Menyhantering i VIS

Daniel Skoglund

1 februari 2008
Examensarbete i Teknisk datavetenskap, 20 poäng
Handledare på CS-UmU: Jerry Eriksson
Handledare på BAE Systems Hägglunds: Stefan Westman
Examinator: Per Lindström

UMEÅ UNIVERSITET
INSTUTITIONEN FÖR TEKNISK DATAVETENSKAP
901 87 UMEÅ

Sammanfattning

I denna rapport beskrivs utvecklingen av ett klassbibliotek för att hantera menyer och menyernas förhållande till varandra. De delar som skapas är färdiga att användas och ger en grund som förenklar arbetet med att bygga applikationer för det informationssystem som används av BAE Systems i militära fordon.

I rapporten presenteras företaget BAE Systems och sedan presenteras planeringen av arbetet och specifikationerna för det bibliotek som skapats. Utöver själva menyhanteraren har ett verktyg som tolkar en XML-fil och genererar kod för att ytterligare förenkla byggandet av en komplett applikation skapats. Detta verktyg presenteras i rapporten.

För att visa ett exempel på hur menyhanteraren samt XML-verktyget kan användas presenteras även arbetet med att ersätta den tillståndsmaskin som används i applikationerna idag.

Utöver detta ingår ett kapitel som handlar om XML och hur XML kan användas i applikationer för att tillåta användaren att specificera inställningar och överföra information på ett professionellt sätt. Här studeras XML-parsning och XML-validering mer djupgående och olika XML-parsrar jämförs och utvärderas.

Menu Management in VIS

Abstract

This report describes the work with creating a class library to handle menus and the menu structure. The different parts created are ready to be used and will act as a base when creating applications for the information systems used by BAE Systems in their combat vehicles.

The company BAE Systems is presentated as well as the planning of the work and the specifications for the different parts of the work. As a complement to the menu manager, a tool that parses an XML file and generates code that can be used together with the menu manager, are created and this tool will be described in the report as well.

To show how the menu manager and the code generation tool can be used, the state machine used to manage menus in the applications today are replaced in one of the test applications and this will be described as the third part of the work.

In addition to the different parts implemented a section about XML and how XML can be used by programmers to specify settings and transfer information in a professional way. XML parsing and validation are discussed and some different XML parsers are evaluated.

Innehållsförteckning

Lista över Figurer

Lista över tabeller

Förkortningar

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BAE	British Aerospace Systems
Boost	Boost C++ libraries
Bv206S	Bandvagn 206S
BvS10	Bandvagn 210
C-ITS	BAE Systems C-ITS AB
CV90	Combat Vehicle 90
CVS	Concurrent Version System
DIS	Diagnostic Information System
DOM	Document Object Model
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
GUI	Graphical User Interface
ISO	International Organization for Standardization
MFC	Microsoft Foundation Classes
MSXML	Microsoft XML Core Services
OPC	Se http://www.opcfoundation.org
RDF	Resource Description File, en "påbyggnad" till XML
SAX	Simple API for XML
SAX2	Simple API for XML 2
SEP	Splitterskyddad Enhetsplattform
SGML	Standard Generalized Markup Language
USB	Universal Serial Bus
UTF	Unicode Transformation Formats
W3, WC3	World Wide Web Consortium
VC6	Visual Studio 6
VC7.1	Visual Studio 7.1
VC8	Visual Studio 8
VCS	Vehicle Control System
VIS	Vehicle Information System
WXS	W3C XML Schema
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	XSL-Transformation, skapa xhtml från xml
ZIP	filtyp för filer som komprimerats

Kapitel 1

1 Introduktion

I detta kapitel presenteras företaget British Aerospace Systems (BAE), Hägglunds, och de produkter de tillverkar. Efter detta presenteras själva idén med examensarbetet och varför det behövs. I slutet på kapitlet presenteras denna rapports upplägg.

1.1 Bakgrund

1.1.1 BAE Systems

BAE Systems är ett internationellt företag som utvecklar och producerar produkter åt många länders försvar. Totalt har BAE Systems 86 000 anställda. Företaget är världens fjärde största inom detta område och har en omsättning på 28 miljarder dollar.

I Sverige är BAE Systems uppdelat i två delar, Bofors och Hägglunds. Det svenska aktiebolaget heter BAE Systems AB och ägs av BAE Systems. Antalet anställda i Sverige är 1739, varav 1220 jobbar på Hägglunds AB i Örnsköldsvik, 454 jobbar på BAE Systems Bofors AB och 65 jobbar på BAE Systems C-ITS AB (C-ITS) som är ett dotterbolag till BAE Systems Bofors AB. Den totala omsättningen för det svenska aktiebolaget är 460 miljoner dollar.

BAE Systems Hägglunds producerar militära fordon och system inom följande områden:

- Markgående stridsfordon, Combat Vehicle 90 (CV90)
- Terrängfordon, Bandvagn 206S (Bv206S)/Bandvagn 210 (BvS10)
- Mellanviktfordon på hjul eller band, Splitterskyddad Enhetsplattform (SEP)
- Torn

BAE Systems Bofors producerar vapensystem och ammunition. Några av de viktigaste produkterna är:

- Artilleri för långa avstånd
- Vapen för marinen
- Intelligent ammunition
- System för att hjälpa till vid ledning samt beslutsfattande

BAE Systems C-ITS utvecklar och producerar produkter för träning, utbildning och simulering och håller till i Stockholm.

1.2 Mål med examensarbetet

Detta examensarbete gjordes på BAE Systems Hägglunds i Örnsköldsvik. Utvecklingen av programvara på Hägglunds är uppdelad i följande tre delar:

- Vehicle Control system (VCS) – Kontrollsystem
- VIS Vehicle Information System (VIS) – Informationssystem
- DIS Diagnostic Information System (DIS)– Verktyg för diagnostik

Målet med detta examensarbete är att skapa en menyhanterare som används vid byggandet av det informationssystem som används i fordonen som produceras. Eftersom varje kund har sina egna krav på hur menyer ska fungera och användas måste nya menyer skapas för varje kundprojekt. Detta gör att samma jobb måste göras flera gånger och kan alltså förbättras genom att en menyhanterare utvecklas, som kan användas som grund för att skapa dessa menyer.

Som grund för de menyer som skapas nu används ett bibliotek som heter Boost (5) och som är tillgängligt som öppen källkod. Den del av Boost som används är en tillståndsmaskin som är avancerad och har funktioner för mycket som inte behövs. Detta gör att det är ganska krångligt att skapa menyer och implementera deras logik. Målet är att den menyhanterare som skapas i examensarbetet ska agera som en tillståndsmaskin och vara lätt att använda för att skapa nya menyer. Om detta visar sig vara lyckat ska Boost tas bort från VIS och ersättas av den menyhanterare som skapas.

För att underlätta ytterligare ska även en XML-parser byggas. Dess uppgift är att läsa en XML-fil som innehåller information om hur menyer hänger ihop och hur navigering mellan olika menysidor ska vara möjlig, och generera en grund för varje menysida samt en struktur över hur menyerna hänger ihop. Detta använder den menyhanterare som skapats och gör alltså mycket av det jobb som är återkommande vid varje kundprojekt.

1.3 Rapportens upplägg

Kapitel 1 ger en kort introduktion till företaget Hägglunds samt målet med examensarbetet.

Kapitel 2 är en djupare beskrivning av uppgiften samt en specifikation över vad som ska finnas med.

Kapitel 3 handlar om Extensible Markup Language (XML) och utvärderar några XML-parsrar mot varandra. Detta kapitel innehåller ingenting som är specifikt för menyhanteraren och kan hoppas över om inte intresse finns.

Kapitel 4 beskriver de olika lösningar som implementerats och är själva resultatet av arbetet.

Kapitel 5 består av de slutsatser som tagits efter att arbetet gjorts och ger förslag på utökning av menyhanteraren och XML-parsern.

Kapitel 6 är med för att tacka alla inblandade.

Kapitel 7 består av en lista med alla källor som används.

Kapitel 8 består av bifogade manualer till de olika lösningarna.

Kapitel 2

2 Metoder och arbetssätt

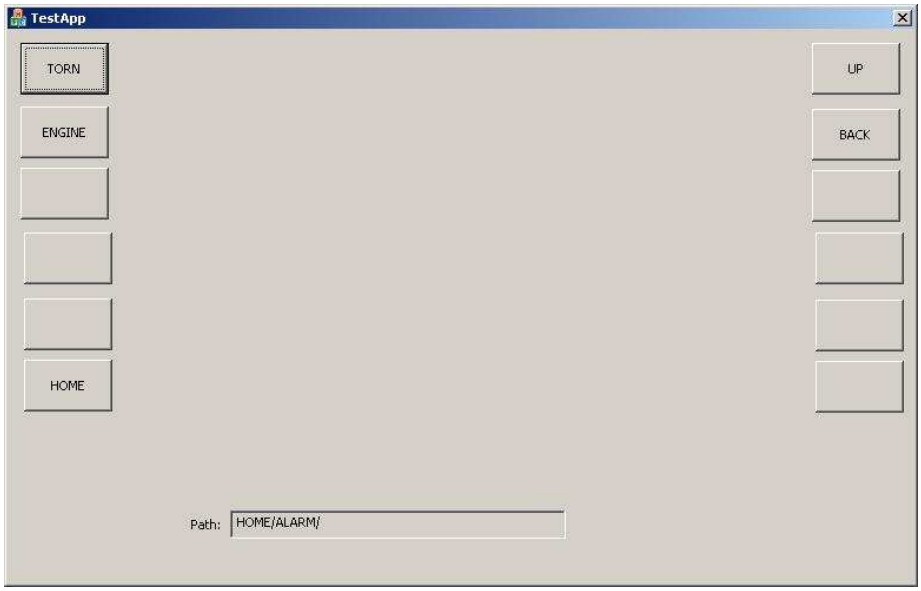
Detta kapitel tar upp de tillvägagångssätt som använts under arbetets gång. Först redovisas vad som gjordes första dagen på Hägglunds och sedan planeringsfasen följt av specifikationerna för examensarbetet. Specifikationerna är uppdelade i tre delar, menyhanteraren, XML-parsern samt införande i VIS Bas, det underliggande systemet för VIS, för testning. Efter detta tas upplägget för själva arbetet upp.

2.1 Introduktion

Det första som gjordes när det var dags att börja examensarbetet var att träffa handledaren på företaget och få en arbetsplats och en introduktion till hur allt fungerade rent praktiskt.

2.2 Planering

Den första fasen i själva arbetet var att reda ut exakt vad som skulle göras och hur menyhanteraren ska användas när den är klar. Det mesta diskuterades muntligt och det tog inte många dagar innan implementationen kunde påbörjas. Det första som skapades var ett Graphical User Interface (GUI) för att kunna testa menyhantering. Detta gjordes som en mycket enkel testapplikation med hjälp av Microsoft Foundation Classes (MFC). En menysida kunde se ut som nedan, där "path" används för att visa vilken menysida som är aktiv och knapparna används för att navigera med hjälp av den underliggande menyhanteraren.



Figur 1, Grafisk testapplikation för menyhanterare.

När detta var gjort kunde en mer detaljerad planering göras och den såg i grova drag ut såhär:

Aktivitet	Varaktighet	Start	Slut
Introduktion	1 dag	2007-09-03	2007-09-03
Samla Bakgrundsinformation	19 dagar	2007-09-04	2007-09-28
Menyhanterare Grund	25 dagar	2007-10-01	2007-11-02
XML-hantering	15 dagar	2007-11-05	2007-11-23
Införande i VIS bas	15 dagar	2007-11-26	2007-12-14
Rapport	20 dagar	2007-12-17	2008-01-11
Redovisning	5 dagar	2008-01-14	2008-01-18

Tabell 1, Huvudaktiviteter från den detaljerade planeringen

Under arbetets gång ändrades planeringen och implementeringen av de tre delarna skedde parallellt.

2.3 Specifikationer

Huvuddelarna av vad examensarbetet gick ut på har redan tagits upp men i detta avsnitt specificeras de olika delarna mer detaljerat. All kod implementeras i språket C++ och operativsystemet som används för att köra VIS-applikationer är Microsoft Windows XP Embedded. Utvecklingsmiljön som används på företaget är Microsoft Visual Studio .NET 2003 och kommunikation med Concurrent Versions System (CVS) sker med Borland StarTeam.

2.3.1 Menyhanterare

Detta är grunden för hela examensarbetet. Menyhanteraren ska gå att använda för att lagra strukturen på ett menyträd och innehålla funktioner för att göra följande:

- Gå till en viss menysida med ett unikt namn
- Gå upp en nivå i trädstrukturen
- Gå tillbaka till föregående menysida
- Gå till den menysida som är rotmeny (överst i trädstrukturen)
- Hantera knapptryckningar och agera efter dessa
- Hantera händelser som genereras av andra klasser i systemet (t.ex. en händelse från en USB-enhet)
- Verifiera att den struktur som finns lagrad är korrekt
- Definiera en grund för menysidor
- Returnera en textrepresentation av aktuell menysidas position i trädstrukturen
- Hantera knapptryckningar och se till att metoder i den aktuella menyn körs utifrån detta

- Hantera grupper som innehåller ett antal menysidor och som kommer ihåg vilken sida som var aktiv när gruppen lämnades och öppnar denna sida då gruppen besöks senare

Själva strukturen ska lagras som en trädstruktur där varje menysida representeras av en nod. Varje barn har exakt en förälder men en förälder kan ha ett valfritt antal barn. Om en nod inte har några barn betyder det således att det är en lövnod och de enda möjligheterna att navigera vidare från en sådan nod är att gå tillbaka till föräldern eller genom att använda en genväg för att gå till en valfri menysida. Denna struktur och de metoder som ska användas för att navigera i menysystemet från VIS-applikationen ska definieras i en klass som är tillgänglig utåt sett från menyhanteraren.

För att definiera en grund för menysidorna ska en klass som kan ärvas skapas. Denna klass ska ha metoder som körs när en menysida öppnas och när en menysida lämnas samt metoder som ska köras när en menysida tar emot en händelse från andra delar av systemet. Det ska även finnas en metod som hanterar knapptryckningar och tar in knappens id-nummer som parameter. Dessa metoder ska deklareras som virtuella i basklassen och detta tvingar alla klasser som representerar menysidor att implementera dessa metoder.

