Blekinge Tekniska Högskola
Sektionen för Teknik

Valet av grafiskt dokumentgränssnitt

Kandidatarbete i datavetenskap

Författare: Per Holmberg

Författare: Per Holmberg, per@perholmberg.net
Examinator: Guohua Bai
Handledare: Christer Rindebäck
Version: 2005-09-25

Sammanfattning

Grafiska dokumentgränssnitt används av de flesta datoranvändare. Det finns i huvudsak tre typer av dokumentgränssnitt; MDI (*multiple document interface*), SDI (*single document interface*) och TDI (*tabbed document interface*). I ena halvan av arbetet har dessa tre analyserats utifrån en rad faktorer med målet att se om någon modells fördelar väger över de andras och om man kan skapa några riktlinjer för när det är bäst att applicera respektive modell när man utvecklar en ny programvara. Analysen är baserad på litteratur samt kvalitativ enkätundersökning och programtest. Andra delen av arbetet har ägnats åt testa om det går att programmera så att slutanvändaren kan välja den modell han eller hon föredrar. Ett kodramverk har utvecklats för att bevisa att detta är möjligt.

I grova drag har följande resultat kommit fram: Olika program kräver olika lösningar. SDI är den modell som fungerar i flest lägen. TDI är inte bara trendigt utan också väldigt omtyckt. MDI är oftast det sämsta valet av de tre, men står inte utan fördelar. Att utveckla en kodramverk som möjliggör för utvecklaren att effektivt skapa program där användaren kan välja mellan MDI, SDI och TDI är fullt möjligt.

Abstract

Graphical document interfaces are used by most computer users. There are mainly three document interfaces; MDI (multiple document interface), SDI (single document interface) och TDI (tabbed document interface). In one half of the work these three have been analyzed on the basis of a number of factors with the aim to see if the advantages of any model have a greater weight than the others and if any guidelines can be created for when it is best to apply each model when you develop new software. The analysis is based on literature, qualitative poll survey and user test of software. The other part of the work has been devoted to test if it is possible to code so that the end user can choose the model he or she prefers to use. A code framework have been developed to prove it possible.

In broad outline, these results have been found out: Different software programs require different solutions. SDI the the model that works in the most cases. TDI is not only trendy but also very liked. MDI is in most cases the worst choice of the three, but is not without advantages. To develop a code framework making it possible for the developer to efficient create software programs where the user can choose between MDI, SDI and TDI is fully possible.

Innehållsförteckning

Inledning

Tidigt i den moderna datorns historia fanns behovet att visa dokument på skärmen. Många av de tidiga programmen[1] kunde inte ha flera dokument öppna samtidigt; när ett nytt dokument öppnades stängdes det förra. Om man hade tur kunde man starta ytterligare ett exemplar av programmet och på så sätt gå runt begränsningen att programmet bara kunde ha ett dokument öppet samtidigt.

Det fanns ett behov att ha flertalet öppna dokument i ett och samma uppstartade program (utan att behöva starta upp flera exemplar av programmet). Hur skulle man lösa detta grafiskt? Hur skulle den grafiska relationen vara mellan stommen i programmet och de olika dokumenten programmet hade öppna?

Idag finns det huvudsakligen tre olika lösningar på problemet: Med **MDI** (*multiple document interface*) har varje program ett fönster. I detta fönster finns det en dokumentyta med ett antal underfönster[2], ett fönster för varje dokument. Med **SDI** (*single document interface*) har varje dokument har ett programfönster[3], varje programfönster kan som maximum innehålla ett dokument. **TDI** (*tabbed document interface*) liknar MDI, men dokument-växlingen görs med flikar.

Grogrund – varför detta ämne?

Det började som en programmeringshistoria.

Jag hade programmerat ett program [AnyOSSplits] där man kunde ha flera dokumentfiler öppna samtidigt tack vare MDI-gränssnitt. Vid ett tillfälle (innan denna uppsats påbörjades) läste jag på någonstans på webben att SDI-modellen var tidens melodi. Jag började fundera på om SDI är bättre än MDI. Något övertygande svar var inte lätt att komma på.

Jag implementerade programmet i SDI-modell också. För att inte ”lägga ner” MDI eller skapa två utvecklinggrenar gjorde jag en superklass med de gemensamma gränssnittsfunktionerna samt två underklasser för MDI respektive SDI innehållande den dokumentmodellspecifika koden. När jag var klar så fungerade både MDI och SDI-varianten, och koddelningen mellan dem var så när på maximal. Jag insåg dock att min lösning var tätt knutet till programmet. Att flytta lösningen till en annan program skulle innebära mycket klipp-och-klistra och småjusteringar. Jag ställde mig frågan: Skulle det gå att bryta ut dokument-gränssnittskoden?

1 Program är i denna skrift en kortform av *dataprogram*.

2 Översättning av engelskans *child window*. Ett fönster som ligger inuti ett annat fönster, sitt föräldrafönster.

3 Även kallat *primärfönster (primary window)*.

Frågeställningar

1. Finns det en modell som är bäst? Vad menas då med ”bäst”; bäst för en typ av användare, bäst för en typ av dataprogram…?

2. Kan man på ett effektivt sätt skriva applikationer som möjliggör att användaren kan välja typ av dokumentgränssnitt? Med effektivt menas att utvecklingstiden ej ska öka med programbiblioteket samt att kvalitén på programmen ej ska försämras genom användningen av dito.

Jag kommer ta upp gränssnittsmodellerna MDI, SDI, TDI och kombinationer däremellan. Andra eventuella modeller som kan tänkas finnas samt fördelar och nackdelar med fönstret som sådant kommer jag inte gå in på.

För programbiblioteket kommer jag koncentrera mig på en lösning i Java.

Målgrupp

Alla slags dataanvändare kan förstå det inledande kapitlet där de olika dg[4]-typerna förklaras. Vana användaren som är väl förtrogen med de olika dg-typerna och fönsterterminilogi kan hoppa över detta faktakapitel. Det efterföljande kapitlet som försöker svara på min första frågeställning är riktad till den utvecklare som skall bestämma om han eller hon ska välja MDI, SDI eller TDI för sitt program. Kapitlet som behandlar programbiblioteket är riktat till de utvecklare som vill utvärdera om de kan göra så att användaren kan välja dg-typ alternativt till de utvecklare som vill göra ett sådant system.

Metod

För frågeställning 1 är tillvägagångssättet följande:

- Jämföra modellerna mot varandra faktor för faktor och på det sättet få fram fördelar och nackdelar för varje modell.
- Analysera befintlig litteratur i ämnet. Litteraturen kommer dels vara böcker från högskolebiblioteket, dels internetsidor. Författarna kommer i huvudsak vara ”kända namn” på användargränssnitt.
- Göra en mindre, kvalitativ undersökning bland användare för att ta reda på vad de tycker om de olika dokumentgränssnitten. Undersökningen kommer gå till så att några personer kommer få svara på en webbenkät. Ett par personer kommer att få göra ett programtest.
- Titta på hur olika program är skapade idag och hur man har kommit dit.

För frågeställning 2:

- Designa ett kodramverk/programbibliotek som skiljer applikationskoden från

4 Ordet dokumentgränssnitt är härifrån förkortat till dg.

implementationen av dokumentgränssnittet.

- Implementera en prototyp som man kan växla mellan olika dokument-gränssnitt, utifrån kodramverket ovan.

Interaktionsdesign och användbarhet

Detta kapitel är en introduktion till interaktionsdesign och förklaring av hur jag ser på ämnet. Den är i första hand baserad på [Preece]. I enighet med bokens författare använder jag termen *interaktionsdesign* istället för den smalare termen *människa-datorinteraktion*. Tyngdpunkten har jag lagt på *användbarhet*.

Vad är interaktionsdesign – och varför?

Vi moderna människor använder varje dag massor av interaktiva produkter. Interaktiva produkter är inte bara datorer utan även föremål som fjärrkontroller och kaffebryggare. Vi vill att dessa interaktiva produkter ska vara lätta att manövrera och trevliga att använda. Ofta är dock inte de tekniska prylarna så trevliga och lättanvända. De är dåligt eller otillräckligt designade. Processen att anpassa interaktiva produkter för människors privat- och yrkesliv kallas *interaktionsdesign*.

Från början användes datorer bara av ingenjörer och vetenskapsmän, alltså högutbildade, tekniskt kunniga personer med ett språk och ordförråd som den ”vanliga” människan inte har. När datorer började komma ut bland gemene man (och kvinna) blev det en utmaning att göra datorerna användbara för den stora massan, som inte har fackspråket och den högt tekniska kunskapen. Design av användargränssnitt kom på tapeten. Ett steg var att uppfinna hög-nivå programmeringspråk. Det stora genombrottet kom dock med det grafiska gränssnittet med sina fönster, menyer, ikoner med mera.

När det gäller affärsvärlden är interaktionsdesign en viktig del i utvecklingen av produkter. Om kunden får en dålig upplevelse av produkt eller tjänst på grund av dålig design är det stor risk att kunden inte vill vara kund hos företaget ifråga något mer.

Hur?

Designprocessen kan se ut på lite olika sett, men det är vanligt med följande princip: Först letar man upp vilka behov som finns och ställer upp krav på vad produkten ska kunna utföra. Sedan tar man fram en eller flera designer. Av designen gör man en prototyp, detta kan vara en mockup eller liknande. Det hela utvärderas genom att till exempel låta en användare säga sina synpunkter om prototypen. Utifrån utvärderingen väljer man om man ska gå tillbaka och ändra designen eller godkänna den för produkten.

Användbarhet

Man har flera mål med interaktionsdesign. Centralt är användbarhet. Vad är då användbarhet? Det består av flera saker. Först och främst ska ett system göra vad det är avsett för, det ska ha rätt funktioner för användarens uppgift. För det andra ska användarens uppgift kunna utföras i så få och enkla steg som möjligt. Vidare ska systemet kännas tryggt att använda; användaren ska inte av misstag kunna utföra operationer som han eller hon inte hade för avsikt att utföra. Ett annat viktigt mål är att systemet ska gå lätt och snabbt att lära sig.

