BlogReader 1.0.0

Af Jonas F. Jensen.

Indholdsfortegnelse

Forord

Jeg vil lige starte denne rapport ud med at beklage at jeg desvære har en noget negativ indstilling til sprog med karaktere som Delphi. Det vil sige sprog, med kun en kommerciel leverandør, mærkelig syntaks og terminologi og manglende platformsuafhængighed. Normalt ville jeg heller aldrig drømme om at udvikle i et sådant sprog, da jeg ikke betegner det som eksistens berettiget i år 2007, måske delphi har haft en smule eksistensberettigelse for 10 år siden. Men jeg vil alligevel forsøge at arbejde med det, og så vil jeg bedre læseren om ikke at tage alle mine negative kommentarer personligt. Delphi er sikkert god nok på bunden, der er bare ret langt ned.

Når men jeg har vedlagt en cd, på den vil du finde digital signeret udgave af denne rapport i ODF (Open Document Format), en PDF udgave samt kildekode og binær assembly af projektet. Kildekoden indeholder alle filer, både for projekt og løsning (projekt-group i delphi termer). Så det skulle være muligt at genopbygge (rebuild) hele applikationen, med borlands IDE. Det er også muligt at installere programmet. Til det formål har jeg skabt en lille installer, den installer båden binær assembly og kildekode; så er det hele ligesom installeret.

Integritet

md5sum (GNU coreutils) 5.96

```
a8c6618daae5904fe44aed4c6d074d05  BlogReader 1.0.0.zip
4271a64a5afff1f7835037982c085a41  BlogReader.exe
119b1636202c705f6df41944fc0ba2cb  BlogReader_uml.png
```

(Rapportens integritet kan verificeres med rodcertifikat fra TDC).

Hvad er BlogReader?

BlogReader en RSS reader rettet mod RSS 2.0 feeds fra blogs. I teorien kan programmet også læse andre almindelige RSS feeds. Formålet med denne RSS læser er at gøre det lettere at følge med i forskellige blogs. Men da fejlhåntering i delphi er af en træls størrelse har jeg valgt at koncentrer mig om RSS feeds af version 2.0, ligesom dem der kommer fra almindelige WordPress blogs. Det vil sige at programmet muligvis/højest sandsynligt vil gå ned hvis man forsøger at læse andre typer feeds. Fordi det ikke er alle felterne der eksistere i alle feeds, og fordi de ikke har samme skema.

RSS, XML og sematic web

I dag er det ved at være normalt af benytte RSS feeds, og XML er næsten blevet et bussword. Vi har også bevæget os godt ind i web 2.0, som alle vores AJAX services hedder Det næste skridt hedder vel naturligt nok web 3.0. Tim Berners-lee, grundlægger af the World-Wide-Web og direktør for W3C, forudser at det næste store ting vi er på vej ind i, kommer til at hedde sematic web. Ideen går på at alle informationer skal være defineret i et maskinlæsbart sprog, og at vi så skal bruge maskiner/computer til læse/analysere disse informationer. På den måde vil fremtiden applikation få adgang til uanede mængder af information. RSS og XML er at godt skidt i denne retning, selvom de første rigtige implementeringer af sematic web med RDF (Resource Description Framework) allerede findes, så er RSS feeds og diverse XML services stadig et skidt i den rigtige retning. Dette gør RSS og XML til nogle interessante teknologier at arbejde med. Jeg har tidligere arbejdet med XML i C#/.Net, tilbåde filformater og kommunikations protokoller, og XML er tydeligtvist fuldstændig uafhængig af bagvedliggende teknologi. Derfor synes jeg RSS/XML er en interessant form for datalagring, hvilket også er grunden til at jeg har valgt at arbejde med det i dette projekt.

Klasse struktur i UML

Før jeg begyndte at implementere selve programmet designede jeg klasse strukturen i UML, i Borlands IDE (Integrated Development Environment) findes der under View > Model View hvor man kan se klasse strukturen i en unit som UML (Unified Modelling Language) diagram. Her kan man også redigere klasse strukturen, tilføje/fjerne klasser, strukturer, interfaces, medlemmer, felter, egenskaber, relationer også videre.