2.3.2 XML-parser

I den första specifikationen fanns det specificerat att inläsning av en XML-fil ska kunna ske och att ett menyträd ska genereras utifrån detta. Denna beskrivning var inte särskilt detaljerad och det fanns möjlighet att diskutera vad som skulle kunna vara bra och vad som skulle vara användbart. Efter några olika funderingar togs beslutet att XML-parsern ska vara ett externt verktyg som används för att göra mycket av det arbete som användare av menyhanteraren måste göra för att använda den vid skapandet av menyer och menyträd. Verktyget ska köras som en konsolapplikation och innehålla en funktion som validerar XML-filen mot ett fördefinierat XSD-schema för att kontrollera att allt som måste finnas med finns och att ingenting som inte får finnas med finns i XML-filen. Vid eventuella valideringsfel ska dessa listas så att det går lätt att rätta till felen. Verktyget ska generera kodfiler av typen cpp med tillhörande h-fil och varje menysida som finns definierad i XML-filen får en egen kodfil. Självklart är de genererade filerna inte fullständiga och användaren måste implementera de delar som inte går att specificera i XML-filen för hand. Förutom alla kodfiler som hör till menysidorna ska en till fil genereras. Denna ska innehålla kod för att skapa objekt av alla andra klasserna samt bygga upp menyträdets struktur. Vid användandet av de genererade filerna ska det alltså räcka med att skapa ett objekt av denna fil för att kunna använda menyhanteraren.

2.3.3 Införande i VIS för testning

I VIS används Boost idag som tillståndsmaskin och grund till menyhanteringen. Detta gör att det blir ganska krångligt att skapa nya menyer för varje kund och allt måste göras om från start. Den sista delen

av examensarbetet går ut på att ersätta Boost och testa den skapade menyhanteraren i VIS.

2.4 Upplägg

Eftersom själva menyhanteraren behövs för att kunna börja med de andra två delarna började arbetet med menyhanteraren. Efter ett par veckor då en användbar grund som fungerade i enkla fall fanns påbörjades även arbetet med XML-parsern. Detta för att eventuella problem skulle kunna upptäckas tidigt och inte när menyhanteraren var färdigimplementerad. De två faserna fortsatte parallellt och efter ytterligare ett par veckor påbörjades även integreringen i VIS. Denna arbetsgång visade sig vara lyckad eftersom många oklarheter förtydligades när användningen av själva menyhanteraren, precis som den är tänkt att användas, påbörjades.

Kapitel 3

3 XML

Den första delen av detta kapitel förklarar vad XML och XSD är och hur de används. Syntax och användningsområden tas upp. Efter det presenteras några XML-parsrar som finns på marknaden idag och hur de används.
En XML-parser har två uppgifter. Den första är att läsa en XML-fil och på något vis presentera informationen till användaren eller applikationen. Den andra är att validera XML-filen eller koden och se till att den följer en standard eller en mall.

3.1 XML (7)

Förkortningen XML betyder på engelska ”eXtensible Markup Language”. XML är alltså ett språk som används för att märka upp saker. Den tionde februari 1998 blev XML en W3C-rekommendation och den fjärde februari 2004 uppdaterades denna specifikation till XML 1.1 som är gällande idag. Rekommendationen beskriver syntax för XML samt vad som krävs av de verktyg som tolkar XML. För att kunna definiera vilka element och attribut som får finnas med i en XML-fil måste en mall användas. Denna kan vara en del av själva XML-filen eller en extern fil som länkas in i XML-koden. Den huvudsakliga användningen av XML är när olika informationssystem behöver skicka data mellan varandra. XML består av helt vanlig text som kan läsas av oss människor och är lätt att tolka för olika typer av datasystem.

3.1.1 Syntax

Den syntax som används i XML ser ut så här:

```
<name attribute="value">content</name>
```

Genom att kombinera flera element och bygga upp en hierarki kan mer avancerad information representeras. Ett exempel på detta är:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<boksamling>
  <bok språk="engelska">
    <titel>XSLT Cookbook</titel>
    <författare>Sal Mangano</författare>
  </bok>
  <bok språk="svenska">
    <titel>Skriv med XML</titel>
    <författare>Åsa Blom</författare>
  </bok>
```

```
</boksamling>
```

Deklaration

För att tala om vilken version av XML som används kan en rad som ser ut som den första raden användas. Den kan även innehålla information om externa filer och vilken kodning på tecken som används. Denna rad är frivillig.

Element

Rad nummer fyra i exemplet ovan anger att ett element som heter titel finns och att innehållet i det elementet är ”XSLT Cookbook”. Element är själva grunden för XML-dokument och inleds med namnet inom vinkelparanteser och avslutas på samma sätt fast namnet inleds av ett snedsträck. Ett element kan innehålla andra element. Ett exempel på detta är elementet boksamling ovan, som innehåller två element som heter bok. Förutom andra element kan det innehålla text.

Attribut

Även attribut kan ingå i ett element. Ett attribut läggs till i starttaggen för ett element och har ett namn samt ett värde. I exemplet ovan har elementet ”bok” ett attribut som har namnet ”språk” och som är satt till ”svenska” eller ”engelska”. Ett element kan inte ha flera attribut med samma namn och värdet på ett element måste anges inom enkla eller dubbla citationstecken.

Rotelement

Varje dokument måste ha ett och endast ett rotelement. Alla andra element måste finnas inuti detta element. I exemplet ovan är elementet med namnet ”boksamling” rotelement.

Tomma element

Om ett element inte innehåller något förutom attribut kan elementet stängas i starttaggen. Detta skulle innebära att

```
<bok></bok>
```

skrivs som

```
<bok/> eller <bok />.
```

Nästlade element

Som det nämnts ovan får varje dokument endast ha ett rotelement och alla andra element måste finnas inuti detta. Detta gör att element måste kunna vara nästlade. I exemplet ovan innehåller elementet ”bok” de två elementen ”titel” och ”författare”. Element får inte vara överlappande. Följande är ett exempel på kod som inte är tillåten i ett dokument eftersom ”h1” och ”em” överlappar varandra:

```
<h1>Här är en <em>Rubrik</h1> <p>Med</em>
en paragraf.</p>
```

Entiteter

Entiteter används i XML för att representera en bit data med ett annat namn. Det finns fem fördefinierade entiteter och de är:

- & - ampersand, &
- < - mindre än, <
- > - större än, >
- ' - apostrof, '
- " - citationstecken, ”

Fler entitetsreferenser kan deklareras och detta görs då i mallen för dokumentet.

3.1.2 Användningsområden

Som det redan nämnts används XML när olika datasystem vill skicka data mellan varandra. De olika systemen kanske körs på olika plattformer och är implementerade i olika språk. Då behövs ett gemensamt format som är lätt att tolka. Genom att strukturera data på ett visst sätt och genom att definiera en mall för hur denna struktur ska se ut kan de olika systemen läsa in data på sitt eget sätt och behöver inte bry sig om hur de andra systemen läser eller skriver den data som ska utbytas.
Några exempel på strukturerad data som kan utbytas i form av XML är kalkylblad, adresslistor, konfigureringsparametrar, finansiella transaktioner och tekniska ritningar.

3.2 XSD (10)

En mall för ett XML-dokument kan skapas i något av formaten Document Type Definition (DTD) eller XSD. I denna rapport tas endast XSD upp eftersom det är det format som kom senast och ses som ersättaren till DTD.

Förkortningen XSD kommer från det engelska uttrycket ”XML Schema Definition” och blev en del av de rekommendationer som World Wide Web Consortium (W3C) ger i Maj 2001. Det har funnits förslag om att ändra filtypens namn från XSD till WXS som då skulle betyda “W3C XML Schema”, men förslaget har avvisats av W3C.
Med hjälp av ett XSD-schema kan en XML-fil valideras och de delar som kan kontrolleras är:

- Namn på element och attribut
- Strukturen på dokumentet och relationen mellan element
- Datatyperna som används

3.2.1 Datatyper

Det finna 19 datatyper som kan användas i ett XSD-schema. Fler datatyper har föreslagits, för att kunna lagra binära data, men ingen av de datatyperna ingår i standarden idag. Ett exempel på binär data som skulle vara bra att kunna lagra är en bild. De datatyper som stöds är:

- string
- boolean
- decimal
- float
- double
- duration
- dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth
- hexBinary
- base64Binary
- anyURI
- QName
- NOTATION

3.2.2 Syntax

Ett XSD-schema har samma syntax som ett XML-dokument och måste även vara skrivet med korrekt XML. Följande är ett exempel på ett korrekt XSD-schema och ett tillhörande XML-dokument.

XSD-schema

Detta XSD-schema beskriver hur ett XML-dokument som innehåller information om ett land ska se ut. Den information som är tillåten att lagra är namnet på landet samt antalet invånare.

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
<xs:element name="country" type="Country"/>
```

```
    <xs:complexType name="Country">
        <xs:sequence>
            <xs:element name="name"
                  type="xs:string"/>
            <xs:element name="population"
                  type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

XML-dokument

Följande XML-dokument validerar mot XSD-schemat. Namnet på filen som schemat är i antas vara "country.xsd".

```
<country
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
xsi:noNamespaceSchemaLocation="country.xsd"
>
    <name>France</name>
    <population>59.7</population>
</country>
```

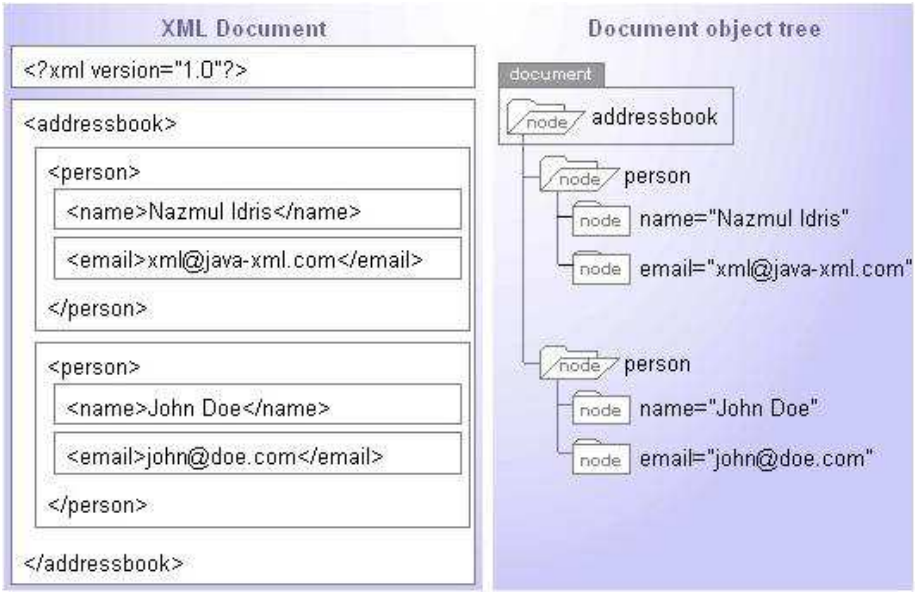
3.3 SAX eller DOM (4)

Både Simple API for XML (SAX) och Document Object Model (DOM) skapades för att ge programmerare ett sätt att komma åt data som finns lagrat i XML-dokument. Den största fördelen med att använda någon av dessa två är att programmeraren slipper skriva en egen parser i just det språk som används vid tillfället. De parsrar som finns idag klarar av att använda någon av dessa två och ofta båda. Detta gör att programmerare bara behöver se till att använda XML standarden för att sedan kunna använda den parser som de tycker verkar bäst. Application Programming Interface (API) för SAX och DOM finns till många språk och några av dem är Java, C++, Perl och Python.

Vad är DOM?

Data som lagras i XML-dokument kan representeras genom att skiljas åt med tabbar, vara listor eller vara vanliga dokument. DOM skapar alltid ett träd utifrån hur datat lagras i XML-filerna. Det spelar alltså ingen roll om det är listor eller vanliga dokument. DOM lagrar ändå allt i en trädstruktur. För att komma åt data måste programmeraren gå igenom detta träd och plocka ut det som behövs. Detta fungerar dock mycket bra eftersom XML är hierarkiskt från början. DOM ser även till att ordningen på datat bevaras från XML-filen och detta är nödvändigt om till exempel

ett dokument parsas. Bilden nedan visar ett exempel på hur trädet kan se ut efter att ett dokument parsats.



Vad är SAX?

SAX läser det aktuella dokumentet och skickar vidare informationen från dokumentet genom att tala om vilka start- och sluttaggar som finns och sedan får programmeraren själv bygga upp en modell för att spara eller använda informationen. Skillnaden mellan SAX och DOM är alltså att DOM går ett steg längre och bygger upp ett träd som innehåller information från dokumentet. Detta gör att DOM blir långsammare men SAX kräver mer arbete från programmeraren.

De olika händelser som skickas från SAX är för start- och sluttaggar samt för sektioner som #PCDATA och CDATA. De metoder som programmeraren skapar för att ta hand om dessa händelser måste alltså ta vara på informationen och själv bygga upp en modell eller använda dem direkt. Ordningen på dessa händelser har alltså stor betydelse.

Ska DOM eller SAX användas i examensarbetet?

Eftersom användare av de projekt som skapas kommer att skriva sina egna XML-filer för att bygga upp menyer och tänka sig en trädstruktur kommer troligtvis DOM att användas. Detta gör att den struktur som DOM skapar kan tolkas och mer eller mindre användas direkt i den klass som lagrar strukturen för menyerna.

3.4 Utvärdering av olika XML-parsrar

I detta avsnitt utvärderas några olika XML-parsrar med avseende på hastighet, storlek och funktionalitet. Först presenteras de olika parsrarna och hur de kan användas.

3.4.1 Xerces (2)

Xerces finns i olika versioner för flera olika språk. Den version som diskuteras här är ”Xerces-C++”. Det är en XML-parser som även har inbyggda funktioner för att validera de data som parsas. Xerces kan både läsa och skriva XML-filer och de olika APIer som stöds är DOM, SAX och SAX2.

Xerces är ett projekt under Apache och all källkod är tillgänglig. Parsern erbjuder hög prestanda, modularitet samt skalbarhet.

Översikt

Det första som måste göras för att använda Xerces är att initiera parsern. Detta görs likadant oberoende av vilken API som ska användas och den kod som används för att göra detta ser ut på följande vis:

```
#include <xercesc/util/PlatformUtils.hpp>
// Other include files, declarations, and
//  non-Xerces-C++
//  initializations.
XERCES_CPP_NAMESPACE_USE

int main(int argc, char* argv[])
{
  try {
    XMLPlatformUtils::Initialize();
  }
  catch (const XMLException& toCatch) {
    // Do your failure processing here
    return 1;
  }

  // Do your actual work with Xerces-C++
  // here.