Användbarhet har ett antal vanliga principer:

- *Synlighet* är att man ska kunna se vad som kan göras. Till detta hör också att till exempel knappar ska vara placerade på ”rätt” ställe så att användaren hittar dem snabbt.
- *Feedback* är att få bekräftelse på de instruktioner användaren ger till systemet, kunna se vad de innebär och vad som utförs; hålla användaren underrättad med andra ord.
- *Restriktioner* hindrar användaren från att göra saker som är orelevanta, olämpliga eller saker som man kan på förhand säga kommer att misslyckas. Ett exempel av detta i grafiska gränssnitt är menyalternativ som får annat utseende och inte går att välja.
- *Mappning* handlar om förhållandet mellan till exempel knappar och vad de gör. Det handlar också om att till exempel knappar på en fjärrkontroll ska vara logiskt utplacerade (inte bara uppradade i slumpvis ordning).
- Ett gränssnitt som är *konsekvent* är lättare att lära och använda genom att samma kommando alltid utför samma åtgärd och liknande åtgärder har liknande kommandon.
- *Affordance*[5] är att det ska gå att gissa hur man kan manövrera ett objekt genom att till exempel titta på det. En knapp inbjuder till att trycka (klicka) på.

Ett annat mål med interaktionsdesign är skapa en ”upplevelse” för användaren. För vissa applikationer är inte det viktigaste effektivitet utan det ska vara roligt, trevligt och allt annat som kan tänkas göra användaren på glatt humör.

Användarna i centrum

Interaktionsdesign är att anpassa för *användarna*, och då är just användarna en viktig hörnpelare i designprocessen. Utvecklarna av en produkt är sällan samma personer som sedan kommer att använda den och kan därför inte bara designa efter egna preferenser. Användaren ska ställas i centrum. Utvecklaren måste *förstå* användaren och vilka behov denna har. Det första man behöver göra för att ta reda på detta är att *hitta användaren*. När man lyckats med det börjar

5 Ingen bra svensk term hittad.

datainsamlingen över vad användarna tycker. Flera tekniker finns. Man kan titta på när användaren använder systemet (direkt observation), gärna med att användaren ”tänker högt”. Användaren kan skriva ner hur han/hon går tillväga med systemet och vad han/hon tycker. Man kan intervjua användare. Om man vill ha kvantitativ data är enkäter ett utmärkt att samla information.

Att förstå användare är långt ifrån trivialt. Människans tänkande och hjärnverksamhet är en hel vetenskap, en del av psykologin som heter kognition. Hur information presenteras kan vara mycket avgörande för hur snabbt och bra vi kan ta in den. Vi är dåliga på att komma ihåg vissa saker även om vi försöker. Det är mycket lättare att känna igen saker än att minnas dem ”från ingenting”. Designen på ett gränssnitt har stor påverkan på hur människan kan ta in, lära sig och komma ihåg hur gränssnittet ska användas.

Gränssnitt påverkar användare. Ett bra gränssnitt håller användaren på gott humör medan ett dåligt gör användaren frustrerad. Användarens förtroende för systemet och organisationen bakom påverkas också i stor skala av gränssnittet. Exempel på saker som gör användare frustrerade är ofärdiga system, dumma felmeddelanden som inte förklarar hur man ska komma rätta med problemet och instruktioner som är alltför komplicerade.

Metaforer

Gränssnittsmetaforer har visat sig vara mycket framgångsrikt för att hjälpa användare att förstå och lära sig hur ett system ska användas. Detta genom att välkänd kunskap är kombinerad med nya koncept. Den välkända kunskapen är hur saker fungerar in den ”riktiga” världen och den kunskapen drar man nytta av genom att låta objekt på skärmen ha liknande egenskaper. Exempel på vanliga metaforer är skrivbordet, kalkylarket och sökmotorn. Användning av metaforer är dock inte utan fallgropar. Till kritiken hör att metaforer skapar dåliga begränsningar och att metaforerna ”gömmer” hur systemet verkligen fungerar.

Datacentrerad design

Data-centrerad design betyder att användaren ska arbeta med data direkt, snarare än att tänka i program. Med data menas exempelvis ett dokument. Användarens fokus ska ligga på utföra en uppgift, inte på att hitta programmen som behövs för det kunna utföra det. [MSDN]

Om MDI, SDI, TDI samt fönster

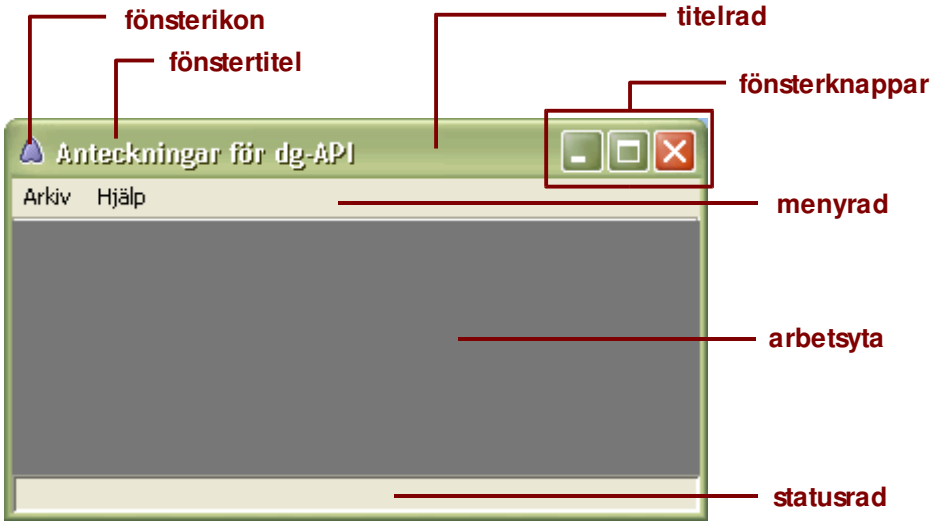
Kort fönsterhistoria

Enligt [Shneiderman, sid 446-449] lutar det åt att det var Doug Engelbart som med systemet NLS uppfann fönstret i det grafiska användargränssnittet på 1960-talet. På 1970-talet utvecklade Xerox PARC (Palo Alto Research Center) sin Smalltalk-miljö där fönster kunde flyttas och överlappas. På 80-talet kom flera fönsterbaserade användargränssnitt som Xerox Star, Apple Lisa och Apple Macintosh. Xerox Star var först med det ”elektroniska skrivbordet”, där man kunde WYSIWYG-redigera dokument.

Microsoft hängde på fönstertrenden med sitt Windows version 1.0, där fönster kunde ställas upp sida vid sida. Möjligheten till överlappande fönster dröjde dock till version 2.0.

År 1990 kom Microsoft med sitt SDK[6] 3.0 för Windows 3.0 som introducerade ett programbibliotek som hade stöd för MDI-applikationer [Crabb]. Möjligheten till underfönster i X Window System (ett fönstersystem för Unix, numera även det vanligaste fönstersystemet för Linux) fanns dock redan i mitten på 80-talet [Scheifler].

Lite fönstertermer



Figur 1: Lite fönstertermer

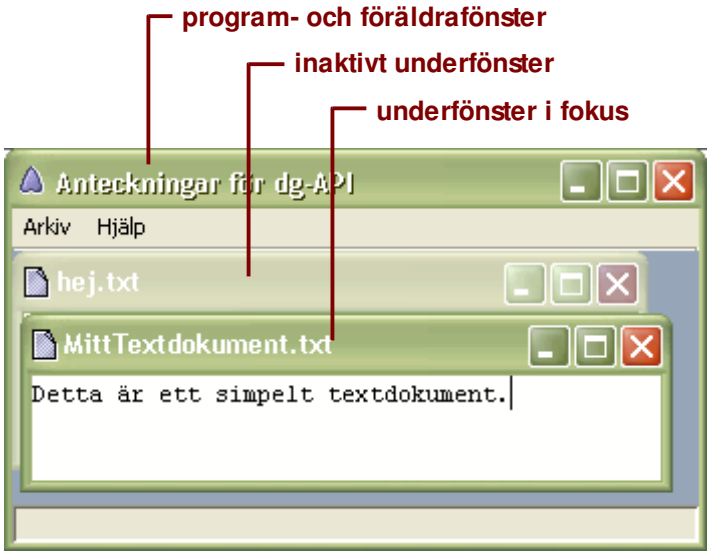
MDI

MDI står för Multiple Document Interface. En MDI-program har *ett* programfönster. I detta fönster finns det en skrivbordsliknande arbetsyta med ett antal fönster, ett fönster för varje dokument. De dokumentåtgärder man utför i

6 Förkortning av *Software Development Kit*.

det gemensamma programfönstret (till exempel ”spara”) appliceras på det dokument vars fönster ligger överst i arbetsytan. Flera dokument delar i regel på de operativa elementen (till exempel menyer och verktygsfält).

Ett MDI-föräldrafönster[7] har max ett dokumentfönster (underfönster) *aktivt*. Ett aktivt dokumentfönster ligger över andra dokumentfönster och har fokus.



Figur 2: MDI-gränssnitt med några termer förklarade.

Dokumentfönsterna i en MDI-program är oftast kompletta fönster i det avseendet att de har ram, titelrad med titel, fönsterikon[8] samt knappar för att minimera, maximera och stänga fönstret. Några skillnader finns dock från ett fönster i till exempel ett SDI-program:

- Dokumentfönstret kan inte flyttas utanför föräldrafönstret (programfönstret).
- Dokumentfönstret kan inte växlas till genom aktivitetsfältet eftersom det aldrig listas där.
- Om man minimerar ett dokumentfönster blir det till en ikon i programmets skrivbordsyta, likt minimerade fönster la sig på bakgrundsbilden i Windows 3.0.
- När man maximerar ett dokumentfönster maximeras det inte över hela skärmen, utan endast över programmets arbetsyta.

Om man ska summera ovanstående märker man att ett programfönsters skrivbordsyta uppför sig som ett eget fönstersystem i (Windows) fönstersystemet. Alltså en väldigt programcentrerad modell att se på dokument (jämför med SDI senare).

Föräldrafönstret har en ikon som representerar programmet, underfönsterna (dokumentfönsterna) har en dokumentikon.

Fönstertiteln innehåller förutom programnamnet ofta (men inte alltid),

7 Från engelskans *parent window*.

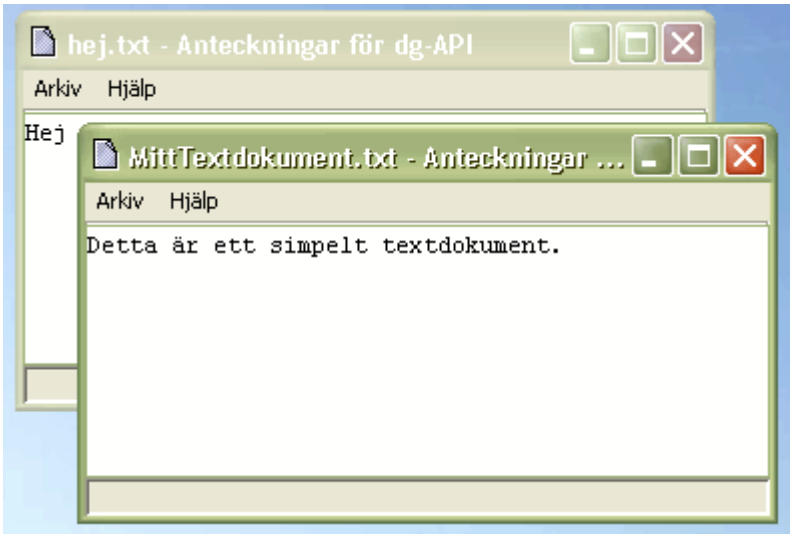
8 Windows fönsterikoner är vanligtvis 16 x 16 pixlar stora.

namnet på det dokument som är aktivt. I vilken ordning programnamn och dokumentnamn förekommer varierar.

