald met behulp van de functie 'g_s_read_blob'. Vervolgens wordt een SGML parse gedaan van deze inhoud die de groep 'INFORMATION_FOR_ROLE' gebruikt.
Voor elke rol (element <Learner>, <Staff> of <Role>) wordt een pagina gemaakt, ongeacht of informatie is gespecificeerd. Dit omdat dit gemakkelijker is voor het role-switch tool dat bij de afspeler hoort. Vervolgens wordt de informatie in de HTML bestanden geschreven. Het element <Information-for-role> bevat elementen van het type <%Extra-p>. Van dit type zijn niet alle elementen vertaald, alleen die typen die statisch van aard zijn (dus bijvoorbeeld wel <Emphasis>, maar niet <Set-property-value>).

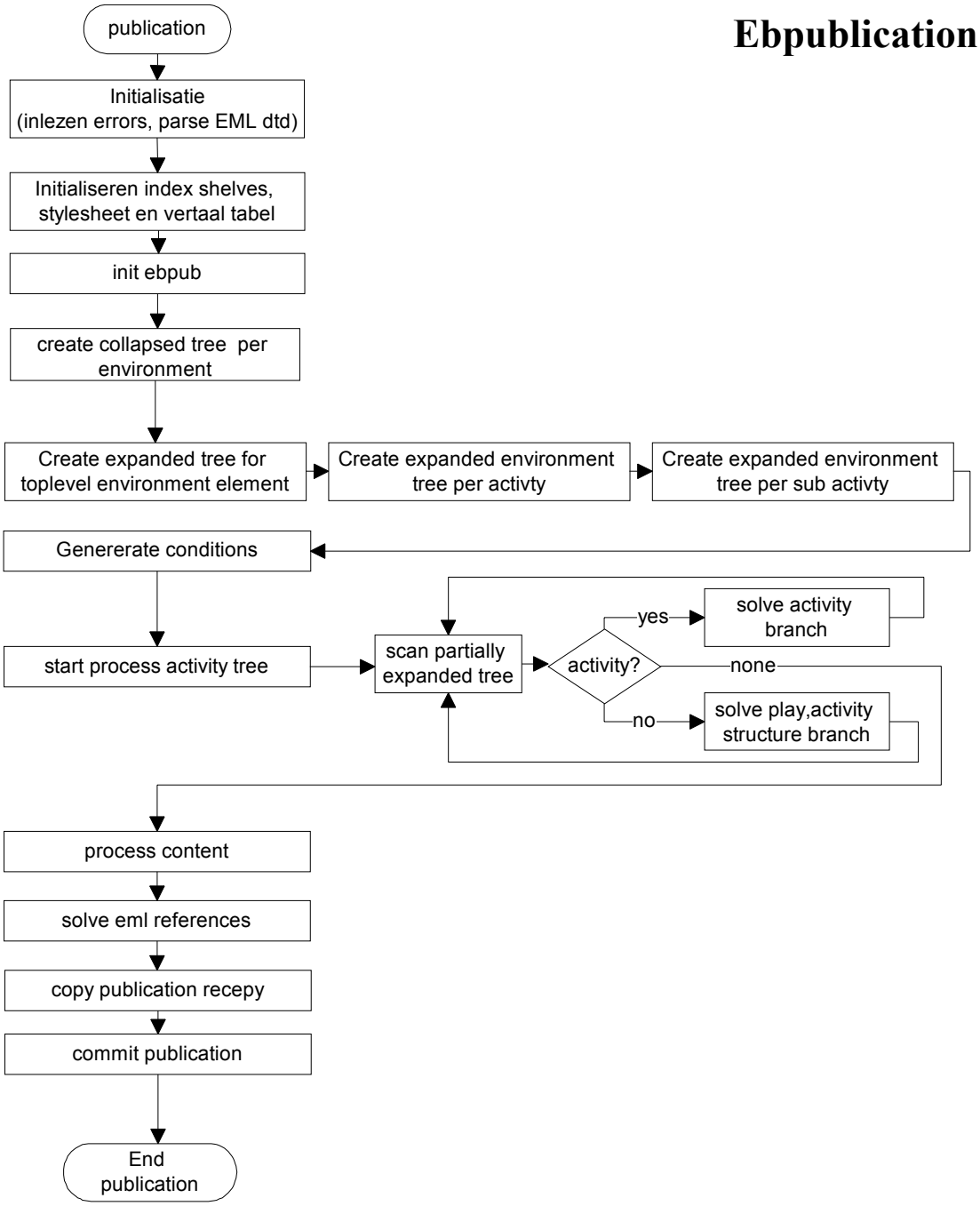
14.11 Always

Dit stukje code wordt altijd uitgevoerd. Het betreft hier de functie 'g_s_generate_xml_response', die de resultaten terugstuurt naar ebrelease, het aanroepende proces.
'g_s_generate_xml_response' onderkent drie verschillende situaties en handelt hiernaar:

- Er zijn geen fouten opgetreden (shelf 'g_s_error_log' is leeg) en er is tekst/inhoud als parameter met de functie meegestuurd. De <OK>-tag wordt teruggestuurd samen met de tekst/inhoud.
- Er zijn geen fouten opgetreden (shelf 'g_s_error_log' is leeg) en er is geen tekst/inhoud als parameter met de functie meegestuurd. De <OK>-tag wordt teruggestuurd zonder tekst/inhoud.
- Er zijn fouten opgetreden. De foutboodschappen worden teruggestuurd, omvat door de <ERROR>-tag.

Door op deze manier het resultaat terug te sturen weet de unit-of-study-manager (de applicatie wordt ook wel 'Runplanner' genoemd) welk soort melding op het scherm te zetten.

Ebpublication



Ebpublication

Globale structuur:

- ServiceInitialize
  - *g_read_settings (1)*
  - *g_init (2)*
- ServiceMain
  - *Voorbereiding (3)*
    - Indexen leegmaken
    - g_parse_language
    - g_parse_style_sheet
    - g_check_host
  - *Initialisatie van de publicatie (4)*
  - *Maken van de losse environment-bomen (5)*
  - *Environment-bomen koppelen aan activiteiten (6)*
  - *Condities genereren (7)*
  - *Opbouwen activiteitenboom (8)*
    - *g_build_tree (8.1)*
  - *Inhoud verwerken (9)*
  - *Referenties oplossen (10)*
  - *Kopiëren bestanden (11)*
  - *Vastleggen publicatie (12)*
  - Always (13)

Uitleg:

15. g_read_settings

Inlezen van het bestand 'settings.xml' dat behoort bij dit proces. Er wordt een XML parse uitgevoerd van deze file. Tijdens deze parse worden de naam, evt. key en de waarde van een setting gelezen en met behulp van de functie 'g_set_initial_value' de juiste variabele van een waarde voorzien. 'g_set_initial_value' is voor elk proces specifiek omdat hierin de namen van de te vullen variabelen hard gecodeerd staan; toevoegen van een variabele aan 'setting.xml' betekent dus aanpassen van 'g_set_initial_value'!

16. g_init

Initialisatie. Het eerste wat gebeurt, is het inlezen van het bestand met foutmeldingen. Het bestand wordt gescand met behulp van pattern-matching: per melding worden het nummer en de bijbehorende melding gelezen en deze op een shelf ('g_s_error_messages') gezet met als key het nummer en als inhoud de inhoud van de boodschap. Als tweede vindt een parse van de DTD plaats, omdat dit maar één keer hoeft te gebeuren en hierdoor in een later stadium tijd wordt bespaard.