  XMLPlatformUtils::Terminate();

  // Other terminations and cleanup.
  return 0;
}
```

Koden visar även hur parsern termineras när arbetet är avslutat. Funktionerna Initialize() och Terminate() måste köras minst en gång i varje process som vill använda parsern, men kan köras flera gånger. Varje körning av Initialize() måste dock matchas med en körning av Terminate().

Bygga/Kompilera Xerces

Det som tas upp här kommer handla om kompilering i Microsoft Visual C++ eftersom detta är aktuellt för examensarbetet. För att använda Xerces kan den färdigkompilerade distributionen laddas hem direkt och det som måste göras då är att packa upp den zip-fil som laddats hem. Detta kan göras med WinZip eller något annat UnZip-program. Detta skapar en katalog med namnet ” xerces-c_2_8_0-{arch}-windows-{compiler}”, där {arch} står för CPU-arkitekturen och {compiler} står för valet av C++-kompilator. När detta är gjort måste katalogen ” xerces-c_2_8_0-{arch}-windows-{compiler}\bin” läggas till som PATH-variabel i systemets variabler.

Skulle inte den färdigkompilerade versionen passa kan källkoden laddas hem men då måste den kompileras innan den kan användas. Färdiga projekt som går att öppna i Visual Studio finns på dessa platser:

```
(VC6)   Projects\Win32\VC6\xerces-
all\xerces-all.dsw
(VC7.1) Projects\Win32\VC7.1\xerces-
all\xerces-all.sln
(VC8)   Projects\Win32\VC8\xerces-
all\xerces-all.sln
```

När detta projekt är öppet ska XercesLib byggas.

Om det passar bättre kan även Xerces-C++ läggas till i ett redan skapat projekt och detta görs genom att öppna en av följande filer:

```
(VC6)    Projects\Win32\VC6\xerces-
         all\XercesLib\XercesLib.dsp
(VC7.1)  Projects\Win32\VC7.1\xerces-
         all\XercesLib\XercesLib.vcproj
(VC8)    Projects\Win32\VC8\xerces-
         all\XercesLib\XercesLib.vcproj
```

Det sista som måste göras är att länka applikationen med xerces-c_2.lib.

Jobba med Xerces

Här tas användning av Xerces och användning av DOM upp.
Som det redan nämnts måste Xerces initieras innan den kan användas. När detta är gjort, enligt ovan, kan ett objekt av klassen XercesDOMParser skapas. Efter detta används olika funktioner för att konfigurera parsern. Exempel på saker som kan konfigureras är om parsern ska validera mot ett schema och att sätta upp en felhanterare.
När detta är gjort parsas hela XML-dokumentet genom att köra metoden parse och skicka in sökvägen till XML-dokumentet som argument. Allt detta kan göras på följande sätt i C++.

```
XercesDOMParser* parser = new
XercesDOMParser();

Parser->setValidationScheme(
        XercesDOMParser::Val_Always);
parser->setDoNamespaces(true);

ErrorHandler* errHandler =
        (ErrorHandler*)new HandlerBase();
parser->setErrorHandler(errHandler);

char* xmlFile = "x1.xml";

try {
    parser->parse(xmlFile);
}
```

När detta är gjort finns all information om innehållet i XML-dokumentet lagrad och kan hämtas genom att köra följande metod.

```
DOMNode* doc = parser->getDocument();
```

För att komma år informationen används sedan metoder för att navigera i trädstrukturen som finns lagrad i doc. Ett exempel på detta är.

```
DOMNode* childNode = doc->getFirstChild();
```

För detaljerad information om vilka metoder som finns tillgängliga se Xerces API´n.

3.4.2 Expat (1)

Expat är ett bibliotek för parsning av XML-dokument. Det är skrivet i C och används bland annat som grund för Mozilla-projektet ”Perls XML::Parser” och andra XML-parsrar med öppen källkod. Expat är mycket snabb och har hög pålitlighet.

Skaparen av Expat är James Clark som även gjort andra projekt som groff (en nroff ”look-alike”), Jade (en implementation av ISO DSSSL stylesheets för SGML), XP (en Java XML parser) och XT (en Java XSL motor). James var även aktiv inom XML Working Group på W3 och hjälpte till att skapa specifikationen för XML.

Översikt

Expat fungerar så att funktioner registreras som tar hand om delar av dokumentet med en viss tagg. Sedan matas delar av dokumentet till parsern och de registrerade funktionerna körs allt eftersom. Detta gör det möjligt att börja parsa delar av dokumentet innan hela dokumentet är inläst. En annan fördel med detta är att stora dokument kan parsas även om inte hela dokumentet går att lagra i minnet på en gång.

Att använda Expat kan verka krångligt och avancerat eftersom det finns många funktioner och inställningar. Men för att använda Expat till de vanligaste uppgifterna som en XML-parser har behöver bara fyra av dessa funktioner användas och de presenteras här:

`XML_ParserCreate`

Skapar ett nytt objekt av Parsern.

`XML_SetElementHandler`

Sätter vilka funktioner som ska ta hand om start- och sluttaggar.

`XML_SetCharacterDataHandler`

Sätter vilken funktion som ska ta hand om text.

`XML_Parse`

Används för att skicka en buffer med en del av dokumentet till parsern.

Bygga/Kompilera Expat

Expat är inte packat som ett bibliotek. Istället finns fyra objektfiler som måste länkas till det projekt där Expat ska användas. Olika makron finns för att anpassa byggandet och hur Expat ska bete sig efter kompilering. Dessa är:

`XML_UNICODE`

Använd UTF-16 internt istället för `UTF-8` och skicka även strängar av typen `UTF-16` som returvärden. Detta ändrar definitionen av `XML_Char`, som annars är definierad som char. Om detta makro aktiveras men inte `XML_UNICODE_WCHAR_T`, kommer strängarna att lagras som `unsigned short`.

`XML_UNICODE_WCHAR_T`

Använd `UTF-16` internt enligt deklarationen i `<stddef.h>` och returnera strängar av typen `UTF-16` på detta sätt. Om detta makro aktiveras kommer även `XML_UNICODE` att aktiveras.

`XML_DTD`

Används för att inkludera kod för att parsa externa DTD.

`XML_NS`

Aktiverar lexikalisk kontroll av namespaces.

`XML_BYTE_ORDER`

Detta ska sättas till 12 för användning av ”little-endian” och till 21 för användning av ”big-endian”.

`XML_MIN_SIZE`

Detta ser till att skapa en parser som är mindre än vanligt men gör även att den blir långsammare i de flesta fallen.

Jobba med Expat

Som det redan nämnts fungerar parsern så att en bit av ett dokument skickas till den och den parsar detta. Det är helt upp till användaren hur stora delar av dokumentet som ska parsas åt gången. Parsern kör hela tiden funktioner som definierats för start- respektive sluttaggar, bland annat. På grund av detta kan det vara bra att lagra den information som fås från parsern i en stack. Detta gör att när en del text kommer från parsern och den tagg som texten tillhör ska identifieras kan värden som ligger på stacken kontrolleras. Detta är möjligt eftersom XML-dokument vanligtvis är uppbyggda med en hierarki och attribut har en föräldra-/barn-relation. Det kan även vara bra att spara information för hur djup stacken är. Detta gör det enklare att hoppa över vissa delar av ett dokument eftersom det helt enkelt kan kollas vilket djup i stacken som är aktuellt vid starttaggen och hoppa över allt tills sluttaggen är på samma djup.

För att kunna skicka värden mellan parsern och de olika funktionerna som kallas från parsern bör en datastruktur som håller dessa värden definieras. Parsern skickar vanligtvis en pekare till denna datastruktur som det första argumentet till funktionerna som registrerats för att ta hand om olika taggar. Detta gör att användandet av globala variabler kan undvikas.

Olika typer av characters

XML-dokument kan ha många olika typer av kodning för characters och Expat har inbyggt stöd för fyra av dessa. Dessa är:

- UTF-8
- UTF-16
- ISO-8859-1
- US-ASCII

För att avgöra vilken kodning som används kan parsern kolla på raden som ser ut på detta vis:

```
<xml? Version=”1.0” encoding=”ISO-8859-1”?>
```

Det går även att specificera vilken typ av kodning som används vid skapandet av parsern. Detta kan vara användbart om kodningen bestäms i en extern fil. Skulle den aktuella kodningen vara av en annan typ än de fyra angivna kallas en funktion som talar om detta och användaren kan välja att hantera kodningen på eget vis och måste då ha definierat hur detta ska hanteras i en struktur som kallas `XML_Encoding`.

3.4.3 MSXML (3)

En annan parser som kommer att utvärderas nedan är Microsoft XML Core Services (MSXML). Användning med mera kommer inte presenteras här men däremot lite kort information om vad MSXML är.
MSXML tillhandahåller tjänster för att bygga applikationer som använder XML 1.0 standarden. MSXML fungerar i Windows 2000, Windows Server 2003 samt Windows XP och finns att ladda ner gratis hos Microsoft.
De APIer som stöds är DOM och SAX2 för XML 1.0 samt XML Schema 1.0, XPath 1.0 och XSLT 1.0. Den senaste versionen, 6.0, har förbättrats markant och har hög tillförlitlighet och säkerhet.

3.5 Jämförelse av olika parsers

Vid jämförande av olika XML-parsrar finns det många saker att kolla på. De som kommer tas upp här är genomströmning, storlek på de filer som bygger upp parsern och användning av minne. Det bästa hade enligt min åsikt varit att utföra egna tester men eftersom inte tid för detta finns kommer utvalda tester som gjorts av andra att användas. De tre parsrarna är inte med i varje test men i slutet diskuteras och jämförs de mot varandra utifrån testernas resultat.

3.5.1 Expat – XercesC (8)

Det första testet är gjort av ”xmlbench” och omfattar Xerces samt Expat.

Inställningar

Hårdvaran som användes vid detta test var:

- Intel Pentium IV 2200 (22x100) MHZ (Nothwood) CPU
- Intel D850MV motherboard (i850E + ICH2)
- 4 x Samsung MR16R 256 MB PC800 Rambus DRAM
- IBM GXP120 60GB hard drive

Mjukvaran som användes var:

- DarkLin III (Mandrake 9.2 Based) Single User Mode
- Kernel 2.4.22
- GNU C Library 2.3.2
- GNU C Compiler 3.3.1
- Intel C++ Compiler 8.0
- J2SE 1.4.2 + JAXP 1.2

Testet omfattade många olika parsers och för att ge läsaren möjlighet att själv jämför de valda parsrarna mot de övriga kommer resultaten för alla parsrar att presenteras, trots att endast de valda diskuteras här.

De olika parsrar/bibliotek som var med i testet var:

- **Expat 1.95.6**
- Sablotron 1.0.1
- Arabica Jan04
- CenterPoint XML 2.1.7
- Gnome XML Library 2.6.5

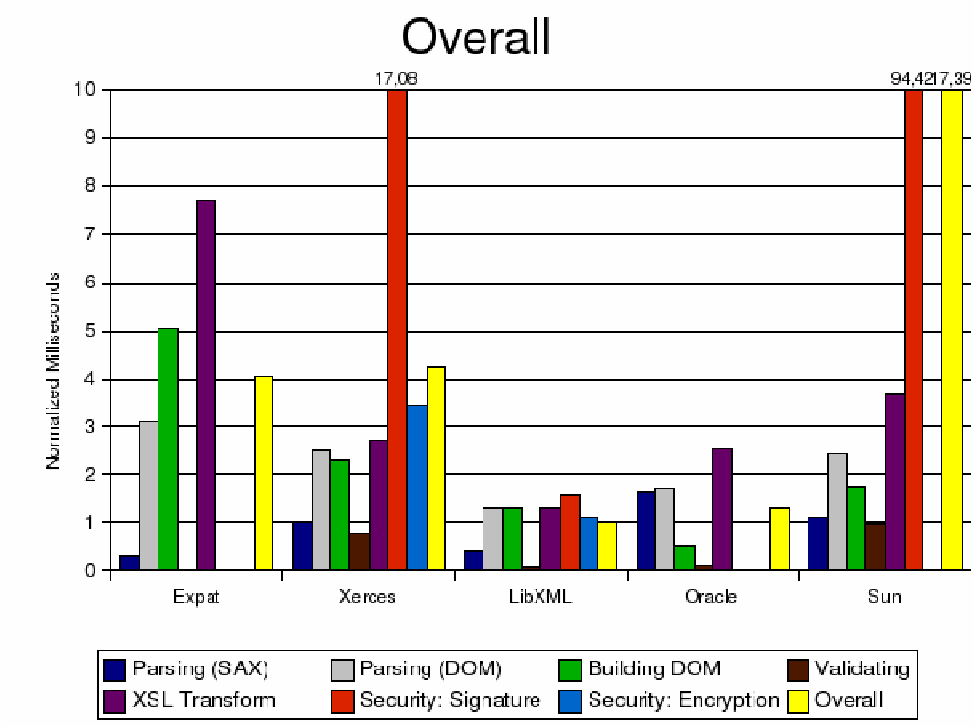
- Gnome XML Library 2.4.30
- Gnome XML DOM Library 0.8.1
- Gnome XSLT Library 1.1.2
- XML Security Library 1.2.4
- **Xerces for C++ 2.4.0**
- Xalan for C++ 1.7
- LTG RXP Parser 1.2.8
- Trolltech QT 3.3.0
- Oracle XML Developers Kit for C 9.2.0.6.0
- Oracle XML Developers Kit for Java 9.2.0.6.0
- Apache XML Security for C++ 08.02.2004
- Apache XML Security for Java 1.0.5D2

Vid testerna användes tre filer med data. Den första bestod av automatiskt genererade värden på tre nivåers djup och hade ett enkelt XML-schema. Den andra bestod av automatiskt genererade OPC-meddelanden och hade ett djup på fem nivåer. Meddelandena skiljde sig från varandra markant. Den tredje filen var en 11 mb stor rdf-fil från Dmoz.org.

Resultat

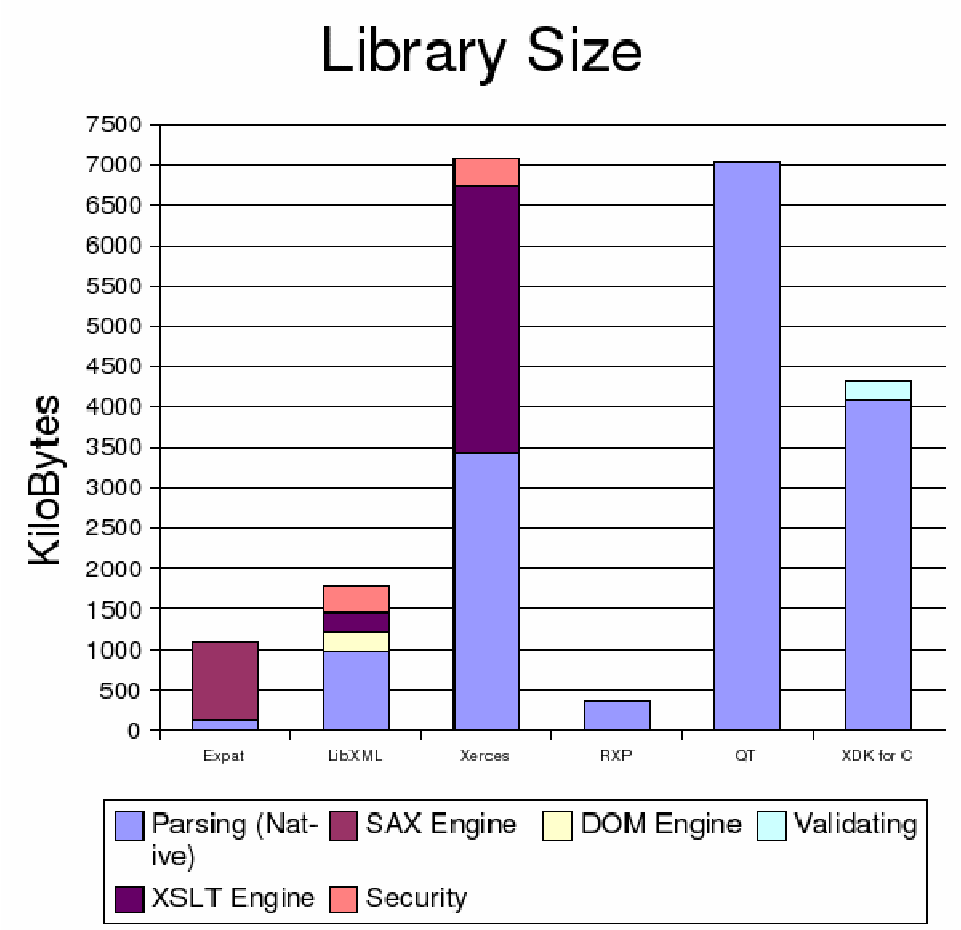
Flera olika saker testades och här redovisas de resultat som är väsentliga för denna rapport.

Det första är ett test av tiden det tog för de olika parsrarna att genomföra de olika uppgifterna. Som vi ser i figuren nedan är Expat mycket snabb när det gäller parsning med SAX jämfört med Xerces. När det gäller DOM-parsning är dock Xerces snabbare, även om det inte skiljer så mycket. Vid uppbyggnaden av DOM-träden är Xerces betydligt snabbare. Enligt testet får de båda parsrarna en total tid som är mycket lika men det bör noteras att Expat saknar stöd för säkerhetsfunktioner och validering. Eftersom validering är av stor betydelse för XML-parsern som skapas i examensarbetet är detta ett stort minus.



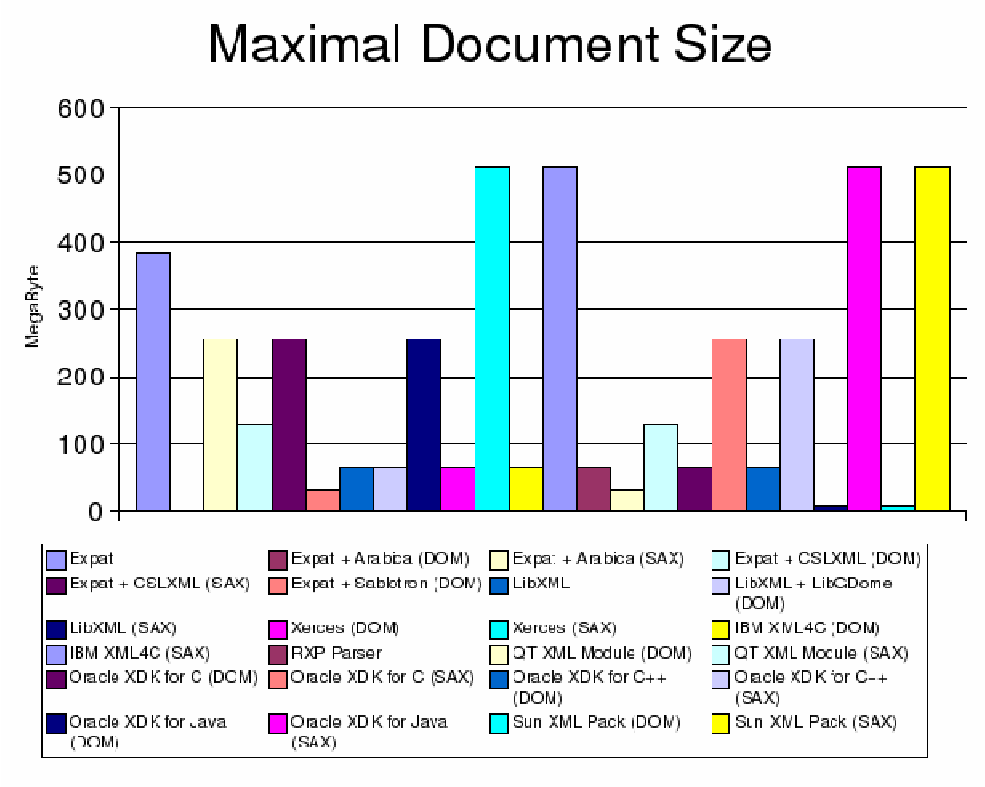
Figur 2, Tid för att utföra olika uppgifter.

Nästa figur visar storleken på parsrarnas olika delar i Kilobyte. Expat är mycket kompakt och det kan vara en stor fördel för vissa system. Xerces tar upp ungefär sju mb vilket är flera gånger mer än Expat. Enligt min åsikt är detta inget problem när det gäller valet av parser för detta arbete. Den XML-parser som skapas kommer att användas för att generera filer vid byggandet av menysystemet och kommer aldrig att behövas vid själva körningen.



Figur 3, Storlek på parsrarnas olika delar.

Den sista figuren för detta test visar den maximala storleken på XML-filerna som parsrarna klarar av. Enligt testet klara både Xerces och Expat filer på över 200 mb när SAX används. Att Xerces även klarar filer på över hela 500 mb är ingen stor fördel enligt min åsikt eftersom XML-filer av den storleken sällan används. När det gäller den maximala storleken vid användning av DOM jämförs Xerces med Expat + Sablotron eftersom Expat inte stödjer DOM själv. Skillnaden mellan de båda är ganska stor och även här klarar Xerces större filer. När det gäller valet för detta arbete tror jag inte det spelar någon större roll eftersom de XML-filer som beskriver menyerna aldrig kommer vara så stora som 20 mb, vilket Expat klarar.



Figur 4, Maximal storlek på XML-filerna.

Slutsatser

Enligt min mening är Xerces ett bättre val för examensarbetet. Storleken på biblioteket har liten betydelse och DOM är det som kommer att användas, vilket betyder att Xerces är snabbare. Den viktigaste orsaken är att Xerces kan validera XML-filerna mot XSD-scheman.

3.5.2 XercesC 2.7.0 – MSXML 4 – MSXML 6 – Expat 2.0.0 (9)

I detta test används validering och för att göra det möjligt i Expat används XSD 2.3.1.b1. Någonting som är viktigt att tänka på i detta test är att XSD inte använder sig av DOM eller SAX utan fungerar så att, utifrån XML-schemat skapas ett antal C++-filer som sedan används för att hämta datat ur XML-filen. Detta gör att det enligt min mening blir ett ganska orättvist test och på grund av detta kommer inte Expat (XSD) att diskuteras särskilt mycket här. Istället jämförs Xerces med MSXML.

Inställningar

Den hårdvara som användes var:

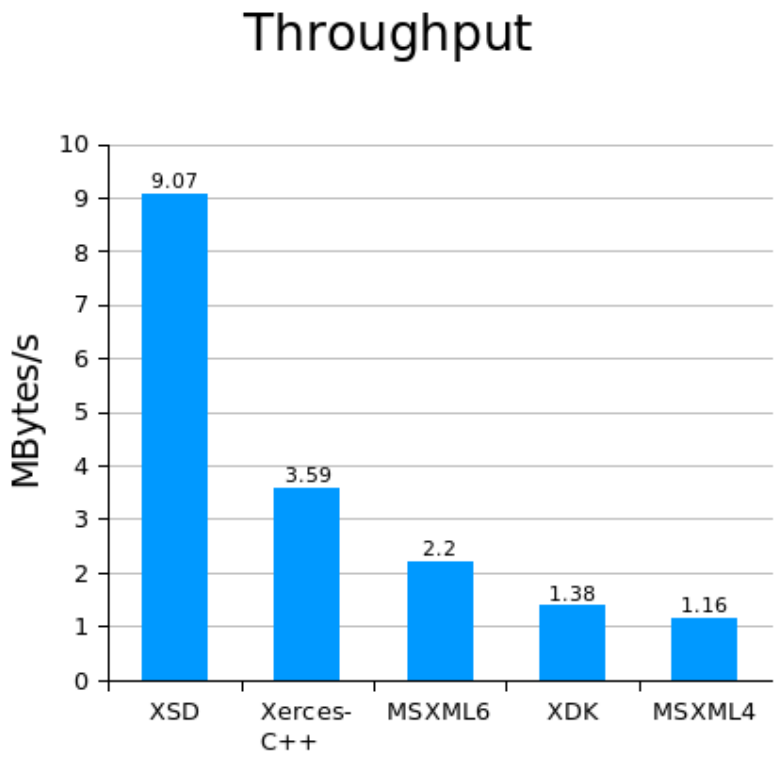
- Intel Pentium III Mobile 1Ghz, 512Kb L2 cache CPU
- IBM ThinkPad R31, based on Intel 830 chipset Motherboard
- 512Mb PC-133/133Mhz Ram

Mjukvara:

- Windows XP
- Visual C++ 7.1
- **XercesC 2.7.0**
- XSD 2.3.1.b1
- Expat 2.0.0 (tillsammans med XSD)
- **MSXML 4 SP2**
- **MSXML 6**
- XDK 10.1.0.2.0

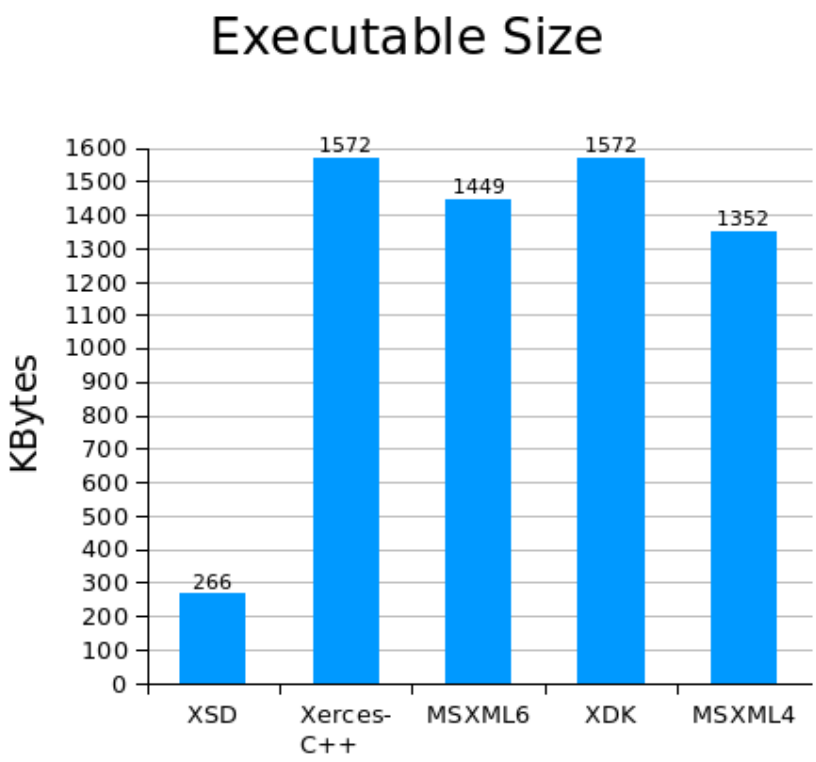
Resultat

Den första figuren visar genomströmningen. Xerces är snabbare än både MSXML 4 och MSXML 6 men MSXML har förbättrats mycket från version 4 till version 6. Värt att notera är att XSD, tillsammans med Expat, är mycket snabb, och den metod som används istället för SAX eller DOM kan vara värd att testa om hög hastighet behövs. Den nackdel jag kan tänka mig med den metoden är att den inte är lika standardiserad och därför krångligare att använda.



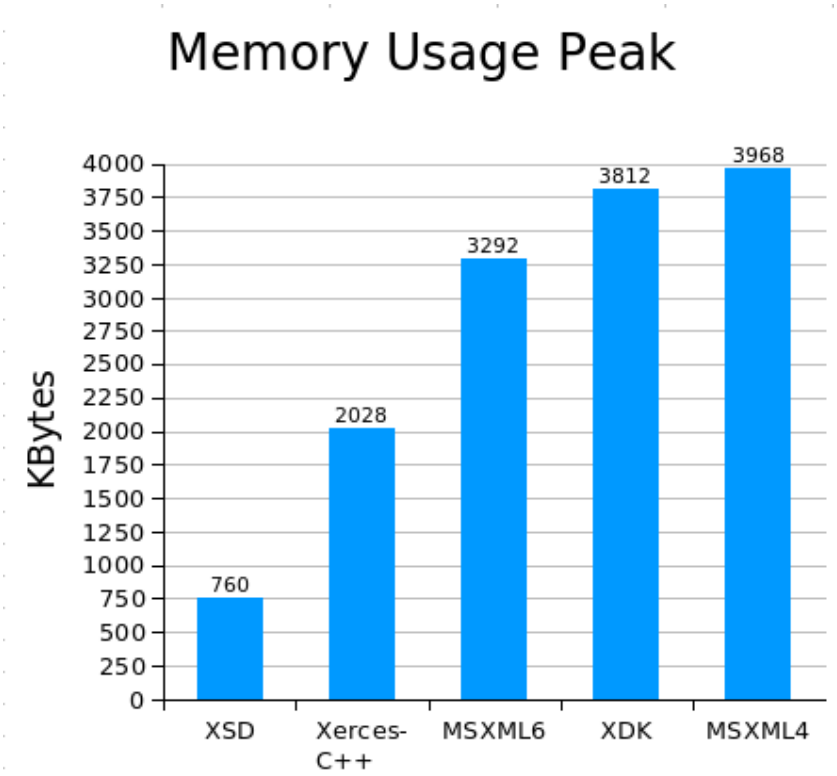
Figur 5, Genomströmning mb/s.

Nästa figur visar storleken för de olika parsrarna. Testet visar att Expat (XSD) är kompakt jämfört med all de andra parsrarna. Om man jämför storleken på Xerces i detta test med storleken som uppmättes i föregående test skiljer det ganska mycket. Orsaken till detta kan vara att en nyare version användes i detta test och storleken har minskat. En annan orsak kan vara vad som inkluderats i mätningen. Trots denna skillnad visar båda testen att Expat tar mycket mindre plats än Xerces och MSXML. Skillnaden mellan MSXML och Xerces är så pass liten att den inte borde ha någon betydelse.



Figur 6, Storlek KB.

Den sista delen av detta test visar minnesanvändningen och redovisas i figuren nedan. För uppmätning av detta noterades det maximala värdet vid körning och det avlästes i aktivitetshanteraren i Windows. Testet visar att Expat (XSD) använder lite minne. Xerces ligger ganska bra till men MSXML använder mycket minne i jämförelse med både Expat och Xerces. MSXML har förbättrats från version 4 till version 6 men ligger ändå en bit efter Xerces.



Figur 7, Maximal minnesanvändning KB

Slutsatser

Återigen visas det att Expat är snabb och använder lite resurser. Som nämnts ovan säger detta inte så mycket eftersom XSD använder en helt annan metod för att parsa datat än de andra parsrarna. Det faktum att den inte klarar av validering utan hjälp av något annat verktyg gör att den enligt min mening inte lämpar sig för användning i examensarbetet. Någonting att notera är att testet inte visar vad av jobbet som görs av XSD och vad som görs av Expat. Om det är så att själva valideringen är det som tar tid så är det inte Expat som gör att resultatet blir så bra, utan XSD.
MSXML får dåliga resultat jämfört med de andra i alla tre delar och därför visar även detta test, enligt min åsikt, att Xerces är ett bättre val.

Kapitel 4

4 De olika lösningarna

I detta kapitel redovisas de lösningar som implementerats. De tre delarna av arbetet tas upp och för varje del presenteras de filer som ingår samt hur de är uppbyggda. Lagring av information som till exempel relationen mellan föräldrar och barn i trädstrukturen för menyerna och lagring av menynamn med mera diskuteras och redovisas.

4.1 Menyhanteraren

4.1.1 Översikt Klasser

Menyhanteraren är uppbyggd av fyra klasser. För att använda sig av menyhanteraren behöver dessa fyra klasser inkluderas i det projekt där menyerna ska byggas på något vis. Nedan följer en beskrivning av de olika klasserna samt vad det har för uppgift.

MenuController (.cpp/.h)

Detta är huvudklassen för systemet. Klassen är ”singleton” vilket betyder att endast ett objekt av den skapas vid körning. Alla anrop till klassen görs på detta objekt och objektet skapas automatiskt vid det första anropet. Här lagras information om menystrukturen, historik, rotmeny, startmeny med mera. I denna klass finns alla metoder som ska användas för att komma åt menysidor från applikationen som använder sig av menyhanteraren. Metoder kallas till exempel när en knapptryckning genererats och då anropar menyhanteraren metoder i den menysida som är aktiv just då. Så fort en ny menysida öppnas anropar menyhanteraren automatiskt en ”enter-metod” i den nya menysidan och en ”leave-metod” i den gamla menysidan. Denna klass har även metoder som används för att konfigurera menystrukturen. Ett exempel på detta är att ange att en menysida är förälder till en annan i den trädstruktur som lagras.

MenuTree(.cpp/.h)

Denna klass används internt av menyhanteraren för att bygga upp den trädstruktur som lagras. Varje menysida representeras av ett objekt av denna klass och information som sparas i objektet är namnet på menysidan, vilka menysidor som är barn till den menysidan i trädstrukturen, vilka grupper menysidan tillhör samt en referens till det ”MenuItemBase-objekt” (se nedan) som skapats för menysidan.

MenuItemBase(.cpp/.h)

Denna klass används som en grund för att skapa en menysida. Varje menysida som skapas måste ärva denna klass. Klassen är virtuell och de metoder som ska köras till exempel när menysidan öppnas och stängs måste implementeras i den ärvda klassen.

MenuEvent(.cpp/.h)

Varje klass som representerar en menysida måste implementera en metod som heter ”update” och som kan köras från applikationen vid olika händelser. Metoden tar ett objekt av klassen ”MenuEvent” som parameter. Denna klass används som grund för ett sådant objekt. Klassen är dock virtuell och meningen är att en ny klass som ärver denna ska skapas för varje typ av händelse. Sedan kan ett objekt av den ärvda klassen skickas till ”update-metoden” och i varje menysida görs det som ska göras beroende på vilken typ av händelse som tas emot.

4.1.2 Viktiga metoder

Här presenteras några av de viktigaste metoderna i varje klass.

MenuItemBase(.cpp/.h)

De metoder som tas upp här är de fyra metoder som måste implementeras av de klasser som ärver denna klass.

- **enterMenu** – Denna metod tar inga parametrar och körs automatiskt när menysidan öppnas. Det som kan göras här är till exempel att uppdatera den grafiska vyn i applikationen.
- **leaveMenu** – Denna metod körs när menysidan lämnas och ett exmepel på vad som kan göras här är att dölja saker som var specifika för just denna menysida i den grafiska vyn.
- **buttonPressed(int keyCode)** – Denna metod är tänkt att köras genom att applikationen anropar en metod med samma namn i ”MenuController”-klassen och den i sin tur anropar denna metod i den menysida som är aktiv just då.
- **Update(MenuEvent* event)** – Som det nämndes ovan kan denna metod anropas då olika händelser sker i applikationen. Här kollas den händelse som skickas som argument och beroende på vilken typ av händelse det är kan olika saker göras. Om en menysida inte ska reagera på några händelser lämnas denna metod tom.

MenuEvent(.cpp/.h)

Denna klass har inga metoder som måste implementeras i de klasser som ärver den, förutom destruktorn. Eftersom klassen är virtuell kan den inte användas som den är för att skapa objekt utan den måste ärvas av en skapad klass.

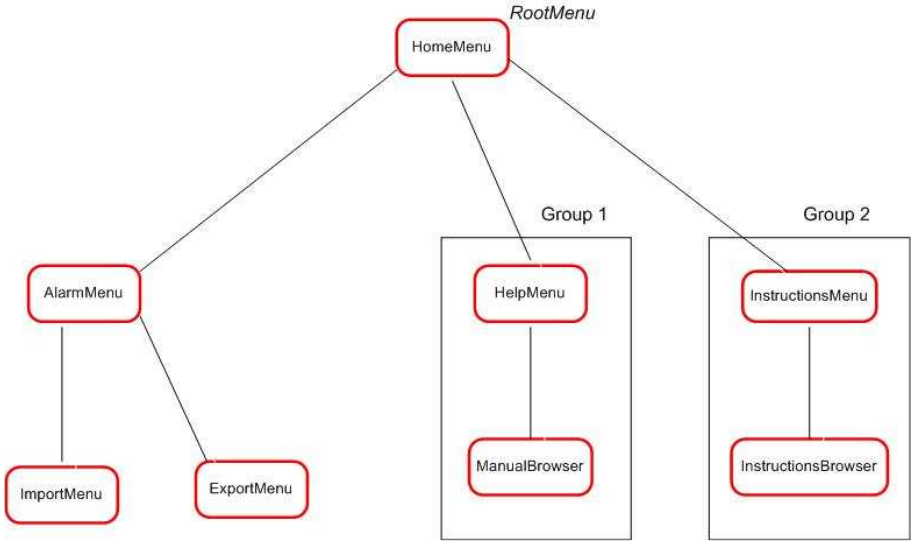
MenuController(.cpp/.h)

Metoderna som ingår i denna klass presenteras här. Först de metoder som används för att komma åt menyhanteraren samt de som används för att ställa in menyhanteraren och sedan de metoder som används för att navigera bland menysidorna. För exempel på hur metoderna kan

användas samt detaljerad API hänvisas till den bifogade manualen med namnet ”Menu Manager – Programmers Guide”.
De metoder som används för att komma åt samt ställa in menyhanteraren är:

- **Instance()** – Denna metod används för att komma åt menyhanteraren. Den returnerar det objekt som är skapat om ett sådant finns och skapar det samt returnerar objektet om det inte finns.
- **Destroy()** – Denna metod ska användas för att terminera menyhanteraren. Det som görs när denna metod kallas är att köra destruktorn i klassen.
- **addMeny(MenuItemBase* newMenu)** – Denna metod används för att lägga till en menysida till strukturen i menyhanteraren.
- **setParentMenu(MenuItemBase* theMenu, MenuItemBase* newParent)** – För att ange hur menyerna hänger ihop i den trädstruktur som finns i menyhanteraren används denna metod. Den säger att ”newParent” är förälder till ”theMenu”.
- **addNewGroup(int groupNumber, MenuItemBase* activeMenuInGroup)** – Menyhanteraren stödjer grupper av menysidor som kommer ihåg vilken menysida som var senast besökt i gruppen och öppnar den när gruppen återbesöks. Denna metod skapar en grupp och bestämmer vilken menysida som ska vara aktiv i gruppen första gången den besöks.
- **addMenuToGroup(MenuItemBase* theMenu, int groupNumber)** – För att lägga till fler menysidor till en redan skapad grupp används denna metod.
- **setRootMenu(MenuItemBase* newRootMenu)** – Denna metod måste användas och sätter vilken menysida som ska vara rot, alltså på den högsta nivån, i trädstrukturen.
- **setStartMenu(MenuItemBase* startMenu)** – Även denna metod måste användas och den sätter vilken menysida som ska visas vid start an applikationen.
- **verifyMenuTree()** – Denna metod kan användas för att kontrollera den trädstruktur som finns lagrad i menyhanteraren. Returvärdet talar om vilken typ av fel som hittats. Denna metod kommer även att returnera ett fel om någon av de metoder som används för att sätta upp trädstrukturen har returnerat ett fel.
- **setHistorySize(int newSize)** – Denna metod används för att ändra det antal steg i historiken som det är möjligt att backa.

Ett exempel på hur trädstrukturen kan se ut visas nedan.

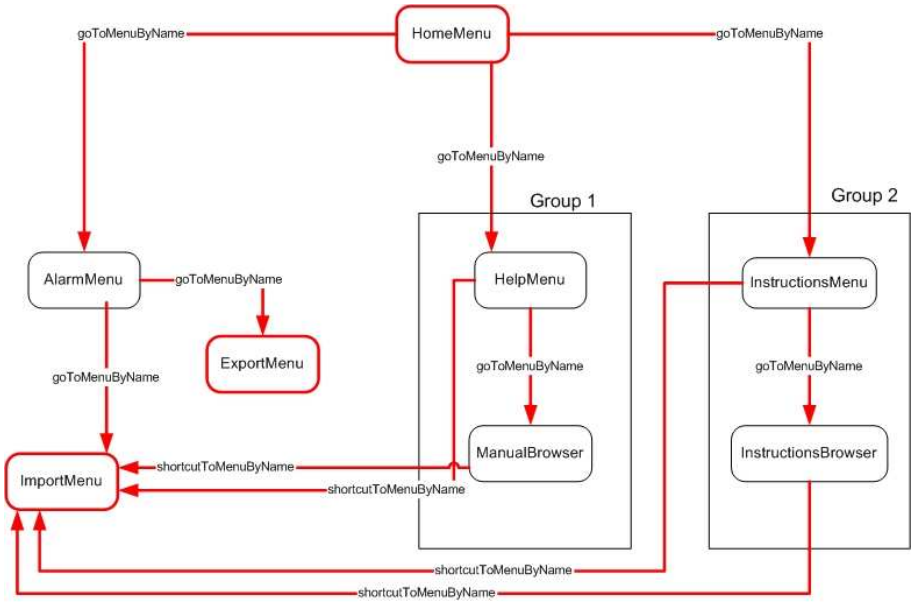


Figur 8, Exempel på trädstruktur i menyhanteraren.

De metoder som används för att navigera mellan menysidorna är:

- **buttonPressed(int keyCode)** – Om olika saker ska göras beroende på vilken menysida som är aktiv används denna metod. Genom att menyhanteraren anropar den menysida som är aktiv vid det tillfälle då metoden anropas och skickar vidare värdet på ”keyCode” kan varje meny göra det den vill beroende på vilken knapp som tryckts ner.
- **goToMenuByName(String menuName)** – För att navigera till en menysida som är barn till den aktiva menysidan i strukturen används denna metod. Om denna metod används för att försöka navigera till en menysida som inte är barn till den aktiva menysidan kommer den att returnera ett fel och inte göra någonting.
- **shortcutToMenuByName(String menuName, bool useGroupState=false)** – Denna metod används för att navigera till en menysida som inte är barn till den aktiva menysidan. Ett exempel på när detta kan vara användbart är för att hoppa till startmenysidan från vilken menysida som helst. Den andra parametern till metoden anger att om den grupp som den nya menysidan tillhör, om den tillhör en grupp, ska öppna den senaste aktiva menysidan i gruppen eller om den menysidan som hoppet görs till ska öppnas. Om den andra parametern utelämnas kommer den att sättas till ”false” och grupphistoriken ignoreras.
- **goToParent()** – Denna metod används för att stega upp en nivå i trädstrukturen.
- **goToRootMenu()** – Denna metod kan användas för att gå till den menysidan som är rot i trädstrukturen.
- **goBack()** – För att gå tillbaka till den menysida som var öppen före den aktiva menyn kan denna metod användas. Det möjliga antalet steg att backa är satt till tio som standard men kan ändras.

Nedan visas ett exempel på hur navigering i den struktur som presenterades i föregående figur kan göras.



Figur 9, Exempel på navigering i trädstrukturen.

4.2 XML-parsern

Den XML-parser som skapats kan användas för att automatiskt, utifrån en XML-fil, generera kodfiler för de menysidor som ska skapas.

4.2.1 Översikt klasser

XML-parsern består av tre klasser. De är MenuBuilder.cpp, XMLParser.cpp och DOMErrorReporter.cpp. Huvudklassen är MenuBuilder.cpp och den innehåller en mainmetod som körs när det kompilerade programmet körs. När den körs skapas ett objekt av klassen XMLParser.cpp och detta objekt används för att parsa den angivna XML-filen och generera kodfilerna. När XML-filen parsas valideras den mot ett XSD-schema som heter XMLMenuBuilderSchema.xsd. Om fel uppstår vid valideringen rapporteras de till användaren med hjälp av DOMErrorReporter.cpp och parsningen avbryts.

4.2.2 Användning

För att kunna använda XML-parsern behövs en korrekt XML-fil. Parsern är en konsolapplikation och det syntax som används för att köra den är,