Exempel på program som använder MDI är Acrobat Reader 6.0.

SDI

SDI står för Single Document Interface. I ett SDI-program har varje dokument har ett programfönster, varje programfönster kan som maximum innehålla ett dokument. För den vanliga användaren kan det se ut som att varje dokument är öppnat i ett helt nytt exemplar av programmet. Att varje dokument faktiskt öppnas i ett nytt exemplar av programmet är en enkel om ej minnesoptimal implementationlösning av SDI.



Figur 3: SDI-gränssnitt

SDI är en dokument-centrerad modell att se på dokument (mer om detta i senare kapitel).

I Windows har varje dokument i ett SDI-system en egen post i aktivitetsfältet.

När man öppnar ett dokument i ett SDI-program öppnas dokumentet i ett nytt programfönster om fönstret man öppnar från inte innehåller ett öppet dokument. Oftast är det så att ett programfönster är utan dokument endast om *inga* dokument är öppna i programmet.

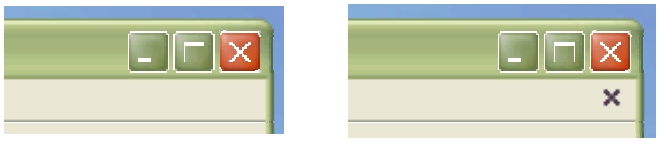
Ett SDI-fönster har vanligtvis en dokumentikon som fönsterikon. Undantaget är när inget dokument är öppet, då visas vanligtvis istället en ikon som representerar programmet.

Enligt Microsofts riktlinjer ska dokumentnamnet komma före programnamnet i fönstertiteln.

När man har flera dokument öppna i ett SDI-system så stänger man dokument genom att klicka på respektive programsfönsters stängknapp. Programfönstret växlar ofta mellan att ha en eller två stängknappar.

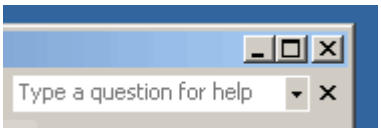
Exempel på program som använder SDI är OpenOffice 1.1. I OpenOffice 1.1 är

det så att om flera dokument är öppna så finns bara en stängknapp. Om bara ett dokument är öppet finns det två stängknappar, en ordinarie stängknapp som stänger både dokument och programmet medan den extra stängknappen bara stänger dokumentet utan att avsluta programmet.



Figur 4: Stängningsknappar i OpenOffice 1.1; till vänster med flera dokument öppna, till höger med ett dokument öppet.

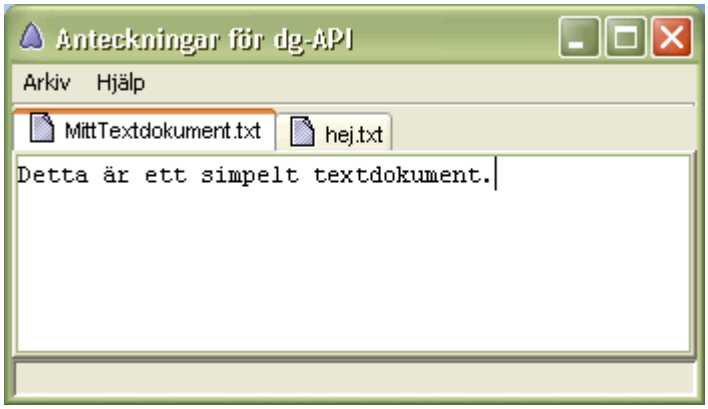
I Word 2002 har fönstret två knappar oavsett om flera eller bara ett dokument är öppet. När flera dokument är öppna har de båda stängknapparna samma funktion; att stänga dokumentet utan att avsluta programmet.



Figur 5: Stängknappar i Word 2002.

TDI

TDI står för Tabbed Document Interface. I likhet med MDI har så använder TDI bara ett programfönster, men i arbetsytan finns en rad med flikar för att välja aktivt dokument. I den enklaste varianten av TDI kan bara ett dokument visas åt gången och resten av arbetsytan tas upp av aktivt dokument.



Figur 6: TDI-gränssnitt

TDI-gränssnitt är också känt som arbetsböcker (*workbooks*), efter liknelsen med ett anteckningsblock med flikar.

Ikonanvändingen för TDI är i princip samma som för MDI; fönstret har programikon och flikarna har dokumentikoner.

TDI/MDI-kombination

TDI och MDI har mycket gemensamt, ibland så mycket att det kan vara svårt att veta om man ska kategorisera ett program till MDI eller TDI. Jag talar om program som har både MDI-underfönster *och* flikar.

Exempel på program som både har MDI-fönster[9] och flikar är Opera och Textpad (för Opera en sanning med modifikation, se nedan). I fallet Opera finns historia. I tidigare versioner (till och med 6.x) hade Opera ett MDI-gränssnitt. För att det skulle bli snabbt att växla mellan dokumentfönster hade man ett valfritt verktygsfält med *knappar*, en knapp för varje MDI-fönster. En klick på en knapp la respektive dokumentfönster överst, precis som flikar i TDI. Vid introduktionen av Opera 7 bytte man utseende på knapparna till flikar. Flikraden kallas *page bar* i Opera.

TDI/MDI-kombination är även känt som *MTI; MDI tabbed interface*. [Dundas]

MDI/SDI/TDI-kombination

Med en liten inställningsändring kan man få Opera att uppföra sig nästan som som ett SDI-program och det utan att ens starta om programmet. Det är möjligt genom att Operas gränssnitt både har de typiska igenkänningstecknena för både MDI, SDI och TDI. En inställning avgör om uppförandet ska vara mer åt SDI-håller eller mot MDI/TDI-hållet.

9 Även känt som *underfönster*.

Ett bästa dokumentgränssnitt?

Om man som utvecklare ska välja en typ av dg för sitt program, vilken typ av dg ska man då välja?

Vad gör en modell bäst?

Jag vill hitta en modell som är bäst, men för att försöka utse en dg-modell som är bäst krävs en förklaring av vad som menas med ”bäst”.

En modell kan vara bäst för en typ av användare. Ett gränssnitt som är effektivast för nybörjaren behöver inte nödvändigtvis vara effektivast för den vana användaren. Relaterat till detta är också vilken modell som användaren arbetat med innan; det inarbetade kan kännas bäst. Kännas bäst kan också vara den modell som överensstämmer med en metafor.

En modell kan vara bäst för en typ av dataprogram. Nedan kommer det nämnas två ”klasser” av program; applikationer och verktyg.

En modell kan vara bäst på så sätt att det är resurseffektivast, alltså tar minst resurser av datorn, framför allt i minnesväg. Ingen människa gillar när datorer är långsamma så genom att göra program resurssnåla och därmed snabba ökar man användarens belåtenhet.

Gemensamt för alla betraktningssätt ovan är att de i slutändan handlar om användbarhet.

Litteratur om val av dokumentgränssnitt

Det har varit mycket debatt om vilken dokumentgränssnittsmodell som är att föredra [Wikipedia MDI]. Trots detta har jag inte hittat så mycket litteratur som faktiskt argumenterar om ämnet.

McKays tillvägagångssätt

Everett McKay föreslår i *Developing User Interfaces for Microsoft Windows* att man ska särskilja programmet till en av följande två klasser; *applikationer* och *verktyg*. Applikationer är de program som man använder under en längre tid medan verktyg är program man bara använder några minuter per gång, oavsett hur sofistikerat programmet är [McKay].

När man väl avgjort om ett program är en applikation eller verktyg, ger McKay följande råd: En applikation bör använda antingen MDI *eller* SDI. Ett verktyg bör använda SDI. MDI är för komplext för att användas på ett verktyg.

McKay är noggrann med att poängtera att när SDI används med program med ”faktiska” dokument så måste det vara möjligt att köra flera instanser samtidigt, det vill säga ha flera dokument öppna samtidigt.

TDI nämns inte i diskussionen.

Val efter användarvana

McKay blandar även in användarvana i sitt resonemang. McKay varnar för lägga fokus på att anpassa applikationsprogram efter mindre vana användare; eftersom användare lägger så mycket tid på applikationer blir de snabbt mer avancerade användare. Som en konsekvens av detta bör applikationer i första hand designas för avancerade användare [McKay].

MSDN Library

MSDN Library diskuterar en del kompromisser man gör när man använder MDI respektive TDI. TDI kallas i MSDN Library för *workbooks* och det talas om vyer snarare än dokument.

Ett par fördelar för MDI nämns. Ett av dessa är att gränssnittskomponenter som menyer, verktygsfält och statusrader delas mellan dokumenten vilket gör MDI utrymmeseffektivt. Nackdelarna väger över i antal. Om flera dokument är öppna i samma program så går det inte att växla mellan dokumenten med varken Alt+Tab eller aktivitetsfältet. Det är inte så lätt att se vilka dokument som är öppna. Användaren kan ha svårt att ha särskilja på underfönster (*child windows*) och ett vanliga primärfönster[10] [MSDN]. Några ytterligare argument för och emot nämns, som dock inte är lika lättförstådda.

Fördelarna som nämns för TDI är att TDI sparar skärmyta genom delningen av gränssnittskomponenter samt att navigeringen mellan vyer är snabb. Den nackdel som nämns är att flera vyer inte kan visas samtidigt [MSDN].

Apple Mac OS riktlinjer

I jämförelse med Microsoft kan man se en mer rakare stil i riktlinjer från Apple. De är starka förespråkare till SDI. Apple propagerar: Använd inte MDI. MDI som används i Microsoft Windows går direkt emot riktlinjerna för Mac OS X. Fönster i Mac OS X är dokument-centriska och inte applikations-centriska. [Apple 2004a]

När det gäller dokumentfönster så har Mac OS har några intressanta detaljer som inte finns i Windows. Det finns en särskild typ av fönster för dokument. Ett dokumentfönster kan ha en *proxy icon* i titelraden.