17. Voorbereiding

Voordat we daadwerkelijk gaan beginnen met het maken van de publicatie moet eerst nog een aantal dingen gebeuren, waarvan de belangrijkste nu genoemd worden.
Ten eerste moeten de indexen leeggemaakt worden. De indexen zijn vijf ('g_s_xxx_index') shelves die de verschillende manieren van indexeren representeren. Hierover meer in de beschrijving van ebdos. Daarna moet een vertaaltabel uit de database worden opgehaald, die tijdens het publiceren de vertaling van alle taalafhankelijke elementen verzorgt. Deze tabel staat in XML-vorm in de MDA-database opgeslagen en wordt met behulp van de

functie ‘g_parse_language’ opgehaald. Deze functie haalt de vertaaltabel op voor het juiste medium en de juiste taal en zet vervolgens met behulp van matching-rules de elementen op een shelf (‘g_s_language_element’ met de elementnaam als key en de vertaling als value). Op dezelfde manier moet ook een ‘stylesheet’ worden opgehaald voor het juiste medium en de juiste stijl. Dit ‘stylesheet’ definieert voor een bepaald element op welke wijze welke zaken in de publicatie terecht dienen te komen. Op deze manier kan de wijze van publiceren worden gewijzigd zonder dat daar een wijziging in de code voor nodig is. Dit ophalen gebeurt met behulp van de functie ‘g_parse_style_sheet’. Deze functie haalt ook de juiste ‘stylesheet’ (in XML formaat) op enzet de onderdelen vervolgens op de shelf ‘g_s_styles’. Nu dit is gebeurd, kunnen we beginnen met het publicatieproces, dat hieronder wordt beschreven.

18. Initialisatie van de publicatie

Het publicatieproces bestaat, net zoals het maken van een release, uit (minimaal) twee processen. In dit geval is ebpublication het aansturende proces en gebeurt het meeste werk in ebpub, het proces dat aangestuurd wordt. Voordat we ebpub kunnen gaan gebruiken moet in dat proces eerst een aantal variabelen worden gevuld. Om dit in gang te zetten wordt het commando M_INITIALISE_PUB naar ebpub gestuurd.

19. Maken van losse environment-bomen

Omdat environments genest kunnen zijn, maar ook op het hoogste niveau kunnen voorkomen (in tegenstelling tot bijvoorbeeld een knowledge-object, dat altijd in een environment zit) wordt voor elk environment met zijn objecten een apart boompje gebouwd, dat los wordt opgeslagen. Bestandsnamen hebben het formaat ‘env_tree_xxx.xml’. Als in een environment een ander environment voorkomt, wordt hiernaar met behulp van een ‘entity’ verwezen en wordt dit dus niet verder uitgeschreven in de boom. Om dit te kunnen bewerkstelligen worden met behulp van de functie ‘g_get_element_id’ alle elementen opgehaald in de betreffende release van het type ‘Environment’. Vervolgens wordt voor elk gevonden element het commando M_ENVIRONMENT_TREE naar ebpub gestuurd samen met het id van het betreffende element.

20. Environment-bomen koppelen aan activiteiten

Nu moeten de specifieke environment-bomen gemaakt worden die bij een activiteit, subactiviteit, sequentie, selectie of de unit-of-study behoren. Het maken van deze bomen en de resulterende bestanden (‘act_env_xxx.xml’) wordt verzorgd door ebpub (commando M_ENVIRONMENT_INSTANCE). Om deze specifieke bomen op te bouwen wordt gebruik gemaakt van de algemene boompjes die in de vorige stap beschreven zijn. Voor verdere uitleg zie beschrijving ebpub.
Het maken van deze bomen (dat wil zeggen het aansturen van ebpub) gebeurt in drie fasen:

- Om te beginnen wordt opdracht gegeven om de unit-of-study te parsen. Hieruit zullen dus de environment-bomen voortkomen die behoren bij de unit-of-study in het algemeen, de selecties en de sequenties (deze elementen komen niet als referenties in de unit-of-study voor; voor meer hierover zie beschrijving ebimport).
- Daarna wordt de opdracht gegeven om alle activiteiten die in deze release voorkomen te parsen. Op deze manier worden alle bomen gemaakt die bij de activiteiten horen. De id’s van de te parsen activiteiten worden opgezocht met behulp van de functie ‘g_get_element_id’.
- Daarna wordt de opdracht gegeven om alle subactiviteiten die in deze release voorkomen te parsen. Op deze manier worden alle bomen gemaakt die bij de subactiviteiten horen. De id’s van de te parsen subactiviteiten worden opgezocht met behulp van de functie ‘g_get_element_id’.

21. Condities genereren

In deze fase van het proces worden de condities in de unit-of-study opgezocht en verzameld. Het commando M_CONDITIONS wordt gegeven aan ebpub en deze doorzoekt de hele unit-of-study naar constructies die een conditie bevatten/nodig hebben. Er zijn grofweg twee soorten condities te onderscheiden:

- Condities die door de auteur achter de <Conditions> tag zijn ingevuld. Deze condities zijn vrij eenvoudig te parsen.
- Condities die niet door de auteur zijn ingevuld maar die een gevolg zijn van een constructie die de auteur heeft opgebouwd. Bijvoorbeeld een activiteit die in een play staat. Er zijn situaties dat deze pas zichtbaar mag worden als de vorige activiteit is afgerond. Deze condities zijn moeilijker te detecteren.

De condities worden op een aantal shelves opgeslagen die in ebpub blijven staan.
Dit is eigenlijk niet helemaal 'clean'. Volgens de methode die in de andere (sub)processen wordt gehanteerd, zou ebpub 'stateless' moeten zijn, wat nu niet het geval is.
In enkele vervolgstappen van het publicatieproces kunnen deze condities nu vrij snel worden opgespoord en worden verwerkt.

22. Opbouwen activiteitenboom

Volgende fase in het publicatieproces is het opbouwen van de activiteitenboom. Het resultaat van dit proces is een XML bestand met de naam 'activities.xml'. In dit bestand staat de boom uitgedrukt in de elementen <Tree>, dat het beginpunt aangeeft van een stuk boom (hier wordt momenteel in de player niets speciaals mee gedaan), en (hoofdzakelijk) <Node>. Deze elementen bevatten behalve gegevens als naam, uniek id etc. ook de condities onder welke deze node zichtbaar of onzichtbaar wordt. Om tot deze activiteitenboom te komen wordt een parse uitgevoerd op de gehele unit-of-study (groep: XML_ACTIVITY_TREE), die de beginpunten van zo'n boom opzoekt. In de huidige implementatie bestaan twee verschillende beginpunten: <Play> en <Unit-of-study>. Wanneer zo'n beginpunt wordt gevonden wordt met behulp van de functie 'g_build_tree' (een stuk van) de boom opgebouwd. Deze functie wordt hieronder nader uitgelegd. Ook worden activiteiten die niet in een structuur zijn geplaatst (maar los onder <Content>) opgevangen en opgeslagen voor latere verwerking (in het geval dat unit-of-study het beginpunt is).
Als de boom is opgebouwd, staat deze opgeslagen op de shelf 'g_s_xml_activity_tree'. Deze wordt vervolgens aan de functie 'g_activity_tree' aangeboden die ervoor zorgt dat er een syntactisch correct XML document uit gegenereerd wordt.