```
./MenuBuilder.exe Filename
```

där ”Filename” är namnet på XML-filen. De genererade kodfilerna hamnar i en mapp som heter ”output”. Om den mappen inte finns skapas den och om den inte är tom ges ett felmeddelande och parsningen avbryts. Om valideringen mot XSD-schemat inte lyckas avbryts också parsningen, och de fel som påträffats presenteras. För varje meny som specificerats i XML-filen skapas en h-fil och en cpp-fil. Utöver detta skapas även en klass som heter MenySystem.cpp och den kan användas för att ställa in menyhanteraren och sätta upp menystrukturen. För att göra detta kan följande kod användas:

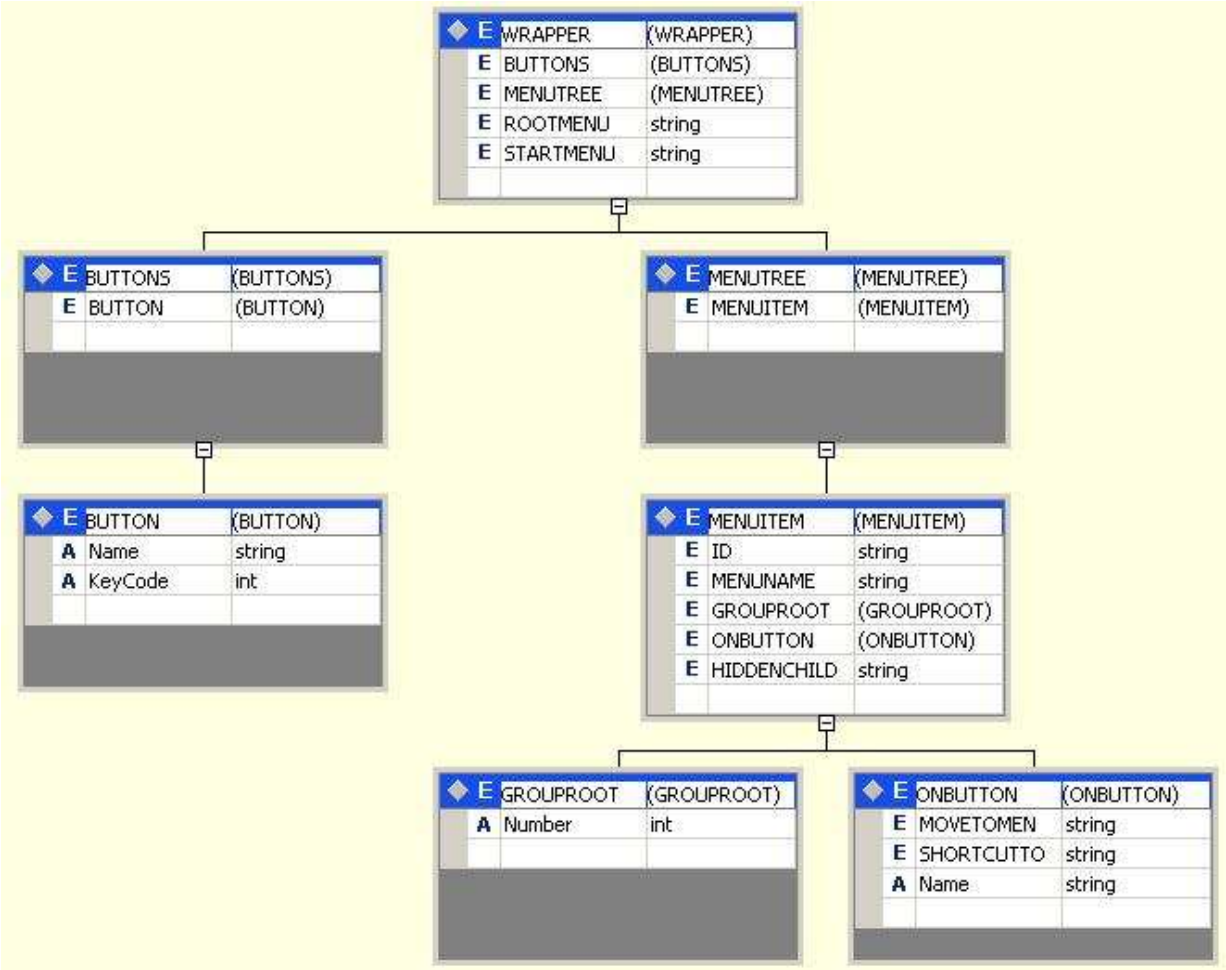
```
MenuSystem* MenuSystemObj = new MenuSystem();
```

När detta är gjort behöver inte det skapade objektet användas för att komma åt menystrukturen, utan den finns lagrad i den menyhanterare som är inkluderad i applikationen som utför detta.

4.2.3 Formatet på XML-filen

Som det nämndes ovan måste XML-filen som används vara korrekt och den måste validera mot det XSD-schema som följer med XML-parsern. Detta avsnitt visar hur en korrekt XML-fil ska se ut.

Strukturen på XML-filen måste följa detta mönster:

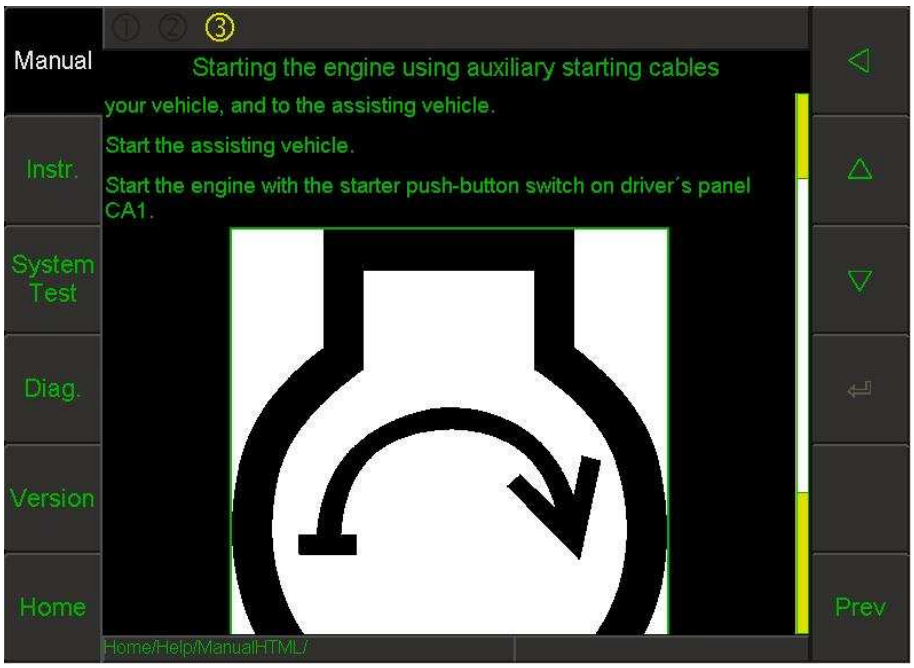


Figur 10, Formatet på XML-filen som används av XML-parsern.

För detaljerad information om hur de olika elementen måste se ut, se den bifogade manualen ”XML Menu Builder – User Guide”.

4.3 Införande i VIS

Den sista delen av examensarbetet gick ut på att införa den skapade menyhanteraren i den testapplikation som används på företaget. Tidigare användes en del av Boost som tillståndsmaskin för menyerna. Genom att förstå hur menyer var uppbyggda och hur de var kopplade till grafiken i testapplikationen kunde mycket kod från de gamla menysidorna användas när menyerna byggdes upp med hjälp av den nya menyhanteraren.
Eftersom koden för VIS och för testapplikationen är ganska omfattande kommer inte koden presenteras här. Följande bild visar hur en menysida kan se ut i testapplikationen.



Figur 11, En menysida i VIS testapplikation,

Knapparna på vänster sida kan användas för att navigera till andra menysidor och när en av knapparna trycks ned kallas funktioner i menyhanteraren. Det är upp till menyhanteraren att se till att rätt menysida öppnas och att historik sparas med mera. Knapparna på höger sida kallar också på funktioner i menyhanteraren. Menyhanteraren skickar vidare knapptryckningarna till den aktiva menysidan. Ett exempel är om knappen med en pil nedåt trycks ned. Då körs kod som gör att grafiken påverkas och sidan skrollas nedåt.
Ungefär 40 menysidor ingår i testapplikationen och mycket av arbetet med denna del gick ut på att konvertera de gamla menysidorna till menysidor som används i den nya menyhanteraren. Förutom att skapa alla nya menysidor byggdes den trädstruktur som lagras i menyhanteraren upp.
Denna del av examensarbetet var viktigt eftersom menyhanteraren testas i en miljö som är mycket lik den som används i företagets produkter. Testapplikationen används för intern testning av olika delar i VIS men även för att se hur en ny kundapplikation kan använda de olika delarna i

VIS. Trots att den manual som skapats till menyhanteraren ska räcka för att skapa ett nytt kundprojekt tror jag att det lättaste sättet är att studera testapplikationen och använda den som en grund när ett nytt projekt skapas.

Kapitel 5

5 Slutsatser

Eftersom examensarbetet gick ut på att implementera tre olika delar kan de olika delarna ses som resultatet av arbetet. Om man jämför specifikationerna i början av rapporten med lösningarna i det föregående kapitlet och i de manualer som bifogas ser man att det mesta stämmer bra överens. Min uppfattning är också att resultatet blev bra och tiden räckte till.
Menyhanteraren, som var huvuddelen i arbetet, fungerar bra och klarar av alla uppgifter som den föregående tillståndsmaskinen klarade av. Funktioner som historik samt möjlighet att skapa grupper av menysidor som kommer ihåg vilken sida som senast var aktiv då gruppen besöks gör att möjligheter att skapa kundapplikationer som är mer avancerade finns.
En av anledningarna att tiden räckte till så bra som den gjorde tror jag var att en bra planering gjordes de första dagarna och att mycket tid för att implementera de olika delarna planerades. Självklart har inte planeringen följts helt och hållet och den har justerats under arbetets gång. Skrivandet av rapporten påbörjades långt innan implementeringen var klar och det tror jag var bra.

5.1 Begränsningar

Några begränsningar som strider mot specifikationerna kan jag inte komma på utan det mesta fungerar som det ska.

5.2 Utökning

Möjligheten att utöka menyhanteraren och XML-parsern är stor. Ett exempel för menyhanteraren är att menystrukturen läses in från en XML-fil vid uppstart av systemet istället för att metoder används för att sätta upp detta manuellt.
När det gäller XML-parsern kan mycket förbättras. Den enda kontroll av XML-dokumentet som görs är att det validerar mot ett XSD-schema som följer med parsern. Om man försöker skapa en XML-fil som är korrekt med avseende på schemat men som innehåller felaktig menystruktur på något vis tror jag man skulle lyckas. Å andra sidan, om man försöker skapa en korrekt struktur tror jag man lyckas med det ganska lätt.
En annan utökning av XML-parsern vore att göra det möjligt att ändra de genererade kodfilerna och sedan generera om dem med till exempel en annan menystruktur, men ändå ha kvar de manuella ändringar man gjort. Detta skulle kunna göras genom att det finns områden i de genererade filerna som man får ändra och områden som man inte får ändra.

5.3 Framtida användning

Som nämnts ovan används redan menyhanteraren i den testapplikation som används för att testa VIS och det verkar fungera bra. De kundapplikationer som redan finns idag och som är byggda med hjälp av Boost skulle kräva ganska mycket arbete för att anpassas till den nya menyhanteraren så det är nog inte aktuellt. Om en ny kundapplikation ska skapas tror jag att man kommer att använda menyhanteraren som skapats. Att veta att arbetet skulle resultera i något som behövdes och som man var intresserad av att använda har varit inspirerande.

Kapitel 6

6 Tack

Jag vill tacka personalen på avdelningen för VIS Bas på Hägglunds i Örnsköldsvik. Att få medverka vid bland annat veckomöten har, trots att det inte alltid haft något med examensarbetet att göra, varit givande för min fortsatta utveckling.

Tack till min handledare Jerry Eriksson och examinator Per Lindström på Umeå Universitet.

Ett extra stort tack till Stefan Westman som var min handledare på Hägglunds. De diskussioner och synpunkter som Stefan bidragit med har gjort detta till ett, i min mening, perfekt examensarbete!

Jag vill tacka Marlene Westman som var min första kontakt på Hägglunds och som gav mig detta examensarbete och har sett till att vi examensarbetare har känt oss som en del av företaget under denna tid.

Till sist vill jag tacka Conny Tegström, som jag delat skrivbord med på Hägglunds och som jag har diskuterat mycket med under arbetets gång, och önska honom lycka till med den fortsatta karriären!

Till de läsare som fortfarande studerar och som ska göra ett examensarbete inom datavetenskap vill jag passa på att rekommendera Hägglunds i Örnsköldsvik!

7 Källförteckning

1. Clark Cooper, Using Expat.
http://www.xml.com/pub/a/199/09/expat/index.html
(besökt 2007-06-26).

2. The Apache Software Foundation, Xerces C++ Parser.
http://xerces.apache.org/xerces-c/
(besökt 2007-09-25).

3. Microsoft Corporation, MSXML.
http://msdn2.microsoft.com/en-us/library/ms763742.aspx
(besökt 2007-09-25).

4. Nazmul Idris, Should I use SAX or DOM?
http://developerlife.com/saxvsdom/default.htm
(besökt 2007-09-26, skapad 1999-05-22).

5. BAE Systems in Sweden
1. BAE Systems in Sweden.PPT.

6. Boost
http://www.boost.org.

7. Wikipedia, XML
http://sv.wikipedia.org/wiki/XML
(besökt 2007-11-12)

8. xmlbench, XML Benchmark Results 08.02.2004
http://xmlbench.sourceforge.net/results/benchmark200402/index.html
(besökt 2007-11-07)

9. XSDBench, XML Schema Benchmark Results
http://www.codesynthesis.com/projects/xsdbench/results/2006-10-16-02/
(besökt 2007-11-07)

10. Wikipedia, XML Schema (W3C)
http://en.wikipedia.org/wiki/XML_Schema
(besökt 2007-12-11, ändrad 2007-12-08)

Bilaga A

8 Manualer

De manualer som bifogas här är skrivna för att användas som dokumentation vid användning av menyhanteraren och XML-parsern. De är skrivna på engelska för att följa företagets regler men bifogas ändå eftersom de förklarar systemens uppbyggnad och hur de ska användas.

MENU MANAGER – PROGRAMMERS GUIDE

Revision history

Date	Issue	Description	Author
2007-11-29	PA1	Programmers guide for VISbas Menu Manager.	DS

Content

1 INTRODUCTION

This document describes how the menu manager should be used to build menu pages and setup the menu structure. First the different parts of the menu manager are discussed and after that a guide on using the menu manager is given. In the last chapter an API is provided.

2 COMPONENTS AND FEATURES

2.1 Classes

This section describes the different classes of the menu manager. For each class the main responsibilities are given. For more detailed description of methods and how to use the classes see the next section.

2.1.1 MENUCONTROLLER (.CPP / .H)

This is the main class of the menu manager. The class is a singleton, which means only one object will be created in runtime. Here all information about menu structure, history, root menu, start menu and more things will be stored.
This class have all methods that will be used to access menus from outside the menu manager. The methods are, for example, used to go to a specific menu page by giving the name of the page or go to the previous menu in the history.
Whenever a new menu page is opened, a enter method will be executed in the new menu and a leave method will be executed in the previous menu.
This class also have methods that are used to setup the whole menu structure. For example by adding new menus to the structure, specifying parent-to-child relations in the menu tree structure and creating groups.

2.1.2 MENUTREE (.CPP/.H)

This class is used by the menu controller to save the information about menu structure. Each menu page will get its own MenuTree object that will include information about the menus children in the structure and the menu name as well as information about which groups this menu are a member of and a pointer to the MenuItemBase object for the menu.

2.1.3 MENUITEMBASE (.CPP/.H)

This class are virtual and have to be extended to build each menu page. This is where methods, which for example will be executed on opening of a new menu, will be implemented. In next section the methods that have to be implemented will be discussed.

2.1.4 MENUEVENT (.CPP/.H)

This is also a virtual class and it will be used to create different events that can be sent from the menu controller to the currently active menu when the menu controller is receiving a certain update. An example of this is if the menu controller is listening for clock ticks, on each tick

the menu controller sends a “ClockTickMenuEvent” to the active menu.

2.2 Methods / Menu building guide

This section shows how methods in the different classes are used to build the menu structure and how to navigate in the structure as well as what are expected to be done when implementing the virtual methods in each menu page. The order of the classes is not same as in the previous section. Instead the classes and methods that are expected to be used first when building the menu system are discussed first.

2.2.1 MENUITEMBASE

First of all, we need some menu pages. To create a menu page a new class have to be created. It will extend this class and this is done with the following syntax (where HomeMenu is the new menu):