> *After pressing a proxy icon for a brief period, users can manipulate it as if they were manipulating the corresponding file-system object.* [Apple 2004b]

Proxyikonen kan dras till ett annat program på ett liknande sätt som filer dras i Utforskaren i Windows. När man har gjort ändringar i dokumentet som man inte har sparat är proxyikonen dämpad för att visa detta. Mac-fönster har även ett annat sätt att visa osparade ändringar; stängknappen ändrar utseende. Ett ändrat dokument får en prick i sin röda stängknapp.

10 Översättning av engelskans *primary window*. Ofta synonymt med *programfönster*.

Befintliga programvaror

Hur har man gjort för de program som finns på marknaden idag? Här tas det upp några vanliga program.

Opera

Opera Software tycker att deras MDI/TDI-kombination är det bästa för deras webbläsare, men för att inte stöta bort användare hemma på SDI så införde man i Opera 6.0 möjligheten att välja mellan SDI och MDI genom en fönster som visades vid uppstart [Opera2001].

Från Opera förväntade man sig att SDI-modellen bara skulle användas under en övergångsperiod för användarna; SDI skulle göra det lättare för användare av Internet Explorer och Netscape att gå över Opera. Användarna skulle med tiden gå över till Operas MDI-gränssnitt [Opera2001].

Till Opera 7 ändrade man sig lite (igen) genom att baka ihop SDI och MDI. Den nya MDI/SDI-kombinationen ska ge användaren ”det bästa av två världar” genom att surfa med både MDI, SDI och flikar, och det utan att starta om programmet [Opera].

Internet Explorer

Internet Explorer 6 är den enda stora webbläsare idag som strikt håller sig till SDI. Microsoft verkar inte på något sätt förespråka att TDI används i ett sammanhang med ett-till-ett-förhållande mellan dokument och flik. Det används när detta skrivs inte i något av företagets program riktat till ”vanliga” användare (dock används en flik-liknande modell i Visual Studio, för utvecklare).

I mars kom det ett rykte att version 7 kommer att ha *tabbed browsing*, alltså TDI [Sherriff]. Ryktet bekräftades i maj i [IEBlog]. Den senare källan berättar också att anledningen varför man inte infört flikar tidigare är för att man varit oroad för att det skulle vara förvirrande att IE har flikar när inte andra Windows-program (från det egna företaget) har det.

Microsoft Office

Microsoft har bytt mellan MDI och SDI för sitt Office-paket flera gånger [Wikipedia MDI]. Till Word 2000 bytte man från MDI till SDI [Rosenbaum].

Även inom samma Officeversion finns inkonsistens, i 2002-utgåvan använder de olika programmen olika dg-modeller. Word och PowerPoint använder sig av SDI medan Excel använder sig av MDI.

Enkätundersökning

En liten kvalitativ enkätundersökning genomfördes. Jag bjöd in ett antal personer från min egna adressbok att svara. Fem svar kom in. Dessutom genomfördes en intervju med ungefär samma frågor. Frågorna handlade om vilka dg-modeller de var bekanta med och vilken modell de föredrog. Exakta

utformningen av enkäten finns som bilaga.

Tre av de fyra som använde och kände igen TDI tyckte den modellen var bäst. Argumenten inkluderade:

- Man får en bra och enkel överblick över vilka filer man har öppna.
- Snabb och enkel växling mellan filerna.
- Den modell där det är enklast att ”hålla ordning” bland dokumenten.
- Det blir inte så många knappar i aktivitetsfältet.

Dock kan man med med TDI inte ha två dokument bredvid varanda och jämföra, vilket nämndes till MDI:s fördel. Till SDI:s fördel nämndes att det är ”bäst om man i Windows vill nå ett öppet dokument på bara ett klick”.

Två personer poängterade vanans makt – man gillade en modell bäst av ”gammal vana”. En person skrev, utan att närmare förklara, att bästa dg:et berodde på vilket program det är som används.

De två som hade minst datorvana än de andra kände endast igen SDI utifrån textbeskrivningar och skärmdumpar. Trots detta listade den ena av dessa personer ett TDI-program i listan av program personen brukade använda, den andra ett MDI-program.

På frågan vad de trodde på idén där användaren kan välja vilken modell han eller hon vill välja för ett program var alla mycket positiva till konceptet.

Som sista fråga fick personerna tycka fritt om något de tyckte var relaterat till ämnet. Mikael Gustavsson kom med två intressanta åsikter:

”Framför allt måste det finnas snabbkommandon för att byta dokument i en vy.”

”Att kunna växla mellan dokument med hjälp av kontextmenyn vid högerklick kanske vore en uppskattad funktion som jag tror skulle underlätta ytterligare. Man behöver då inte flytta runt musen så mycket. ”

Programtest

Jag bjöd in orienterare att testa ett program för att analysera sträcktider [AnyOSSplits]. Programmet använder programbiblioteket från detta arbete och kan därmed växlas mellan de olika dg-modellerna. Uppgiften som gavs var att testa de tre modellerna och tycka vilken som var bäst. Det var trögt att få folk att ställa upp, endast tre personer lyckades jag att få att göra så.

Person 1 och 2, som lämnade omdöme tillsammans, röstade på TDI. I fas med enkätundersökningen var det argumenten enkel filväxling och bra överblick primära. En mer originell betraktelse var att det var positivt att flikarna (och därmed dokumentväxlingen) var i ovandelen av fönstret eftersom det är där ”man ju oftast arbetar med musen”.

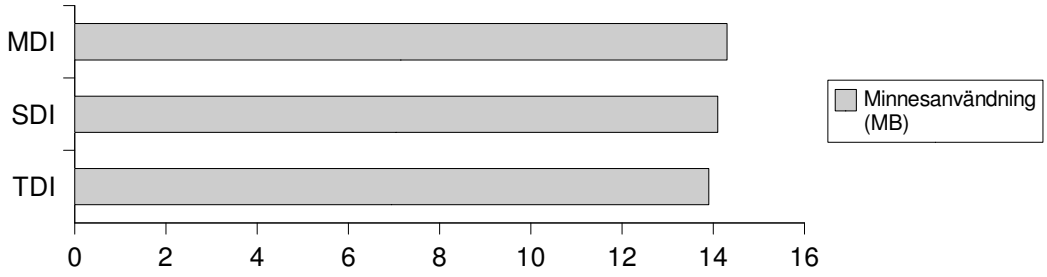
Person 3 tyckte att TDI-varianten var ”kanonbra” och gav bra överblick. I SDI-varianten upplevde personen att det var svårt att byta mellan dokumenten. MDI-varianten upplevdes som ”helt fruktansvärd”, vilket delvis dock kan bero på att personen saknade en Fönster-meny för att dokumentväxlingen, vilket brukar

finnas i MDI-program.

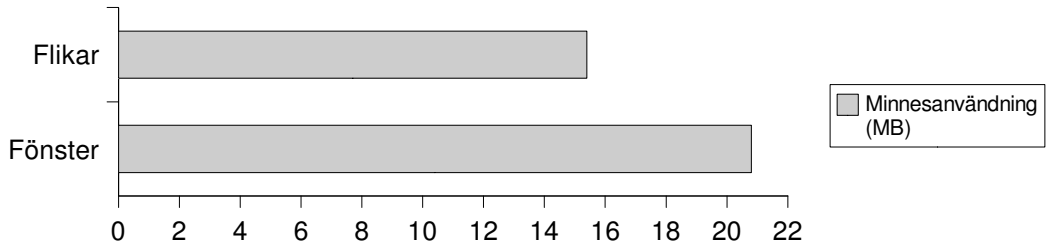
Ett enkelt minnestest

Mer resurs- och minneskrävande program tar längre tid att starta och om användningen av arbetsminne är tillräckligt hög måste operativsystemet använda virtuellt minne vilket gör programmet långsamt. Långsamma program är en källa till frustration, därför är resurseffektivitet en sak att tänka på.

Med demoprogrammet för programbiblioteket som tas fram i nästa kapitel lät jag öppna 20 dokument i MDI, SDI respektive TDI och antecknade minnesanvändningen. Java-programmet kördes med Sun Java 1.4.2 under Windows XP. Skillnaden i minnesanvändningen mellan de tre modellerna var liten:



Med Opera 7.54 gjorde jag ett liknande test; jämförde 20 flikar med tomma dokument i ett fönster mot 20 fönster med ett tomt dokument i varje.



Analys

Av den litteratur jag har tittat på är det McKay och Apple de två som kommer med de mest praktiska och raka råden; McKay där det främst är *hur lång tid* man använder programmet per gång som är avgörande, Apple med sitt enkla men logiska yttrande för SDI.

Faktorer

Om jag ska försöka utse en modell som är bäst jämtöver, måste jag gå in på mer detalj och gå igenom de faktorer som finns.

Fönsterhantering

MDI:s fönster-i-fönster-grej kan i allmänhet vara svår att förstå för nybörjaranvändaren. Exempelvis, som antytt i [MSDN], kanske inte användaren märker eller förstår att ett dokumentfönster är underfönster utan tror i själva verket att dokumentfönstret är ”vanligt” primärfönster som fungerar likadant programfönstret.

Med MDI kan ett dokumentfönster kan aldrig flyttas så det någongång, helt eller delvis, ligger utanför föräldrafönstret. Underfönster är ”fångade” i programfönstret.

SDI har en betydligt enklare fönsterhantering, vilket är bra för den mindre vana användaren.

Dokumentväxling

MDI har fördelen att det blir mindre antal poster i aktivitetsfältet då endast en post för programmet visas i aktivitetsfältet, oavsett antal dokument som är öppnade i programmet.

Å andra sidan har detta sin nackdel: Det går inte växla mellan dokument varken med Alt+Tab eller med aktivitetsfältet. [MSDN]

Det är dock inte alltid MDI är på detta sätt. Ta Microsoft Excel 2002. Där har varje dokument, MDI-modellen till trots, en post i aktivitetsfältet. Program-fönstret självt finns dock inte i aktivitetsfältet så länge det finns minst ett dokument öppet. Konstigt/ironiskt nog så nämns inte denna typ av undantag i Microsofts egna [MSDN].

Det skulle inte gå att skriva ett Java-program för att uppnå samma effekt som i Excel 2002. Javas API och Windows-implementation är sådan att programfönster och poster i aktivitetsfältet alltid har ett-till-ett-förhållande. Att koppla MDI underfönster till poster i aktivitetsfältet är i dagsläget inte möjligt genom Javas API.

TDI erbjuder en superb växling mellan dokument så länge man befinner sig i samma program.

Operationer över flera dokument

När operationer ska utföras på flera eller alla dokument så är MDI logiskt.