22.1 g_build_tree

Dit is een recursieve functie die de activiteitenboom opbouwt. De functie scant de binnengekomen stream (die de boom voorstelt die gedeeltelijk is opgelost en waarschijnlijk nog wat verwijzingen bevat) en zoekt de elementen op die een verwijzing bevatten (deze beginnen met het teken '@') en stuurt alle andere elementen gewoon door. De functie breekt verwijzende elementen op in losse onderdelen, zoals type, id etc. Vervolgens wordt gekeken om welk type het hier gaat. Afhankelijk van het type wordt een aantal acties uitgevoerd:

- In geval van een play of activity-structure: als eerste worden de attributen van dit element gescand en wordt het id opgeslagen in een variabele voor later gebruik. Dan wordt met behulp van de functie 'g_launch_server' opgezocht welk proces bij deze EML-versie van dit element hoort. Dan worden nog enkele parameters gezet voor het uitvoeren van het betreffende ebpub-proces. Deze parameters zijn:
  - het id van de unit-of-study waar dit element deel van uitmaakt
  - het id van de release waar de publicatie van gemaakt wordt
  - de directory waarnaar gepubliceerd wordt

  - het ebpub-commando dat moet worden uitgevoerd
  - het id van de activity-structure waar dit element deel van uitmaakt
  - het id van de play waar dit element deel van uitmaakt.
- In het geval van een actvity: de attributen van dit element worden gescand en de attributen die van belang zijn worden in variabelen opgeslagen. Met behulp van deze variabelen kan het betreffende element worden opgezocht in de mda-database (via functie 'g_get_pct_element'). Dan wordt met behulp van de functie 'g_launch_server' opgezocht welk proces bij deze EML-versie van dit element hoort. Dan worden nog enkele parameters gezet voor het uitvoeren van het betreffende ebpub-proces. Deze parameters zijn:
  - het database-id van dit element in de release-tabel van de mda-database
  - het id van de release waar de publicatie van gemaakt wordt
  - de directory waarnaar gepubliceerd wordt
  - het ebpub-commando dat moet worden uitgevoerd
  - het id van de property die gebruikt moet worden voor de show/hide conditie
  - het id van het element
  - de naam van de environment-file die bij dit element hoort
  - het id van de activity-structure waar dit element deel van uitmaakt
  - het type element dat verwerkt moet worden
  - het id van de play waar dit element deel van uitmaakt.

Nu wordt het ebpub-proces aangeroepen (met behulp van functie 'g_issue_command') waarvoor al deze parameters zijn gezet. Dit proces gaat het betreffende element (dus een play, een activity-structure of een activity) verwerken en levert een boom op die weer een stukje verder is opgelost. Deze boom wordt recursief aan 'g_build_tree' aangeboden en dus wordt weer een volgend stukje van de boom opgelost.

23. Inhoud verwerken

In dit stadium wordt alle inhoud verwerkt en per toplevel element in een tijdelijk bestand ('env_xxx.tmp') opgeslagen. Dit betekent dat de inhoud voor zowel activiteiten als environment objecten etcetera gegenereerd wordt. De bestanden worden gecodeerd in XHTML formaat. Op deze manier kunnen alle statische onderdelen (zoals een stuk tekst, een kopje etc.) gewoon met behulp van HTML tags gecodeerd worden en worden alle dynamische onderdelen (zoals het zetten van een property of een special) met behulp van XML tags gecodeerd zodat ze gemakkelijk door de player kunnen worden opgepakt en verwerkt.
Eerst wordt het juiste proces opgezocht dat bij deze EML-versie van dit element (hier de unit-of-study, omdat alle inhoud hier verwerkt wordt) hoort. Dan wordt een aantal parameters gezet en dan wordt ebpub aangeroepen (commando M_GENERATE_CONTENT) om de daadwerkelijke verwerking te verzorgen.
Wanneer de verwerking klaar is, wordt een aantal shelves teruggestuurd die verschillende indexen voorstellen. Deze indexen moeten nu worden toegevoegd aan een aantal shelves die alle indexen bijhouden voor latere verwerking. Dit gebeurt met behulp van de functie 'g_add_index_from_stream'.

24. Referenties oplossen

Tijdens het verwerken van de inhoud (zie stap 9) kan een verwijzing worden aangetroffen (bijv. EML-ref) of kan een opgenomen inhoudsopgave worden aangetroffen. Op dat moment is vaak nog niet bekend waarnaar deze verwijzing verwijst (omdat we nog niet op dat punt in de verwerking zijn aangekomen). Daarom wordt deze verwijzing, gemarkeerd door een aantal tekens, op dat moment onopgelost in het bestand geschreven. In deze fase is alles verwerkt en kunnen we dus ook de referenties oplossen. Daartoe wordt de functie 'g_solve_references' aangeroepen. Deze functie doorloopt alle tijdelijke bestanden en lost

de referenties op met behulp van de index-shelves die we hebben bijgehouden (zie ook stap 9). Dit levert de definitieve inhoudsfiles op (‘env_xxx.xml’).

25. Kopiëren bestanden

Allereerst moet bepaald worden op welke directory de bestanden definitief geplaatst moeten worden. Dit gebeurt met behulp van de functie 'g_create_pubdir'. Deze functie zorgt er ook voor dat de directory is aangemaakt. Dan worden de zojuist gegenereerde bestanden gekopieerd naar de betreffende directory met behulp van de functie 'g_copy_files_from'. Daarna worden alle MSE's gekopieerd met behulp van de functie 'g_publish_MSE'. De functie 'g_copy_files_for_style' kopieert vervolgens de bestanden die allemaal nodig zijn voor deze stijl (titelplaatjes, css-stylesheets etc.).