Da jeg så havde opbygget en fornuftig klasse struktur, som blev skellet skrevet ind i min unit (.pas fil). Derefter kunne jeg så begynde at implementere logikken. Det var selvfølgelig ikke alting der kunne implementeres på samme måde som jeg havde designet klasse strukturen, men sådan er det jo altid, specielt når man ikke har udviklet i sproget før.

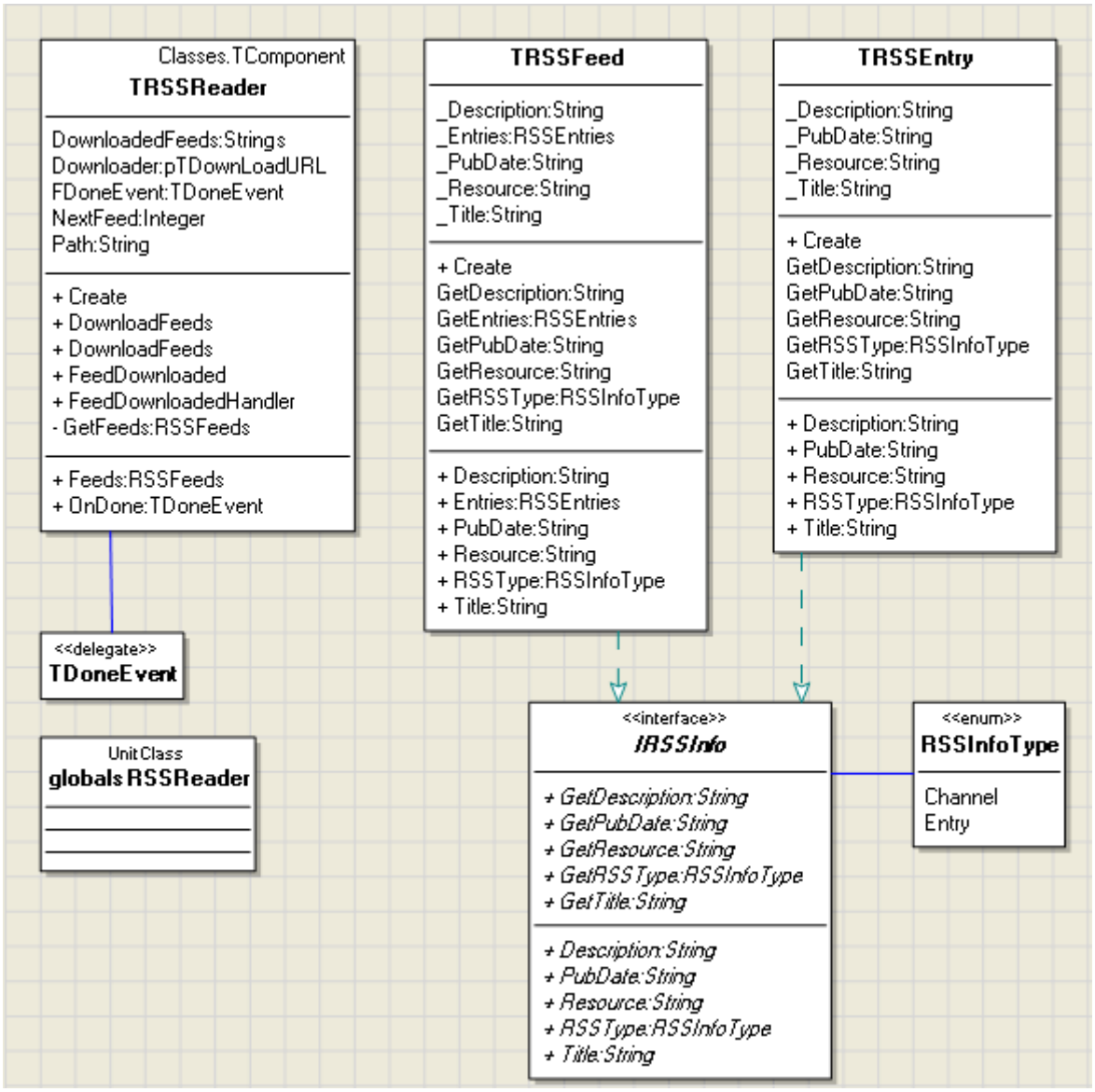
Overordnet opbygning

Jeg opbyggede mit delphi projekt med et library og en frontend. Dette går ud på at implementere et bibliotek/library/package, hvor man placere alt funktionaliteten. Det vil sige at alle funktioner til at parse/downloade xml/rss ligger isoleret fra brugerfladen/frontend. Denne strenge isolering af logik, betyder at man kan implementere funktionaliteten uden at tænke på brugerfladen, da interfacet mellem brugerflade og library er defineret i klasse strukturen. Det betyder også at andre kan genanvende logikken, fordi den ligger isoleret fra brugerfladen. Sidst men ikke mindst er det en klar fordel, hvis man ligesom jeg ofte porter sine programmer til andre platformer og anden GUI (Grafical User Interface). Det er ikke så aktuelt med Delphi, selvom man sikkert forholdsvist enkelt kunne porter biblioteket til .Net/Mono platformen eller lave en binær portering til GNU/Linux med Kylix. En anden fordel ved denne opdeling, som god nok ikke er aktuel i dette projekt, er at man kan opdele implementerings arbejdet mellem flere personer, så længe man har defineret et fælles interface f.eks. i form af en UML specificeret klasse struktur.

Det største problem ved denne form for opdeling af logik er at det kan være svært at teste en implementering før det hele er færdig implementeret, da man ofte bygger op imod ikke færdigt implementerede interfaces. F.eks. kunne jeg ikke teste min RSS parser, før jeg havde implementeret resten af klasse strukturen og brugerfladen. I større projekter ville man skrive nogle ”unit tests”, tvivler på at man benytter samme terminologi i Delphi, men det hedder unit testing alle andre steder. En unit test er en test der går ind og tester om et stykke software virker som det skal. Jeg kunne godt gå mere i dybden med mock objekter og XP (Extreme Programming), men så bevæger vi os uden for emnet.

UML diagram over mit library

UML diagram fra borlands delphi IDE (Intergrated Development Environment)



Dokumentation

I dette afsnit vil jeg gennemgå de vigtigste klasser, strukture og interfaces jeg har benyttet i mit library/bibliotek. Derudover vil jeg undervejs gennemgå nogle af de forskellige teknikker jeg har benyttet.

