Praktijkvoorbeeld evolueerbare software-architecturen: Elgg

Salvi Jansen

8 januari 2011

Elgg: Open source social networking engine

Introductie tot Elgg

Elgg is een open source framework voor het ontwikkelen van sociale netwerken in een webomgeving. De software is geschreven voor php en MySql-databases. Deze bespreking beperkt zich tot het bijzondere datamodel en de onderliggende databasestructuur. De software kenmerkt zich namelijk door een aantal basisstructuren die eveneens kenmerkend zijn voor sociale netwerken, en bij uitbreiding zou alle software op gelijke manier kunnen worden uitgedrukt.

Basisstructuren binnen Elgg: het datamodel ontleed

ElggEntity

De meest fundamentele basisstructuur is de *entiteit*, binnen het framework aangeduid als *ElggEntity*. Dergelijke entiteit omvat de klassieke data-elementen: een gebruiker, een groep, een blog en bij uitbreiding ook een site, waarmee men meteen een mogelijkheid tot multisite-implementatie creëert. Het framework voorziet vier basisentiteiten met elk een aantal unieke karakteristieken: ElggSite, ElggUser, ElggObject, ElggGroup. Als men bijvoorbeeld een blogsysteem wil implementeren, kan men met enkele simpele lijnen code op een dynamische manier blogs aanmaken. Gezien het databasemodel is het niet nodig een tabel “Blogs” te voorzien. Men kan dynamisch het subtype “blog” aanmaken, dat overerft van ElggObject:

```
$object = new ElggObject();

$object->subtype = "blog";

$object->access_id = 2;

$object->save();
```

Het valt op te merken hoe men initieel enkel de meest fundamentele informatie over een object meegeeft: een unieke naam om zich te onderscheiden van andere objecten, en een access-identifier die in dit voorbeeld op “public” (2) staat. Men heeft op dynamische manier de klasse

“Blog” aangemaakt die vanaf nu geïnstantieerd kan worden. Zelf is “Blog” een instantiatie van het type ElggObject. Meta-informatie kan men pas achteraf toevoegen.

ElggRelationship

Het is binnen alle software wenselijk dat men relaties bouwt tussen bovenstaande entiteiten. Een “lidmaatschap” zou een gebruiker aan een groep koppelen, of twee gebruikers kunnen gelinkt worden door een “vriendschap”. Hiervoor dient men binnen het datamodel een *ElggRelationship* te creëren. Dit data-element doet niets anders dan het leggen van een simpele link tussen twee instanties van dezelfde of twee verschillende entiteiten.

ElggMetadata

Wat databasetabellen snel onhandelbaar maakt omdat het nogal een wijzigbaar gegeven is, zijn metadata. Hoewel metadata vaak “vastzit” in de tabel/klasse waarvan het metadata is, is voor het datamodel van Elgg alle metadata gelijk. Bij de bespreking van de databasestructuur wordt hier verder op ingegaan. Metadata wordt simpelweg voorzien via een oproep in de code:

```
$entity->metadata_name = $metadata_value;
```

ElggAnnotation

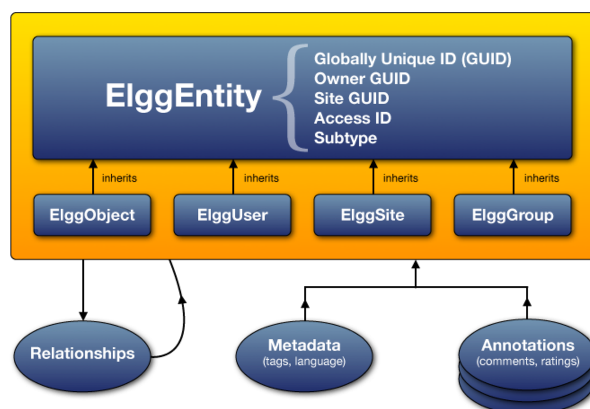
Een *annotatie* komt tegemoet aan informatie die men wil “vasthangen” aan een object, maar die niet relationeel is, noch metadata. Het beste voorbeeld hiervan is bijvoorbeeld een commentaar op een blogbericht. Een commentaar zou er programmatisch dus zo kunnen uitzien:

```
$entity->annotate('comment', $comment_text, $entity->access_id);
```

Het Elgg-datamodel compleet

Een grafische voorstelling van het datamodel ziet er zo uit:

Elgg Data Model



Een innovatief databasemodel

Elgg stelt dat een relationele database perfect in staat moet zijn metadata te koppelen aan objecten, zonder dat die metadata daarvoor een kolom in de object-tabel moeten zijn. Het is niet optimaal een titel, beschrijving, datum... op te slaan als kolommen in de tabel “blog”, want dat maakt dat men ze niet dynamisch kan aanmaken. Elgg zal een unieke referentie naar het metadata-element opslaan in de database die het koppelt met een bepaalde entiteit.