26. Vastleggen publicatie

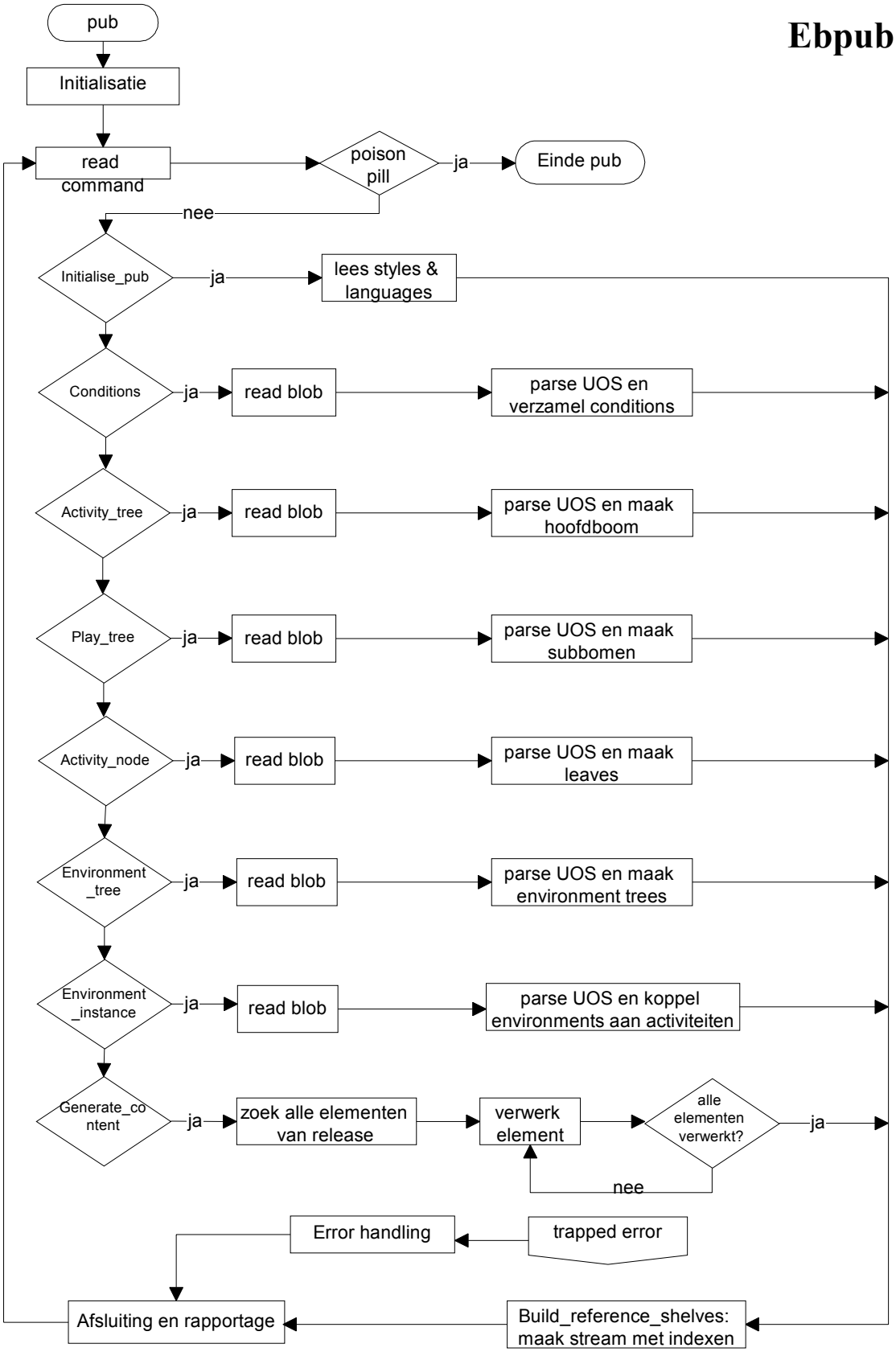
Als al het voorafgaande foutloos is verlopen dan is de publicatie gelukt. We kunnen dan nu de publicatie aanmelden in de dossier database. Hiervoor wordt de functie 'g_add_publication_recipe' gebruikt.

27. Always

Dit stuk code wordt altijd uitgevoerd, ongeacht of het publiceren goed of fout gegaan is. Hier worden de beide databases gesloten, de tijdelijke bestanden opgeruimd en het resultaat van het publicatieproces teruggegeven aan de aanroepende applicatie.

Ebpub

Ebpub


pub
Initialisatie
read command
poison pill
ja
Einde pub
nee
Initialise_pub
ja
lees styles & languages
Conditions
ja
read blob
parse UOS en verzamel conditions
Activity_tree
ja
read blob
parse UOS en maak hoofdboom
Play_tree
ja
read blob
parse UOS en maak subbomen
Activity_node
ja
read blob
parse UOS en maak leaves
Environment _tree
ja
read blob
parse UOS en maak environment trees
Environment _instance
ja
read blob
parse UOS en koppel environments aan activiteiten
Generate_co ntent
ja
zoek alle elementen van release
verwerk element
alle elementen verwerkt?
ja
nee
Error handling
trapped error
Afsluiting en rapportage
Build_reference_shelves: maak stream met indexen

Globale structuur:

- ServiceInitialize
  - *g_read_settings (1)*
  - *g_init (2)*
- *ServiceMain (3)*
  - *[M_INITIALISE_PUB] (3.1)*
  - *[M_CONDITIONS] (3.2)*
  - *[M_ACTIVITY_TREE] (3.3)*
  - *[M_PLAY_TREE] (3.4)*
  - *[M_ACTIVITY_NODE] (3.5)*
  - *[M_ENVIRONMENT_TREE] (3.6)*
  - *[M_ENVIRONMENT_INSTANCE] (3.7)*
  - *[M_GENERATE_CONTENT] (3.8)*
    - [M_UOS_CONTENT]
    - [M_QUESTIONNAIRE_CONTENT]
    - [M_COMMUNICATION_OBJECT_CONTENT]
    - [M_KNOWLEDGE_OBJECT_CONTENT]
    - [M_INDEX_SEARCH_OBJECT_CONTENT]
    - [M_TOOL_OBJECT_CONTENT]
    - [M_ROLE_INFORMATION_OBJECT_CONTENT]
    - [M_ENVIRONMENT_CONTENT]
  - *M_BUILD_REFERENCE_SHELVES (3.9)*
  - *Always (3.10)*

Uitleg:

28. g_read_settings

Inlezen van het bestand 'settings.xml' dat behoort bij dit proces. Er wordt een XML parse uitgevoerd van deze file. Tijdens deze parse worden de naam, evt. key en de waarde van een setting gelezen en met behulp van de functie 'g_set_initial_value' de juiste variabele van een waarde voorzien. 'g_set_initial_value' is voor elk proces specifiek omdat hierin de namen van de te vullen variabelen hard gecodeerd staan; toevoegen van een variabele aan 'settings.xml' betekent dus aanpassen van 'g_set_initial_value'!

29. g_init

Initialisatie. Dit houdt hier in het inlezen van het bestand met foutmeldingen. Het bestand wordt gescand met behulp van pattern-matching: per melding worden het nummer en de bijbehorende melding gelezen en deze op een shelf ('g_s_error_messages') gezet met als key het nummer en als inhoud de inhoud van de boodschap.

30. ServiceMain

Dit is het hoofdonderdeel van deze service. Telkens als een aanroep vanuit ebpublication komt dan wordt hier gekeken welk commando gegeven is. A.d.h.v. het gegeven commando gaat de service een aantal opdrachten uitvoeren. Hieronder staat beschreven wat deze commando's inhouden. Wanneer het uitvoeren van een commando beëindigd is (ofwel het is klaar ofwel er is een error opgetreden) dan wordt het resultaat teruggestuurd naar ebpublication. Na het uitvoeren van ieder commando wordt de MACRO M_BUILD_REFERENCE_SHELVES uitgevoerd. Voor meer uitleg hierover zie paragraaf 3.9.

30.1 M_INITIALISE_PUB

Dit commando heeft tot doel het publicatierecept te initialiseren door middel van een tweetal functies: g_parse_language en g_parse_style_sheet. Beide functies lezen op basis van meegegeven medium, language en style id's XML publicatierecepten uit de MDA database en slaan deze op in shelves zodat ze gedurende het publicatieproces gebruikt kunnen worden.

30.2 M_CONDITIONS

Dit commando vertaalt alle EML condities (impliciet en expliciet) in Jscript-achtige condities. Momenteel is een tweetal concessies gedaan ten aanzien van van deze functie. Ten eerste worden de Jscript condities bewaard op diverse shelves zodat ze later gebruikt kunnen worden. Dit veronderstelt 'state' van ebpub. Dit laatste druist in tegen de filosofie dat er meerdere versies van ebpub kunnen zijn voor meerdere versies van EML. Dit laatste veronderstelt een 'stateless' benadering. Ten tweede zouden de condities eigenlijk in ebdos vertaald moeten worden. Immers deze condities zijn in principe publicatierecept-onafhankelijk. Echter wegens tijdsdruk is een volledige medium-neutrale afhandeling niet bereikt. Verdere analyse is nog noodzakelijk.