TRSSReader

TRSSReader havde til formål at download et array af RSS feeds til en midlertidig fil, og derefter oprette og hoste et array af TRSSFeeds, der repræsentere de enkelte feeds. På den måde kunne min brugerflade/frontend hoste et object af denne type, så brugerfladens logik ikke skulle i behandle http anmodninger eller xml parsing.

Hændelser

Et event/hændelse er en ting der gå det muligt for en klasse at kalde en metode der er ukendt ved kompilering (at compile-time), og derfor først er kendt ved afvikling (at runtime). Dette er f.eks. aktuelt ved udvikling af brugerflade uafhængige libraries. Nu ved jeg godt at mit library er i samme assembly (exe-fil), som min frontend. Men i teorien kunne mit library kompileres som en uafhængig dll (dynamic link library), uden at blive linket til min brugerflade ved kompilering (at compile-time). Fordi mit library kun linker til borlands units og alt linking til andre unit kan se dynamisk, ved afvikling (at runtime).

Helt konkret foregår denne dynamiske linking ved at man lader sin klasse udbyde et event, som andre klasser så kan abonnere på, under afvikling (at runtime). Før man kan gøre dette skal man definere et interface til dette event, ved kompilering (at compile-time). I C# kalder man disse interfaces for delegates, i Java er der tale om regulere interfaces, som abonnement klassen skal implementere (Jeg kommer nærmere ind på interfaces, og implementering af disse senere). I delphi sker det ved at man definere en type, af typen *procedure of object*. Dernæst skal man oprette en property (egenskab) af den type som man skabte tidligere. Efterfølgende kan man så løfte det event/hændelse man tidligere skabte ved at kalde den metode den håndtere read/write af din event/hændelse/property. På samme måde kan man abonnere på et event, ved at tildele en metode til event/hændelse/property. Man skal bare huske at den metode man tildeler er af samme type som den *procedure of object* man definerede tidligere, eller får man en kompileringsfejl. Jeg ville ikke gå i dybden med hændelse håndtering i delphi, ligesom jeg heller ikke vil forklare begreber som properties/egenskaber, felter, metoder og access modifiers, hvilket jeg vil overlade til borlands dokumentation af delphi. Helt konkret har jeg benyttet et event i TRSSReader til at fortælle hvornår objektet er færdigt med at downloade og parse alle RSS feeds.