```
class HomeMenu : public MenuItemBase
```

The new menu class have to implement the following methods:

- `void enterMenu()`
- `void leaveMenu()`
- `void buttonPressed(int keyCode)`
- `void update(MenuEvent* event)`

2.2.1.1 enterMenu

This method takes no arguments and will be executed whenever the menu page is opened. Things that could be done here is to setup GUI buttons and update the current view.

2.2.1.2 leaveMenu

This method is executed when a menu is leaving the active state. One thing that could be done here is to hide things from the view that was specific for this menu.

2.2.1.3 buttonPressed

As we will see later, hardware button (or software GUI buttons) will be mapped to integers in the menu controller. When a button is pressed the menu controller call this function in the active menu. This method should do the right thing depending on which button that was pressed. An example is to open another menu page by calling one of

the functions in the menu controller. However, things like updating the current view or other things can also be done here.

2.2.1.4 update

This method will be executed by the menu controller when the menu controller receives an event from something it is listening too. To make it possible for a menu page to listen to different kinds of events the MenuEvent class can be used and an object of a class that extends MenuEvent is needed as argument to this method.
If the menu should not listen to any events from the menu controller this method should be empty, but it have to be implemented because it is pure virtual.

2.2.2 MENUEVENT

This class does not include any methods that have to be implemented, other than the destructor, but the class have to be extended. The main reason for the class is to serve as a ground for classes that will represent events passed as argument from the menu controller to the active menu. Since the update method in the menu class takes a MenuEvent* as argument, any object of a class that extends this class can be used.
Note that the menu controller does not necessarily have to be used to send events to menus. Another way of doing it is to use the menu controller to get the current active menu and then pass the event directly to the menu.

2.2.3 MENUCONTROLLER

Instead of describing every method in the menu controller class we will look at an example on how to setup the menu controller and the menu structure as well as examples on how to navigate in the menu structure after it is setup. View the API in the next chapter for detailed description of each method.

2.2.3.1 Setup the menu controller

When we have some menus and maybe some events to use it is time to setup the menu controller. The syntax to access the menu controller is:

```
MenuController::Instance()
```

This will get the already created object of the menu controller if there is one and create a new object if it doesn't exist already.

To create objects of the menu classes we have implemented and add them to the menu structure, the following can be used:

```
menuObj = new Menu();
menuObj->setName("Home");
MenuController::Instance()->addMeny(menuObj);
alarmMenuObj = new AlarmMenu();
alarmMenuObj->setName("Alarm");
MenuController::Instance()->addMeny(alarmMenuObj);
```

Now we have two menus that are accessible from the menu controller. The next step is to setup the relationship between the menus. The structure will be saved as a tree and to set the menu with the name "Home" as a parent to the menu with the name "Alarm" the following can be used:

```
MenuController::Instance()->setParentMenu(
                        alarmMenuObj,
                        menuObj);
```

We can create a group that include a menu and all menus under that menu in the structure with the following syntax:

```
MenuController::Instance()->addNewGroup(1, menuObj);
MenuController::Instance()->
            addMenuToGroup(alarmMenuObj, 1);
```

The purpose of a group is to remember the last visited menu when the group is left and to open that menu the next time the group is visited. If a menu is used to start a new group, like "menuObj" is in the

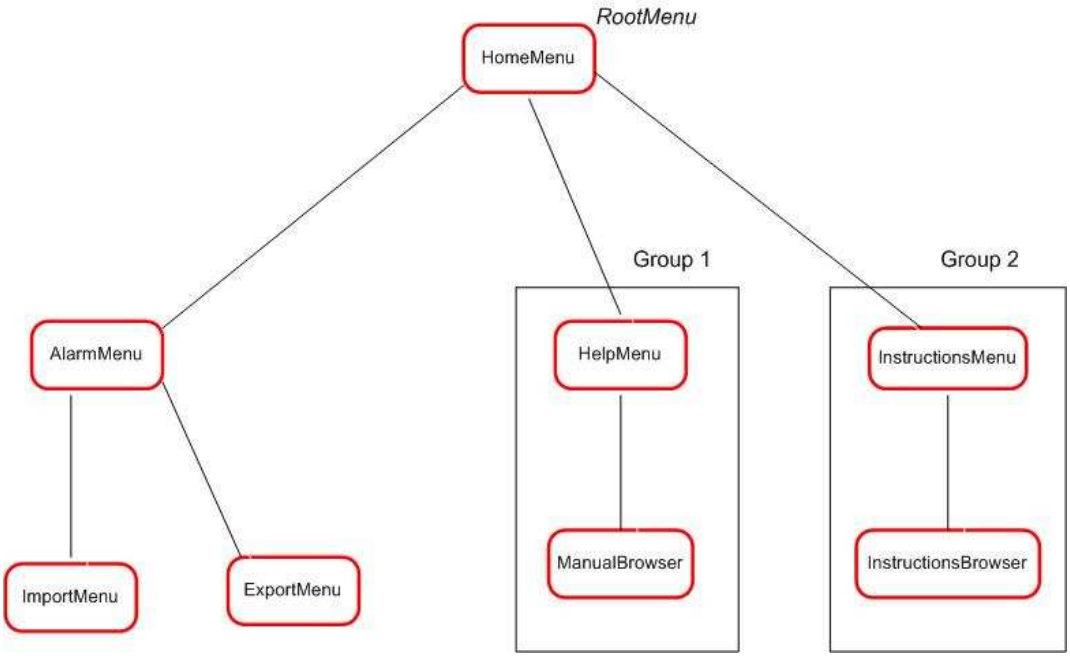
example, all groups under that menu in the structure should be added to the group as well.

The last thing we want to do is to set the root menu in the structure and to specify which menu to open on start up. The two methods for this are used like this:

```
MenuController::Instance()->setRootMenu(menuObj);
MenuController::Instance()->setStartMenu(menuObj);
```

The root menu and the start menu do not necessarily have to be the same menu.

An example of the tree structure setup can look like this:



2.2.3.2 Navigate in the menu structure

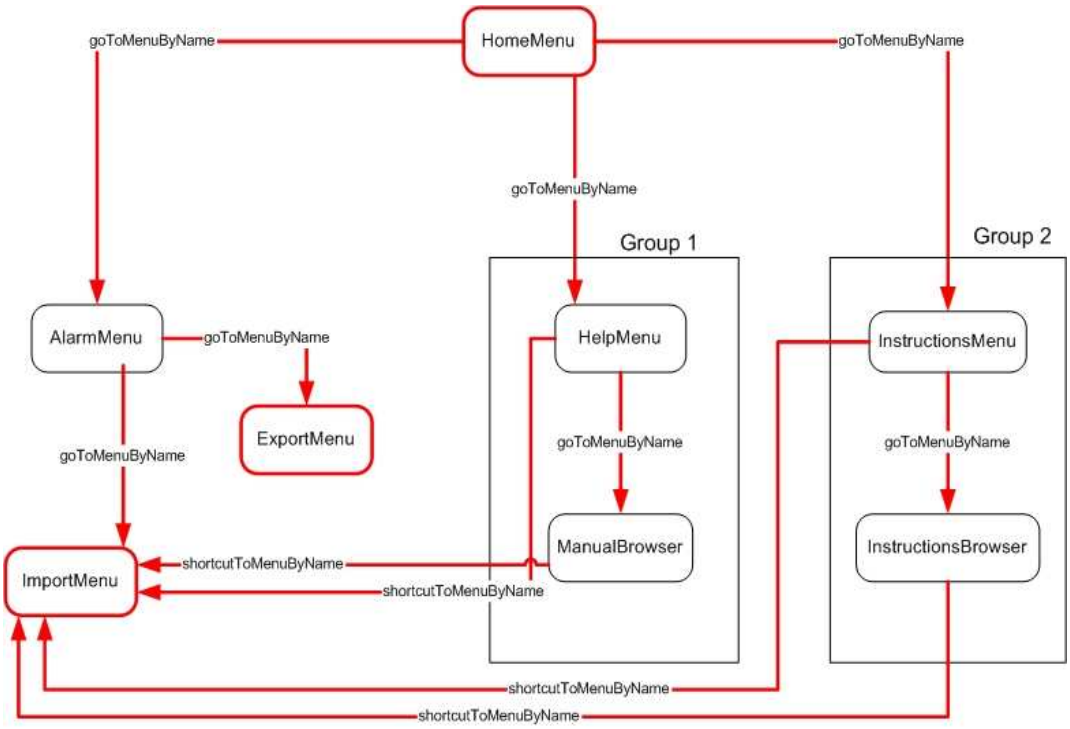
All navigation between menus should be done using the menu controller. The methods provided are:

- `buttonPressed(int keyCode)`
- `goToMenuByName(String menuName)`
- `shortcutToMenuByName(String menuName, bool useGroupState=false)`
- `goToParent()`
- `goToRootMenu()`

- `goBack()`

For information about each method see the API in next chapter. Something to mention is that when navigating to a child in the menu structure the goToMenuByName method should be used and when navigating to a menu that is not a child the shortcutToMenuByName method have to be used. The second parameter to shortcutToMenuByName is not needed, but can be used to specify that if a group is entered, the group's active menu will be opened, in the same way that goToMenuByName uses group history.

Here is an example of how navigation can be done between the menus that was setup in the previous section:



2.2.3.3 Terminating the menu controller

The standard destructor should not be used. Instead, the following method should be used to terminate the menu controller:

- `Destroy()`

Note that, if the menu controller is terminated, the next call to it will create a new object and the old setup will be lost.

3 API

3.1 MenuController Class Reference

This controls the state of the menu system and holds information about main menu, current menu and history.

3.1.1 PUBLIC TYPES

Public Constants