In principe worden de condities ongeveer één op één vertaald van EML naar Jscript. Beide systemen werken op basis van Poolse notatie. Bij de vertaling wordt gebruik gemaakt van een stack mechanisme met referents. Iedere parameter van een functie (lees conditie) wordt vertaald in een referent. Bij het parsen van iedere parameter wordt deze ook weer vervangen door een expressie die op zijn beurt weer referents kan bevatten. Op deze manier ontstaat recursief een Jscript expressie waarbij een stack van openstaande referents (de parameters) wordt bijgehouden.

Iedere vertaalde conditie wordt op een van de volgende shelves bewaard waarbij de show en de hide condities apart worden opgeslagen:

- g_s_show_content en g_s_hide_content
- g_s_show_activity en g_s_hide_activity
- g_s_show_uos en g_s_hide_uos
- g_s_show_activity_structure en g_s_hide_activity_structure
- g_s_show_play en g_s_hide_play

Sommige condities zijn niet als zodanig in EML opgenomen. We gebruiken deze condities om workflow die wel in EML (selections en sequences) is gedefinieerd te realiseren. Ook deze gegenereerde condities worden op shelves opgeslagen, te weten:

- g_s_enable_selection
- g_s_enable_sequence
- g_s_enable_activity
- g_s_enable_activity_structure
- g_s_enable_uos

In EML kan worden aangegeven wanneer selecties zijn afgerond. Een sequentie is afgerond als alle elementen in de sequentie zijn afgerond. We gebruiken een expressie om vast te stellen of dit inderdaad het geval is. Deze expressie wordt opgeslagen in één van de volgende shelves:

- g_s_selection_completed
- g_s_sequence_completed

Tot slot kunnen notificaties tot gevolg hebben dat delen van de activiteitenboom zichtbaar moeten worden. Dit gedrag wordt ook vertaald naar condities die worden opgeslagen op:

- g_s_notification_conditions

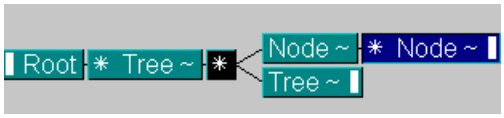
Uit voorgaand overzicht wordt duidelijk dat er eigenlijk drie typen conditie te onderscheiden zijn van verschillend gewicht. Oplopend in prioriteit zijn dit:

1. Condities zoals die in EML zijn opgenomen. De eigenaar van deze condities zijn Activiteiten, Unit-of-Study, Plays en Activity-structures.
2. Condities die voor de workflow zorgen. De eigenaar van deze condities zijn de elementen van een Play en/or Activity-structure.
3. Condities t.b.v. notificaties waarvan de eigenaar een activiteit is.

Voor het ophalen van de diverse condities dienen verschillende keys gebruikt te worden op de diverse shelves. Voor type 1 en 3 wordt altijd de id van het element gebruikt omdat het element ook de eigenaar van de conditie is. Voor type 2 wordt de relatieve positie in de boom gebruikt. De gebruikte id bestaat uit het id van het element plus een teller (g_c_activity_structure_count) die de positie aangeeft die het element in de play of activity-structure heeft.

30.3 M_ACTIVITY_TREE

Leukste procedure van allemaal. Dit commando zorgt ervoor dat de activiteiten boom in XML formaat wordt gezet volgens de volgende DTD:



Figuur 1: ActivityTree DTD

De boom wordt maar ten dele opgelost. Alles, behalve activiteiten en geneste activity-structures, wordt vertaald naar bovengenoemde DTD. Waar vertaling nog niet mogelijk is wordt een zgn. 'token' geplaatst om aan te geven dat de boom hier nog verder moet worden uitgesplitst. Zie hiervoor ebpublication hoofstuk 8. Reden hiervoor was onder andere dat in een eerste EML versie ook nog subactivities bestonden die meegenomen moesten worden zodra een er naar een activiteit werd gerefereerd. Dit betekent dat een activiteit dan opnieuw geparsed moest worden. Geneste parses worden niet door Omnimark ondersteund. Daarom dit model. Inmiddels zijn subactiviteiten vervallen.

Per node worden de volgende attributen gezet:

Attribuut	Inhoud
Completed	Bevat optioneel een conditie die aangeeft wanneer de node is afgerond. Bovendien wordt meegegeven in welke dossier variabele deze status wordt bewaard.
Condition	Bevat de zichtbaarheidsconditie voor deze node. De zichtbaarheidsconditie wordt gesplitst in een show en hide conditie. Deze is weer samengesteld uit de drie typen conditie zoals besproken in paragraaf 3.2. De condities worden als volgt opgebouwd: Hide: ((type1 OR type1) AND NOT (type 2)) OR NOT(type 3) Show: ((type1 OR type1) AND (type2 AND type2)) OR type3
Content	Naam van de XML file waarin de inhoud van de node staat
Enable	Een enabled conditie voor elementen van een sequence
Environment	De naam van een XML bestand volgens dezelfde DTD die de environment boom representeert die bij de betreffende activiteit hoort. Indien er geen environment is gedefinieerd bij een activiteit wordt een default environment gebruikt (indien dit is gedefinieerd).
Id	Unieke id van iedere node
Minimum	Niet in gebruik
Name	Naam van de node zoals deze verschijnt in het interface
Record	Niet in gebruik
Type	Type van de node (Activity, Activity-structure, Activity-selection, Activity-sequence, Environment, Knowledge-object, enz.). Deze typering is noodzakelijk voor de functionele ordening (i.t.t. hierarchische) van de nodes.
Url	????? Niet in gebruik

30.4 M_PLAY_TREE

Verwerkt nieuwe geneste Activity-structure ten behoeve van de activiteitenboom. Met name elementtellers die de id per node bepalen worden hier gereset en verhoogd. De inhoud wordt vervangen door een token. Zie paragraaf 3.3 voor verdere details.

30.5 M_ACTIVITY_NODE

Genereert de activiteitnode in de activiteitenboom. Dit commando verwerkt alle activiteiten in de activiteitenboom. De activiteiten worden apart verwerkt omdat deze in eerdere versies van EML ook subactiviteiten konden bevatten die onderdeel uitmaakten van de content van deze activiteit. Met andere woorden deze subactiviteiten waren weliswaar geen toplevel elementen maar kwamen wel afzonderlijk in de activiteitenboom voor. Voor de verdere afhandeling zie paragraaf 3.3

30.6 M_ENVIRONMENT_TREE

Dit commando zorgt ervoor dat een environment wordt vertaald naar een XML document volgens de ActivityTree.dtd. Een environment bestaat uit andere environments of uit bladeren. Deze bladeren zijn:

- Knowledge-object
- Announcement-object
- Communication-object
- Questionnaire-object
- Role-information-object
- Tool-object
- Index-search-object
- Personal-object.

Voor ieder blad wordt een node aangemaakt. Voor ieder genest environment wordt een entity-reference aangemaakt. De complete bomen worden in een volgende stap in hun geheel opgelost (zie paragraaf 3.7)

30.7 M_ENVIRONMENT_INSTANCE

Dit commando traceert elk startpunt voor een nieuw environment (Activity, Activity-sequence, Activity-selection en Content). Vanuit dit startpunt wordt vervolgens via de functie g_new_generate_xml_document de volledige XML environment boom opgebouwd. De bouwstenen voor deze boom vormen de kant en klare brokjes uit paragraaf 3.6. De functie g_new_generate_xml_document opent een fragmentje uit paragraaf 3.6 en spoort iedere entity-reference op en vervangt deze door een bijbehorende inhoud. Een en ander gebeurt recursief. Tot slot wordt de buitenste environment node verwijderd zodat deze niet in het interface verschijnt.