Pointer

Det anden teknik jeg har benyttet i denne klasse, og som jeg lige vil sige lidt om er pointere. Normalt, når vi kopier og/eller parser strukturere/objekter mellem forskellige funktioner; sker der det at vi skaber en identisk kopi af objektet og giver den sit eget scope/levebegrænsning/levetid. Men lige som vi i PHP kunne benytte reference baseret overførsel, kan vi i delphi/C/C++ og andre unmananged sprog benytte pointer. Der findes sikkert også referencer i delphi på samme måde som i C++, men jeg har kun beskæftiget mig med pointere i delphi, så derfor vil jeg ikke komme ind på referencer (Det er også næsten det samme). En pointer er kort fortalt en variabel der indeholder hukommelses adressen på en anden variabel. Så hvis man parser en pointer til en metode, vil det ikke bare give metoden mulighed for at læse indholdet af variablen som pointeren peger på. Nej, metoden vil også kunne modificere den variabel som pointeren peger på i det oprindelige scope.

Jeg har f.eks. brugt pointere til at lade min TRSSReader få adgang til et TDownLoadURL komponent. Uden at TRSSReader hoster et sådan komponent. Dette er relevant fordi man kunne lave noget design-time integration med sit TDownLoadURL, f.eks. kunne man forholdsvis let vise en progressbar over downloadningen, da TDownLoadURL ejes af min brugerflade og ikke af TRSSReader. Det skaber selvfølgelig også det problem at min TDownLoadURL komponent kunne gå ud af scope eller blive ødelagt, hvilket ville betyde at min TRSSReader har en pointer der peger på et sted i hukommelsen hvor der engang lå en TDownLoadURL. Så ville der opstå et stort problem, oftest vil programmet gå ned øjeblikkeligt og i værstefald kan den tage hele computeren med sig. Normalt vil operativ systemet dog gribe ind før hukommelsen bliver inficeret, så det går ikke nødvendigvis helt galt. Et andet problem med pointere er at man kan have en null pointer, dette vil f.eks. hvis en pointer ikke er initialiseret eller hvis pointeren er blevet nulstillet. Hvis man forsøger at tilgå en nullpointer, vil man oftest også opleve nogle sjove ting. Der er tidgengæld også store fordele ved pointere, hvis man skal benytte store data strukturer i forskellige metoder, kan man med fordel benytte pointere for at sænke hukommelses forbruget.

IRSSInfo

IRSSInfo skal repræsentere den mindste del af RSS information man kan præsentere. Alle RSS entries eller items (som de hvis nok hedder i specifikationen), og alle channels har alle de felter der er defineret i IRSSInfo. Derfor ville det være let af benytte IRSSInfo som parameter til en metode skal skal præsenter noget information fra mit library, da både TRSSFeed og TRSSEntry implementere dette interface.

Interfaces/Grænseflader

Et interface eller en grænseflade som man også kan kalde det er en definition af medlemmer som en klasse der implementere dette interface skal udbyde. Normalt skal disse medlemmer også have en *public* access modifier. Man implementere et interface på samme måde som man nedarver fra en klasse. Faktisk kan man godt tale om et interface som en abstrakt klasse der ikke implementere noget logik. Nu vil jeg ikke forklare nedarvning, abstrakte klasser og polymorfisme; da alle begreber indenfor OOP (objekt orienteret programmering) er en mindre bog for sig selv.

I delphi skal interface kun defineres i det afsnit der hedder interface, det samme sted som man definere de medlemmer som de respektive klasser skal implementere. Samme sted skal man også definere de interfaces som de respektive klasser skal implementere. I delphi kan en klasse godt implementere flere interfaces, men den kan kun nedarve fra en klasse. Det er også tilfældet i det fleste andre sprog, med C++ som en undtagelse.