```
enum            verifyReturn
                {
                  BAD_GROUP = -5,
                  NO_MENU_NAME_OR_MENU_REFERENCE,
                  CHILD_TO_MANY_MENUS,
                  NO_PARENT_SET,
                  MENU_SETUP_ERROR,
                  SUCCESS
                }
```

3.1.2 PUBLIC MEMBER FUNCTIONS

static MenuController*	**Instance**()
void	**Destroy**()
bool	**addMeny**(MenuItemBase* newMenu)
bool	**addMenuToGroup**(MenuItemBase* theMenu, Int groupNumber)
bool	**addNewGroup**(int groupNumber, MenuItemBase* activeMenuInGroup)
bool	**buttonPressed**(int keyCode)
bool	**clearHistory**()
MenuItemBase*	**getActive**()
std::list<String>	**getCurrentPath**()
std::list<int>	**getCurrentPathAsId**()

MenuItemBase*	**getMenuByName**(String menuName)
bool	**goBack**()
bool	**goToMenuByName**(String menuName)
bool	**goToParent**()
bool	**goToRootMenu**()
bool	**isGoBackPossible**()
void	**setHistorySize**(int newSize)
bool	**setParentMenu**(MenuItemBase* theMenu, MenuItemBase* newParent)
bool	**setRootMenu**(MenuItemBase* newRootMenu)
bool	**setStartMenu**(MenuItemBase* startMenu)
bool	**shortcutToMenuByName**(String menuName, bool useGroupState=false)
verifyReturn(int)	**verifyMenuTree**()

3.1.3 PROTECTED MEMBER FUNCTIONS

Hidden Constructor and Destructor

MenuController();
~**MenuController**();
Use the Instance() and Destroy() methods instead.

3.1.4 DETAILED DESCRIPTION

This controls the state of the menu system and holds information about main menu, current state and history. Only one instance of this class will be created and all menu pages can interact with this object.

3.1.5 MEMBER FUNCTION DOCUMENTATION

static MenuController* **Instance**()
Used to get the current instance if this class or if it don't exist create a new instance.

Returnvalue

MenuController - A pointer to the current menucontroller.

void **Destroy**()
Use this to delete this singleton object. This will run the destructor.

bool **addMeny**(MenuItemBase* newMenu)
Add a new menu to the structure

Parameters

newMenu - [in] A pointer to the MenuItemBase object that will be added.

Returnvalue

bool - false if the menu already exist or the name stored in the menuitem is not set, true if successful

bool **addMenuToGroup**(
MenuItemBase* theMenu,
Int groupNumber)

Add a menu to a group.

Parameters

theMenu - [in] the menu that should be added.
groupNumber - [in] the group number.

Returnvalue

bool - false if the menu or the group don't exist, true if success

bool **addNewGroup**(int groupNumber,
MenuItemBase* activeMenuInGroup)
Add a new group and set its current active menu. This will also add the menu to the group if it's not already added.

Parameters

groupNumber - [in] the number of the new group
activeMenuInGroup - [in] a pointer to the active menu.

Returnvalue

bool - false if the group already exist or the menu don't exist, true if success

bool **buttonPressed**(int keyCode)
The method is executed when the MenuController receives a button press from the main application. The keyCode are passed to the active menu.

Parameters

keyCode - [in] the code mapping the key to the GUI

Returnvalue

bool - false if the active menu is not set, true if successful

bool **clearHistory**()
Clear the history.

Returnvalue

bool - true if successful, false if error on list operation

MenuItemBase* **getActive**()
Get the currently active menu pointer.

Returnvalue

MenuItemBase* - The pointer to the currently active menu

std::list<String> **getCurrentPath**()
Used to get the path to the current menu. The returned list is ordered so that the root menu is in first place and the current menu is in last place of the list.

Returnvalue

std::list<String> - a list with String's representing the path. This list will be empty if the root menu is not set.

std::list<int> **getCurrentPathAsId**()
Used to get the path to the current menu. The returned list is ordered so that the root menu is in first place and the current menu is in last place of the list.

Returnvalue

std::list<int> - a list with id's representing the path. This list will be empty if the root menu is not set.

MenuItemBase* **getMenuByName**(String menuName)
Get menu from a name. This will simply get it form the menuMap.

Parameters

menuName – [in] the name of the menu

Returnvalue

MenuItemBase* - A pointer to the menu with the given name.

bool **goBack**()
Go back in the navigation history.

Returnvalue

bool - false if no history are present, true if successful

bool **goToMenuByName**(String menuName)
Go to a menu from a name. This will run the enter method in the menu and make it active. Only children menus will be accessible this way. To go to root menu or use a shortcut use goToRootMenu() or shortcutToMenuByName() methods.

Parameters

menuName – [in] the name of the menu

Returnvalue

bool - false if the menu step is forbidden or the menus don't exist, true if success

bool **goToParent**()
Navigate up to the parent in the menu tree structure.

Returnvalue

bool - false if parent don't exist or if the current menu is the root menu, true if success

bool **goToRootMenu**()
Go to root menu.

Returnvalue

bool - false if the root menu is not set, true if success

bool **isGoBackPossible**()
Used to know if the back function has anything to go back to or if the current menu page is the first that was shown.

Returnvalue

bool - true if it is possible to go back and false if not

void **setHistorySize**(int newSize)
Set the size of the history. Default value is 10.
The current history data will be lost.

Parameters

newSize - [in] the new size.

bool **setParentMenu**(MenuItemBase* theMenu,
MenuItemBase* newParent)
Set the parent menu for another menu.

Parameters

theMenu - [in] the menu that want to set a new parent.
newParent - [in] the parent menu.

Returnvalue

bool - false if any of the two menus don't exist, true if success

bool **setRootMenu**(
MenuItemBase* newRootMenu)
Set the Root menu. This will be used to access the menu tree and must be set.

Parameters

newRootMenu - [in] the menu that will be the root menu.

Returnvalue

bool - false if the menu don't exist, true if success

bool **setStartMenu**(MenuItemBase* startMenu)
Set the menu to show at startup. This must be used because it also set the activeMenu pointer which has to be set before menu usage. This method also enters the startup menu.

Parameters

startMenu - [in] A pointer to the MenuItemBase object that are going to be startup menu.

Returnvalue

bool - true if success, false if not

bool **shortcutToMenuByName**(String menuName, bool useGroupState=false)

Shortcut to a menu by name. This will run the enter method in the menu and make it active. Children menus should NOT be accessed this way. To navigate to a child menu use the goToMenuByName() method.

Parameters

menuName - [in] the name of the menu to enter
useGroupState - [in] default value is false. if false the menu that is given in parameter 1 will be entered without checking the group state, if true the group state will be checked and the active menu in the visited group will be entered.

Returnvalue

bool - false if the menu don't exist, true if success

verifyReturn(int) **verifyMenuTree**()

Verifies the menu tree to see that all menus are accessible and that all menus have a parent set.

Returnvalue

verifyReturn(int) –

SUCCESS if the menu tree is ok,
MENU_SETUP_ERROR if any of the methods used to setup the menu structure failed. Check the return value for each method to see what went wrong,
NO_PARENT_SET if not all menus except the root menu have a parent or the menuTreeRoot is not set,
CHILD_TO_MANY_MENUS if one or more menus are child to more than one menu,
NO_MENU_NAME_OR_MENU_REFERENCE if one or more menus don't have a name or a MenuItemBase reference,
BAD_GROUP if not all menus under a group root menu is member of the group or the group only have one member.

3.2 MenuItemBase Class Reference

The base class that all menus have to extend and implement. All virtual functions have to be implemented.

3.2.1 PUBLIC MEMBER FUNCTIONS

	MenuItemBase()
	~**MenuItemBase**()
virtual void	**buttonPressed**(int keyCode) = 0
virtual void	**enterMenu**() = 0
int	**getId**()
String	**getName**()
virtual void	**leaveMenu**() = 0
void	**setId**(int newId)
void	**setName**(String newName)
virtual void	**update**(MenuEvent* event) = 0

3.2.2 MEMBER FUNCTION DOCUMENTATION

MenuItemBase()

Standard constructor. Empty in the base class.

~**MenuItemBase**()

Standard destructor. Empty in the base class.

virtual void **buttonPressed**(int keyCode) = 0
Virtual method that must be implemented by the classes that inheritate this class. The method is executed when the MenuController receives a button press from the main application.

Parameters

keyCode - [in] the code mapping the key to the ”GUI”

virtual void **enterMenu**() = 0
Virtual method that must be implemented by the class that inheritate this class. The method is executed when the menu is entered.

int **getId**()
Get the id of the menu.

Returnvalue

int - The id of the menu.

String **getName**()
Get the name of the menu.

Returnvalue

String - The name/path of the menu.

virtual void **leaveMenu**() = 0
Virtual method that must be implemented by the class that inheritate this class. The method is executed when the menu is left.

void **setId**(int newId)
Set the id of the menu.

Parameters

newId - [in] The id of the menu.

void **setName**(String newName)
Set the name/path of the menu.

Parameters

newName - [in] The name of the menu.

virtual void **update**(MenuEvent* event) = 0
This will be run on the active menu when the MenuController receive an event from any subject it listens to. If the menu don't need to do anything on any events declare an empty method.

Parameters

event - [in] A pointer to a object from a subclass to MenuEvent.

XML MENU BUILDER – USER GUIDE

Revision history

Date	Issue	Description	Author
2007-12-17	PA1	User guide for VISbas XML Menu Builder.	DS

Content

1 INTRODUCTION

This document describes how the XML menu builder can be used to generate the base code files for a menu system and how to write the XML file that describes the menu system. To understand this document the reader should be familiar with how the menu manager that will use the menu files work.

2 USER GUIDE

2.1 Overview

The three classes of the menu builder are MenuBuilder.cpp, XMLParser.cpp and DOMErrorReporter.cpp. The menu builder is run as a console application. The main method in MenuBuilder.cpp creates a object of the XMLParser class and this object is used to parse the XML file and generate the code files. If validation errors are found the DOMErrorReporter are used to present this to the user.

2.2 Usage

To use the menu builder a XML file is needed. The XML file has to validate against the provided XSD schema named XMLMenuBuilderSchema.xsd. In the next section we will see how to write a correct XML file and for now we just assume we have one. As mentioned above the menu builder is a console application. The syntax to use it is,

```
./MenuBuilder.exe Filename
```

where Filename is the name of the XML file.

The generated files will be placed in a created directory named “output”. If this directory exist and is not empty the application will inform about this and abort.

If the validation against the XSD schema fails during the parsing the application will inform about the errors and abort.

For each menu specified in the XML file a .h and a .cpp file will be created and these files are meant to be used as a base for each menu. An additional class will be created, called MenuSystem.cpp (and .h), and this class is supposed to be used to create and setup the menu structure in the application that uses the menus.

To do this the application can use the following code:

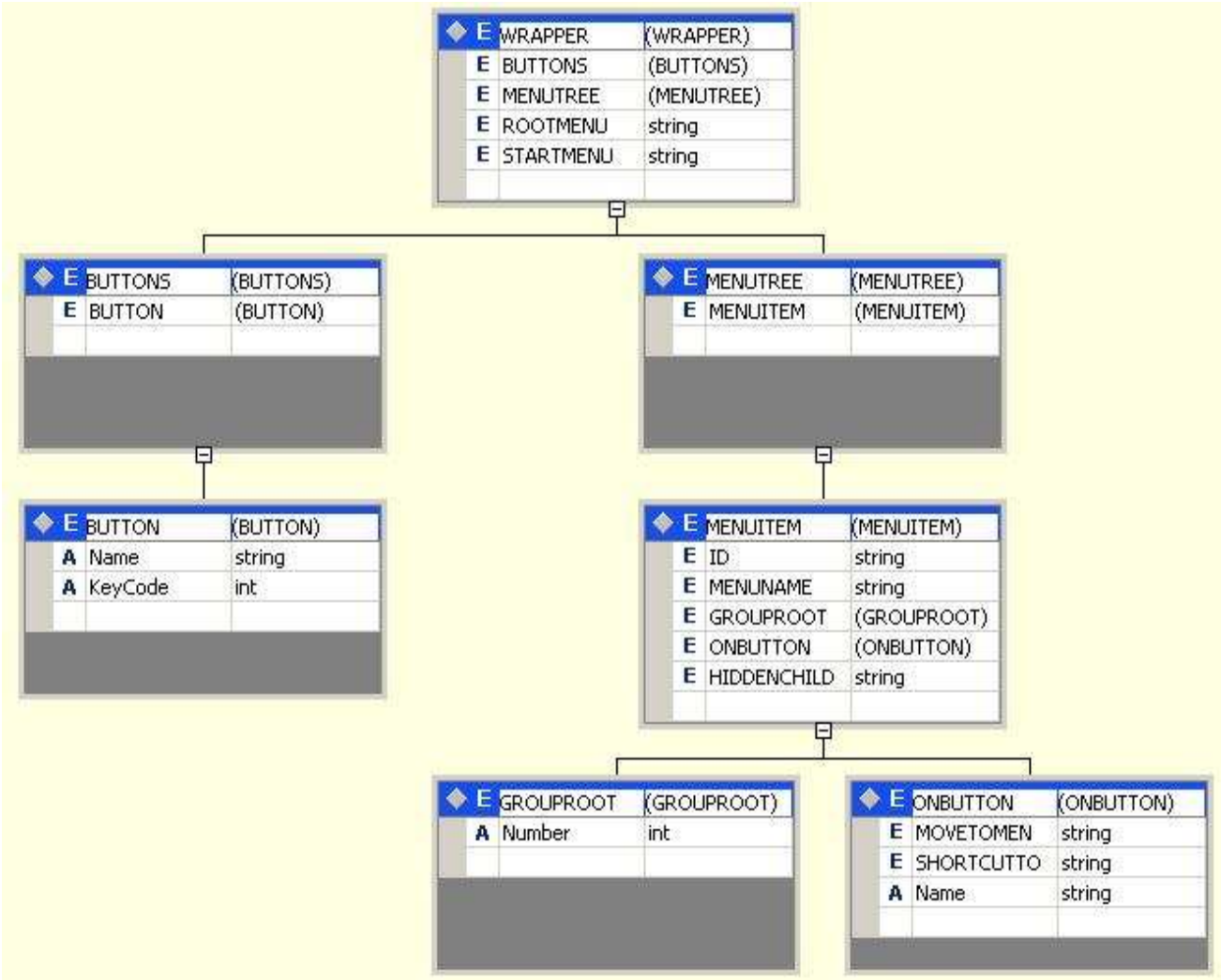
```
MenuSystem* MenuSystemObj = new MenuSystem();
```

The menu structure is setup in the menu controller that has to be included in the application. After this the menu controller are used exactly as if the menu structure was setup manually and the “MenuSystemObj” object never have to be used to access anything.

2.3 The XML file

As mentioned above, the XML file has to validate against the provided XSD schema. This section will show how to write a valid XML file.

The structure of the XML file has to be like the following picture shows it:



2.3.1 WRAPPER

Every valid XML document has one and only one element on top level. This element has to be called “WRAPPER” for the XML file used by the menu builder. As we can see in the picture the

"WRAPPER" element can include four different elements and all four have to be included.

2.3.2 BUTTONS

The "BUTTONS" element includes mappings from button names to key codes. The minimum number of buttons that have to be included are one. An example of a correct "BUTTONS" element is:

```
<BUTTONS>
        <BUTTON Name="Left1" KeyCode="11" />
        <BUTTON Name="Left2" KeyCode="12" />
        <BUTTON Name="Left3" KeyCode="13" />
        <BUTTON Name="Left4" KeyCode="14" />
        <BUTTON Name="Left5" KeyCode="15" />
        <BUTTON Name="Left6" KeyCode="16" />
        <BUTTON Name="Right2" KeyCode="17" />
        <BUTTON Name="Right3" KeyCode="18" />
        <BUTTON Name="Right4" KeyCode="19" />
</BUTTONS>
```

These mappings are only used in the XML file and the code files for each menu will have the key codes in the code instead of the button names. As we will see, the button names will be used in the rest of the XML file and this is to make it easier to write the XML file. Another reason to have this is to make it possible to have the same mappings in the XML file as in the application that uses the menus. For example, the button called "Left1" in the application will call the "buttonPressed" method in the menu controller and give the key code "11" as argument.

2.3.3 MENUTREE

The next element in the XML file is the "MENUTREE" element. This is where all the menus and the menu structure are specified. A simple example of this is:

```
<MENUTREE>
    <MENUITEM>
        <ID>0</ID>
        <MENUNAME>Home</MENUNAME>
        <GROUPROOT Number="1" />
        <ONBUTTON Name="Left1">
            <MOVETOMENU>Alarm</MOVETOMENU>
        </ONBUTTON>
        <ONBUTTON Name="Left2">
            <MOVETOMENU>Status</MOVETOMENU>
        </ONBUTTON>
```

```
            <ONBUTTON Name="Left6">
                  <SHORTCUTTOMENU>Home</SHORTCUTTOMENU>
            </ONBUTTON>
            <HIDDENCHILDMENU>Help</HIDDENCHILDMENU>
      </MENUITEM>
      <MENUITEM>
            <ID>1</ID>
            <MENUNAME>Alarm</MENUNAME>
            <ONBUTTON Name="Left6">
                  <SHORTCUTTOMENU>Home</SHORTCUTTOMENU>
            </ONBUTTON>
      </MENUITEM>
</MENUTREE>
```

As we can see above, the “MENUTREE” include two “MENUITEM” elements and this means two menu pages. Each “MENUITEM” have an “ID” and a “MENUNAME”. The first menu also has an element called “GROUPROOT” and this says that the menu is the top menu of a group in the menu tree structure. All menus under this menu in the tree will be members of the group too.
The “MENUITEM” elements also have an element called “ONBUTTON”. This is used to specify that if a button is pressed in that menu page, the menu manager will go to a new menu page. If the “MOVETOMENU” element is used to specify this, the menu that is given will also be a child to the “MENUITEM” menu page in the structure. If the “SHORTCUTTOMENU” element is used, the menu will not be a child to the “MENUITEM” menu page.
The “MENUITEM” element can include one more element and this is the one called “HIDDENCHILDMENU”. This is used to specify that another menu is a child to this menu just like the “MOVETOMENU” element does. However, if it is a hidden child, the menu doesn’t have a navigation to the menu with a key press. This can be useful to add a menu that is only accessible from a shortcut in another menu or from an event in the application.

2.3.4 ROOTMENU

This is used to specify the root menu of the tree structure and an example is:

```
<ROOTMENU>Home</ROOTMENU>
```

2.3.5 STARTMENU

This is used to specify the menu that will be shown at the startup of the application and an example is:

```
<STARTMENU>Home</STARTMENU>
```