Säg till exempel en operation ”Spara alla”, som kan vara mycket nyttig. Om du väljer ”Spara alla” i ett MDI programfönster så är det naturligt att kommandot appliceras på alla underfönster. I SDI skulle detta inte vara lika naturligt eftersom dess dokumentcentrerade modell gör det svårare att avgöra vilka dokument som tillhör programmet.

Samtida visning

Endast ett dokument kan visas samtidigt i TDI:s enklaste form, en klar nackdel. Med SDI och MDI kan flera dokument visas samtidigt på skärmen.

Utrymmeseffektivitet

MDI kan vara utrymmeseffektivt på så sätt att alla dokument delar på gränssnittskomponenter som verktygs-, meny- och statusrader.

Detta gäller då man har flera dokument öppna samtidigt i samma program och dokumenten visas sida vid sida. I annat fall så blir det ett-till-ett-förhållande mellan program och dokument, ur visningssynpunkt.

Med dagens stora skärmar är borde dock inte ett par menyrader och liknande ha så stor skillnad. Å andra sidan känns ett utrymmeseffektivt gränssnitt mera ”rent” – och det är ju allmänt erkänt att sådan enkelhet är positivt att sträva efter.

Stängning

Som sagt i förra kapitlet, om bara ett dokument är öppet i OpenOffice (som använder SDI) finns det två stängknappar, en ordinarie stängknapp som stänger både dokument och programmet medan den extra stängknappen bara stänger dokumentet utan att avsluta programmet.

För den person som är inkörd på den dokument-centrerade modellen faller sig detta logiskt; ordinarie stängknappen på fönsterna stänger alltid dokumentet. För den person som tänker i program kanske det inte faller sig lika logiskt; ett klick på stängknappen stänger i vissa fall (ett dokument öppet) programmet medan i andra fall (flera dokument öppna) *inte* programmet.

Eftersom Word 2002 har två stängknappar även vid flera dokument öppna blir det alltid att en knapp stänger fönstret (ordinarie) och en som stänger dokumentet utan att avsluta programmet (den extra). Den förvirrande faktorn blir då istället att de båda stängknapparna får samma praktiska betydelse när flera dokument är öppna i programmet.

Resurseffektivitet

Lite förvånande var skillnaden i minnesanvändningen liten mellan de olika modellerna på det enkla minnestestet på ett Java-program; TDI tog marginellt mindre minne än SDI och SDI. Ett mer förväntat resultat uppenbarade sig på testet med Opera; flikar tog betydligt mindre minne än fönster.

Det *kan* alltså bli en klar minnesvinst med att använda en ett-fönsterlösning, men minnesvinsten kan också bli marginell.

Datacentrerad design

Det dg som bäst passar in på datacentrerade designmodellen är SDI. Detta för att i SDI har varje dokument ett eget fönster, till skillnad från MDI och TDI där dokumenten blir grupperade efter program, vilket ger en fokus på program som skulle undvikas.

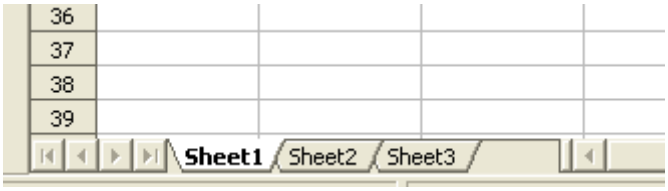
Fönstermetaforen

När det gäller dg är fönster den stora metaforen. TDI använder metaforen av en bok eller anteckningsbok.

När det gäller MDI kan man ställa sig frågan: Hur ofta brukar det finnas fönster i andra fönster? Å andra sidan, vem har fönster på sitt skrivbord?

Flikar – för annat ändamål?

[MSDN] tar upp flikar, men inte för växling av dokument utan för att växla mellan olika sektioner/vyer i samma dokument. Exempelvis kan det finnas en flik för varje ark i ett kalkyldokument, se bild.



Figur 7: Flikar i ett kalkyldokument.

Det kan tänkas att det för användaren är knepigt om samma grafiska komponent används för både växling av dokument och sektioner, särskilt om det är i ett och samma program. Genom att ha flikar för dokumentväxling i överkanten av arbetsytan (som till exempel Firefox) och flikar för sektionsväxling i nederkanten (som till exempel OpenOffice) borde begreppen dock med stor sannolikhet kunna särskiljas.

Summering av faktorer

I ett försök att summera faktorerna ovan och ta fram en dg-modell som är ”bäst” jämtöver görs här en matris där varje faktor viktas och betygssätts för respektive dg-modell.

Betygssättningen görs i en tregradig skala. ”OK” betyder att faktorn uppfylls men med vissa begränsningar. ”Bra” betyder faktorn uppfylls på ett betydande bättre sätt än OK. Det tredje betyget är dåligt/saknas då faktorn uppfylls dåligt eller helt saknas. För poängberäkning översätts dåligt/saknas till 0, ”OK” till 1 och ”bra” till 2. Denna siffra multipliceras med vikten för faktorn. Till slut summeras dessa produkter för att få en poäng för hela dg-modellen.

Betygen har satts på bas av argumentationen/analysen ovan, så motiveringarna här hålls korta.

Betygsättning

MDI:s fönsterhantering har konstaterats betydligt krångligare än TDI:s och SDI:s, därför får MDI bara OK på faktorn medan de andra får Bra. TDI:s dokumentväxling är överlägsen så länge man håller sig inom samma program, och betyget blir därmed Bra medan de andra modellerna får OK. När det gäller dokumentväxlingen med olika program är det dock SDI som utmärker sig där endast ett klick i aktivitetsfältet krävs för att utföra bytet.

MDI och TDI har en gruppering av dokument efter program, varav operationer som ”Spara alla” faller sig Bra. För data-centrerade SDI passar det

inte alls. MDI och TDI kan visa två dokument från samma program sida vid sida, det går inte med TDI. MDI och TDI återanvänder verktygsfält, menyer och liknande och använder skärmytan därmed lite mer effektivt (Bra) än SDI (OK).

MDI har en klar distinktion mellan att stänga program och dokument (Bra), i SDI är det inte lika klart (OK). TDI är den modell som oftast verkar ta minst minne (Bra), även om skillnaden till de andra kan vara marginell (OK). SDI är den enda av modellerna som har en data-centrerad design (Bra).

McKay tycker både MDI och SDI är lämpligt för applikationer (enligt McKays egna definition) (Bra). TDI nämner han inte, och TDI får därmed kompromissbetyget OK. För den andra sortens program; verktyg, tycker han SDI är det rätta (Bra) medan MDI är direkt olämpligt. TDI nämns inte här heller, kompromissbetyg (OK) igen. Flikar används för andra saker i det grafiska gränssnittet än som i TDI, därav en möjlig ändåmålskonflikt (OK).

Enligt Mac OS riktlinjer ska SDI användas (OK), MDI avrekommenderas starkt. TDI nämns inte, kompromissbetyg.

Matris

Faktor	**Vikt**	**MDI**		**SDI**		**TDI**	
Fönsterhantering	1	OK	1	Bra	2	Bra	2
Dokumentväxling, samma program	1	OK	1	OK	1	Bra	2
Dokumentväxling, olika program	1	OK	1	Bra	2	OK	1
Flerdokumentsoperationer	1	Bra	2	-	0	Bra	2
Samtida visning	1	OK	1	OK	1	-	0
Utrymmeseffektivitet	1	Bra	2	OK	1	Bra	2
Stängning	1	Bra	2	OK	1	Bra	2
Resurseffektivitet	1	OK	1	OK	1	Bra	2
Data-centrerad design	1	-	0	Bra	2	-	0
Lämplighet för applikationer	1	Bra	2	Bra	2	OK	1
Lämplighet för verktyg	1	-	0	Bra	2	OK	1
Ändamålskonflikt	1	Bra	2	Bra	2	OK	1
Kompatibel Mac OS riktlinjer	1	-	0	Bra	2	OK	1
Summa poäng			**15**		**19**		**17**

Tabell 1: Analysmatris med samma vikt på alla egenskaper.

Baserat på detta är vinnaren... SDI!

I avsaknad på vetenskapliga motiveringar på viktning har varje egenskap fått samma vikt. Om jag viktar egenskaperna enligt eget tycke blir rangordningen samma, men skillnaden mellan modellerna större:

Faktor	Vikt	MDI		SDI		TDI	
Fönsterhantering	3	OK	3	Bra	6	Bra	6
Dokumentväxling, samma program	3	OK	3	OK	3	Bra	6
Dokumentväxling, olika program	2	OK	2	Bra	4	OK	2
Flerdokumentsoperationer	1	Bra	2	-	0	Bra	2
Samtida visning	2	OK	2	OK	2	-	0
Utrymmeseffektivitet	1	Bra	2	OK	1	Bra	2
Stängning	2	Bra	4	OK	2	Bra	4
Resurseffektivitet	1	OK	1	OK	1	Bra	2
Data-centrerad design	2	-	0	Bra	4	-	0
Lämplighet för applikationer	1	Bra	2	Bra	2	OK	1
Lämplighet för verktyg	1	-	0	Bra	2	OK	1
Ändamålskonflikt	1	Bra	2	Bra	2	OK	1
Kompatibel Mac OS riktlinjer	2	-	0	Bra	4	OK	2
Summa poäng			**23**		**33**		**29**

Tabell 2: Analysmatris med personlig viktning.

För den som vill sätta egna vikter finns en redigerbar kalkylfil att ladda ner från: http://perholmberg.net/2004/kandidatarbete/

Alla vägar leder till Microsoft?

Av den litteratur jag har hittat så är Microsoft inblandad i en betydande del. [MSDN] är Microsoft, [Galitz] baserar sitt på [MSDN]. [McKay] är utgiven på Microsoft Press, hävdande dock att Microsoft inte gjorde något försök att censurera hans kritik av Microsoft-produkter.

Microsoft lägger ner pengar i användbarhetsforskning som få andra, så såvida man inte är för konspiratorisk av sig borde detta betyda deras slutsatser är väl begrundade.

Samtidigt finns det vissa glapp i argumentationen. [MSDN] säger att det är dåligt att dokument grupperas efter program (MDI), utan att ge en direkt förklaring *varför* det är dåligt.

Oavsett vad Microsoft tycker kommer den enorma spridningen av deras program att göra att vad Microsoft förespråkar bli någon ”standard”.

Utveckling av en dg-API

I tidigare avsnitt har valet av dokumentgränssnitt gjorts genom att bestämma modell vid programmets skapande. Frågan är: Är det nödvändigt att bestämma sig för en modell vid applikationens[11] utvecklande? Kan man låta slutanvändaren välja, när programmet ska användas?