Het is op eenzelfde manier niet nodig aparte tabellen te maken voor elk subtype. Elk subtype dat wordt aangemaakt in de code krijgt een eigen vermelding in de tabel “elgg_entity_subtypes”. Met die unieke referentie wordt in de andere tabellen verdergewerkt.

Uitgewerkt voorbeeld

We testen de structuur van Elgg door dynamisch een blogsysteem aan te maken, een blogbericht te posten via dit systeem én een reactie hierop. Dit demonstreert de verwerking van het ElggObject-element, ElggMetadata-element en het ElggAnnotation-element. Dit heeft volgende effecten op de database:

Subtype “blog” aanmaken. Door aan te geven dat we een blog-functionaliteit willen inbouwen (zie eerder), zal Elgg “blog” registreren als een subtype waarvan instantiaties kunnen worden gedaan. In de tabel “elgg_entitity_subtypes” krijgt dit blog-subtype de unieke identifier “4”. Een “blog” erft over van het basistype “object”.

+ Opties

id	type	subtype	class
1	object	file	ElggFile
2	object	plugin	ElggPlugin
3	object	widget	ElggWidget
4	object	blog	
5	object	custom_profile_field	ProfileManagerCustomProfileField
6	object	custom_group_field	ProfileManagerCustomGroupField
7	object	custom_profile_type	ProfileManagerCustomProfileType
8	object	custom_profile_field_category	ProfileManagerCustomFieldCategory
9	object	about	
10	object	privacy	

Selecteer alles / Deselecteer alles *Met geselecteerd:*

Een aantal attributen (metadata) toevoegen aan een blog. Elke “blog” kent attributen zoals een titel, het bericht zelf, aanmaakdatum, bewerkdatum... Deze informatie vinden we terug in de code en in de aparte instantiaties van de blog-entiteit. We vinden evenwel niet terug in de database dat “standaard genomen” een blogbericht attributen X, Y en Z bevat. Dat is niet nodig gezien het de software niet evolvable maakt. Het moet gemakkelijk zijn om attributen toe te voegen of weg te nemen. We kunnen dus enkel de concrete waardes voor attributen per blogbericht terugvinden.

Een instantie van "blog" aanmaken / Een blogbericht posten. In de tabel "elgg_entities" vinden we alle instanties van de subtypes terug. Elk concreet blogbericht, concrete user, concrete groep... zit hier, evenwel zonder attributen (!) opgeslagen. Elk blogbericht is een entiteit van subtype "4", dus we vinden een blogbericht met guid "9". Dit is dus de unieke identifier van dat concrete blogbericht.

+ Opties

guid	type	subtype	owner_guid	site_guid	container_guid	access_id	time_created	time_updated	last_action	enabled
1	site	0	0	0	0	2	1288646796	1288646796	1288646796	yes
2	user	0	0	1	0	2	1288646817	1294446661	1288646817	yes
8	object	3	2	1	2	1	1291685756	1291685756	1291685756	yes
4	group	0	2	1	2	2	1288723886	1288724446	1288723886	yes
5	object	2	2	1	2	2	1290547302	1290551759	1290547302	yes
7	object	2	2	1	2	2	1290558764	1290558764	1290558764	yes
9	object	4	2	1	2	1	1291686013	1293497286	1291686013	yes
10	user	0	0	1	0	2	1291829503	1291829534	1291829503	yes
11	object	5	1	1	1	2	1291831301	1291831301	1291831301	yes

De meta-informatie van blogbericht "9". In een andere tabel moeten we vervolgens op zoek naar alle meta-informatie over blogbericht met guid "9". We hebben zowel de naam van de "key" als de "value" nodig gezien de keys niet in een kolomhoofd staan als bij een gewone tabelstructuur. De tabel "elgg_metadata" geeft als resultaat:

id	entity_guid	name_id	value_id	value_type	owner_guid	access_id	time_created	enabled
379	9	73	74	text	2	1	1293497286	yes
378	9	73	66	text	2	1	1293497286	yes
281	9	75	76	text	2	1	1291686013	yes

In de tabel "elgg_metastrings" vinden we vervolgens dat *name_id* 73 staat voor "tags", wat verklaart waarom er voor dezelfde *name_id* twee *value_id's* zijn. *Name_id* 75 herbergt de sleutel "comments_on", dewelke voor zichzelf spreekt en de waarde "on" bevat. Binnen deze databasestructuur staan zowel attribuutnamen als hun waardes in dezelfde tabel opgeslagen in de vorm van rijen!