30.8 M_GENERATE_CONTENT

Dit commando genereert alle inhoudspagina's per toplevel elementtype. Als concessie t.b.v. performancewinst is besloten om in deze versie van Edubox geen rekening te houden meer meerdere EML versies en alle inhoud ongeacht versie in een slag te genereren. Indien het systeem wel met meerdere ebpub modules gaat werken moet dit commando zodanig aangepast worden dat alleen elementen van deze release en deze EML versie worden verwerkt.

Er is een shelf met alle toplevel-element-types die per pagina oproepbaar zijn. Dit is een subset van alle toplevel elementen!! (denk aan secties). Voor ieder element van deze shelf worden alle MDA elementen opgevraagd bij de betreffende release. Voor iedere EML blob wordt vervolgens een XML pagina volgens het tussenformaat aangemaakt.

Voordat er met de parsing kan worden begonnen moet via een findrule worden vastgesteld of de EML blob geen referenties bevat naar toplevel elementen die op zichzelf geen pagina's vormen. Deze elementtypes (bijvoorbeeld secties) zijn opgeslagen op de shelf S_ELEMENTS_TO_EXPAND. Met andere woorden de EML blob wordt voor sommige toplevel elementen geëxpandeerd alvorens met de parsing wordt begonnen.

De parsing van de diverse inhoudspagina's ligt voor de hand en zal niet voor elk EML element worden besproken. Daarentegen worden de algemene principes besproken die per element gelden.

Er is een viertal te onderscheiden klassen van EML elementen.

1. Elementen die de start van een nieuwe pagina opleveren. Dit zijn altijd top-level elementen.
2. Elementen die de start van een nieuwe pagina opleveren als onderdeel van een andere pagina. Weliswaar zijn ook dit toplevel elementen, maar op de omvattende pagina moet bijvoorbeeld een link naar deze subpagina's worden geschreven. Voorbeelden zijn Metadata, Prerequisites.
3. Toplevel elementen die geen nieuwe paginastart opleveren zoals bijvoorbeeld Section.
4. 'Gewone' elementen die geen toplevel elementen zijn.

Naast de 'semantische' parsing per element is er ook een standaard parsing per element klasse. Deze vindt plaats aan het begin en aan het einde van de element rule. Hiertoe zijn acht macro's aanwezig, te weten

1. M_PAGE_ELEMENT_START en M_PAGE_ELEMENT_END
2. M_SUB_PAGE_START en M_SUB_PAGE_END
3. M_TOPLEVEL_ELEMENT_START en M_TOPLEVEL_ELEMENT_END

4. M_NONE_PAGE_ELEMENT_START en M_NONE_PAGE_ELEMENT_END

De macro's vervullen de volgende functies:

1. **M_PAGE_ELEMENT_START**
   - Creëer de standaard XML header
   - Creëer een anchor voor het element. Een anchor is nodig voor eventuele verwijzingen naar dit element
   - Maak verwijzing naar metadata
   - Reset de sectie teller
   - Reset de shelves met titels, subtitels en afterflow. De afterflow shelf is nodig voor die elementen die niet sequentieel dienen te worden gerepresenteerd. (Zie Special elementen in EML).

   M_PAGE_ELEMENT_END
   - Vul de naam in die gebruikt moet worden als naar het element gerefereerd wordt. Afhankelijk van de aanwezige attributen en elementen wordt hiervoor gebruikt: Link-name, Title, Id, Wordwide-unique-id of de elementnaam.
   - Voeg dit element toe aan de index-shelves. Er zijn vier index shelves te weten: g_s_type_index (tbv indexering op attribute type), g_s_object_index (tbv indexering op object type), g_s_content_type_index (tbv indexering op attribuut content-type) en tot slot g_s_element_id_index (tbv indexering op element id). Per shelf wordt als key de zoekwaarde bijgehouden waarbij als inhoud weer een shelf met bestandsnamen en anchor id's wordt bijgehouden. Op deze manier is een verwijzing altijd te vertalen naar een bepaalde positie in een document.
   - Alle gebruikte shelf entries voor dit element worden nu verwijderd (title, metadata enz)
   - Schrijf XML footer weg.

2. **M_SUB_PAGE_START**
   Idem als M_PAGE_ELEMENT_START m.u.v. het feit dat sommige shelves (g_c_section_level en g_s_afterflow) tijdelijk worden bewaard (op een stack gepushed) op lokale shelves.

   M_SUB_PAGE_START_END
   Idem als M_PAGE_ELEMENT_END m.u.v. het terugzetten van de shelves g_c_section_level en g_s_afterflow (pop van de stack) en het niet indexeren op element id.

3. **M_TOPLEVEL_ELEMENT_START**
   Idem als M_PAGE_ELEMENT_START m.u.v. het niet aanmaken van de XML header.

   M_TOPLEVEL_ELEMENT_END
   Idem als M_PAGE_ELEMENT_END m.u.v. het niet wegschrijven van een XML footer.

4. **M_NONE_PAGE_ELEMENT_START**
   Plaats een anchor voor dit element.

   M_NONE_PAGE_ELEMENT_END
   - Bepaal de refname (zie M_PAGE_ELEMENT_START)
   - Vul de index shelves g_s_ type_index, g_s_content_type_index, g_s_object_index.

De opgebouwde indexen worden na de parsingslag verpakt in een XML pakketje dat als response teruggeven wordt aan ebpublication. Hier wordt op deze manier een totale index opgebouwd. Zie ebpublication.

De metadata van ieder toplevel element wordt weggeschreven naar een apart bestand waarbij via de hierboven beschreven macro's een link wordt gegeneerd op de pagina om de metadata eenvoudig op te kunnen roepen. Bovendien wordt de titel en de subtitel op hiervoor bestemde shelves bewaard. Deze titel wordt onder andere gebruikt voor referenties.

Evenals metadata worden voor learning-objectives en prerequisites aparte bestanden gegenereerd. Verder worden deze hetzelde afgehandeld als metadata.

De vormgeving van ieder element wordt bepaald door een stylesheet in XML-achtig formaat. Dit stylesheet vormt een onderdeel van het publicatierecept. De stylesheet is opgebouwd op basis van de volgende structuur:

```
<Style Id='id van het element, vaak de naam'>
  <tags>
</Style>
```

De tags bestaan enerzijds uit XHTML volgens het gehanteerde tussenformaat (zie Edubox tussenformaat) of speciale tagging. Deze speciale tagging zijn invulvelden. Op deze plekken dient de inhoud van een element gesubstitueerd te worden. Hiertoe zijn er twee opties, te weten:

- **^id^**
  hier komt de inhoud weergegeven door het id
- **^#language-id#^**
  hier komt taalafhankelijk woord met de id in de gekozen taal van het presentatierecept

Voor de taalafhankelijke elementen is er een speciaal XML-achtig formaat gedefinieerd waar per taal de specifieke vertaling voor een language-id wordeng gegeven. Het formaat ziet er als volgt uit:

```
<LSE Name='language-id'>Vertaling</LSE>
```

De functie g_s_generate_content_for zorgt voor de interpretatie van de stylesheet en de vertaaltabel. Hiervoor wordt het element id meegegeven plus een shelf met de inhoud van het element. De key van deze content shelf komt overeen met de ^id^ in de stylesheet. Welke taal en welke stylesheet (dus het publicatierecept) wordt gebruikt, wordt als parameter meegegeven. De inhoud van het stylesheet en de vertaaltabel staat op 2 shelves die tijdens de initialisatie van het proces zijn ingelezen (zie 3.1 M_INITIALISE_PUB).