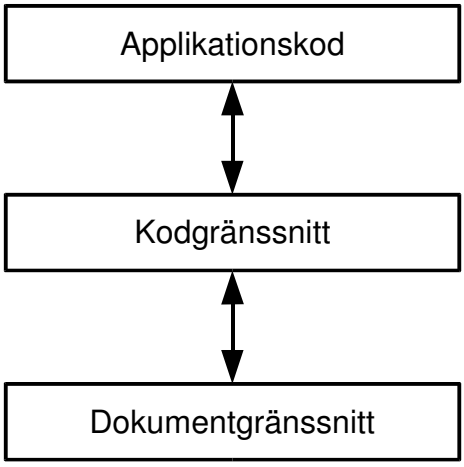
Ett sätt vore att programmera ett grafiskt gränssnitt för endera dokumentgränssnitt man vill att användaren ska kunna välja mellan. Det är dock tveksamt om man vill lägga energi/pengar på något sådant i utvecklingsprojekt. Proceduren skulle i mångt och mycket behövas göras om varje program också.

Dokumentgränssnitt som ADT

Kan man på ett effektivt sätt programmera så att dokumentgränssnittet kan bytas ut utan att ändra applikationsspecifik kod? En lösning är att låta dokumentgränssnittet vara en ADT. Så här beskrivs ADT [Susning ADT]:

> *En ADT är en beskrivning av en mängd data och operationer som är helt oberoende av programspråk. Abstrakta datatyper är abstrakta i den bemärkelsen att det inte finns definierat hur operationerna är implementerade.*

Idén är att låta applikationskoden arbeta med ADT:n oavsett vilken typ av dokumentgränssnitt det är eller hur det är implementerat. Implementationen av ADT kan bytas ut utan nämnvärda ändringar i applikationskoden. En annan fördel är att dg-implementationer skulle kunna användas till många typer av applikationer, vilket banar för kvalité genom kodåteranvändning.



Figur 8: Konceptuell arkitekturskiss

Kort sagt: Jag vill göra en API[12]!

11 *Applikation* är i detta kapitel synonym för *program*.

12 Förkortning av *Application Programming Interface*. Ett programbibliotek med (i sammanhanget) vanliga operationer.

Befintliga dg-implementationer

Jag har inte hittat någon befintlig (öppen) design/implementation av en dg-API som har de ADT-egenskaper beskrivna ovan. Partiella dg-implementationer som faller utanför ADT-klassen finns det dock flera. Dessa är dock främst fönsterhanteringssystem (*window management systems*), till exempel i Java finns klasserna JDesktopPane och JInternalFrame som implementerar MDI-fönster.

Exempel på minimala implementationer i Java av MDI och SDI var för sig finns i [Springmann].

Vissa kommersiella applikationer kan växla dg-typ vilket antyder på en någon slags stängd, propetariär API.

Design av en dg-API

För att formellt beskriva design använder jag UML[13]-notation (se till exempel [Larman]).

Min API-lösning kommer vara lite skriven runt språket Java och befintliga klasser i det språket. Med ett annat språk kanske en annan struktur faller sig mer naturligt.

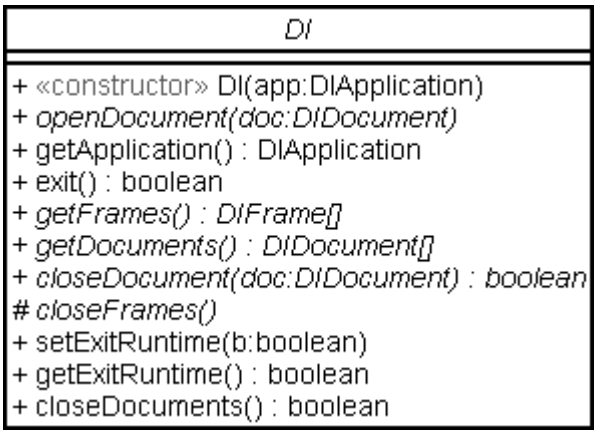
En första klass

För att veta hur en dg-API ska utformas utgår jag från ADT-definitionen. Vilken *mängd data* och vilka *operationer* ska finnas?

För det första måste man kunna **öppna ett dokument**. Med öppna dokument menar jag här att öppna dokumentet i dokumentgränssnittet, inte att läsa in en dokument från fil eller liknande. Förr eller senare är det sannolikt att samma dokument ska **stängas**. Till slut ska förmodligen **hela dg:et stängas**.

Det var de tre huvudsakliga operationerna, nu över till data. När ett dokument öppnas eller stängs ändras listan över **öppna dokument**. Dg:et kommer tillsammans med applikationen att ha ett antal **applikationsfönster** öppna. UML-klass:

13 *Unified Modeling Language*, ett välspritt språk för att (halv)grafiskt beskriva den kodmässiga designen på ett program.



Figur 9

DI är en förkortning av *Document Interface*.

När man avslutar ett hela dg:et är det vanligt att körningen av hela programmet är klart och programmet bör laddas ur ur minnet och så. Därför har jag behändighetsmetod *setExitRuntime( )* som, om satt till sann, terminerar programprocessen när dg:et avslutas.

Ett dg-instans och en applikationskodinstans behöver kommunicera med varandra och måste därmed hålla minst en referens till varandra, därför användningen av *DIApplication* här.

Applikations- och dokumentegenskaper

Dg:et behöver veta ett antal egenskaper från applikationen och dess dokument för att kunna göra sitt arbete.

- **Applikationens namn** står i applikationsfönstret titelrad, som sköts av dg:et.
- En **applikationsikon** visas i vissa fall vid applikationsfönstret titel.
- **Dokumenttitel** används i dg:et. I MDI står titeln på MDI-fönstret ram, i SDI står titeln i applikationsfönstrets ram och i TDI står titeln på fliken för respektive dokument. Dokumenttiteln kan ändras under körning, exempelvis att genom att användaren kan ändra dokumentets faktiska titel eller genom att filen sparas om under nytt namn.
- En särskild **dokumentikon** kan finnas. Det kan användas flera olika dokumentikoner i samma applikation, till exempel i OpenOffice är det en ikon för ordbehandlingsdokument, en ikon för för kalkylark och så vidare.
- Den **grafiska presentationen och interaktionen** av ett dokument tar vanligtvis upp större delen ett applikationsfönstret. Ofta kan man som bekant WYSIWYG-redigera dokumentet genom denna yta. Mer generellt kan man säga att en sådan här yta sköts av en GUI[14]-komponent. Superklassen för GUI-komponenter i Java är *java.awt.Component*.
- Ett dokument bör kunna motsäga sig från att bli stängt. Det kan hindra användaren från att oavsiktligt stänga ett dokument med osparade ändringar.

14 Förkortning av *Graphical User Interface*, grafiskt användargränssnitt.

En dialogruta utlöst vid förfrågan till applikationskoden, i stil med ”Vill du verkligen stänga?”, kan rapportera till dg:et huruvida det är okej att stänga dokumentet.

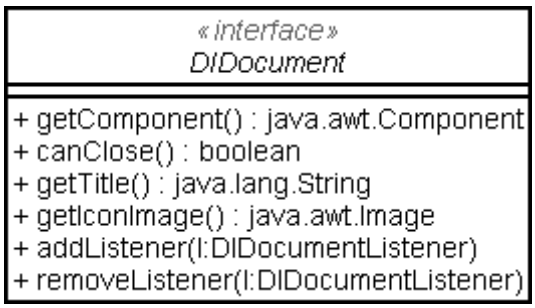
Egenskaperna summeras i följande tabell:

Egenskap	***Antal***	***Dokumentspecifik***	***Dynamisk***	***Datatyp Java***
Applikationsnamn	1	Nej	Nej	String
Applikationsikon	1	Nej	Nej	Image
Dokumenttitel	*	Ja	Ja	String
Dokumentikon	*	Ibland	Nej	Image
Dokumentkomponent	*	Ja	Nej	Component
Dokument får stängas?	*	Ja	Ja	boolean
Fönster	1..*	Nej	Nej	JFrame

Tabell 3: Applikationsegenskaper till dokumentgränssnittet; egenskaper applikationen har som dokumentgränssnittet behöver veta eller ha tillgång till.

Av tabellen, särskilt med ”dokumentspecifik”-kolumnen, kan utläsas att egenskaperna bör skiljas i olika klasser för att efterfölja god objektorienteringssed.

Jag kan börja med att samla alla dokumentspecifika egenskaper i en klass:



Figur 10

Operationerna *addListener* och *removeListener* är en del av ett Observer designmönster för att implementation av kodgränssnittet ska kunna meddela andra objekt när en egenskap i den egna instansen ändras, se nedan.

Sedan har jag egenskaper som är gemensamma för applikationen:

Figur 11

Antalet applikationsfönster och ett applikationsfönsters antalsrelation till dokument är beroende på dg och därmed platsar inte fönster som en egenskap

under varken dokument eller applikations-klassen. Applikationen måste dock bidraga med att skapa fönstret, så därför en skapa-fönster-operation i applikationsklassen.

Den dynamiska dokumenttiteln

Eftersom dokumenttiteln är dynamisk, det vill säga kan ändras under körning, måste dg:et få veta när den ändras så dess GUI kan uppdateras. Det faller sig naturligt att göra detta med klassiska Observer designmönster.

Så här den synopsis till Observer som [Grand, sid 347] har:

> *Allows objects to dynamically register dependencies between objects, so that an object will notify those objects that are dependent on it when its state changes.*

I fallet så ska alltså dg:et ”registrera sig” hos dokumentet för att sedan bli underrättad när/om dokumenttiteln ändras.

Ett ObserverInterface för ändåmålet kan i UML se ut som nedan. Observers kallas i Java för Listeners och jag följer den modellen här.

Figur 12

Vem har ansvaret för applikationsfönstret?

Vem har ansvaret för att hanteringen av applikationsfönster, dg:et eller applikationskoden? Detta kräver en noggrannare eftertanke. Jag listar ett applikationsfönsters egenskaper och operationer:

- Fönstrets titel bör nog bestämmas dg:et på grund av dess utformning kan variera mellan olika dg. Exempelvis har Acrobat Reader, ett typisk MDI-program, programnamnet först i titelraden, medan en SDI-applikation som OpenOffice oftast har dokumentnamnet först. Undantag från regeln finns, men det finns ytterligare ett skäl att lägga denna bit i dg:et; det kommer förmodligen (förhoppningsvis!) skrivas mindre antal dg-implementationer än applikationer som använder dessa dg, så lägga titelhanteringen i dg:et skulle därmed bidra till mest kodåteranvändning.
- Fönstrets ikon måste bestämmas av dg:et; vid SDI används en ”dokumentikon” medan vid MDI och TDI används vanligen applikationsikonen.
- Hanteringen av ett försök att stänga fönstret, i Windows till exempel genom att klicka på x-knappen, måste hanteras av dg:et eftersom operationen har olika betydelser i olika dg; i SDI är det ett försök att stänga dokumentet, i MDI och TDI ett försök att stänga hela applikationen.