Men kan zich afvragen waar de titel en het eigenlijke blogbericht heen zijn. De *title* en *body* van het blogbericht werden opgeslagen in de attributen die "blog" overerft van ElggObject, namelijk *title* en *description*. Alle subtypes van ElggObject hebben deze twee attributen standaard, dus Elgg heeft er een aparte tabel "elgg_objects_entity" voor voorzien. Dit is een puur praktische optimalisatiemaatregel gezien de *description* een behoorlijk groot tekstveld kan worden. Dit wordt hier bewezen omdat het volledige blogbericht erin opgeslagen wordt. Men kiest er dan ook voor deze grote stukken in een aparte tabel weg te schrijven.

Reageren op blogbericht "9". Op elke entititeit kan men reageren. Alle reacties (of andere soorten van annotaties) bevinden zich dan logischerwijs in één tabel, "elgg_annotations". We reageerden op blogbericht "9":

+ Opties

id	entity_guid	name_id	value_id	value_type	owner_guid	access_id	time_created	enabled
1	9	80	79	text	2	1	1291686119	yes

Selecteer alles / Deselecteer alles *Met geselecteerd:*

De referenties *name_id* en *value_id* verwijzen naar waardes opgeslagen in de tabel "elgg_metastrings".

Uitbreidingen op Elgg

Het origineel blijft behouden

Elgg is op meerdere vlakken een innovatief stukje software. Via zijn interne API die de core-functies aanspreekbaar maakt en de automatische detectie van *plug-ins* ("mods" genaamd) kan men het systeem moeiteloos uitbreiden. Het meest fundamentele probleem bij de ontwikkeling van software zoals gezien in de cursus zijn de propagatie-effecten van aanpassingen in de software, die de grootte van het systeem gaan aannemen. Elgg is gebaseerd op een "core" met *events, hooks* en een doorgedreven *view*-systeem.

Via een plug-in kan men niet enkel functionaliteit toevoegen, men kan ook bestaande functionaliteit herschrijven. Zo zal elke *view* uit een plug-in voorrang krijgen op de originele view van de core. Zo kan men de footer van de site, die men in de core vindt onder *footer.php* makkelijkerwijze overschrijven door een eigen versie door een eigen *footer.php* te schrijven in de plug-in-folder. Hetzelfde geldt voor extra actielogica. Het systeem detecteert automatisch de bestanden in de plug-ins en geeft deze voorrang op wat standaard voorzien is. Zo zal men nooit de core-bestanden moeten aanpassen en kan men deze systeemwijde rimpeleffecten grotendeels vermijden. De modulaire opbouw met plug-ins die een voorgeschreven structuur volgen en die men met één klik kan uitschakelen, samen met de interne API zorgt voor een ongestoorde expansie van de eigenlijke core-elementen, met de mogelijkheid tot extra overlayschermen en custom implementatieacties.

Een doorgedreven logsysteem, cachesysteem...

Gegeven het feit dat Elgg draait op een aantal basisconstructen waar de gebruiker op kan verder bouwen, en een view- en plug-in-systeem dat modulair op de core kan "geplugd" worden, is het mogelijk om een stabiel logsysteem in te bouwen of een "pluggable" cachesysteem voor de server te voorzien (simple cache of MEMcache).

Elgg op de keeper beschouwd

Elgg kent *principes van evolvable software* omdat het afstapt van het klassieke databasemodel dat een tabel per klasse voorziet en dat de attributen converteert naar kolommen. Wat we als gebruiker wél dynamisch kunnen aanmaken en verwijderen zijn rijen. Elke klasse die programmatorisch wordt aangemaakt, wordt een tabelrij. Elk attribuut dat wordt toegekend aan een object, wordt ook een tabelrij en refereert direct aan de instantie van de klasse en niet de klasse zelf, met andere woorden is het geen kolomhoofd.

De relationele database wordt optimaal benut door de scheiding van data en referenties. Van tabellen wordt niet langer verwacht dat ze flexibel zijn in de breedte, want daar lenen ze zich niet toe. Men dient evenwel op te merken dat men de dynamiek van het toevoegen en verwijderen van attributen programmatorisch moet kunnen beheersen. Zo zal een extra attribuut vanaf tijdstip T niet aanwezig zijn bij alle blogberichten voor tijdstip T. De applicatielogica moet zich hiertegen wapenen.

Het modulaire pluginsysteem, de folderstructuur en de interne API zorgen ervoor dat men makkelijk kan bijprogrammeren en deze nieuwe (of vernieuwde) acties en views automatisch voorrang krijgen op het origineel en hiervan ook volledig losstaan, wat aan de belangrijke "core" een deterministisch gedrag koppelt waar men naar op zoek is. De werking van de core wordt afgeschermd door de API en aanpassingen door het ontwikkelteam (in de core) zitten nooit gemengd met aanpassingen van de end-user (in de plug-ins). De kans op systeemwijde propagatie-effecten wordt geminimaliseerd.

Meer informatie op www.elgg.org.

Het ElggData-model werd ontleend van http://docs.elgg.org/wiki/File:Elgg_data_model.png