30.9 M_BUILD_REFERENCE_SHELVES

Deze macro wordt aan het eind van elke ebpub-cyclus uitgevoerd. De macro neemt de inhoud van de 4 reeds besproken indexerings-shelves en zet de inhoud ervan achter elkaar (gescheiden door een gedefinieerd teken) in één stream zodat de indexeringen in een latere fase gemakkelijk terug naar ebpublication gestuurd kunnen worden.

30.10 Always

Dit stukje code wordt altijd uitgevoerd. De stream met het resultaat van de indexeringen wordt gesloten indien deze nog open staat en wordt vervolgens aan de functie ‘g_s_generate_xml_response’ meegegeven, die de resultaten samen met deze inhoud terugstuurt naar ebpublication, het aanroepende proces.
‘g_s_generate_xml_response’ onderkent 3 verschillende situaties en handelt hiernaar:

- Er zijn geen fouten opgetreden (shelf ‘g_s_error_log’ is leeg) en er is tekst/inhoud als parameter met de functie meegestuurd. De <OK>-tag wordt teruggestuurd samen met de tekst/inhoud.
- Er zijn geen fouten opgetreden (shelf ‘g_s_error_log’ is leeg) en er is geen tekst/inhoud als parameter met de functie meegestuurd. De <OK>-tag wordt teruggestuurd zonder tekst/inhoud.
- Er zijn fouten opgetreden. De foutboodschappen worden teruggestuurd, omvat door de <ERROR>-tag.

Door op deze manier het resultaat terug te sturen weet de unit-of-study-manager (de applicatie, ook wel ‘Runplanner’ genoemd) welk soort melding op het scherm te zetten.

Starten Omnimark Programma's

Inleiding

De Omnimark programma's (import, release, dos, publication en pubgen) worden als service gestart via een speciaal hiervoor ontwikkeld programma: ProcessManager. Deze aanpak heeft diverse voordelen:

- de Omnimark programma's hoeven niet met de hand te worden opgestart
- de Omnimark programma's zijn niet meer hinderlijk zichtbaar in het userinterface
- er hoeft geen gebruiker meer in te loggen. De Omnimark programma's lopen onder de rechten van de ProcessManager, die op zijn beurt weer onder de rechten van een gebruiker draait.
- de Omnimark processen kunnen continu en op afstand gemonitored worden via een aparte userinterface client.
- de Omnimark processen worden opnieuw gestart na een crash
- er wordt een logging bijgehouden van iedere start of stop van een Omnimark proces
- via het userinterface is het mogelijk om een of meerdere Omnimark processen te stoppen
- via het userinterface is het ook mogelijk om de service (ProcessManager) zelf te stoppen en te starten
- de service kan op via standaard NT commando's (rc, net stop enz.) gestart en gestopt worden.

Het enige nadeel van deze aanpak is een additionele ini-file die geconfigureerd moet worden.

Installatie

De installatie van de service zal via de uniform setup verlopen en behoeft in principe geen handmatige ingrepen. De volgende files worden geïnstalleerd in de <environment>\<customer>\Admin\Componenten\Shared directory:

ProcessManager.exe (de service zelf)
ProcessManager.ini (de ini files met paden, urls etc.)
StopOmnimark.exe (een applicatie die een stop verzoek aan de Omnimark processen geeft).

De service kan initieel geregistreerd worden door het volgende commando:
ProcessManager /install
De service moet onder een user account draaien omdat de Omnimark processen in staat moeten zijn om bestanden op de andere server weg te schrijven via een share. Ons voorstel is het om hier de OMISAPIR gebruiker te gebruiken. Een en ander kan ingesteld worden via de standaard NT Services applicatie (via controle panel) onder de optie Startup.

Op een of meerdere machine(s) (het mag dezelfde zijn waarop ProcessManager draait) in hetzelfde domein mag/moet ook nog een derde applicatie worden geïnstalleerd, te weten: ProcessUi.exe. Deze applicatie is het userinterface voor ProcessManager.

ProcessManager ini file

Hieronder staat een voorbeeld van een ProcessManager ini file. Deze wordt automatisch aangepast door de setup procedure van het uniforme Edubox deel:

```
[General]
AppCount=5                                  << Aantal applicaties die gestart worden
ShareCount=0                                << Nvt
ServiceName=OMProcessOU                     <<Naam van de service in service manager
etc

[Application1]                              <<Setup per applicatie
Application=c:\omnimark\omnivm.exe          <<Pad naar de te starten applicatie
Verb=open                                   <<Commando (is starten applicatie)
Parameters=-f goc.f                         <<Paramaters voor applicatie
Directory=d:….\.componenten\Publication      <<Directory waar applicatie staat
Timeout=10000                               <<Refresh userinterface time-out
Style=2                                     <<Modus waarin applicatie start
Title=Publication                           <<Titel van de applicatie, verschijnt in ui
[Kill1]                                     <<Stop applicatie bij vorige applicatie
Application=StopOmnimark.exe                <<Naam van stop applicatie
Verb=open                                   <<Start programma
Parameters=http://int.admin.ou.nl/scripts/omisapir.dll/ebpublication_ou?stop_service=1
omclient eloari                             <<Parameters voor stop applicatie
Directory=d:\int\ou\admin\componenten\shared<<Directory van de stop applicatie
Timeout=600000                              <<Timeout voordat zowel de stop als
omnimark                                           <<hard worden gekilled
Style=2                                     <<Opstart modus

[Application2]
Application=c:\omnimark\omnivm.exe
Verb=open
Parameters=-f goc.f
Directory=d:\int\ou\admin\componenten\PubGen
Timeout=10000
Style=2
Title=PubGen

[Application3]
Application=c:\omnimark\omnivm.exe
Verb=open
Parameters=-f goc.f
Directory=d:\int\ou\admin\componenten\Release
Timeout=10000
Style=2
Title=Release
[Kill3]
Application=StopOmnimark.exe
Verb=open
Parameters=http://int.admin.ou.nl/scripts/omisapir.dll/ebrelease_ou?stop_service=1
omclient eloari
Directory=d:\int\ou\admin\componenten\shared
Timeout=600000
Style=2

[Application4]
Application=c:\omnimark\omnivm.exe
Verb=open
Parameters=-f goc.f
Directory=d:\int\ou\admin\componenten\Dos
Timeout=10000
Style=2
Title=Dos

[Application5]
Application=c:\omnimark\omnivm.exe
Verb=open
Parameters=-f goc.f
Directory=d:\int\ou\admin\componenten\Import
Timeout=10000
Style=2
Title=Import
[Kill5]
Application=StopOmnimark.exe
Verb=open
Parameters=http://int.admin.ou.nl/scripts/omisapir.dll/ebimport_ou?stop_service=1
omclient eloari
Directory=d:\int\ou\admin\componenten\shared
Timeout=600000
```