- Ett applikationsfönster har noll eller ett dokument *aktivt*. Det aktiva dokumentet är det dokument som applikationsoperationerna (som till exempel ”Spara”) appliceras på. I MDI håller det aktiva MDI-fönstret det aktiva dokumentet. I SDI är aktivt dokument det dokument som är öppet (SDI kan inte har några *inaktiva* dokument eftersom det som maximalt finns ett dokument per applikationsfönster). I TDI är dokumentet för den markerade fliken det dokument som är aktivt. Av detta drar jag slutsatsen att dg:et bestämmer vilket som är aktivt dokument.
- Fönstrets menyer måste bestämmas av applikationen, dg:et kan inte veta vilka operationer som finns i särskilda applikationer.
- Verktygsfält som knapprader och statusrader är definitivt applikations-specifikt.
- Arbetsytan ser olika ut i olika dg, alltså en egenskap av dg-systemet.

Fönsteregenskap	***Applikationskodens ansvar***	***Dg:et avsvar***
Titel		X
Ikon		X
Fönsterstängning		X
Bestämning av aktivt dokument		X
Menyer	X	
Verktygsfält	X	
Arbetsyta		X

Tabell 4: Ansvarsfördelning hos dg programfönster

Applikationskoden och dg:et måste alltså dela ansvaret över att skapa och hantera fönstret, vilket är knepigt eftersom att jag inte vill att applikationskoden ska vara knutet till ett visst dg, eller tvärtom.

Eftersom jag planerar att implementera API:n i Java, låt mig anta att jag vill ärva JFrame för min fönsterklass. Om jag vill ha en klasstruktur med rakt arv (att att applikationskod och dg ligger i samma arvshierarki) finns två kombinationer:

1. Låta dg-koden ärva från JFrame och låta det tillsammans ärvas av applikationskoden.
2. Låta applikationskoden ärva från JFrame och låta *det* tillsammans ärvas av dg-koden.

Båda dessa varianter är dock omöjliga rakt upp och ner med ett vanligt programmeringsspråk. I första fallet skulle applikationskoden ärva från klassen med dg-koden men då skulle klassen att ärva från tvingas att vara konkret (ej abstrakt) på så sätt att dess dg-del är implementerad. Det innebär att om applikationskoden skulle vara tvingad att ärva från en viss dg-implementation,

och att låsa en applikation till en dg var inte det jag ville! För att gå vägen skulle man behöva bestämma vilken klass som ska ärvas vid körning, något som inte är möjligt med något kompilerat programmeringsspråk jag känner till.

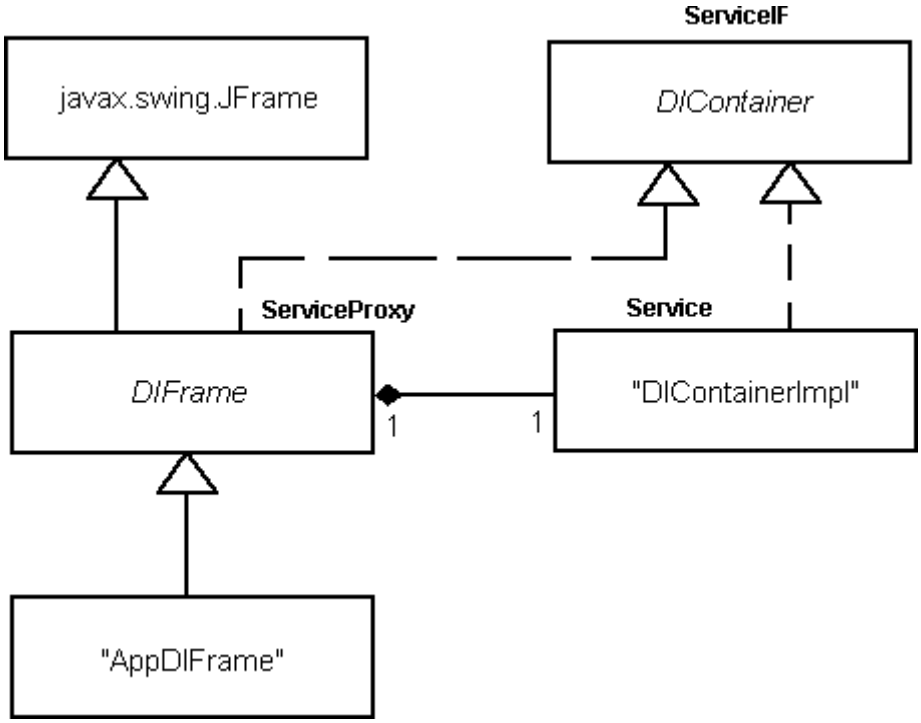
I det andra fallet kan man komma fram till en liknande slutsats, lämnas som tankeövning åt läsaren.

En lösning på problemet är en variant på designmönstret *Proxy*. Jag återgår till fallet ett (1) ovan. Dg-delen kan inte låtas implementeras i JFrame-trädet, men man kan implementera en proxy-implementation som pekar den faktiska dg-implementationen, en sådan pekning kan bindas dynamiskt vid körning.

Ur synopsisen för Proxy [Grand, sid 79]:

> *The Proxy pattern forces method calls to an object to occur indirectly through a proxy object that acts as a surrogate for the other object, delegating method calls to that object.*

Jag kallar dg-delen för DIContainer. Så här blir UML-diagrammet:



Figur 13: Delat ansvar av programfönster med Proxy designmönster. (Proxy-mönstrens namn i mindre, fet stil.)

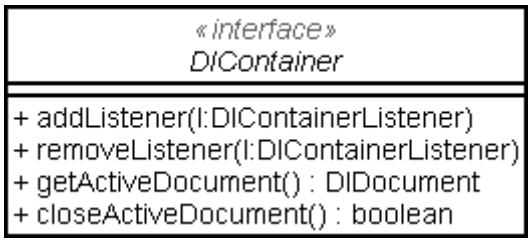
På grund av att hanteringen av ett försök till stängning av programfönstret hanteras av dg:et och dg:et avgör om fönstret faktiskt ska stängas eller ej, så kan applikationskoden inte lyssna på WindowListener.windowClosing (WindowEvent) för att utföra något precis något innan fönstrets (faktiskt) stängs. Därför behöver applikationskoden en funktion i dg:et för att ”lyssna efter” när fönstret håller på att stängas.

Man behöver också från applikationskodens sida lyssna efter när aktivt dokument ändras, för att till exempel aktivera eller avaktivera menykommandon.

Med dessa två saker blir Observer-gränssnittet för DIContainer följande:

Figur 14

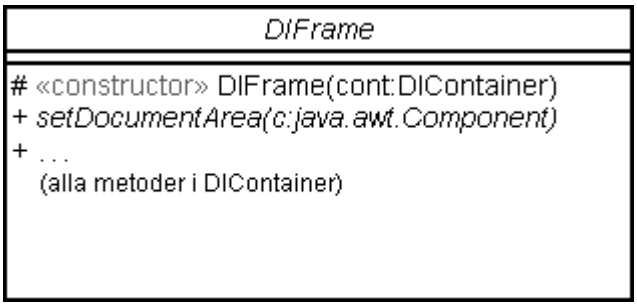
Och så DIContainer självt:



Figur 15

Metoden *getActiveDocument( )* kan härledas direkt från punktlistan ovan; dg:et bestämmer vilket dokument det är som är aktivt. Två metoder behövs för att applikationskodens observers ska kunna registrera och avregistrera sig. Till sist, *closeActiveDocument( )* är en behändighetsmetod[15], tänkt att vara ekvivalent med *DI.closeDocument(DIContainer.getActiveDocument( ))*.

Dg:et behöver påverka applikationsfönstret i avseendet att ändra titel, ändra ikon, hantering försök till stängning samt äga fönstrets arbetsyta, enligt punktlistan ovan. För de tre första (titel, ikon, stängning) finns befintliga metoder i Javas JFrame, alltså behövs inte några sådana i DIFrame som ärver från JFrame. För arbetsytan behövs dock något göras. Jag möjliggör för dg:et att sätta sin arbetsyta i fönstret genom en ny operation, låt mig kalla den *setDocumentArea(: Component)*.



Figur 16

Eftersom DIFrame realiserar gränssnittet DIContainer på grund av Proxy designmönster, måste DIFrame ha alla metoder som finns definierade i DIContainer[16]. Implementationerna av dessa metoder är raka (okomplicerade);

15 Översättning av engelskans *convenience method*.

16 Om man ska vara petig: Eftersom DIFrame är en abstrakt klass skulle alternativt DIFrames underklass(er) implementera fritt antal av metoderna, men det skulle bryta Proxy mönstret.

det är bara att skicka vidare begäran till instansen faktiska implementation av DIContainer. Referensen till den instansen skickas lämpligtvis till DIFrame-instansen när den skapas, genom konstruktorn (i enighet med UML-diagrammet ovan).

I praktiken: källkod

Det skulle varit ett väldans torrsim att bara skriva om hur en API kan skapas utan att gå vidare därifrån. Därför översatte jag naturligtvis API:n till Javakod och implementerade den för MDI, SDI samt TDI.

Källkoden är släppt under LGPL-licensen och finns att hämta på adressen: http://perholmberg.net/2005/kandidatarbete/

För att försäkra mig om att API-koden fungerade skrev jag ett exempelprogram likt Anteckningar som använder API:n. Källkoden för denna finns även den på adressen ovan och visar bra hur API:n används.

Slutsatser

Det finns inte en dg-modell som fungerar bäst i alla lägen. Det är många faktorer som spelar in. Det är i mångt och mycket situationen/sammanhanget och hur man väger de olika faktorerna som avgör vilken modell som är lämpligast för ett visst program.

SDI i vinnaren enligt min summering av analyserade faktorer. Modellen fungerar helt i okej i de flesta lägen, om ej alltid optimalt. SDI är även den enda modell som rekommenderas av både Microsoft och Apple.

MDI är modellen som får sämst poäng i min analys. Modellen lider av en komplex fönsterhantering och en icke data-centrerad design. MDI är dock inte helt utan fördelar. Det finns en stor skara vana användare som uppskattar grupperingen efter program – om för inte annat vanans skull.