Hele ideen med at benytte interfaces er at man kan opnå samme grænseflade til 2 klasser uden at klasserne nødvendigvis skal nedarve fra samme klasse. Jeg vil ikke komme ind på hvordan man implicit kan konvertere/caste en nedarvet klasse til en stamklasse (parent class). Men den samme ting kan man gøre hvis 2 klasser implementere det samme interface. Man selvfølgelig ikke oprette en udgave (instance of an object) af et interface, da et interface kun er en grænseflade definition og ikke en implementer noget logik.

TRSSFeed

TRSSFeed har til formål at skabe en abstraktion over et RSS feed, TRSSReader download alle feeds til midlertidige filer og holder et array over TRSSFeed. Dette array af TRSSFeed skal så repræsentere de downloadede feeds. Derfor kan TRSSFeed skabes med et filnavn som parameter. TRSSFeed loader den givne fil ind i en dynamisk udgave af TXMLDocument. En dynamisk udgave betyder at den bliver oprette af en statistisk metode, som parser den ud til et interface (IXMLDocument). Dette har fordele med hensyn til hukommelses håndtering (her vist nok tale om delphi specifikke egenskaber, der ikke altid er en fordel), fordi et komponent der bliver opbevaret i et interface, bliver frigivet når interfacet glider uden for scope. Når TRSSFeed har fået loaded sit TXMLDocument, vil den læse de forskellige data for RSS kanalen (adresse, beskrivelse osv...), og derefter parse alle TXMLNode af type items (dvs. xpath: rss/channel/item) videre til TRSSEntry, disse TRSSEntry skal så repræsentere de individuelle blog indlæg, og vil blive opbevaret i et array af TRSSEntry (typen TRSSEntries). Til min personlige overraskelse implementerede TXMLDocument ikke nogen support for xpath, som så man andre lign. frameworks gør, men det lykkedes dog alligevel at parse og gennemløbe de forskellige tags med lidt besvær.

TRSSEntry

TRSSEntry har til formål at skabe en abstraktion over et blog indlæg. TRSSEntry bliver opbevaret i et array på hver udgave (instance of) af TRSSFeed. Ligesom TRSSFeed implementere denne klasse også interfacet IRSSInfo, hvilket skulle gøre det lettere og mere effektivt at behandle data fra TRSSReader. TRSSEntry tager en TXMLNode som parameter i konstruktøren (the constructor), derfra læser den de data den skal indholde, for at kunne implementere IRSSInfo.

Perspektivering

Jeg startede ud med at forklarer om semantic web, hvilket helt klart er en af de ting der gør rss, xml og hele dette projekt til et interessant emne at arbejde med. Derfor vil jeg ikke snakke om det engang til, min i stedet benytte muligheden til at forklare hvordan mit program kunne forbedres. Det skal nemlig siges at jeg ikke hararbejdet ret meget med flere ting der kunne være meget vigtigt. Disse emne inkludere:

- Hukommelses håndtering
- asynkron tilbagekald (asynchronous callback)
- Fejl/undtagelses håndtering (Error/exception handling)
- I18n/L10n (Internationalization/localization)

Asynkron tilbagekald og multithreadning, ligger uden for rammerne for dette projekt. Da jeg ikke har nogle erfaringer med delphi. Derfor bliver alle http anmodninger lavet synkront, hvilket også godt kan mærkes hvis man forsøger at fremprovokere en repaint af formen. Så kommen den først når http anmodningen er blevet behandlet. I en seriøs applikation ville dette være utænkeligt. Fejl håndtering er også manglende, hvis internetter eller URI'en ikke kan lokaliseres, er der store problemer og programmet gå højest sandsynligt ned. Der er heller ikke lavet fejlhåndtering ved parsing af XML, hvilket betyder at programmet risikere at gå ned ved uventede XML skemaer. Disse ting skulle meget gerne være i orden før man frigiver et sådant projekt.

Ved evt. videre udvikling, hvilket jeg ikke finder sandsynligt da programmet er skrevet i delphi og derfor ikke er eksistens berettiget. Men skulle man videre udvikle programmet, ville ting som overvågning af RSS feeds bestemt være interessant. Og eller mulighed for at vise kun at nye historier. Men der findes allerede mange andre RSS reader rund omkring i verden, og det er en af de ting der er udsat for meget innovation.