Edubox

Style=2

Log file

Hieronder staat een voorbeeld van een deel van een logfile die door de ProcessManager wordt gegenereerd De file heet processmanager.log en wordt in dezelde directory geplaats als processmanager.exe

```
---------------------------15:19:06 23/11/2000--------------------------
Service Starting.
OMProcessOU Thu 23/11/2000 15:19:06 Command Slot: \\E-CHI-BDC-
3\OMProcessOU\PrjManagerCommand
OMProcessOU Thu 23/11/2000 15:19:06 Service Name: OMProcessOU
OMProcessOU Thu 23/11/2000 15:19:06 Connecting to Network.
OMProcessOU Thu 23/11/2000 15:19:06 Launching Applications.
OMProcessOU Thu 23/11/2000 15:19:06 Start of all Processes pending.
Service Started.
OMProcessOU Thu 23/11/2000 15:19:06 Process 1 Started.
OMProcessOU Thu 23/11/2000 15:19:07 Process 2 Started.
OMProcessOU Thu 23/11/2000 15:19:07 Process 3 Started.
OMProcessOU Thu 23/11/2000 15:19:07 Process 4 Started.
OMProcessOU Thu 23/11/2000 15:19:07 Process 5 Started.
Service Stopping.
OMProcessOU Thu 30/11/2000 11:17:22 Killer Process 1 started.
OMProcessOU Thu 30/11/2000 11:17:28 Killer Process 1 terminated normally.
OMProcessOU Thu 30/11/2000 11:17:28 Process 1 Stopped with exitcode:0x00000000.
OMProcessOU Thu 30/11/2000 11:17:28 Killer Process 3 started.
OMProcessOU Thu 30/11/2000 11:17:28 Process 2 Stopped with exitcode:0x00000000.
OMProcessOU Thu 30/11/2000 11:17:32 Killer Process 3 terminated normally.
OMProcessOU Thu 30/11/2000 11:17:32 Process 3 Stopped with exitcode:0x00000000.
OMProcessOU Thu 30/11/2000 11:17:32 Process 4 Stopped with exitcode:0x00000000.
OMProcessOU Thu 30/11/2000 11:17:32 Killer Process 5 started.
OMProcessOU Thu 30/11/2000 11:17:32 Killer Process 5 terminated normally.
OMProcessOU Thu 30/11/2000 11:17:33 Process 5 Stopped with exitcode:0x00000000.
---------------------------11:17:38 30/11/2000--------------------------
Service Starting.
OMProcessOU Thu 30/11/2000 11:17:38 Command Slot: \\E-CHI-BDC-
3\OMProcessOU\PrjManagerCommand
OMProcessOU Thu 30/11/2000 11:17:38 Service Name: OMProcessOU
OMProcessOU Thu 30/11/2000 11:17:38 Connecting to Network.
OMProcessOU Thu 30/11/2000 11:17:38 Launching Applications.
OMProcessOU Thu 30/11/2000 11:17:38 Start of all Processes pending.
Service Started.
OMProcessOU Thu 30/11/2000 11:17:38 Process 2 Started.
OMProcessOU Thu 30/11/2000 11:17:38 Process 3 Started.
OMProcessOU Thu 30/11/2000 11:17:38 Process 4 Started.
OMProcessOU Thu 30/11/2000 11:17:38 Process 5 Started.
OMProcessOU Thu 30/11/2000 11:17:38 Process 1 Started.
```

Stoppen Omnimark processen

De Omnimark processen worden via de service gestopt. Hiertoe roept de ProcessManager het Programma StopOmnimark aan. Als parameter wordt de volgende URL meegegeven: http://<DNS admin site>/scripts/omisapir.dll/import_<environment>_<customer>?stop_service=1 omisapir <password>

De volgende Omnimark programma's kunnen op deze wijze gestopt worden:

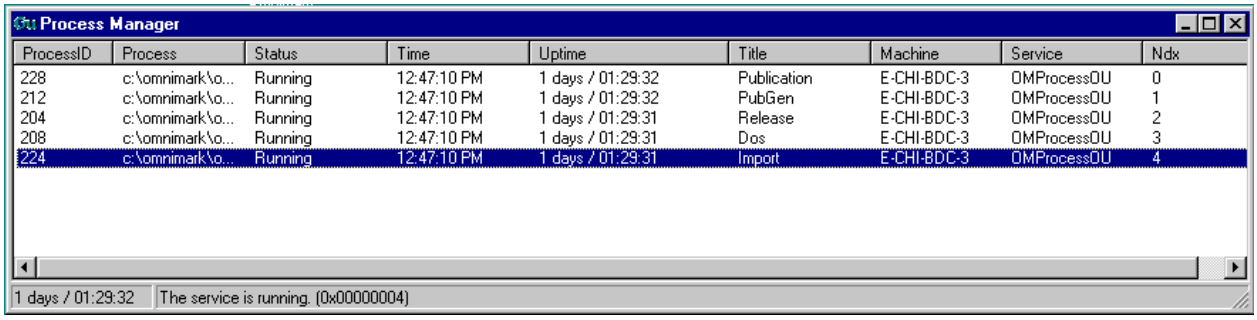
- Import
- Release
- Publication

De programma's Pubgen en Dos worden dan ook vanzelf gestopt. Mochten dit stoppen niet werken, dan zal de ProcessManager na het verstrijken van een time-out alle programma's op de harde manier stoppen (killen). Hetzelfde gebeurt als de service (ProcessManager) wordt gestopt.

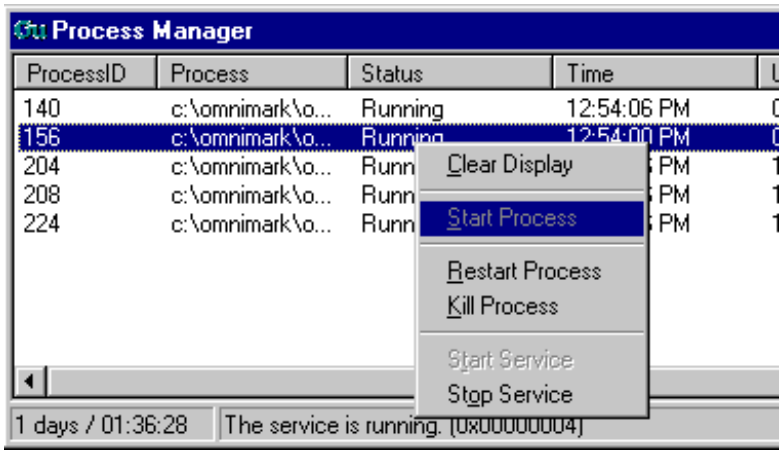
Gebruikersinterface

ProcessUI is het userinterface voor de ProcessManager. ProcessUI mag op iedere PC in het domein worden gestart. Binnen enkele seconden wordt de status van alle Omnimark processen getoond voor iedere klant binnen het domein.

De interface ziet er als volgt uit:



Per proces wordt nu onder andere bijgehouden wat de status, tijd van laatste refresh, uptime is. Via de rechter muisknop wordt het menu beschikbaar. Dit ziet er als volgt uit:



Via het menu is het mogelijk om:

- het scherm te wissen. Na een poosje bouwt het scherm zich weer op. Dit heeft met name zin als er een of meerdere ProcessManagers zijn gestopt en deze niet meer in het user interface zichtbaar moeten zijn.
- een proces te herstarten. Het programma wordt nu op een gecontroleerde manier gestopt en vervolgens automatische weer gestart.
- een proces te killen. Het programma wordt nu op een ongecontroleerde manier gestopt!!!!!. Deze functie is met name zinvol als een programma niet meer reageert op een reguliere stop opdracht.
- de service te stoppen. De ProcessManager wordt gestopt nadat alle processen zijn gestopt.
- de service weer te starten. Dit is natuurlijk alleen maar mogelijk als er nog informatie over de service in het user-interface te zien is. Als men een service stopt en vervolgens

het scherm wist, kan die service niet meer via het interface gestart worden. Het is natuurlijk wel nog mogelijk om de service op de traditionele manier te starten (net start …. enz.) waarna de service weer in het userinterface verschijnt.