TDI är så pass nytt att det inte tas upp i litteraturen som en allvarlig utmanare till MDI och TDI. Flikmodellen har dock under en kort tid vunnit en stor användarskaras gillande. Både enkätundersökningen och programtestet inom detta arbete stärker den bilden – inte mycket ont sas om TDI.

Vad som är bäst beror på vilken betydelse man valt av ”bäst”. McKay väljer utifrån om program är applikationer eller verktyg. Han menar att SDI alltid är en bra lösning, medan MDI endast är lämpligt för applikationer, en uppfattning som jag inte hittat något som talar emot.

Kort sagt: SDI är ett säkert kort, TDI är trendigt, MDI är ute.

Ett sätt att komma runt en modells svagheter är att kombinera den med en annan modell. Så har man gjort i flera välkända program.

Apple har i sina riktlinjer dömt ut MDI till förmån av data-centrerade SDI. Microsoft valt en mer ”det beror på programmet”-väg. I sina egna program har Microsoft gått mot mer SDI från MDI.

Går det att skapa en API som låter användaren välja? Svaret är: Ja! Jag designade och implementerade en API i språket Java och den fungerade.

Diskussion

Det är lite motsägande att TDI hyllas när MDI sågas. TDI är ändå en slags vidarutveckling av MDI.

De två största företagen på operativsystem, Microsoft respektive Apple, har en stor inverkan till att skapa en ”standard”. TDI har inte kommit i dessa jättars riktlinjer i större omfattning än. En flik-baserad IE7 banar för en TDI-värld för en större publik. Hur Apple ska tackla fliktrenden, som inte helt går överens med den data-centrerade modellen, återstår att se.

När jag började detta arbete trodde både jag och andra att jag skulle hitta mycket litteratur inom det direkta ämnet. Nu var det inte så; det var svårt och jag lyckades inte hitta så mycket och därmed blev inte den teoretiska grunden så stabil som önskvärt.

Jag har inte hittat några direkta motsägelser för att en dg-API skulle vara något direkt hinder eller betydande nackdel vid utveckling ett vanligt, fullgott program. Lite flexibilitet går dock lätt förlorad när man programmerar sin programlösning runt en färdig API.

API:n och implementationen inom ramen för detta arbete ska ses som grundläggande; den är fullt fungerande men lämnar vissa saker att önska, ta till exempel Fönster-meny för växling mellan dokument i MDI-gränssnittet, testpersonen i programtestet saknade den funktionen.

Mina två frågeställningar var väldigt olika. Den ena var inriktad på användbarhet medan den andra var väldigt teknisk. Men jag tycker de ändå har en stark koppling. Resultatet på den första frågeställningen, att det ofta inte finns *ett* bästa dg, berättigar andra frågan; om man kan låta användaren välja.

Bilaga: Enkätfrågor

Enkät

Detta är en enkät vars resultat kommer att användas i Per Holmbergs kandidatarbete i datavetenskap (BTH). Det skulle vara snällt om du kunde spendera några minuter på att svara på mina frågor. Mer utförliga svar är naturligtvis bättre. Du kan lämna tomt på frågor du inte förstår, men du får gärna försöka svara även om du inte förstår frågan helt och hållet.

"Beta av" sidan uppifrån och ner.

Först, vem är du?

Ditt namn:

Lämna tomt om du vill vara anonym.

Din e-postadress: (får lämnas tomt)

☐ Jag vill bli krediterad (nämnd med namn) om jag blir citerad.

Här kommer några frågor för att ta se vilken typ av datoranvändare du är.
Det du skriver under denna avdelning kommer inte bli citerat även om du markerat kryssrutan ovan.

Hur skulle du beskriva din datorvana? Hur länge har du till exempel haft dator?

Vilka operativsystem använder nu och vilka har du tidigare använt? Ange versionsnummer om du vet. (Exempel på operativsystem är Windows 98 och Mac OS X.)

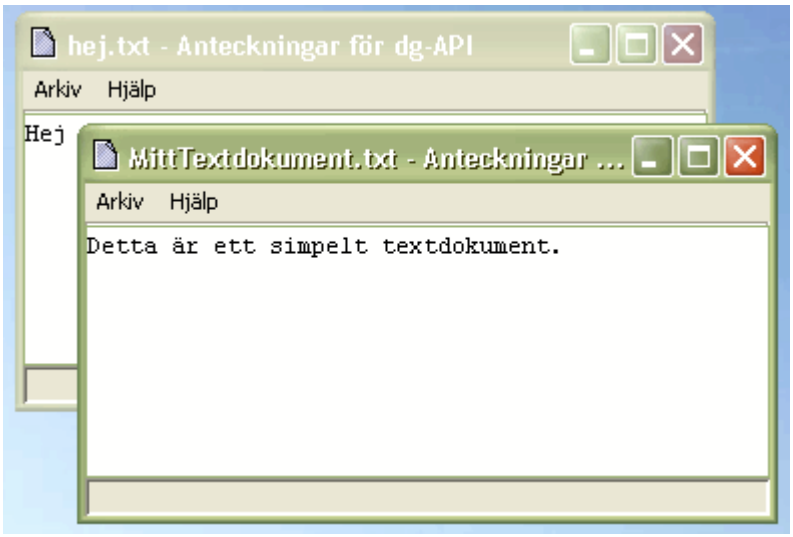
Vilka program brukar du använda? Ange versionsnummer om du vet dem, annars räcker det med namnet på programmet. (Exempel på program: Word, WordPad, Anteckningar, Acrobat Reader, Excel, OpenOffice, Internet Explorer, Firefox)

Här nedan förklaras några termer som används i frågorna längre ner på sidan.

Saken handlar om hur dokument visas på skärmen och hur användaren arbetar med dem i den grafiska fönstermiljön. Det finns lite olika varianter hur det ser ut och går till. Varje variant är en typ av dokumentgränssnitt. De tre vanligaste dokumentgränssnitten heter (eller snarare förkortas) MDI, SDI respektive TDI.

Ett **MDI**-program har ett enda programfönster. Inuti detta programfönster finns ett antal fönster, ett fönster för varje dokument. Alla dokument "delar" på samma uppsättning av menyer och verktygsfält. En enkel textredigerare som använder MDI kan se ut så här:

I ett **SDI**-program har varje dokument har ett eget programfönster, varje programfönster kan som maximum innehålla ett dokument. I Windows har varje dokument i ett SDI-system har en egen post i aktivitetsfältet (raden med startade program längst ner på skärmen). Samma program som ovan fast med SDI-gränssnitt ser ut så här:



I likhet med MDI har så använder **TDI** bara ett programfönster, men det finns en rad med flikar för att välja aktivt dokument. Samma program som tidigare, men med TDI-gränssnitt, visas här:

Och nu frågorna.

Vilka av dokumentgränssnittssmodellerna kände du igen?

☐MDI
☐SDI
☐TDI

I fall du kryssade minst två alternativ på frågan ovan: Vilken modell (typ av dokumentgränssnitt) föredrar du? Motivera ditt svar.

I fall du skrev på förra frågan: Tänk att du som användare skulle kunna välja

vilken typ av dokumentgränssnitt (MDI, SDI eller TDI) du vill använda i ett program. Vad tror du om den idén?

Övriga synpunkter som du tror kan vara relevanta:

Referenser

[AnyOSSplits]
http://javasoftware.perholmberg.net/anyossplits_sv

[Apple 2004a]
Apple Software Design Guidelines: Windows Considerations, http://developer.apple.com/documentation/MacOSX/Conceptual/AppleSWDesign/PortingTips/chapter_8_section_3.html 2005-03-23.

[Apple 2004b]
Apple Human Interface Guidelines: Windows, http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html, 2005-03-23.

[Dundas]
A multi document tabbed interface, http://www.codeproject.com/docview/tabbedmdi.asp 2005-04-07.

[Galitz]
Galitz, Wilbert O., The Essential Guide to User Interface Design: An Introduction to GUI Design, Second Edition, Wiley Computer Publishing, 2002. ISBN 0-471-08464-6.

[Crabb]
Crabb, Don, Windows SDK 3.0 Has the Tools That Developers Want, Infoworld, volume 12, issue 36, page 71-74, 1990.

[Grand]
Grand, Mark, Patterns in Java Volume 1, Wiley Computer Publishing.

[IEBlog]
IEBlog: IE7 Has Tabs, http://blogs.msdn.com/ie/archive/2005/05/16/417732.aspx 2005-05-20.

[Microsoft 2004]
Microsoft Research, Scalable Fabric: Flexible Task Management, 2004.

[MSDN]
MSDN: Fundamentals of Designing User Interaction, Microsoft Corporation 2004, http://msdn.microsoft.com/library/en-us/dnwue/html/part1.asp 2005-03-16.

[McKay]
McKay, Everett N., Developing User Interfaces for Microsoft Windows, Microsoft Press.

[Larman]
Larman, Craig, Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition,

Prentice Hall, 2002.

[Opera]
Opera 7 for Windows Changelogs: New in Opera 7.0 Beta 1 http://www.opera.com/windows/changelogs/700b1/index.dml 2005-03-12.

[Opera2001]
Opera Software Press Release: Opera 6.0 for Windows Beta 1 Released, http://www.opera.com/pressreleases/en/2001/11/20011113.dml 2005-03-12.

[Preece]
Preece, Jenny / Rogers, Yvonne / Sharp, Helen, Interaction design – beyond human-computer interaction, Wiley, 2002.

[Rosenbaum]
Rosembaum, Beverly, Office Inconsistencies, http://www.hal-pc.org/journal/aug00/Column/trumors/trumors.html 2005-04-07.

[Scheifler]
Robert W. Scheifler, MIT Laboratory for Computer Science and Jim Gettys, Digital Equipment Corporation and MIT Project Athena, The X Window System, ACM Transactions on Graphics, Vol. 5, No. 2, April 1986.

[Sherriff]
Sherriff, Lucy, IE7 details leak onto web, The Register, http://www.theregister.co.uk/2005/03/16/ie7_leak/ 2005-03-18.

[Shneiderman]
Shneiderman, Ben, Designing the User Interface – Strategies for Effective Human-Computer Interaction, Third Edition, Addison Wesley Publishing Company, 1997.

[Springmann]
Springmann, Michael, Using the Object–Orientation Paradigm and Design Patterns to Improve the Development of GUI-Based Applications, http://michaelspringmann.de/arbeiten/masterthesis.pdf 2005-05-12.

[Susning ADT]
http://susning.nu/Abstrakt_datatyp 2005-03-29.

[Wikipedia MDI]
Wikipedia: Multiple document interface, http://en.wikipedia.org/wiki/MDI 2005-04-07.